Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Бакалавр
Освітній рівень

Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДППІЗ.190156.19.08.ПЗ

Виконав: студент III курсу, група ІПЗс-19-1 _____ О.В. Новацький
Підпис Ініціали, прізвище

Керівник _____ Л.П.Бедратюк
Підпис Ініціали, прізвище

Нормоконтролер _____ Н.І. Праворська
Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

1 червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти бакалавр
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма
«Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
05.02.2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ

Новацькому Олегові Валерійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram
Керівник проекту Бедратюк Леонід Петрович
доктор фізико-математичних наук, професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджено наказом ректора університету від 01.03.2022 р. № 18
2. Строк подання студентом проекту на кафедру:
3. Вихідні дані до проекту Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація системи, тестування програмної системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 18 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н.І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання «_05_» __лютого_ 2022 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту	Строк виконання етапів проекту	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02.2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	18.05.2022 (згідно графіка)	
8 Перевірка ДП на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки)	26.05 – 30.05.2022	
9 Підготовка до захисту та захист ДП	з 01.06.2022	

Студент _____ О.В. Новацький
Підпис Ініціали, прізвище

Керівник проекту _____ Л.П. Бедратюк
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: «Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram».

Автор проекту: Новацький Олег Валерійович.

Керівник проекту: Бедратюк Леонід Петрович.

Пояснювальна записка: 215 арк., 52 рис., 4 табл., 4 дод., 13 джерел.

Графічна частина: 18 презентаційних слайдів.

PHP, SYMFONY, POSTGRESQL, TELEGRAM, MESSENGER, REST API, VUE.JS, VUETIFY, NGINX, DOCKER, DOCKER-COMPOSE, RABBITMQ.

Мета проекту – розробка програмної системи збору інформації про виконані роботи на підприємстві текстильних виробів. Програма забезпечить своїх користувачів швидким та зручним способом обміну інформації на підприємстві. Також спростить процес підрахунку виконаних робіт, дасть можливість щодня виконувати модерацію зібраних даних за потреби. Система використовує сучасні засоби та інструменти для обміну і обробки цієї інформації, навіть для людей які не тісно взаємодіють із IT сферою.

У дипломному проекті визначено потреби підприємств малого бізнесу; проведено аналіз методів та інструментів розробки ПЗ; визначено способи та підходи до реалізації дипломного проекту.

Програмна система розроблена за допомогою мови програмування PHP (фреймворк Symfony 5). Дані зберігаються в СКБД PostgreSQL. Для обробки запитів використано сервер Nginx. Інтерфейс користувача реалізовано за допомогою фронтенд фреймворку Vue.js.

Створена програмна система може використовуватися на всіх підприємствах де необхідно проводити збір даних, та не витрачати на це зайві ресурси та час.

30.05.2022
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДППЗ.190156.19.08.ПЗ	Пояснювальна записка	215		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	18		

Змн.	Арк.	№ докум.	Підпис	Дат	ДППЗ.190156.19.08.ВД		
Виконав		Новацький О.В		30.05	Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram **Відомість документів**	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П.		30.05			
Н. контр.		Праворська Н.І.		30.05		ХНУ, ІПЗс-19-1	
Зав. каф.		Бедратюк Л.П.		30.05			

ЗМІСТ

ВСТУП 5
1 Дослідження предметної області та постановка задачі 9
1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей 9
1.2 Аналіз наявного програмно-технічного забезпечення предметної області. 15
1.3 Визначення вимог до ПЗ та постановка задачі 21
2 Проектування програмного забезпечення 27
2.1 Вибір типу архітектури та шаблонів проектування 27
2.2 Декомпозиція системи 29
2.3 Опис структури даних та моделі бази даних 36
2.4 Проектування користувацької частини 41
2.5 Аналіз та вибір технологій для реалізації програмної системи 46
3 Програмна реалізація 49
3.1 Детальне проектування модулів 49
3.2 Програмна реалізація модулів 54
3.3 Керівництво користувача 60
3.4 Вимоги до технічних та програмних засобів 71
4 Тестування програмного забезпечення 73
4.1 Вибір та обґрунтування методів тестування 73
4.2 Доведення працездатності програмного продукту 77
4.3 Аналіз результатів тестування 82
ВИСНОВКИ 83
ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ 86
ДОДАТОК А - ТЕХНІЧНЕ ЗАВДАННЯ 88

ДОДАТОК Б - ДІАГРАМА ПОСЛІДОВНОСТІ ПРОЦЕСУ СТВОРЕННЯ ОБ’ЄКТА ТИПУ РОБОТИ..... 93
ДОДАТОТ В - КОД (ЛІСТИНГ) ПРОГРАМИ 94
ДОДАТОК Г - ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ 206

ВСТУП

Сучасна економіка складається із багатьох компонентів і різних видів бізнесу. Значну частину бізнесів складають середні та малі на яких працюють різні кількості працівників. Всі вони виконують частинку певної роботи що є важливою для кожного з нас.

Серед таких бізнесів є підприємства, які займаються виробництвом текстильних виробів різного типу. Всі ці речі, які виготовляються на таких фабриках чи цехах, використовуються кожним із нас без винятку. Тому підприємства цього напрямку завжди існували і будуть існувати.

З кожним роком все більше підприємств пробують автоматизувати певні робочі процеси, не є виключенням і текстильна галузь. В залежності від напрямку робіт таких фабрик, автоматизація може бути різною. Організації використовують різні програми та засоби для тайм-менеджменту, обліку доходів та інших різних задач, що пов’язані із обробкою даних.

Дуже часто на таких підприємствах працюють люди, які не мають значного досвіду в IT сфері, що може спричиняти певні труднощі в робочих процесах. Тож якщо на таких підприємствах виконується автоматизація певних процесів, це вимагаю часу щоб навчити працівників працювати і керувати нововведеннями.

Одним процесом який є невід’ємним на кожному підприємстві, незалежно від його типу робіт, є облік виконаних задач. В багатьох цехах цей процес ще здійснюється за допомогою щоденників із нотатками про виконанні роботи. Таких підхід до ведення обліку спричиняє певні труднощі в процесі роботи підприємства, які вимагають певних затрат часу на їх вирішення. А так як в кожному бізнесі “час - це гроші”, то такий підхід до ведення обліку спричиняє лишні витрати підприємства.

Великі та середні бізнеси мають більші матеріальні ресурси, що дозволяють їм створювати власні великі програмні системи керування підприємствами або використовувати вже готові рішення за певну плату, які

полегшують процеси обліку і обробки даних підприємства, не залежно від їхньої форми і складності. В цей же самий час малі та частина середніх бізнесів потребують більш простих і дешевших рішень, які можуть вирішувати прості але рутині та більш специфічні речі, які не потребують встановлення великих та складних систем. Саме такою потребує є збір інформації про виконані роботи працівників протягом дня на постійній основі.

На сьогоднішній день існує багато рішень, які використовуються підприємствами різних типів і розмірів з метою обробки даних. Такими програмними засобами є CRM системи. Такі системи, зазвичай, містять в собі дуже багато функціоналу, яким потрібно вміти користуватися. Тож вся ця велика і складна система не потрібна для вирішення проблем які має вирішити цей дипломний проект.

Оскільки сьогодні смартфони і месенджери є частиною майже кожної людини то потрібно розробити рішення яке могло б використовувати ці речі, що в свою чергу знизило б витрати на закупівлі спеціального обладнання для персоналу і на покупку чи розробку складних систем.

Отже темою дипломного проекту є створення системи яка спростить процеси збору та модерації результатів виконаних робіт на підприємстві текстильних виробів за допомогою бота в Telegram.

Метою цього проекту є надання можливості для працівників підприємства використовувати підручні засоби для ведення обліку робіт та для керівників підприємства полегшення в процесі обробки та модерації цих облікових даних.

Щоб досягти поставленої мети було виконано наступні задачі:

– аналіз предметної області на прикладі підприємства, що займається створенням ковдр, покривал, подушок і багато чого іншого;

– визначення особливостей обробки даних на підприємстві;

– виявлення залежностей між складовими підприємства;

– аналіз існуючих рішень, які використовуються на підприємствах різних типів і порівняння цих рішень між собою;

– визначення доцільності застосування існуючих рішень на підприємствах предметної області проекту;

– визначення способів розробки і створення архітектури системи, яка задовольнить всі вимого до проекту;

– проектування системи ти визначення із технологіями і засобами для реалізації ПЗ;

– розробка та тестування програмної системи;

– пробація розробленої системи.

Розроблена система може використовуватися і на інших типах підприємств, де потрібно здійснювати облік виконаних робіт за допомогою простих та зручних рішень.

1 Дослідження предметної області та постановка задачі

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Предметна область цього дипломного проекту є підприємство по виготовленню текстильних виробів. Це підприємство малого бізнесу є схожим до тисяч тих, які виконують різні види робіт в цехах та фабриках. Роботи які виконуються в таких місцях, виконуються на спеціальному устаткуванні спеціально навченими людьми. Процеси виробництва та їх взаємодія можуть відрізнятися одне від одного, в залежності від галузі в якій працює підприємство.

Провівши змістовний аналіз предметної області було сформовано схеми що відображають структуру і процеси підприємства текстильних виробів. Бізнес процеси фірми і їх послідовність зображено на рисунку 1.1.1. Ось детальний опис всіх компонентів схеми та їхніх залежностей:

– процес виробництва розпочинається із прийому замовлення від клієнта та визначення необхідних ресурсів які необхідні для цього замовлення;

– після формування списку необхідних ресурсів, здійснюється покупка необхідної сировини і її доставка до підприємства;

– для розвантаження сировини залучаються працівники цього цеху;

– рулони тканини, які були доставлені на виробництво, потрапляють на стіл де виконується покрій за розмірами та заготовленими формами;

– після підготовки тканини до пошиття, швея бере заготовки і з’єднує потрібні частини між собою;

– всі заготовлені частини продукту відносяться до місця де їх наповнюють необхідною сировиною;

– наповненні продукти відправляються до місця пошиву і лямування;

– вже готовий продукт відноситься в склад де проходить процес пакування і підготовки на відправку;

– всі підготовлені пакунки із продуктами завантажуються в автомобіль що здійснює доставку до замовника;

– по завершенню завантаження, машина відправляється до замовника.

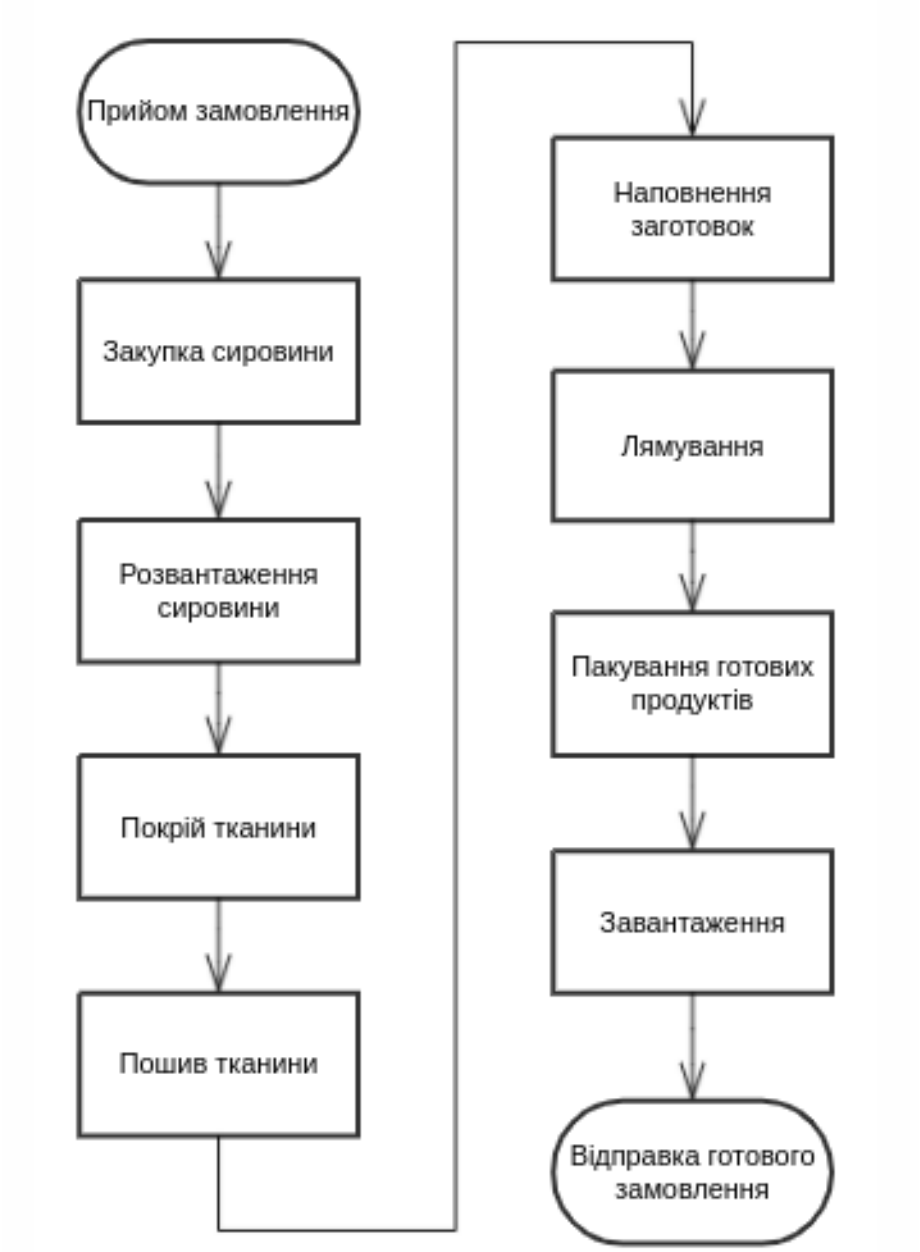


Рисунок 1.1.1 - Схема процесів залучених до виконання замовлення

Всі працівники підприємства залежні від виконання своєї роботи іншими працівниками. Хронологічну послідовність залежностей можна прослідкувати на схемі яку зображено вище. Залежність між ролями працівників показано на рисунку 1.1.2.

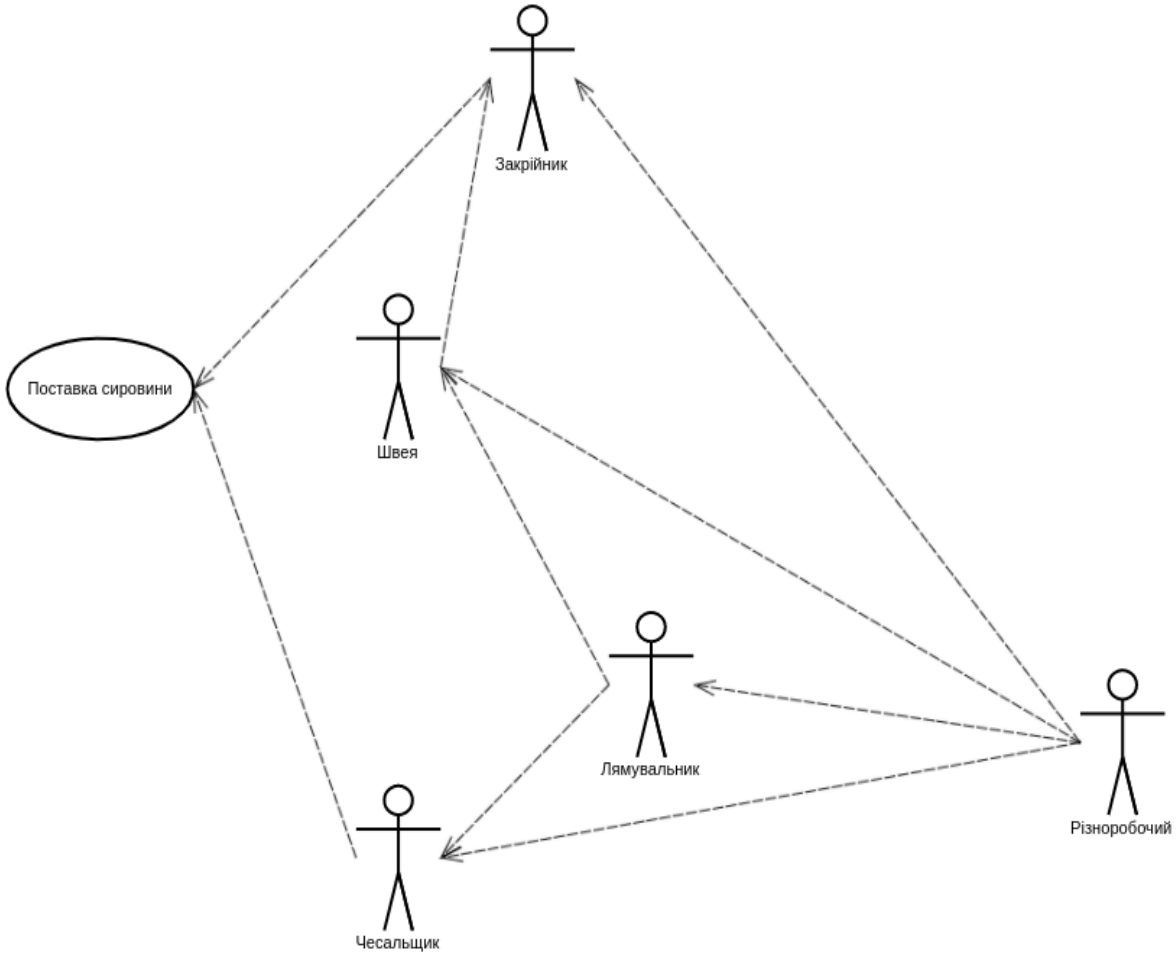


Рисунок 1.1.2 - Залежність між ролями працівників

Не дивлячись на ці залежності можна виділити структуру підприємства, яка складається із трьох ролей: керівник, технолог і працівник.

На рисунку 1.1.3 зображено обов’язки та задачі які покладені на керівника, а саме:

– прийом замовлень і комунікація із клієнтами;

– закупівля сировини та засобів для виготовлення готової продукції;

– передача деталей замовлення та вказівок по його виконанню технологу;

– інші організаційні задачі, що пов'язані із функціонуванням підприємства.

На рисунку 1.1.4 зображено обов’язки та задачі які покладені на технолога, а саме:

– комунікація із керівником, прийом деталей замовлення та планування виконання цього замовлення;

– слідкування за процесом роботи та виконанням поставлених задач працівникам;

– встановлення задач працівникам та координація взаємодії працівників між собою.

Рисунок 1.1.3 - Обов’язки та задачі керівника

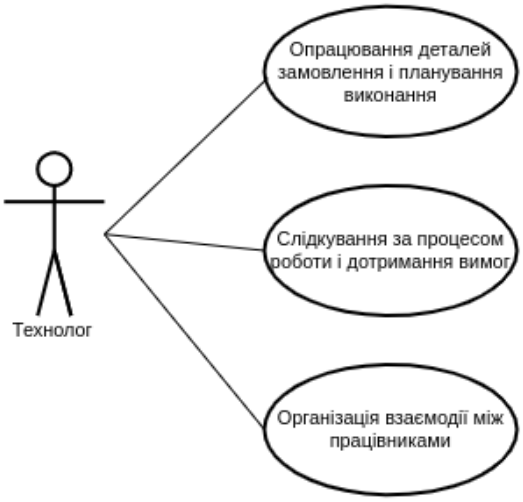


Рисунок 1.1.4 - Обов’язки та задачі технолога

На рисунку 1.1.5 зображено обов’язки та задачі які покладені на працівника, а саме:

– виконання завдань, які поставив технолог;

– ведення обліку виконаних робіт.

Рисунок 1.1.5 - Обов’язки та задачі працівника

Задачі технолога відрізняються від задач простих працівників, але якщо розглядати їх із загальної точки зору, то вони також є завданнями які технолог виконує кожного дня і їх потрібно звітувати керівництву. Тож працівники і технолог по завершенню робочого дня записують результати виконаних робіт в щоденники, які використовуються як звітність для керівництва.

Роботодавцю потрібно щомісяця здійснювати обрахунки виконаних робіт для кожного працівника щоб винагородити всіх відповідно. Для цього виконується збір щоденників із звітностями та перерахунок всіх результатів.

В даний час процес обліку виконаних робіт виконується в паперовому форматі, що є досить незручно і вимагає постійного слідкування за своїм нотатником. На рисунку 1.1.6 зображено схему взаємодії працівників із керівництвом, яка здійснюється через щоденники (нотатники).

Рисунок 1.1.6 - Схема взаємодії працівників із керівником

Такі підходи до організації праці вже відходять в минуле, але все ще присутні на таких малих підприємствах, які не мають можливості або бояться впроваджувати нові методи автоматизації праці. Кожна автоматизація вимагає від всіх учасників взаємодії певного часу навчання. Це той час яким працівники підприємства не хочуть витрачати, але як показує досвід, майже всі процеси автоматизації приносять позитивний результат у підприємство що його використовує.

Проблемою підприємства є застарілий спосіб обміну і обробки інформації між працівниками і керівниками. Для кожної із ролей можна визначити такі інформаційні потреби. Працівникам необхідно зручний та простий спосіб введення обліку своїх робіт. Керівнику необхідно спростити процес збору, опрацювання і підрахунку результатів роботи працівників. Впровадження ПЗ що вирішать цю проблему спростить процеси взаємодії, усуне зайві процеси, що в результаті буде мати позитивний вплив на підприємство. На рисунку 1.1.7 зображено очікуваний, спрощений варіант взаємодії працівника і керівника.

Рисунок 1.1.7 - Схема взаємодії працівника і керівника

Тож після аналізу предметної області, визначення проблеми та інформаційних потреб працівників підприємства чітко видно напрямок в якому потрібно рухатися щоб автоматизувати і поліпшити бізнес процеси підприємства.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На сьогоднішній день на ринку є багато засобів, які автоматизовують процеси обробки та введення обліку виконаних робіт та загалом виконують різні операції над даними. Такими засобами є CRM системи.

CRM-система - це програма, яка допомагає організувати процес комунікації з клієнтами та персоналом. Її головна мета - навести порядок у процесах продажу, стандартизувати їх, зробити роботу простішою, швидшою та зручнішою, налаштувати процеси взаємодії із працівниками підприємства. Залежно від завдань та функціоналу який потрібно виконувати, CRM може бути встановлена на комп'ютер або використовуватися як хмарний сервіс. Більшість

сучасних CRM-систем також мають мобільний додаток для роботи с базою клієнтів звідусіль.

Коли бізнес заснований на роботі з клієнтами, потрібно контролювати кожну деталь. Вести базу клієнтів та облік заявок, історію комунікації, стежити за ходом продажу, фіксувати розклад зустрічей і дзвінків, призначати задачі. У бізнесі часто використовуються масові розсилки. Окрім цього, компаніям необхідно контролювати роботу співробітників, забезпечувати комунікацію між відділами та проводити аналітику. CRM-система виконує всі вище перераховані функції, об'єднуючи кілька сервісів. Співробітники позбавляються від необхідності переключатися між вкладками та вікнами, вся робота ведеться в одній програмі. Робота з CRM скорочує час та допомагає автоматизувати рутинні задачі.

У CRM можна призначати задачі, розподіляти їх між виконавцями, відстежувати хід виконання, встановлювати пріоритетність завдань. Уніфікований спосіб постановки завдань дозволяє самим співробітникам нічого не забути і не випустити з уваги.

Здатність CRM-системи підлаштовуватися під завдання бізнесу робить її універсальною для різних сфер і компаній різного складу. Не варто вірити упередженню, що CRM-система - це програма тільки для великого бізнесу. У невеликих компаніях, де співробітники часто суміщають кілька посад, CRM також виявиться незамінним помічником.

Для того, щоб CRM підійшла малому бізнесу, вона повинна бути гнучкою в налаштуваннях, недорогою та максимально простою у використанні. Тому виділяються певні CRM-системи для індивідуальних підприємців. Фахівці, які працюють на себе, змушені поєднувати відразу кілька посад. Це і маркетинг, і продажі, і адміністрування. І навіть якщо клієнтів небагато, немає необхідності тримати всю інформацію в голові або витрачати час на заповнення щоденника.

Система представить хід роботи по кожному клієнту в зручному вигляді, нагадає про зустріч, спростить створення однотипних документів і зведе до мінімуму всю ручну роботу. Особливо актуальна CRM для юристів, ріелторів та аналогічних спеціальностей.

CRM-системи поділяють на такі види:

Операційні CRM - системи даного виду націлені насамперед на спрощення операційної сторони роботи. В її функціонал входить автоматизація процесів, організація клієнтської бази та фіксація даних на всіх етапах продажів, постановка завдань і контроль роботи співробітників.

Аналітичні CRM - у системах цього виду основний наголос зроблений не тільки на зборі, але й на наступному аналізі зібраних даних. Такі CRM-системи використовуються, коли необхідно сегментувати базу, визначити рентабельність, проаналізувати воронку продажів, простежити поведінку клієнтів на різних етапах. Ці CRM допоможуть у складанні прогнозів і оцінки ефективності маркетингової стратегії.

Колабораційна CRM - мало поширений вид програм. Це рішення для тих продуктів, які розробляються за безпосередньої участі споживачів із використанням різних каналів зв'язку (через інтернет-портал, телефонію, особисті контакти та ін.).

Комбіновані CRM - найбільш універсальні системи які поєднують автоматизації процесів, аналітику та інше функції всіх перерахованих вище типів систем. Це забезпечує максимальну зручність та комфорт роботи для великих та складних корпорацій із великими списками працівників, клієнтів та інформації які потрібно опрацювати.

Зараз на українському ринку є багато CRM систем. Ось наведено певний перелік цих систем. Детальніший опис можна переглянути у статті [1].

KeyCRM (https://keycrm.app/) - система для інтернет-торгівлі. Допомагає автоматизувати інтернет-торгівлю: автоматично збирає замовлення з усіх усюд (Rozetka, Prom, Amazon, Etsy, Ebay, Instagram, Facebook тощо), надає гнучкий

доступ для співробітників, дозволяє оформити відправку в кілька кліків та провести замовлення до моменту отримання його покупцем. Головна сторінка системи показана на рисунку 1.2.1.

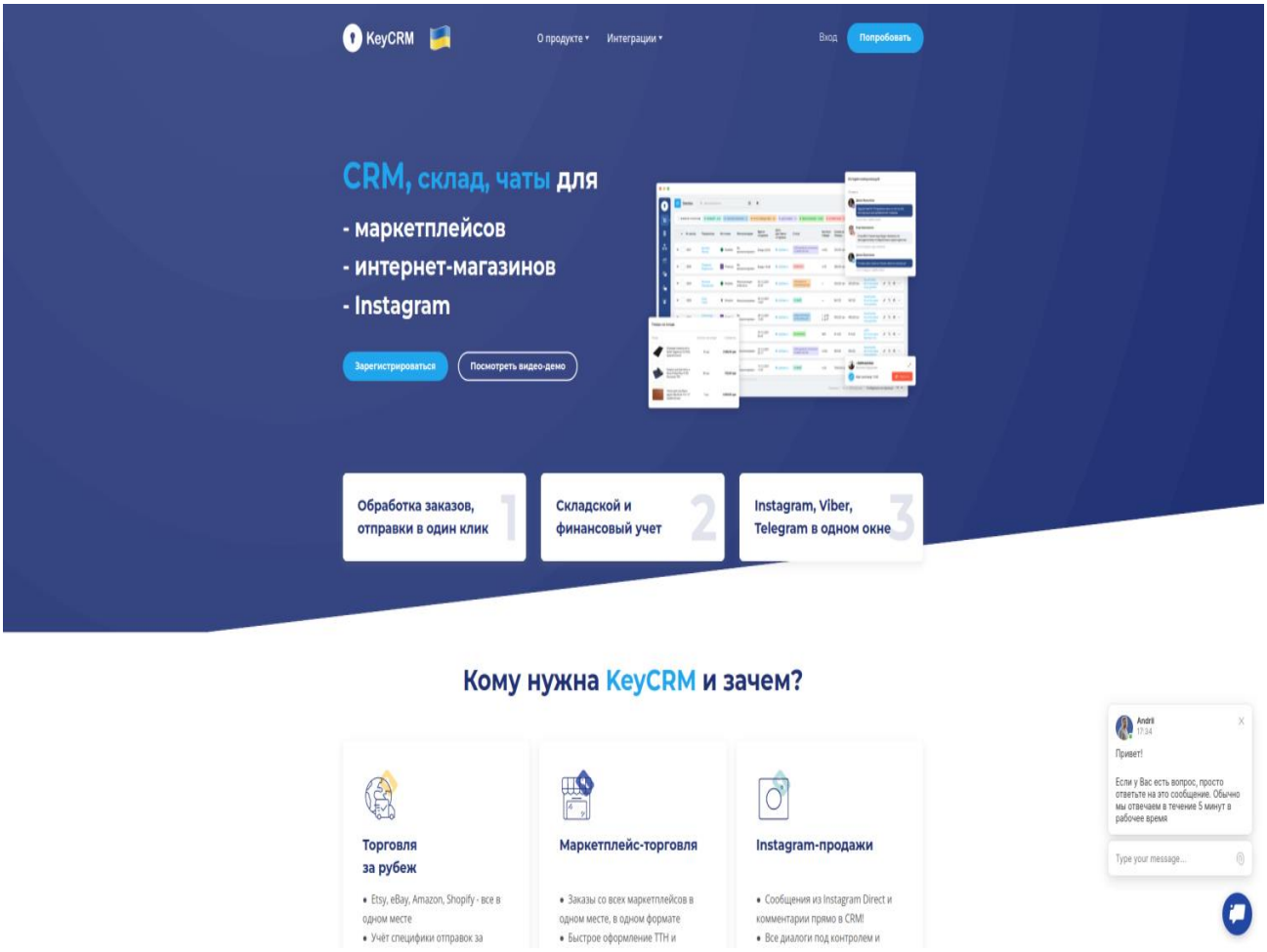


Рисунок 1.2.1 - Головна сторінка веб-сайту KeyCRM

Sales Creatio від Terrasoft (https://www.terrasoft.ua/sales) - готовий продукт для автоматизації та прискорення повного циклу продажу: від «ліда» до повторних замовлень. Крім цього, існують зміни для управління маркетингом, сервісом та бізнес-процесами. На онлайн-майданчику Marketplace представлені готові доповнення та галузеві рішення, конектори та шаблони для розширення можливостей платформи та автоматизації різноманітних бізнес-завдань. Головна сторінка системи показана на рисунку 1.2.2.

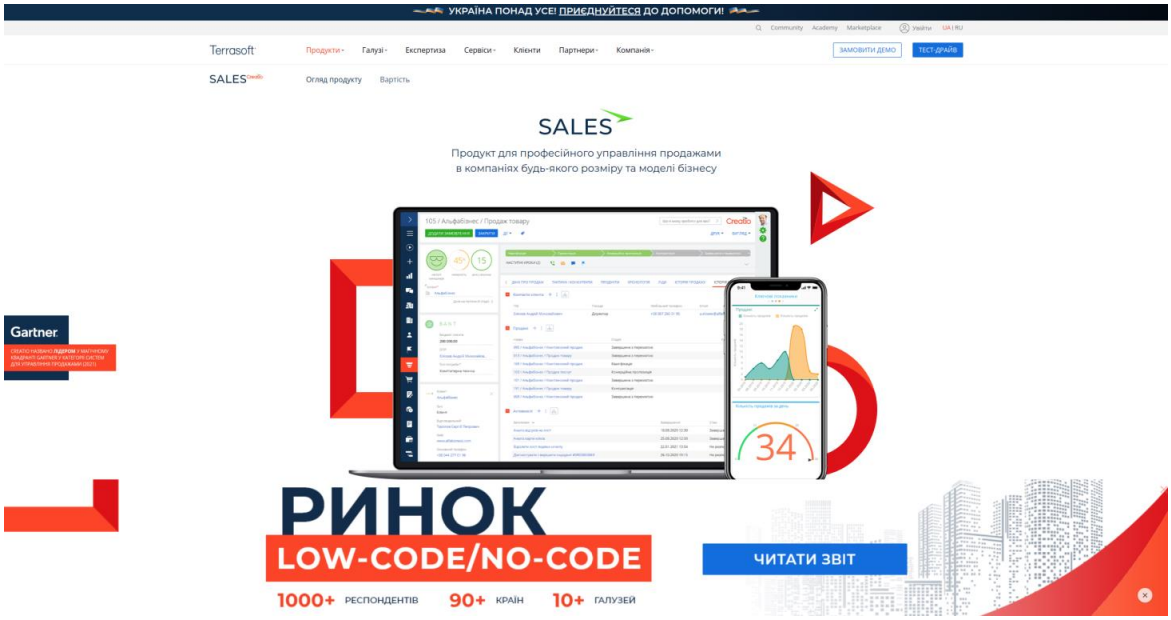


Рисунок 1.2.2 - Сторінка веб-сайту Sales Creatio

Бітрикс24 (https://www.bitrix24.ua/) - Соціальний інтранет за схемою SaaS. Містить мікроблоги, завдання, файлосховища (з контролем версій), календар, фотогалереї, месенджер, екстранет, CRM, бізнес-процеси, облік робочого часу, профайли, звіти. Є мобільна версія. Є безкоштовна версія. Головна сторінка системи показана на рисунку 1.2.3.

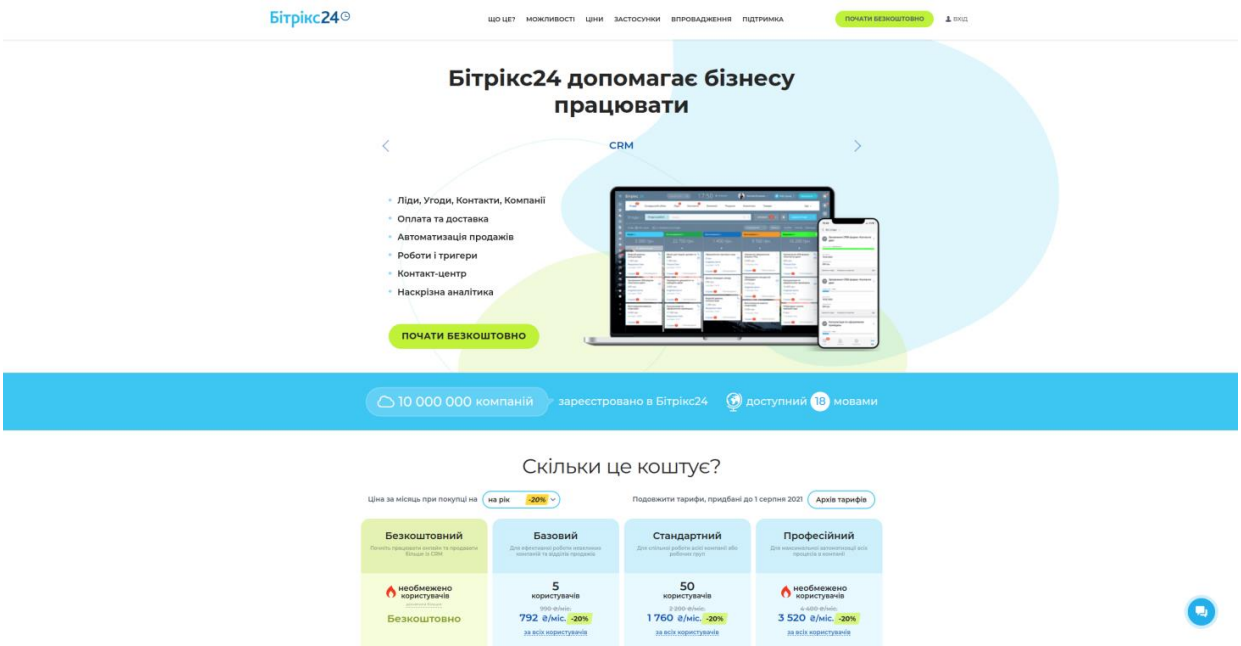


Рисунок 1.2.3 - Головна сторінка Бітрікс24

Dynamics 365 (https://dynamics.microsoft.com/en-us/) - CRM/ERP система зі звичним інтерфейсом від Microsoft. Повна інтеграція з платформою та програмами MS. Містить інструменти для управління продажами, маркетингом, сервісом та бізнес-процесами. Можлива робота із системою прямо з Outlook. Головна сторінка системи показана на рисунку 1.2.4.

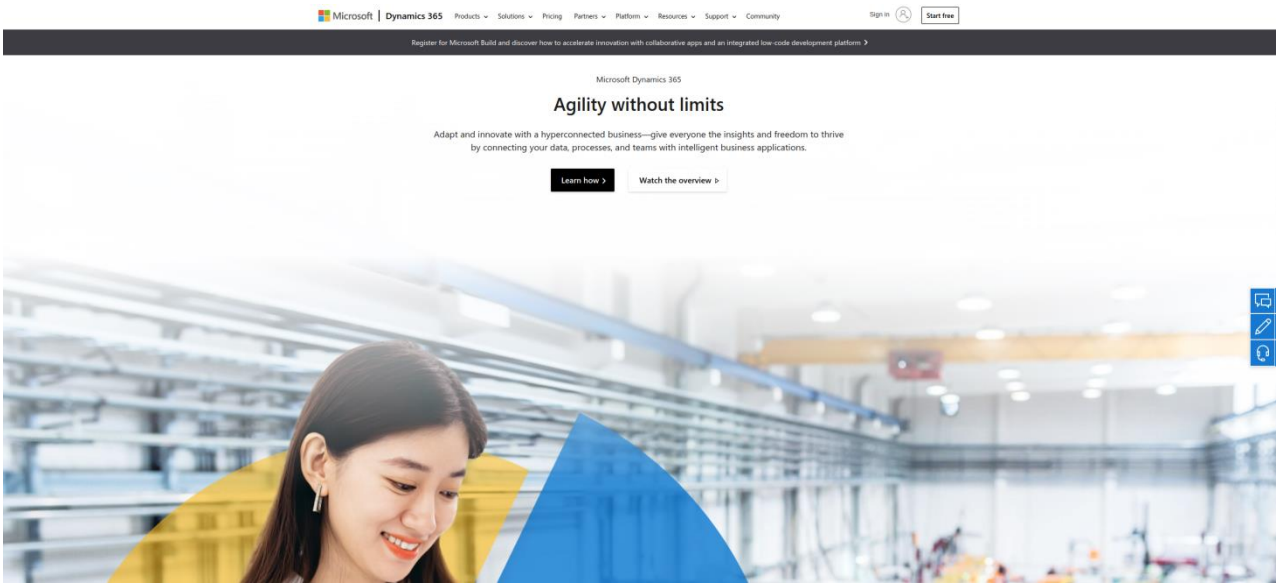


Рисунок 1.2.4 - Головна сторінка веб-сайту Dynamics 365

SalesDrive (https://salesdrive.com.ua/) - Система управління продажами. Можливість налаштування під свій бізнес. Готові рішення CRM для різних галузей. Інтеграція з сайтом, телефонією, SMS, email, Новою Поштою. Чудово підходить для інтернет-магазинів.

OneBox (https://1b.app/ua/) - Платформа, операційна система для бізнесу, вона складається із сотень додатків: CRM система, ERP система, побудова бізнес-процесів, телефонія, складський та товарний облік, електронний документообіг... Можна створювати свої програми або замовляти у сертифікованих розробників, які зроблять вам «софт» на OneBox OS в десятки разів швидше і дешевше, ніж будь-хто на ринку.

KeepinCRM (https://keepincrm.com/) - Онлайн CRM система для малого та середнього бізнесу. Зручний та інтуїтивно зрозумілий інструмент для

оптимізації роботи з клієнтами, лідами, завданнями, прайс-листом, угодами, складом, запровадженням фінансів та документів.

Частина згаданих вище систем є російського виробництва, тобто були розроблені російськими IT компаніями. Тому не рекомендується їх використання у час війни із Росією, щоб гроші які будуть сплачені за користування системи не йшли у вигляді податків до держави агресора.

У всіх цих систем є складові, якими користуються майже всі. Це канбан дошка, звітність по часу який потрачено на виконання задач (трекінг часу), календар із датами відпусток, запити на відпустку, перелік працівників, перелік клієнтів, та інше.

Всі описані вище CRM системи містять в собі надлишок функціоналу, який не потрібен на підприємствах таких розмірів та де рівень кваліфікації працівників по роботі із подібними сервісами дуже малий або відсутній взагалі.

Тож програмне забезпечення, яке буде розроблено в рамках дипломного проекту, буде позбавлено всіх надлишкових функцій які можуть заплутувати користувача системи, та міститиме кілька подібних, на які спрямований цей дипломний проект. Користувачі системи будуть забезпечені усіма необхідними функціями, якими досить легко користуватися. Ці переваги розроблюваного ПЗ відповідають потребам ринку програмних продуктів.

1.3 Визначення вимог до ПЗ та постановка задачі

Відповідно до потреб підприємства, які необхідно вирішити в рамках цього ДП потрібно розробити ПЗ, яке спростить та автоматизує процеси обробки даних, щоб економити час роботодавця, який і так завантажений іншими організаційними питаннями підприємства

Початок розробки будь якого програмного забезпечення розпочинається із проектування системи та розробки діаграм (UML діаграм), які наглядно відображаються вимоги до системи, її складові, структуру, процеси, функції і взаємозв’язки між компонентами. Як будувати діаграми описано в роботі [2].

Зазвичай проектування діаграм починається із створення діаграм варіантів використання. Для створення діаграми варіантів використання потрібно визначити акторів, які будуть взаємодіяти із системою. В програмній системі, яка буде розроблятися буде всього лише дві ролі: працівник і керівник.

Роль “Працівник” представляє собою працівників підприємства. Діаграма використання для цієї ролі зображена на рисунку 1.3.1.

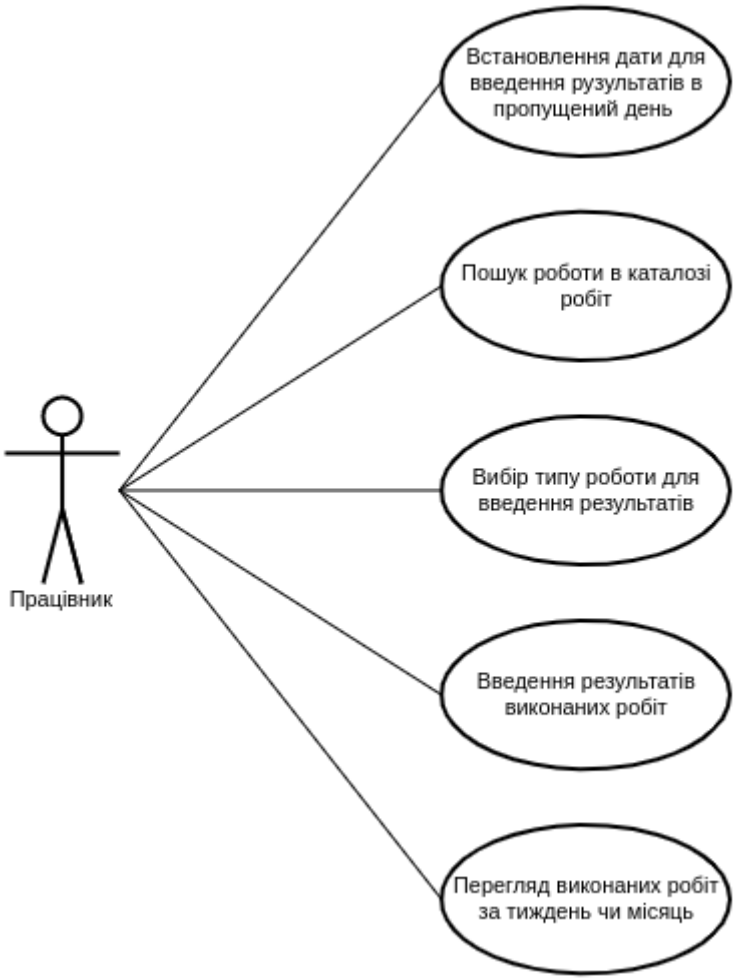


Рисунок 1.3.1 - UML діаграма використання для ролі “Працівник”

На цій діаграмі зображено наступні функції, які будуть доступні користувачам із роллю “Працівник”:

– пошук типу роботи в каталозі робіт;

– вибір типу роботи для введення результатів виконаних робіт;

– введення результатів виконаних робіт;

– перегляд виконаних робіт за тиждень/місяць чи конкретний день.

Роль “Керівник” представляє собою керівника підприємства чи особу яка займається організацією процесів на підприємстві. Діаграма використання для цієї ролі зображена на рисунку 1.3.2. На цій діаграмі зображено наступні функції, які доступні користувачеві із цією роллю:

– створення облікових записів працівників;
– створення та редагування категорій типів робіт;
– створення та редагування типів робіт;
– встановлення цін та величин виміру для типів робіт;
– створення кнопок бота;
– перегляд виконаних робіт працівником(-ами) за певні періоди;
– формування звіту із зарплатами за зарплатний період.

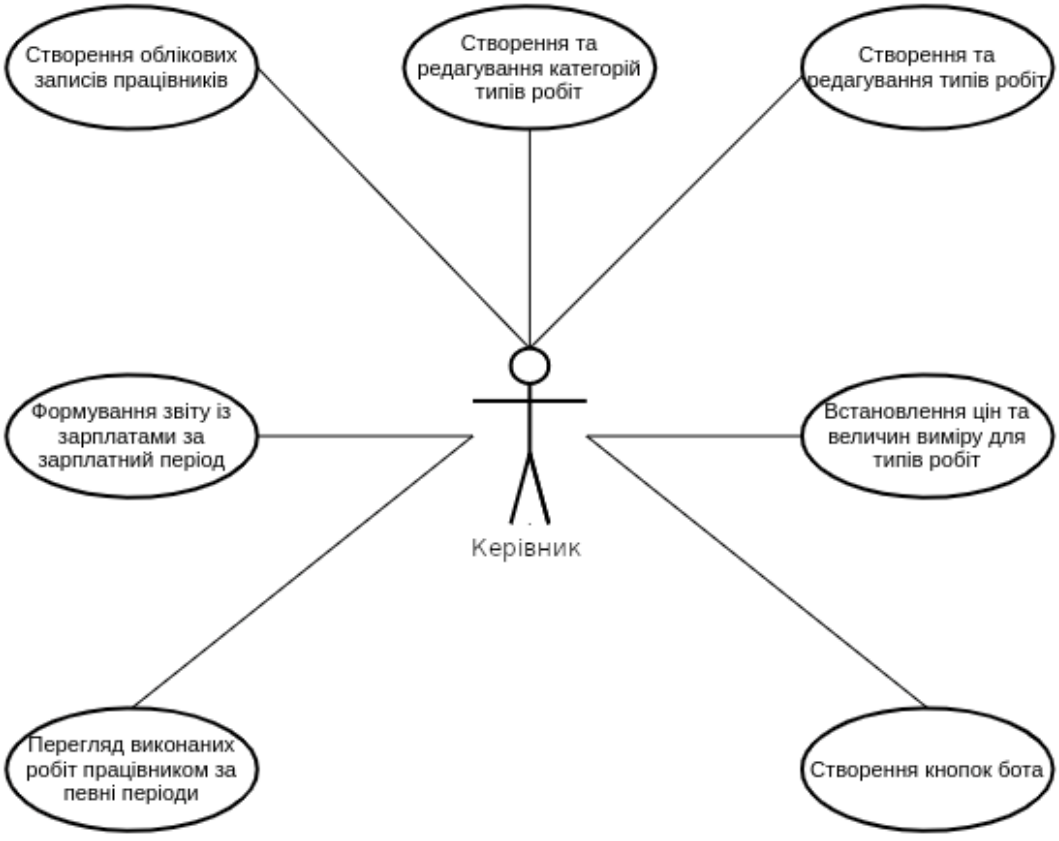


Рисунок 1.3.2 - UML діаграма використання для ролі “Керівник”

Також було створено діаграму діяльності яка зображає увесь процес введення результатів користувачем із роллю “Працівник”. Користувачеві потрібно буде перейти в режим введення результатів, де він матиме змогу знайти потрібний тип робіт які він виконував протягом дня. Коли користувач завершить ввід то він виходить із режиму введення і повертається на початок. Ця діаграма зображена на рисунку 1.3.3.

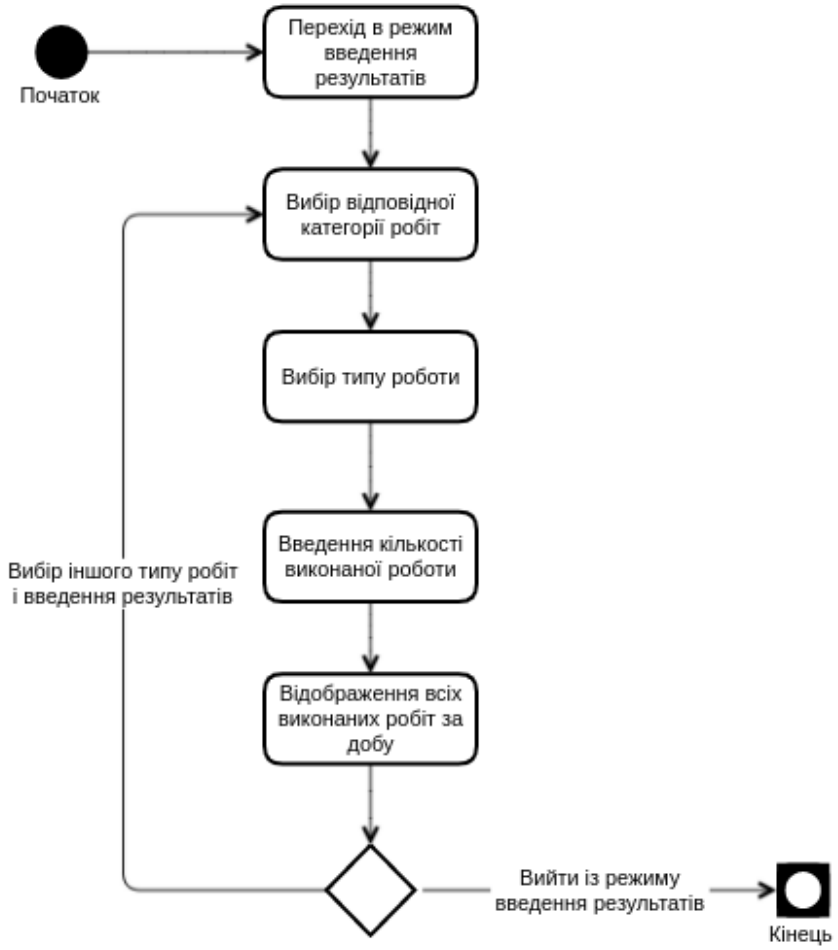


Рисунок 1.3.3 - UML діаграма діяльності процесу введення результатів

На діаграмі, що на рисунку 1.3.4, зображено діаграму діяльності що описує процес створення звіту в веб-інтерфейсі керівника. Користувачеві потрібно відкрити сторінку звітів у веб-застосунку, на формі вибрати всі необхідні фільтри для формування звіту. На формі мають бути присутні фільтри типу

звіту, фільтр по працівнику та по звітному періоді. Після формування звіту він завантажується на пристрій користувача.

Рисунок 1.3.4 - Діаграма діяльності процесу створенні звіту

Керівник підприємства має представити список робіт які можуть виконуватися на підприємстві. Тому було створено діаграму діяльності в якій зображено процес створення категорій і типів робіт які будуть використовуватися працівниками під час подачі звітності. Цю діаграму діяльності показано на рисунку 1.3.5. Користувачу системи потрібно перейти на сторінку створення категорій товарів та створити новий запис, який є обов'язковим під час створення типу роботи. На сторінці заповнюється форма всіма необхідними даними та виконується збереження. Після цього можна

перейти на сторінку створення типу роботи, де буде обиратися відповідна категорія, одиниці виміру та інша необхідна інформація для типу роботи.

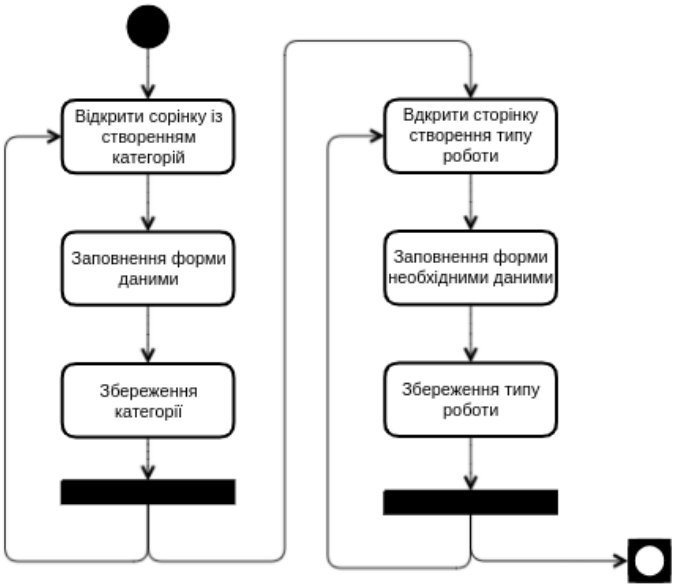


Рисунок 1.3.5 - Діаграма діяльності процесу створення списку робіт

При аналізі вимог до інтерфейсу було встановлено, що взаємодія працівників із системою буде здійснюватися за допомогою кнопок телеграм бота та поля ведення тексту, тож для користувачів із роллю “Працівник” не потрібно створювати особливого дизайну, тому як буде використовуватися особистий дизайн кожного користувача Telegram. Керівник підприємства буде користуватися веб-інтерфейсом, в якому будуть використовуватися стандартні елементи Vuetify. Vuetify це модуль для фреймворку Vue.js, який містить вже готові елементи, які використовуються для швидкої розробки SPA. Більш детально про фреймворк можна почитати на ресурсі [3]. Цей веб-інтерфейс має на меті забезпечити свого користувача потрібними функціями в першу чергу, тоді як дизайн не несе вагомого значення.

Під час проведення аналізу вимог до ПЗ було визначені функціональні та не функціональні вимоги і створено технічне завдання, яке подано у додатку А. Також створено різні діаграми, що наочно відображають вимоги до проекту.

2 Проектування програмного забезпечення

2.1 Вибір типу архітектури та шаблонів проектування

За темою дипломного проекту, потрібно розробити систему яка буде взаємодіяти із Telegram ботом, використовуючи API Telegram. Для такої взаємодії можна реалізувати сервер, що робитиме запити на API телеграму, щоб передавати інформацію і отримувати повідомлення від користувачів бота. Проте телеграм забезпечує своїх користувачів механізмом “web-hook”. Цей підхід дозволяю не виконувати постійні запити на отримання повідомлень, а всього лише вказати URL адресу на яку відправляти повідомлення від бота. Ця адреса і є “web-hook”. Повідомлення передаються у форматі JSON, який сьогодні дуже широко використовується і витісняє XML. Детальніше про цей метод взаємодії описано в статті [4]. Отож сервер повинен представити кінцеву точку на яку телеграм зможе відправляти повідомлення. Ця частина системи, буде необхідна лише для здійснення комунікації із працівниками підприємства - збору даних про виконані роботи. Такий архітектурний підхід комунікації між серверами називається “сервер - сервер”.

При роботі із веб-хуком телеграму є одна особливість. Кінцева точка, що приймає повідомлення від сервера телеграма, повинна за кілька секунд відповісти із успішним статусом. Деколи цього часу не вистачає щоб опрацювати повідомлення, так як можуть бути запущені “важкі” процеси. Щоб вирішити це питання, використовують брокери повідомлень, які накопичуються в себе всі вхідні повідомлення. Сервер системи поступово опрацьовує збережені повідомлення, які накопилися в брокері. Таке рішення дозволяє використати підхід із “web-hook” і зняти навантаження на систему. Навантаження зніметься за рахунок поступового опрацювання повідомлень замість миттєвого опрацювання всіх паралельно. Цей підхід використовується в багатьох проектах для реалізації різних потреб, але із одною метою - оптимізація складних і тривалих процесів.

Для відображення зібраних даних потрібно розробити веб-інтерфейс, що дозволить користувачам із роллю керівник взаємодіяти із системою, налаштовувати її, переглядати зібрану інформацію та робити запити на генерацію необхідних звітів. Сьогодні це можна зробити багатьма способами, використовуючи різні інструменти. Але це найпростіше зробити, реалізувавши SPA. Що таке SPA описано в цій роботі [5]. Для цього існує багато фронтенд фреймворків, які мають уже багато шаблонів, що можна використати знову і зекономити час на розробку продукту.

Ці дві складові системи представляють собою загально прийнятий підхід до розробки сучасних веб-додатків, де SPA звертається до сервера із метою отримання даних та запитом на виконання певних дій. На таких серверах реалізовуються REST API, що використовує. JSON для комунікації. Такий архітектурний підхід комунікації називається “клієнт - сервер”.

Загальна послідовність дій виглядає ось так. Керівник використовує веб-інтерфейс для комунікації із сервером. SPA виконує запити на отримання та збереження даних. Сервер використовує сервіс бази даних для збереження інформації. Працівники використовують телеграм додаток для спілкування із ботом, який являється телеграм сервером. Всі повідомлення перенаправляються на сервер системи і відразу потрапляють до брокера повідомлень. Обробники на сервері опрацьовують повідомлення із брокера, виконують обробку отриманих даних, зберігають їх в базу даних та відправляють запити не сервера телеграму. Телеграм у вигляді бота, відповідає працівнику.

Всі ці процеси являють собою потоки даних та відношення між компонентами системи. Сукупність всіх компонентів і потоків даних зображено на рисунку 2.1.1.

Реалізація продукту саме цим методом забезпечить взаємодію всіх акторів і компонентів системи між собою та використає в реалізації два архітектурних підходи.

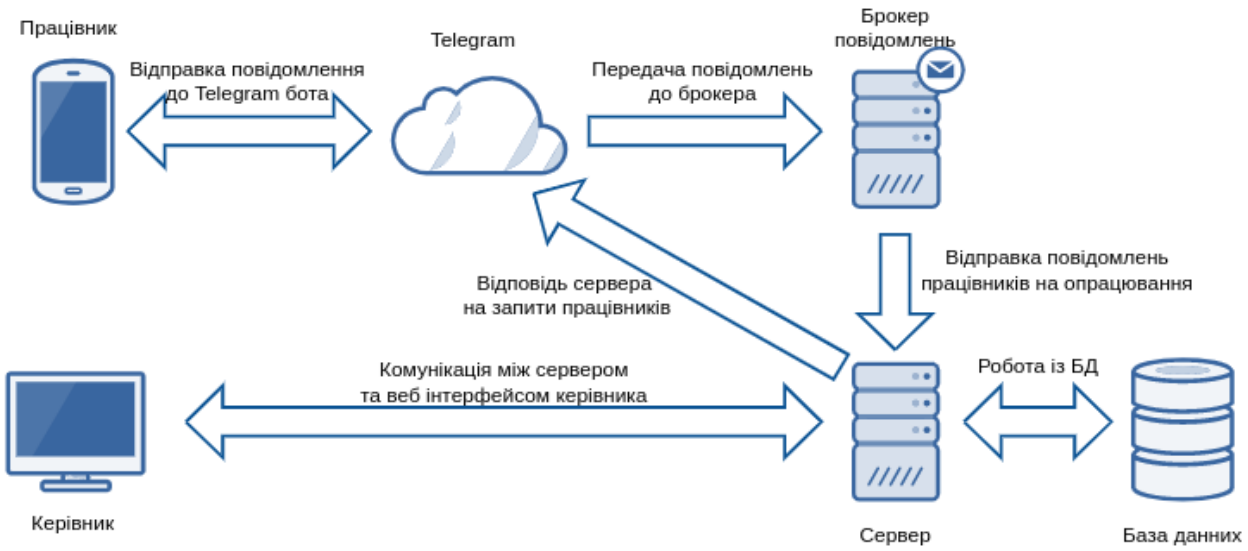


Рисунок 2.1.1 - Схема компонентів системи і їх взаємодія

2.2 Декомпозиція системи

Для більш детального проектування системи і визначення внутрішнього змісту потрібно виконати декомпозицію системи на менші, більш прості підсистеми. Розглядаючи систему загалом, виділяється чотири підсистеми:

– підсистема прийому повідомлень від телеграму і збереження їх в брокері повідомлень;

– підсистема опрацювання отриманих повідомлень і збереження результатів в БД;

– підсистема спілкування із ботом, використовуючи Telegram API;

– підсистема для опрацювання запитів із веб-інтерфейсу.

Найкращим шляхом відображення структури підсистеми є діаграма класів. На рисунку 2.2.1 відображено діаграму класів підсистеми прийому повідомлень. На діаграма присутній клас «TelegramWebHookController», який представляє собою точку входу для запиту із повідомленням користувача від телеграма. Він передає запит на подальше опрацювання і повертає відповідь із успішним статусом. Запит потрапляє на опрацювання в один із класів, що реалізовують інтерфейс «WebhookRequestBodyProcessorInterface». Для вибору реалізації

інтерфейсу використовується клас «WebhookRequestBodyProcessorFactory». Він визначає яку реалізацію потрібно повернути, враховуючи активний метод опрацювання і доступність брокера повідомлень. Якщо система в стадії розробки чи брокер недоступний, то використовується «LiveWebhookRequestBodyProcessor», який відправляє повідомлення відразу на обробку в клас «MessageProcessor», який вже є частиною другої підсистеми - опрацювання повідомлень. Якщо ж система в «продакшин» режимі і брокер доступний, то повідомлення потрапляє в «QueueWebhookRequestBodyProcessor», який відправляє повідомлення в чергу для відкладеної обробки. «MessageBus» використовує налаштування для доступу до брокера повідомлень.

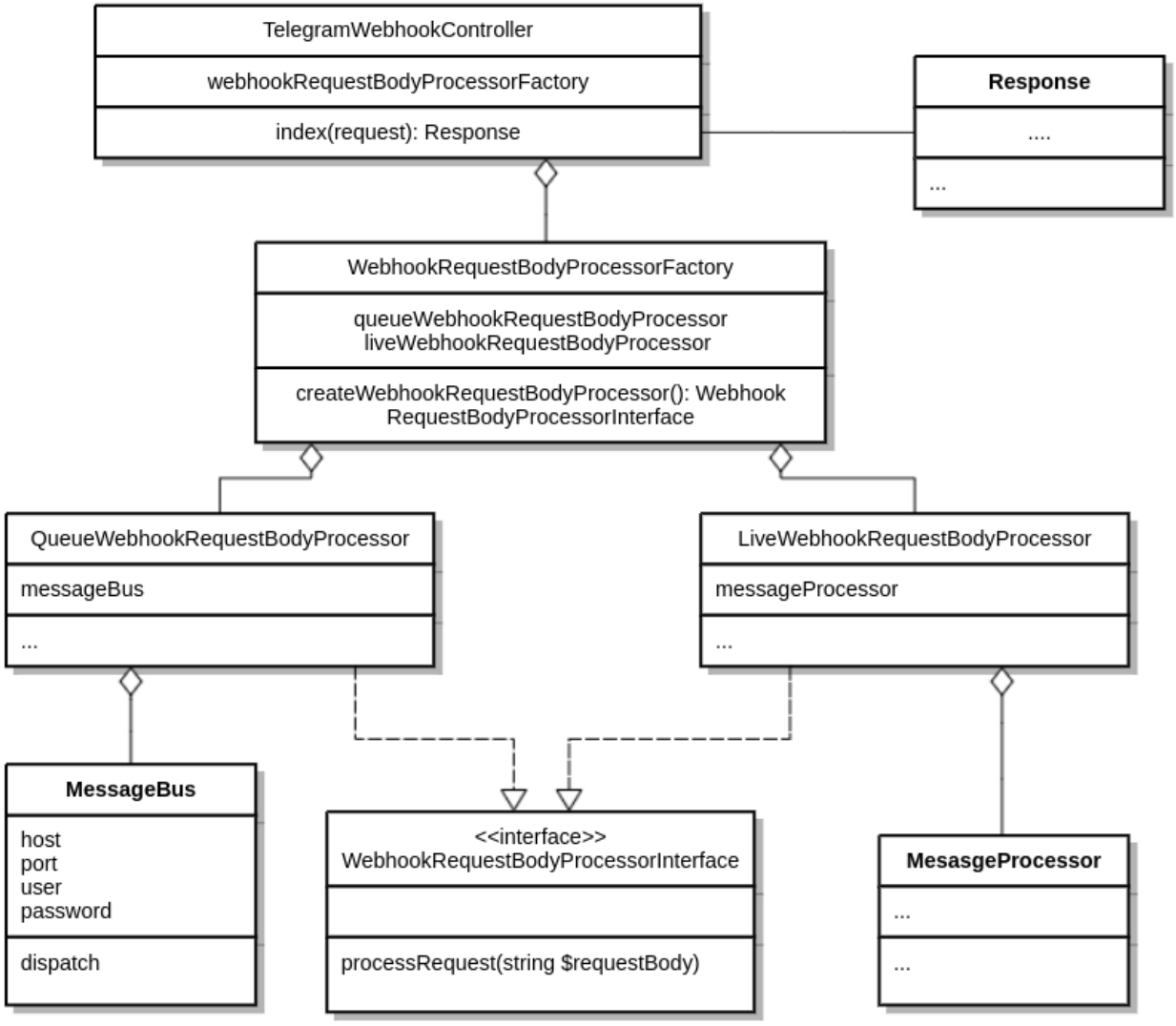


Рисунок 2.2.1 - Діаграма класів підсистеми прийому повідомлень

Коли повідомлення потрапляє до класу «MessageProcessor», то тут уже починається робота підсистеми опрацювання повідомлень користувачів. Структура класів цієї підсистеми зображено на рисунку 2.2.2.

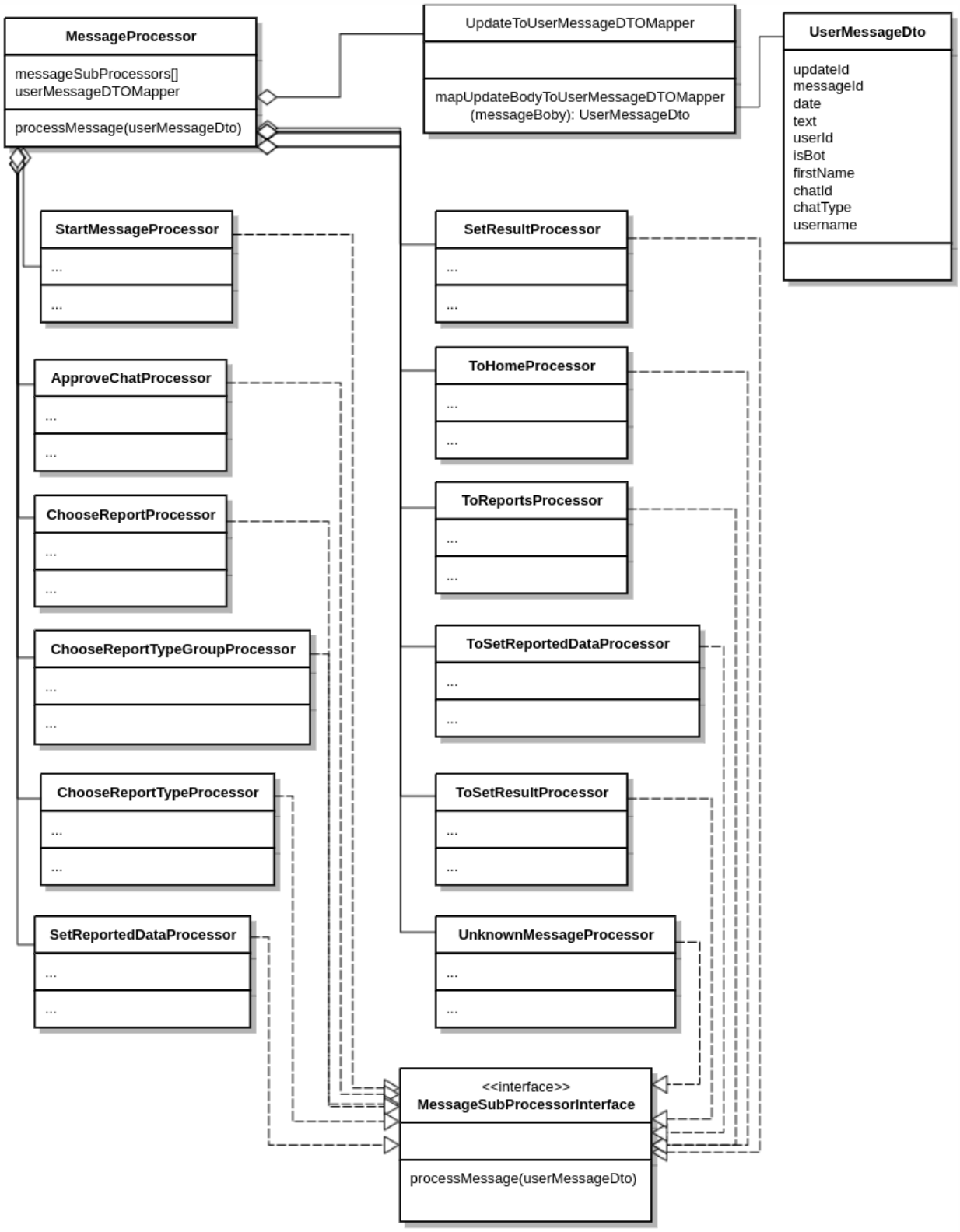


Рисунок 2.2.2 - Діаграма класів підсистеми опрацювання повідомлень

В цю підсистему повідомлення може потрапити з брокера повідомлень, який використовується для відкладеної обробки, а також у випадку недоступності брокера під час отримання повідомлення, що вимагає негайного опрацювання запиту, щоб його не втратити. Для трансофрмації запиту із повідомленням в клас «UserMessageDTO», використовується мапер «UpdateToUserMessageDTOMapper». Це необхідно для зручнішої роботи із даними запиту. Після отримання об’єкта повідомлення, він потрапляє на подальше опрацювання до об’єктів класів що реалізовують інтерфейс «MessageSubProcessorInterface». Основний клас обробки визначає послідовність та необхідність опрацювання повідомлення певною реалізацією інтерфейсу. В залежності від дії, яка очікується від користувача, та статусу користувача в системі визначається потрібний обробник повідомлення.

Обробник повідомлень «StartMessageProcessor» використовуєтсья у випадку коли користувач реєструється в системі вперше і вводить боту обов'язкову команду «/start». Цей обробник повинен ідентифікувати, що за користувач прислав це повідомлення, перевірити чи воно актуальне і створити сутність чату в БД. Ця сутність буде використовуватися для комунікації із працівником, а саме дані цієї сутності будуть використовуватися у запитах на Telegram API. Опис АПІ можна знайти на цьому ресурсі [6].

Після реєстрації в системі, потрібно виконати підтвердження. Для цього є обробник «ApproveChatProcessor», який отримує повідомлення і звіряє чи його текст відповідає таємному ключеві реєстрації.

Коли працівник хоче ввести результати своїх робіт і вибрав відповідний тип, то його повідомлення буде опрацьовано обробником «SetResultProcessor», що використовується при запиті на збереження результатів виконаних робіт. Перед збереженням застосовується перевірка формату введення. Якщо число не вірне, то обробник відправить користувачеві повідомлення про це.

Якщо в користувача буде необхідність ввести результати не за поточною датою, то у нього є можливість вказати іншу звітну дату. Обробник

«SetReportedDateProcessor» застосовується для встановлення цієї дати в системі. Перед її встановленням, виконується перевірка формату чи користувач увів її вірно.

У користувача є можливість отримувати звіти про виконані роботи за певний період. Обробник «ChooseReportProcessor» застосовується для встановлення цього звітного періоду. У результаті виконання інформація за потрібний період буде надіслана користувачеві.

Обробник «UnknownMessageProcessor» використовується у випадках коли повідомлення надіслане користувачам не знайоме для системи і не може бути опрацьоване.

Інші обробники виконують навігаційні ролі. Тобто відповідають користувачам певним описовим повідомленням і змінюють клавіатуру для виконання наступних дій. Деякі дії є обов'язковими після зміни клавіатури, тож обробники визначають вказівники на дії в сутності чату, щоб система при наступному повідомленні від користувача знала які дії потрібно виконати.

Всі класи обробників описані вище, відправляються користувачам повідомлення у відповідь із результатами. Для цього використовується підсистема спілкування із ботом. Діаграму класів цієї підсистеми показано на рисунку 2.2.3.

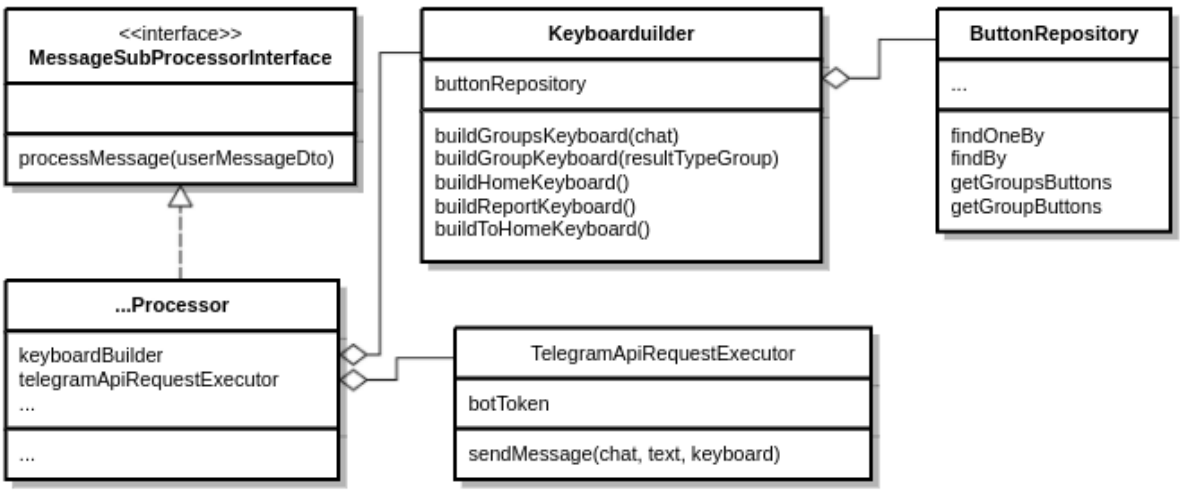


Рисунок 2.2.3 - Діаграма класів підсистеми спілкування із ботом

Для відправки повідомлень через Telegram API використовується клас «TelegramApiRequestExecutor». Цей клас містить метод, який приймає параметрами текст повідомлення, яке відображається у відповідь, клавіатуру з кнопками, які буде доступні, та об’єкт чату на який потрібно все це відправити. Тож кожний обробник повинен згенерувати клавіатуру, якщо її потрібно змінити, та текст, що потрібно відобразити користувачеві. Для генерації клавіатури використовується клас «KeyboardBuilder». Цей клас містить різні методи, які застосовуються різними обробниками. Всі методи повертаються різні набори кнопок. Деякі набори містять загальні кнопки, які можуть використовуватися для навігації. Кнопки отримуються із бази даних за допомогою класу «ButtonRepository».

Підсистема для опрацювання запитів із веб-інтерфейсу є досить великою, бо міститиме багато однакових класів, які будуть використовуватися для налаштування компонентів системи. Такими компонентами системи є: користувачі системи (працівники), типи робіт, категорії робіт, кнопки клавіатури. Діаграма класів для налаштування компонента типів робіт зображена на рисунку 2.2.5.

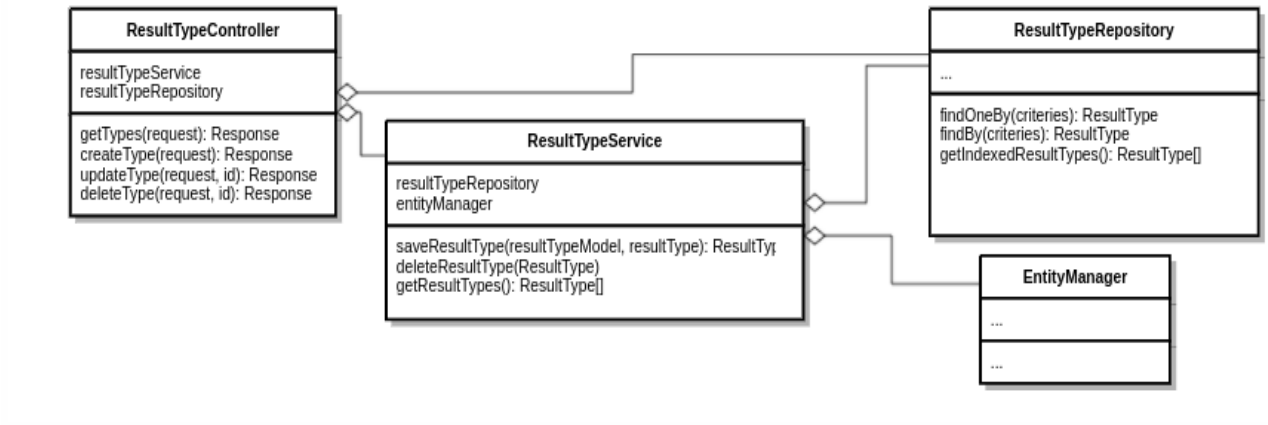


Рисунок 2.2.4 - Діаграма класів для налаштування типів робіт

Контролер «ResultTypeController» є класом, який містить методи для різних дій над компонентом . Ці методи є вхідними Rest API точками в систему.

Безпосередньо, всі дії над компонентом виконуються в класі «ResultTypeService». Він виконує трансформацію одних транспортних класів компонента в інші і виконує збереження даних в БД. Для пошуку компонентів застосовується клас «ResultTypeRepository». Для видалення і збереження в БД застосовується об’єкт класу «EntityManager».

Налаштування всіх інших компонентів виконуються за схожою схемою, тож діаграма класів буде однакова для всіх за відмінності назв класів і методів.

Також ця підсистема містить логіку для генерації звітів виконаних робіт. Структура класів цієї підсистеми зображена на рисунку 2.2.5.

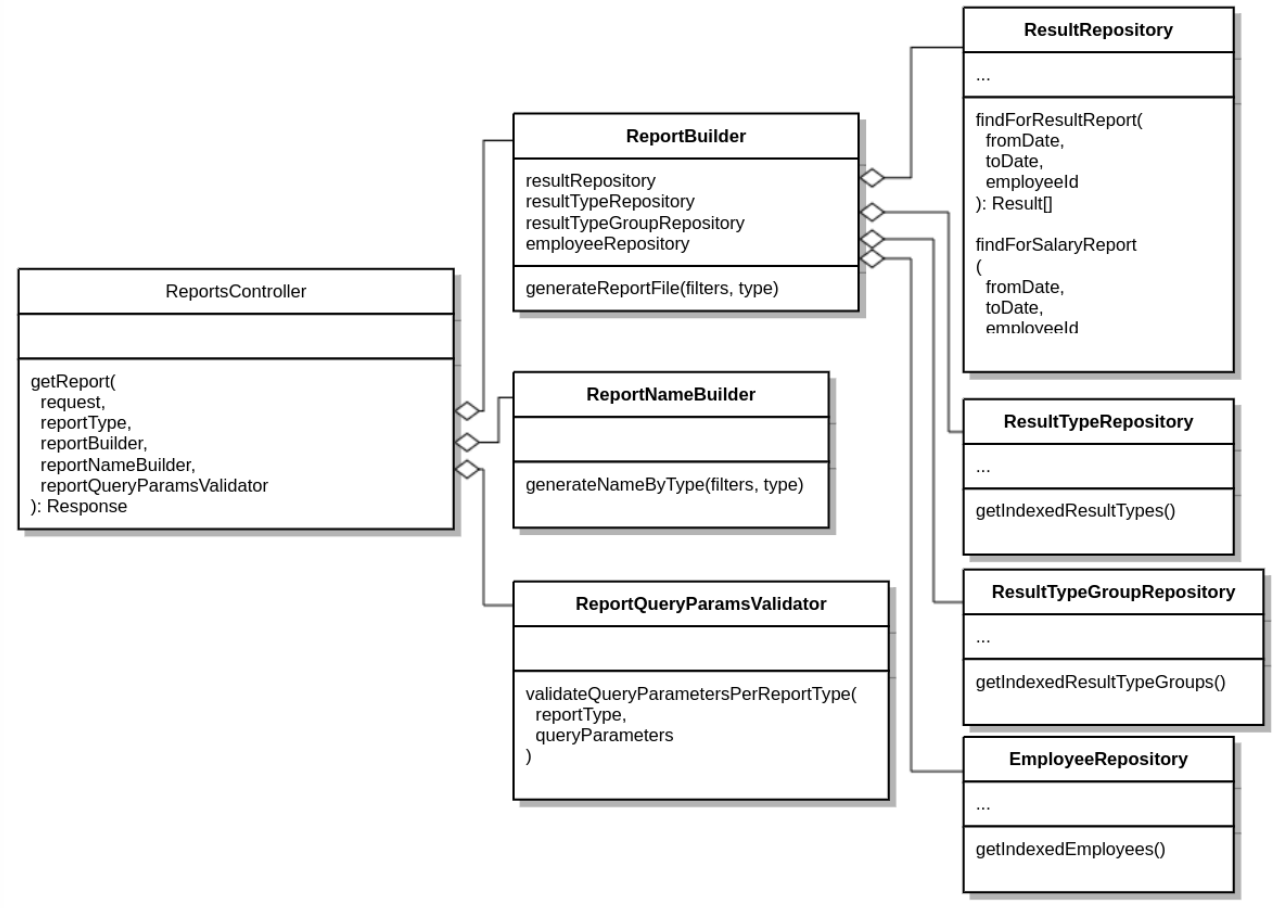


Рисунок 2.2.5 - Діаграма класів генерування звітів

Вхідною точкою для генерації є клас «ReportsController». Цей клас делегує генерацію звіту на клас «ReportBuilder». Він використовує всі необхідні репозиторію для отримання даних із БД. Вся отримана інформація формується у

відповідні структури даних відносно критеріїв генерування звітів, які були передані із веб-інтерфейсу. Всі створені структури поміщаються у CSV файл, який представляє собою звіт. Перед створенням звіту, всі параметри генерування, що передані із веб-інтерфейсу, валідуються. Тобто система не дозволить згенерувати звіт, так як параметри є некоректними, тому його побудова неможлива. Коли звіт уже згенеровано, то клас «ReportNameBuilder» застосовується для створення імені файлу який буде завантажено користувачем.

Отже було виконано декомпозицію системи на підсистеми. Визначено всі їхні складові і дані з якими вони працюють. Все це було зображено в об'єктних моделях - діаграмах класів.

2.3 Опис структури даних та моделі бази даних

Для представлення даних використовують логічну модель (LDM). Логічна модель даних забезпечує детальний огляд всього набору даних, створених та підтримуваних організацією. Це схематичне представлення даних, що необхідні для реалізації ПЗ, незалежно від основної технології бази даних. Логічна модель даних, як правило, складається з сутностей даних та зв'язків між ними. Це спосіб визначення даних організації та правил ведення бізнесу, що регулюють відносини між ними. LDM також допомагає у створенні фізичної моделі даних та надає "дорожню карту" для її проектування фізичної бази даних. Логічну модель даних зображено на рисунку 2.3.1.

Компонент моделі «employee» представлятиме собою сутність працівника підприємства. Вона міститиме всі необхідні дані про працівника, які необхідні для функціонування системи. Компонент «chat» зберігатиме бані про сутність чату в системі Telegram, яка ідентифіковуватиме певного користувача системи. Ця інформація буде використовуватися у спілкуванні із Telegram API. Між компонентами «employee» і «chat» є зв'язок «один-до-багатьох». Цей зв'язок показує, що один користувач може мати кілька чатів. Це необхідно для

майбутнього покращення системи, коли працівники матимуть змогу перереєстровуватися в інших облікових записах Telegram.

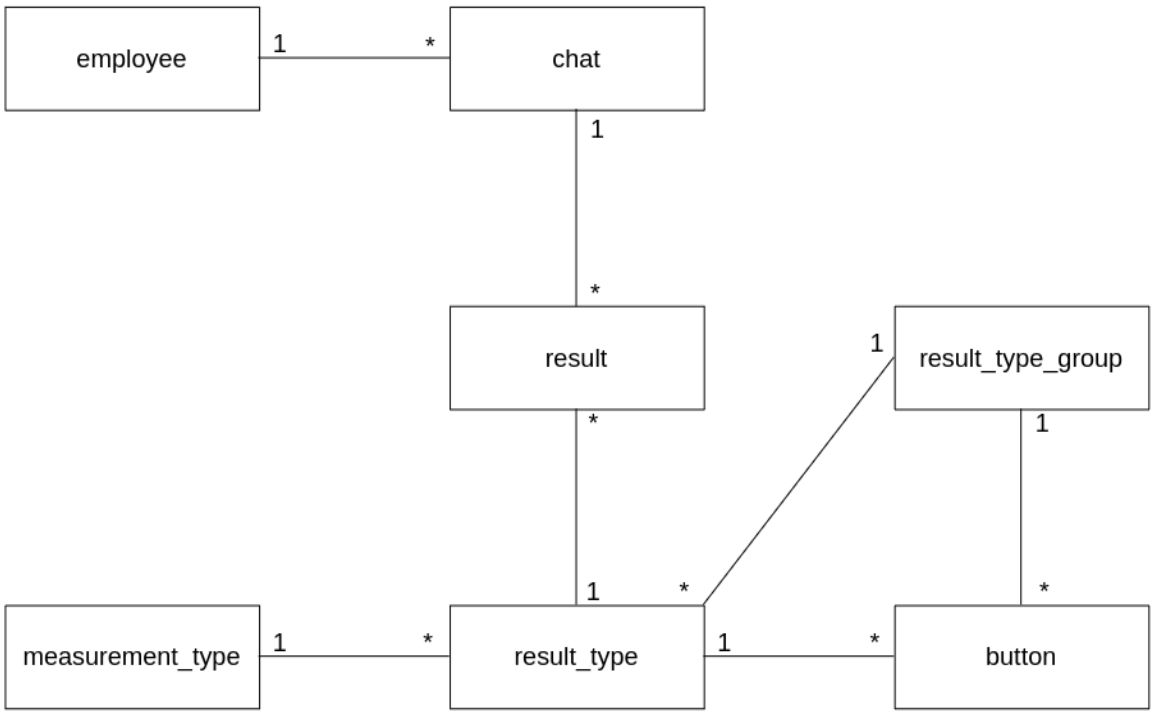


Рисунок 2.3.1 - Логічна модель даних

Компонент «result» зберігатиме результати виконаних робіт. Тут зберігатимуться безпосередньо самі значення результату, зв'язок до чату в якому ці результати було вказано і зв'язок до типу роботи, який визначає метод опрацювання і відображення результатів на інтерфейсі користувача. Зв'язок необхідний до компоненту «chat», а не до «employee» лише із метою мати потім можливість відслідкувати, який обліковий запис телеграму використовувався для введення результатів працівником.

Компонент «result_type» представляє собою тип робіт про який можна звітувати в системі. Він містить зв'язок до компонента, що представляє собою тип вимірювання одиниць виконаних результатів – «measurement_type». Також вказівник на категорію типів робіт, який необхідний для групування робіт при відображенні – «result_type_group».

Компонент «button» відображає об’єкти кнопок які будуть використовуватися для побудови клавіатури, за допомогою якої буде виконуватися спілкування працівників із ботом. Цей компонент має посилання на типи робіт і категорії. Ці зв’язки необхідні щоб сервер міг зрозуміти як реагувати на натискання тієї чи іншої кнопки.

Всі логічні структури даних мають бути представлені в фізичній моделі для детальнішого представлення даних, але спершу необхідно визначитися із типом бази даних: реляційна БД, не реляційна БД.

Реляційна база даних зберігає дані в таблицях. Таблиця складається з рядків і стовпців. Де, рядки представляють кожну сутність, а стовпці представляють атрибути. Таблиці асоційовані один з одним. Можна нормалізувати дані в таблицях, щоб мінімізувати надмірність даних. Крім того, мова структурованих запитів (SQL) допомагає запитувати дані в реляційній базі даних. Основна особливість – надійність та незмінність даних, низький ризик втрати інформації. При оновленні даних їхня цілісність гарантується, вони заміняються у одній таблиці. Реляційні бази даних, на відміну від не реляційних, відповідають ACID – це вимоги до транзакційних систем:

– atomicity;
– consistency;
– isolation;
– durability.

Відповідність до цих вимог гарантує цілісність та збереження даних, а також передбачуваність роботи бази даних. Детальніше про ці вимоги описано в роботі про SQL [7].

Не реляційна база даних (NoSQL) – зберігає дані без чітких зв’язків між собою та без чіткої структури. Замість структурованих таблиць, всередині бази знаходиться безліч різнорідних документів, в тому числі і зображення, відео та навіть публікації у соціальних мережах. На відміну від реляційних БД, NoSQL бази не підтримують SQL запити. Схема даних є динамічною та може

змінюватись у будь який момент часу. До даних складніше отримати доступ. Проте такі СУБД відрізняються швидкістю та продуктивністю. Фізичні об'єкти у NoSQL зазвичай можна зберігати прямо у тому вигляді, у якому з ними потім працює додаток. Бази даних NoSQL підходять для зберігання великих об'ємів неструктурованої інформації, а також для швидкої розробки та тестування гіпотез. У них можна зберігати дані будь яких типів та додавати нові в процесі роботи. Детальніше про NoSQL в цій роботі [8].

Отже було обрано реляційний спосіб представлення даних. Реляційні бази даних допомагають досягти складних запитів, вони забезпечують гнучкість і допомагають аналізувати дані. Фізична модель бази даних представлено на рисунку 2.3.2 у форматі таблиць реляційної БД. Опис таблиць цієї фізичної моделі даних представлений на таблиці 2.3.1.

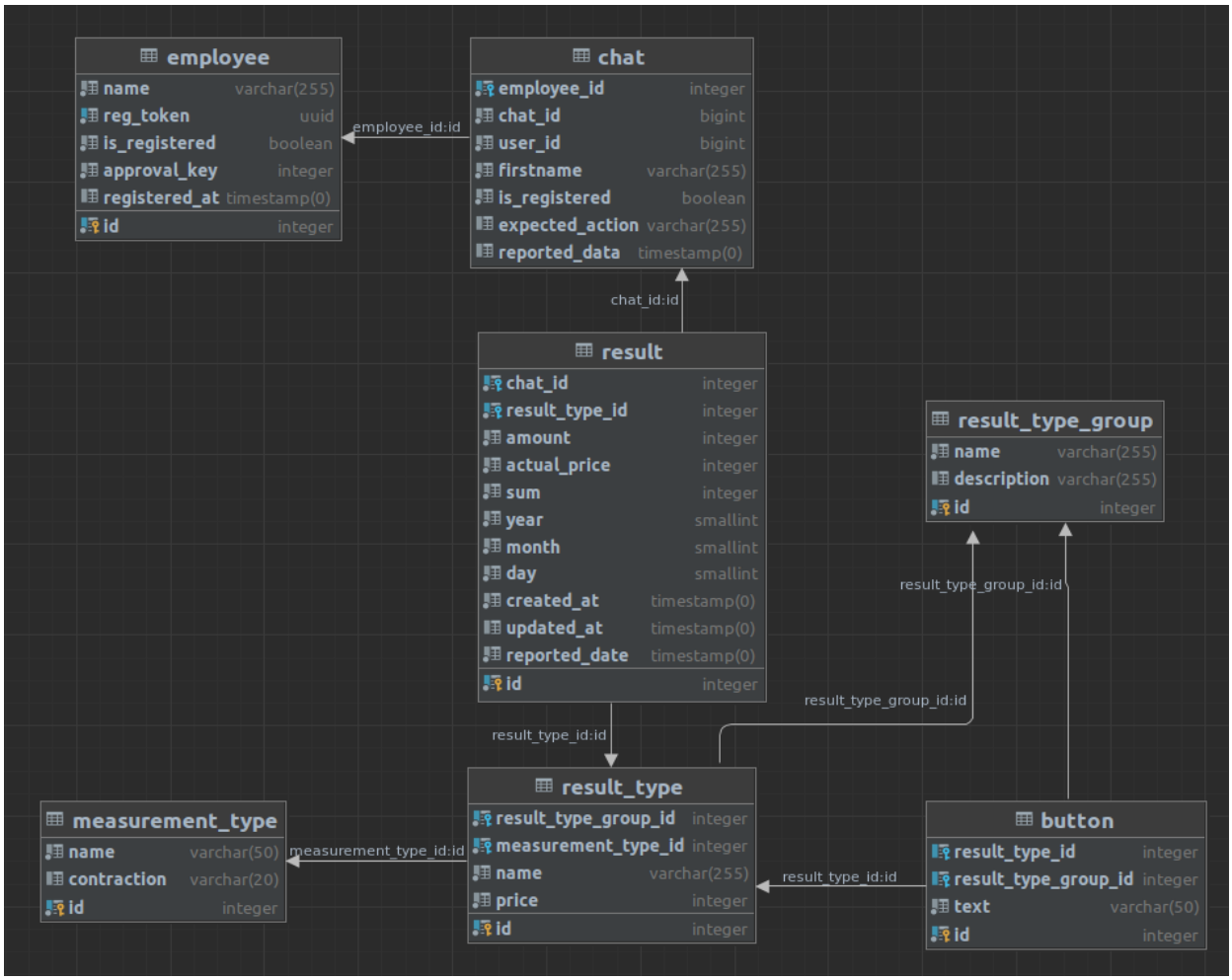


Рисунок 2.3.2 - Фізична модель бази даних

Таблиця 2.3.1 - Опис таблиць фізичної моделі даних

Назва таблиці	Дані збережені в таблиці
1	2
employee	– назва що ідентифікує працівника; – унікальне реєстраційне посилання; – ключ підтвердження реєстрації; – вказівник на статус реєстрації; – дата завершення реєстрації.
chat	– посилання на таблицю «employee»; – ідентифікатор чату в системі телеграму; – ідентифікатор користувача в системі телеграму; – назва користувача в системі телеграму; – вказівник на статус реєстрації; – назва очікуваної дії працівника в спілкуванні з ботом; – дата звітності працівника, яка встановлюється за необхідності.
result_type_group	– назва категорії; – опис до категорії.
result_type	– посилання на таблицю «result_type_group»; – посилання на таблицю «measurement_type»; – назва типу роботи та ціна.
measurement_type	– назва типу вимірювання; – скорочення назви типу вимірювання.
button	– посилання на таблицю «result_type»; – посилання на таблицю «result_type_group»; – текст кнопки, що відображається на клавіатурі бота при відображенні категорій і типів робіт.

Кінець таблиці 2.3.1

1	2
result	– посилання на таблицю «chat»; – посилання на таблицю «result_type»; – кількість виконанної роботи; – актуальна ціна на момент звітування; – вартість виконаної роботи; – рік, місяць, день звітного періоду; – дата звітного періоду; – дата добавлення запису в таблицю; – дата оновлення запису в таблиці.

Отже було описано структуру даних, яка необхідна для реалізації ПЗ. Структура містить всі необхідні компоненти і зв’язки між ними, що дозволять функціонувати розробленій системі в повній мірі.

2.4 Проектування користувацької частини

Працівники підприємства - це користувачі розроблюваної системи. Вони користуватимуться системою за допомогою телеграм-бота. Найпростіший спосіб використовувати бота - це використання клавіатури із кнопками, або ще так звані «меню». В поєднанні із можливістю вводити текст незалежно від того, який вводиться за допомогою кнопок, система буде спроможна виконати всі покладені на неї задачі за допомогою телеграм-бота. Так як основними елементами управління комунікації є клавіатури, то необхідно виконати проектування їхньої структури. Ці клавіатури будуть інтерфейсом користувача.

Після реєстрації користувача в системі, йому буде доступна головна клавіатура. Це меню міститиме дві кнопки, які дозволять користувачу перейти в режим введення результатів або в режим перегляду підсумкових результатів. Макет цієї клавіатури зображена на рисунку 2.4.1.

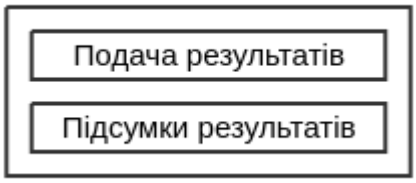


Рисунок 2.4.1 - Макет клавіатури головного меню

В режимі введення результатів для користувача буде відображено клавіатуру що міститиме перелік кнопок які представляють категорії типів робіт. Такі кнопки і самі категорії необхідні, щоб було зручніше виконувати навігацію між типами робіт. Також меню містить ще дві кнопки. Одна призначена для повернення до головної клавіатури і виходу із режиму введення результатів. Інша для переходу в режим встановлення звітної дати. Якщо не натискати одну із цих двох функціональних кнопок, то вибір категорії є обов'язковим. Макет цієї клавіатури зображена на рисунку 2.4.2.

Рисунок 2.4.2 - Макет клавіатури вибору категорії типів робіт

Після вибору категорії з'являється клавіатура вибору типу роботи, що відноситься до обраної категорії. На клавіатурі будуть кнопки, що представляються собою певні типи робіт. Потрібно зауважити, що текст кнопок для типів робіт повинен відрізнятися навіть між типами робіт, що відносяться до різних категорій. Цієї умови потрібно дотримуватися, щоб не виникло плутанини між типами робіт під час їх ідентифікації на сервері. Також на

клавіатурі є дві функціональні кнопки за допомогою яких можна повернутися до переліку категорій або до головного меню. Якщо жодна із цих двох кнопок не натискалася, то вибір типу роботи є обов'язковим. Макет цієї клавіатури зображено на рисунку 2.4.3.

Рисунок 2.4.3 - Макет клавіатури вибору типу роботи

При натисненні кнопки для підсумків результатів, що на головному меню, буде відображено клавіатуру на якій є кнопки, що дозволяють отримати інформацію про виконані роботи за певні періоди. Також буде доступна кнопка повернення до головного меню. Макет цього меню зображено на рисунку 2.4.4.

Рисунок 2.4.4 - Макет клавіатури для вибору звітного періоду

Було спроектовано інтерфейс користувачів, що є працівниками підприємства. Інтерфейсом для керівника підприємства буде веб-інтерфейсом. Особливих вимог до інтерфейсу немає, тож буде використано прості та загальні

підходи до створення компонентів адміністративної панелі розроблюваної системи.

Для зручності навігації, меню буде присутнє на всіх сторінках адмін. панелі. Загальний шаблон сторінок зображено на рисунку 2.4.5. Всі наступні прототипи які будуть описані, розміщуватимуться на сторінках з правої сторони від навігації в робочій зоні.

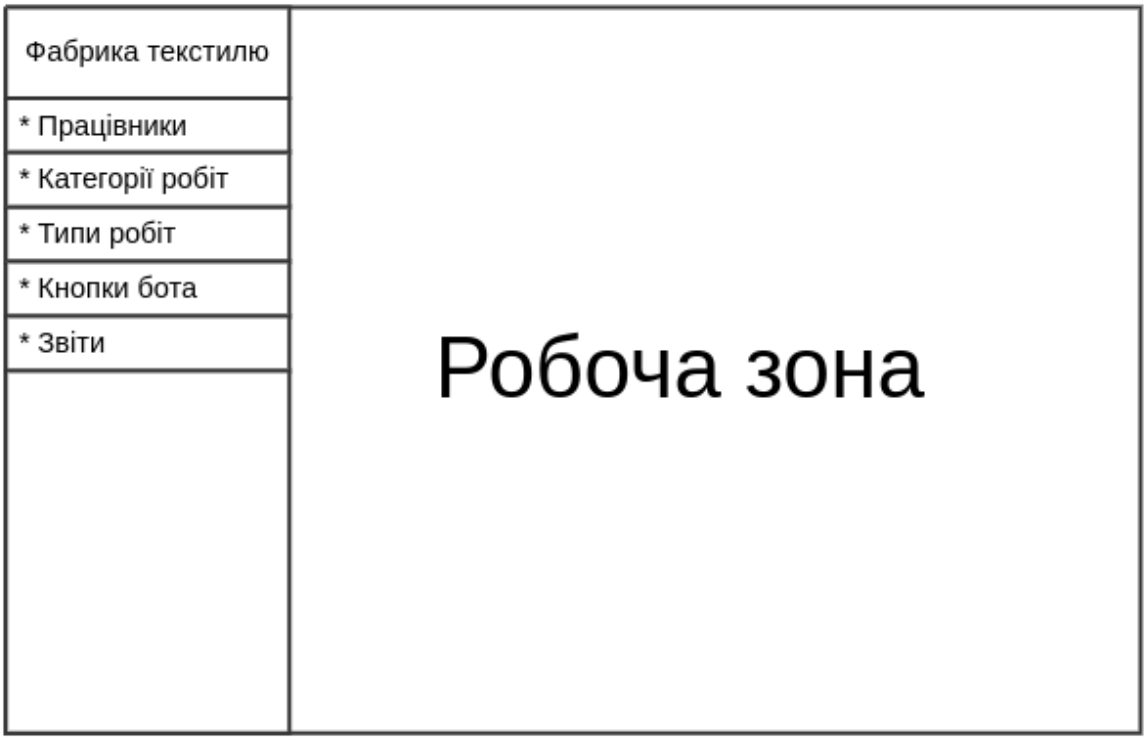


Рисунок 2.4.5 - Макет головного шаблону сторінок адмін. панелі

Сторінка налаштування типів робіт буде містити таблицю із існуючими типами, кнопку що відкриватиме модальне вікно для створення нового запису та заголовок сторінки. Прототип цієї сторінки зображено на рисунку 2.4.6. Прототип модального вікна для створення нових записів зображено на рисунку 2.4.7.

Всі інші сторінки редагування компонентів системи використовуватимуть всі ті самі компоненти, тож структура прототипів буде однакова за відмінності колонок в таблиці відображення записів і полів форми на модальному вікні.

Рисунок 2.4.6 - Прототип робочої зони сторінки відображення типів робіт

Рисунок 2.4.7 - Прототип форми, що застосовується для створення і редагування типів робіт

Сторінка створення звітів містить форму із полями для вводу параметрів, що використовуються для генерації звіту. На сторінці також присутні кнопки, які дозволять швидко виконати заповнення полів із датами звітного періоду. Прототип цієї сторінки зображено на рисунку 2.4.8.

Рисунок 2.4.8 - Прототип сторінки створення звітів

Було розроблено прототипи користувацьких інтерфейсів, які можуть використовуватися під час розробки дизайну веб-інтерфейсу та генерації структури кнопок бота під час відповідей користувачам.

2.5 Аналіз та вибір технологій для реалізації програмної системи

Виконавши аналіз існуючих технологій і засобів для реалізації Rest API застосунків та веб-інтерфейсів, було визначено наступні технології, що дозволять реалізувати розроблюване ПЗ у найкращій формі:

– Symfony 5 (фреймворк на мові програмування PHP):
– система управління реляційними базами даних PostgreSQL;
– брокер повідомлень RabbitMQ;

– технологія контейнеризації Docker та сервіс обгортка над нею Docker-Compose, який спрощує його використання;

– фронтенд фреймворк Vue.js та пакет Vuetify;

– об’єктно реляційний мапер Doctrine 2 для роботи із БД;

– система контролю версій Git;

– GitHub - репозиторій для збереження проектів, що використовують Git.

Symfony 5 - це високопродуктивний PHP фреймворк, що дозволяє створювати RestAPI. Дуже зручний у застосуванні, добре задокументований, має багато вже готових рішень, які дозволять уникнути роботи із тривіальними задачами налаштування взаємодії між різними сервісами системи. Про те як працює PHP можна ознайомитися у цій роботі [9], а із Symfony 5 в цій роботі[10].

Doctrine 2 - це об’єктно реляційний маппер, що використовується як адаптер між базою даних та програмним кодом. Ця технологія широко використовується фреймворком Symfony. Вона надає вже готові рішення для роботи і перетворення даних із класів в таблиці БД.

PostgreSQL - це система управління реляційними базами даних. Ця система широко використовується у високонавантажених проектах. Вона дуже схожа із MySQL, оскільки відносяться до одного сімейства реляційних БД, проте забезпечує користувачів більшим спектром доступних функцій та більшою продуктивністю.

RabbitMQ - це повноцінний брокером повідомлень що дозволяє виконувати асинхронну передачу повідомлень. Його головна ціль приймати повідомлення і віддавати їх тим хто їх очікує. В програмному продукті що буде розроблятися, цей брокер буде використовувати для отримання повідомлень від бота, та передавати їх на опрацювання. Детальніше про цей брокер у статті [11].

Docker - інструмент для контейнеризації. Сьогодні, у розробці ПЗ він широко використовується, так як забезпечує розробників швидким методом

запуску потрібно середовища для розробки програмних продуктів. Дуже швидко можна підключити новий сервіс за необхідності, замість того щоб встановлювати такі сервіси локально або переключатися між версіями, якщо різні проекти вимагають різні версії одних і тих самих сервісів. Більш детальніше для чого потрібен Docker описано в статті [12].

Git - система контролю версій. Це консольна утиліта для відслідковування і ведення змін файлів у проектах. Найчастіше використовується для коду, але також може використовуватися і для інших файлів. За допомогою Git можна відкачувати зміни до більш старих версій, порівнювати, аналізувати і зливати свої зміни до репозиторіїв.

Vue.js - прогресивний JavaScript фреймворк, доступний, продуктивна та універсальна інструмент для створення веб-інтерфейсів користувача. Vuetify - це бібліотека для Vue.js з чудово виготовленими вручну компонентами які використовуються в побудові SPA. Не потрібні навички дизайну - все, що потрібно для створення дивовижних програм, у вас під рукою.

В цьому розділі було визначено архітектурні підходи, що будуть реалізовані в розроблюваному ПЗ. Виконано декомпозицію системи на підсистеми, для кращого визначення вмісту кожної із них та встановлення зв’язків між ними та в середині них. На основі цього було спроектовано моделі даних, що візуально представляють всі компоненти системи і інформацію яка їм буде належати. Також спроектовано макети інтерфейсів користувачів, що дають наглядне представлення системи у використані, визначення головних компонентів інтерфейсів. Для реалізації всіх компонентів системи, що було визначено під час проектування, також було обрано набір технологій, методів і рішень які будуть застосовуватися під час розробки ПЗ.

3 Програмна реалізація

3.1 Детальне проектування модулів

Для реалізації серверної частини, налаштування взаємодії всіх сервісів системи між собою та впровадження всіх функцій необхідно визначити модулі, або так звані пакети, що будуть використовуватися під час розробки.

Оскільки для реалізації серверної частини було обрано фреймворк Symfony 5, то встановлення модулів буде виконуватися за допомогою менеджера пакетів Composer для мови програмування PHP. Нище наведено перелік основних бібліотек які необхідні для реалізації.

Найголовніший пакет в будь якому проекті розроблюваному на фреймворку Symfony є «symfony/symfony». Цей пакет є головним, оскільки він містить в собі “серцевий” код фреймворку і всі необхідні для нього дочірні залежності, без яких функціонування фреймворку неможливе. Цей пакет не потрібно встановлювати окремо, так як він завантажується під час створення проекту. Також під час створення проекту встановлюються додаткові бібліотеки, які необхідні для забезпечення розробника зручними методами встановлення додаткових модулів, за допомогою виконання «рецептів», та виконання консольних команд, які можуть бути добавлені будь яким іншим окремими модулями, в тому рахунку і головним пакетом «symfony/symfony». Цими пакетами є «symfony/console» та «symfony/flex».

Також модулі, що дозволяють встановлювати налаштування для всіх інших модулів. Цими пакетами є «symfony/dotenv» та «symfony/yaml». «symfony/dotenv» впроваджує функції, що дозволяють виконувати читання конфігурацій із середовища, або із «.env» файлів. Цей спосіб збереження конфігурацій сьогодні широко застосовується багатьма проектами. «symfony/yaml» використовується проектом для визначення конфігурацій проекту, встановлення залежностей між класами, якщо таку не можуть буди виконані за допомогою методу «DependencyInjection». Всі вище згадані модулі

встановлюються окремими пакетами від «кору», так як можуть бути використанні як самостійні модулі в будь якому PHP проекті, що не використовує Symfony.

Перелік допоміжних модулів потрібно продовжити із пакетів, які застосовуються для роботи і комунікації із базою даних: «doctrine/annotations», «doctrine/doctrine-bundle», «doctrine/doctrine-migrations-bundle», «doctrine/orm». Ці модулі забезпечують функціями спілкування із БД, встановлення з'єднання, трансформації класів з даними в запити SQL для запису даних в БД, створення міграцій та можливість їх використати, визначення структури таблиць , використовуючи анотації в PhpDoc над властивостями класів сутностей, виконання запитів на вибірку до БД та інше.

Пакет «guzzlehttp/guzzle» необхідний для створення HTTP клієнта, який буде виконувати запити на Telegram API при відправці повідомлень до користувачів системи.

Бібліотека «ramsey/uuid» використовується для генерування унікальних стрічок коду, які ще називаються унікальними ідентифікаторами (UUID). Ці ідентифікатори будуть використовуватися під час створення унікального реєстраційного посилання для користувачів системи.

Під час опрацювання запитів із веб-інтерфейсу необхідно виконувати перевірку вхідних даних, ще так звану валідацію даних. Для цього застосовуються модулі «symfony/form» та «symfony/validator». Логіка валідації знаходиться в пакеті «symfony/validator». Процес перевірки даних виконується над об'єктами форм, які є представленням даних, та визначаються правила перевірки для даних. Для створення таких об'єктів застосовується модуль «symfony/form». Він використовується як будівельник для структури форми та описує правила за якими слід перевіряти дані.

Після опрацювання запитів, сервер повинен повернути відповідь до користувача веб-інтерфейсу. Для цього необхідно виконувати трансформацію даних із об'єктів у необхідний формат, що передасться клієнту як JSON.

Трансформація об’єктів в масиви даних виконується за допомогою пакета «symfony/serializer». Ця бібліотека забезпечує кілька варіантів трансформацій даних, які можуть бути використані у різних проектах.

Модуль «symfony/messenger» забезпечує систему засобами для комунікації із різними типами брокерів та сервісів кешування даних, що можуть застосовуватися як брокери. Цю бібліотеку було обрано для роботи із брокером повідомлень RabbitMQ, так як комунікація із ним буде здійснювати дуже зручним способом, що не вимагатиме на це великих зусиль. Також надається простий спосіб налаштування черг і їх обробників. Це не вимагатиме у розробника додаткових знань у налаштуванні комунікації із брокером на низькому рівні. Передача повідомлень між цим модулем і основною програмою здійснюється за допомогою об’єктів окремих класів. Назви цих класів застосовуються у налаштуванні обробників повідомлень брокера. Цих обробників ще називають консьюмерами.

Опишемо в деталях процес створення об’єктів типів робіт. В ньому будуть задіяні майже всі із перерахованих вище пакетів. Цей процес зображено на рисунку 3.1.1. Цю схему можна розглянути в більму масштабі в додатку Б.

Вхід в систему здійснюється через клас контролера, який є реалізацією компонента, що опрацьовується “кором” фреймворку. Контролер створює форму, звертаючись до класу «FormBuilder». В об’єкт форми передаються вхідні дані, які разом потрапляють на валідацію. За результатами перевірки контролер визначає, чи потрібно повернути помилки на веб-інтерфейс, чи продовжити опрацювання даних і їх збереження в БД. Якщо помилок у даних не виявлено, то модель даних потрапляє до сервісу, який виконує трансформацію моделі в об’єкт Entity для збереження в БД. Перед збереженням сутності здійснюється перевірка на дублікат, чи такий тип робіт уже присутній. Для цього застосовується репозиторій. Для збереження використовується об’єкт EntityManager, який виконує трансформаціє ентіті в запит на добавлення даних в таблицю БД.

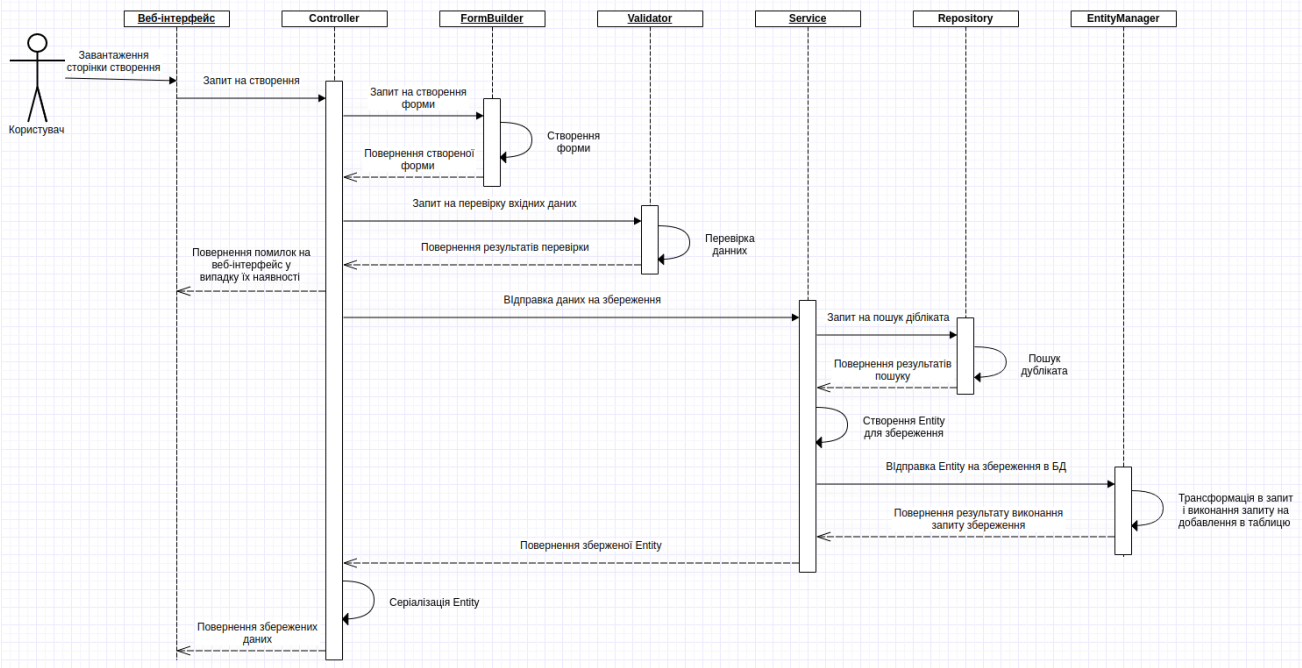


Рисунок 3.1.1 - Процес створення об’єкта типу робіт

Структура бази даних в проектах, що використовують модуль Doctrine, описується в класах, які називаються сутностями. Отже необхідно створити класи, що будуть представляти собою компоненти фізичної моделі даних яка зображена на рисунку 2.3.2. Продемонструємо приклад одного такого класу із налаштуваннями:

```
/**
* @ORM\Entity(repositoryClass=ResultTypeRepository::class)
*/
class ResultType
{
	/**
	* @ORM\Id
	* @ORM\GeneratedValue
	* @ORM\Column(type=«integer»)
	*/
	private $id;

	/**
	* @ORM\Column(type=«string», length=255)
	*/
	private $name;

	/**
	* @ORM\Column(type=«integer»)
	*/
	private $price;
```

```
        /**
        * @ORM\ManyToOne(targetEntity=ResultTypeGroup::class)
        * @ORM\JoinColumn(nullable=false)
        */
        private $resultTypeGroup;

        /**
        * @ORM\ManyToOne(targetEntity=MeasurementType::class)
        * @ORM\JoinColumn(nullable=false)
        */
        private $measurementType;

        ...
}
```

Всі інші класи буде зображено в додатку В, де розміщується увесь код програми. Для того щоб усі налаштування були трансформовані в таблиці і колонки БД необхідно створити міграції. Міграції баз даних - це механізм, який використовується для створення структури реляційних БД та наступних інткрементальних змін в неї. При використанні Doctrine, міграції це класи які мають два методи. Один для застосування змін, які необхідно виконати. Інший для відміни застосованих змін якщо це можливо. Ці зміни описані у вигляді SQL запитів до БД, що змінюють її структуру. Після змін налаштувань мапингу в класах сутностей, необхідно створити нову міграцію, яка змінить структуру таблиць відповідно. Це виконується за допомогою виконання консольної команди, яка буде імпортована та інтегрована разом із пакетами для роботи із Doctrine та буде запущена за допомогою засобів, які було встановлено в пакеті «symfony/console»: **`bin/console doctrine:migration:diff`**. Ця команда всього лише створить міграцію, яку потрібно виконати. Структуру класів міграцій можна подивитися в додатку В.1. Запуск виконання міграцій робиться за допомогою команди: **`bin/console doctrine:migration:migrate`**.

Веб-інтерфейс для користувача буде розроблено за допомогою фронтенд фреймворку Vue.js. Веб-застосунки які розробляються такими засобами, як цей фреймворк, називаються Single Page Application (SPA). Новий проект можна створити різними способами. Найзручніший за допомогою «vue/cli», який дозволяє в інтерактивному режимі вказати, які пакети необхідно встановити при

створенні проекту. Для реалізації будь якого SPA використовується багато базових пакетів для фронтенд розробки. Головні модуля, що необхідні для функціонування SPA, використовують базові за основу, тож немає необхідності вдаватися в глибину кожного пакету. Головними модулями є: vue, axios, vuex, vue-route. Пакет “vue” є основою фреймворку, що забезпечує застосунок необхідними функціями для його розробки. Бібліотека “axios” використовується для відправки запитів на сервер. Пакет “vue-route” дає механізми для навігації між сторінками додатку. Компонент “vuex” забезпечує зручний інструмент, для збереження даних на стороні браузера під час роботи із додатком.

В цьому підрозділі було визначено модулі системи, встановлено методи їх взаємодії і обов’язки. Це дає ще більш детальну картину системи, що дозволяє перейти безпосередньо до етапу реалізації системи.

3.2 Програмна реалізація модулів

Щоб розпочати програмування модулів необхідно підготовити середовище розробки, встановити всі необхідні сервіси та створити “пустий” проект. Для зручності налаштування середовища і сервісів, було вирішено використовувати інструмент контейнеризації Docker та обгортку над ним Docker-Compose. Тож для початку потрібно створити docker-compose.yml файл, що визначатиме необхідні сервіси та їхні налаштування.

Одним із сервісів який необхідний для функціонування є контейнер, що містить встановлену мову програмування PHP. Так як кожний проект потребує різних налаштувань, то потрібно створити Dockerfile, який буде використовується як образ для побудови контейнера і визначає, що повинно бути встановлено в середину контейнера. Вміст цього файлу буде розміщено в додатку В.3. Отже після налаштування контейнера потрібно зайти в середину і встановити інструмент для створення Symfony проектів, вона називається Symfony CLI. За допомогою цього інструменту можна досить просто створити новий проект. Це виконується за допомогою команди, яку необхідно виконати в

середині контейнера: **`symfony new report_bot_api`**. Після створення проекту, необхідно встановити всі модулі, які було згадано в попередньому підрозділ, за допомогою менеджера пакетів Composer. Це потрібно виконувати всередині контейнера так як для встановлення також необхідно щоб було доступне ядро мови програмування PHP.

Окрім сервісу із PHP, також потрібні сервіси для бази даних, брокера повідомлень та веб-сервера. Ці сервіси із відповідними налаштуваннями потрібно також добавити в файл docker-compose.yml. Після того, як контейнери для всіх сервісів побудовані і готові до роботи, можна виконувати налаштування конфігурацій на рівні проекту для взаємодії із цими сервісами. Налаштування доступу до цих сервісів виконується у спеціальному файлі .env.

Зазвичай першими розробляються класи сутностей, що представляють мапинг між таблицями БД та об'єктами цих даних в коді. Для цього використовуємо консольну команду, що в інтерактивному режимі допомагає створювати такі класи: `bin/console make:entity Name\\Space`. Після створення всіх сутностей, необхідно згенерувати міграцію і виконати її, щоб структура даних потрапила до бази даних. Після створення цих класів, які використовуються в системі для транспортування даних між підсистемами, можна розпочинати роботу над реалізацією підсистем та модулів розроблюваного ПЗ.

Вхідними точками майже для всіх підсистем є методи контролерів. Тож доцільно розпочинати розробку із контролерів. Кожен метод контролера повинен мати опис в анотаціях, що розміщується в PhpDoc, який визначає HTTP метод за допомогою якого буде виконуватися запит на цю кінцеву точку, та URL шлях до цієї точки із усіма параметрами, які необхідні для встановлення роутингу в системі. Для цього використовується анотація **`@Route`**. Реалізації всіх контролерів буде розміщено в додатку В.2. Ось продемонстровано структуру контролера на прикладі класу ResultTypeController:

```
/**
* @Route(path=«/api/result-types»)
*/
class ResultTypeController extends BaseApiController
{
    /**
    * @var \App\Service\Result\ResultTypeService
    */
    private $resultTypeService;

    /**
    * @var \App\Repository\Result\ResultTypeRepository
    */
    private $resultTypeRepository;

    /**
    * @param \App\Service\Result\ResultTypeService $resultTypeService
    * @param \App\Repository\Result\ResultTypeRepository $resultTypeRepository
    */
    public function __construct(
        ResultTypeService $resultTypeService,
        ResultTypeRepository $resultTypeRepository
    ) {
        $this->resultTypeService = $resultTypeService;
        $this->resultTypeRepository = $resultTypeRepository;
    }

    /**
    * @Route(methods={«GET»})
    *
    * @param \Symfony\Component\HttpFoundation\Request $request
    *
    * @return \Symfony\Component\HttpFoundation\Response
    */
    public function getTypes(Request $request): Response
    {
        ...
    }

    /**
    * @Route(methods={«POST»})
    *
    * @param \Symfony\Component\HttpFoundation\Request $request
    *
    * @return \Symfony\Component\HttpFoundation\Response
    */
    public function createType(Request $request): Response
    {
        ...
    }

    /**
    * @Route(methods={«PUT»}, path=«/{id}»)
    *
    * @param \Symfony\Component\HttpFoundation\Request $request
    *
    * @return \Symfony\Component\HttpFoundation\Response
    */
    public function updateType(Request $request, int $id): Response
    {
        ...
    }
```

```
    /**
    * @Route(methods={«DELETE»}, path=«/{id}»)
    *
    * @param \Symfony\Component\HttpFoundation\Request $request
    *
    * @return \Symfony\Component\HttpFoundation\Response
    */
    public function deleteType(Request $request, int $id): Response
    {
        ...
    }
}
```

Складовою модулів налаштування компонентів системи, є логіка створення і редагування компонентів. Саме в методах контролерів, які використовуються для цих дій, використовуються форми і валідація даних. Цей процес було візуально продемонстровано на рисунку 3.1.1. Ось покажемо фрагмент коду цього метода, який відображає послідовність використання об’єктів різних класів та як виконується обмін інформації між ними:

```
public function updateType(Request $request, int $id): Response
{
    $resultType = $this->resultTypeRepository->find($id);

    // Перевірка існування сутності, що має бути оновлена
    if (!$resultType) {
        return $this->json(['error' => 'ResultType was not found.'],
Response::HTTP_NOT_FOUND);
    }

    // Створення моделі із даними яка передається в форму
    $resultTypeModel = (new ResultTypeModel())
        ->setName($resultType->getName())
        ->setResultTypeGroup($resultType->getResultTypeGroup())
        ->setMeasurementType($resultType->getMeasurementType());

    // Створення форми
    $form = $this->createForm(ResultTypeType::class, $resultTypeModel, ['method'
=> 'PUT']);
    $form->handleRequest($request);

    if (!$form->isSubmitted()) {
        return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
    }

    // Валідація даних
    if (!$form->isValid()) {
        return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
    }
```

```
    //Виконання збереження отриманих даних із запиту
    $resultType = $this->resultTypeService->saveResultType($form->getData(),
$resultType);

    // Трансформація сутності в клас який буде використовуватися під час
серіалізації
    $resultTypeDto = (new ResultTypeDto())
        ->setId($resultType->getId())
        ->setName($resultType->getName())
        ->setPrice($resultType->getPrice() / 100)
        ->setMeasurementType($resultType->getMeasurementType()->getId())
        ->setResultTypeGroup($resultType->getResultTypeGroup()->getId());

    // Повернення відповіді на веб інтерфейс
    return $this->json(['resultType' => $resultTypeDto]);
}
```

Основна бізнес логіка будь-якої підсистеми виконується в сервісах. Для роботи із даними та їх пошуку, сервіси використовують класи репозиторіїв. Ці класи містять методи, які формують DQL запити із метою отримання даних із БД. Як писати DQL, і що це таке, описано на цьому ресурсі [13]. Для кожної сутності було створено окремий клас репозиторію, що містить специфічні запити на отримання даних. Ось показано приклад репозиторія із методом отримання результатів виконаних робіт для відображення цих даних в звіті:

```
class ResultRepository extends ServiceEntityRepository
{
    /**
    *                                                                    @var
\App\Repository\Telegram\ChatRepository|\Doctrine\Persistence\ObjectRepository
    */
    private $chatRepository;

    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Result::class);
        $this->chatRepository = $registry->getRepository(Chat::class);
    }

    public function findForResultReport(\DateTime $from, \DateTime $to, ?int
$employeeId): array
    {
        $chatIds = [];

        if ($employeeId) {
            $chats = $this->chatRepository->findBy(['employee' => $employeeId]);

            $chatIds = array_map(function (Chat $chat): int {
                return $chat->getId();
            }, $chats);
        }
```

```
        $query = $this->createQueryBuilder('result')
            ->select('IDENTITY(result.resultType)        as        resultTypeId,
result.reportedDate, SUM(result.amount) as amount, SUM(result.sum) as sum')
            ->andWhere('result.reportedDate BETWEEN :from AND :to')
            ->setParameter('from', $from)
            ->setParameter('to', $to)
            ->groupBy('result.reportedDate')
            ->addGroupBy('result.resultType');

        if ($chatIds) {
            $query->andWhere('result.chat in (:chatIds)')
            ->setParameter('chatIds', $chatIds);
        }

        return $query->getQuery()->getArrayResult();
    }
}
```

Запит, що продемонстровано вище як приклад методу репозиторія, використовується для отримання результатів за певний період для відображення в звіті про виконанні роботи. Отримання цих даних є першим кроком перед створенням файлу звіту. Дані, що будуть отримані із запиту, є у “сирому” форматі та виконано групування результатів за колонками тип роботи і звітна дата. Такий формат даних ще неготовий для того щоб потрапити у файл, тому над цим масивом даних виконуються підготовчі дії. Всі результати колекціонуються в окремі масиви, індексовані по типам робіт і по датам. Ось зображено частину коду метода, який попередньо описаний:

```
// Отримання результатів із БД
$results = $this->resultRepository->findForResultReport($fromDate, $toDate,
$employeeId);
$mappedResults = [];

// Переформатування отриманих даних
foreach ($results as $result) {
    $mappedResults[$result['resultTypeId']]['results'][$result['reportedDate']-
>format('d/m/Y')] = $result['amount'];

    $mappedResults[$result['resultTypeId']]['amount'] =
    (isset($mappedResults[$result['resultTypeId']]['amount']) ?
$mappedResults[$result['resultTypeId']]['amount'] : 0) + $result['amount'];

    $mappedResults[$result['resultTypeId']]['sum'] =
    (isset($mappedResults[$result['resultTypeId']]['sum']) ?
$mappedResults[$result['resultTypeId']]['sum'] : 0) + $result['sum'];
}

// Визначення заголовків звіту
$headers = [
['','','','','',],
```

```
['Тип роботи','Категорія робіт','Одиниця вимірювання','Загалом виконано за
період','До сплати за період',],
];

$isLastDayReached = false;
$currentDate = clone $fromDate;

// Цикл в якому виконується добавлення форматованих даних у файл
while (!$isLastDayReached) {
    $headers[0][] = $this->getDayOfWeek($currentDate->format('N'));
    $headers[1][] = $currentDate->format('d.m.Y');
    $currentDate->add(new \DateInterval('P1D'));

    $isLastDayReached = $currentDate->getTimestamp() > $toDate->getTimestamp();
}
```

Система буде забезпечувати користувача двома типами звітів. Один описаний вище. Інший тип звіту міститиме інформацію про зарлати які повинен отримати працівник за той чи інший звітній період. Ці звіти відрізняються групуванням даних, тому цей будується трішки за іншою логікою. Увесь код класу будівельника звітів буде розміщено в додатку В.3.

На цьому етапі роботи із проектом було виконано реалізацію програмних модулів системи. Всі кроки реалізації є логічно послідовні і повинні виконуватися однин за одним.

3.3 Керівництво користувача

Для запуску системи в користування необхідно здійснити необхідні налаштування системи. Тому керівництво користувача розпочнеться із опису користування веб-інтерфейсу, що призначений для керівника підприємства.

Першим кроком є добавлення працівників в систему. Після створення запису компонента працівник в базі даних, його користувач уже може реєструватися в системі за унікальним посиланням, яке він зможе отримати від керівника. Добавлення працівників виконується на сторінці працівників. Сторінку працівників зображено на рисунку 3.3.1. Для відкриття форми вводу даних про працівника, необхідно натиснути на кнопку із текстом “добавити працівника”. З’явиться вспливаюче вікно із формою. Форму зображено на рисунку 3.3.2.

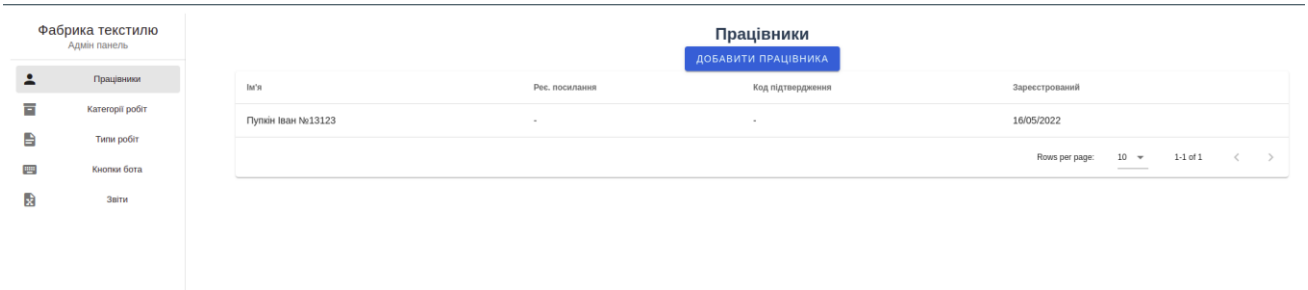


Рисунок 3.3.1 - Сторінка відображення списку працівників

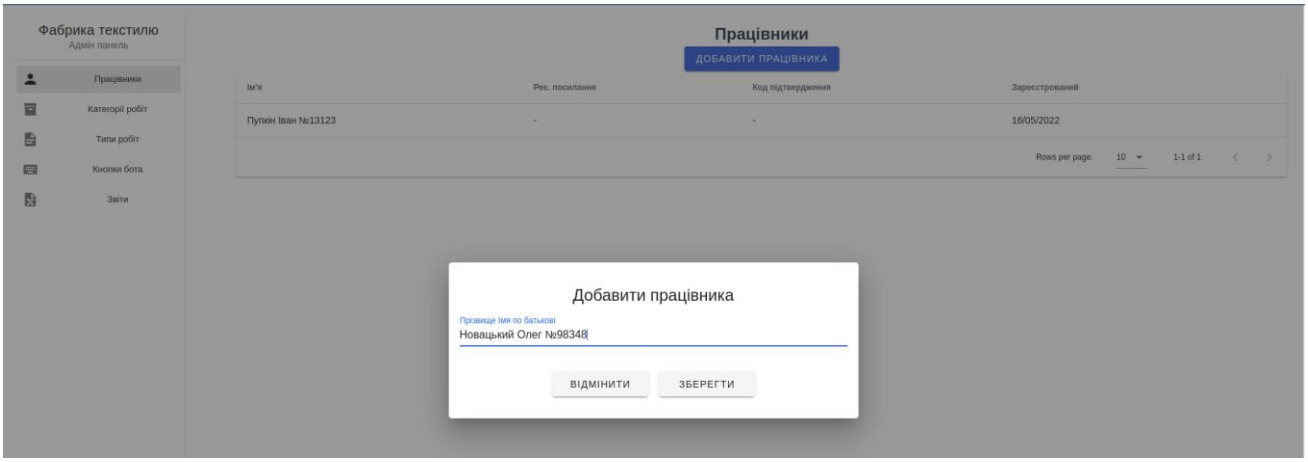


Рисунок 3.3.2 - Форма для створення працівника

Після збереження форми, запис з'являється в загальному списку працівників. Для цього працівника будуть відображатися таємне посилання для реєстрації та код підтвердження реєстрації. Ці дані вже недоступні для працівників, які виконали реєстрацію. Для них відображається лише дата коли вони завершили реєстрацію. Сторінка із списком, після збереження форми, зображена на рисунку 3.3.3.

Для налаштування компонентів категорії типів робіт, необхідно перейти на сторінку типів робіт. Цю сторінку зображено на рисунку 3.3.4. Добавлення категорії можна виконати на формі, які можна відкрити за допомогою натискання на кнопку "Добавити категорію". Форму добавлення зображено на рисунку 3.3.5. На формі потрібно увести назву категорії і опис, який буде відображатися для користувача бота після вибору категорії робіт.

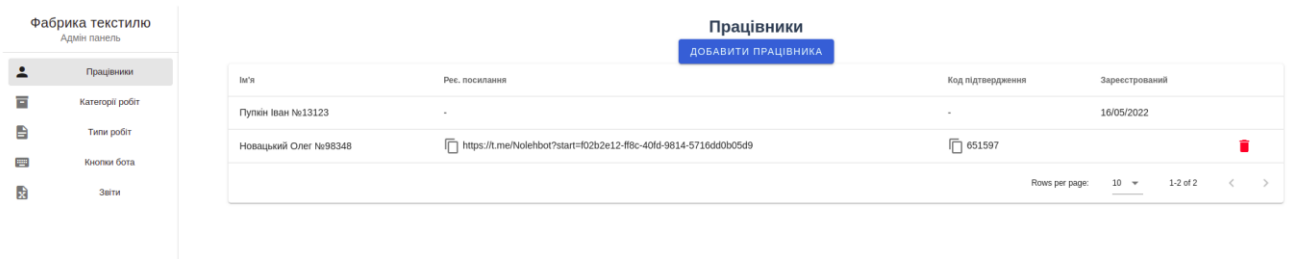


Рисунок 3.3.3 - Список працівників після добавлення нового запису

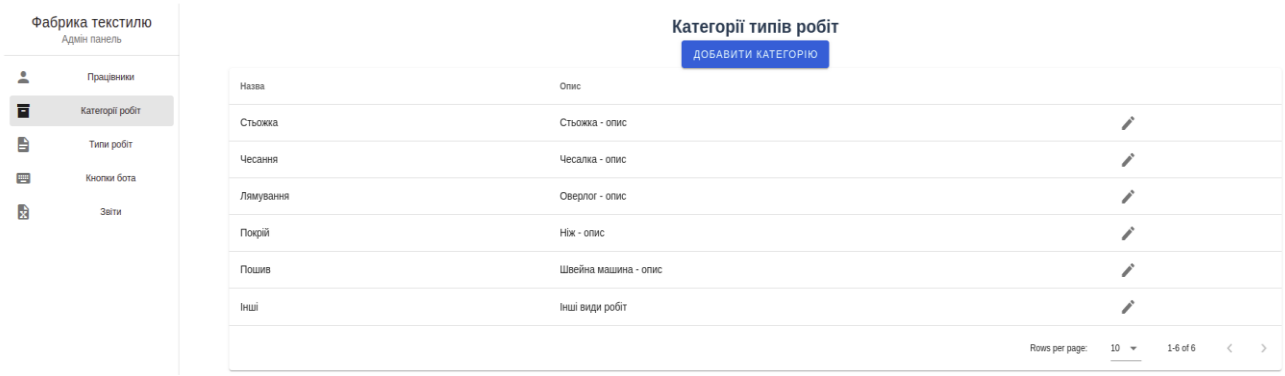


Рисунок 3.3.4 - Сторінка категорій типів робіт

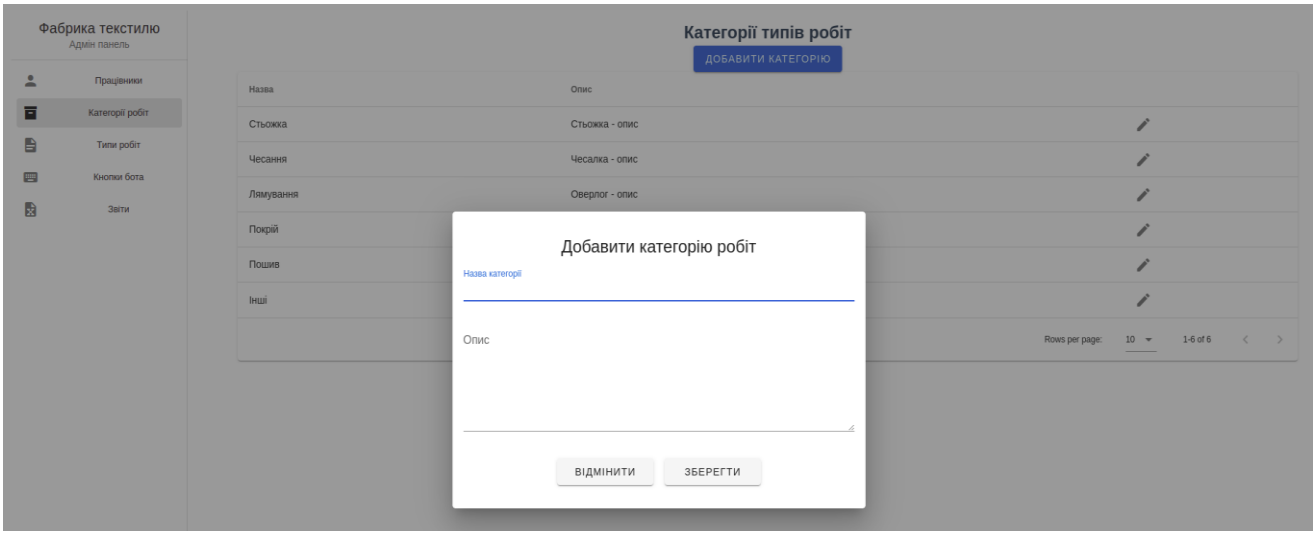


Рисунок 3.3.5 - Форма добавлення категорії

Сторінка із типами робіт зображено на рисунку 3.3.6. Там відображається таблиця із записами, де кожен запис містить назву типу роботи, інформацію про категорію до якої відноситься тип роботи, одиниця вимірювання, що застосовується для цього типу, та ціна за одиницю виконаної роботи. Форма

створення типу роботи містить всі поля, відповідно до попередньо перелічених колонок таблиці. Форму зображено на рисунку 3.3.7.

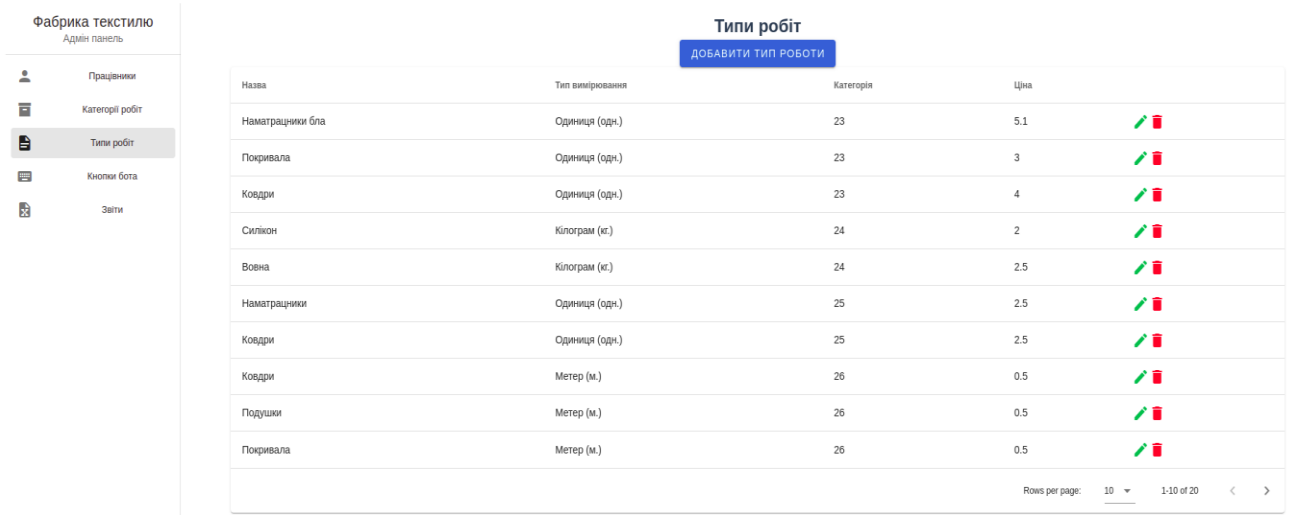


Рисунок 3.3.6 - Сторінка із типами робіт

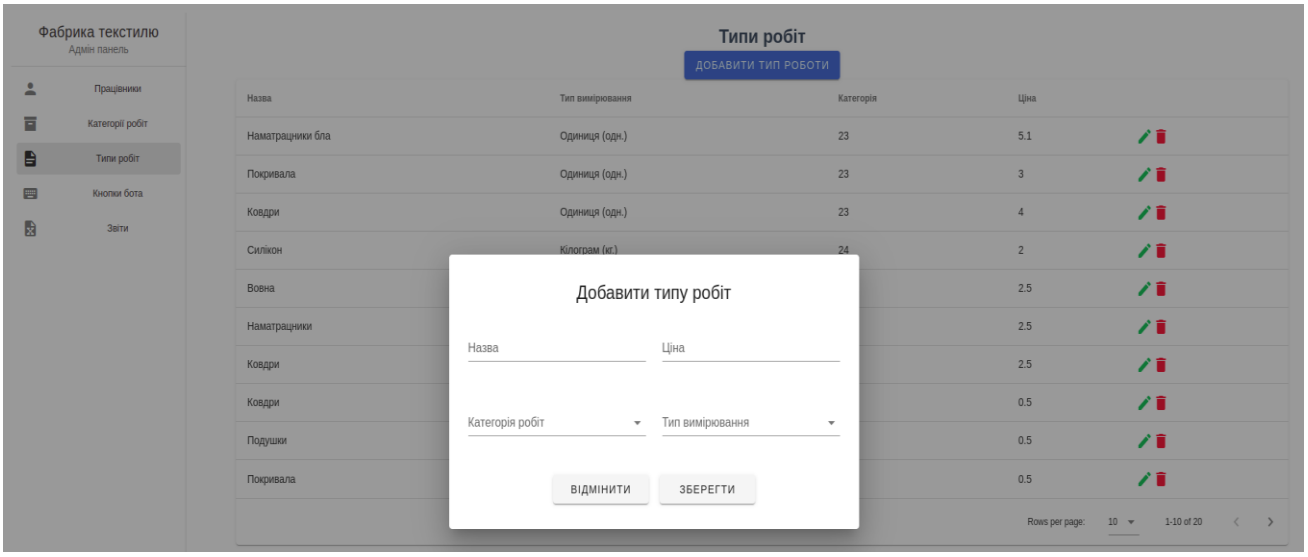


Рисунок 3.3.7 - Форма створення нового типу роботи

Коли вже готові всі компоненти структури робіт, то необхідно для них створити кнопки. Текст кожної кнопки буде відображатися на клавіатурі бота для вибору роботи. В залежності від того для якого компонента кнопки створені, то вони будуть зображені в різних клавіатурах. Це вже залежить від структури самих кнопок і категорій. Добавлення, видалення та редагування кнопки виконується на одній сторінці. Сторінку зображено на рисунку 3.3.8.

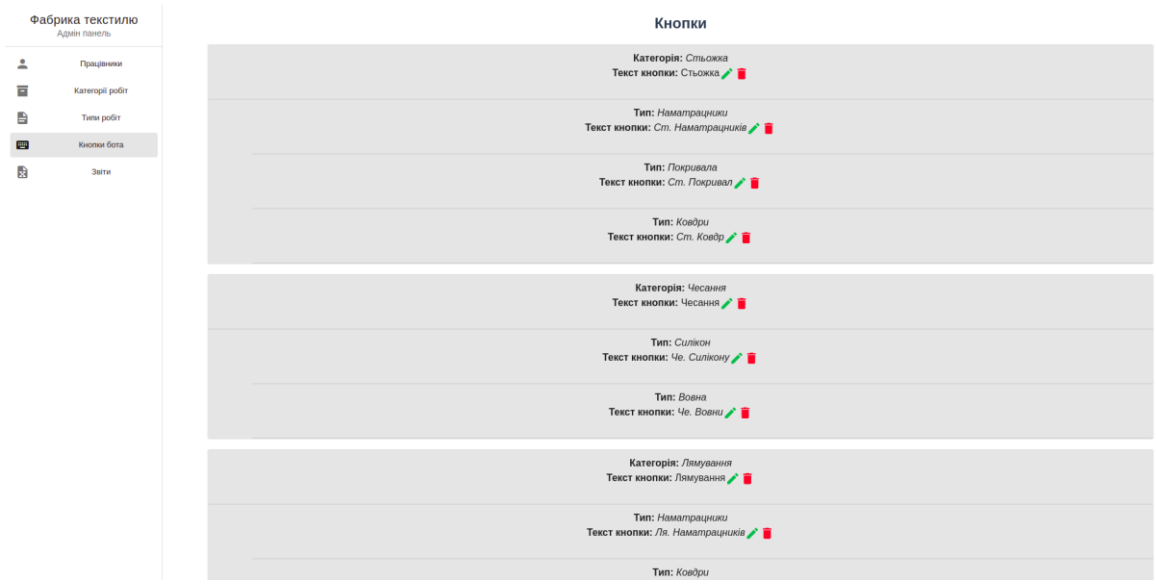


Рисунок 3.3.8 - Сторінка для роботи із кнопками бота

На цій сторінці відображаються всі типи робіт, згруповані по категоріях. Якщо для якогось компонента видалити кнопку, то вона не з’явиться в клавіатурі і не зможе бути розпізнана системою. Тож якщо є необхідність приховати і деактивувати якусь категорію робіт, то для цього потрібно видалити кнопку на цій сторінці для відповідної категорії. Також для кожної кнопки є можливість змінити текст. Це можна виконати, натиснувши на іконку олівці біля відповідного тексту. З’явиться форма для введення тексту. Та якщо кнопка відсутня, то замість тексту біля пункту “Текст кнопки”, буде відображатися знак питання. Він є ідентифікатором відсутності кнопки для того чи іншого компонента структури кнопок. Форму редагування чи створення, вони однакові, кнопок зображено на рисунку 3.3.9.

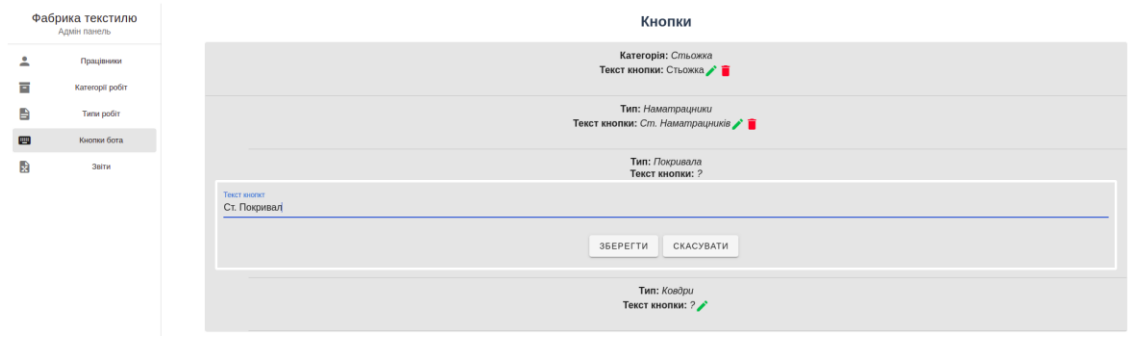


Рисунок 3.3.9 - Форма створення і редагування тексту кнопок

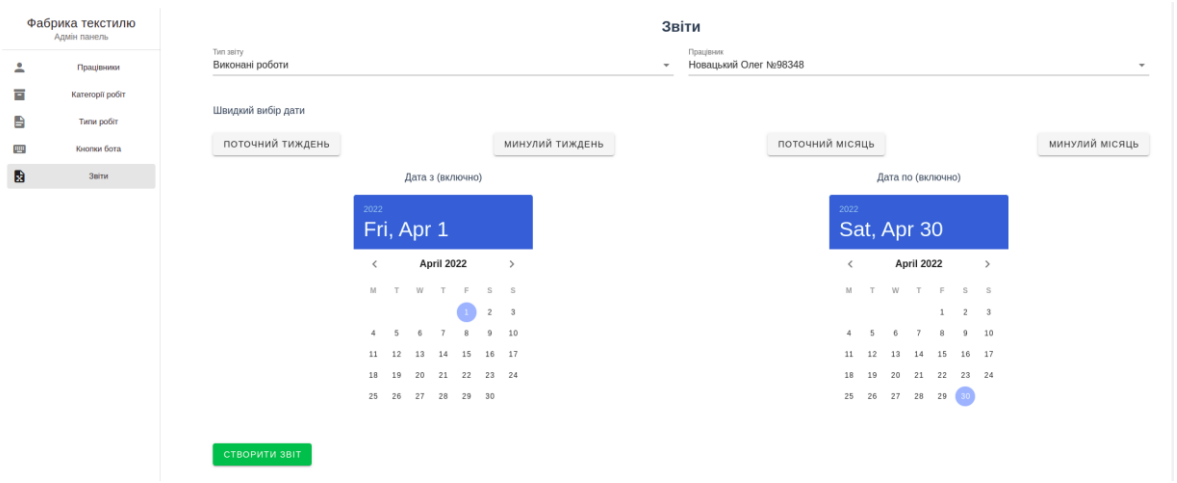


Рисунок 3.3.10 - Сторінка генерації звітів

На цій сторінці є поля для вибору параметрів формування звіту. Обов'язковими параметрами є дати і вибір типу звіту. Типів звіту може бути тільки два: зарплати, виконані роботи. Приклад звіту для зарплат зображено на рисунку 3.3.11, а для виконаних робіт на рисунку 3.3.12. Також є можливість генерувати звіти для окремого працівника, для цього необхідно обрати працівника в списку доступних перед генерацією.

		Чесання		Лямування	
		Силікон		Ковдри	
	Загалом зароблено	Виконано	Зароблено	Виконано	Зароблено
Новацький Олег №98348	45	10	20	10	25
Загальна сума	45				

Рисунок 3.3.11 - Приклад звіту по зарплатах

					Пятниця	Субота	Неділя	Понеділок	Вівторок	Середа	Чете
Тип роботи	Категорія робіт	Одиниця вимірювання	Загалом виконано за період	До сплати за період	01.04.2022	02.04.2022	03.04.2022	04.04.2022	05.04.2022	06.04.2022	07.0
Наматрацники	Лямування	Одиниця (одн.)	678	1695	0	0	40	0	0	94	
Ковдри	Лямування	Одиниця (одн.)	613	1532.5	64	0	52	0	0	0	
Погрузка	Інші	Година (год.)	666	33300	0	0	14	0	50	0	
Ковдри	Пошив	Одиниця (одн.)	793	1982.5	0	97	0	0	77	0	
Подушки	Пошив	Одиниця (одн.)	811	2433	0	0	80	56	0	0	
Постелі	Пошив	Одиниця (одн.)	1114	5570	155	68	149	0	83	142	
Покривала	Стьожка	Одиниця (одн.)	740	2220	0	143	0	3	0	0	
Ремонт	Інші	Година (год.)	668	26720	12	0	82	0	32	46	
Розгрузка	Інші	Година (год.)	730	29200	62	0	0	99	18	61	
Напірники	Пошив	Одиниця (одн.)	757	757	0	0	0	88	0	45	
Постелі	Покрій	Метер (м.)	970	727.5	0	24	100	15	0	0	
Покривала	Покрій	Метер (м.)	877	438.5	0	183	0	91	0	0	
Ковдри	Покрій	Метер (м.)	902	451	0	0	0	0	11	0	
Подушки	Покрій	Метер (м.)	1121	560.5	88	141	0	0	161	0	
Ковдри	Стьожка	Одиниця (одн.)	814	3256	76	0	0	0	0	134	
Наматрацники	Стьожка	Одиниця (одн.)	400	2000	0	0	0	108	0	0	
Вовна	Чесання	Кілограм (кг.)	655	1637.5	0	0	44	0	58	0	
Силікон	Чесання	Кілограм (кг.)	1520	3040	0	0	95	127	33	83	
Поїздка	Інші	Година (год.)	418	20900	34	0	0	0	0	0	
			Загалом:	138421							

Рисунок 3.3.12 - Приклад звіту по виконаних роботах

Завершили із керівництвом для веб-інтерфейсу. Тепер можна описати як користуватися усіма створеними компонентами в боті. Для того щоб почати користуватися ботом, необхідно отримати своє унікальне посилання для реєстрації в керівника. Також іншим методом комунікації отримати код підтвердження реєстрації. Коли користувач перейде за посиланням, йому буде запропоновано відправити повідомлення боту у веб-версії телеграму, або буде запит на відкриття чату із ботом в додатку телеграма на телефоні чи ПК, якщо такий додаток встановлено. Як тільки відкриється відкривається чат із ботом необхідно натиснути на кнопку із текстом “Start”. Якщо система розпізнає користувача то вона попросить увести код підтвердження реєстрації. Якщо уведений код підтвердження не є вірним, то бот повідомляє про це користувача. Ці перші кроки реєстрації користувача зображено на рисунку 3.3.13, 3.3.14 та 3.3.15.

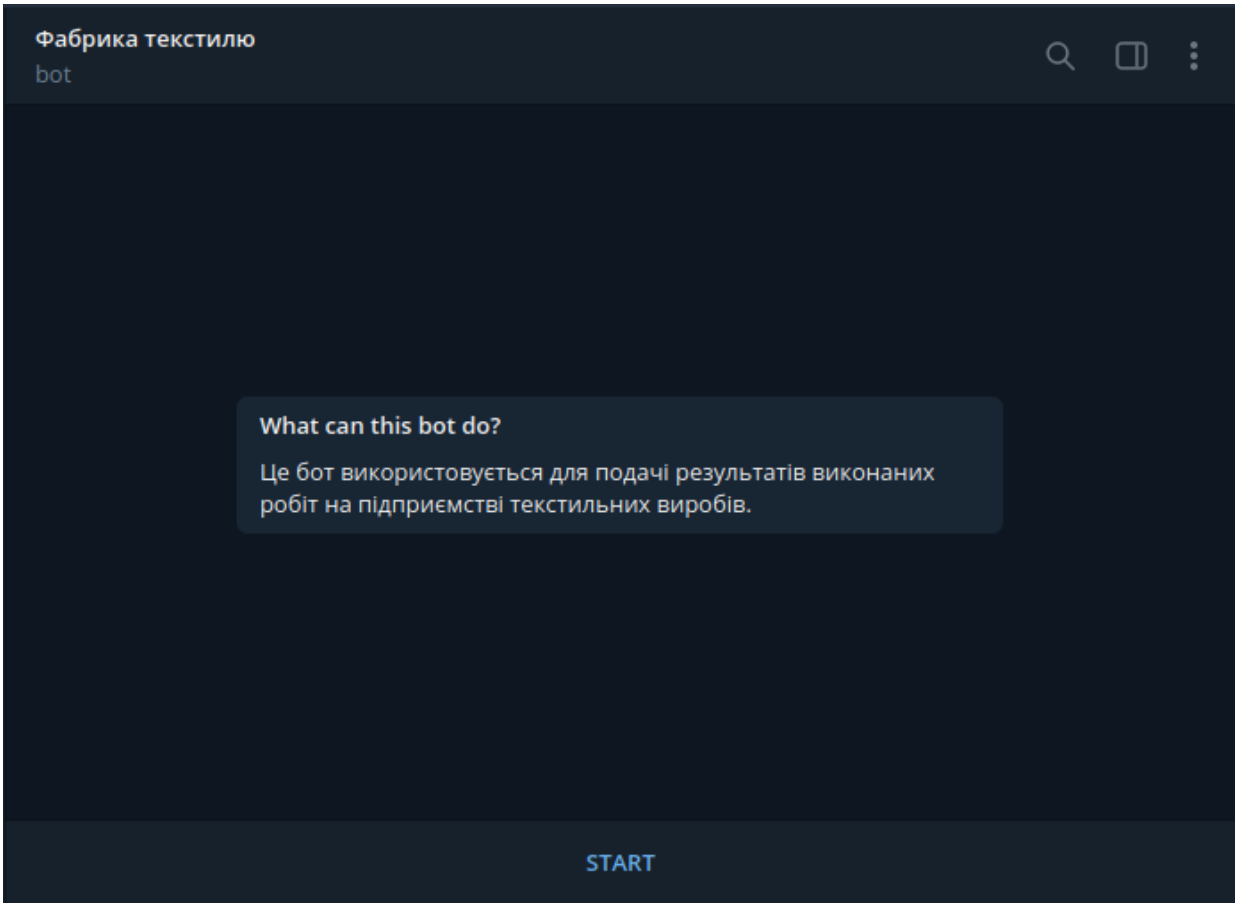


Рисунок 3.3.13 - Початковий інтерфейс бота після переходу за посиланням

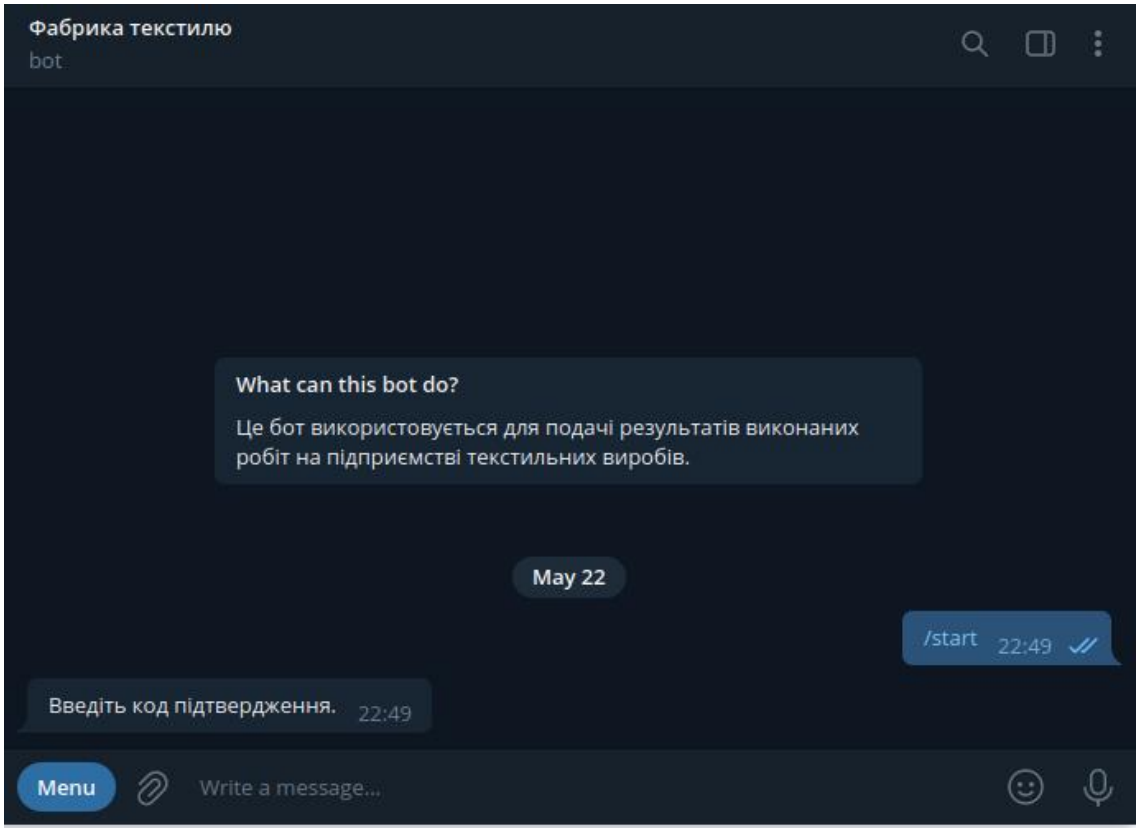


Рисунок 3.3.14 - Запит бота на введеня коду підтвердження

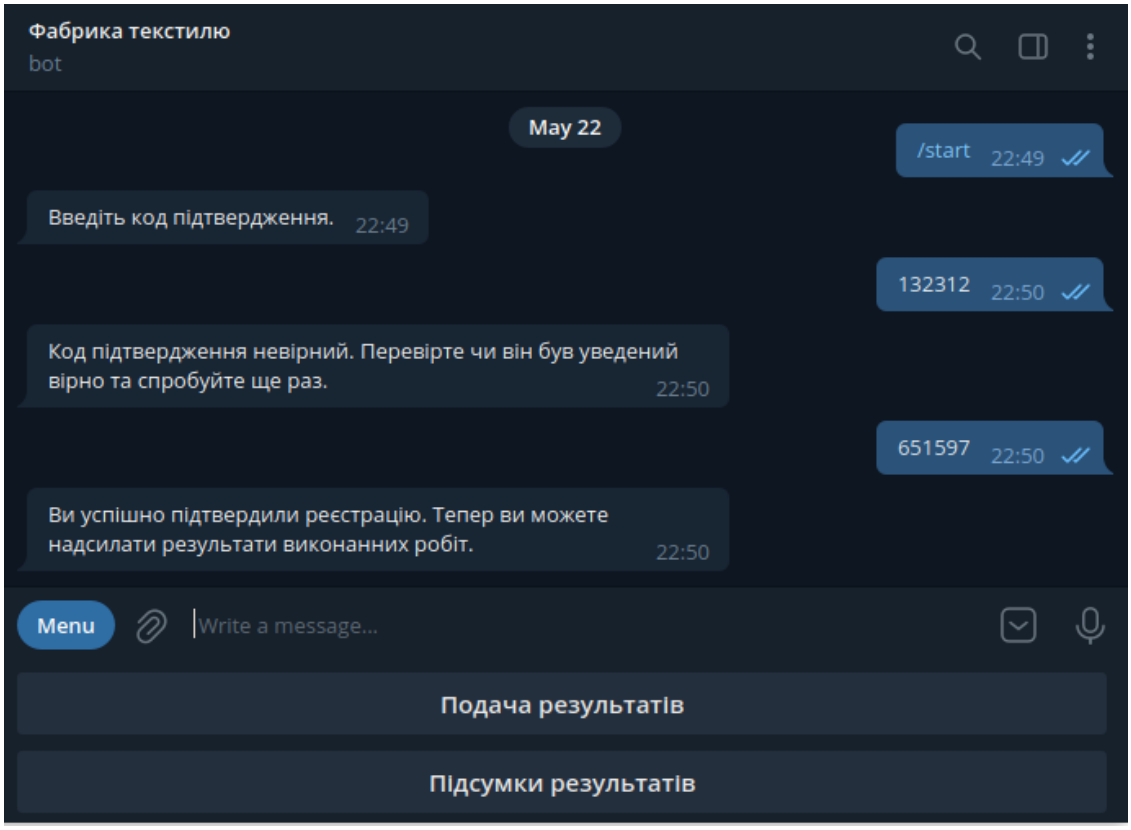


Рисунок 3.3.15 - Виконання перевірки коду підтвердження реєстрації

Після завершення реєстрації, бот знаходиться у звичайному робочому стані. Тепер користувач у змозі передавати результати виконаних робіт. Для цього йому необхідно натиснути кнопку “Подача результатів”. Тоді для користувача відобразиться інша клавіатура, яка відображатиме кнопки, що представляють категорії робіт. Користувачеві необхідно обрати категорію, щоб із наступної клавіатури можна було вже обрати тип роботи який потрібно звітувати. Після вибору типу роботи, бот відповість користувачеві повідомленням із форматом, якого потрібно дотримуватися під час введення результатів. У користувача також є можливість вводити від’ємні значення результатів виконаних робіт із метою нормалізації результатів, якщо користувач попередньо відправив некоректне значення. Після збереження результатів, користувач повертається до клавіатури вибору категорії. На інтерфейсі вибору категорії робіт, бот відобразить користувачеві усі результати за звітну дату. Увесь цей процес зображено на рисунку 3.3.16.

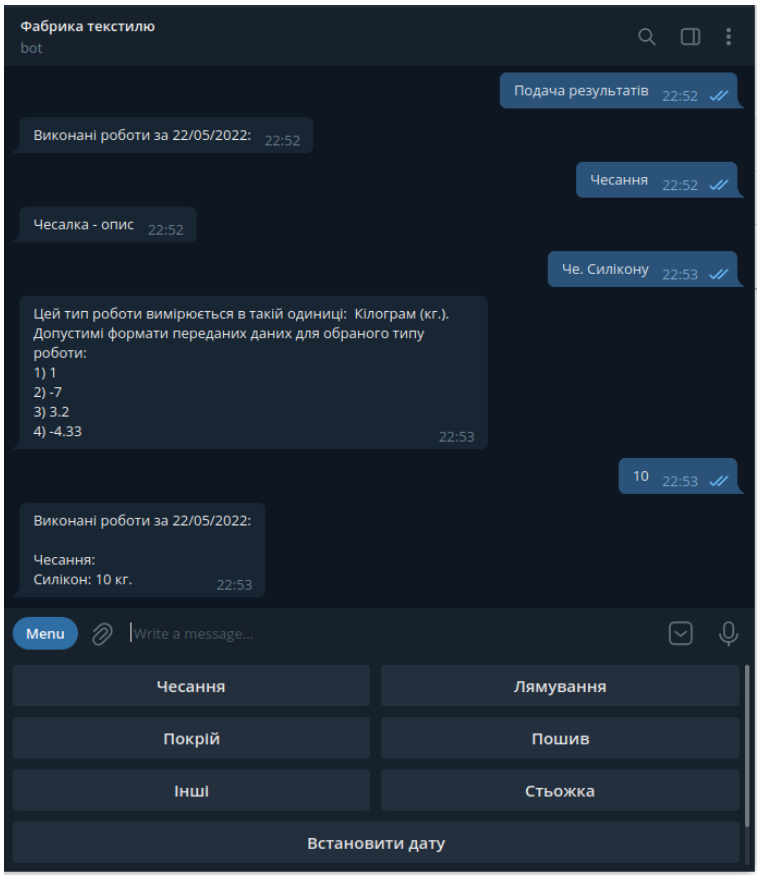


Рисунок 3.3.16 - Процес введення результатів

Також у користувача є можливість встановлювати дату для якої виконується звітність. Це можна зробити натиснувши на кнопку “Встановити дату”. Після цього бот попросить користувача увести необхідну дату в певному форматі. Якщо користувач уведе дату некоректно, то система не зможу її опрацювати. Тому перед встановленням звітної дати виконується перевірка формату і коректність дати. Якщо є помилки то бот відправить користувачеві повідомлення про це, та ще раз нагадає формат уведення дати. Після зміни дати, процес введення результатів такий самий як і попередньо описаний. Процес зміни звітної дати показано на рисунку 3.3.17.

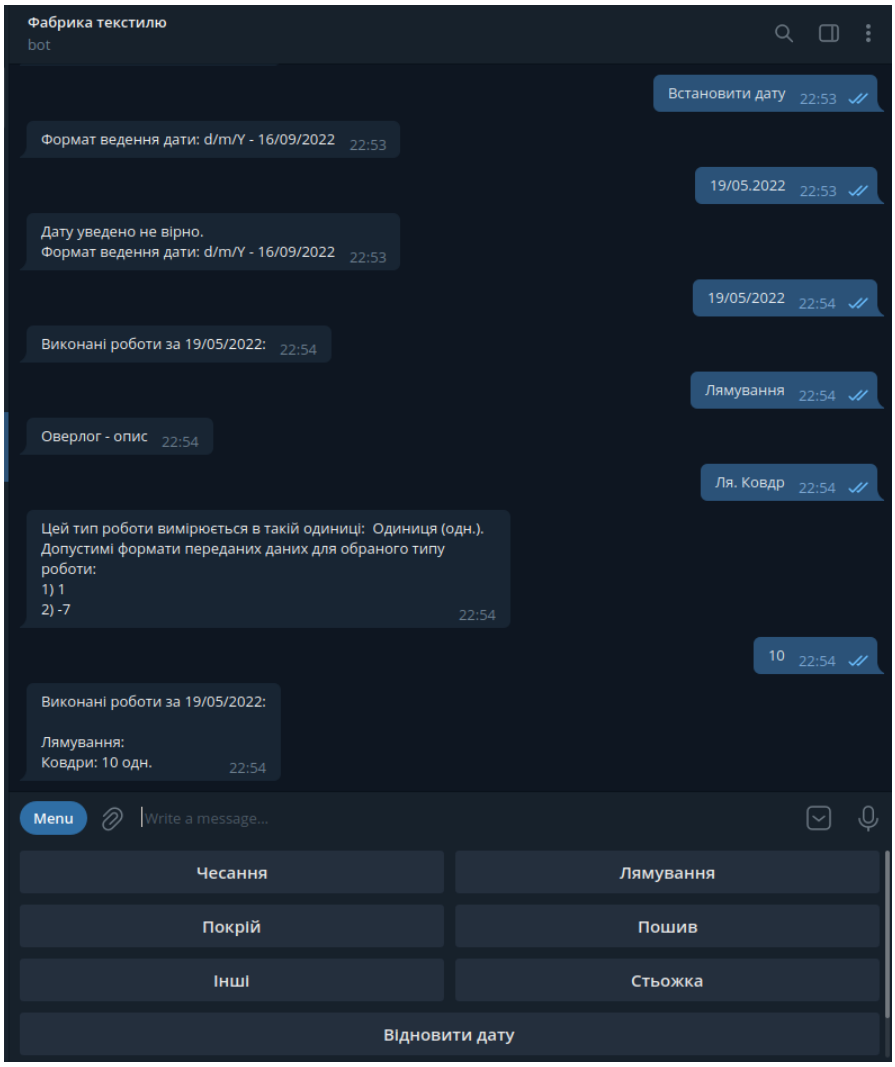


Рисунок 3.3.17 - Процес зміни звітної дати

Після введення даних за необхідну дату користувач має змогу відновити поточну дату для звітності. Для цього необхідно натиснути на кнопку

“Відновити дату”, що знаходиться на клавіатурі вибору категорій. Цей процес зображено на рисунку 3.3.18.

У працівника також є можливість отримати результати виконаних робіт за певний період. Для цього користувачеві необхідно натиснути кнопку “Підсумки результатів” на головному меню. В користувача з’явиться клавіатура із вибором звітного періоду. Після вибору періоду, бот відповість користувачеві із результатами. Цей процес зображено на рисунку 3.3.19.

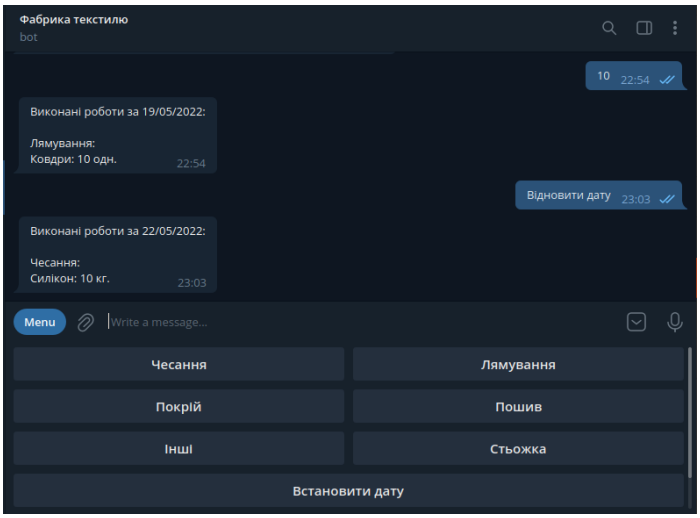


Рисунок 3.3.18 - Відновлення звітної дати

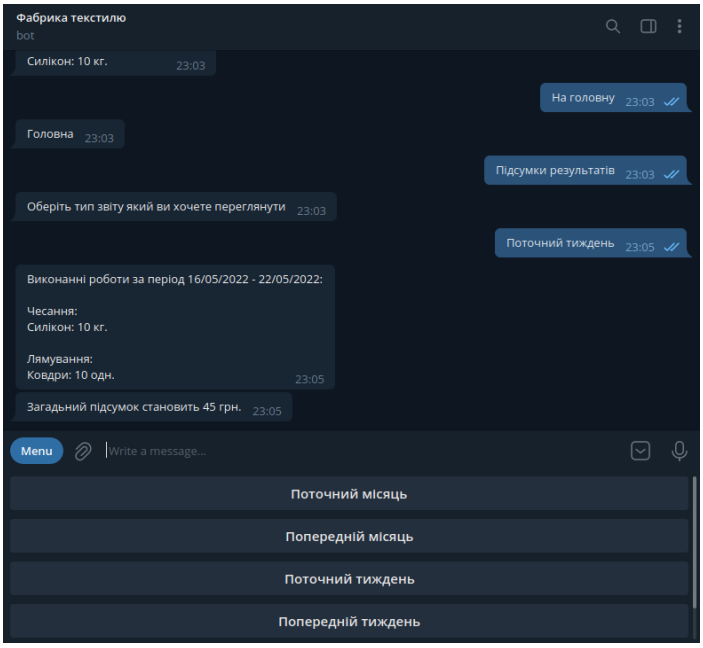


Рисунок 3.3.19 - Процес отримання результатів за певний період

В цьому підрозділі було описано керівництва для користувачів системи, а саме, як користуватися веб-інтерфейсом для налаштування компонентів системи та як користуватися розробленими можливостями бота.

3.4 Вимоги до технічних та програмних засобів

Сьогодні є багато способів запуску проект, що розроблений на мові програмування PHP. Для цього можуть використовуватися веб-хостинги, окремі виділені сервери. Всі ці сутності в поєднанні із іншими сервісами, що також можуть бути розміщені на сторонніх ресурсах дозволяються запускати проекти різних складностей, виконувати масштабування та інші дії для оптимізації. Тому щоб визначитися із методом запуску проекту, необхідно проводити аналіз вимог, які поставлені перед проектом, навантаження, яке проект буде опрацьовувати, та визначитися на скільки складним може бути «деплой».

Для роботи системи потрібно, щоб на сервері було встановлено PHP, який буде доступний для веб-сервера, це дозволить опрацьовувати вхідні запити. Та повинен бути доступний пакетний менеджер Composer для завантаження сторонніх пакетів, які використовуються в ПЗ. Також потрібно щоб були доступні база даних PostgreSQL та брокер повідомлень RabbitMQ. Процес встановлення і налаштування всіх цих компонентів системи займає досить багато часу і вимагає детальних знань у структурі і роботі сервера, в якому все це буде налаштовано. Щоб уникнути усіх подібних труднощів встановлення і налаштування, сьогодні використовують сервіс контейнеризації Docker. Він дає можливість своїм користувачам дуже швидко та легко встановлювати всі необхідні сервіси, без зайвих витрат часу, який коштує грошей. Всі створенні сервіси можуть бути в одній локальній мережі, що дозволить не витрачати ресурси на організацію взаємодії сервісів між собою, та приховає сервіси від зовнішніх мереж, що можуть бути небезпечними.

Тож для встановлення проекту необхідно будь-який сервер на якому можна встановити Docker. Також необхідно, щоб з’єднання, яке буде

встановлюватися між сервером і Telegram, було захищене SSL сертифікатом, тому що телеграм не працює із не захищеними веб-хуками. Тому вимоги до сервера, де буде запускатися веб-сервер системи, можна зіставити із вимогами для встановлення Docker, який забезпечить проект усіма необхідними сервісами і достатніми ресурсами для їх нормальної роботи. Ось перелік вимог до апаратури необхідний для запуску веб-сервера за допомогою Docker:

– дисковий простір > 30 гігабайт;

– оперативна пам'ять > 8 гігабайт.

Також необхідно встановити Docker-Compose і Git. Docker-Compose забезпечує користувача можливістю налаштувати всі сервіси за допомогою одного конфігураційного файлу. Git необхідний для клонування проекту із репозиторію і подальшому завантаженню оновлень. Завантажений проект містить файл «docker-compose.yml», що містить налаштування для всіх необхідних сервісів.

Для запуску веб-інтерфейсу достатньо будь-якого хостингу, де можна розмістити статичні файли. На цьому хостингу будуть розміщені файли HTML, CSS та JS уже готового SPA. Для того щоб отримати ці файли необхідно спочатку добавити URL адресу, за якою доступний сервер із Rest API, у файлі налаштувань .env.local. Після цього необхідно виконати будування проекту і файли з’являться у папці dist/.

Виконавши всі дії, що описані в цього розділі, ви отримаєте готове програмне забезпечення, яке виконуватиме всі поставлені задачі.

4 Тестування програмного забезпечення

4.1 Вибір та обґрунтування методів тестування

Програмні продукти в силу різних причин мають модульну структуру і рівневу організацію, і, як правило, функціонують не відокремлено, а в рамках деякої інфраструктури (спільно з іншими додатками, програмними комплексами, а також в рамках програмних середовищ). Це призводить до виникнення різних рівнів тестування:

– компонентне;

– інтеграційне;

– системне;

– приймальне.

Компонентне або модульне тестування перевіряє функціональність і шукає дефекти в частинах програми, які доступні і можуть бути протестовані окремо. Ознакою, що утворює даний рівень, є те, що тестування окремих модулів можна виконувати на ранніх етапах розробки, поки робота над іншими модулями триває. Один з найбільш ефективних підходів до компонентного (модульного) тестування – це підготовка автоматизованих тестів до початку основного кодування ПЗ. Це називається розробкою від тестування або підходом тестування спочатку.

Інтеграційне тестування призначено для перевірки зв’язків між компонентами програми, а також дослідження взаємодії додатку з середовищем, в рамках якого воно буде виконуватися.

Системне тестування направлено на дослідження функціональних і нефункціональних особливостей системи в цілому. При цьому виявляються дефекти, такі як невірне використання ресурсів системи, непередбачені комбінації даних рівня користувача, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання, тощо.

Приймальне тестування проводиться на фінальному етапі розробки та спрямовано на з’ясування того, чи відповідає система критеріям до вимог замовника. Рішення про проведення приймального тестування приймається, коли: продукт досяг необхідного рівня якості та замовник ознайомлений з планом приймальних робіт або іншим документом, де описаний набір дій, пов’язаних з проведенням приймального тестування, дата проведення, відповідальні тощо. Методи приймального тестування:

– тестування замовником самостійно;
– тестування третьою стороною (аудит);
– спільне тестування за сценаріями із замовником.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або реліз програми. Незважаючи на те, що приймання знаходиться в кінці етапу (а в невеликих проектах і в кінці проекту) – готуватися до нього потрібно заздалегідь і перший прогін потрібно робити трохи раніше.

Залежно від цілей тестування, виділяють три основних види тестування програмного забезпечення:

– функціональні;
– нефункціональні;
– пов'язані із змінами.

Функціональні види тестування пов’язані з дослідженням зовнішньої поведінки системи, тобто виконуваних нею функцій. До них належать:

– функціональне тестування, що спрямоване на перевірку коректності виконуваних системою функцій і може бути присутнім на всіх рівнях тестування;

– тестування безпеки, яке направлене на перевірку безпеки системи, а також оцінку цілісності підходу до захисту додатку від несанкціонованого доступу і захисту конфіденційних даних;

– тестування взаємодії, яке направлене на оцінку можливості додатка взаємодіяти із зовнішніми компонентами або системами, а також включає тестування сумісності та інтеграційне тестування.

Нефункціональні види тестування, які спрямовані на перевірку всіх нефункціональних особливостей системи. Сюди належать тести специфічних для програмних продуктів характеристик:

– тестування встановлення, яке направлене на перевірку процесу інсталяції системи, а також процесів настойки, вилучення і оновлення програмного забезпечення;

– тестування зручності використання, яке пов’язане з оцінкою ступенем зручності використання, а також зрозумілості та привабливості користувальницького інтерфейсу додатку;

– тестування на відмову і відновлення, яке пов’язане з оцінкою засобів забезпечення відмово-стійкості та надійності системи;

– конфігураційне тестування, яке спрямоване на перевірку функціональності системи при всіх можливих конфігураціях програмного забезпечення і устаткування, що підтримується системою;

– тестування продуктивності , яке складається з таких методів тестування як: тестування навантаження, стресове тестування, стабільності і надійності, об’ємне тестування.

Види тестування, які пов’язані із змінами:

– димове тестування;

– регресивне тестування;

– тестування збірки;

– санітарне тестування;

– альфа-тестування;

– бета-тестування.

Димове тестування, яке спрямоване на оглядову перевірку всіх компонентів програми на предмет працездатності, а також на виявлення грубих

дефектів, наявність яких можна визначити дуже просто. За результатами димового тестування робиться висновок про те, приймається чи ні встановлена версія програмного забезпечення на тестування, експлуатацію або на постачання замовнику. Димові тести повинні виконуватися на всьому проекті від початку до кінця. Вони не повинні бути вичерпними і всебічними, але повинні містити перевірку всіх основних функцій

Регресивне тестування – збірна назва для всіх видів тестування програмного забезпечення, спрямованих на виявлення помилок у вже протестованих ділянках вихідного коду. Регресивне тестування в основному призначено для перевірки здійснених в системі змін, а також на підтвердження того, що функціональність, яка існувала до зміни, працює так же як і до змін. Регресивне тестування є невід'ємною частиною екстремального програмування. У цій методології проектна документація замінюється на розширюване, повторюване та автоматизоване тестування всього програмного пакету на кожній стадії процесу розробки програмного забезпечення.

Тестування збірки, яке направлене на перевірку відповідності версії програмного продукту, що випускається, критеріям якості, необхідним для початку тестування. За своєю метою є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію.

Санітарне тестування чи інакше – перевірка узгодженості/справності полягає в проведенні тесту достатнього для підтвердження того, що певна окремо взята функція працює відповідно до заявлених специфікацій.

Альфа-тестування – імітація реальної роботи з системою штатних розробників, або реальна робота з системою потенційних користувачів. Найчастіше альфа-тестування проводиться на ранній стадії розробки продукту, але в деяких випадках може застосовуватися для закінченого продукту в якості внутрішнього приймального тестування. Іноді воно виконується з використанням оточення, яке допомагає швидко виявляти знайдені помилки.

Бета-тестування – у деяких випадках виконується поширення попередньою для деякої більшої групи осіб з тим, щоб переконатися, що продукт містить досить мало помилок. Іноді бета-тестування виконується для того, щоб отримати зворотній зв'язок про продукт від його майбутніх користувачів.

Для тестування розроблюваного програмного забезпечення буде використовуватися інтеграційне тестування, регресійним видам для визначення можливих дефектів створеної системи. Всі перевірки будуть виконуватися на основі тестових випадків самостійно, без залучення програмних рішень, що застосовуються в автономному тестуванні. Таке тестування буде ще називатися тестуванням «білого ящика», тому як тестувальнику відомий програмний код системи.

4.2 Доведення працездатності програмного продукту

Перед виконанням тестування необхідно створювати тест-кейси, щоб під час самого тестування не упустити якусь функцію системи. Для деяких тест кейсів також потрібні певні набори даних, створення і генерація яких займе багато часу, тому у таких випадках застосовуються генератори даних. Ці генератори даних називають фікстурами. Тож для створення категорій робіт, типів робіт, кнопок бота та великої кількості результатів робіт було створено фікстури які згенерували ці дані.

Розроблена система має два розділених інтерфейси взаємодії із своїми користувачами. Ці користувачі мають різні ролі, тому є доцільним виконувати тестування цих двох інтерфейсів окремо. Опис процесу тестування також необхідно виконувати роздільно.

Тестування інтерфейсу слід виконувати в логічній послідовності, від меншого і до більшого, або кажучи правильно, від початку і до кінця. Тож першим кроком у використанні бота є реєстрація користувача в системі. Тут необхідно виконати всі кроки для завершення цього процесу, але перед

виконанням дій що приведуть до успішного завершення, необхідно виконувати дії які можуть спричинити помилки в роботі програми із метою перевірки певних валідацій дій користувача. Цей принцип перевірки введення некоректних даних для бота, потрібно виконувати під час перевірки всіх кроків.

Тож після завершення перевірки реєстрації необхідно виконати перевірку можливості введення результатів. До моменту самого введення користувачеві ще потрібно обрати тип виконаної роботи. Для цього йому необхідно виконати дві дії, які можуть супроводжуватися некоректним веденням даних, тож не потрібно забувати це перевіряти. Також на клавіатурах які відображатимуться до моменту вводу будуть присутні навігаційні кнопки, їх функціональність також необхідно перевірити. При відправці результату виконаної роботи боту, можна отримати повідомлення про невірний формат введення. Весь цей процес введення результату потрібно протестувати із різними переходами між клавіатурами за допомогою навігаційних кнопок. Після введення результатів користувач також може побачити підсумок своїх робіт у повідомлені від бота, які він надсилає що разу при будь якій дії із ним.

Наступник кроком тестування бота є введення результатів на певну дату, тобто не ту яка є на момент введення результатів. Це виконується в режимі введення результатів, для цього є окрема кнопка, яка запускає процес зміни дати. Після зміни дати необхідно виконати всі дії, які було описано вище, при звичайному введені результатів.

Всі добавлені результати можна побачити у звітах. Бот забезпечує користувачів можливістю отримувати результати виконаних робіт. Це виконується в підсистему по генерації звітів по виконаних роботах. Там є доступні кнопки, що представляють різні періоди для отримання звітів. Результати цієї генерації потрібно перевірити для всіх доступних періодів. Також необхідно брати до уваги, що можна вплинути на зміну даних в цьому звіті за допомогою добавлення нових результатів. Комбінація всіх цих дій дозволить виконати комплексне тестування всіх доступних функцій бота.

Певні приклади головних тест-кейсів показано в таблиці 4.2.1. Так як Telegram можна використовувати на смартфонах, персональних комп'ютерах і веб-версії, то всі тестові випадки потрібно повторити в кожному із доступних місць де цей бот може бути використаний, щоб зменшити ризики некоректної поведінки системи.

Таблиця 4.2.1 - Тест-кейси для тестування бота в Telegram

№	Інтерфейс системи	Підсистема	Кроки виконання тест-кейсу	Очікуванні результати по кожному кроці тест-кейсу
1b	Бот	Реєстрація	Підготовка: отримати унікальне посилання та код підтвердження для реєстрації. 1) Перехід за отриманим посиланням; 2) Введення невірного коду підтвердження; 3) Введення вірного коду підтвердження.	1) Відкривається чат із ботом, є кнопка START; 2) З'являється повідомлення про некоректний код; 3) З'являється головна клавіатура і бот готовий для своєї роботи..
2b	Бот	Введення результатів робіт	Підготовка: користувача зареєстрований. 1) Натиснути кнопку «Подача результатів»; 2) Надіслати боту слово «Привіт»; 3) Натиснути на кнопку із назвою необхідної категорії; 4) Натиснути на кнопку із потрібним типом роботи; 5) Ввести будь-які букви і цифри; 6) Ввести коректне значення.	1) Відображення клавіатури із назвами категорій робіт; 2) Бот відповідає, що невірна назва категорії; 3) Відображається клавіатури із переліком типів робіт; 4) Відображається одна кнопка «На головну» і бот просить ввести результати робіт у форматі; 5) Бот поверне повідомлення що результати некоректні; 6) Користувач потрапляє до меню із категоріями та бот відображає результати за звітну дату.

Веб-інтерфейс розроблено для керівника підприємства, тож масового доступу до цього ресурсу не буде, тому в адмін. Панелі відсутній процес

авторизації. Всі необхідні засоби захисту від стороннього доступу виконуватимуться засобами веб-сервера Nginx. Тому при відкриті сторінки веб-інтерфейсу, можна розпочинати тестування цієї частини системи.

Для забезпечення працівників можливістю реєструватися в боті, потрібно добавити їх в систему. Це робиться на сторінці налаштування працівників. Там відображається список працівників, їхні дані та кнопки для налаштування цих записів. Кнопка видалення працівника функціонує лише у випадку якщо користувач для якого цей запис був створений, ще не зареєструвався в системі. Для добавлення користувача потрібно натиснути синю кнопку із відповідним текстом, після цього з'являється форма для ведення даних про працівника. На цій формі потрібно перевірити валідацію введених даних. Після відправки форми, новий запис повинен з'явитися в таблиці. Весь цей процес є досить прозорим і простим, тому як використовуються інтуїтивні компоненти дизайну, всі кнопки містять відповідний текст до їхніх дій, тому перевірку всіх інших сторінок налаштування системи потрібно виконувати за таким самим принципом.

Тестування головної сторінки адмін. панелі виконується за наявності в базі даних достатньої кількості результатів виконаних робіт. Це дозволить наглядним способом показати роботу генерації звітів. Для генерації звітів необхідно вказати необхідний тип звіту на формі, це поле є обов'язковим для введення. Також на формі присутні поля для вибору дати, які будуть використовуватися під час отримання даних. Ця функція забезпечить керівника можливістю генерувати звіти за різні періоди часу. Так як кожне поле дозволяє обрати будь яку дату, то необхідно виконати тестування різних значень цих полів, оскільки «Дату з» можна встановити більшою а ніж «Дата по». Ця невідповідність між датами може спричинити проблеми під час генерування звіту, ому важливо перевірити валідації із цими даними. Також на формі присутній фільтр по працівниках. Вибірку даних можна виконати для конкретного працівника. Цим можна скористатися, щоб перевірити система

правильно виконує обрахування результатів. Деякі приклади тест-кейсів для веб-інтерфейсу показано в таблиці 4.2.2

Таблиця 4.2.2 - Тест-кейси для тестування веб-інтерфейсу

№	Інтерфейс системи	Підсистема	Кроки виконання тест-кейсу	Очікуванні результати по кожному кроці тест-кейсу
1w	Веб-інтерфейс	Типи робіт	1)Натиснути кнопку навігації «Типи робіт»; 2)Натиснути кнопку «Добавити тип роботи»; 3)Відправити форму на збереження; 4)Заповнити поля форми; 5)Відправити форму на збереження; 6)Натисну ти на видалення для типу що вже використався в звіті; 7)Натиснути на кнопку видалення для новоствореного типу.	1) Відображено таблицю із типами робіт; 2) З'являється форма; 3) З'являються повідомлення валідації; 4) Повідомлення зникають; 5) Тип створюється і потрапляє до таблиці із записами в кінець; 6) З'являється повідомлення про помилку видалення; 7) Новостворений тип видаляється та зникає із списку.
2w	Веб-інтерфейс	Звіти	1) Натиснути кнопку навігації «Звіти»; 2) Обрати тип звіту «Зарплати»; 3) Натиснути допоміжну кнопку «Попередній місяць»; 4) Натиснути кнопку «Створити звіт». 5) Обрати працівника у полі вибору працівників; 6) Натиснути кнопку «Створити звіт»; 7) Відмінити вибір типу звіту; 8) Встановити значення поля «Дата по», раднішим ніж дата поля «Дата з»	1) Відображається сторінка із полями для вибору параметрів звіту; 2) Змінюється значення поля; 3) В полях для дати обираються перший і останній день попереднього місяця; 4) Завантажується звіт що місти зарплати за минулий місяць для всіх працівників; 5) Змінюється значення поля; 6) Завантажується звіт із зарплатою працівника за минулий місяць. 7) Помилка валідації форми; 8) Відображається вспливаюче повідомлення про помилку вибору звітного періоду.

4.3 Аналіз результатів тестування

Під час тестування було виконано перевірку всіх компонентів системи та інтерфейсів користувачів. В результаті проведених експериментів було виявлено, що базова функціональність реалізована у відповідності з вимогами до ПЗ і є повністю працездатною.

Існують некритичні проблеми, пов’язані з текстом повідомлень від бота та із текстом у веб-інтерфейсі. Також було виявлено, що відсутня валідація даних на сторінці веб-інтерфейсу при створенні нової категорії робіт.

Тестування проводилося протягом розробки програмного забезпечення та по завершенню його реалізації. Всі ці кроки тестування необхідні для того, щоб користувачі даної системи отримали готовий програмний продукт із мінімальною кількістю багів та неточностей. Чим більше, та детальніше, тест кейсів буде описано для тестування, ти краще буде протестована система.

ВИСНОВКИ

Для виконання дипломного проекту було виконано ряд робіт, що необхідні при розробці будь-якого програмного забезпечення. Все розпочинається із аналізу предметної області. Предметною областю для цього дипломного проекту є приватне підприємство із виготовлення текстильних виробів. Було проведено дослідження всіх існуючих процесів підприємства із метою визначення місць які потрібно автоматизувати. Одним із процесів, що потрібно автоматизувати це збір інформації про виконанні роботи. Саме цей процес був досліджений детальніше за всі інші. Під час дослідження було встановлено всі вимоги до програмного забезпечення, яке має покращити процес збору і обробки інформації на підприємстві. Розроблена система має бути зручною у використані, не вимагати додаткових ресурсів та техніки при його використанні. Інтерфейс користувача має бути простим і інтуїтивним. Інтерфейсом користувачів буде Telegram-бот.

На основі поставлених вимог, було виконано проектування майбутнього програмного забезпечення. На цьому етапі було здійснено аналіз програмних засобів та методів, які необхідно використати під час розробки системи. Було визначено, що в реалізації цього проекту буде застосовано два архітектурних підходи «сервер-сервер» та «клієнт-сервер». Комбінація цих архітектурних підходів необхідна для організації взаємодії всіх компонентів системи між собою. Загальними складовими системи є сервера Telegram, сервер розроблюваного ПЗ та веб-інтерфейс для користувачів системи, що займають роль керівників бізнесу. Для реалізації серверної частини розроблюваного ПЗ було обрано високопродуктивний та зручний у розробці PHP фреймворк Symfony 5. Це сучасний засіб для розробки серверних проектів різних напрямків та складності. Для цього фреймворку існує багато уже готових модулів, які дозволяють економити ресурси під час розробки на тривіальні задачі. Для реалізації веб-інтерфейсу застосовуватиметься фронтенд фреймворк Vue.js. Він

використовується для швидкої розробки SPA, що забезпечуються своїх користувачів зручним інтерфейсом користування складних систем. Саме тому було обрано цей фреймворк, як основний засіб для створення інтерфейсу для керівника підприємства.

По завершенню аналізу і проектування розпочався етап реалізації. На цьому етапі було виконане детальне проектування модулів системи та їх програмування. Було створено підсистему опрацювання повідомлень від бота. Саме ця система виконує основну функцію опрацювання даних, для якої виконується розробка цього програмного продукту. Але ця підсистема не матиме сенсу, якщо зібраними даними ніхто не буде користуватися. Тому реалізація всіх інших підсистем дозволяє користуватися цими даними. Реалізація систем виконувалася відносно до прийнятих стандартів кодування і дотримання загальних рекомендацій створення додатків на Symfony 5 для Rest API систем.

Під час реалізації, всі частини системи підвалися початковому функціональному тестуванню, щоб визначити робото здібність розроблених модулів. Але по завершенню розробки будь-якого програмного забезпечення, необхідно виконувати повне інтеграційне тестування системи, щоб виявити недоліки реалізації і запустити проект із найменшою кількістю багів. Саме із цієї причини, було виконано повне тестування системи збору і обробки інформації, що було реалізовано рамках цього дипломного проекту. Під час процесу тестування було виявлено кілька неточностей в текстах програми, що відображаються в інтерфейсах користувачів.

Результатом виконання всіх описаних вище етапів розробки ПЗ, було отримано систему, яка автоматизує процес збору інформації на підприємстві зручним способом для працівників, а саме за допомогою Telegram-бота, та відображає оброблену інформацію у простому і зручному інтерфейсі для керівника. Ці покращення дозволить скоротити час на задачі, що виконуються працівниками щодня та забезпечить їх можливість переглядати результати своїх виконаних робіт за певні періоди. Для керівників бізнесу, це ПЗ дозволить

повністю замінити процес збору і обробки інформації про виконані роботи. Керівнику потрібно буде лише запускати генерацію потрібних звітів і отримати вже готові підраховані числа. Але при запуску системи, потрібно буде одноразово виконати налаштування компонентів системи. Всі ці покращення позбавлять своїх користувачів зайвої роботи, яка може бути автоматизована.

Розроблене ПЗ можна використовувати в будь-якій галузі, де ще не автоматизовані процес збору інформації про виконані роботи. Єдиною відмінністю між іншими галузями і підприємством текстильних виробів, буде перелік можливих виконаних робіт. Тобто дане рішення є універсальним для будь-якого бізнесу, що потребує простого у використані засобу автоматизації процесів.

Будь який програмний продукт може розвиватися далі та впроваджувати нові функції, необхідні для його користувачів. Не є виключенням і розроблений програмний продукт. Для користувачів веб-інтерфейсу можна добавити можливість генерувати звіти із іншими наборами даних та у іншій формі. Наприклад, можна створити звіт, який буде зіставляти результати різних типів робіт, які виконуються для виготовлення одного типу продукту. Це може знадобитися для виявлення недобросовісних працівників на підприємстві, якщо цьому є місце. Для користувачів ботів, було б добре надати можливість перереєструватися в системі під іншим акантом у випадку втрати доступу до попереднього.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Топ 10 найкращих CRM систем для України [Електронний ресурс] // Портал IT новин для українських компаній LiveBusiness. – Режим доступу: https://www.livebusiness.com.ua/tools/crm/ (дата звернення 23.05.2022). – Назва з екрану.

2. Мартін Фаулер: UML. Основи (3-е видання) / М.Фаулер. – Харків : Символ-Т, 2013. – 192 с.

3. Введення. Що таке Vue? [Електронний ресурс] // Офіційний сайт фреймворку Vue.ls. – Режим доступу: https://vuejs.org/guide/introduction.html (дата звернення 23.05.2022). – Назва з екрану.

4. Реалізація веб-хука на прикладі взаємодії сервісів із онлайн касами [Електронний ресурс] // Портал обміну новин в IT сфері HABR. – Режим доступу: https://habr.com/ru/company/dreamkas/blog/332798/ (дата звернення 23.05.2022). – Назва з екрану.

5. Single Page Web Applications: JavaScript end-to-end / M. Mikowski, J. Powell. – USA : Manning Publications, 2013. – 432 с.

6. Telegram Bot API [Електронний ресурс] // Офіційний сайт організації Telegram. – Режим доступу: https://core.telegram.org/bots/api (дата звернення 23.05.2022). – Назва з екрану.

7. Learning SQL: Master SQL Fundamentals / A. Beaulieu. – USA : O'Reilly Media, 2009. – 338 с.

8. NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence / P. J. Sadalage M. Fowler. – USA : Addison Wesley, 2012. – 234 с.

9. PHP 8: об'єкти, шаблони і методики програмування / М. Зандстра. – Київ : Діалектика, 2021. – 866 с.

10. Symfony: Швидкий старт / F. Potencier. – Fabien Potencier, 2020. – 364 с.

11. RabbitMQ. Частина 1. Введення. Erlang, AMQP [Електронний ресурс] // Портал обміну новин в IT сфері HABR. – Режим доступу: https://habr.com/ru/post/488654/ (дата звернення 23.05.2022). – Назва з екрану.

12. Що таке Docker і навіщо він? [Електронний ресурс] // Портал обміну новин в IT сфері QualityAssurenceGroup. – Режим доступу: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення 23.05.2022). – Назва з екрану.

13. Мова запитів Doctrine [Електронний ресурс] // Офіційний сайт проекту Doctrine. – Режим доступу: https://www.doctrine-project.org/projects/doctrine-orm/en/2.9/reference/dql-doctrine-query-language.html (дата звернення 23.05.2022). – Назва з екрану.

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Розробка програмного продукту виконується в рамках проекту по автоматизації процесів збору та моніторингу результатів виконаних робіт на підприємстві за допомогою бота в Telegram.

А.1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення. Тема проекту: Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram.

А.2 Призначення розробки

Розроблений програмний продукт призначений для автоматизації процесу збору і обробки інформації про виконані роботи на підприємстві текстильних виробів. Також це ПЗ повинно надати зручний і простий інтерфейс для налаштування цієї системи і моніторингу опрацьованих даних.

Користувачами системи є працівники підприємства та його керівництво. Працівники використовують Telegram-бот для подачі результатів, а керівник використовує веб-інтерфейс для налаштування компонентів системи і генерації звітів. Ці функції дозволять відмовитися від процесів, які вимагали певного часу та ресурсів на їх виконання.

А.3 Вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Telegram-бот повинен забезпечити своїх користувачів такими функціями:

– реєстрація в боті, для можливості користуватися системою;
– вибір типу виконаної роботи;
– введення результатів виконаних робіт для обраного типу;
– встановлення дати для якої виконується звітність;
– отримання результатів виконаних робіт за попередні періоди.

Реєстрація в боті необхідна, щоб система могла ідентифікувати, що за користувач подає результати. А також для того щоб чат із ботом можна було асоціювати із записом працівника в БД.

Необхідно розробити спосіб представлення типів робіт в боті, щоб користувачеві було зручно шукати і обирати необхідний тип роботи для звітності.

Можливість встановлення дату звітності необхідно реалізувати для того щоб працівники могли вносити дані за пропущені дні, або правити внесені дані, якщо ті були подані не вірно.

Можливість перегляду даних про виконані роботи необхідна, щоб працівник міг проводити певний аналіз своєї продуктивності.

Веб-інтерфейс повинен забезпечити свого користувача наступними функціями:

– добавлення працівників в систему;
– створення категорій робіт;
– створення типів робіт;
– створення кнопок, що представлятимуть категорії і типи робіт в боті;
– генерація звітів про виконані роботи та зарплати.

Перш ніж працівник зможе зареєструватися в системі, керівник повинен добавити запис користувача в БД і отримати унікальне посилання реєстрації та код підтвердження для працівника.

Список робіт і їхнє групування виконується за допомогою простих форм, які містять поля для введення назв і іншої необхідної інформації. Для кожного із цих об’єктів потрібно створити кнопку в системі, щоб працівник міг натиснути на неї. Тому якщо є необхідність прибрати якийсь тип роботи із вибору чи цілу групу робіт, то необхідно видалити відповідну кнопку із системи використовуючи веб-інтерфейс. Всі вище описані дії потрібно буде виконати раз при початковому налаштуванні системи, за винятком роботи із записами працівників. Є необхідність добавляти їх час від часу, так як дуже часто на підприємство приходять нові працівники.

Головною задачею цієї системи є збір інформації та її обробка. Всі ці дані можуть бути представлені керівникові у вигляді звіту у форматі CSV файлу. Система повинна генерувати два типи звітів: про обсяги виконаних робіт та обраховані зарплати на основі цих виконаних робіт. Керівник повинен мати можливість вибирати звітний період та окремого працівника за необхідності.

А.3.2 Вимоги до надійності

Розроблюване ПЗ повинно мати:

– можливість не втрачати дані із БД у випадку будь-яких збоїв на сервері чи при виконанні програми;

– всі дії користувачів системи повинні супроводжуватися відповідними валідаціями, щоб запобігти можливим спробам нашкодити системі;

– система не повинна втратити жодного повідомлення чи запиту від своїх користувачів;

– помилки при роботі з Telegram API або із вхідними даними користувача повинні певним чином оброблятись, щоб не спричиняти ризиків працездатності ПЗ.

А.3.3 Вимоги до складу та параметрів технічних засобів

Системні вимоги для роботи ПЗ повинні бути наступними:

– тактова частота процесора >= 1.6 МГц;

– кількість ядер >= 2;

– обсяг оперативної пам’яті >= 8 Гб;

– обсяг вільного дискового простору >= 30 Гб;

– підключення до Інтернету.

Для роботи з системою із сторони працівників, повинен бути встановлений додаток Telegram або використана його веб-версії в браузері, що дозволить взаємодіяти із ботом. Для керівника підприємства потрібно будь-який засіб де можна відкрити веб-сайт.

А.3.4 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення може бути запущене на любій операційній системі де є встановлений Docker і Docker-Compose.

А.3.5 Вимоги до транспортування та зберігання

Програма, її документація поставляються у цифровому вигляді. Умови експлуатації програмного забезпечення збігаються з умовами експлуатації серверу, на якому буде розміщене ПЗ.

А.3.6 Спеціальні вимоги

Програма повинна мати дружній інтерфейс, розрахований на користувача середньої кваліфікації (з точки зору комп'ютерної грамотності). Засоби та методи реалізації визначаються виконавцем.

А.4. Вимоги до програмної документації

В ході розробки програми повинні бути підготовлені: текст програми, опис програми, інструкція з запуску та зупинки ПЗ, програма і методика випробувань, керівництво користувача.

А.5. Стадії та етапи розробки

Стадії та етапи розробки програмного продукту подані в таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки

Стадія розробки	Зміст робіт
Технічне завдання	Визначення вимог до розроблюваного ПЗ;
Проектування системи	Визначення необхідних складових системи; Визначення архітектурного підходу для реалізації; Визначення засобів і методів реалізації.
Детальне моделювання системи	Декомпозиція спроектованої систем на модулі; Деталізація кожного модуля окремо.
Розробка ПЗ	Реалізація програмного забезпечення.
Тестування системи	Тестування; Виправлення помилок.
Здача проекту	Передача вихідного коду та документації замовнику.

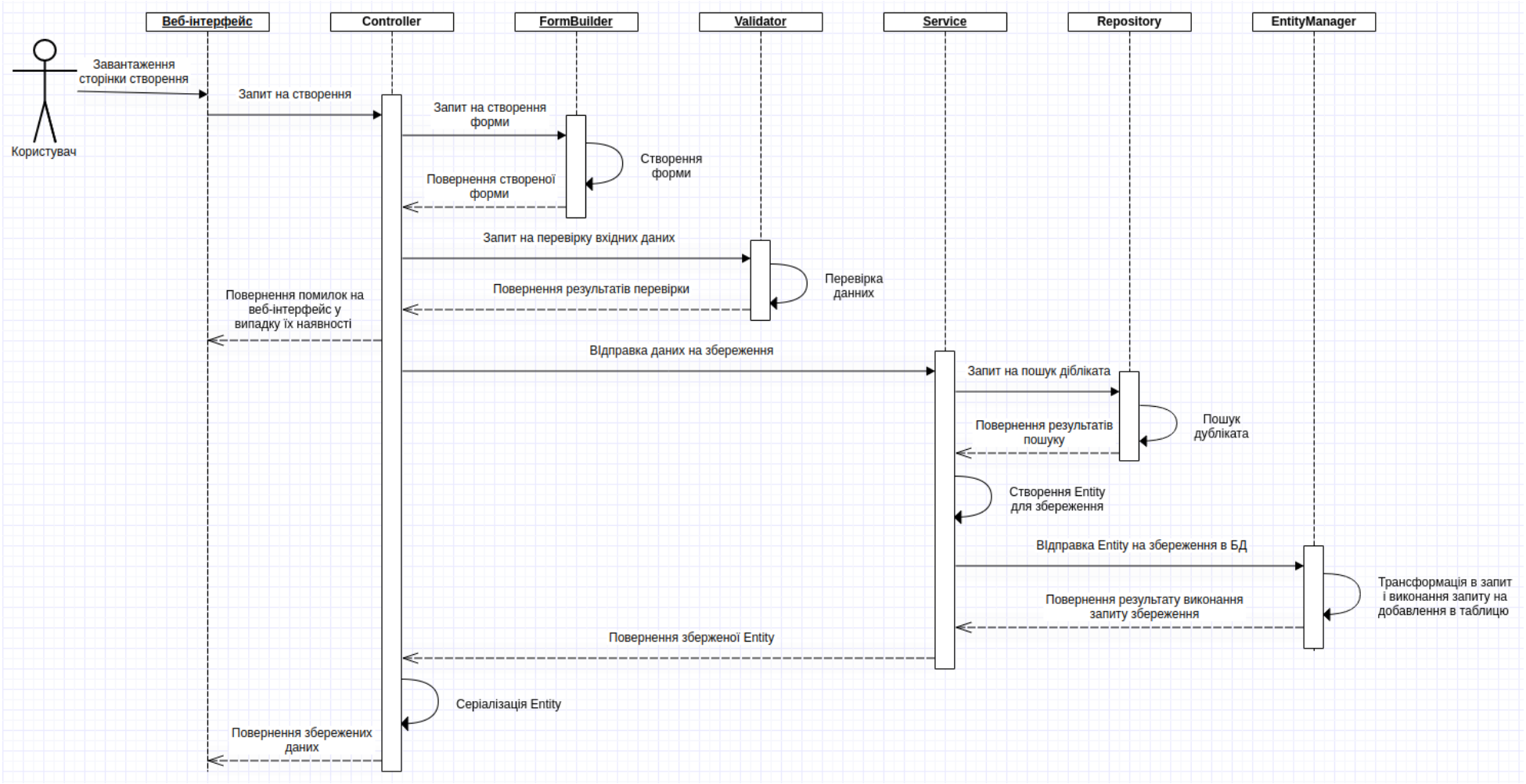
А.6. Порядок контролю та приймання

Контроль і приймання розробки здійснюються на основі розробленої методики випробувань. При цьому перевіряється виконання всіх функцій програми групою користувачів та QA-спеціалістів. Прийом ПЗ замовником здійснюється після успішного тестування.

ДОДАТОК Б

(довідковий)

ДІАГРАМА ПОСЛІДОВНОСТІ ПРОЦЕСУ СТВОРЕННЯ ОБ’ЄКТА ТИПУ РОБОТИ



ДОДАТОТ В

(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

В.1 Класи міграцій

Клас міграції Version20220505194726

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220505194726 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE button (id INT NOT NULL, result_type_id INT
DEFAULT NULL, result_type_group_id INT DEFAULT NULL, text VARCHAR(50) NOT NULL,
PRIMARY KEY(id))');
        $this->addSql('CREATE INDEX IDX_3A06AC3DA7750E77 ON button
(result_type_id)');
        $this->addSql('CREATE INDEX IDX_3A06AC3DB328ECB8 ON button
(result_type_group_id)');
        $this->addSql('CREATE TABLE chat (id INT NOT NULL, employee_id INT NOT NULL,
chat_id BIGINT NOT NULL, user_id BIGINT NOT NULL, firstname VARCHAR(255) NOT NULL,
is_registered BOOLEAN DEFAULT false NOT NULL, expected_action VARCHAR(255) DEFAULT
NULL, PRIMARY KEY(id))');
        $this->addSql('CREATE INDEX IDX_659DF2AA8C03F15C ON chat (employee_id)');
        $this->addSql('CREATE TABLE employee (id INT NOT NULL, name VARCHAR(255) NOT
NULL, reg_token UUID NOT NULL, is_registered BOOLEAN DEFAULT false NOT NULL,
approval_key INT NOT NULL, registered_at TIMESTAMP(0) WITHOUT TIME ZONE DEFAULT NULL,
PRIMARY KEY(id))');
        $this->addSql('CREATE UNIQUE INDEX UNIQ_5D9F75A18A8BCF4D ON employee
(reg_token)');
        $this->addSql('CREATE TABLE measurement_type (id INT NOT NULL, name
VARCHAR(50) NOT NULL, contraction VARCHAR(20) DEFAULT NULL, PRIMARY KEY(id))');
        $this->addSql('CREATE TABLE result (id INT NOT NULL, chat_id INT NOT NULL,
result_type_id INT NOT NULL, amount INT NOT NULL, actual_price INT NOT NULL, sum INT
```

```
NOT NULL, year SMALLINT NOT NULL, month SMALLINT NOT NULL, day SMALLINT NOT NULL,
created_at TIMESTAMP(0) WITHOUT TIME ZONE NOT NULL, PRIMARY KEY(id))');
        $this->addSql('CREATE INDEX IDX_136AC1131A9A7125 ON result (chat_id)');
        $this->addSql('CREATE INDEX IDX_136AC113A7750E77 ON result
(result_type_id)');
        $this->addSql('CREATE TABLE result_type (id INT NOT NULL,
result_type_group_id INT NOT NULL, measurement_type_id INT NOT NULL, name
VARCHAR(255) NOT NULL, price INT NOT NULL, PRIMARY KEY(id))');
        $this->addSql('CREATE INDEX IDX_1CE54C67B328ECB8 ON result_type
(result_type_group_id)');
        $this->addSql('CREATE INDEX IDX_1CE54C678B4CC8FE ON result_type
(measurement_type_id)');
        $this->addSql('CREATE TABLE result_type_group (id INT NOT NULL, name
VARCHAR(255) NOT NULL, description VARCHAR(255) DEFAULT NULL, PRIMARY KEY(id))');
        $this->addSql('ALTER TABLE button ADD CONSTRAINT FK_3A06AC3DA7750E77 FOREIGN
KEY (result_type_id) REFERENCES result_type (id) NOT DEFERRABLE INITIALLY
IMMEDIATE');
        $this->addSql('ALTER TABLE button ADD CONSTRAINT FK_3A06AC3DB328ECB8 FOREIGN
KEY (result_type_group_id) REFERENCES result_type_group (id) NOT DEFERRABLE INITIALLY
IMMEDIATE');
        $this->addSql('ALTER TABLE chat ADD CONSTRAINT FK_659DF2AA8C03F15C FOREIGN
KEY (employee_id) REFERENCES employee (id) NOT DEFERRABLE INITIALLY IMMEDIATE');
        $this->addSql('ALTER TABLE result ADD CONSTRAINT FK_136AC1131A9A7125 FOREIGN
KEY (chat_id) REFERENCES chat (id) NOT DEFERRABLE INITIALLY IMMEDIATE');
        $this->addSql('ALTER TABLE result ADD CONSTRAINT FK_136AC113A7750E77 FOREIGN
KEY (result_type_id) REFERENCES result_type (id) NOT DEFERRABLE INITIALLY
IMMEDIATE');
        $this->addSql('ALTER TABLE result_type ADD CONSTRAINT FK_1CE54C67B328ECB8
FOREIGN KEY (result_type_group_id) REFERENCES result_type_group (id) NOT DEFERRABLE
INITIALLY IMMEDIATE');
        $this->addSql('ALTER TABLE result_type ADD CONSTRAINT FK_1CE54C678B4CC8FE
FOREIGN KEY (measurement_type_id) REFERENCES measurement_type (id) NOT DEFERRABLE
INITIALLY IMMEDIATE');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE result DROP CONSTRAINT FK_136AC1131A9A7125');
        $this->addSql('ALTER TABLE chat DROP CONSTRAINT FK_659DF2AA8C03F15C');
        $this->addSql('ALTER TABLE result_type DROP CONSTRAINT FK_1CE54C678B4CC8FE');
        $this->addSql('ALTER TABLE button DROP CONSTRAINT FK_3A06AC3DA7750E77');
        $this->addSql('ALTER TABLE result DROP CONSTRAINT FK_136AC113A7750E77');
        $this->addSql('ALTER TABLE button DROP CONSTRAINT FK_3A06AC3DB328ECB8');
        $this->addSql('ALTER TABLE result_type DROP CONSTRAINT FK_1CE54C67B328ECB8');
        $this->addSql('DROP TABLE button');
        $this->addSql('DROP TABLE chat');
        $this->addSql('DROP TABLE employee');
        $this->addSql('DROP TABLE measurement_type');
        $this->addSql('DROP TABLE result');
        $this->addSql('DROP TABLE result_type');
        $this->addSql('DROP TABLE result_type_group');
    }
}
```

Клас міграції Version20220505194829

```
<?php
```

```
declare(strict_types=1);

namespace DoctrineMigrations;

use App\Entity\Result\MeasurementType;
use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220505194829 extends AbstractMigration
{
    public function up(Schema $schema): void
    {
        $this->connection->insert('measurement_type', ['id' =>
MeasurementType::TYPE_UNIT, 'name' => 'Одиниця', 'contraction' => 'одн.']);
        $this->connection->insert('measurement_type', ['id' =>
MeasurementType::TYPE_METER, 'name' => 'Метер', 'contraction' => 'м.']);
        $this->connection->insert('measurement_type', ['id' =>
MeasurementType::TYPE_KILOGRAM, 'name' => 'Кілограм', 'contraction' => 'кг.']);
        $this->connection->insert('measurement_type', ['id' =>
MeasurementType::TYPE_HOUR, 'name' => 'Година', 'contraction' => 'год.']);
    }

    public function down(Schema $schema): void
    {
        $measurementTypeIds = [MeasurementType::TYPE_UNIT,
MeasurementType::TYPE_METER, MeasurementType::TYPE_KILOGRAM,
MeasurementType::TYPE_HOUR];
        $this->connection->executeStatement(
            'DELETE FROM measurement_type WHERE id in (' . implode(',',
$measurementTypeIds) . ')'
        );
    }
}
```

Клас міграції Version20220506142312

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220506142312 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
```

```
        $this->addSql('ALTER TABLE chat ADD reported_data TIMESTAMP(0) WITHOUT TIME
ZONE DEFAULT NULL');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE chat DROP reported_data');
    }
}
```

Клас міграції Version20220506182331

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220506182331 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE chat ADD expected_result_type_id INT DEFAULT
NULL');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE chat DROP expected_result_type_id');
    }
}
```

Клас міграції Version20220506213401

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;
```

```
/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220506213401 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE result ADD updated_at TIMESTAMP(0) WITHOUT TIME
ZONE DEFAULT NULL');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE result DROP updated_at');
    }
}
```

Клас міграції Version20220507214807

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220507214807 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE result ADD reported_date TIMESTAMP(0) WITHOUT TIME
ZONE NOT NULL');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE result DROP reported_date');
    }
}
```

В.2 Класи контролерів

Клас BaseApiController

```
<?php

namespace App\Controller\Api;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\FormInterface;

abstract class BaseApiController extends AbstractController
{
    /**
     * @var string
     */
    public const ERROR_MESSAGE_FORM_NOT_SENT = 'The form in not sent.';

    /**
     * @param FormInterface $form
     *
     * @return array
     */
    protected function getErrorsFromForm(FormInterface $form): array
    {
        $errors = [];

        foreach ($form->getErrors() as $error) {
            $errors[] = $error->getMessage();
        }

        foreach ($form->all() as $childForm) {
            if ($childForm instanceof FormInterface) {
                if ($childErrors = $this->getErrorsFromForm($childForm)) {
                    $errors[$childForm->getName()] = $childErrors;
                }
            }
        }

        return $errors;
    }
}
```

Клас EmployeeController

```
<?php

namespace App\Controller\Api\Employee;

use App\Controller\Api\BaseApiController;
use App\Exception\EmployeeRegisteredException;
use App\Form\Type\Employee\EmployeeType;
use App\Mapper\Request\EmployeeDtoMapper;
use App\Repository\Employee\EmployeeRepository;
use App\Service\Employee\EmployeeService;
use Symfony\Component\HttpFoundation\Request;
```

```
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route(path="/api/employees")
 */
class EmployeeController extends BaseApiController
{
    /**
     * @var \App\Service\Employee\EmployeeService
     */
    protected $employeeService;

    /**
     * @var \App\Repository\Employee\EmployeeRepository
     */
    protected $employeeRepository;

    /**
     * @param \App\Service\Employee\EmployeeService $employeeService
     * @param \App\Repository\Employee\EmployeeRepository $employeeRepository
     */
    public function __construct(
        EmployeeService $employeeService,
        EmployeeRepository $employeeRepository
    ) {
        $this->employeeService = $employeeService;
        $this->employeeRepository = $employeeRepository;
    }

    /**
     * @Route(methods={"POST"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     * @param \App\Mapper\Request\EmployeeDtoMapper $employeeDtoMapper
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function postEmployee(Request $request, EmployeeDtoMapper
$employeeDtoMapper): Response
    {
        $form = $this->createForm(EmployeeType::class);
        $form->handleRequest($request);

        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
        }

        $employee = $this->employeeService->createEmployee($form->getData());

        return $this->json(
            ['employee' => $employeeDtoMapper->mapEmployeeToEmployeeDto($employee)],
            Response::HTTP_CREATED
        );
    }

    /**
     * @Route(methods={"DELETE"}, path="/{id}")
```

```
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function deleteEmployee(Request $request, int $id): Response
    {
       $employee = $this->employeeRepository->find($id);

       if (!$employee) {
           return $this->json(['message' => 'The employee is not found.'],
Response::HTTP_NOT_FOUND);
       }

        try {
            $this->employeeService->deleteEmployee($employee);
        } catch (EmployeeRegisteredException $employeeRegisteredException) {
            return $this->json(['message' => $employeeRegisteredException-
>getMessage()], Response::HTTP_BAD_REQUEST);
        }

        return $this->json([], Response::HTTP_NO_CONTENT);
    }

    /**
     * @Route(methods={"GET"})
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function getEmployees(): Response
    {
        return $this->json(['employees' => $this->employeeService->getEmployees()]);
    }
}
```

Клас MeasurementTypeController

```
<?php

namespace App\Controller\Api\Result;

use App\Controller\Api\BaseApiController;
use App\Repository\Result\MeasurementTypeRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route(path="/api/measurement-types")
 */
class MeasurementTypeController extends BaseApiController
{
    /**
     * @Route(methods={"GET"})
     *
     * @param \App\Repository\Result\MeasurementTypeRepository
$measurementTypeRepository
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
```

```
    public function getEmployees(MeasurementTypeRepository
$measurementTypeRepository): Response
    {
        return $this->json(['measurementTypes' => $measurementTypeRepository-
>findAll()]);
    }
}
```

Клас ResultTypeController

```
<?php

namespace App\Controller\Api\Result;

use App\Controller\Api\BaseApiController;
use App\Dto\Request\ResultTypeDto;
use App\Form\Model\Result\ResultTypeModel;
use App\Form\Type\Result\ResultTypeType;
use App\Repository\Result\ResultTypeRepository;
use App\Service\Result\ResultTypeService;
use Symfony\Component\HttpFoundation\Exception\BadRequestException;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route(path="/api/result-types")
 */
class ResultTypeController extends BaseApiController
{
    /**
     * @var \App\Service\Result\ResultTypeService
     */
    private $resultTypeService;

    /**
     * @var \App\Repository\Result\ResultTypeRepository
     */
    private $resultTypeRepository;

    /**
     * @param \App\Service\Result\ResultTypeService $resultTypeService
     * @param \App\Repository\Result\ResultTypeRepository $resultTypeRepository
     */
    public function __construct(
        ResultTypeService $resultTypeService,
        ResultTypeRepository $resultTypeRepository
    ) {
        $this->resultTypeService = $resultTypeService;
        $this->resultTypeRepository = $resultTypeRepository;
    }

    /**
     * @Route(methods={"GET"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
```

```
    public function getTypes(Request $request): Response
    {
        return $this->json(['resultTypes' => $this->resultTypeService-
>getResultTypes($request->query->all())]);
    }

    /**
     * @Route(methods={"POST"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function createType(Request $request): Response
    {
        $form = $this->createForm(ResultTypeType::class);
        $form->handleRequest($request);

        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
        }

        $resultType = $this->resultTypeService->saveResultType($form->getData());

        $resultTypeDto = (new ResultTypeDto())
            ->setId($resultType->getId())
            ->setName($resultType->getName())
            ->setPrice($resultType->getPrice() / 100)
            ->setMeasurementType($resultType->getMeasurementType()->getId())
            ->setResultTypeGroup($resultType->getResultTypeGroup()->getId());

        return $this->json(['resultType' => $resultTypeDto]);
    }

    /**
     * @Route(methods={"PUT"}, path="/{id}")
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function updateType(Request $request, int $id): Response
    {
        $resultType = $this->resultTypeRepository->find($id);

        if (!$resultType) {
            return $this->json(['error' => 'ResultType was not found.'],
Response::HTTP_NOT_FOUND);
        }

        $resultTypeModel = (new ResultTypeModel())
            ->setName($resultType->getName())
            ->setResultTypeGroup($resultType->getResultTypeGroup())
            ->setMeasurementType($resultType->getMeasurementType());

        $form = $this->createForm(ResultTypeType::class, $resultTypeModel, ['method'
=> 'PUT']);
        $form->handleRequest($request);
```

```
        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
        }

        $resultType = $this->resultTypeService->saveResultType($form->getData(),
$resultType);

        $resultTypeDto = (new ResultTypeDto())
            ->setId($resultType->getId())
            ->setName($resultType->getName())
            ->setPrice($resultType->getPrice() / 100)
            ->setMeasurementType($resultType->getMeasurementType()->getId())
            ->setResultTypeGroup($resultType->getResultTypeGroup()->getId());

        return $this->json(['resultType' => $resultTypeDto]);
    }

    /**
     * @Route(methods={"DELETE"}, path="/{id}")
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function deleteType(Request $request, int $id): Response
    {
        $resultType = $this->resultTypeRepository->find($id);

        if (!$resultType) {
            return $this->json(['message' => 'ResultType was not found.'],
Response::HTTP_NOT_FOUND);
        }

        try {
            $this->resultTypeService->deleteResultType($resultType);
        } catch (BadRequestException $exception) {
            return $this->json(['message' => $exception->getMessage()],
Response::HTTP_BAD_REQUEST);
        }

        return new Response(null, Response::HTTP_NO_CONTENT);
    }
}
```

Клас ResultTypeGroupController

```
<?php

namespace App\Controller\Api\Result;

use App\Controller\Api\BaseApiController;
use App\Form\Model\Result\ResultTypeGroupModel;
```

```
use App\Form\Type\Result\ResultTypeGroupType;
use App\Repository\Result\ResultTypeGroupRepository;
use App\Service\Result\ResultTypeGroupService;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route(path="/api/result-type-groups")
 */
class ResultTypeGroupController extends BaseApiController
{
    /**
     * @var \App\Service\Result\ResultTypeGroupService
     */
    private $resultTypeGroupService;

    /**
     * @var \App\Repository\Result\ResultTypeGroupRepository
     */
    private $resultTypeGroupRepository;

    /**
     * @param \App\Service\Result\ResultTypeGroupService $resultTypeGroupService
     */
    public function __construct(
        ResultTypeGroupService $resultTypeGroupService,
        ResultTypeGroupRepository $resultTypeGroupRepository
    ) {
        $this->resultTypeGroupService = $resultTypeGroupService;
        $this->resultTypeGroupRepository = $resultTypeGroupRepository;
    }

    /**
     * @Route(methods={"GET"})
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function getGroups(): Response
    {
        return $this->json(['resultTypeGroups' => $this->resultTypeGroupRepository-
>findAll()]);
    }

    /**
     * @Route(methods={"POST"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function createGroup(Request $request): Response
    {
        $form = $this->createForm(ResultTypeGroupType::class);
        $form->handleRequest($request);

        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
```

```
        }

        $resultTypeGroup = $this->resultTypeGroupService->saveResultTypeGroup($form->getData());

        return $this->json(['resultTypeGroup' => $resultTypeGroup]);
    }

    /**
     * @Route(methods={"PUT"}, path="/{id}")
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function updateGroup(Request $request, int $id): Response
    {
        $resultTypeGroup = $this->resultTypeGroupRepository->find($id);

        if (!$resultTypeGroup) {
            return $this->json(['error' => 'ResultTypeGroup was not found.'], Response::HTTP_NOT_FOUND);
        }

        $resultTypeGroupModel = (new ResultTypeGroupModel())
            ->setName($resultTypeGroup->getName())
            ->setDescription($resultTypeGroup->getDescription());

        $form = $this->createForm(ResultTypeGroupType::class, $resultTypeGroupModel, ['method' => 'PUT']);
        $form->handleRequest($request);

        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT], Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form), Response::HTTP_BAD_REQUEST);
        }

        $resultTypeGroup = $this->resultTypeGroupService->saveResultTypeGroup($form->getData(), $resultTypeGroup);

        return $this->json(['resultTypeGroup' => $resultTypeGroup]);
    }
}
```

Клас ButtonController

```
<?php

namespace App\Controller\Api\Telegram;

use App\Controller\Api\BaseApiController;
use App\Form\Model\Telegram\ButtonModel;
use App\Form\Type\Telegram\ButtonType;
use App\Repository\Telegram\ButtonRepository;
```

```
use App\Service\Telegram\ButtonService;
use Symfony\Component\HttpFoundation\Exception\BadRequestException;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route(path="/api/buttons")
 */
class ButtonController extends BaseApiController
{
    /**
     * @var \App\Service\Telegram\ButtonService
     */
    private $buttonService;
    /**
     * @var \App\Repository\Telegram\ButtonRepository
     */
    private $buttonRepository;

    /**
     * @param \App\Service\Telegram\ButtonService $buttonService
     * @param \App\Repository\Telegram\ButtonRepository $buttonRepository
     */
    public function __construct(
        ButtonService $buttonService,
        ButtonRepository $buttonRepository
    ) {
        $this->buttonService = $buttonService;
        $this->buttonRepository = $buttonRepository;
    }

    /**
     * @Route(methods={"GET"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function getButtons(Request $request): Response
    {
        return $this->json(['buttons' => $this->buttonService->getButtons()]);
    }

    /**
     * @Route(methods={"POST"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function createButtons(Request $request): Response
    {
        $form = $this->createForm(ButtonType::class);
        $form->handleRequest($request);

        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
```

```
        }

        $button = $this->buttonService->saveButton($form->getData());

        return $this->json(['button' => $button]);
    }

    /**
     * @Route(methods={"PUT"}, path="/{id}")
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function updateButton(Request $request, int $id): Response
    {
        $button = $this->buttonRepository->find($id);

        if (!$button) {
            return $this->json(['error' => 'Button was not found.'],
Response::HTTP_NOT_FOUND);
        }

        $ButtonModel = (new ButtonModel())
            ->setText($button->getText())
            ->setResultTypeGroup($button->getResultTypeGroup())
            ->setResultType($button->getResultType());

        $form = $this->createForm(ButtonType::class, $ButtonModel, ['method' =>
'PUT']);
        $form->handleRequest($request);

        if (!$form->isSubmitted()) {
            return $this->json(['error' => static::ERROR_MESSAGE_FORM_NOT_SENT],
Response::HTTP_BAD_REQUEST);
        }

        if (!$form->isValid()) {
            return $this->json($this->getErrorsFromForm($form),
Response::HTTP_BAD_REQUEST);
        }

        try {
            $button = $this->buttonService->saveButton($form->getData(), $button);
        } catch (BadRequestException $exception) {
            return $this->json(['error' => $exception->getMessage()],
Response::HTTP_BAD_REQUEST);
        }

        return $this->json(['button' => $button]);
    }
}
```

Клас TelegramWebhookController

```
<?php

namespace App\Controller;
```

```
use App\Service\Telegram\WebhookRequestBodyProcessorFactory;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class TelegramWebhookController extends AbstractController
{
    /**
     * @Route("/hook", name="telegram_webhook", methods={"POST"})
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     * @param \App\Service\Telegram\WebhookRequestBodyProcessorFactory
$webhookRequestBodyProcessorFactory
     *
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function index(
        Request $request,
        WebhookRequestBodyProcessorFactory $webhookRequestBodyProcessorFactory
    ): Response {
        $webhookRequestBodyProcessorFactory->createWebhookRequestBodyProcessor()
            ->processRequest($request->getContent());

        return new Response('', Response::HTTP_NO_CONTENT);
    }
}
```

Клас ReportsController

```
<?php

namespace App\Controller\Api\Result;

use App\Controller\Api\BaseApiController;
use App\Service\Result\Report\ReportBuilder;
use App\Service\Result\Report\ReportNameBuilder;
use App\Service\Result\Report\ReportQueryParamsValidator;
use App\Service\Result\Report\ReportTypeConstants;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route(path="/api/reports")
 */
class ReportsController extends BaseApiController
{
    /**
     * @Route(methods={"GET"}, path="/{reportType}")
     *
     * @param \Symfony\Component\HttpFoundation\Request $request
     * @param string $reportType
     * @param \App\Service\Result\Report\ReportBuilder $reportBuilder
     * @param \App\Service\Result\Report\ReportNameBuilder $reportNameBuilder
     * @param \App\Service\Result\Report\ReportQueryParamsValidator
$reportQueryParamsValidator
     *
     * @return \Symfony\Component\HttpFoundation\Response
```

```
     */
    public function getReport(
        Request $request,
        string $reportType,
        ReportBuilder $reportBuilder,
        ReportNameBuilder $reportNameBuilder,
        ReportQueryParamsValidator $reportQueryParamsValidator
    ): Response {
        $queryParameters = $this->filterQueryParameters($request->query->all());

        if (!$reportQueryParamsValidator-
>validateQueryParametersPerReportType($reportType, $queryParameters)) {
            return $this->json(['error' => 'Невірні параметри запиту на формування
звіту'], Response::HTTP_BAD_REQUEST);
        }

        $reportResource = $reportBuilder->generateReportFile($queryParameters,
$reportType);
        $response = new Response(stream_get_contents($reportResource));
        fclose($reportResource);
        $filename = $reportNameBuilder->generateNameByType($queryParameters,
$reportType);
        $response->headers->set('Content-Type', 'text/csv;charset=utf-8');
        $response->headers->set('Content-Disposition', 'attachment;
filename="'.$filename.'"');

        return $response;
    }

    protected function filterQueryParameters(array $queryParameters): array
    {
        $results = [];

        foreach ($queryParameters as $parameterName => $queryParameter) {
            if (in_array($parameterName,
ReportTypeConstants::AVAILABLE_QUERY_FILTERS)) {
                $results[$parameterName] = $queryParameter;
            }
        }

        return $results;
    }
}
```

В.3 Інші класи програми

Клас ResultTypeFixture

```
<?php

namespace App\DataFixtures;

use App\Entity\Result\MeasurementType;
use App\Entity\Result\ResultType;
use App\Entity\Result\ResultTypeGroup;
use App\Entity\Telegram\Button;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;
```

```
class ResultTypeFixture extends Fixture
{
    /**
     * @var \App\Repository\Result\MeasurementTypeRepository
     */
    private $measurementTypeRepository;

    /**
     * @var \App\Entity\Result\MeasurementType[]
     */
    private $measurementTypeMap = [];

    private $resultTypeGroupsData = [
        [
            'Стьожка',
            'Стьожка - опис',
            [
                [
                    MeasurementType::TYPE_UNIT,
                    'Наматрацники',
                    500,
                    'Ст. Наматрацників'
                ],
                [
                    MeasurementType::TYPE_UNIT,
                    'Покривала',
                    300,
                    'Ст. Покривал'
                ],
                [
                    MeasurementType::TYPE_UNIT,
                    'Ковдри',
                    400,
                    'Ст. Ковдр'
                ],
            ],
        ],
        [
            'Чесання',
            'Чесалка - опис',
            [
                [
                    MeasurementType::TYPE_KILOGRAM,
                    'Силікон',
                    200,
                    'Че. Силікону',
                ],
                [
                    MeasurementType::TYPE_KILOGRAM,
                    'Вовна',
                    250,
                    'Че. Вовни',
                ],
            ],
        ],
        [
            'Лямування',
            'Оверлог - опис',
            [
                [
                    MeasurementType::TYPE_UNIT,
                    'Наматрацники',
                    250,
```

```
                'Ля. Наматрацників',
            ],
            [
                MeasurementType::TYPE_UNIT,
                'Ковдри',
                250,
                'Ля. Ковдр',
            ],
        ],
    ],
    [
        'Покрій',
        'Ніж - опис',
        [
            [
                MeasurementType::TYPE_METER,
                'Ковдри',
                50,
                'Пок. Ковдр',
            ],
            [
                MeasurementType::TYPE_METER,
                'Подушки',
                50,
                'Пок. Подушок',
            ],
            [
                MeasurementType::TYPE_METER,
                'Покривала',
                50,
                'Пок. Покривал',
            ],
            [
                MeasurementType::TYPE_METER,
                'Постелі',
                75,
                'Пок. Постелей',
            ],
        ],
    ],
    [
        'Пошив',
        'Швейна машина - опис',
        [
            [
                MeasurementType::TYPE_UNIT,
                'Ковдри',
                250,
                'Пош. Ковдр',
            ],
            [
                MeasurementType::TYPE_UNIT,
                'Подушки',
                300,
                'Пош. Подушок',
            ],
            [
                MeasurementType::TYPE_UNIT,
                'Постелі',
                500,
                'Пош. Постелей',
            ],
            [
                MeasurementType::TYPE_UNIT,
```

```
                    'Напірники',
                    100,
                    'Пош. Напірників',
                ],
            ],
        ],
        [
            'Інші',
            'Інші види робіт',
            [
                [
                    MeasurementType::TYPE_HOUR,
                    'Погрузка',
                    5000,
                    'Погрузка',
                ],
                [
                    MeasurementType::TYPE_HOUR,
                    'Розгрузка',
                    4000,
                    'Розгрузка',
                ],
                [
                    MeasurementType::TYPE_HOUR,
                    'Поїздка',
                    5000,
                    'Поїздка',
                ],
                [
                    MeasurementType::TYPE_HOUR,
                    'Ремонт',
                    4000,
                    'Ремонт',
                ],
            ],
        ],
    ];

    public function load(ObjectManager $manager): void
    {
        $this->measurementTypeRepository = $manager-
>getRepository(MeasurementType::class);
        $this->loadMeasurementTypes();
        foreach ($this->resultTypeGroupsData as $resultTypeGroupData) {
            $resultTypeGroup = (new ResultTypeGroup())
                ->setName($resultTypeGroupData[0])
                ->setDescription($resultTypeGroupData[1]);

            $manager->persist($resultTypeGroup);

            $groupButton = (new Button())
                ->setText($resultTypeGroup->getName())
                ->setResultTypeGroup($resultTypeGroup);

            $manager->persist($groupButton);

            foreach ($resultTypeGroupData[2] as $resultTypeData) {
                $resultType = (new ResultType())
                    ->setResultTypeGroup($resultTypeGroup)
                    ->setMeasurementType($this-
>measurementTypeMap[$resultTypeData[0]])
                    ->setName($resultTypeData[1])
                    ->setPrice($resultTypeData[2]);
```

```
                $manager->persist($resultType);

                $typeButton = (new Button())
                    ->setText($resultTypeData[3])
                    ->setResultType($resultType);

                $manager->persist($typeButton);
            }

            $manager->flush();
        }

    }

    protected function loadMeasurementTypes(): void
    {
        $measurementTypes = $this->measurementTypeRepository->findAll();
        foreach ($measurementTypes as $measurementType) {
            $this->measurementTypeMap[$measurementType->getId()] = $measurementType;
        }
    }
}
```

Клас ResultsFixture

```
<?php

namespace App\DataFixtures;

use App\Entity\Result\MeasurementType;
use App\Entity\Result\Result;
use App\Entity\Result\ResultType;
use App\Entity\Result\ResultTypeGroup;
use App\Entity\Telegram\Button;
use App\Entity\Telegram\Chat;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Bundle\FixturesBundle\FixtureGroupInterface;
use Doctrine\Persistence\ObjectManager;

class ResultsFixture extends Fixture implements FixtureGroupInterface
{
    public function load(ObjectManager $manager): void
    {
        $date = new \DateTime('01/01/2022');
        do {
            $chat = $manager->getRepository(Chat::class)->findOneBy([]);
            $resultTypes = $manager->getRepository(ResultType::class)->findAll([]);
            $day = $date->format('d');
            $month = $date->format('m');
            $year = $date->format('Y');

            for ($i = 0; $i < 10; $i++) {
                $resultType = $resultTypes[random_int(0, count($resultTypes)-1)];
                $amount = random_int(1, 100) * 100;
                $result = (new Result())
                    ->setChat($chat)
                    ->setResultType($resultType)
                    ->setActualPrice($resultType->getPrice())
                    ->setAmount($amount)
```

```
                    ->setSum($amount * $resultType->getPrice())
                    ->setCreatedAt(new \DateTime())
                    ->setDay($day)
                    ->setMonth($month)
                    ->setYear($year)
                    ->setReportedDate($date);

                $manager->persist($result);
            }

            $manager->flush();
            $manager->clear();

            $date->add(new \DateInterval('P1D'));
        } while ($date->getTimestamp() < time());
    }

    public static function getGroups(): array
    {
        return [
            'results',
        ];
    }
}
```

Клас EmployeeDto

```
<?php

namespace App\Dto\Request;

use DateTimeInterface;

class EmployeeDto
{
    /**
     * @var int
     */
    protected $id;

    /**
     * @var string
     */
    protected $name;

    /**
     * @var string|null
     */
    protected $regLink;

    /**
     * @var string|null
     */
    protected $registeredAt;

    /**
     * @var int|null
     */
    protected $approvalKey;
```

```
/**
 * @return int
 */
public function getId(): int
{
    return $this->id;
}

/**
 * @param int $id
 *
 * @return $this
 */
public function setId(int $id): self
{
    $this->id = $id;

    return $this;
}

/**
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * @param string $name
 *
 * @return $this
 */
public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

/**
 * @return string
 */
public function getRegLink(): string
{
    return $this->regLink;
}

/**
 * @param string|null $regLink
 *
 * @return $this
 */
public function setRegLink(?string $regLink): self
{
    $this->regLink = $regLink;

    return $this;
}

/**
 * @return string|null
 */
```

```
    public function getRegisteredAt(): ?string
    {
        return $this->registeredAt;
    }

    /**
     * @param string|null $registeredAt
     *
     * @return $this
     */
    public function setRegisteredAt(?string $registeredAt): self
    {
        $this->registeredAt = $registeredAt;

        return $this;
    }

    /**
     * @return int|null
     */
    public function getApprovalKey(): ?int
    {
        return $this->approvalKey;
    }

    /**
     * @param int|null $approvalKey
     */
    public function setApprovalKey(?int $approvalKey): self
    {
        $this->approvalKey = $approvalKey;

        return $this;
    }
}
```

Клас ResultTypeDto

```
<?php

namespace App\Dto\Request;

class ResultTypeDto
{
    /**
     * @var int
     */
    protected $id;

    /**
     * @var string
     */
    protected $name;

    /**
     * @var int
     */
    protected $resultTypeGroup;
```

```
/**
 * @var int
 */
protected $measurementType;

/**
 * @var float
 */
protected $price;

/**
 * @return int
 */
public function getId(): int
{
    return $this->id;
}

/**
 * @param int $id
 */
public function setId(int $id): self
{
    $this->id = $id;

    return $this;
}

/**
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * @param string $name
 */
public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

/**
 * @return int
 */
public function getResultTypeGroup(): int
{
    return $this->resultTypeGroup;
}

/**
 * @param int $resultTypeGroup
 */
public function setResultTypeGroup(int $resultTypeGroup): self
{
    $this->resultTypeGroup = $resultTypeGroup;

    return $this;
}
```

```
    /**
     * @return int
     */
    public function getMeasurementType(): int
    {
        return $this->measurementType;
    }

    /**
     * @param int $measurementType
     */
    public function setMeasurementType(int $measurementType): self
    {
        $this->measurementType = $measurementType;

        return $this;
    }

    /**
     * @return float
     */
    public function getPrice(): float
    {
        return $this->price;
    }

    /**
     * @param float $price
     */
    public function setPrice(float $price): self
    {
        $this->price = $price;

        return $this;
    }
}
```

Клас ButtonDto

```
<?php

namespace App\Dto\Telegram;

class ButtonDto
{
    /**
     * @var int
     */
    protected $id;

    /**
     * @var string
     */
    protected $text;

    /**
     * @var int|null
     */
    protected $resultTypeGroup;
```

```
/**
 * @var int|null
 */
protected $resultType;

/**
 * @return int
 */
public function getId(): int
{
    return $this->id;
}

/**
 * @param int $id
 */
public function setId(int $id): self
{
    $this->id = $id;

    return $this;
}

/**
 * @return string
 */
public function getText(): string
{
    return $this->text;
}

/**
 * @param string $text
 *
 * @return $this
 */
public function setText(string $text): self
{
    $this->text = $text;

    return $this;
}

/**
 * @return int|null
 */
public function getResultType(): ?int
{
    return $this->resultType;
}

/**
 * @param int|null $resultType
 *
 * @return $this
 */
public function setResultType(?int $resultType): self
{
    $this->resultType = $resultType;

    return $this;
}
```

```
    /**
     * @return int
     */
    public function getResultTypeGroup(): ?int
    {
        return $this->resultTypeGroup;
    }

    /**
     * @param int|null $resultTypeGroup
     */
    public function setResultTypeGroup(?int $resultTypeGroup): self
    {
        $this->resultTypeGroup = $resultTypeGroup;

        return $this;
    }
}
```

Клас RawMessageDTO

```
<?php

namespace App\Dto\Telegram;

class RawMessageDTO
{
    public string $data = '';

    public function __construct(string $data)
    {
        $this->data = $data;
    }
}
```

Клас Employee

```
<?php

namespace App\Entity\Employee;

use App\Repository\Employee\EmployeeRepository;
use Doctrine\ORM\Mapping as ORM;
use Ramsey\Uuid\Uuid;

/**
 * @ORM\Entity(repositoryClass=EmployeeRepository::class)
 */
class Employee
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
```

```
private $id;

/**
 * @ORM\Column(type="string", length=255)
 */
private $name;

/**
 * @ORM\Column(type="guid", unique=true)
 */
private $regToken;

/**
 * @ORM\Column(type="boolean", options={"default": false})
 */
private $isRegistered = false;

/**
 * @ORM\Column(type="integer")
 */
private $approvalKey;

/**
 * @ORM\Column(type="datetime", nullable=true)
 */
private $registeredAt;

public function __construct()
{
    $this->regToken = Uuid::uuid4()->toString();
    $this->approvalKey = random_int(111111, 999999);
}

public function getId(): ?int
{
    return $this->id;
}

public function getName(): ?string
{
    return $this->name;
}

public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

public function getRegToken(): ?string
{
    return $this->regToken;
}

public function setRegToken(string $regToken): self
{
    $this->regToken = $regToken;

    return $this;
}

public function getIsRegistered(): ?bool
{
```

```
        return $this->isRegistered;
    }

    public function setIsRegistered(bool $isRegistered): self
    {
        $this->isRegistered = $isRegistered;

        return $this;
    }

    public function getRegisteredAt(): ?\DateTimeInterface
    {
        return $this->registeredAt;
    }

    public function setRegisteredAt(?\DateTimeInterface $registeredAt): self
    {
        $this->registeredAt = $registeredAt;

        return $this;
    }

    /**
     * @return int
     */
    public function getApprovalKey(): int
    {
        return $this->approvalKey;
    }
}
```

Клас MeasurementType

```
<?php

namespace App\Entity\Result;

use App\Repository\Result\MeasurementTypeRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=MeasurementTypeRepository::class)
 */
class MeasurementType
{
    public const TYPE_UNIT = 1;
    public const TYPE_METER = 2;
    public const TYPE_KILOGRAM = 3;
    public const TYPE_HOUR = 4;

    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50)
```

```
     */
    private $name;

    /**
     * @ORM\Column(type="string", length=20, nullable=true)
     */
    private $contraction;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getName(): ?string
    {
        return $this->name;
    }

    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }

    public function getContraction(): ?string
    {
        return $this->contraction;
    }

    public function setContraction(?string $contraction): self
    {
        $this->contraction = $contraction;

        return $this;
    }
}
```

Клас Result

```
<?php

namespace App\Entity\Result;

use App\Entity\Telegram\Chat;
use App\Repository\Result\ResultRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ResultRepository::class)
 */
class Result
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;
```

```
/**
 * @ORM\ManyToOne(targetEntity=Chat::class)
 * @ORM\JoinColumn(nullable=false)
 */
private $chat;

/**
 * @var \App\Entity\Result\ResultType
 *
 * @ORM\ManyToOne(targetEntity=ResultType::class)
 * @ORM\JoinColumn(nullable=false)
 */
private $resultType;

/**
 * @ORM\Column(type="integer")
 */
private $amount = 0;

/**
 * @ORM\Column(type="integer")
 */
private $actualPrice;

/**
 * @ORM\Column(type="integer")
 */
private $sum = 0;

/**
 * @ORM\Column(type="smallint")
 */
private $year;

/**
 * @ORM\Column(type="smallint")
 */
private $month;

/**
 * @ORM\Column(type="smallint")
 */
private $day;

/**
 * @ORM\Column(type="datetime")
 */
private $reportedDate;

/**
 * @ORM\Column(type="datetime")
 */
private $createdAt;

/**
 * @ORM\Column(type="datetime", nullable=true)
 */
private $updatedAt;

public function getId(): ?int
{
    return $this->id;
}
```

```
public function getChat(): ?Chat
{
    return $this->chat;
}

public function setChat(?Chat $chat): self
{
    $this->chat = $chat;

    return $this;
}

public function getAmount(): int
{
    return $this->amount;
}

public function setAmount(int $amount): self
{
    $this->amount = $amount;

    return $this;
}

public function getActualPrice(): ?int
{
    return $this->actualPrice;
}

public function setActualPrice(int $actualPrice): self
{
    $this->actualPrice = $actualPrice;

    return $this;
}

public function getSum(): ?int
{
    return $this->sum;
}

public function setSum(int $sum): self
{
    $this->sum = $sum;

    return $this;
}

public function getYear(): ?int
{
    return $this->year;
}

public function setYear(int $year): self
{
    $this->year = $year;

    return $this;
}

public function getMonth(): ?int
{
    return $this->month;
```

```
}

public function setMonth(int $month): self
{
    $this->month = $month;

    return $this;
}

public function getDay(): ?int
{
    return $this->day;
}

public function setDay(int $day): self
{
    $this->day = $day;

    return $this;
}

public function getCreatedAt(): ?\DateTimeInterface
{
    return $this->createdAt;
}

public function setCreatedAt(\DateTimeInterface $createdAt): self
{
    $this->createdAt = $createdAt;

    return $this;
}

/**
 * @return \App\Entity\Result\ResultType
 */
public function getResultType(): ResultType
{
    return $this->resultType;
}

/**
 * @param \App\Entity\Result\ResultType $resultType
 *
 * @return $this
 */
public function setResultType(ResultType $resultType): self
{
    $this->resultType = $resultType;

    return $this;
}

/**
 * @return mixed
 */
public function getUpdatedAt()
{
    return $this->updatedAt;
}

/**
 * @param mixed $updatedAt
 *
```

```
     * @return $this
     */
    public function setUpdatedAt($updatedAt): self
    {
        $this->updatedAt = $updatedAt;

        return $this;
    }

    /**
     * @return \DateTime
     */
    public function getReportedDate()
    {
        return $this->reportedDate;
    }

    /**
     * @param \DateTime $reportedDate
     *
     * @return $this
     */
    public function setReportedDate(\DateTime $reportedDate): self
    {
        $this->reportedDate = $reportedDate;

        return $this;
    }
}
```

Клас ResultType

```
<?php

namespace App\Entity\Result;

use App\Repository\Result\ResultTypeRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ResultTypeRepository::class)
 */
class ResultType
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;

    /**
     * @ORM\Column(type="integer")
     */
```

```
private $price;

/**
 * @ORM\ManyToOne(targetEntity=ResultTypeGroup::class)
 * @ORM\JoinColumn(nullable=false)
 */
private $resultTypeGroup;

/**
 * @ORM\ManyToOne(targetEntity=MeasurementType::class)
 * @ORM\JoinColumn(nullable=false)
 */
private $measurementType;

public function getId(): ?int
{
    return $this->id;
}

public function getName(): ?string
{
    return $this->name;
}

public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

public function getPrice(): ?int
{
    return $this->price;
}

public function setPrice(int $price): self
{
    $this->price = $price;

    return $this;
}

public function getResultTypeGroup(): ?ResultTypeGroup
{
    return $this->resultTypeGroup;
}

public function setResultTypeGroup(?ResultTypeGroup $resultTypeGroup): self
{
    $this->resultTypeGroup = $resultTypeGroup;

    return $this;
}

public function getMeasurementType(): ?MeasurementType
{
    return $this->measurementType;
}

public function setMeasurementType(?MeasurementType $measurementType): self
{
    $this->measurementType = $measurementType;
```

```
        return $this;
    }
}
```

Клас ResultTypeGroup

```
<?php

namespace App\Entity\Result;

use App\Repository\Result\ResultTypeGroupRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ResultTypeGroupRepository::class)
 */
class ResultTypeGroup
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $description;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getName(): ?string
    {
        return $this->name;
    }

    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }

    public function getDescription(): ?string
    {
        return $this->description;
    }

    public function setDescription(?string $description): self
    {
```

```
        $this->description = $description;

        return $this;
    }
}
```

Клас Button

```
<?php

namespace App\Entity\Telegram;

use App\Entity\Result\ResultType;
use App\Entity\Result\ResultTypeGroup;
use App\Repository\Telegram\ButtonRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ButtonRepository::class)
 */
class Button
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $text;

    /**
     * @ORM\ManyToOne(targetEntity=ResultType::class)
     */
    private $resultType;

    /**
     * @ORM\ManyToOne(targetEntity=ResultTypeGroup::class)
     */
    private $resultTypeGroup;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getText(): ?string
    {
        return $this->text;
    }

    public function setText(string $text): self
    {
        $this->text = $text;

        return $this;
```

```
    }

    public function getResultType(): ?ResultType
    {
        return $this->resultType;
    }

    public function setResultType(?ResultType $resultType): self
    {
        $this->resultType = $resultType;

        return $this;
    }

    public function getResultTypeGroup(): ?ResultTypeGroup
    {
        return $this->resultTypeGroup;
    }

    public function setResultTypeGroup(?ResultTypeGroup $resultTypeGroup): self
    {
        $this->resultTypeGroup = $resultTypeGroup;

        return $this;
    }
}
```

Клас Chat

```
<?php

namespace App\Entity\Telegram;

use App\Entity\Employee\Employee;
use App\Repository\Telegram\ChatRepository;
use DateTime;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ChatRepository::class)
 */
class Chat
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="bigint")
     */
    private $chatId;

    /**
     * @ORM\Column(type="bigint")
     */
    private $userId;
```

```
    /**
     * @ORM\Column(type="string", length=255)
     */
    private $firstname;

    /**
     * @ORM\ManyToOne(targetEntity=Employee::class)
     * @ORM\JoinColumn(nullable=false)
     */
    private $employee;

    /**
     * @ORM\Column(type="boolean", options={"default": false})
     */
    private $isRegistered = false;

    /**
     * @var string|null
     *
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $expectedAction;

    /**
     * @var int|null
     *
     * @ORM\Column(type="integer", nullable=true)
     */
    private $expectedResultTypeId;

    /**
     * @var \DateTime|null
     *
     * @ORM\Column(type="datetime", nullable=true)
     */
    private $reportedData;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getChatId(): ?string
    {
        return $this->chatId;
    }

    public function setChatId(string $chatId): self
    {
        $this->chatId = $chatId;

        return $this;
    }

    public function getUserId(): ?string
    {
        return $this->userId;
    }

    public function setUserId(string $userId): self
    {
        $this->userId = $userId;
```

```
    return $this;
}

public function getFirstname(): ?string
{
    return $this->firstname;
}

public function setFirstname(string $firstname): self
{
    $this->firstname = $firstname;

    return $this;
}

public function getEmployee(): ?Employee
{
    return $this->employee;
}

public function setEmployee(?Employee $employee): self
{
    $this->employee = $employee;

    return $this;
}

public function getIsRegistered(): bool
{
    return $this->isRegistered;
}

public function setIsRegistered(bool $isRegistered): self
{
    $this->isRegistered = $isRegistered;

    return $this;
}

/**
 * @return string|null
 */
public function getExpectedAction(): ?string
{
    return $this->expectedAction;
}

/**
 * @param string|null $expectedAction
 *
 * @return $this
 */
public function setExpectedAction(?string $expectedAction): self
{
    $this->expectedAction = $expectedAction;

    return $this;
}

/**
 * @return \DateTime|null
 */
public function getReportedData(): ?DateTime
{
```

```
        return $this->reportedData;
    }

    /**
     * @param \DateTime|null $reportedData
     */
    public function setReportedData(?DateTime $reportedData): self
    {
        $this->reportedData = $reportedData;

        return $this;
    }

    /**
     * @return int|null
     */
    public function getExpectedResultTypeId(): ?int
    {
        return $this->expectedResultTypeId;
    }

    /**
     * @param int|null $expectedResultTypeId
     */
    public function setExpectedResultTypeId(?int $expectedResultTypeId): self
    {
        $this->expectedResultTypeId = $expectedResultTypeId;

        return $this;
    }
}
```

Клас RequestListener

```
<?php

namespace App\Event\Listener\Request;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpKernel\Event\RequestEvent;

class RequestListener
{
    /**
     * @var string
     */
    private const CONTENT_TYPE_JSON = 'json';

    /**
     * @param \Symfony\Component\HttpKernel\Event\RequestEvent $event
     *
     * @return void
     */
    public function onKernelRequest(RequestEvent $event)
    {
        if (!$event->isMainRequest()) {
            return;
        }
```

```
        $request = $event->getRequest();

        if (empty($request->getContent())) {
            return;
        }

        if (!is_string($request->getContentType()) || $request->getContentType() !==
self::CONTENT_TYPE_JSON) {
            return;
        }


        if (!$this->transformJsonBody($request)) {
            $response = new Response('Unable to parse request.', 400);
            $event->setResponse($response);
        }
    }

    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return bool
     */
    private function transformJsonBody(Request $request)
    {
        $data = json_decode($request->getContent(), true);

        if (json_last_error() !== JSON_ERROR_NONE) {
            return false;
        }

        if ($data === null) {
            return true;
        }

        $request->request->replace($data);

        return true;
    }
}
```

Клас EmployeeRegisteredException

```
<?php

namespace App\Exception;

use Exception;

class EmployeeRegisteredException extends Exception
{



}
```

Клас EmployeeModel

```
<?php

namespace App\Form\Model\Employee;

use Symfony\Component\Validator\Constraints as Assert;

class EmployeeModel
{
    /**
     * @Assert\NotBlank(allowNull=false)
     * @Assert\Length(min=5, max=255)
     *
     * @var string|null
     */
    protected $name;

    /**
     * @return string|null
     */
    public function getName(): ?string
    {
        return $this->name;
    }

    /**
     * @param string|null $name
     *
     * @return $this
     */
    public function setName(?string $name): self
    {
        $this->name = $name;

        return $this;
    }
}
```

Клас ResultTypeGroupModel

```
<?php

namespace App\Form\Model\Result;

use Symfony\Component\Validator\Constraints as Assert;

class ResultTypeGroupModel
{
    /**
     * @Assert\NotBlank(allowNull=false)
     * @Assert\Length(min=5, max=255)
     *
     * @var string|null
     */
    protected $name;
```

```
    /**
     * @Assert\NotBlank(allowNull=true)
     * @Assert\Length(min=5, max=255)
     *
     * @var string|null
     */
    protected $description;

    /**
     * @return string|null
     */
    public function getName(): ?string
    {
        return $this->name;
    }

    /**
     * @param string|null $name
     *
     * @return $this
     */
    public function setName(?string $name): self
    {
        $this->name = $name;

        return $this;
    }

    /**
     * @return string|null
     */
    public function getDescription(): ?string
    {
        return $this->description;
    }

    /**
     * @param string|null $description
     *
     * @return $this
     */
    public function setDescription(?string $description): self
    {
        $this->description = $description;

        return $this;
    }
}
```

Клас ResultTypeModel

```
<?php

namespace App\Form\Model\Result;

use App\Entity\Result\MeasurementType;
use App\Entity\Result\ResultTypeGroup;
use Symfony\Component\Validator\Constraints as Assert;
```

```
class ResultTypeModel
{
    /**
     * @Assert\NotBlank(allowNull=false)
     * @Assert\Length(min=5, max=255)
     *
     * @var string|null
     */
    protected $name;

    /**
     * @Assert\NotBlank(allowNull=false)
     *
     * @var float|null
     */
    protected $price;

    /**
     * @Assert\NotBlank(allowNull=false)
     *
     * @var \App\Entity\Result\MeasurementType|null
     */
    protected $measurementType;

    /**
     * @Assert\NotBlank(allowNull=false)
     *
     * @var \App\Entity\Result\ResultTypeGroup|null
     */
    protected $resultTypeGroup;

    /**
     * @return string|null
     */
    public function getName(): ?string
    {
        return $this->name;
    }

    /**
     * @param string|null $name
     *
     * @return $this
     */
    public function setName(?string $name): self
    {
        $this->name = $name;

        return $this;
    }

    /**
     * @return float|null
     */
    public function getPrice(): ?float
    {
        return $this->price;
    }

    /**
     * @param float|null $price
     *
     * @return $this
     */
```

```
    public function setPrice(?float $price): self
    {
        $this->price = $price;

        return $this;
    }

    /**
     * @return \App\Entity\Result\MeasurementType|null
     */
    public function getMeasurementType(): ?MeasurementType
    {
        return $this->measurementType;
    }

    /**
     * @param \App\Entity\Result\MeasurementType|null $measurementType
     *
     * @return $this
     */
    public function setMeasurementType(?MeasurementType $measurementType): self
    {
        $this->measurementType = $measurementType;

        return $this;
    }

    /**
     * @return \App\Entity\Result\ResultTypeGroup|null
     */
    public function getResultTypeGroup(): ?ResultTypeGroup
    {
        return $this->resultTypeGroup;
    }

    /**
     * @param \App\Entity\Result\ResultTypeGroup|null $resultTypeGroup
     *
     * @return $this
     */
    public function setResultTypeGroup(?ResultTypeGroup $resultTypeGroup): self
    {
        $this->resultTypeGroup = $resultTypeGroup;

        return $this;
    }
}
```

Клас ButtonModel

```
<?php

namespace App\Form\Model\Telegram;

use App\Entity\Result\ResultType;
use App\Entity\Result\ResultTypeGroup;
use Symfony\Component\Validator\Constraints as Assert;

class ButtonModel
```

```
{
    /**
     * @Assert\NotBlank(allowNull=false)
     * @Assert\Length(min=5, max=50)
     *
     * @var string|null
     */
    protected $text;

    /**
     * @Assert\NotBlank(allowNull=true)
     *
     * @var \App\Entity\Result\ResultTypeGroup|null
     */
    protected $resultTypeGroup;

    /**
     * @Assert\NotBlank(allowNull=true)
     *
     * @var \App\Entity\Result\ResultType|null
     */
    protected $resultType;

    /**
     * @return string|null
     */
    public function getText(): ?string
    {
        return $this->text;
    }

    /**
     * @param string|null $text
     *
     * @return $this
     */
    public function setText(?string $text): self
    {
        $this->text = $text;

        return $this;
    }

    /**
     * @return \App\Entity\Result\ResultTypeGroup|null
     */
    public function getResultTypeGroup(): ?ResultTypeGroup
    {
        return $this->resultTypeGroup;
    }

    /**
     * @param \App\Entity\Result\ResultTypeGroup|null $resultTypeGroup
     *
     * @return $this
     */
    public function setResultTypeGroup(?ResultTypeGroup $resultTypeGroup): self
    {
        $this->resultTypeGroup = $resultTypeGroup;

        return $this;
    }

    /**
```

```
     * @return \App\Entity\Result\ResultType|null
     */
    public function getResultType(): ?ResultType
    {
        return $this->resultType;
    }

    /**
     * @param \App\Entity\Result\ResultType|null $resultType
     *
     * @return $this
     */
    public function setResultType(?ResultType $resultType): self
    {
        $this->resultType = $resultType;

        return $this;
    }
}
```

Клас EmployeeType

```
<?php

namespace App\Form\Type\Employee;

use App\Form\Model\Employee\EmployeeModel;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class EmployeeType extends AbstractType
{
    /**
     * @param \Symfony\Component\Form\FormBuilderInterface $builder
     * @param array $options
     *
     * @return void
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name');
    }

    /**
     * @param \Symfony\Component\OptionsResolver\OptionsResolver $resolver
     *
     * @return void
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => EmployeeModel::class,
        ]);
    }
}
```

Клас ResultTypeGroupType

```
<?php

namespace App\Form\Type\Result;

use App\Form\Model\Result\ResultTypeGroupModel;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ResultTypeGroupType extends AbstractType
{
    /**
     * @param \Symfony\Component\Form\FormBuilderInterface $builder
     * @param array $options
     *
     * @return void
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name')
            ->add('description');
    }

    /**
     * @param \Symfony\Component\OptionsResolver\OptionsResolver $resolver
     *
     * @return void
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => ResultTypeGroupModel::class,
        ]);
    }
}
```

Клас ResultTypeType

```
<?php

namespace App\Form\Type\Result;

use App\Entity\Result\MeasurementType;
use App\Entity\Result\ResultTypeGroup;
use App\Form\Model\Result\ResultTypeModel;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ResultTypeType extends AbstractType
{
    /**
```

```
     * @param \Symfony\Component\Form\FormBuilderInterface $builder
     * @param array $options
     *
     * @return void
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name', TextType::class)
            ->add('price', NumberType::class)
            ->add(
                'measurementType',
                EntityType::class,
                [
                    'class' => MeasurementType::class,
                ]
            )
            ->add(
                'resultTypeGroup',
                EntityType::class,
                [
                    'class' => ResultTypeGroup::class,
                ]
            )
        ;
    }

    /**
     * @param \Symfony\Component\OptionsResolver\OptionsResolver $resolver
     *
     * @return void
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => ResultTypeModel::class,
        ]);
    }
}
```

Клас ButtonType

```
<?php

namespace App\Form\Type\Telegram;

use App\Entity\Result\ResultType;
use App\Entity\Result\ResultTypeGroup;
use App\Form\Model\Telegram\ButtonModel;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ButtonType extends AbstractType
{
    /**
     * @param \Symfony\Component\Form\FormBuilderInterface $builder
     * @param array $options
```

```
     *
     * @return void
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('text', TextType::class)
            ->add(
                'resultType',
                EntityType::class,
                [
                    'class' => ResultType::class,
                ]
            )
            ->add(
                'resultTypeGroup',
                EntityType::class,
                [
                    'class' => ResultTypeGroup::class,
                ]
            );
    }

    /**
     * @param \Symfony\Component\OptionsResolver\OptionsResolver $resolver
     *
     * @return void
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => ButtonModel::class,
        ]);
    }
}
```

Клас EmployeeDtoMapper

```
<?php

namespace App\Mapper\Request;

use App\Dto\Request\EmployeeDto;
use App\Entity\Employee\Employee;

class EmployeeDtoMapper
{
    /**
     * @var string
     */
    protected $botName;

    /**
     * @param string $botName
     */
    public function __construct(string $botName)
    {
        $this->botName = $botName;
    }
```

```
    /**
     * @param \App\Entity\Employee\Employee $employee
     *
     * @return \App\Dto\Request\EmployeeDto
     */
    public function mapEmployeeToEmployeeDto(Employee $employee): EmployeeDto
    {
        $employeeDto = new EmployeeDto();

        return $employeeDto
            ->setId($employee->getId())
            ->setName($employee->getName())
            ->setRegisteredAt($employee->getIsRegistered() ? $employee-
>getRegisteredAt()->format('d/m/Y') : null)
            ->setApprovalKey($employee->getIsRegistered() ? null : $employee-
>getApprovalKey())
            ->setRegLink($employee->getIsRegistered() ? null : $this-
>buildRegLink($employee->getRegToken()));
    }

    /**
     * @param string $regToken
     *
     * @return string
     */
    protected function buildRegLink(string $regToken): string
    {
        return sprintf(
            'https://t.me/%s?start=%s',
            $this->botName,
            $regToken,
        );
    }
}
```

Клас EmployeeRepository

```
<?php

namespace App\Repository\Employee;

use App\Entity\Employee\Employee;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Employee>
 *
 * @method Employee|null find($id, $lockMode = null, $lockVersion = null)
 * @method Employee|null findOneBy(array $criteria, array $orderBy = null)
 * @method Employee[]    findAll()
 * @method Employee[]    findBy(array $criteria, array $orderBy = null, $limit =
null, $offset = null)
 */
class EmployeeRepository extends ServiceEntityRepository
{
```

```
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Employee::class);
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function add(Employee $entity, bool $flush = true): void
    {
        $this->_em->persist($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function remove(Employee $entity, bool $flush = true): void
    {
        $this->_em->remove($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    public function getIndexedEmployees(): array
    {
        return $this->createQueryBuilder('r')
            ->indexBy('r', 'r.id')
            ->getQuery()
            ->getResult();
    }
}
```

Клас MeasurementTypeRepository

```
<?php

namespace App\Repository\Result;

use App\Entity\Result\MeasurementType;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<MeasurementType>
 *
 * @method MeasurementType|null find($id, $lockMode = null, $lockVersion = null)
 * @method MeasurementType|null findOneBy(array $criteria, array $orderBy = null)
 * @method MeasurementType[]    findAll()
 * @method MeasurementType[]    findBy(array $criteria, array $orderBy = null, $limit
= null, $offset = null)
```

```
 */
class MeasurementTypeRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, MeasurementType::class);
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function add(MeasurementType $entity, bool $flush = true): void
    {
        $this->_em->persist($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function remove(MeasurementType $entity, bool $flush = true): void
    {
        $this->_em->remove($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    // /**
    //  * @return MeasurementType[] Returns an array of MeasurementType objects
    //  */
    /*
    public function findByExampleField($value)
    {
        return $this->createQueryBuilder('m')
            ->andWhere('m.exampleField = :val')
            ->setParameter('val', $value)
            ->orderBy('m.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult()
        ;
    }
    */

    /*
    public function findOneBySomeField($value): ?MeasurementType
    {
        return $this->createQueryBuilder('m')
            ->andWhere('m.exampleField = :val')
            ->setParameter('val', $value)
            ->getQuery()
            ->getOneOrNullResult()
        ;
    }
    */
}
```

Клас ResultRepository

```
<?php

namespace App\Repository\Result;

use App\Entity\Result\Result;
use App\Entity\Telegram\Chat;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Result>
 *
 * @method Result|null find($id, $lockMode = null, $lockVersion = null)
 * @method Result|null findOneBy(array $criteria, array $orderBy = null)
 * @method Result[]    findAll()
 * @method Result[]    findBy(array $criteria, array $orderBy = null, $limit = null,
$offset = null)
 */
class ResultRepository extends ServiceEntityRepository
{
    /**
     * @var
\App\Repository\Telegram\ChatRepository|\Doctrine\Persistence\ObjectRepository
     */
    private $chatRepository;

    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Result::class);

        $this->chatRepository = $registry->getRepository(Chat::class);
    }

    /**
     * @param \DateTime $startDateTime
     * @param \DateTime $endDateTime
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return \App\Entity\Result\Result[]
     */
    public function findForDate(
        \DateTime $startDateTime,
        \DateTime $endDateTime,
        Chat $chat
    ): array {
        return $this->createQueryBuilder('result')
            ->andWhere('result.chat = :chat')
            ->andWhere('result.reportedDate BETWEEN :startDate AND :endData')
            ->setParameter('chat', $chat)
            ->setParameter('startDate', $startDateTime)
            ->setParameter('endData', $endDateTime)
            ->getQuery()
            ->getResult();
    }

    public function findForResultReport(\DateTime $from, \DateTime $to, ?int
$employeeId): array
```

```
    {
        $chatIds = [];

        if ($employeeId) {
            $chats = $this->chatRepository->findBy(['employee' => $employeeId]);

            $chatIds = array_map(function (Chat $chat): int {
                return $chat->getId();
            }, $chats);
        }

        $query = $this->createQueryBuilder('result')
            ->select('IDENTITY(result.resultType) as resultTypeId,
result.reportedDate, SUM(result.amount) as amount, SUM(result.sum) as sum')
            ->andWhere('result.reportedDate BETWEEN :from AND :to')
            ->setParameter('from', $from)
            ->setParameter('to', $to)
            ->groupBy('result.reportedDate')
            ->addGroupBy('result.resultType');

        if ($chatIds) {
            $query->andWhere('result.chat in (:chatIds)')
                ->setParameter('chatIds', $chatIds);
        }

        return $query->getQuery()
            ->getArrayResult();
    }

    public function findForSalaryReport(\DateTime $from, \DateTime $to, ?int
$employeeId): array
    {
        $chatIds = [];

        if ($employeeId) {
            $chats = $this->chatRepository->findBy(['employee' => $employeeId]);

            $chatIds = array_map(function (Chat $chat): int {
                return $chat->getId();
            }, $chats);
        }

        $query = $this->createQueryBuilder('result')
            ->select('IDENTITY(result.resultType) as resultTypeId,
IDENTITY(chat.employee) as employeeId, SUM(result.amount) as amount, SUM(result.sum)
as sum')
            ->innerJoin('result.chat', 'chat')
            ->andWhere('result.reportedDate BETWEEN :from AND :to')
            ->setParameter('from', $from)
            ->setParameter('to', $to)
            ->groupBy('chat.employee')
            ->addGroupBy('result.resultType');

        if ($chatIds) {
            $query->andWhere('result.chat in (:chatIds)')
                ->setParameter('chatIds', $chatIds);
        }

        return $query->getQuery()
            ->getArrayResult();
    }
}
```

Клас ResultTypeGroupRepository

```
<?php

namespace App\Repository\Result;

use App\Entity\Result\ResultTypeGroup;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<ResultTypeGroup>
 *
 * @method ResultTypeGroup|null find($id, $lockMode = null, $lockVersion = null)
 * @method ResultTypeGroup|null findOneBy(array $criteria, array $orderBy = null)
 * @method ResultTypeGroup[]    findAll()
 * @method ResultTypeGroup[]    findBy(array $criteria, array $orderBy = null, $limit
= null, $offset = null)
 */
class ResultTypeGroupRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, ResultTypeGroup::class);
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function add(ResultTypeGroup $entity, bool $flush = true): void
    {
        $this->_em->persist($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function remove(ResultTypeGroup $entity, bool $flush = true): void
    {
        $this->_em->remove($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    public function getIndexedResultTypeGroups(): array
    {
        return $this->createQueryBuilder('r')
            ->indexBy('r', 'r.id')
            ->getQuery()
            ->getResult();
    }
}
```

Клас ResultTypeRepository

```
<?php

namespace App\Repository\Result;

use App\Entity\Result\ResultType;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<ResultType>
 *
 * @method ResultType|null find($id, $lockMode = null, $lockVersion = null)
 * @method ResultType|null findOneBy(array $criteria, array $orderBy = null)
 * @method ResultType[]    findAll()
 * @method ResultType[]    findBy(array $criteria, array $orderBy = null, $limit =
null, $offset = null)
 */
class ResultTypeRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, ResultType::class);
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function add(ResultType $entity, bool $flush = true): void
    {
        $this->_em->persist($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function remove(ResultType $entity, bool $flush = true): void
    {
        $this->_em->remove($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    public function getIndexedResultTypes(): array
    {
        return $this->createQueryBuilder('r')
            ->indexBy('r', 'r.id')
            ->getQuery()
            ->getResult();
    }
}
```

Клас ButtonRepository

```
<?php

namespace App\Repository\Telegram;

use App\Entity\Result\ResultType;
use App\Entity\Result\ResultTypeGroup;
use App\Entity\Telegram\Button;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\ORM\Query\Expr\Join;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Button>
 *
 * @method Button|null find($id, $lockMode = null, $lockVersion = null)
 * @method Button|null findOneBy(array $criteria, array $orderBy = null)
 * @method Button[]    findAll()
 * @method Button[]    findBy(array $criteria, array $orderBy = null, $limit = null,
$offset = null)
 */
class ButtonRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Button::class);
    }

    /**
     * @return \App\Entity\Telegram\Button[]
     */
    public function getGroupsButtons(): array
    {
        return $this->createQueryBuilder('b')
            ->andWhere('b.resultTypeGroup IS NOT null')
            ->orderBy('b.id', 'ASC')
            ->getQuery()
            ->getResult();
    }

    /**
     * @return \App\Entity\Telegram\Button[]
     */
    public function getGroupButtons(ResultTypeGroup $resultTypeGroup): array
    {
        return $this->createQueryBuilder('b')
            ->join(
                ResultType::class,
                'rt',
                Join::WITH,
                'b.resultType = rt.id'
            )
            ->andWhere('rt.resultTypeGroup = :resultTypeGroup')
            ->setParameter('resultTypeGroup', $resultTypeGroup)
            ->orderBy('b.id', 'ASC')
            ->getQuery()
            ->getResult();
    }
```

```
    /**
     * @return \App\Entity\Telegram\Button[]
     */
    public function getTypeButtons(ResultTypeGroup $resultTypeGroup): array
    {
        return $this->createQueryBuilder('b')
            ->join(
                ResultType::class,
                'rt',
                Join::WITH,
                'b.resultType = rtg.id'
            )
            ->andWhere('rt.resultTypeGroup = :resultTypeGroup')
            ->setParameter('resultTypeGroup', $resultTypeGroup)
            ->orderBy('b.id', 'ASC')
            ->getQuery()
            ->getResult();
    }
}
```

Клас ChatRepository

```
<?php

namespace App\Repository\Telegram;

use App\Entity\Telegram\Chat;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Chat>
 *
 * @method Chat|null find($id, $lockMode = null, $lockVersion = null)
 * @method Chat|null findOneBy(array $criteria, array $orderBy = null)
 * @method Chat[]    findAll()
 * @method Chat[]    findBy(array $criteria, array $orderBy = null, $limit = null,
$offset = null)
 */
class ChatRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Chat::class);
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function add(Chat $entity, bool $flush = true): void
    {
        $this->_em->persist($entity);
        if ($flush) {
            $this->_em->flush();
        }
```

```
    }

    /**
     * @throws ORMException
     * @throws OptimisticLockException
     */
    public function remove(Chat $entity, bool $flush = true): void
    {
        $this->_em->remove($entity);
        if ($flush) {
            $this->_em->flush();
        }
    }

    // /**
    //  * @return Chat[] Returns an array of Chat objects
    //  */
    /*
    public function findByExampleField($value)
    {
        return $this->createQueryBuilder('c')
            ->andWhere('c.exampleField = :val')
            ->setParameter('val', $value)
            ->orderBy('c.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult()
        ;
    }
    */

    /*
    public function findOneBySomeField($value): ?Chat
    {
        return $this->createQueryBuilder('c')
            ->andWhere('c.exampleField = :val')
            ->setParameter('val', $value)
            ->getQuery()
            ->getOneOrNullResult()
        ;
    }
    */
}
```

Клас EmployeeService

```
<?php

namespace App\Service\Employee;

use App\Entity\Employee\Employee;
use App\Exception\EmployeeRegisteredException;
use App\Form\Model\Employee\EmployeeModel;
use App\Mapper\Request\EmployeeDtoMapper;
use App\Repository\Employee\EmployeeRepository;
use Doctrine\ORM\EntityManagerInterface;

class EmployeeService
{
```

```
    /**
     * @var \App\Repository\Employee\EmployeeRepository
     */
    protected $employeeRepository;

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected $entityManager;

    /**
     * @var \App\Mapper\Request\EmployeeDtoMapper
     */
    protected $employeeDtoMapper;

    /**
     * @param \App\Repository\Employee\EmployeeRepository $employeeRepository
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     */
    public function __construct(
        EmployeeRepository $employeeRepository,
        EntityManagerInterface $entityManager,
        EmployeeDtoMapper $employeeDtoMapper
    ) {
        $this->employeeRepository = $employeeRepository;
        $this->entityManager = $entityManager;
        $this->employeeDtoMapper = $employeeDtoMapper;
    }

    /**
     * @param \App\Form\Model\Employee\EmployeeModel $employeeModel
     *
     * @return Employee
     */
    public function createEmployee(EmployeeModel $employeeModel): Employee
    {
        $employee = new Employee();

        $employee->setName($employeeModel->getName());

        $this->entityManager->persist($employee);
        $this->entityManager->flush();

        return $employee;
    }

    /**
     * @param \App\Entity\Employee\Employee $employee
     *
     * @return void
     * @throws \App\Exception\EmployeeRegisteredException
     */
    public function deleteEmployee(Employee $employee): void
    {
        if ($employee->getIsRegistered()) {
            throw new EmployeeRegisteredException('The employee is already
activated.');
        }

        $this->entityManager->remove($employee);
        $this->entityManager->flush();
    }

    /**
```

```
     * @return \App\Dto\Request\EmployeeDto[]
     */
    public function getEmployees(): array
    {
        $employeeDtos = [];
        $employees = $this->employeeRepository->findBy([]);

        foreach ($employees as $employee) {
            $employeeDtos[] = $this->employeeDtoMapper-
>mapEmployeeToEmployeeDto($employee);
        }

        return $employeeDtos;
    }
}
```

Клас ReportBuilder

```
<?php

namespace App\Service\Result\Report;

use App\Repository\Employee\EmployeeRepository;
use App\Repository\Result\ResultRepository;
use App\Repository\Result\ResultTypeGroupRepository;
use App\Repository\Result\ResultTypeRepository;
use Symfony\Component\HttpFoundation\Exception\BadRequestException;

class ReportBuilder
{
    /**
     * @var \App\Repository\Result\ResultRepository
     */
    private $resultRepository;

    /**
     * @var \App\Repository\Result\ResultTypeRepository
     */
    private $resultTypeRepository;

    /**
     * @var \App\Repository\Result\ResultTypeGroupRepository
     */
    private $resultTypeGroupRepository;

    /**
     * @var \App\Repository\Employee\EmployeeRepository
     */
    private $employeeRepository;

    /**
     * @param \App\Repository\Result\ResultRepository $resultRepository
     */
    public function __construct(
        ResultRepository $resultRepository,
        ResultTypeRepository $resultTypeRepository,
        ResultTypeGroupRepository $resultTypeGroupRepository,
        EmployeeRepository $employeeRepository
    ) {
```

```
        $this->resultRepository = $resultRepository;
        $this->resultTypeRepository = $resultTypeRepository;
        $this->resultTypeGroupRepository = $resultTypeGroupRepository;
        $this->employeeRepository = $employeeRepository;
    }

    /**
     * @param array $filters
     * @param string $type
     *
     * @return resource|void
     */
    public function generateReportFile(array $filters, string $type)
    {
        if (!in_array($type, $this->getAvailableTypes())) {
            throw new BadRequestException('Not valid type of report');
        }

        switch ($type):
            case ReportTypeConstants::RESULTS_REPORT:
                return $this->generateResultsReport($filters, $type);
            case ReportTypeConstants::SALARY_REPORT:
                return $this->generateSalaryReport($filters, $type);
        endswitch;
    }

    /**
     * @return resource
     */
    protected function generateResultsReport(array $filters)
    {
        $fromDate = (\DateTime::createFromFormat('d/m/Y',
$filters[ReportTypeConstants::QUERY_FILTER_FROM]))->setTime(0, 0);
        $toDate = (\DateTime::createFromFormat('d/m/Y',
$filters[ReportTypeConstants::QUERY_FILTER_TO]))->setTime(23, 59);
        $employeeId = isset($filters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID])
? (int)$filters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID] : null;

        $results = $this->resultRepository->findForResultReport($fromDate, $toDate,
$employeeId);

        $mappedResults = [];
        foreach ($results as $result) {

$mappedResults[$result['resultTypeId']]['results'][$result['reportedDate']-
>format('d/m/Y')] = $result['amount'];
            $mappedResults[$result['resultTypeId']]['amount'] =
                (isset($mappedResults[$result['resultTypeId']]['amount']) ?
$mappedResults[$result['resultTypeId']]['amount'] : 0) + $result['amount'];
            $mappedResults[$result['resultTypeId']]['sum'] =
                (isset($mappedResults[$result['resultTypeId']]['sum']) ?
$mappedResults[$result['resultTypeId']]['sum'] : 0) + $result['sum'];
        }

        $headers = [
            ['','','','','',],
            ['Тип роботи','Категорія робіт','Одиниця вимірювання','Загалом виконано
за період','До сплати за період',],
        ];

        $isLastDayReached = false;
        $currentDate = clone $fromDate;
        while (!$isLastDayReached) {
            $headers[0][] = $this->getDayOfWeek($currentDate->format('N'));
```

```
            $headers[1][] = $currentDate->format('d.m.Y');
            $currentDate->add(new \DateInterval('P1D'));

            $isLastDayReached = $currentDate->getTimestamp() > $toDate-
>getTimestamp();
        }

        $file = fopen('php://temp', 'w');
        fputcsv($file, $headers[0]);
        fputcsv($file, $headers[1]);

        $resultTypeMap = $this->resultTypeRepository->getIndexedResultTypes();

        $totalSum = 0;
        foreach ($mappedResults as $resultTypeId => $mappedResult) {
            $sum = $mappedResult['sum'] ? $mappedResult['sum'] / 10000 : 0;
            $totalSum += $sum;
            $row = [
                $resultTypeMap[$resultTypeId]->getName(),
                $resultTypeMap[$resultTypeId]->getResultTypeGroup()->getName(),
                $resultTypeMap[$resultTypeId]->getMeasurementType()->getName() . ' ('
. $resultTypeMap[$resultTypeId]->getMeasurementType()->getContraction() . ')',
                $mappedResult['amount'] ? $mappedResult['amount'] / 100 : 0,
                $sum,
            ];

            $isLastDayReached = false;
            $currentDate = clone $fromDate;
            while (!$isLastDayReached) {
                $amount = $mappedResult['results'][$currentDate->format('d/m/Y')] ??
0;
                $row[] = $amount ? $amount/100 : 0;

                $currentDate->add(new \DateInterval('P1D'));
                $isLastDayReached = $currentDate->getTimestamp() > $toDate-
>getTimestamp();
            }


            fputcsv($file, $row);
        }
        fputcsv($file, ['','','','Загалом:',$totalSum]);
        rewind($file);

        return $file;
    }

    /**
     * @param int $numberOfDay
     *
     * @return string
     */
    protected function getDayOfWeek(int $numberOfDay): string
    {
        $days = [
            1 => 'Понеділок',
            2 => 'Вівторок',
            3 => 'Середа',
            4 => 'Четвер',
            5 => 'Пятниця',
            6 => 'Субота',
            7 => 'Неділя',
        ];
```

```
        return $days[$numberOfDay];
    }

    /**
     * @return resource
     */
    protected function generateSalaryReport(array $filters, string $type)
    {
        $fromDate = (\DateTime::createFromFormat('d/m/Y',
$filters[ReportTypeConstants::QUERY_FILTER_FROM]))->setTime(0, 0);
        $toDate = (\DateTime::createFromFormat('d/m/Y',
$filters[ReportTypeConstants::QUERY_FILTER_TO]))->setTime(23, 59);
        $employeeId = isset($filters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID])
? (int)$filters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID] : null;

        $results = $this->resultRepository->findForSalaryReport($fromDate, $toDate,
$employeeId);
        $mapperResults = [];
        $totalSum = 0;

        foreach ($results as $result) {
            $sum = $result['sum'] / 10000;
            $mapperResults[$result['employeeId']]['results'][$result['resultTypeId']]
= [
                'sum' => $sum,
                'total' => $result['amount'] / 100,
            ];
            $mapperResults[$result['employeeId']]['totalSum'] =
($mapperResults[$result['employeeId']]['totalSum'] ?? 0) + $sum;
            $totalSum += $sum;
        }
        $resultTypeGroupMap = $this->mapTypesByResultTypeGroupId($this-
>resultTypeRepository->getIndexedResultTypes());
        $resultTypeGroupsMap = $this->resultTypeGroupRepository-
>getIndexedResultTypeGroups();
        $employeesMap = $this->employeeRepository->getIndexedEmployees();

        $headers = [
            ['',''],
            ['',''],
            ['','Загалом виконано',],
        ];
        foreach ($resultTypeGroupMap as $resultTypeGroupId => $resultTypesMap) {
            $headers[0][] = $resultTypeGroupsMap[$resultTypeGroupId]->getName();
            $isFirst = true;
            foreach ($resultTypesMap as $resultType) {
                if ($isFirst) {
                    $headers[0][] = '';
                } else {
                    $headers[0][] = '';
                    $headers[0][] = '';
                }

                $headers[1][] = $resultType->getName();
                $headers[1][] = '';
                $headers[2][] = 'Виконано';
                $headers[2][] = 'Зароблено';

                $isFirst = false;
            }
        }

        $file = fopen('php://temp', 'w');
        fputcsv($file, $headers[0]);
```

```
        fputcsv($file, $headers[1]);
        fputcsv($file, $headers[2]);


        foreach ($mapperResults as $employeeId => $employeeData) {
            $row = [
                $employeesMap[$employeeId]->getName(),
                $employeeData['totalSum'],
            ];

            foreach ($resultTypeGroupMap as $resultTypesMap) {
                foreach ($resultTypesMap as $resultType) {
                    if (!isset($employeeData['results'][$resultType->getId()])) {
                        $row[] = 0;
                        $row[] = 0;

                        continue;
                    }

                    $row[] = $employeeData['results'][$resultType->getId()]['total'];
                    $row[] = $employeeData['results'][$resultType->getId()]['sum'];
                }
            }

            fputcsv($file, $row);
        }
        fputcsv($file, ['Загальна сума', $totalSum]);
        rewind($file);

        return $file;
    }

    protected function getAvailableTypes(): array
    {
        return [
            ReportTypeConstants::RESULTS_REPORT,
            ReportTypeConstants::SALARY_REPORT,
        ];
    }

    /**
     * @param \App\Entity\Result\ResultType[] $resultTypeMap
     *
     * @return array
     */
    protected function mapTypesByResultTypeGroupId(array $resultTypeMap): array
    {
        $result = [];

        foreach ($resultTypeMap as $resultTypeId => $resultType) {
            $result[$resultType->getResultTypeGroup()->getId()][$resultTypeId] =
$resultType;
        }

        return $result;
    }
}
```

Клас ReportNameBuilder

```
<?php

namespace App\Service\Result\Report;

use App\Repository\Employee\EmployeeRepository;

class ReportNameBuilder
{
    public function generateNameByType(array $filters, string $type): string
    {
        $name = $type === ReportTypeConstants::RESULTS_REPORT ? 'RESULTS' :
'SALARIES';

        if (isset($filters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID])) {

            $name .= '-'. ($filters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID] ??
'?');
        }

        return sprintf(
            '%s-%s_%s-%s.csv',
            $name,
            str_replace('/', '.', $filters[ReportTypeConstants::QUERY_FILTER_FROM]),
            str_replace('/', '.', $filters[ReportTypeConstants::QUERY_FILTER_TO]),
            (new \DateTime())->format('d.m.Y_H:i')
        );
    }
}
```

Клас ReportQueryParamsValidator

```
<?php

namespace App\Service\Result\Report;

class ReportQueryParamsValidator
{
    public function validateQueryParametersPerReportType(string $reportType, array
$queryParameters): bool
    {
        if (!in_array($reportType, ReportTypeConstants::AVAILABLE_REPORT_TYPES)) {
            return false;
        }

        if (isset($queryParameters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID]) &&
$queryParameters[ReportTypeConstants::QUERY_FILTER_EMPLOYEE_ID] <= 0) {
            return false;
        }

        if (!isset($queryParameters[ReportTypeConstants::QUERY_FILTER_FROM]) ||
!isset($queryParameters[ReportTypeConstants::QUERY_FILTER_TO])) {
            return false;
        }
```

```
        $fromDate = (\DateTime::createFromFormat('d/m/Y',
$queryParameters[ReportTypeConstants::QUERY_FILTER_FROM]))->setTime(0, 0);
        $toDate = (\DateTime::createFromFormat('d/m/Y',
$queryParameters[ReportTypeConstants::QUERY_FILTER_TO]))->setTime(23, 59);

        return $toDate > $fromDate;
    }
}
```

Клас ReportTypeConstants

```
<?php

namespace App\Service\Result\Report;

interface ReportTypeConstants
{
    public const SALARY_REPORT = 'SALARY_REPORT';
    public const RESULTS_REPORT = 'RESULTS_REPORT';

    public const QUERY_FILTER_FROM = 'from';
    public const QUERY_FILTER_TO = 'to';
    public const QUERY_FILTER_EMPLOYEE_ID = 'employeeId';

    public const AVAILABLE_REPORT_TYPES = [
        self::SALARY_REPORT,
        self::RESULTS_REPORT,
    ];

    public const AVAILABLE_QUERY_FILTERS = [
        self::QUERY_FILTER_FROM,
        self::QUERY_FILTER_TO,
        self::QUERY_FILTER_EMPLOYEE_ID,
    ];
}
```

Клас ResultAmountValidator

```
<?php

namespace App\Service\Result;

use App\Entity\Result\MeasurementType;
use App\Entity\Result\ResultType;

class ResultAmountValidator
{
    protected const TYPE_FLOAT = 'float';
    protected const TYPE_INT = 'int';

    protected array $validationRules = [
        MeasurementType::TYPE_UNIT => self::TYPE_INT,
        MeasurementType::TYPE_METER => self::TYPE_FLOAT,
        MeasurementType::TYPE_KILOGRAM => self::TYPE_FLOAT,
        MeasurementType::TYPE_HOUR => self::TYPE_FLOAT,
```

```
    ];

    /**
     * @param string $value
     * @param \App\Entity\Result\ResultType $resultType
     *
     * @return bool
     */
    public function isValid(string $value, ResultType $resultType): bool
    {
        $matches =[];
        preg_match('/^-?\d+(\.\d{1,2})?$/', $value, $matches);

        if (!count($matches)) {
            return false;
        }

        $validationRule = $this->validationRules[$resultType->getMeasurementType()-
>getId()];

        if ($validationRule === static::TYPE_INT) {
            return count(explode('.', $value)) === 1;
        }

        return true;
    }
}
```

Клас ResultTypeFormatBuilder

```
<?php

namespace App\Service\Result;

use App\Entity\Result\MeasurementType;
use App\Entity\Result\ResultType;

class ResultTypeFormatBuilder
{
    protected const TYPE_FLOAT = 'float';
    protected const TYPE_INT = 'int';

    protected array $formatMapping = [
        MeasurementType::TYPE_UNIT => self::TYPE_INT,
        MeasurementType::TYPE_METER => self::TYPE_FLOAT,
        MeasurementType::TYPE_KILOGRAM => self::TYPE_FLOAT,
        MeasurementType::TYPE_HOUR => self::TYPE_FLOAT,
    ];

    public function getAvailableFormats(ResultType $resultType): string
    {
        $formatType = $this->formatMapping[$resultType->getMeasurementType()-
>getId()];

        if ($formatType === static::TYPE_FLOAT) {
            return "1) 1\n2) -7\n3) 3.2\n4) -4.33";
        }

        return "1) 1\n2) -7";
```

```
    }
}
```

Клас ResultTypeGroupService

```
<?php

namespace App\Service\Result;

use App\Entity\Result\ResultTypeGroup;
use App\Form\Model\Result\ResultTypeGroupModel;
use Doctrine\ORM\EntityManagerInterface;

class ResultTypeGroupService
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    private $entityManager;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     */
    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }

    /**
     * @param \App\Form\Model\Result\ResultTypeGroupModel $resultTypeGroupModel
     * @param \App\Entity\Result\ResultTypeGroup|null $resultTypeGroup
     *
     * @return \App\Entity\Result\ResultTypeGroup
     */
    public function saveResultTypeGroup(
        ResultTypeGroupModel $resultTypeGroupModel,
        ?ResultTypeGroup $resultTypeGroup = null
    ): ResultTypeGroup {
        if (!$resultTypeGroup) {
            $resultTypeGroup = new ResultTypeGroup();
            $this->entityManager->persist($resultTypeGroup);
        }

        $resultTypeGroup->setName($resultTypeGroupModel->getName())
            ->setDescription($resultTypeGroupModel->getDescription());

        $this->entityManager->flush();

        return $resultTypeGroup;
    }
}
```

Клас ResultTypeService

```
<?php
```

```
namespace App\Service\Result;

use App\Dto\Request\ResultTypeDto;
use App\Entity\Result\ResultType;
use App\Form\Model\Result\ResultTypeModel;
use App\Repository\Result\ResultRepository;
use App\Repository\Result\ResultTypeRepository;
use App\Repository\Telegram\ButtonRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Exception\BadRequestException;

class ResultTypeService
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    private $entityManager;
    /**
     * @var \App\Repository\Result\ResultTypeRepository
     */
    private $resultTypeRepository;
    /**
     * @var \App\Repository\Result\ResultRepository
     */
    private $resultRepository;
    /**
     * @var \App\Repository\Telegram\ButtonRepository
     */
    private $buttonRepository;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        ResultTypeRepository $resultTypeRepository,
        ResultRepository $resultRepository,
        ButtonRepository $buttonRepository
    ) {
        $this->entityManager = $entityManager;
        $this->resultTypeRepository = $resultTypeRepository;
        $this->resultRepository = $resultRepository;
        $this->buttonRepository = $buttonRepository;
    }

    public function saveResultType(
        ResultTypeModel $resultTypeModel,
        ?ResultType $resultType = null
    ): ResultType {
        if (!$resultType) {
            $resultType = new ResultType();
            $this->entityManager->persist($resultType);
        }

        $resultType = $this->mapModelToEntity($resultTypeModel, $resultType);

        $this->entityManager->flush();

        return $resultType;
    }

    public function deleteResultType(
        ResultType $resultType
```

```
    ): void {
        $result = $this->resultRepository->findOneBy(['resultType' => $resultType]);

        if ($result) {
            throw new BadRequestException('Цей тип робіт вже використовувся в
репортах, тож не може бути видаленим.');
        }

        $button = $this->buttonRepository->findOneBy(['resultType' => $resultType]);

        if ($button) {
            $this->entityManager->remove($button);
        }

        $this->entityManager->remove($resultType);
        $this->entityManager->flush();
    }

    /**
     * @param array $criteria
     *
     * @return \App\Dto\Request\ResultTypeDto[]
     */
    public function getResultTypes(array $criteria = []): array
    {
        $allowedCriteria = [
            'id',
            'resultTypeGroup',
            'measurementType',
        ];

        $criteria = array_filter($criteria, function ($criteriaName) use
($allowedCriteria): bool {
            return in_array($criteriaName, $allowedCriteria);
        }, ARRAY_FILTER_USE_KEY);
        $resultTypeDtos = [];
        $resultTypes = $this->resultTypeRepository->findBy($criteria, ['id' =>
'ASC']);

        foreach ($resultTypes as $resultType) {
            $resultTypeDtos[] = (new ResultTypeDto())
                ->setId($resultType->getId())
                ->setName($resultType->getName())
                ->setPrice($resultType->getPrice() / 100)
                ->setMeasurementType($resultType->getMeasurementType()->getId())
                ->setResultTypeGroup($resultType->getResultTypeGroup()->getId());
        }

        return $resultTypeDtos;
    }

    protected function mapModelToEntity(
        ResultTypeModel $resultTypeModel,
        ResultType $resultType
    ): ResultType {
        return $resultType
            ->setName($resultTypeModel->getName() ?? $resultType->getName())
            ->setPrice($resultTypeModel->getPrice() ? (int)(round($resultTypeModel-
>getPrice() * 100)) : $resultType->getPrice())
            ->setMeasurementType($resultTypeModel->getMeasurementType() ??
$resultType->getMeasurementType())
            ->setResultTypeGroup($resultTypeModel->getResultTypeGroup() ??
$resultType->getResultTypeGroup());
    }
```

```
}
```

Клас ResultsMessageFormatter

```
<?php

namespace App\Service\Result;

use App\Entity\Telegram\Chat;
use App\Repository\Result\ResultRepository;

class ResultsMessageFormatter
{
    /**
     * @var \App\Repository\Result\ResultRepository
     */
    private ResultRepository $resultRepository;

    /**
     * @param \App\Repository\Result\ResultRepository $resultRepository
     */
    public function __construct(
        ResultRepository $resultRepository
    ) {
        $this->resultRepository = $resultRepository;
    }

    /**
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return array
     */
    public function buildResultsMessage(Chat $chat): array
    {
        $messageParts = [];

        $reportedDate = $chat->getReportedData() ?? new \DateTime();
        $messagePart = sprintf("Виконані роботи за %s:\n", $reportedDate-
>format('d/m/Y'));

        $results = $this->resultRepository->findBy([
            'chat' => $chat,
            'year' => $reportedDate->format('Y'),
            'month' => $reportedDate->format('m'),
            'day' => $reportedDate->format('d'),
        ]);

        $resultsMap = $this->groupByGroups($results);

        foreach ($resultsMap as $groupName => $results) {
            $messagePrePart = "\n" . $groupName . ":\n";

            $collectedResults = [];

            foreach ($results as $result) {
                if (!isset($collectedResults[$result->getResultType()->getName()])) {
                    $collectedResults[$result->getResultType()->getName()] = $result;

                    continue;
```

```
                }

                $collectedResults[$result->getResultType()->getName()]->setSum(
                    $collectedResults[$result->getResultType()->getName()]->getSum()
                    + $result->getSum()
                );

                $collectedResults[$result->getResultType()->getName()]->setAmount(
                    $collectedResults[$result->getResultType()->getName()]-
>getAmount()
                    + $result->getAmount()
                );
            }

            foreach ($collectedResults as $result) {
                $messagePrePart .= sprintf(
                    "%s: %s %s\n",
                    $result->getResultType()->getName(),
                    $result->getAmount() / 100,
                    $result->getResultType()->getMeasurementType()->getContraction(),
                );
            }

            if (mb_strlen($messagePart . $messagePrePart) > 4000) {
                $messageParts[] = $messagePart;
                $messagePart = $messagePrePart;
                continue;
            }

            $messagePart .= $messagePrePart;
        }

        $messageParts[] = $messagePart;

        return $messageParts;
    }

    /**
     * @param \App\Entity\Result\Result[] $results
     * @param string $period
     *
     * @return string[]
     */
    public function buildForResults(array $results, string $period): array
    {
        $mapperResults = $this->groupByGroups($results);
        $totalSum = 0;
        $messagePart = sprintf("Виконанні роботи за період %s:\n", $period);

        foreach ($mapperResults as $groupName => $resultsInGroup) {
            $messagePrePart = "\n" . $groupName . ":\n";

            $collectedResults = [];

            foreach ($resultsInGroup as $result) {
                if (!isset($collectedResults[$result->getResultType()->getName()])) {
                    $collectedResults[$result->getResultType()->getName()] = $result;

                    continue;
                }

                $collectedResults[$result->getResultType()->getName()]->setSum(
                    $collectedResults[$result->getResultType()->getName()]->getSum()
                    + $result->getSum()
```

```
                );

                $collectedResults[$result->getResultType()->getName()]->setAmount(
                    $collectedResults[$result->getResultType()->getName()]-
>getAmount()
                    + $result->getAmount()
                );
            }

            foreach ($collectedResults as $result) {
                $messagePrePart .= sprintf(
                    "%s: %s %s\n",
                    $result->getResultType()->getName(),
                    $result->getAmount() / 100,
                    $result->getResultType()->getMeasurementType()->getContraction(),
                );
                $totalSum += $result->getSum();
            }

            if (mb_strlen($messagePart . $messagePrePart) > 4000) {
                $messageParts[] = $messagePart;
                $messagePart = $messagePrePart;
                continue;
            }

            $messagePart .= $messagePrePart;
        }

        $messageParts[] = $messagePart;
        $messageParts[] = sprintf("\nЗагадьний підсумок ствновить %s грн.", $totalSum
/ 10000);

        return $messageParts;
    }



    /**
     * @param \App\Entity\Result\Result[] $results
     *
     * @return \App\Entity\Result\Result[][]
     */
    protected function groupByGroups(array $results): array
    {
        $map = [];

        foreach ($results as $result) {
            $map[$result->getResultType()->getResultTypeGroup()->getName()][] =
$result;
        }

        return $map;
    }
}
```

Клас ApproveChatProcessor

```
<?php
```

```
namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use DateTime;
use Doctrine\ORM\EntityManagerInterface;

class ApproveChatProcessor
{
    public const APPROVE_CHAT_ACTION = 'bot_action:type_approve_code';

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        if ((int)$userMessageDTO->getText() !== $chat->getEmployee()-
>getApprovalKey()) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                'Код підтвердження невірний. Перевірте чи він був уведений вірно та
спробуйте ще раз.',
                []
            );

            return;
        }

        $chat
            ->setIsRegistered(true)
```

```
            ->setExpectedAction(null)
            ->getEmployee()
                ->setIsRegistered(true)
                ->setRegisteredAt(new DateTime());

        $this->entityManager->flush();

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Ви успішно підтвердили реєстрацію. Тепер ви можете надсилати результати
виконанних робіт.',
            $this->builder->buildHomeKeyboard()
        );
    }
}
```

Клас ChooseReportProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Repository\Result\ResultRepository;
use App\Repository\Telegram\ButtonRepository;
use App\Service\Result\ResultsMessageFormatter;
use App\Service\Result\ResultTypeFormatBuilder;
use App\Service\Telegram\ButtonTextConstants;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class ChooseReportProcessor
{
    public const CHOOSE_REPORT_ACTION = 'bot_action:choose_report_type';

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;
    /**
     * @var \App\Repository\Result\ResultRepository
     */
    private ResultRepository $resultRepository;
    /**
     * @var \App\Service\Result\ResultsMessageFormatter
     */
    private ResultsMessageFormatter $resultsMessageFormatter;
```

```
    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder,
        ResultRepository $resultRepository,
        ResultsMessageFormatter $resultsMessageFormatter
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
        $this->resultRepository = $resultRepository;
        $this->resultsMessageFormatter = $resultsMessageFormatter;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        switch ($userMessageDTO->getText()):
            case ButtonTextConstants::PREVIOUS_WEEK_REPORT_TEXT:
                $this->buildPreviousWeekReport($chat);
                return;
            case ButtonTextConstants::THIS_WEEK_REPORT_TEXT:
                $this->buildCurrentWeekReport($chat);
                return;
            case ButtonTextConstants::PREVIOUS_MONTH_REPORT_TEXT:
                $this->buildPreviousMonthReport($chat);
                return;
            case ButtonTextConstants::THIS_MONTH_REPORT_TEXT:
                $this->buildCurrentMonthReport($chat);
                return;
        endswitch;

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Невідомий тип звіту',
        );
    }

    protected function buildCurrentWeekReport(Chat $chat): void
    {
        $dayOfWeek = date('N');
        $currentDate = new \DateTime();
        $dateStartOfWeek = (new \DateTime())
            ->sub(new \DateInterval('P'  . ($dayOfWeek - 1) . 'D'))
            ->setTime(0, 0);

        $results = $this->resultRepository->findForDate($dateStartOfWeek,
$currentDate, $chat);

        $this->sendReport($results, $chat, $dateStartOfWeek, $currentDate);
    }

    protected function buildPreviousWeekReport(Chat $chat): void
    {
```

```
        $dayOfWeek = date('N');
        $endOfPreviousWeek = (new \DateTime())
            ->sub(new \DateInterval('P'  . $dayOfWeek. 'D'))
            ->setTime(23, 59);
        $startOfPreviousWeek = (clone $endOfPreviousWeek)
            ->sub(new \DateInterval('P6D'))
            ->setTime(0, 0);

        $results = $this->resultRepository->findForDate($startOfPreviousWeek,
$endOfPreviousWeek, $chat);

        $this->sendReport($results, $chat, $startOfPreviousWeek, $endOfPreviousWeek);
    }

    protected function buildCurrentMonthReport(Chat $chat): void
    {
        $startOfMonth = (new \DateTime())
            ->setDate(date('Y'), date('n'), 1)
            ->setTime(0, 0);

        $results = $this->resultRepository->findForDate($startOfMonth, new
\DateTime(), $chat);

        $this->sendReport($results, $chat, $startOfMonth, new \DateTime());
    }

    protected function buildPreviousMonthReport(Chat $chat): void
    {
        $endOfMonth = (new \DateTime())
            ->setDate(date('Y'), date('n'), 1)
            ->setTime(23, 59)
            ->sub(new \DateInterval('P1D'));
        $startOfMonth = (new \DateTime())
            ->setDate($endOfMonth->format('Y'), $endOfMonth->format('n'), 1)
            ->setTime(0, 0);

        $results = $this->resultRepository->findForDate($startOfMonth, $endOfMonth,
$chat);

        $this->sendReport($results, $chat, $startOfMonth, $endOfMonth);
    }

    /**
     * @param \App\Entity\Result\Result[] $results
     * @param \App\Entity\Telegram\Chat $chat
     * @param \DateTime $startPeriodDate
     * @param \DateTime $endPeriodDate
     *
     * @return void
     */
    protected function sendReport(
        array $results,
        Chat $chat,
        \DateTime $startPeriodDate,
        \DateTime $endPeriodDate
    ): void {
        $messageParts = $this->resultsMessageFormatter->buildForResults(
            $results,
            sprintf(
                '%s - %s',
                $startPeriodDate->format('d/m/Y'),
                $endPeriodDate->format('d/m/Y'),
            )
        );
```

```
        foreach ($messageParts as $messagePart) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                $messagePart,
            );
        }
    }
}
```

Клас ChooseResultTypeGroupProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Repository\Telegram\ButtonRepository;
use App\Service\Telegram\ButtonTextConstants;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class ChooseResultTypeGroupProcessor
{
    public const CHOOSE_RESULT_TYPE_GROUP_ACTION =
'bot_action:choose_result_type_group';

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;
    /**
     * @var \App\Repository\Telegram\ButtonRepository
     */
    private ButtonRepository $buttonRepository;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ToSetResultProcessor
     */
    private ToSetResultProcessor $toSetResultProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ToSetReportedDateProcessor
     */
    private ToSetReportedDateProcessor $toSetReportedDateProcessor;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
```

```
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder,
        ButtonRepository $buttonRepository,
        ToSetResultProcessor $toSetResultProcessor,
        ToSetReportedDateProcessor $toSetReportedDateProcessor
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
        $this->buttonRepository = $buttonRepository;
        $this->toSetResultProcessor = $toSetResultProcessor;
        $this->toSetReportedDateProcessor = $toSetReportedDateProcessor;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        if ($userMessageDTO->getText() === ButtonTextConstants::RESET_DATE_TEXT) {
            $chat->setReportedData(null);
            $this->toSetResultProcessor->processMessage($userMessageDTO, $chat);

            return;
        }

        if ($userMessageDTO->getText() === ButtonTextConstants::SET_DATE_TEXT) {
            $this->toSetReportedDateProcessor->processMessage($userMessageDTO,
$chat);

            return;
        }

        $groupButton = $this->buttonRepository->findOneBy(['text' => $userMessageDTO-
>getText()]);

        if (!$groupButton || !$groupButton->getResultTypeGroup()) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                'Невідома категорія',
            );

            return;
        }

        $chat-
>setExpectedAction(ChooseResultTypeProcessor::CHOOSE_RESULT_TYPE_ACTION);

        $this->entityManager->flush();

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            $groupButton->getResultTypeGroup()->getDescription(),
            $this->builder->buildGroupKeyboard($groupButton->getResultTypeGroup())
        );
    }
}
```

Клас ChooseResultTypeProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Repository\Telegram\ButtonRepository;
use App\Service\Result\ResultTypeFormatBuilder;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class ChooseResultTypeProcessor
{
    public const CHOOSE_RESULT_TYPE_ACTION = 'bot_action:choose_result_type';

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;
    /**
     * @var \App\Repository\Telegram\ButtonRepository
     */
    private ButtonRepository $buttonRepository;
    /**
     * @var \App\Service\Result\ResultTypeFormatBuilder
     */
    private ResultTypeFormatBuilder $resultTypeFormatBuilder;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder,
        ButtonRepository $buttonRepository,
        ResultTypeFormatBuilder $resultTypeFormatBuilder
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
        $this->buttonRepository = $buttonRepository;
        $this->resultTypeFormatBuilder = $resultTypeFormatBuilder;
    }
```

```
    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $typeButton = $this->buttonRepository->findOneBy(['text' => $userMessageDTO-
>getText()]);

        if (!$typeButton || !$typeButton->getResultType()) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                'Невідомий тип роботи',
            );

            return;
        }

        $chat->setExpectedAction(SetResultProcessor::SET_RESULT_ACTION)
            ->setExpectedResultTypeId($typeButton->getResultType()->getId());

        $this->entityManager->flush();

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            "Цей тип роботи вимірюється в такій одиниці:  {$typeButton-
>getResultType()->getMeasurementType()->getName()} ({$typeButton->getResultType()-
>getMeasurementType()->getContraction()}).\nДопустимі формати переданих даних для
обраного типу роботи:\n" .
            $this->resultTypeFormatBuilder->getAvailableFormats($typeButton-
>getResultType()),
            $this->builder->buildToHomeKeyboard()
        );
    }
}
```

Клас MessageProcessorInterface

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;

interface MessageProcessorInterface
{
    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     *
     * @return bool
     */
    public function isMessageApplicable(UserMessageDTO $userMessageDTO): bool;

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     *
```

```
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO): void;
}
```

Клас SetReportedDateProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class SetReportedDateProcessor
{
    public const SET_REPORTED_DATE_ACTION = 'bot_action:set_reported_data';

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;
    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ToSetResultProcessor
     */
    private ToSetResultProcessor $toSetResultProcessor;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     * @param \App\Service\Telegram\KeyboardBuilder $builder
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder,
        ToSetResultProcessor $toSetResultProcessor
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
        $this->toSetResultProcessor = $toSetResultProcessor;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
```

```
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $dateInfo = explode('/', $userMessageDTO->getText());
        if (count($dateInfo) !== 3 || !checkdate($dateInfo[1], $dateInfo[0],
$dateInfo[2])) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                "Дату уведено не вірно.\nФормат ведення дати: d/m/Y - 16/09/2022",
            );

            return;
        }

        $reportedDate = new \DateTime(implode('/', [$dateInfo[1], $dateInfo[0],
$dateInfo[2]]));

        if ($reportedDate->getTimestamp() > time()) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                'Не можна вказувати майбутню дату. Уведіть дату знову.',
            );

            return;
        }

        $chat->setReportedData($reportedDate);

        $this->toSetResultProcessor->processMessage($userMessageDTO, $chat);
    }
}
```

Клас SetResultProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Result\Result;
use App\Entity\Result\ResultType;
use App\Entity\Telegram\Chat;
use App\Repository\Result\ResultRepository;
use App\Repository\Result\ResultTypeRepository;
use App\Service\Result\ResultAmountValidator;
use App\Service\Result\ResultsMessageFormatter;
use App\Service\Result\ResultTypeFormatBuilder;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use DateTime;
use Doctrine\ORM\EntityManagerInterface;

class SetResultProcessor
{
    public const SET_RESULT_ACTION = 'bot_action:set_result';

    /**
```

```
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;
    /**
     * @var \App\Repository\Result\ResultTypeRepository
     */
    private ResultTypeRepository $resultTypeRepository;
    /**
     * @var \App\Repository\Result\ResultRepository
     */
    private ResultRepository $resultRepository;
    /**
     * @var \App\Service\Result\ResultAmountValidator
     */
    private ResultAmountValidator $resultAmountValidator;
    /**
     * @var \App\Service\Result\ResultTypeFormatBuilder
     */
    private ResultTypeFormatBuilder $resultTypeFormatBuilder;
    /**
     * @var \App\Service\Result\ResultsMessageFormatter
     */
    private ResultsMessageFormatter $resultsMessageFormatter;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder,
        ResultTypeRepository $resultTypeRepository,
        ResultRepository $resultRepository,
        ResultAmountValidator $resultAmountValidator,
        ResultTypeFormatBuilder $resultTypeFormatBuilder,
        ResultsMessageFormatter $resultsMessageFormatter
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
        $this->resultTypeRepository = $resultTypeRepository;
        $this->resultRepository = $resultRepository;
        $this->resultAmountValidator = $resultAmountValidator;
        $this->resultTypeFormatBuilder = $resultTypeFormatBuilder;
        $this->resultsMessageFormatter = $resultsMessageFormatter;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
```

```
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $value = $userMessageDTO->getText();
        $resultType = $this->resultTypeRepository->find($chat-
>getExpectedResultTypeId());

        if (!$this->resultAmountValidator->isValid($value, $resultType)) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                "Не вірний формат подачі результаттів обраного типу роботи. Спробуйте
увести значення ще раз.\n" .
                "Вірні формати введення:\n" .
                $this->resultTypeFormatBuilder->getAvailableFormats($resultType),
            );

            return;
        }

        $result = $this->getResultItem($chat, $resultType);
        $this->updateResultAmount($value, $result);

        $chat->setExpectedResultTypeId(null)
            -
>setExpectedAction(ChooseResultTypeGroupProcessor::CHOOSE_RESULT_TYPE_GROUP_ACTION);

        $this->entityManager->flush();

        $messageParts = $this->resultsMessageFormatter->buildResultsMessage($chat);

        foreach ($messageParts as $messagePart) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                $messagePart,
                $this->builder->buildGroupsKeyboard($chat)
            );
        }
    }

    /**
     * @param \App\Entity\Telegram\Chat $chat
     * @param \App\Entity\Result\ResultType $resultType
     *
     * @return \App\Entity\Result\Result
     */
    protected function getResultItem(Chat $chat, ResultType $resultType): Result
    {
        $reportedDate = $chat->getReportedData() ?? new DateTime();
        $result = $this->resultRepository->findOneBy([
            'year' => (int)$reportedDate->format('Y'),
            'month' => (int)$reportedDate->format('m'),
            'day' => (int)$reportedDate->format('d'),
            'chat' => $chat,
            'resultType' => $resultType,
        ]);

        if (!$result) {
            $result = (new Result())
                ->setChat($chat)
                ->setResultType($resultType)
                ->setYear((int)$reportedDate->format('Y'))
                ->setMonth((int)$reportedDate->format('m'))
                ->setDay((int)$reportedDate->format('d'))
                ->setReportedDate($reportedDate)
                ->setCreatedAt(new DateTime())
```

```
                ->setUpdatedAt(new DateTime());

            $this->entityManager->persist($result);
        } else {
            $result->setUpdatedAt(new DateTime());
        }

        return $result;
    }

    /**
     * @param \App\Entity\Result\Result $result
     *
     * @return void
     */
    protected function updateResultAmount(string $value, Result $result): void
    {
        $amount = $result->getAmount() + (int)((float)$value * 100);

        if ($amount === 0) {
            $this->entityManager->remove($result);
        } else {
            $result->setAmount($amount)
                ->setActualPrice($result->getResultType()->getPrice())
                ->setSum($amount * $result->getResultType()->getPrice());
        }

    }
}
```

Клас StartMessageProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Employee\Employee;
use App\Entity\Telegram\Chat;
use App\Repository\Employee\EmployeeRepository;
use App\Repository\Telegram\ChatRepository;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;
use Ramsey\Uuid\Uuid;

class StartMessageProcessor implements MessageProcessorInterface
{
    public const MAIN_MESSAGE_PART = '/start';

    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Repository\Employee\EmployeeRepository
     */
    protected EmployeeRepository $employeeRepository;
```

```
    /**
     * @var \App\Repository\Telegram\ChatRepository
     */
    protected ChatRepository $chatRepository;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Repository\Employee\EmployeeRepository $employeeRepository
     * @param \App\Repository\Telegram\ChatRepository $chatRepository
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        EmployeeRepository $employeeRepository,
        ChatRepository $chatRepository,
        TelegramApiRequestExecutor $telegramApiRequestExecutor
    ) {
        $this->entityManager = $entityManager;
        $this->employeeRepository = $employeeRepository;
        $this->chatRepository = $chatRepository;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
    }

    public function isMessageApplicable(UserMessageDTO $userMessageDTO): bool
    {
        return substr($userMessageDTO->getText(), 0, 6) === self::MAIN_MESSAGE_PART;
    }

    public function processMessage(UserMessageDTO $userMessageDTO): void
    {
        $chat = $this->chatRepository->findOneBy(['chatId' => $userMessageDTO-
>getChatId()]);

        if ($chat) {
            //TODO::chat should not exists
            // TODO::IF employee is activated then return main keyboard
            dd('chat exists');

            return;
        }

        $messageParts = explode(' ', $userMessageDTO->getText());

        if (count($messageParts) !== 2 || !Uuid::isValid($messageParts[1])) {
            //TODO::Send appropriate message
            dd('wrong message structure');

            return;
        }

        $employee = $this->employeeRepository->findOneBy(['regToken' =>
$messageParts[1]]);

        if (!$employee || $employee->getIsRegistered()) {
            // TODO::Return appropriate message
            dd('employee is not found');

            return;
```

```
        }

        $chat = $this->createChat($userMessageDTO, $employee);
        $chat->setExpectedAction(ApproveChatProcessor::APPROVE_CHAT_ACTION);

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Введіть код підтвердження.',
            []
        );
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Employee\Employee $employee
     *
     * @return \App\Entity\Telegram\Chat
     */
    protected function createChat(UserMessageDTO $userMessageDTO, Employee
$employee): Chat
    {
        $chat = (new Chat())
            ->setChatId($userMessageDTO->getChatId())
            ->setUserId($userMessageDTO->getUserId())
            ->setFirstname($userMessageDTO->getFirstName())
            ->setEmployee($employee);

        $this->entityManager->persist($chat);
        $this->entityManager->flush();

        return $chat;
    }
}
```

Клас ToHomeProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class ToHomeProcessor
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
```

```
     */
    protected KeyboardBuilder $builder;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $chat
            ->setExpectedAction(null)
            ->setReportedData(null)
            ->setExpectedResultTypeId(null);

        $this->entityManager->flush();

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Головна',
            $this->builder->buildHomeKeyboard()
        );
    }
}
```

Клас ToReportsProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class ToReportsProcessor
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;
```

```
    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $chat
            ->setExpectedAction(ChooseReportProcessor::CHOOSE_REPORT_ACTION)
            ->setReportedData(null)
            ->setExpectedResultTypeId(null);

        $this->entityManager->flush();

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Оберіть тип звіту який ви хочете переглянути',
            $this->builder->buildReportKeyboard(),
        );
    }
}
```

Клас ToSetReportedDateProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;
```

```
class ToSetReportedDateProcessor
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;
    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     * @param \App\Service\Telegram\KeyboardBuilder $builder
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $chat->setExpectedAction(SetReportedDateProcessor::SET_REPORTED_DATE_ACTION);
        $this->entityManager->flush();

        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Формат ведення дати: d/m/Y - 16/09/2022',
            $this->builder->buildToHomeKeyboard()
        );
    }
}
```

Клас ToSetResultProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
```

```
use App\Service\Result\ResultsMessageFormatter;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;
use Doctrine\ORM\EntityManagerInterface;

class ToSetResultProcessor
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    protected EntityManagerInterface $entityManager;

    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;
    /**
     * @var \App\Service\Result\ResultsMessageFormatter
     */
    private ResultsMessageFormatter $resultsMessageFormatter;

    /**
     * @param \Doctrine\ORM\EntityManagerInterface $entityManager
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        EntityManagerInterface $entityManager,
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder,
        ResultsMessageFormatter $resultsMessageFormatter
    ) {
        $this->entityManager = $entityManager;
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
        $this->resultsMessageFormatter = $resultsMessageFormatter;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $chat-
>setExpectedAction(ChooseResultTypeGroupProcessor::CHOOSE_RESULT_TYPE_GROUP_ACTION);
        $this->entityManager->flush();

        $messageParts = $this->resultsMessageFormatter->buildResultsMessage($chat);

        foreach ($messageParts as $messagePart) {
            $this->telegramApiRequestExecutor->sendMessage(
                $chat->getChatId(),
                $messagePart,
                $this->builder->buildGroupsKeyboard($chat)
            );
        }
    }
```

```
}
```

Клас UnknownMessageProcessor

```
<?php

namespace App\Service\Telegram\MessageProcessors;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Service\Telegram\KeyboardBuilder;
use App\Service\Telegram\TelegramApiRequestExecutor;

class UnknownMessageProcessor
{
    /**
     * @var \App\Service\Telegram\TelegramApiRequestExecutor
     */
    protected TelegramApiRequestExecutor $telegramApiRequestExecutor;
    /**
     * @var \App\Service\Telegram\KeyboardBuilder
     */
    protected KeyboardBuilder $builder;

    /**
     * @param \App\Service\Telegram\TelegramApiRequestExecutor
$telegramApiRequestExecutor
     */
    public function __construct(
        TelegramApiRequestExecutor $telegramApiRequestExecutor,
        KeyboardBuilder $builder
    ) {
        $this->telegramApiRequestExecutor = $telegramApiRequestExecutor;
        $this->builder = $builder;
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return void
     */
    public function processMessage(UserMessageDTO $userMessageDTO, Chat $chat): void
    {
        $this->telegramApiRequestExecutor->sendMessage(
            $chat->getChatId(),
            'Не можемо опрацювати ваш запит',
        );
    }
}
```

Клас UpdateToUserMessageDTOMapper

```
<?php
```

```
namespace App\Service\Telegram\UpdateMappers;

use App\Dto\Telegram\UserMessageDTO;

class UpdateToUserMessageDTOMapper
{
    /**
     * @param mixed $updateRequestBody
     *
     * @return UserMessageDTO
     */
    public function mapUpdateBodyToUserMessageDTOMapper(array $updateRequestBody):
UserMessageDTO
    {
        $message = $updateRequestBody['message'] ?? null;
        $from = $message['from'] ?? null;
        $chat = $message['chat'] ?? null;

        return (new UserMessageDTO())
            ->setUpdateId($updateRequestBody['update_id'] ?? null)
            ->setMessageId($message['message_id'] ?? null)
            ->setText(isset($message['text']) ? trim($message['text']) : null)
            ->setDate($message['date'] ?? null)

            ->setChatId($chat['id'] ?? null)
            ->setChatType($chat['type'] ?? null)

            ->setUserId($from['id'] ?? null)
            ->setIsBot($from['is_bot'] ?? null)
            ->setFirstName($from['first_name'] ?? null)
            ->setLastName($from['last_name'] ?? null)
            ->setUsername($from['username'] ?? null)
            ->setLanguageCode($from['language_code'] ?? null);
    }
}
```

Клас LiveWebhookRequestBodyProcessor

```
<?php

namespace App\Service\Telegram\WebhookRequestBodyProcessor;

use App\Service\Telegram\MessageProcessor;
use App\Service\Telegram\UpdateMappers\UpdateToUserMessageDTOMapper;

class LiveWebhookRequestBodyProcessor implements WebhookRequestBodyProcessorInterface
{
    private MessageProcessor $messageProcessor;
    private UpdateToUserMessageDTOMapper $updateToUserMessageDTOMapper;

    public function __construct(
        MessageProcessor $messageProcessor,
        UpdateToUserMessageDTOMapper $updateToUserMessageDTOMapper
    ) {
        $this->messageProcessor = $messageProcessor;
        $this->updateToUserMessageDTOMapper = $updateToUserMessageDTOMapper;
    }

    public function processRequest(string $requestBody): void
```

```
    {
        $userMessageDto = $this->updateToUserMessageDTOMapper-
>mapUpdateBodyToUserMessageDTOMapper(json_decode($requestBody, true));

        $this->messageProcessor->processMessage($userMessageDto);
    }
}
```

Клас QueueAvailableInterface

```
<?php

namespace App\Service\Telegram\WebhookRequestBodyProcessor;

use Symfony\Component\HttpFoundation\Request;

interface QueueAvailableInterface
{
    /**
     * Specifications:
     * - It should check if connection exists and is stable.
     *
     * @return bool
     */
    public function isQueueAvailable(): bool;
}
```

Клас WebhookRequestBodyProcessorInterface

```
<?php

namespace App\Service\Telegram\WebhookRequestBodyProcessor;

use Symfony\Component\HttpFoundation\Request;

interface WebhookRequestBodyProcessorInterface
{
    public function processRequest(string $requestBody): void;
}
```

Клас ButtonService

```
<?php

namespace App\Service\Telegram;

use App\Dto\Telegram\ButtonDto;
use App\Entity\Telegram\Button;
use App\Form\Model\Telegram\ButtonModel;
use App\Repository\Telegram\ButtonRepository;
```

```
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Exception\BadRequestException;

class ButtonService
{
    /**
     * @var \Doctrine\ORM\EntityManagerInterface
     */
    private $entityManager;
    /**
     * @var \App\Repository\Telegram\ButtonRepository
     */
    private $buttonRepository;

    public function __construct(
        EntityManagerInterface $entityManager,
        ButtonRepository $buttonRepository
    ) {
        $this->entityManager = $entityManager;
        $this->buttonRepository = $buttonRepository;
    }

    /**
     * @param \App\Form\Model\Telegram\ButtonModel $buttonModel
     * @param \App\Entity\Telegram\Button|null $button
     *
     * @return \App\Dto\Telegram\ButtonDto
     */
    public function saveButton(ButtonModel $buttonModel, ?Button $button = null):
ButtonDto
    {
        if (!$button) {
            $button = new Button();
            $this->entityManager->persist($button);
        }

        if ($buttonModel->getResultType() !== null && $buttonModel-
>getResultTypeGroup() !== null) {
            throw new BadRequestException('We can not assign button to both
entities.');
        }

        $existButton = $this->buttonRepository->findOneBy(['text' => $buttonModel-
>getText()]);

        if ($existButton && (!$button->getId() || $button->getId() !== $existButton-
>getId())) {
            throw new BadRequestException('This text is already in use.');
        }

        $button->setText($buttonModel->getText())
            ->setResultTypeGroup($buttonModel->getResultTypeGroup())
            ->setResultType($buttonModel->getResultType());

        $this->entityManager->flush();

        return (new ButtonDto())
            ->setId($button->getId())
            ->setText($button->getText())
            ->setResultType($button->getResultType() ? $button->getResultType()-
>getId() : null)
            ->setResultTypeGroup($button->getResultTypeGroup() ? $button-
>getResultTypeGroup()->getId() : null);
    }
```

```
    public function deleteButton(Button $button): void
    {
        $this->entityManager->remove($button);
        $this->entityManager->flush();
    }

    /**
     * @return \App\Dto\Telegram\ButtonDto[]
     */
    public function getButtons(): array
    {
        $buttonDtos = [];
        $buttons = $this->buttonRepository->findAll();

        foreach ($buttons as $button) {
            $buttonDtos[] = (new ButtonDto())
                ->setId($button->getId())
                ->setText($button->getText())
                ->setResultType($button->getResultType() ? $button->getResultType()-
>getId() : null)
                ->setResultTypeGroup($button->getResultTypeGroup() ? $button-
>getResultTypeGroup()->getId() : null);
        }

        return $buttonDtos;
    }
}
```

Клас ButtonTextConstants

```
<?php

namespace App\Service\Telegram;

interface ButtonTextConstants
{
    public const SET_RESULT_TEXT = 'Подача результатів';

    public const HOME_TEXT = 'На головну';
    public const SET_DATE_TEXT = 'Встановити дату';
    public const RESET_DATE_TEXT = 'Відновити дату';

    public const TO_GROUPS_TEXT = 'До категорій';

    public const TO_REPORTS_TEXT = 'Підсумки результатів';
    public const THIS_WEEK_REPORT_TEXT = 'Поточний тиждень';
    public const PREVIOUS_WEEK_REPORT_TEXT = 'Попередній тиждень';
    public const THIS_MONTH_REPORT_TEXT = 'Поточний місяць';
    public const PREVIOUS_MONTH_REPORT_TEXT = 'Попередній місяць';
}
```

Клас KeyboardBuilder

```
<?php
```

```
namespace App\Service\Telegram;

use App\Entity\Result\ResultTypeGroup;
use App\Entity\Telegram\Chat;
use App\Repository\Telegram\ButtonRepository;

class KeyboardBuilder
{
    /**
     * @var \App\Repository\Telegram\ButtonRepository
     */
    private $buttonRepository;

    /**
     * @param \App\Repository\Telegram\ButtonRepository $buttonRepository
     */
    public function __construct(
        ButtonRepository $buttonRepository
    ) {
        $this->buttonRepository = $buttonRepository;
    }

    /**
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return string[][]
     */
    public function buildGroupsKeyboard(Chat $chat): array
    {
        $keyboard = [];

        $buttons = $this->buttonRepository->getGroupsButtons();
        $buttonsRow = [];

        foreach ($buttons as $button) {
            $buttonsRow[] = $button->getText();

            if (count($buttonsRow) === 2) {
                $keyboard[] = $buttonsRow;
                $buttonsRow = [];
            }
        }

        if (count($buttonsRow)) {
            $keyboard[] = $buttonsRow;
        }

        $keyboard = $this->addReportedDateButton($keyboard, $chat);
        return $this->addHomeButton($keyboard);
    }

    /**
     * @param \App\Entity\Result\ResultTypeGroup $resultTypeGroup
     *
     * @return string[][]
     */
    public function buildGroupKeyboard(ResultTypeGroup $resultTypeGroup): array
    {
        $keyboard = [];

        $buttons = $this->buttonRepository->getGroupButtons($resultTypeGroup);

        foreach ($buttons as $button) {
```

```
            $keyboard[] = [$button->getText()];
        }

        $keyboard[] = [ButtonTextConstants::TO_GROUPS_TEXT];
        return $this->addHomeButton($keyboard);
    }

    /**
     * @return string[][]
     */
    public function buildHomeKeyboard(): array
    {
        return [
            [ButtonTextConstants::SET_RESULT_TEXT],
            [ButtonTextConstants::TO_REPORTS_TEXT],
        ];
    }

    /**
     * @return string[][]
     */
    public function buildReportKeyboard(): array
    {
        return $this->addHomeButton([
            [ButtonTextConstants::THIS_MONTH_REPORT_TEXT],
            [ButtonTextConstants::PREVIOUS_MONTH_REPORT_TEXT],
            [ButtonTextConstants::THIS_WEEK_REPORT_TEXT],
            [ButtonTextConstants::PREVIOUS_WEEK_REPORT_TEXT],
        ]);
    }

    /**
     * @return string[][]
     */
    public function buildToHomeKeyboard(): array
    {
        return $this->addHomeButton([]);
    }

    /**
     * @param string[][] $keyboard
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return string[][]
     */
    protected function addReportedDateButton(array $keyboard, Chat $chat): array
    {
        $buttonText = $chat->getReportedData() ? ButtonTextConstants::RESET_DATE_TEXT
: ButtonTextConstants::SET_DATE_TEXT;
        $keyboard[] = [$buttonText];
        return $keyboard;
    }

    /**
     * @param string[][] $keyboard
     *
     * @return string[][]
     */
    protected function addHomeButton(array $keyboard): array
    {
        $keyboard[] = [ButtonTextConstants::HOME_TEXT];

        return $keyboard;
    }
```

```
}
```

Клас MessageProcessor

```
<?php

namespace App\Service\Telegram;

use App\DTO\Telegram\UserMessageDTO;
use App\Entity\Telegram\Chat;
use App\Repository\Telegram\ChatRepository;
use App\Service\Telegram\MessageProcessors\ApproveChatProcessor;
use App\Service\Telegram\MessageProcessors\ChooseReportProcessor;
use App\Service\Telegram\MessageProcessors\ChooseResultTypeGroupProcessor;
use App\Service\Telegram\MessageProcessors\ChooseResultTypeProcessor;
use App\Service\Telegram\MessageProcessors\SetReportedDateProcessor;
use App\Service\Telegram\MessageProcessors\SetResultProcessor;
use App\Service\Telegram\MessageProcessors\ToReportsProcessor;
use App\Service\Telegram\MessageProcessors\ToSetResultProcessor;
use App\Service\Telegram\MessageProcessors\StartMessageProcessor;
use App\Service\Telegram\MessageProcessors\ToHomeProcessor;
use App\Service\Telegram\MessageProcessors\UnknownMessageProcessor;

class MessageProcessor
{
    /**
     * @var \App\Service\Telegram\MessageProcessors\StartMessageProcessor
     */
    protected $startMessageProcessor;
    /**
     * @var \App\Repository\Telegram\ChatRepository
     */
    protected ChatRepository $chatRepository;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ApproveChatProcessor
     */
    protected ApproveChatProcessor $approveChatProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ToHomeProcessor
     */
    protected ToHomeProcessor $toHomeProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ToSetResultProcessor
     */
    private ToSetResultProcessor $toSetResultProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\UnknownMessageProcessor
     */
    private UnknownMessageProcessor $unknownMessageProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ChooseResultTypeGroupProcessor
     */
    private ChooseResultTypeGroupProcessor $chooseResultTypeGroupProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ChooseResultTypeProcessor
     */
    private ChooseResultTypeProcessor $chooseResultTypeProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\SetResultProcessor
```

```
     */
    private SetResultProcessor $setResultProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\SetReportedDateProcessor
     */
    private SetReportedDateProcessor $setReportedDateProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ToReportsProcessor
     */
    private ToReportsProcessor $toReportsProcessor;
    /**
     * @var \App\Service\Telegram\MessageProcessors\ChooseReportProcessor
     */
    private ChooseReportProcessor $chooseReportProcessor;

    /**
     * @param \App\Service\Telegram\MessageProcessors\StartMessageProcessor
$startMessageProcessor
     * @param \App\Service\Telegram\MessageProcessors\ApproveChatProcessor
$approveChatProcessor
     * @param \App\Repository\Telegram\ChatRepository $chatRepository
     */
    public function __construct(
        StartMessageProcessor          $startMessageProcessor,
        ApproveChatProcessor           $approveChatProcessor,
        ToHomeProcessor                $toHomeProcessor,
        ToSetResultProcessor           $toSetResultProcessor,
        UnknownMessageProcessor        $unknownMessageProcessor,
        ChooseResultTypeGroupProcessor $chooseResultTypeGroupProcessor,
        ChooseResultTypeProcessor      $chooseResultTypeProcessor,
        SetResultProcessor             $setResultProcessor,
        SetReportedDateProcessor       $setReportedDateProcessor,
        ToReportsProcessor             $toReportsProcessor,
        ChooseReportProcessor          $chooseReportProcessor,
        ChatRepository                 $chatRepository
    ) {
        $this->startMessageProcessor = $startMessageProcessor;
        $this->chatRepository = $chatRepository;
        $this->approveChatProcessor = $approveChatProcessor;
        $this->toHomeProcessor = $toHomeProcessor;
        $this->toSetResultProcessor = $toSetResultProcessor;
        $this->unknownMessageProcessor = $unknownMessageProcessor;
        $this->chooseResultTypeGroupProcessor = $chooseResultTypeGroupProcessor;
        $this->chooseResultTypeProcessor = $chooseResultTypeProcessor;
        $this->setResultProcessor = $setResultProcessor;
        $this->setReportedDateProcessor = $setReportedDateProcessor;
        $this->toReportsProcessor = $toReportsProcessor;
        $this->chooseReportProcessor = $chooseReportProcessor;
    }

    public function processMessage(UserMessageDTO $userMessageDTO): void
    {
        if ($this->startMessageProcessor->isMessageApplicable($userMessageDTO)) {
            $this->startMessageProcessor->processMessage($userMessageDTO);

            return;
        }

        $chat = $this->chatRepository->findOneBy(['chatId' => $userMessageDTO-
>getChatId()]);

        if (!$chat) {
            //TODO::Send relevant message
```

```
            return;
        }

        if (!$chat->getIsRegistered()) {
            $this->approveChatProcessor->processMessage($userMessageDTO, $chat);

            return;
        }

        if ($userMessageDTO->getText() === ButtonTextConstants::HOME_TEXT) {
            $this->toHomeProcessor->processMessage($userMessageDTO, $chat);

            return;
        }

        if ($chat->getExpectedAction()) {
            if ($this->processExpectedAction($userMessageDTO, $chat)) {
                return;
            }
        }

        if ($userMessageDTO->getText() === ButtonTextConstants::SET_RESULT_TEXT) {
            $this->toSetResultProcessor->processMessage($userMessageDTO, $chat);

            return;
        }

        if ($userMessageDTO->getText() === ButtonTextConstants::TO_REPORTS_TEXT) {
            $this->toReportsProcessor->processMessage($userMessageDTO, $chat);

            return;
        }

        $this->unknownMessageProcessor->processMessage($userMessageDTO, $chat);
    }

    /**
     * @param \App\DTO\Telegram\UserMessageDTO $userMessageDTO
     * @param \App\Entity\Telegram\Chat $chat
     *
     * @return bool
     */
    protected function processExpectedAction(UserMessageDTO $userMessageDTO, Chat
$chat): bool
    {
        if ($userMessageDTO->getText() === ButtonTextConstants::TO_GROUPS_TEXT) {
            $this->toSetResultProcessor->processMessage($userMessageDTO, $chat);

            return true;
        }

        switch ($chat->getExpectedAction()):
            case ChooseResultTypeGroupProcessor::CHOOSE_RESULT_TYPE_GROUP_ACTION:
                $this->chooseResultTypeGroupProcessor-
>processMessage($userMessageDTO, $chat);
                return true;
            case ChooseResultTypeProcessor::CHOOSE_RESULT_TYPE_ACTION:
                $this->chooseResultTypeProcessor->processMessage($userMessageDTO,
$chat);
                return true;
            case SetResultProcessor::SET_RESULT_ACTION:
                $this->setResultProcessor->processMessage($userMessageDTO, $chat);
                return true;
            case SetReportedDateProcessor::SET_REPORTED_DATE_ACTION:
```

```
                $this->setReportedDateProcessor->processMessage($userMessageDTO,
$chat);
                return true;
            case ChooseReportProcessor::CHOOSE_REPORT_ACTION;
                $this->chooseReportProcessor->processMessage($userMessageDTO, $chat);
                return true;
            default:
                return false;
        endswitch;
    }
}
```

Клас TelegramApiRequestExecutor

```
<?php

declare(strict_types=1);

namespace App\Service\Telegram;

use GuzzleHttp\Client;
use GuzzleHttp\Exception\GuzzleException;
use GuzzleHttp\Exception\RequestException;
use Psr\Log\LoggerInterface;

class TelegramApiRequestExecutor
{
    /**
     * First %s is bot token, Second %s is api method
     */
    protected const API_ENDPOINT_URL_TEMPLATE = 'https://api.telegram.org/bot%s/%s';
    protected const ACTION_SEND_MESSAGE = 'sendMessage';

    protected ?string $botToken;
    protected LoggerInterface $logger;

    /**
     * @param string $botToken
     */
    public function __construct(
        string $botToken,
        LoggerInterface $logger
    ) {
        $this->botToken = $botToken;
        $this->logger = $logger;
    }

    public function sendMessage(int $chatId, string $text, ?array $keyboard = null,
?bool $disableNotification = false): int
    {
        $messageBody = [
            'chat_id' => $chatId,
            'text' => $text,
            'parse_mode' => 'HTML',
            'disable_web_page_preview' => true,
        ];

        if ($disableNotification === true) {
            $messageBody['disable_notification'] = true;
```

```
        }

        if ($keyboard !== null) {
            $messageBody['reply_markup'] = empty($keyboard)
                ? ['remove_keyboard' => true]
                : [
                    'keyboard' => $keyboard,
                    'resize_keyboard' => true,
                ];
        }

        $responseCode = $this->executeRequest(
            sprintf(self::API_ENDPOINT_URL_TEMPLATE, $this->botToken,
self::ACTION_SEND_MESSAGE),
            'POST',
            $messageBody
        );

        return $responseCode;
    }

    public function sendForbiddenMessage(int $chatId): int
    {
        return $this->sendMessage(
            $chatId,
            'Ваш тариф - «Тест», доступ к разделу ограничен. Для полного доступа к
разделу проведите оплату в меню Финансы/Оплата.'
        );
    }

    /**
     * Make a request to the Telegram API endpoint.
     *
     * @param string $url
     * @param string $method
     * @param array $body
     *
     * @return int
     */
    private function executeRequest(
        string $url,
        string $method,
        array $body
    ): int {
        $client = new Client();

        try {
            $res = $client->request(
                $method,
                $url,
                [
                    'body' => json_encode($body),
                    'headers' => [
                        'Content-type' => 'application/json'
                    ]
                ]
            );

            return $res->getStatusCode();
        } catch (RequestException $requestException) {
            $this->logger->critical(
                "Telegram responded with status {$requestException->getResponse()-
>getStatusCode()}. It is unexpected.",
                [
```

```
                    'exception' => $requestException,
                    'body' => $body,
                ]
            );

            throw $requestException;
        } catch (GuzzleException $exception) {
            $this->logger->critical(
                $exception->getMessage(),
                [
                    'exception' => $exception,
                    'body' => $body,
                ],
            );

            throw $exception;
        }
    }
}
```

Клас WebhookRequestBodyProcessorFactory

```
<?php

namespace App\Service\Telegram;

use App\Service\Telegram\WebhookRequestBodyProcessor\LiveWebhookRequestBodyProcessor;
use
App\Service\Telegram\WebhookRequestBodyProcessor\WebhookRequestBodyProcessorInterface
;

class WebhookRequestBodyProcessorFactory
{
    public const MODE_LIVE_PROCESSING = 'live';
    public const MODE_QUEUE_PROCESSING = 'queue';

    private WebhookRequestBodyProcessorInterface $liveWebhookRequestBodyProcessor;

    public function __construct(
        LiveWebhookRequestBodyProcessor $liveWebhookRequestBodyProcessor
    ) {
        $this->liveWebhookRequestBodyProcessor = $liveWebhookRequestBodyProcessor;
    }

    public function createWebhookRequestBodyProcessor():
WebhookRequestBodyProcessorInterface
    {
        return $this->liveWebhookRequestBodyProcessor;
    }
}
```

Клас Kernel

```
<?php
```

```
namespace App;

use Symfony\Bundle\FrameworkBundle\Kernel\MicroKernelTrait;
use Symfony\Component\HttpKernel\Kernel as BaseKernel;

class Kernel extends BaseKernel
{
    use MicroKernelTrait;
}
```

Файл .env

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=bb01ad78ae5dcd28a2580153d8c3f942
###< symfony/framework-bundle ###
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7&ch
arset=utf8mb4"
DATABASE_URL="postgresql://root:secret@db:5432/report_bot?serverVersion=13&charset=ut
f8"
###< doctrine/doctrine-bundle ###

BOT_NAME=
BOT_TOKEN=
```

Файл composer.json

```
{
    "type": "project",
    "license": "proprietary",
    "minimum-stability": "stable",
    "prefer-stable": true,
    "require": {
        "php": ">=7.2.5",
        "ext-ctype": "*",
        "ext-iconv": "*",
        "ext-json": "*",
        "doctrine/annotations": "^1.0",
        "doctrine/doctrine-bundle": "^2.6",
        "doctrine/doctrine-migrations-bundle": "^3.2",
        "doctrine/orm": "^2.12",
        "guzzlehttp/guzzle": "^7.4",
        "phpdocumentor/reflection-docblock": "^5.3",
        "phpstan/phpdoc-parser": "^1.4",
        "ramsey/uuid": "^4.2",
        "sensio/framework-extra-bundle": "^6.2",
        "symfony/console": "5.4.*",
        "symfony/dotenv": "5.4.*",
        "symfony/flex": "^1.17|^2",
        "symfony/form": "5.4.*",
        "symfony/framework-bundle": "5.4.*",
        "symfony/property-access": "5.4.*",
        "symfony/property-info": "5.4.*",
        "symfony/proxy-manager-bridge": "5.4.*",
        "symfony/runtime": "5.4.*",
```

```
        "symfony/serializer": "5.4.*",
        "symfony/validator": "5.4.*",
        "symfony/yaml": "5.4.*"
    },

    "config": {
        "allow-plugins": {
            "composer/package-versions-deprecated": true,
            "symfony/flex": true,
            "symfony/runtime": true
        },

        "optimize-autoloader": true,
        "preferred-install": {
            "*": "dist"
        },
        "sort-packages": true
    },

    "autoload": {
        "psr-4": {
            "App\\": "src/"
        }
    },

    "autoload-dev": {
        "psr-4": {
            "App\\Tests\\": "tests/"
        }
    },

    "replace": {
        "symfony/polyfill-ctype": "*",
        "symfony/polyfill-iconv": "*",
        "symfony/polyfill-php72": "*"
    },

    "scripts": {
        "auto-scripts": {
            "cache:clear": "symfony-cmd",
            "assets:install %PUBLIC_DIR%": "symfony-cmd"
        },
        "post-install-cmd": [
            "@auto-scripts"
        ],
        "post-update-cmd": [
            "@auto-scripts"
        ]
    },

    "conflict": {
        "symfony/symfony": "*"
    },
    "extra": {
        "symfony": {
            "allow-contrib": false,
            "require": "5.4.*"
        }
    },
    "require-dev": {
        "doctrine/doctrine-fixtures-bundle": "^3.4",
        "symfony/maker-bundle": "^1.40"
    }
}
```

Файл docker/php/Dockerfile

```
FROM php:7.4-fpm

# Install Composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin
--filename=composer

ENV \
    OUR_PACKAGES="git \
    wget \
    vim \
    zip \
    unzip \
    libpq-dev \
    librabbitmq-dev \
    libssh-dev \
    supervisor"

# This updates the OS to latest state and cleans up to reduce image size
RUN     apt-get -yqq update && \
        apt-get install -yq --no-install-recommends ${OUR_PACKAGES} && \
        apt-get autoremove -y && \
        apt-get clean -y && \
        pecl install 'xdebug-2.9.8' amqp redis && \
        docker-php-ext-enable xdebug amqp redis

RUN docker-php-ext-install pdo pdo_pgsql json bcmath sockets

ENV TZ=Europe/Kiev
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
RUN printf '[PHP]\ndate.timezone = "${TZ}"\n' > /usr/local/etc/php/conf.d/tzone.ini

RUN rm -rf /var/lib/apt/lists/*

CMD ["/usr/bin/supervisord", "-c", "/etc/supervisor/supervisord.conf"]
```

ДОДАТОК Г

(обов’язків)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Кафедра інженерії програмного забезпечення

Дипломний проект

Програмна система збору та модерації результатів виконанних робіт на підприємстві текстильних виробів з використанням API Telegram

Виконав: Новацький Олег - студент групи ІПЗс-19-1

Керівник: Бедратюк Л. П. - д-р фіз.-мат. наук, професор

Тема цього дипломного проекту є об’єктивною потребою у модернізації бізнес-процесів на підприємствах малого бізнесу. А саме цехи і фабрики для виробництва текстильних виробів. На підприємствах відсутні засоби автоматизації обміну інформації між працівниками і керівництвом.

Актуальність

Проблеми

- Застарілі методи обміну інформації
- Складність опрацювання даних
- Зайві процеси на підприємстві
- Витрати надлишкового часу та ресурсів на тривіальні задачі

Мета проекту

Розробити програмну системи збору інформації про виконанні роботи на підприємстві текстильних виробів.

Система повинна:

- бути зручна, швидка та надійна у використанні
- автоматизувати робочі процесів для економії часу та ресурсів
- надати можливість модерації зібраних даних

Завдання проекту

Створити Telegram-бота, що буде виконувати такі функції:

- Реєстрація користувача в системі
- Відображення типів робіт
- Прийом результатів виконаних робіт
- Встановлення дати для звітності
- Відображення результатів

Завдання проекту

Створити адміністративну панель, що буде виконувати такі функції:

- Добавлення користувачів в систему
- Добавлення типів робіт в систему
- Моніторинг результатів виконанних робіт
- Генерація звітів

Пошук існуючих рішень

Customer Relationship Management

Недоліки та обмеження CRM

- Надлишкова функціональність
- Складні в розумінні
- Потребують тривалого навчання у використанні
- Дороговартісні
- Тривалий процес налаштування CRM для потреб певного бізнесу

Проектування

Трансформація взаємодії акторів

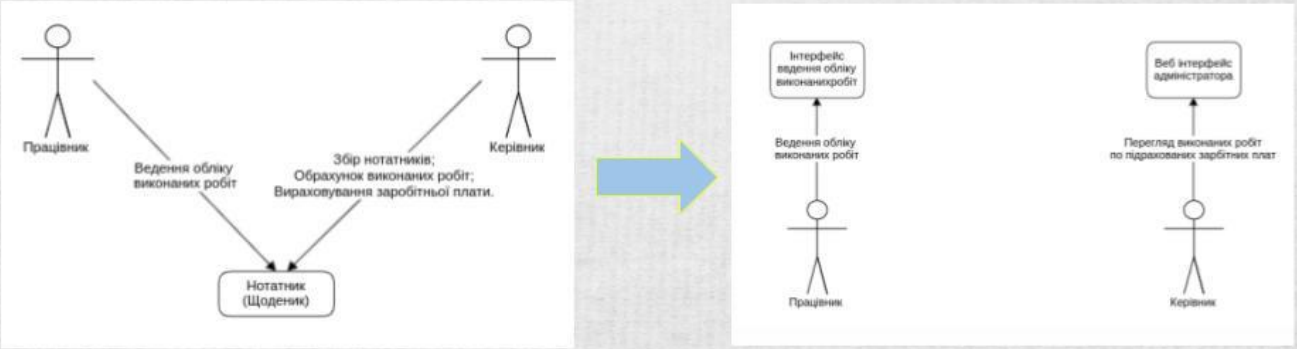
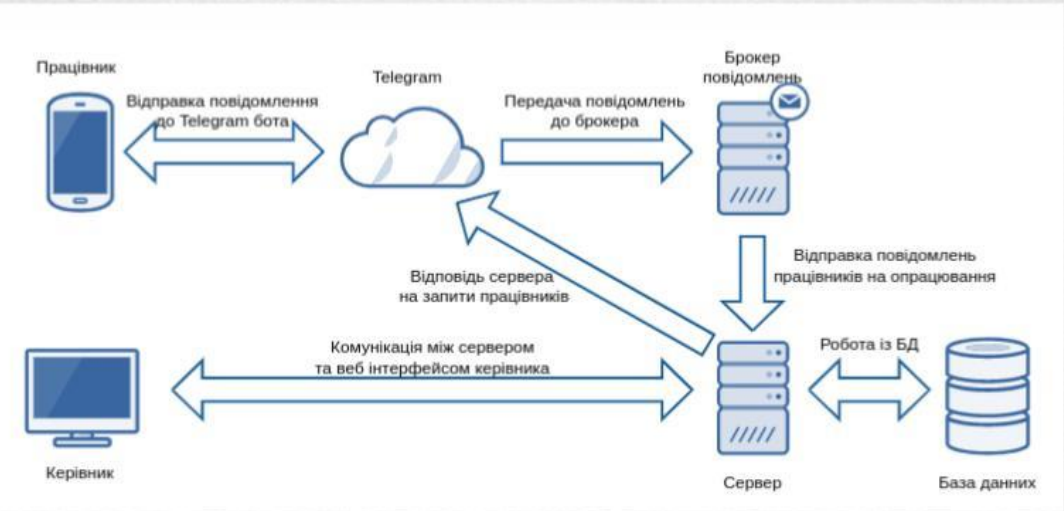
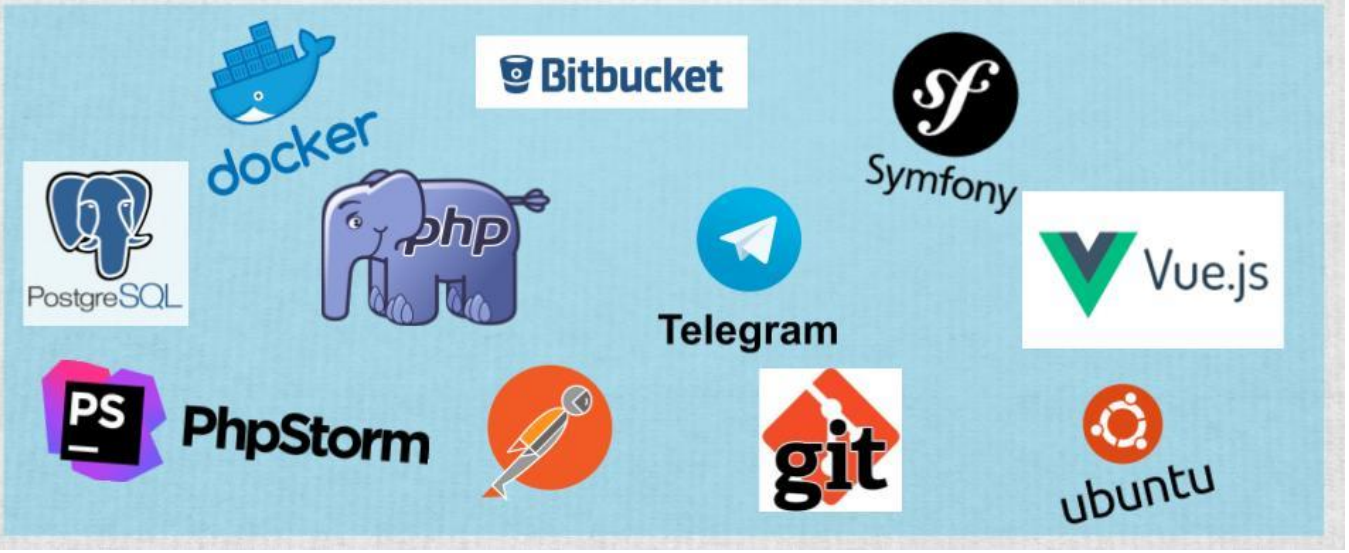


Схема взаємодії компонентів системи

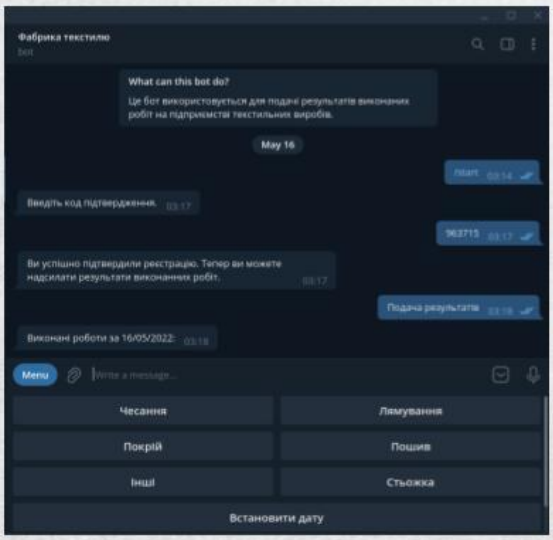


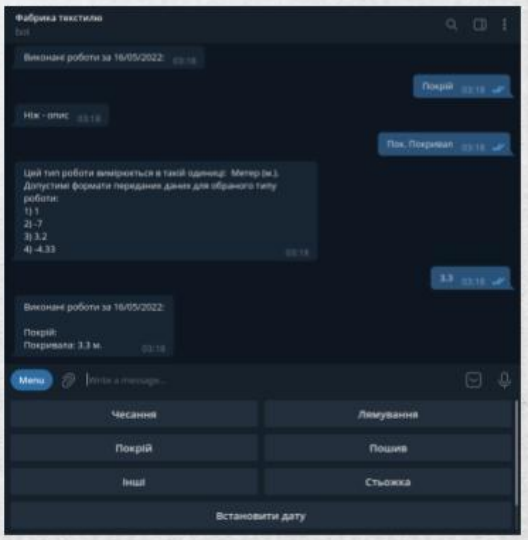
Засоби та інструменти для реалізації



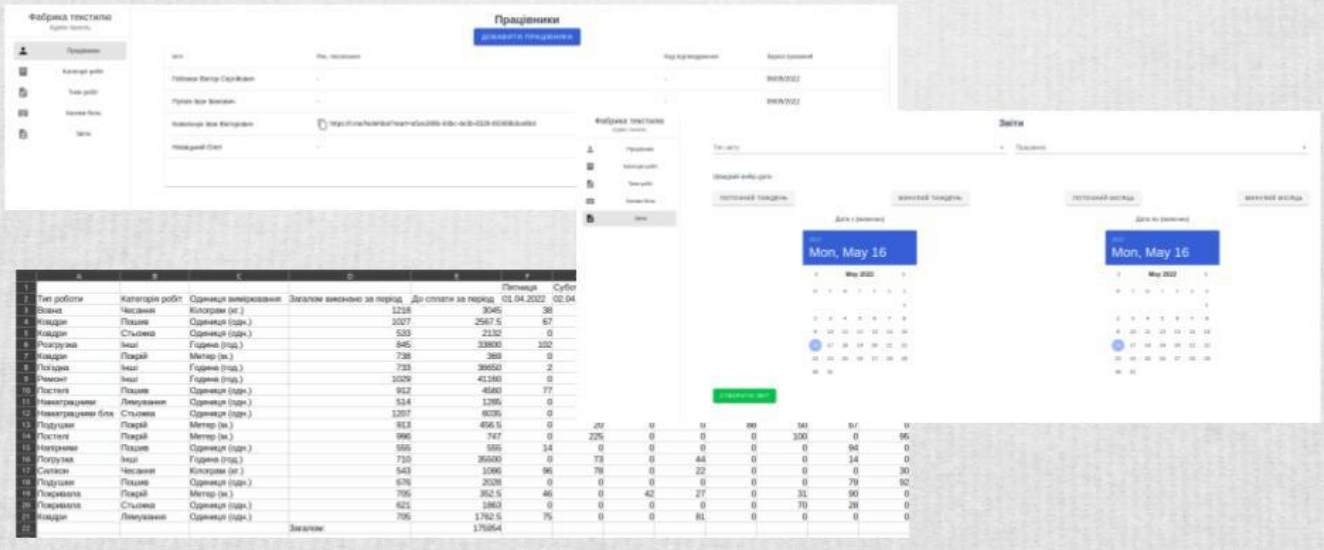
Результати виконаних робіт

Комунікація із ботом в Telegram





Використання адмін. частини



Висновки

Розроблено програмний продукт, що взаємодіє із ботом в Telegram. Ця взаємодія дозволяє працівникам підприємств зручним способом здійснювати звітність виконанних робіт.

Для власників бізнесу це є покращенням, яке зменшує витрати часу та ресурсів на прості, в цей самий час тривалі але тривіальні, задачі.

Дякую =)

За Вашу увагу :)

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Новацького О. В.
Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.02.2022
дата

підпис

Fri May 27 10:14:50 EEST 2022, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

	Документ		Сумарний збіг по Базі Даних	
ID: 104082 Назва: Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram Додано в БД: 2022-05-27 Автора: О.В. Новацький Керівники: Л.П.Бедратюк Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	83566	1328	3960 (5%)	60 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1011347851

Дата перевірки:
27.05.2022 11:11:31 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
27.05.2022 11:14:32 EEST

ID користувача:
100005589

Назва документа: **Пояснювальна_записка_ІПЗс_19_1_Новацький_О_В_резерв(без додатків)**

Кількість сторінок: **87** Кількість слів: **14929** Кількість символів: **116606** Розмір файлу: **3.16 MB** ID файлу: **1011233753**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.55%
Схожість

Найбільша схожість: **4.84%** з Інтернет-джерелом (http://eprints.cdu.edu.ua/1482/1/testyvan.pdf)

5.63% Джерела з Інтернету	25	Сторінка 89
1.31% Джерела з Бібліотеки	96	Сторінка 89

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування	20 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Новацький Олег Валерійович

Тема Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _0_; кількість сторінок записки _215_

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті було здійснено аналіз предметної області. Проаналізовано всі функціональні і нефункціональні вимоги до ПЗ. Було здійснено аналіз існуючих рішень на ринку, що можуть вирішити поставлені задачі, також розглянуто їхні переваги та недоліки. На основі цього аналізу було доведено актуальність розробки нового програмного забезпечення. На основі вимог, було спроектовано систему та виконано декомпозицію на модулі. Для реалізації цих модулів було проведено аналіз існуючих засобів та методів. По завершенню реалізації програмного забезпечення, було виконано тестування системи. За результатами тестування було доведено, що система працює справно та готова до експлуатації.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконаний відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуть існуючі рішення та визначено функціональні та нефункціональні вимоги до програмного забезпечення. У другому розділі було виконано проектування системи, визначено архітектурні підходи, що будуть використовуватися під час розробки, виконано декомпозицію системи та опис структури даних. Спроектовано та описано вимоги до користувацького інтерфейсу. У третьому розділі було визначено всі залежні модулі, для розроблюваної системи. Описано їх особливості і методи взаємодії із основним кодом програми. На основі виконаних дій, було створено новий програмний продукт. У четвертому розділі було виконане інтеграційне тестування системи. Всі тести виконувалися відповідно до функціональних вимог, в результатів цього було підтверджено керктну роботу створеного програмного забезпечення.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки розроблений програмний продукт забезпечить своїх користувачів зручним способом збору і обробки інформації. Цей спосіб є підходящим для підприємств малого бізнесу, де працюють люди із невеликим рівнем кваліфікації у використанні інформаційних технологій. Також для реалізації цього програмного продукту було використано найкращі архітектурні рішення та сучасні методи їх реалізації.

5. Негативні сторони проекту У інтерфейсі користувачів, що виконують роль працівника на підприємстві, було реалізовано можливість перегляду результатів за конкретні періоди часу. Було б доцільно добавити можливість вибирати будь-який необхідний період для отримання результатів.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломного проекту та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про дипломний проект в цілому Всі кроки реалізації дипломного проекту виконувалися відповідно до визначеної послідовності. Матеріал пояснювальної записки структурований, послідовний, чіткий. Це дозволяє чітко зрозуміти викладений матеріал у рамках проекту. Графічні матеріали дають можливість наочно побачити деталі проектування системи. На основу цих факторів, цей дипломний проект заслуговує позитивної оцінки.

8. Інші зауваження

9. Оцінка дипломного проекту Дипломний проект виконаний у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Муляр Ігор Володимирович, доцент, кафедра кібербезпеки

"27" травня 202 2 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КФАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:
Назва: «Програмна система збору та модерації результатів виконаних робіт на підприємстві текстильних виробів з використанням API Telegram»
Автор: Новацький Олег Валерійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідальність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту(наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломного проекту системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання на проектування, відомість документів, у структурі змісту, написах в рамках, назвах розділів/підрозділів тощо) та в назвах переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформлені посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлених збігів ідентичності схожості, складає 6.55% і адресується до 25 джерел, що з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту

Керівник _____ Л. П. Бедратюк

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк