Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Програмна система для продажу годинників з використанням API Telegram

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДПШЗ.170108.01.08.ПЗ

Виконав студент IV курсу група ІПЗ-17-1 _____ Я. Ю. Маєвський
Підпис Ініціали, прізвище

Керівник канд. тех. наук, доцент _____ І.В. Гурман
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Г. І. Радельчук
Підпис Ініціали, прізвище

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

4. 06 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
05 02 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Маєвському Ярославу Юрійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмна система для продажу годинників з використанням API Telegram
Керівник проекту (роботи) Гурман Іван Васильович
кандидат технічних наук, доцент
Затверджена наказом ректора університету від 05.02.2021 р. № 11
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2021 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмної системи
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 16 шт.)

6. Консультанти розділів дипломного проекту (роботи)

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

7. Дата видачі завдання «05» лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних тем ДП	01.12– 30.12.2020	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2021	
3 Проектування програмного забезпечення	01.02 – 28.02 2021	
4 Програмна реалізація	01.03 – 10.04.2021	
5 Тестування програмного забезпечення	11.04 – 30.04.2021	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2021	
7 Попередній захист ДП	травень 2021 (згідно графіка)	
8 Перевірка ДП на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2021	
9 Підготовка до захисту та захист ДП	з 01.06.2021	

Студент _____ Я. Ю. Маєвський
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ І. В. Гурман
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: «Програмна система для продажу годинників з використанням API Telegram».

Автор проекту: Маєвський Ярослав Юрійович.

Керівник проекту: Гурман Іван Васильович.

Пояснювальна записка: 132 с., 36 рис., 3 табл., 4 дод., 11 джерел.

Графічна частина: 16 презентаційних слайдів.

Метою даного проекту є реалізація програмного забезпечення, яке дозволить оптимізувати і автоматизувати продаж товарів серед користувачів магазину, забезпечить зручний інтерфейс використовуючи API Telegram.

Об'єкт дослідження. Об'єктом дослідження даної дипломної роботи є процес використання чат-ботів в месенджерах для рішення комерційних проблем.

У дипломному проекті проведено аналіз предметної області та її інформаційного забезпечення, визначені вимоги до системи, спроектована загальна архітектура додатку, серверний клієнт на основі мікросервісів, користувацький інтерфейс, структура бази даних.

Для розробки програмної системи використано мову програмування Java, фреймворк Spring, та середовище розробки Intellij Idea, база даних Mysql, Mongo.

В результаті проектування і реалізації програмного забезпечення було побудовано комплексну програмну систему для продажу годинників з інтерфейсом у вигляді Telegram чат-бота.

1.06.2021 р.
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ.170108.01.08.ПЗ	Пояснювальна записка	132		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні	16		
			слайди			

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.170108.01.08.ПЗ		
Виконав		Мягський Я. Ю.		1.06	Програмна система для продажу годинників з використанням API Telegram	Літ.	Арк. 1 / Аркушів 1
Керівник		Гурмон І.В.		2.06			
Н. контр.		Радельчук Г. І.		3.06	Відомість документів	ХНУ, ІПЗ-17-1	
Зав. каф.		Бедротюк Л.П.		4.06			

ЗМІСТ

					ДПШЗ.170108.01.08.ПЗ	
Змн.	Арк.	№ докум.	Підпис	Дата		
Виконав		Маєвський Я.Ю.		1.06	Програмна система для продажу годинників з використанням API Telegram	Літ. / Арк. 4 / Аркушів 132
Керівник		Гурман І.В.		2.06		
Н. контр.		Радельчук Г. І.		3.06	Пояснювальна записка	ХНУ, ІПЗ-17-1
Зав. каф.		Бедратюк Л.П.		4.06		

ВСТУП

В наші дні сучасне суспільство не можливо уявити без засобів зв’язку. У кожної людини завжди є з собою мобільний телефон, планшет, смарт-часи, на яких встановленні засоби комунікації.

Більшість свого вільного часу люди проводять, використовуючи свої гаджети. Люди читають новини, дивляться цікаві відеоролики або просто смішні картинки. Також вони спілкуються мж собою за допомогою соціальних мереж і месенджерів. Месенджер – програмний засіб, з допомогою якого користувачі можуть обмінюватись текстовими повідомленнями або іншою інформацією представленню в альтернативному вигляді, в реальному часу. З часом, месенджери еволюціонували з простого інструмента обміну повідомлення в неймовірно потужні комерційні інструменти, в яких люди проводять більше часу, а ніж на будь-яких інших інтернет ресурсах.

Важливу роль відіграли в цьому боти – спеціальні програми, що виконує автоматично або за певним розкладом запрограмовані дії. Програми, які працюють в месенджері. Така програма може відповідати на запитання, а також самостійно задавати їх. Чат-боти в комерції сильно зменшують витрати, покращують конверсію, і покращують обслуговування клієнтів. Сотні компаній роблять все можливе, щоб досягти вищезгаданих цілей, і для їх досягнення необхідні боти як пріоритетний інструмент. На даний момент існує не велика кількість їх варіацій – від звичайних чат-ботів до систем розсилки певної інформації до користувача.

Звісно таким корисним інструментом цікавляться власники бізнесу, які зацікавлені залишатись поруч з їх споживачем. І хоча боти уже декілька років активно використовуються закордоном в великих підприємствах як Microsoft чи Amazon, в Україні вони не так поширені та тільки набирають свою популярність, а отже є перспективним та досить актуальним рішенням для розробників та власників бізнесу. Слід мати на увазі, що для потенційних користувачів чат-ботів

є досвід користувача при роботі з ботом, який включає те як бот сприймає ввід користувача, наскільки добре він виділяє суть запиту користувача та наскільки зрозумілі та доречні його відповіді. Всі ці аспекти формують враження від бота та відіграють важливу роль в подальшому використанні його та рекомендації іншим користувачам.

Інтернет-магазин, що відповідає всім сучасним вимогам, є торговим каналом, за допомогою якого підприємство має можливість продавати свої товари та послуги багатомільйонної аудиторії, що користується послугами інтернет мережі. У сучасних умовах створення інтернет-магазину набуває все більшої актуальності. Це дозволяє компанії зміцнювати свої позиції не тільки на традиційних ринках, а й виходити далеко за їх межі.

Актуальність теми полягає в тому, що на даний момент в мережі інтернет є безліч сайтів, платформ, сервісів для покупки товарів, про те в Україні месенджери як інструмент для продажу товарів не є популярним і розвинутим. Інтеграція бізнесу в межі чат-бота покращить взаємодію підприємства з користувачем, так як месенджери є майже у всіх користувачів на пристроях.

Також актуальністю розробки чат-ботів є те, що реалізація і підтримка чат-ботів є набагато дешевшою, чим створення і підтримка веб-сервісів, що дозволить клієнту зменшити витрати на інтернет магазин.

Телеграм обрано як месенджер для чат-боту, тому що він самий популярний в країнах СНД, активно розвивається, крос платформний, і надає саме зручне API для взаємодії і контролю ботів.

Метою проекту є реалізація програмного забезпечення, яке дозволить оптимізувати і автоматизувати продаж товарів серед користувачів ПЗ, також забезпечить швидкий та зручний інтерфейс використовуючи API Telegram.

Для реалізації веб-додатку необхідно виконати наступні завдання:

– детально розглянути особливості та визначити специфіку предметної області сучасних месенджерів;

– проаналізувати використання реляційних, нереляційних та графових баз даних при реалізації серверної сторони додатку;

– виконати повне порівняння актуальних серверних архітектур, реалізувати серверну частину програмного забезпечення;

– провести аналіз способів реалізації програмного забезпечення, виявити основні проблеми та знайти вихід для їх вирішення;

– встановити особливості та загальні правила для побудови користувацького інтерфейсу та реалізувати клієнтську частину ПЗ;

– провести аналіз програмних модулів та інструментів контейнеризації та оркестровки сервісів програмного забезпечення;

– провести модульне, функціональне та інтеграційне тестування.

Результатом виконання дипломного проекту є функціонуючий серверний додаток, який надає клієнтам можливість користуватись магазином годинників з будь-якого пристрою за допомогою клієнта Telegram бота.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

На даний момент чат-боти можуть приймати замовлення (1-800-Flowers, Pizza Express, TacoBot), бронювати квитки на літак (Skyscanner), пропонувати поради щодо покупки (Sephora, H&M) та надавати оновлення балансу рахунку в реальному часі (Bank of America). Чат-боти також допомагають користувачам робити покупки (Spring), дозволяють друкувати документи (HP), навіть з'єднувати їх із справжніми лікарями (HealthTap).

З ростом ринку чат-ботів віртуальні помічники постійно вдосконалюються. Вони стають надійнішими та людянішими, все більше і більше людей ними користуються. Згідно з дослідженнями VentureBeat, середній рівень утримання чат-ботів дорівнює 4%. Вони також виявили, що чат-бот із 8% коефіцієнтом залучення можна вважати успішним.

Що стосується мобільних додатків, Localytics, агентство з маркетингових досліджень, виявило, що 43% користувачів все ще користуються програмами через місяць після завантаження. Чат-боти в основному використовуються як швидке рішення простих проблем, таких як необхідність перекладу пари слів. Наприклад, Instant Translator Bot допомагає користувачам перекладати з та на 19 мов. Він має понад 50 тис. щоденних користувачів у Facebook та Viber та понад 600 тис. активних користувачів щомісяця. Один з найвідоміших додатків для мобільних перекладів, який називається iTranslate, має 6 мільйонів активних користувачів щомісяця. Додатки мають свої плюси, з якими важко посперечатися. Але чат-боти точно замінять деякі програми.

Чат-бот – це програмний інструмент, який взаємодіє з користувачами на певну тему або в певній області природним, розмовним способом, використовуючи текст та голос. Для багатьох цілей чат-боти використовуються

у різних областях, включаючи маркетинг, обслуговування клієнтів, технічну підтримку, а також освіту та навчання. Сучасні розробки в цій галузі говорять про те, що взаємодія з технологіями природною річчю цілком можлива, оскільки технологія розвивається, і користувачі звикають до взаємодії з цифровими сутностями. Замість того, щоб створити схожий на людину розумний машинний додаток, мова йде про створення ефективних цифрових помічників, здатних надати інформацію, відповісти на запитання, обговорити певну тему або виконати певне завдання.

Персональні цифрові помічники, такі як Siri від Apple, Alexa від Amazon, Cortana від Microsoft або Assistant від Google, займають передові позиції в технології розпізнавання голосу та штучного інтелекту. Ці цифрові асистенти використовують методи машинного навчання і здатні керувати деякими повсякденними завданнями традиційних помічників або секретарів (наприклад, встановлення пріоритетів електронної пошти, виділення найважливішого вмісту та взаємодії з ним), щоб допомогти своїм користувачам стати ефективнішими. Велика кількість простих і більш конкретних областей на основі текстових чат-ботів може виконувати такі функції як зв’язок з підтримкою того чи іншого продукту, або форма для залишення відгуків, розповсюдження вмісту для публікації сайтів, бронювання номера в готелі, бронювання ресторанів, тощо. Текст боти зазвичай відповідають набору встановлених правил або послідовностей, щоб відповісти на запитання опубліковані користувачем. Ці правила або послідовності дозволяють їм ефективно реагувати на запити в межах певного домену, але не ефективно відповідати на запитання, шаблон яких не відповідає правилам, на яких навчається чат-бот.

Додатки чат-ботів існують вже давно, визначними прикладами є ELIZA, ALICE, Claude та HeX. ELIZA була першим у світі чат-ботом. Розроблена Джозефом Вайценбаумом у 1956р. ELIZA могла емулювати відповіді психотерапевта, та мала певну базу знань у цій сфері. Приклад роботи наведений на рисунку 1.1.

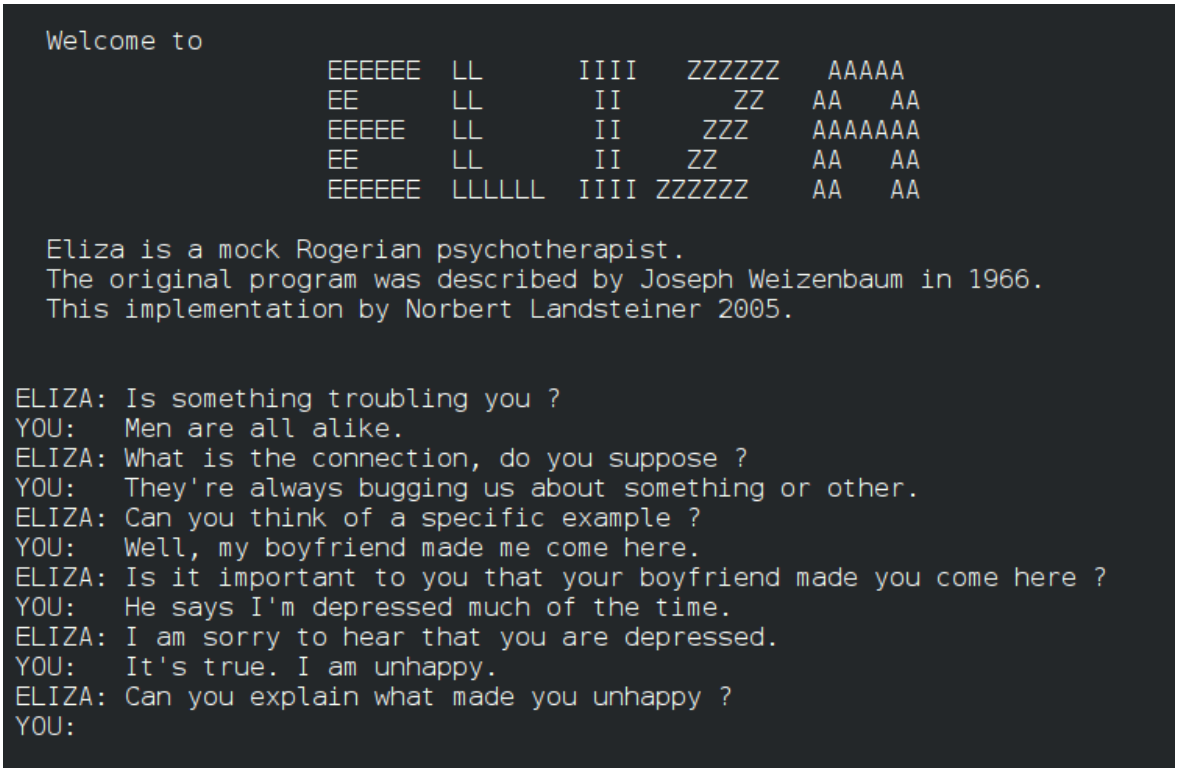


Рисунок 1.1 – ELIZA перший у світі чат-бот

ALICE (Штучний лінгвістичний Інтернет-комп'ютер) – це розроблений Річардом С. Уоллесом чат-бот, який фокусувався на обробці природної речі. У ньому використовується мова розмітки штучного інтелекту, (AIML) яка відповідає за узгодження шаблонів та пов’язує введені користувачем дані із понад 40 000 записами. Деякі сучасні чат-боти навіть сьогодні використовують систему ALICE. В свою чергу Брайан Маклафлін розробив чат-бот під назвою Claude, який використовує стандартну відповідність шаблонів для пошуку відповідної відповіді. Claude розпізнає ввід користувача, потім розробляє відповідь на основі цього вводу, використовуючи записи у своїй базі даних, а потім створює відповідь. Джейсон Хатченс розробив чат-бот HeX, який міг не тільки давати відповідь на питання, а й також задавати нову тему розмови.

Чат-боти мали давню історію використання як педагогічних помічників в освітніх умовах. З початку 1970-х років були розроблені педагогічні помічники в середовищі цифрового навчання, відомі як «Інтелектуальні системи навчання».

Розмовні педагогічні помічники використовують методи штучного інтелекту для покращення та більшої персоналізації навчання.

Досягнення у сфері архітектури та дослідження зіграли важливу роль у розробці привабливих, корисних та цінних педагогічних засобів, які не тільки включають в себе технічні аспекти, але й розуміють емоційні, когнітивні та соціальні освітні проблеми. Крім того, розмовні агенти вбудовані в програмне забезпечення та пристрої. Більш того, в останні роки все більше і більше організацій почали використовувати можливості, що виходять за рамки простого запиту інформації з подальшою запрограмованою відповіддю.

Включення чат-ботів в освітню сферу за останнє десятиліття означає збільшення інтересу до способів застосування чат-ботів для навчання та розвитку. Системи чат-ботів можуть забезпечити переваги миттєвої доступності та можливості природного реагування за допомогою розмовного інтерфейсу, які вже важко відрізнити від реальної співбесіди.

Крім того, чат-боти демонструють здатність створювати чітку взаємодію з користувачами, яка сприяє більшій зацікавленості, та постановці вищих цілей, розробці стратегій навчання та розвитку.

Мобільні пристрої пропонують широкий спектр видів спільної роботи, спілкування та навчання. Однак молоде покоління не розрізняє корисні інструменти між мобільними пристроями для соціальних мереж. Перевагами мобільних пристроїв було визначено доступ до мультимедійного вмісту, портативність, гнучкість та можливість негайного пошуку інформації.

До недоліків мобільних пристроїв можна віднести читання навчального вмісту на малому екрані, відсутність фокусу та ефективності уваги, технологічні проблеми (час автономної роботи, підключення) або проблеми сумісності пристроїв. Дослідження в галузі мобільних повідомлень обміну миттєвими повідомленнями зосереджувались на використанні мобільного засобу обміну миттєвими повідомленнями WhatsApp для підтримки навчання та здійснення покупок. Учасники експерименту продемонстрували позитивне сприйняття та

прийняття використання мобільного засобу обміну миттєвими повідомленнями для навчання та покупок.

Сьогодні вибір чат-ботів достатньо широкий. Чат-боти не є єдиною категорією, але вони охоплюють ширший спектр. Ми пропонуємо наступну класифікацію за введенням або за каналами обміну повідомленнями.

За введенням:

– за допомогою кнопок. Рішення дерева ієрархії представляються користувачеві у вигляді кнопок;

– за допомогою ключових слів. Користувач вводить фразу або просто слово, і чат-бот відповідає попередньо завантаженою відповіддю;

– контекстний ввід. використання машинного навчання і штучного інтелекту для самовдосконалення на основі того запиту користувача, і того як вони це виконують та відображають;

– голосовий ввід. введення даних за допомогою голосу. чат-бот за допомогою розпізнавання голосу відповідає на запити користувача або виконує креативні задачі.

По каналах обміну повідомленнями:

а) автономний додаток:

1) для персональних комп’ютерів (приклад: Briana Virtual Assistant);

2) мобільний (приклади: програми Andy English, DoNotPay або Replika);

б) веб-сервіс:

1) інтегрований в Інтернеті (приклади: спливаюче вікно з допомогою служби обслуговування клієнтів);

2) індивідуальний (приклади: Mitsuku, Cleverbot);

в) інтегрований додаток:

1) програми обміну миттєвими повідомленнями (приклади: Facebook Messenger, WhatsApp, WeChat, Skype);

2) платформа для спілкування та співпраці (приклади: Slack, Microsoft Teams, Cisco Webex Teams).

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Від веб-сайтів до мобільних додатків до соціальних мереж люди спілкуються з ботами на всіляких платформах. Зокрема, програми обміну миттєвими повідомленнями є одними з найпоширеніших місць для розгортання чат-ботів, особливо коли нам необхідно вибрати програму обміну повідомленнями, яку вже використовує наша передбачувана аудиторія [4].

Кожен додаток для обміну повідомленнями пропонує унікальні переваги та недоліки. Додатки чат-ботів, які досягають високої активності в одній програмі, можуть демонструвати набагато більші показники в іншій.

Щоб наш чат-бот мав високу активність, нам потрібно розгорнути його в потрібній програмі обміну повідомленнями. Давайте розглянемо найбільш популярні платформи для розгортання чат-ботів, та їх приклади.

а) Чат-бот Facebook Messenger

Що стосується активних користувачів щомісяця, Facebook Messenger є другим в світі додатком для обміну повідомленнями. У США та Канаді він займає перше місце. Само собою зрозуміло, що програма обміну повідомленнями з понад 1,3 мільярда користувачів у всьому світі є чудовим місцем для розгортання багатьох видів чат-ботів.

1) Особливості, плюси та мінуси Facebook Messenger

Окрім величезної кількості користувачів, Facebook Messenger включає різноманітний набір функцій для розробників ботів. Для початку є вбудовані засоби обробки платежів для електронної комерції (наприклад користувачі можуть використати бота для того щоб замовити піцу або придбати новий годинник) та аналітику, яка надає інформацію про ефективність роботи ботів на низькому рівні. Інтернет-продавці можуть скористатися перевагами оголошень, які доступні для натискання, що дозволяє вашій цільовій аудиторії відразу розпочати спілкування з вашим ботом.

Facebook Messenger також включає в себе кілька унікальних переваг, таких як короткі URL-адреси, які користувачі легко запам'ятовують, та коди, які користувачі можуть сканувати, щоб перейти безпосередньо до нашого бота.

2) Найкращі фреймворки для використання

Існує багато способів розробки ботів для Facebook Messenger. Якщо ви новачок у створенні ботів і не хочете - або не можете - мати справу з кодом, Chatfuel - чудовий інструмент для базової розробки. У ньому є все необхідне для створення базового бота Facebook Messenger та його запуску.

Chatfuel, являє собою графічний інструмент, і можливо, це не "справжній" фреймворк чат-ботів. Wit.ai є пропрієтарним фреймворком Facebook, і оснащений можливостями обробки природних мов (NLP) та машинного навчання. Він також являється безкоштовним, тому ми можемо створити свого бота та розгорнути його у Facebook Messenger, не платячи ні копійки.

Інші фреймворки з готовими до використання інструментами розгортання Facebook Messenger включають в себе Amazon Lex, Microsoft Bot Framework, Diasoutflow і Botkit.

3) Приклади чат-ботів Facebook Messenger

Великі та малі організації використовують Facebook Messenger для взаємодії людини та бота. Розглянемо бота під назвою TechCrunch, який може надсилати запити на зразок: "Що говорять про програми для спільної організації поїздок цього тижня?" Після такого питання, бот відображає останні новини TechCrunch на цю тему.

Прикладом електронного комерційного боту на Facebook Messenger може виступати 1-800-Flowers. Чат-бот 1-800-Flowers достатньо гнучкий. Він може запитувати адреси доставки, показувати користувачам конкретні квіткові композиції та приймати замовлення на доставку - безпосередньо з екрану чату. Приклад роботи чат боту наведено на рисунку 1.2.

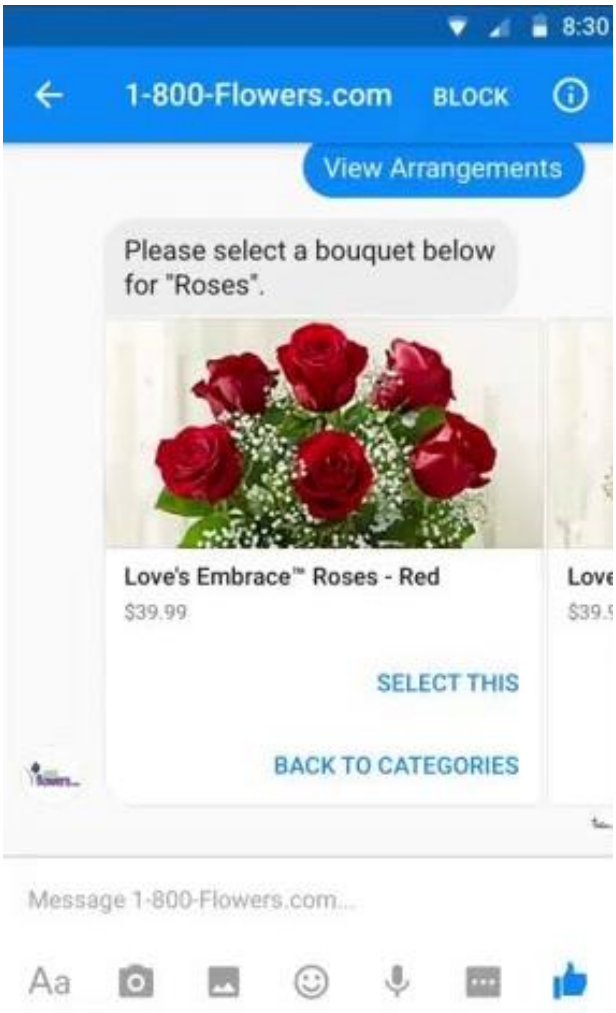


Рисунок 1.2 – Приклад роботи чат боту 1-800-Flowers

Якщо нам необхідно надіслати або переказати гроші бізнесу чи фізичній особі за межами країни, то бот RemitRadar Messenger надає миттєві розміри комісії плати за переказ (залежно від місця, куди ми надсилаємо, та валюти, яку ми використовуємо) та допомагає ініціювати транзакцію.

б) Чат-бот WhatsApp

WhatsApp - це найпоширеніша програма обміну повідомленнями у світі. Це також найпопулярніший додаток у своєму роді в Центральній та Південній Америці, а також частинах Африки, Азії та Європи.

1) Особливості, плюси та мінуси WhatsApp

По-перше, WhatsApp, ймовірно ще не стане платформою для розгортання складних ботів, здатних виконувати різні функції. Це тому що WhatsApp відносний новачок в світі ботів, і вони повільно рухаються у цьому напрямку.

API для WhatsApp Business, який використовується для розгортання бота на WhatsApp, став доступним через довгий час після того, як Messenger, Kik, Slack та інші ввели чат-боти на своїх платформах. Більше того, API можуть використовувати лише великі та середні підприємства. Кожен повинен подати заявку на розміщення бота на WhatsApp, і ви не отримаєте схвалення доки WhatsApp не вирішить підходите ви чи ні.

Отримавши схвалення, ви отримуєте необмежений доступ до набору інструментів розробника, що включає шаблони повідомлень та інтеграцію з вашими API для покупок, бронювання та інших транзакцій. Однак, окрім цих досить простих дій, WhatsApp не підтримує багаторівневий діалог як наприклад Facebook Messenger та інші.

Якщо ви займаєтеся малим бізнесом, то ви обмежені додатком WhatsApp Business. Відверто кажучи, це не платформа чат-ботів. Окрім автоматизованих миттєвих повідомлень та привітань, це людський чат, який спростили за допомогою простої автоматизації.

2) Кращі фреймворки для використання

Gupshup - це єдиний фреймворк, який має спеціальну інтеграцію WhatsApp для бізнесу. Його зручний інструмент скриптів Gupshup дозволяє легко інтегрувати свого бота на платформу.

Звичайно, якщо ви віддаєте перевагу йти альтернативним шляхом, є обхідні шляхи. Розробники RedBus, сервісу бронювання автобусів, фактично використовували фреймворк Google Diasoutflow для створення бота, який потім інтегрували з WhatsApp, використовуючи різні інструменти. Інший розробник побудував WhatsApp-бота за 30 хвилин, використовуючи Twilio API для WhatsApp і DuckDuckGo Instant Answer API. В результаті вийшов простий бот, який відображає інформацію з Вікіпедії у відповідь на запит користувача. Для організацій, яким вже дозволено використовувати API WhatsApp Business, Botsociety пропонує набір інструментів, який допоможе створити чат-бота для месенджера WhatsApp.

3) Приклади чат-ботів WhatsApp

Бот KLM для WhatsApp може бути використаний для отримання підтверджень рейсів, сповіщень про реєстрацію та оновлення статусу рейсу. Ви також можете отримати посадковий талон безпосередньо на WhatsApp. Якщо необхідно поговорити з агентом підтримки, ви можете скористатися ботом WhatsApp, для отримання допомоги. Приклад роботи чат-боту для замовлення квитків наведено на рисунку 1.3.

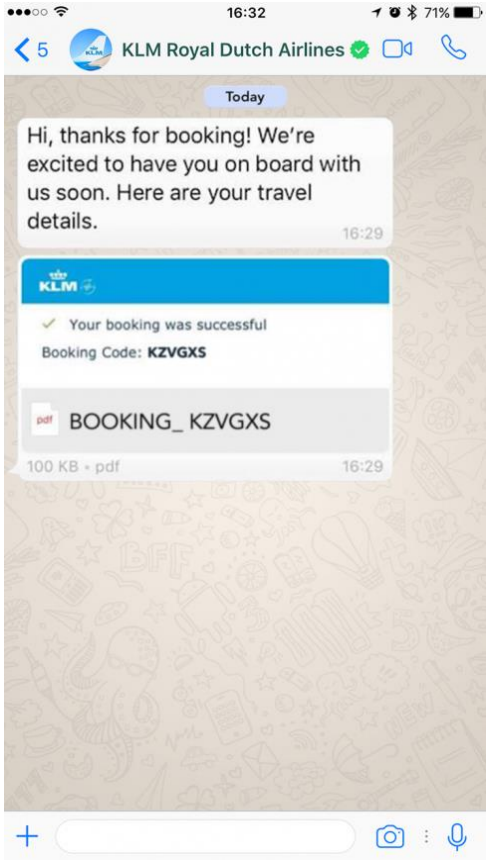


Рисунок 1.3 – Приклад роботи чат-боту KLM

Індонезійський банк BRI використовує WhatsApp для надання клієнтам особистого помічника. Бот банку Sabrina показує клієнтам, як знайти сусіднє відділення або банкомат, і навіть може подавати скарги. Раніше доступний у Facebook Messenger і Telegram, банк BRI використовує WhatsApp Business API для зв'язку з клієнтами на платформі, яку вони вже використовують.

в) Чат-бот Viber

Viber належить токійському Rakuten, та має глобальну розповсюдженість, і багато великих брендів почали використовувати його для зв’язку з клієнтами. Хоча Viber стверджує, що у світі є близько одного мільярда унікальних користувачів, деякі сторонні дослідження говорять про те, що більш імовірними є 260 мільйонів.

Незалежно від цього, Viber - це широко використовуваний додаток для обміну повідомленнями. Особливо поширений в Центральній та Східній Європі, Росії, Південно-Східній Азії, Близькому Сході та Океанії.

1) Особливості, плюси та мінуси Viber

Якщо ви націлені на ринки, де домінує WhatsApp, але не в захваті від його обмежених можливостей для чат-ботів, подумайте про розгортання свого чат бота у месенджері Viber.

По-перше, нам не потрібно бути великим або середнім бізнесом або проходити процес подачі заявок. Ми можемо зареєструватися і почати створювати свого бота відразу же. Тим не менш, є кілька рекомендацій, пов'язаних з ботом від Viber, які нам необхідно враховувати його при розробці:

– можливість відповідати на всі типи повідомлень Viber, включаючи текст, наклейки і зображення.

– швидкий час відгуку (менше п'яти секунд)

– вітання всіх користувачів, які запускають бота.

– забезпечення того, щоб наш бот функціонував так як він має функціонувати при різних умовах.

Найбільшими перевагами Viber є широкі інструменти для розробників, включаючи частини з відкритим кодом Viber API Community, які можуть допомогти нам зв’язатися з користувачами на тих самих ринках, де WhatsApp не зможе. Документація також включає довідкові матеріали для API ботів Viber Node.js, API Python Bot та REST, які можуть пришвидшити процес розгортання. Серед мінусів Viber можна сказати що поки що у нього не так багато користувачів як у Messenger або WhatsApp.

2) Кращі фреймворки для використання

Diasoutflow – це єдиний великий фреймворк з документацією, який крок за кроком проведе вас через інтеграцію Viber. Це не означає, що ви не можете використовувати інший фреймворки. Це означає, що з Diasoutflow це можна зробити трохи простіше, і налаштувати його відразу перед розгортанням.

Отримати бота готового бота побудованого на Diasoutflow для Viber дуже просто. Все, що нам потрібно зробити, це перейти в область інтеграції Diasoutflow, натиснути Viber, та ввести дані в кілька полів.

Найважливіше, що потрібно пам'ятати про інтеграцію фреймворків, - це те, що нам не потрібно використовувати фреймворк, створений з урахуванням додатка. Наприклад, Amazon Lex може інтегруватися з будь-яким сервісом обміну повідомленнями. Можливо для правильної працездатності нам доведеться використовувати кілька допоміжних інструментів і API. Інші розробники використовували Microsoft Bot Framework і Microsoft Azure для створення ботів, які в кінцевому підсумку були розгорнуті в Viber.

3) Приклади чат-ботів Viber

Боти Viber виконують різні функції, і аудиторія дуже сильно зміщується у бік споживчого/соціального (think Messenger) та професійного/продуктивного (think Slack) сторін. Наприклад, ви можете використовувати бота Tech Talk для пошуку статей на задані технічні теми. Можна просто ввести простий запит по типу «відеокарта», і бот покаже вам останні історії, пов'язані з цією темою. Міса – це ще один інформаційний чат-бот Viber. Приклад роботи чат-боту наведено на рисунку 1.4. Якщо ви знаходитесь в певному місті і шукаєте, скажімо, магазин сендвічів, Міса покаже вам правильний напрямок. Чат боти від Viber дуже схожі до Telegram ботів, але між ними є велика різниця в API. Боти Viber гірші в підтримці і налаштуванні ніж конкуренти від Telegram. Також важливим недоліком Viber є те, що вони не синхронізують інформацію між клієнтами автоматично в хмарі, як це вже реалізовано в конкурентів від Telegram чи Whatsup, для цього користувачу потрібно робити це самому.

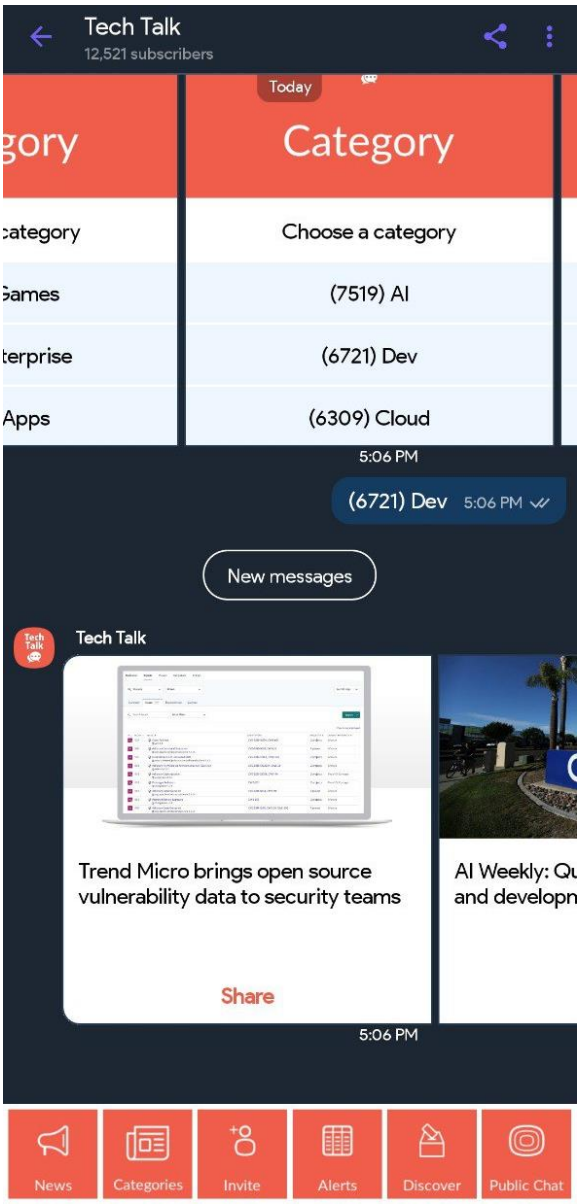


Рисунок 1.4 – Приклад роботи чат-боту Tech Talk

Щоб побачити, як великі бренди використовують Viber, можна звернутись до прикладу чат-бота Glamour. Glamour Viber bot ділиться новим контентом зі своєю російською аудиторією, в тому числі тим, що, на думку бота, може бути привабливим для конкретних користувачів. За даними Viber, ця кампанія призвела до 30-відсоткової конверсії.

г) Telegram чат-бот

На своїй сторінці найчастіших запитань Telegram включає в себе наступне питання: "Чим Telegram відрізняється від WhatsApp?" Відповідь дуже обширна, але основна її частина включає у себе API програми. API Telegram відкритий для

всіх, і в той же час API WhatsApp - ні.Telegram безпечніший, так як надає закрите шифрування інформації, в той час як WhatsApp використовує звичайну передачу.

1) Особливості, плюси та мінуси Telegram

Спочатку запущений як безпечна альтернатива WhatsApp (Telegram пропонує наскрізне шифрування), додаток швидко стало популярним у Росії, яка згодом спробувала заборонити його, але безуспішно. І хоча деякі все ще сумніваються в тому , чи дійсно дані Telegram безпечні, користувачі по всьому світу покладаються на цей додаток для не для обміну повідомленнями, а для обміну потенційно конфіденційними файлами.

Для розробників ботів сприйняття Telegram як безпечного програми для обміну повідомленнями означає, що це може бути гарна платформа для ботів які обробляють банківську інформацію, інтегруються з криптовалютними гаманцями або відправляють гроші. Дійсно, в документації Telegram bot підкреслюється проста інтеграція його платіжної платформи, а також підтримка вбудованих запитів, ігрових ботів HTML5 і різних кнопок для простих відповідей та взаємодій.

Як і у випадку з іншими менш широко використовуваними платформами, такими як Viber і WhatsApp, єдиною реальною перешкодою для розробки Telegram-бота є його впровадження. Facebook Messenger і WhatsApp мають набагато більше потенційних користувачів, ніж Telegram. Двісті мільйонів користувачів, є досить великим числом, але інші платформи можуть похвалитися більше ніж одним мільярдом користувачів.

2) Кращі фреймворки для використання

Серед основних фреймворків тільки Diasoutflow пропонує просту інтеграцію з Telegram в рамках своїх основних функцій. Звичайно, ми можемо використовувати будь який фреймворк, щоб створити бота і розгорнути його в Telegram. Наприклад, інструмент botpress-telegram дозволяє легко побудувати бота на платформі з відкритим вихідним кодом Botpress і відразу запустити бота на платформі Telegram.

3) Приклади чат-ботів Telegram

У Telegram є всі види ботів, але найбільш поширенішими порівняно з іншими платформами [9] є боти які пов'язані з платежами, криптовалютами і безпечними комунікаціями. Візьмемо наприклад, XirkleBot, який одночасно є криптовалютним гаманцем, і ботом який дозволяє відправляти і отримувати біткоїн, ефириум та інші цифрові валюти. Приклад роботи чат-боту XirkleBot наведено на рисунку 1.5.

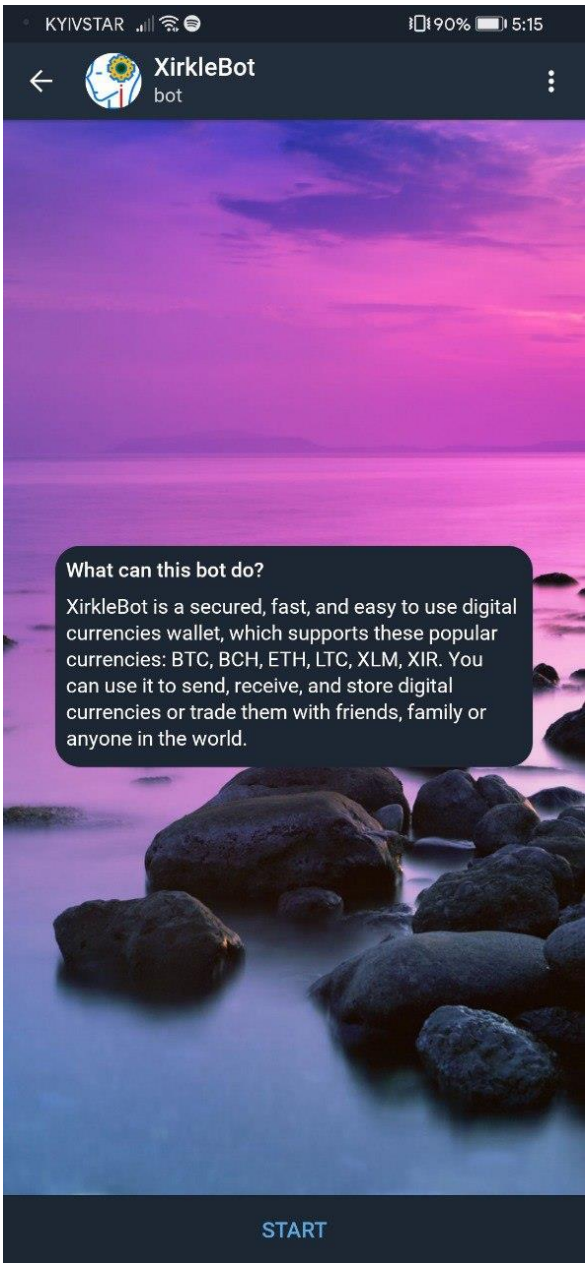


Рисунок 1.5 – Приклад роботи чат-боту XirkleBot

Щодо великих компаній, які використовують платформу Telegram як інструмент для безпечних транзакцій, то можна навести приклад банку Іспанії. BBVA тепер дозволяє клієнтам в Іспанії відправляти гроші прямо через Telegram-чат-бот банку.

В результаті аналізу існуючого програмного забезпечення було виявлено, що жодна з наведених вище чат-бот реалізацій не вирішує проблеми, які описані в постановці завдання, тому на даний момент у нас немає можливості проаналізувати подібні рішення, так як їх просто немає.

Далі переходимо до визначення функціональних та нефункціональних вимог додатку.

1.3 Аналіз вимог до програмного забезпечення та розробка технічного завдання

У програмній та системній інженерії функціональна вимога визначає функцію системи або її компонента, де функція описується як специфікація поведінки між виходами та входами.

Функціональні вимоги можуть включати розрахунки, технічні деталі, обробку та обробку даних та іншу специфічну функціональність, яка визначає, що система повинна виконати. Поведінкові вимоги описують усі випадки, коли система використовує функціональні вимоги, вони враховуються у випадках використання. Функціональні вимоги підтримуються нефункціональними вимогами (також відомими як "вимоги до якості"), які накладають обмеження на дизайн або реалізацію (наприклад, вимоги до продуктивності, безпеки чи надійності). Як правило, функціональні вимоги виражаються у формі "система повинна виконувати <вимогу>", тоді як нефункціональні вимоги набувають форми "система повинна бути <вимога>". План реалізації функціональних вимог детально викладено в проекті системи , тоді як нефункціональні вимоги детально описані в архітектурі системи.

Як визначено в розробці вимог, функціональні вимоги визначають конкретні результати системи. Цьому слід протиставити нефункціональні вимоги, що визначають загальні характеристики, такі як вартість та надійність. Функціональні вимоги керують архітектурою додатків системи, тоді як нефункціональні вимоги керують технічною архітектурою системи.

Проаналізувавши наявні реалізація чат-ботів націлених на комерцію, можна зробити висновок що основними цілями для майбутнього додатку буде перш за все інтерфейс користувача та його доступність для користувачів з низьким рівнем знань у комп'ютерній сфері. Звичайно ж не потрібно забувати про функціонал нашого додатку, який має бути достатнім для користувачів з різним рівнем знань. В результаті були визначені наступні функціональні вимоги:

- покупка годинників;
- можливість оплати використовуючи сторонні платіжні системи;
- зміна локалізації в залежності від регіону користувача;
- відображення товарів використовуючи пагінацію;
- валідація усіх введених даних, в тому числі таких як ім'я, прізвище, пошта, номер, адреса;
- бронювання товару в корзину;
- швидка та зрозуміла відповідь бота.

У системній інженерії та інженерії вимог нефункціональна вимога (NFR) - це вимога, яка визначає критерії, за якими можна судити про роботу системи, а не про конкретну поведінку. Їм протиставляються функціональні вимоги, що визначають конкретну поведінку або функції. Нефункціональні вимоги мають форму "система повинна бути <вимога>", загальною властивістю системи в цілому або певного аспекту, а не конкретною функцією. Загальні властивості системи, як правило, позначають різницю між успіхом проекту чи його невдачею.

Нефункціональні вимоги часто називають "атрибутами якості" системи, однак між ними існує різниця. Нефункціональні вимоги - це критерії оцінки ефективності програмної системи, і програмна система повинна мати певні

атрибути якості, щоб відповідати нефункціональним вимогам. Отже, коли ми говоримо, що система повинна бути «безпечною», «доступною», «портативною», «масштабованою» тощо, то ми говоримо про її атрибути якості. Іншими термінами для нефункціональних вимог є "якості", "цілі щодо якості", "вимоги до якості обслуговування", "обмеження", "вимоги до поведінки", або "технічні вимоги". Неофіційно їх іноді називають "ілюзіями", від таких характеристик, як стабільність та портативність. Нефункціональні вимоги можна розділити на дві основні категорії:

а) виконавчі якості, такі як безпека, та зручність використання, які можна спостерігати під час роботи;

б) еволюційні якості, такі як можливість тестування системи, обслуговування, розширюваність та масштабованість, які втілені у статичній структурі системи.

в результаті було сформовано наступні нефункціональні вимоги:

1) коректне відображення даних;

2) швидкий запуск веб-додатку;

3) зрозуміла користувацька документація;

4) інтуїтивно зрозумілий інтерфейс.

Отже, в цьому розділі було проведено аналіз предметної області та встановлено, що інтернет магазини – популярні веб-сервіси, що відіграють значну роль у сучасному житті людини, забезпечують комунікацію між людьми, виступають у якості основи для ведення бізнесу. Усі вони повинні бути продуктивними, захищеними та легкими у користуванні.

Також було проведено аналіз існуючого програмного та інформаційного забезпечення предметної області, в результаті якого визначено, що існує багато подібних програмних систем і додатків, усі вони відрізняються своїм зовнішнім виглядом, функціональністю та зручністю у користуванні, проте більшість мають усі необхідні для соціальних мереж функції. Також визначені функціональні та нефункціональні вимоги до програмного забезпечення та виконано опис основних прецедентів системи, та розроблене технічне завдання на розробку програмної системи, яке наведено в додатку А.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕНЯ

2.1 Аналіз та проектування архітектури системи

З розвитком технологій веб стає все більш важливим в нашому повсякденному житті, в якій практично все що ми робимо в даний час, пов'язано з використанням Інтернету. Більш того, застосування Інтернету не обмежується комп'ютерами та воно відкрите для різних видів інтелектуальних цифрових пристроїв, наприклад мобільних телефонів. Крім того, архітектура мережі використовує модель клієнт-сервер, в якій результатом є зв'язок між клієнтом і сервером. Система клієнт / сервер все більше зводить до мінімуму час розробки додатків, розділяючи функції обміну інформацією як на клієнта, так і на сервер. Клієнт є запитувачем, в той час як сервер є постачальником послуг. У більшості клієнт-серверних середовищ обробка даних обробляється сервером, а результати повертаються клієнтам, і в результаті цього зростає продуктивність. Наприклад, на робочій станції принтер може бути підключений до комп'ютера (клієнта), в той час як інші комп'ютери, які спільно використовують його, є сервером. Приклад системи клієнт сервер наведений на рисунку 2.1.

Рисунок 2.1 – Клієнт-серверна система

У сучасному комп'ютерному світі система клієнт-сервер стала настільки популярною, тому що вона використовується практично кожен день для різних додатків. Деякі з стандартизованих протоколів, які клієнт і сервери використовують для зв'язку між собою, включають в себе: протокол передачі файлів (FTP), простий протокол передачі пошти (SMTP) і протокол передачі

гіпертексту (HTTP). Таким чином, систему клієнт-сервер можна визначити як програмну архітектуру, що складається як з клієнта, так і з сервера, в якій клієнти завжди відправляють запити, в той час як сервер відповідає на відправлені запити. Клієнт-сервер забезпечує міжпроцесний зв'язок (наведений на рисунку 2.2), оскільки він включає в себе обмін даними як від клієнта, так і від сервера, в результаті чого кожен з них виконує різні функції.

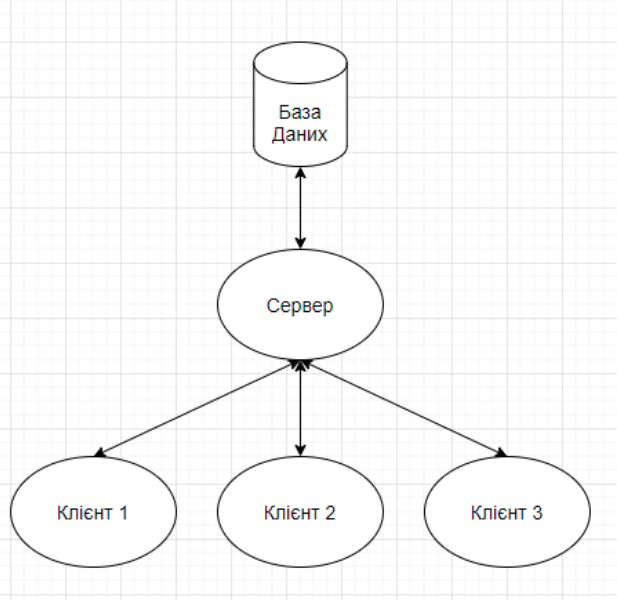


Рисунок 2.2 – Міжпроцесний звязок

Переваги клієнт-серверної моделі:

– поділ обробки додатків на кілька машин;

– дозволяє спростити спільне використання ресурсів від клієнта до серверів і навпаки;

– зменшує реплікацію даних, зберігаючи дані на сервері, а не на клієнті.

приклад додатків, що використовують клієнт-серверну систему:

– передача файлів: передача файлів між клієнтом і сервером. він також дозволяє зберігати файли на сервері. можна зберігати такі файли, як фільми, зображення, музика;

– передача пошти: передача повідомлень, таких як електронна пошта, з використанням протоколу передачі пошти (MTP);

– протокол передачі гіпертексту (HTTP): передача мультимедійних файлів, таких як зображення, текст, між клієнтом і сервером . HTTP використовується для поліпшення зв'язку між клієнтом і сервером, виступаючи в якості протоколу запиту-відповіді.

Клієнт-серверна архітектура. Архітектура клієнт-сервер зазвичай складається з сервера додатків, сервера баз даних і ПК. Дві основні архітектури-це 2-рівнева і 3-рівнева архітектура.

2-рівнева архітектура клієнт-серверної системи: це архітектура, яка включає в себе тільки сервер бази даних і клієнтський комп'ютер. У 2-рівневій архітектурі користувачі будуть запускати додатки на своєму ПК (клієнті), який підключається через мережу до сервера. Клієнтська програма виконує як кодування, так і бізнес-логіку, а потім відображає вихідні дані користувачеві. Його ще називають великим клієнтом. Використовується тоді, коли клієнт має доступ до бази даних безпосередньо, без залучення будь-якого посередника.

Також використовується для виконання логіки програми, відповідно до якої код програми буде призначений кожному клієнту на робочій станції. Приклад 2-рівневої архітектури наведено на рисунку 2.3

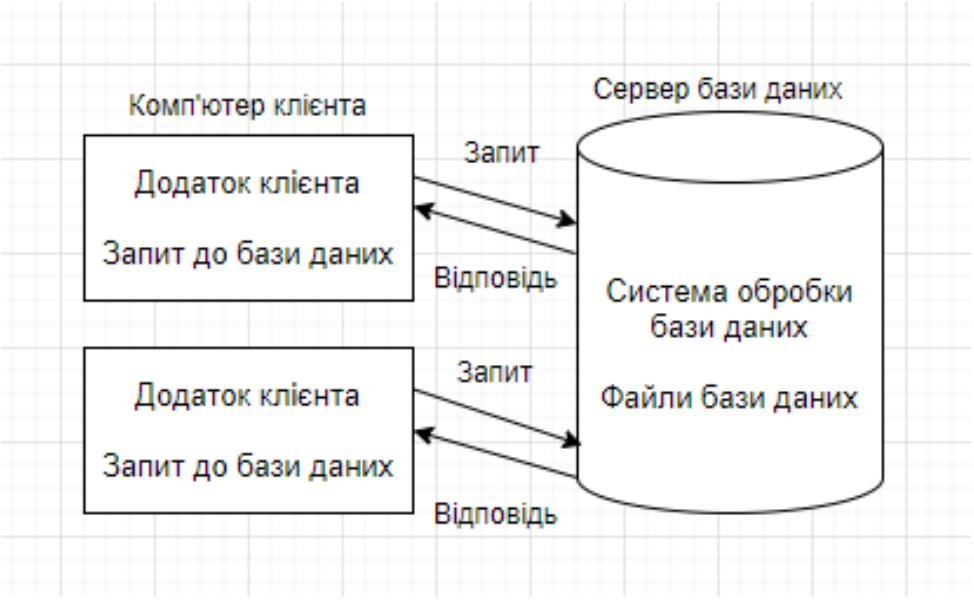


Рисунок 2.3 – Приклад 2-рівневої архітектури

3-рівнева архітектура системи клієнт-сервер: ця архітектура включає в себе клієнтський ПК, сервер баз даних і сервер додатків. 3-рівнева Архітектура може бути розширена до n-рівневої, в результаті чого вона зможе включати в себе більше серверів додатків. У цій архітектурі клієнт містить тільки логіку відображення, в результаті чого потребує менше ресурсів і менше коду необхідного для клієнта. Вона підтримує один сервер, який відповідає за безліч клієнтів, і надає більше ресурсів на сервері. Також включає в себе посередника (сервер додатків), також відомого як проміжне програмне забезпечення. Проміжне програмне забезпечення: 3-рівнева архітектура включає в себе сервер додатків, який служить проміжним програмним забезпеченням між клієнтським ПК і сервером баз даних. Проміжне програмне забезпечення це окреме програмне забезпечення, що працює на окремій машині і виконує логіку програми. У такої архітектури основними недоліками є те, додаток складніше спроектувати і побудувати, розгортувати і підтримувати. Також у такої архітектури зазвичай високі вимоги до продуктивності серверів застосунків і до швидкості каналу мережі між сервером і базою даних. Приклад наведено на рисунку 2.4.

Рисунок 2.4 – Приклад 3-рівневої архітектури

При аналізі клієнт серверної архітектури були виявлені наступні переваги:

– дані централізовані в системі, яка підтримується в одному місці;

– модель ефективна в доставці ресурсів клієнту, а також вимагає недорогого обслуговування;

– даними легко керувати, і вони можуть бути легко та швидко доставлені клієнту;

– оскільки дані централізовані, ця система більш безпечна і забезпечує додаткову безпеку даних;

– в рамках цього типу моделі в сервер може бути вбудовано більше клієнтів і серверів, що робить продуктивність досить високою і підвищує загальну гнучкість моделі.

2.2 Аналіз, вибір та проектування архітектури серверної частини ПЗ

Наступним важливим кроком в конструюванні додатку буде проектування серверної частини з детальним описом. Для того щоб обрати архітектурний шаблон проведемо аналіз вже існуючих варіантів, а саме:

– монолітна архітектура;

– мікросервісна архітектура.

Проведемо порівняння існуючих варіантів і аналіз для того, щоб зрозуміти їхні відмінності, переваги і недоліки, а також щоб зрозуміти, яку серверну архітектуру варто обрати для реалізації серверної частини в додатку [10].

Монолітна архітектура – традиційна уніфікована модель для проектування програмного забезпечення. Монолітний додаток будується як одне і нероздільне ціле ПЗ [11]. Зазвичай, таке рішення включає в себе користувацький інтерфейс на стороні клієнта, серверний додаток і базу даних. Він уніфікований, і усі функції керуються і обслуговуються в одному місці. Зазвичай, монолітні додатки вмістять в собі абсолютно весь код програми а також відсутність модульності.

Якщо розробники хочуть щось оновити або замінити – вони звертаються до одного і того ж самого сховища коду, таким чином вони вносять зміни, які впливають повністю на все програмне забезпечення.

Даний тип серверної архітектури є поширеною практикою при побудові веб-додатків, також зручний в використанні для маленьких груп розробників. Важливим моментом, який потрібно враховувати є те, що компоненти монолітного програмного забезпечення взаємозв'язані і залежні.

Модель наведеної архітектури відображена на рисунку 2.5.

Рисунок 2.5 – Модель монолітної архітектури

Найбільшою перевагою використання монолітної архітектури є те, що її набагато простіше і швидше, зручніше реалізовувати. В такій архітектурі ми можемо швидко реалізовувати бізнес-логіку, замість того, щоб тратити час і ресурси на міжпроцесну взаємодію.

Важливо також сказати, що моноліт простий в розгортанні, а також легко масштабується. Для розгортання достатньо написати маленький скрипт, який завантажить і запустить додаток.

Також важливим аргументом в користь моноліту є те, що у таких додатків менше «скрізних» проблем. Скрізні проблеми – проблеми, які впливають на весь додаток, такі як ведення журналу, обробка, кешування, моніторинг швидкодії. В

монолітному додатку ця область функціонування відноситься лише до одного додатку, тому їх набагато легше опрацьовувати, так як не потрібно витрачати ресурси на інтеграцію з зовнішніми механізмами для підтримки, як це відбувається в мікросервісній архітектурі.

Модульний додаток простіше тестувати, оскільки не потрібно враховувати інтеграцій з іншими сторонніми ресурсами, так як вся логіка відбувається в межах одного веб-додатку.

Проте, у такого архітектурного рішення є цілий ряд недоліків, які впливають на процес розробки ПЗ. Зазвичай, проекти які основані на такій архітектурній моделі дуже швидко ростуть, і стають складними в розумінні. Між фрагментами програми, які вирішують різні задачі розмиваються кордони, що може призвести до різних проблем та багів, які дуже важко переписуються. Також, великою системою коду рамках одного додатку важко керувати. Великий моноліт лякає розробників, особливо коли вони новачки в команді. Такий веб додаток важко зрозуміти і модифікувати.

Дуже важко реалізовувати нову логіку в великому і складному додатку з щільним зв’язком в логікі. Будь-які зміни коду впливають на всю систему, так як тому воно має бути точно і обережно скоординовано. В такому випадку, реалізація нового функціонала значно розтягує процес розробки.

Досить проблематично інтегрувати нову технологію в монолітному додатку, тому що зазвичай, немає інструментів для підтримування двох різних мов програмування в одній програмі. Окрім того, якщо програма використовує одну платформу, яка з часом старіє – буде дуже складно почергово перенести програму на більш новішу і кращу платформу.

Також не менш важливим недоліком монолітної структури є те, що у таких програмах будь яке невелике оновлення вимагає повного розгортання системи на сервері. Таким чином, команди розробників повинні чекати, поки розгортання не закінчиться. Коли кілька команд працюють над монолітним проектом, швидкість процесу розробки може значно вирости.

Мікросервісна архітектура – архітектура, при якій розроблювана система ділиться на набір невеликих незалежних блоків [11]. Кожний з цих блоків представляє з себе окремий сервіс. Кожний з таких сервісів має свою бізнес логіку, базу даних, і чітку визначену ціль. В такій архітектурі сервісів вся функціональність розділена на незалежні розгортанні модулі, які взаємодіють друг з другом за допомогою певних методів, які називаються API (інтерфейси прикладного програмування). Кожний сервіс покриває свою задачу, а також може оновлюватись, розгортуватись, масштабуватись незалежно від інших компонентів. Підхід на основі мікросервісів орієнтований головним чином на бізнес пріоритети і можливості, тоді коли монолітна архітектура організована технологічних рівнів, користувацьких інтерфейсів і баз даних. Зазвичай, компоненти такої архітектури спілкуються між собою за допомогою HTTP запитів. Модель мікросервісної архітектури зображена на рисунку 2.6.

Головною перевагою мікросервіносної архітектури є те, що веб-додаток може розроблюватись, тестуватись, розгортуватись паралельно і незалежно від інших компонентів. Оскільки одиниця розгортання не велика, це спрощує і прискорює розробку і реліз програми. Окрім того, реліз одного компонента ніяк не обмежується релізом іншого компонента, який ще не завершений. Так як ця архітектура складається з незалежних модулів, можемо повністю керувати його життєвим циклом не прив’язуючись до інших компонентів. Кожний такий модуль можна реалізувати використовуючи будь-яку мову програмування, яка найкраще підходить для рішення певної бізнес задачі. Тобто, не важливо як реалізований модуль, тому що компоненти мікросервісної архітектури спілкуються між собою використовуючи один стандарт. Зазвичай, таким стандартом виступає HTTP стек. Саме тому в мікросервісній архітектурі, ми можемо зосередитись на рішенні бізнес задачі, а не тратити час і ресурси на планування і розгляд інструментів реалізації веб-додатку. Також, усі компоненти можуть бути розгорнутими і оновленими окремо, це дає велику гнучкість та зручність у роботі сервісу[11].

З допомогою мікросервісів декілька команд можуть одночасно працювати над різними компонентами паралельно. Кожний модуль веб-додатку може бути побудований незалежно, так як вони повністю ізольовані один від одного, що дуже спрощує процес розробки окремих мікросервісів.

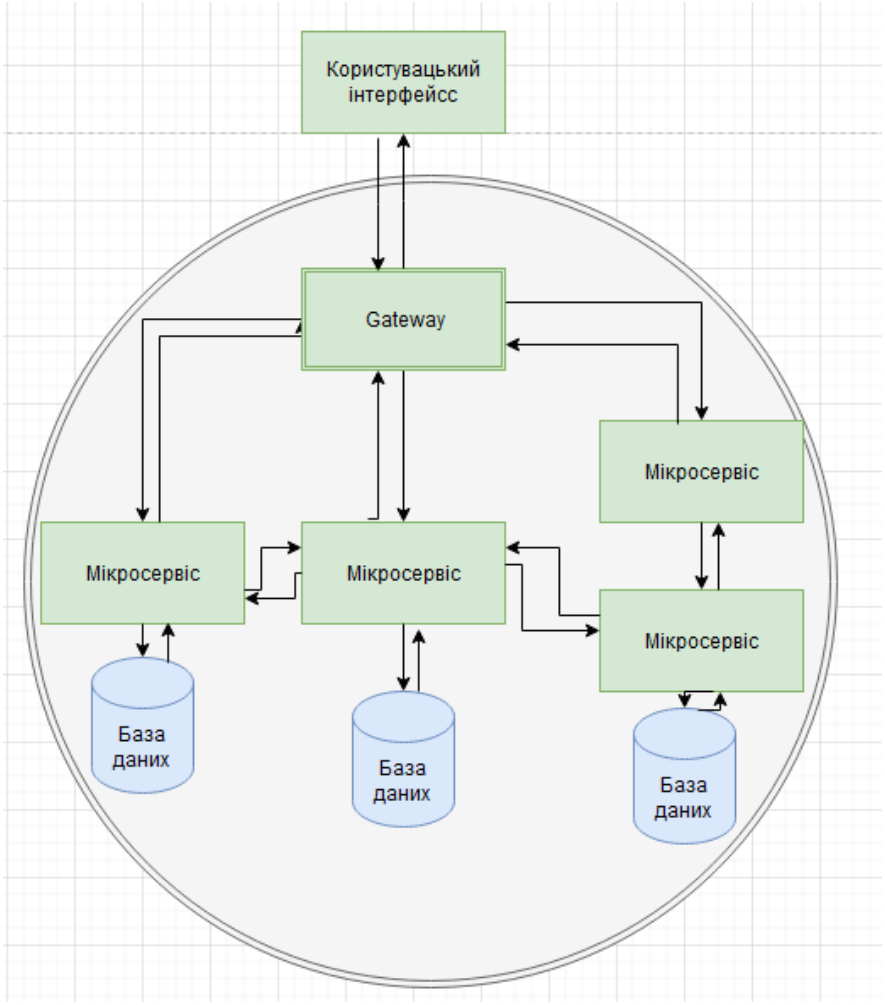


Рисунок 2.6 – Модель мікросервісної архітектури

Висока гнучкість такої архітектури дозволяє розробникам обновляти компоненти системи, не вимикаючи весь додаток. Окрім того, гнучкість забезпечує більш безпечний процес розгортання і більше часу безвідмовної роботи. Нові функції можна добавити по мірі необхідності, не очікуючи запуска усього веб-додатку.

Також важливим моментом є те, що якщо в якогось компонента такої архітектури відбулась поламка – вся система і далі буде функціонувати, тому що

ця поламка буде впливати лише на одну конкретну службу, а не на весь додаток. Як результат, знайти проблему і рішити її не буде складати труднощів, адже область потенційної проблеми складається лише з цього компонента, а не всієї системи продукту, як це може відбуватись в монолітній архітектурі.

В системі, яка реалізована на такій архітектурі завжди є можливість масштабування по горизонталі. Зазвичай, вертикальне масштабування (при використанні того ж ПЗ, але на більш сильних машинах) може бути обмежено пропускною можливістю кожного сервісу. Але горизонтальне масштабування (створення великої кількості сервісів в одному пулі) не обмежено, і може працювати з мікросервісами динамічно. Окрім того, горизонтальне масштабування може бути повністю автоматизоване.

Останньою ключовою перевагою, яку варто вказати для такої архітектури є те, що деякі компоненти можна використовувати в інших продуктах. Якщо реалізувати модуль повністю незалежним – його можна використовувати повторно. Наприклад, це може бути певний інструмент, який може щось обраховувати і вертати результат, або поштовий клієнт, завданням якого є просто відправити інформацію на вказану пошту, тощо. Проте, в описаній архітектурі є ряд свій ряд недоліків, які обов'язково потрібно враховувати при проектуванні веб-додатку. Перш за все, це складність, розділення веб-додатку на незалежні мікросервіси тягне за собою більше артефактів керування. Цей тип архітектури потребує більш ретельного планування, тяжких зусиль, командних ресурсів і навичок. Причини високої складності наступні:

– великий запит на автоматизацію, оскільки кожний компонент має бути провірений і підконтрольний певним інструментам оркестрації;

– доступні інструменти не працюють з сервісними залежностями;

– узгодженість даних і управління компонентами стає складніше, так як кожен сервіс має свою власну базу даних.

Наступним важливим недоліком такої архітектури є проблеми в безпеці, тому як в мікросервісному веб-додатку кожний функціонал який взаємодіє через

API ззовні, збільшує імовірність атак. Ці атаки можуть трапитись тільки в тому випадку, якщо при проектуванні і реалізації не будуть виконані необхідні міри безпеки. Зазвичай, щоб уникнути такого ряду проблем достатньо реалізувати додатковий мікросервіс, який візьме на себе відповідальність перевірки безпеки. В практиці, такий сервіс називається «gateway», він приймає запити з зовні, перевіряє токени доступу, або права користувача, і в разі успішної варіації – перенаправляє запити на API веб додатку. Така архітектура описана на рис 2.7.

Також, незначним недоліком мікросервісної архітектури є те, що такий веб-додаток набагато складніше описати тестами, так як потрібно враховувати множину інтеграцій з іншими компонентами системи [11].

Провівши детальний аналіз і порівняння наведених сеорверних архітектурних шаблонів, було вирішено обрати мікросевісну архітектуру, оскільки така архітектура оптимальніше підходить для реалізації продукту. Реалізацію веб-додатку можна розділити на окремі незалежні мікросервіси, які будуть виконувати певну бізнес логіку.

Перш за все варто виділити логіку яка відповідає за контроль і зберігання інформації про товари, користувачів, покупки, транзакції в окремий центральний сервіс, назвемо його для простішого розуміння be-shop (be як скорочення абревіатури back-end). Цей мікросервіс буде надавати API для інших сервісів продукту. Також він буде зберігати ключові дані магазину, а саме: всю інформацію стосовно товарів, інформацію про користувачів а також їхні права та доступи, інформацію про замовлення товарів а також активні транзакції системи. Тобто можна зрозуміти, що be-shop буде центральним компонентом, який зберігає основну інформацію усього веб-додатку, а також надає API для інших мікросервісів, що дозволяє в майбутньому реалізувати інтеграцію з магазином в будь-якому вигляді, окрім телеграм ботів.

Наступним важливим компонентом цієї архітектури є панель адміністратора, яка надає ключові інструменти взаємодії в веб-додатку, а саме: керування товарами - добавлення нових або редагування існуючих товарів

магазину, керування персональною інформацією користувачів, тобто перегляд і редагування інформації про користувача, а також надання доступу редактора або адміністратора магазину.

Також потрібно виокремити в окремий мікросервіс логіку, яка відповідає за взаємодію звичайних користувачів з магазином. Перш за все, це перегляд товарів магазину, їхньої детальної інформації. Також, це логіка яка відповідає за створення нових покупок, а саме – перегляд товару, добавлення його в корзину і подальша оплата замовлення. Користувач може переглянути існуючі замовлення, і відмінити їх. Мікросервіс буде зберігати тимчасову інформацію про користувача а також телеграм сесію в базі даних. Відносно активної мови сесії телеграм з'єднання в інтерфейсі магазину динамічно міняти мову користувацького інтерфейсу на вибрану.

В магазині має бути реалізована можливість оплати замовлень, реалізацію цього бізнес завдання можна винести в окремий мікросервіс, який буде відповідати за створення нових замовлень а також відслідковування їхнього актуального статусу в платіжній системі.

Згідно вище з проаналізованою предметною областю, можна виділити основні компоненти веб додатку:

– редагування і керування товарами і обліковими записами користувачів виокремимо в мікросервіс з назвою «admin-tg»;

– реєстрація, перегляд товарів, добавлення товарів в кошик, покупка виокремити в мікросервіс з назвою «client-tg»;

– зберігання і управління інформації про користувачів, керування транзакціями покупок, виокремити в мікросервіс з назвою «be-core»;

– інтеграцію з платіжними системами для оплати винести в мікросервіс з назвою «wrapper».

Описані вище мікросервіси будуть «спілкуватись» між собою використовуючи HTTP запити. Взаємодія з веб-додатком буде відбуватись також за допомогою HTTP запитів, які будуть приходити з сторони серверів Telegram,

а мікросервіси веб-додатку будуть їх опрацьовувати і повертати назад відповідь як результат запитів. Описана архітектура зображена на рисунку 2.7.

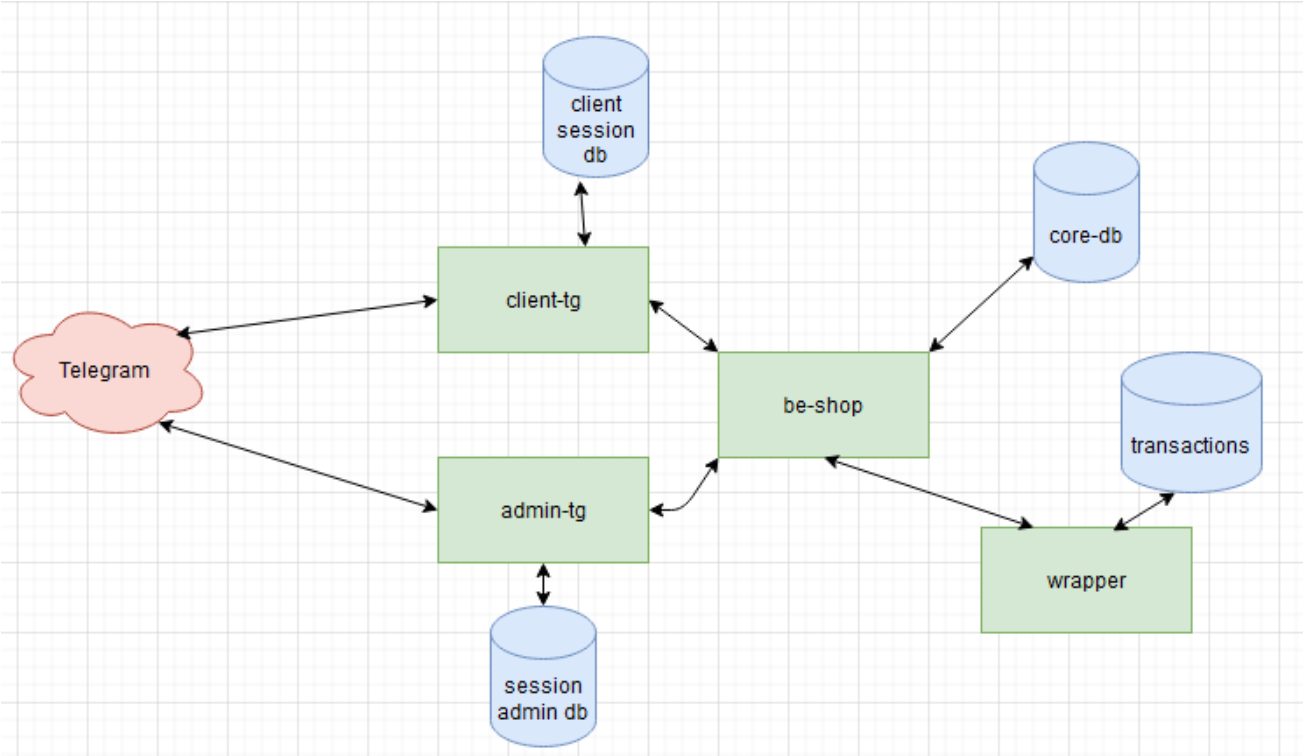


Рисунок 2.7 – Архітектура серверної частити магазину

Наступним важливим кроком потрібно проаналізувати взаємодію вказаних сервісів при виконанні основного функціоналу, який вказаний в постановці задачі: перегляд наявних в магазині товарів, надання системі інформації про користувача, добавлення товарів в корзину і оформлення замовлення, добавлення товарів в базу даних, перегляд інформації про користувача в системі. Для цього буде зручно використати діаграми послідовності, що відображають взаємодії об’єктів, які впорядковані за часом [8].

Такі діаграми можна використовувати для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання, так як це відмінний інструмент з точки зору сценаріїв використання [3]. Зазвичай, на таких діаграмах основними об’єктами є абстракції об’єктно орієнтованої підходу, але тим не менш таку діаграму можна використати і для мікросервісної архітектури але на

абстрактному рівні, тобто на заміну класам виступають сервіси, а кожний запит є повідомленням від одного сервісу до іншого [10].

На рисунку 2.8 зображена діаграма послідовності перегляду користувачем наявного товару в магазині:

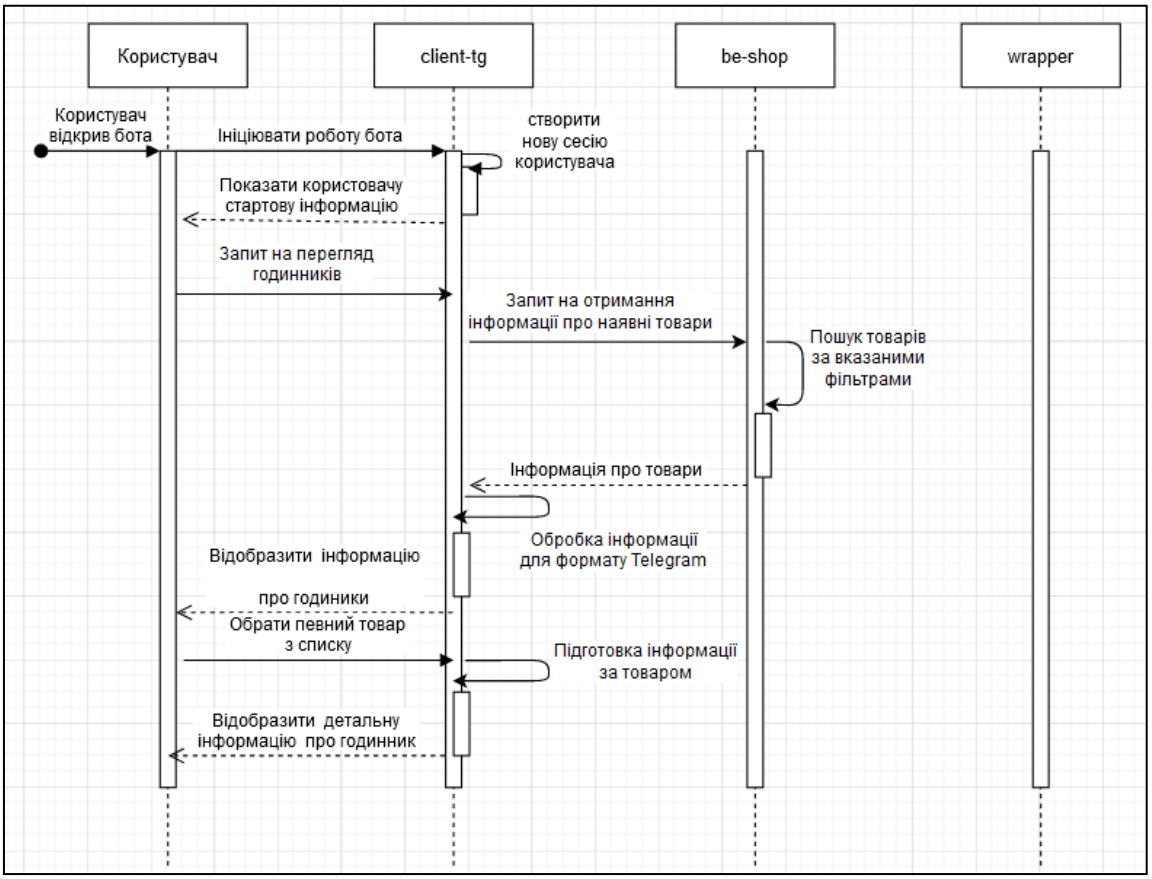


Рисунок 2.8 – Діаграма послідовності перегляду товарів магазину

Такий сценарій відбувається у випадку, коли користувач розпочав розмову з ботом, як результат – бот має створити сесію, тобто зібрати і записати в базу даних інформацію про користувача. Після створення сесії, бот вітає користувача тією мовою, яка активна в користувача в клієнті телеграму. Коли користувач обирає в меню пункт годинники, client-tg отримує запит на перегляд товарів в базі даних, і щоб отримати - робить запит на API be-shop. Be-shop отримавши такий запит починає процес збору інформації про товари відповідно фільтрів, які йому передали в запиті, в цьому запиті немає жодних фільтрів, так як це перший запит на перегляд інформації товарів. Зібравши усю інформацію про товари, be-shop

повертає усю відповідну інформацію в результаті запиту для client-tg, а він в свою чергу отримавши цю інформацію оброблює її в формат відображення Telegram, і повертає обновлену інформацію на запит перегляд товарів. Як результат, в користувача відображається активна пагінація з товарами, які є в базі даних.

Наступним кроком користувач може переглянути будь-який товар з списку пагінації, і вибравши годинник – client-tg дістає відповідну інформацію з власної бази даних, яка зберігає усі тимчасові об’єкти сесії, і редагувавши опис товару під формат Telegram вертає на інтерфейс користувачу опис годинника.

На рисунку 2.9 зображена діаграма послідовності додання товарів в корзину, створення нового замовлення:

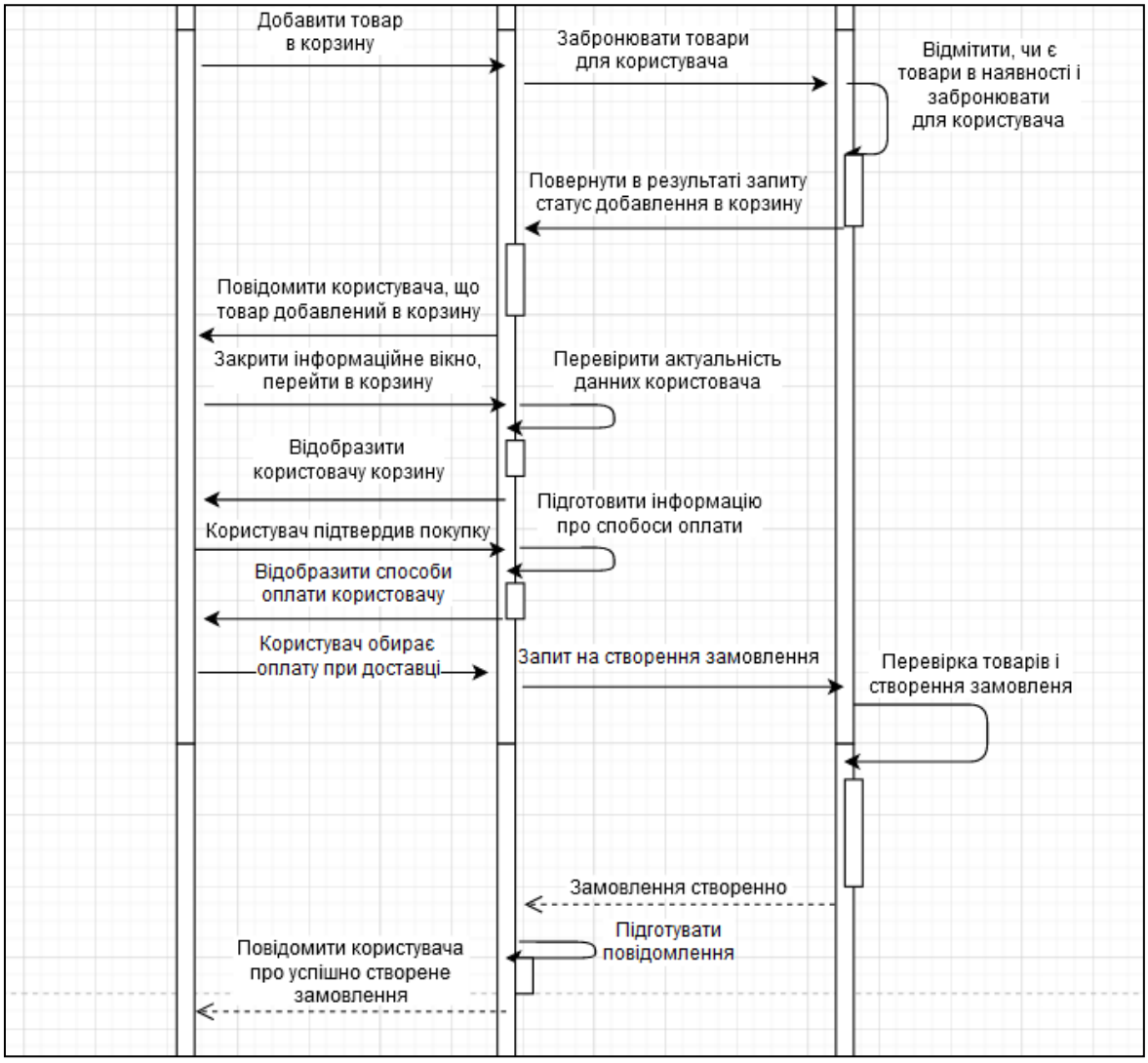


Рисунок 2.9 – Діаграма послідовності для нового замовлення

Сценарій створення нового замовлення відбувається послідовно після сценарію «перегляд товару», який зображений на рисунку 2.4. Після того, як користувач переглянув детальний опис годинника чи аксесуара, він може той товар добавити в корзину. Але, користувач не може добавити в корзину ті предмети, кількість яких в магазині менше одного, в такому випадку кнопка добавити товар в корзину не буде відображатись на інтерфейсі, але товар все ще можна буде переглянути. Після того як товар добавили в кошик, користувачу відобразиться відповідне повідомлення, а в базі даних кількість товарів буде зарезервована користувачем, тобто цей товар не буде доступний для інших клієнтів веб-додатку. Також, користувач може далі переглядати товари, також добавляти товари в корзину. Для того щоб оформити замовлення, юзеру потрібно перейти в «кошик» і підтвердити замовлення.

Користувачі, які не надали контактної інформації про себе не можуть оформити покупку. В такому випадку, відобразиться відповідне повідомлення, і розпочнеться сценарій «надання контактної інформації» який зображений на рисунку 2.10. Під час цього сценарію користувач має вказати про себе контактну інформацію, яка потрібна для оформлення замовлення, а саме: номер мобільного телефону, електронна пошта, і адреса для отримування по пошті куплених товарів в магазині. Уся інформація в обов'язково має перевірятись на правильність введення. Після того як користувач ввів усю відповідну інформацію про себе, client-tg відправляє оновлену інформацію про користувача на мікросервіс be-shop.

Важливим моментом є те, що користувач має обрати метод оплати за товари, які є в кошику. Відповідно завданню в постановці задачі, користувач може оплатити замовлення використовуючи сторонні засоби оплати – рис 2.11, або оплатити при отримуванні товару, в цьому сценарії користувач обрав варіант «оплатити при отримуванні на пошті». Оплата при отримуванні товару опрацьовується не автоматично, а інформацію про доставку потрібно вносити вручну в базу даних магазину.

Після надання інформації, користувач може оформити замовлення, як результат з client-tg буде відправлено запит на оформлення замовлення be-shop, а користувачу буде відображено інформацію про успішне замовлення.

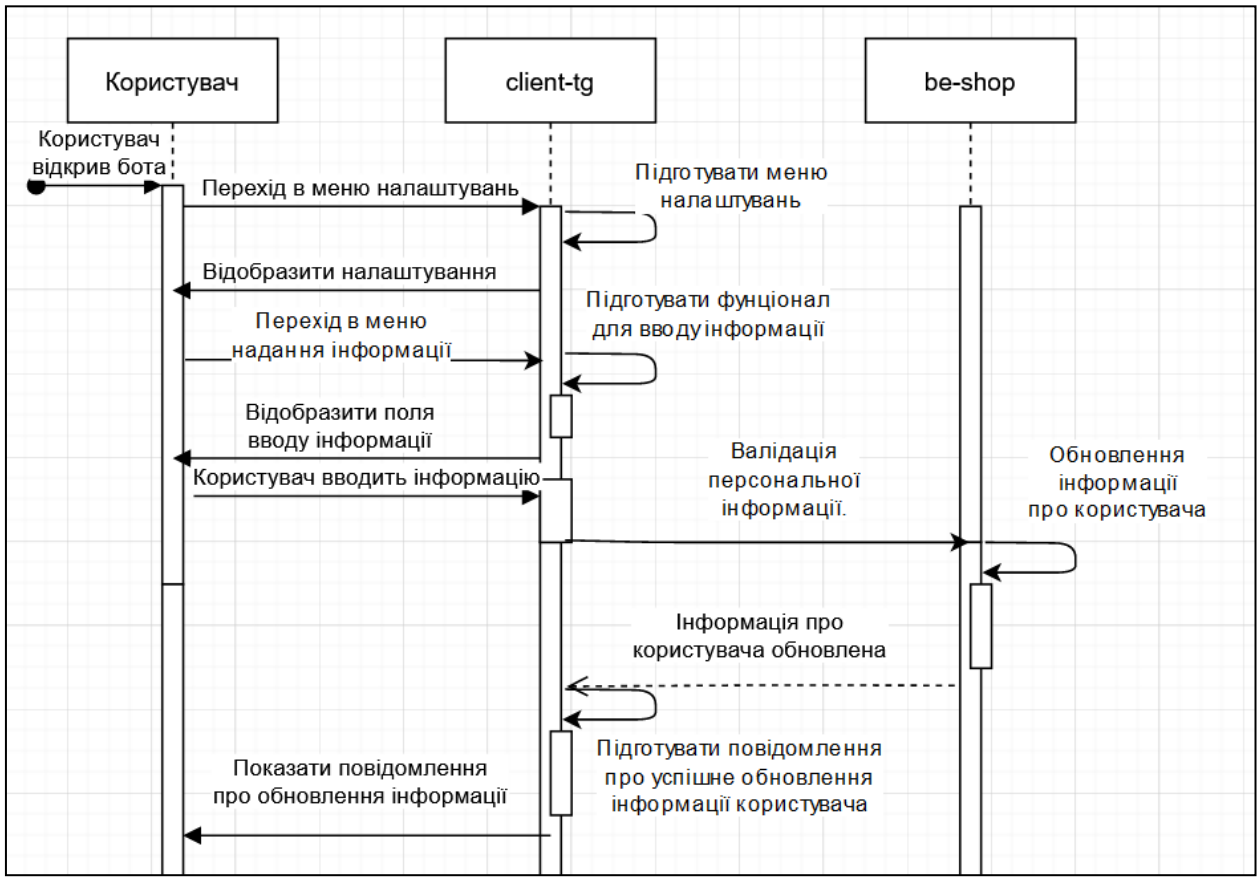


Рисунок 2.10 – Діаграма послідовності обновлення інформаці

У випадку, якщо користувач вибрав оплату замовлення використовуючи стороні засоби оплати – відбудеться сценарій «оплата використовуючи сторонні засоби оплати», який зображений на рисунку 2.11. Client-tg перевіряє наявні в корзині товари, а лише після того звертається до be-shop з запитом на отримання актуального статусу платіжних систем. Послідовно, be-shop звертається до відповідальних мікросервісів, які повертають актуальний статус сторонніх систем. Відповідно, якщо система не працює – відповідне повідомлення буде відображене користувачу, якщо система працює стабільно, client-tg відобразить користувачу повідомлення, шо користувач може сплатити замовлення вибраним способом. Якщо користувач не бажає оплачувати вибраним способом, він може

повернутись в попереднє меню. При оформленні замовлення wrapper створить активну транзакцію, згенерує посилання для оплати замовлення, і лише тоді client-tg відобразить її користувачу. У випадку, якщо користувач не буде сплачувати товар певний час, товар замовлення буде позначене на be-shop як застаріле, а товари стануть знову доступними для наступних замовлень. Усі діаграмі станів наведені в додатку Б.

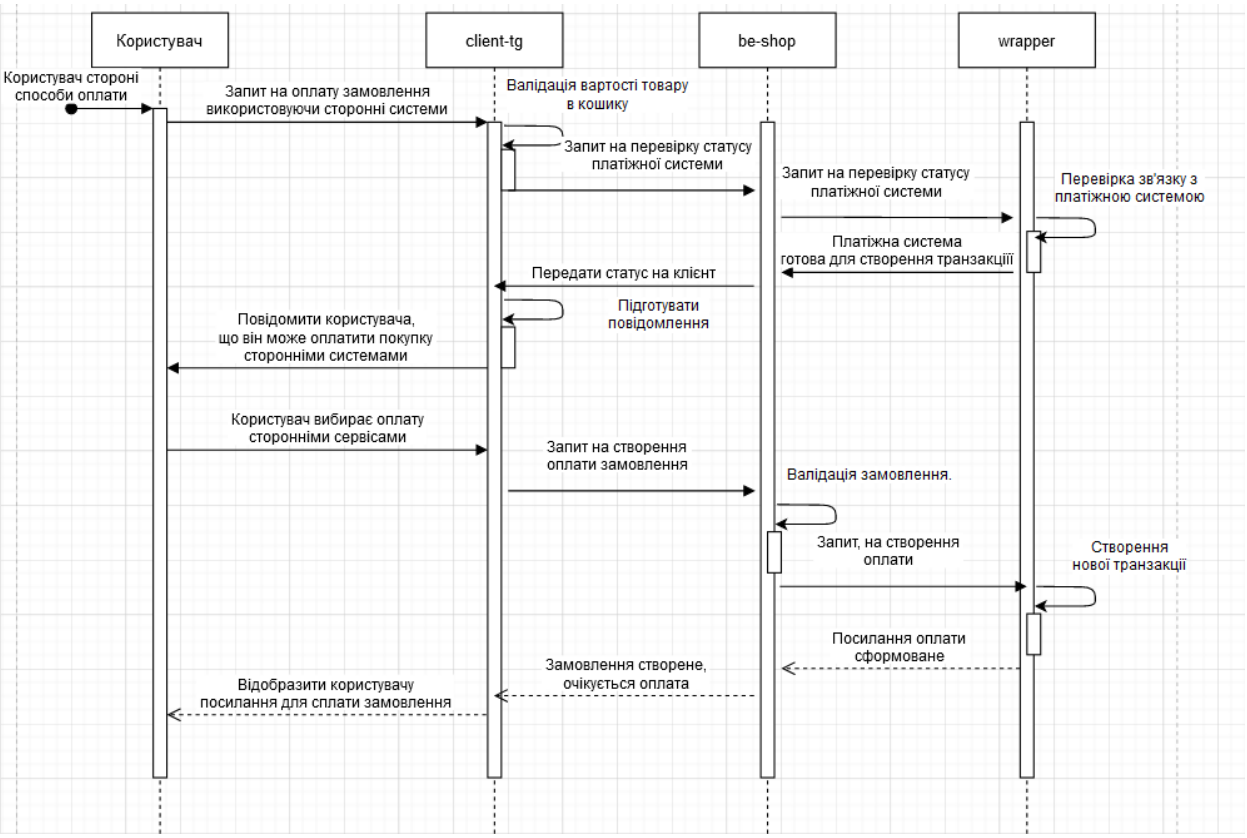


Рисунок 2.11 – Діаграма послідовності створення транзакції

2.3 Аналіз та вибір типу бази даних, проектування структури бази даних

Для того щоб обрати оптимальну базу даних (БД) для зберігання інформації веб-додатку потрібно провести детальний аналіз з порівнянням основного функціоналу. На да даний момент за моделлю зберігання і організації інформації для серверних додатків існують два основних типи баз даних, а саме це - реляційна і нереляційна бази даних, які детально розглянемо далі.

Реляційна база даних – це перш за все логічно-структурований набір інформації, заснований на реляційній моделі даних [3]. Назва походить з оригінальної назви на англійській relation – відношення. У таких сховищах інформації досягається високий рівень абстракції даних, ніж у ієрархічній або мережевій. Це досягається тим, що реляційні БД надає можливість описувати певний набір інформації тільки на лише основі їх природньої інформації, тобто не потрібно додатково надавати інформацію для машинного розуміння. Це забезпечується за рахунок використання математичної теорії відносин.

Відносини зручно і просто представляти у вигляді таблиць, де кожний рядок є кортежом, а кожний стовпець – атрибут, який визначили на певному домені. Такий підхід надає більш зручний вид представлення, коли база даних являє собою певний набір таблиць з суворою структурою. Іменоване безліч пар як ім’я атрибуту – ім’я домену називають схемою відношення, а перелік іменованих схем відносин представляють із себе повноцінну схему бази даних

Для того, щоб описати зв’язки з сутностями в таких базах даних використовується асоціації між кортежами різних відносин, а саме – дублювання їх ключів в обох таблицях.

У такої БД є ряд переваг на які варто звернути увагу при виборі сховища інформації, а саме:

– реляційні БД використовуються вже 40 років, простіше отримати підтримку, додаткові продукти і інтегрувати дані з інших систем;

– інформацію простіше розуміти, коли вона структурована у таблиці;

– БД підтримує набір принципів ACID, яка вимагає атомарність, узгодженість, ізольованість і довговічність інформації;

– повна незалежність даних, тобто якщо змінити ПЗ для роботи з БД, нічого не потрібно міняти в самій БД;

– сувора структура в таблицях надає чудову швидкодію, в порівнянні з нереляційними базами даних.

Також, у реляційних БД є відповідний ряд недоліків, а саме:

– БД може працювати лише на одному сервері, коли нереляційну можливо розподілити по різним серверам;

– реляційні БД важко масштабувати в порівняні з нереляційними;

– не усі сутності можуть бути представлені у вигляді таблиць;

– немає можливості інтеграції з нереляційними даними.

Важливим критерієм вибору БД є підтримка принципів ACID. На даний момент самими популярними реляційними базами даних є – Postgres, Oracle, а також MySQL, їхнє використання повністю безкоштовне [4].

Нереляційні бази даних, які ще називають noSQL, або розподіленими БД служать альтернативою для реляційних БД. Такі БД можуть зберігати і оброблювати неструктуровану інформацію, наприклад фото, аудіо файли, тощо. Як результат, такі можливості надають розробникам велику гнучкість і чудове масштабування. Інформація в таких БД може змінюватись під час роботи без впливу на існуючі дані. Додатково, нереляційні БД можуть працювати на декількох серверах, як результат – їх масштабування дешевше і простіше, чим масштабування даних в реляційних БД. А оскільки нереляційні БД не залежать від жодного сервера, вони більш стійкі до відмов. Це означає, що у випадку збою одного компонента, БД може продовжувати працювати.

Нереляційні БД діляться на дві основні групи: зберігання по ключ-значення та орієнтовані на зберігання цілого документа, або графи. Самим простим типом нереляційних БД є основані на ключ-значення, відповідно в такому випадку БД може зберігати лише ключ і відповідний об’єкт. Функціонал в такому випадку складається лише з базових функцій: отримання та запис об’єкта по ключу. Такий тип сховища найкраще підходить у випадку, якщо потрібно знайти відповідну інформацію знаючи відповідний ключ. Самими популярними БД такого типу є : Amazon DynamoDB, Redis.

Бази даних, орієнтовані на документи зберігають усю інформацію, яка відноситься до одного об’єкта в одному файлі JSON, BSON або XML. Документи одного і того же типу можуть бути згруповані в колекції або списки. Такі БД

дозволяють розробникам не перейматись про типи даних і сильних зв’язках об’єктів. БД орієнтована на документи зазвичай має модель дерева або «лісу». Деревовидна структура означає, що корінний вузол має один або декілька кінцевих вузлів. Така структура складається з декількох дерев. Така структура даних надає сховищам документів виконувати швидкий пошук. Хоч це і затрудняє керування складними системами з багаточисельними зв’язками між об’єктами, це дозволяє розробниками створювати колекції документів по темам або типам. Самими популярними представниками нереляційних БД такого типу є Apache Cassandra або Scylla.

Проаналізувавши нереляційні сховища можна описати їхні переваги:

– швидкий доступ до об’єктів;
– легке масштабування на різних серверах;
– проста конфігурація і швидкий запуск середовища сервера;
– відмінне керування даними big data;
– відсутність лімітів на зберігання.

Відповідно, у цього типу сховища є свій ряд недоліків, а саме:

– потребує багато фізично пам’яті сервера;
– відсутність підтримки транзакцій;
– проблеми стандартизації і сумісності;
– відносини між об’єктами не доступні;
– складність в пошуку потрібної інформації, так як нереляційні БД ще «молоді технології» на відміну реляційним БД;
– нестача інструментів для роботи з такими БД.

Проаналізувавши нереляційні бази даних, можна зрозуміти, що вони чудово підходять для зберігання динамічної інформації, яка потрібна для функціонування Telegram ботів. Основними об’єктами, які будуть зберігатись в БД є сесії користувачів, які містять багато динамічної інформації. Кожну сесію можна зберігати як об’єкт-документ, в якому оригінальним ключем буде ідентифікатор користувача в системі Telegram. Також в цій структурі можна

зберігати інформацію про те, в якому меню на даний момент користувач, відповідно зберігати потрібну тимчасову інформацію в цьому об‘єкті [6].

Описаний об’єкт не потребує жодних зовнішніх ключів, так як вся потрібна інформація буде в одній цілій структурі. Зберігання цієї інформації надає можливість показувати завжди актуальне меню в чат-боті, на вибраній в налаштуванні, користувачем мові.

Модель "сутність-зв'язок" – це модель даних, що використовується при проектуванні різноманітних моделей (інформаційних систем, баз даних, архітектури комп'ютерних додатків та інших систем) і є високорівневою концептуальною моделлю. Вона ґрунтується на деякій важливій семантичній інформації про реальний світ і є графічною нотацією, за допомогою якої можна описувати об'єкти логічних моделей даних і відношення між об'єктами.

У даному контексті модель "сутність-зв'язок" є мета моделлю даних, тобто засобом специфікації логічних моделей даних, що будуються на основі вихідної концептуальної моделі даних. Модель "сутність-зв'язок" використовують при концептуальному моделюванні для отримання концептуальної моделі, яку потім транслюють у логічні моделі, зазвичай реляційні або об’єктно орієнтовані.

Сутність – це об'єкт визначеного типу. Тип сутності визначає набір однорідних сутностей деякого типу. Множина всіх сутностей типу сутності в деякий момент часу називається множиною сутності [5].

Відношення у ER-моделі – це один із найважливіших компонентів, що застосовується для опису зв'язків між об’єктами. Існує три основних типи відношень між сутностями.

Існують 3 види відношень між сутностями, а саме:

– один до багатьох;

– один до одного;

– багато до багатьох.

Побудова коректної ER-моделі і, відповідно, проектування реляційної моделі бази даних вимагає розбиття одного відношення, та формування декількох

відношень на базі функціональної залежності. Такий процес називається нормалізацією даних. Нормалізація дозволяє зменшити кількість повторюваних даних, а також покращити загальну організацію бази даних, таким чином досягається максимальна гнучкість архітектури бази даних і також найголовніше – забезпечити повну узгодженість даних.

Процес перетворення відносин бази даних до вигляду, який відповідає нормальним формам, називається нормалізацією відношень. Нормалізація призначена для приведення структури бази даних до певного виду, що забезпечує мінімальну логічну надмірність, і не має на меті зменшення чи збільшення продуктивності роботи, або ж зменшення або збільшення фізичного обсягу БД. Кінцевою метою нормалізації є зменшення потенційної суперечливості збереженої в БД інформації. Тобто, загальне призначення процесу нормалізації полягає в таких тезах:

- виключення усіх можливих надлишковостей в БД;
- видалення можливих аномалій, які виникли при проектуванні БД;
- спрощення застосування необхідних обмежень цілісності;
- розробка проекту бази даних, який є якісним представленням реального світу, інтуїтивно зрозумілий і може бути основою для подальшого розширення за необхідності.

Існують три основні нормальні форми. При 1НФ (НФ – нормальна форма) відношення знаходиться в тільки тоді, коли в будь-якому допустимому значені відносини кожний його кортеж містить тільки одне значення для кожного з атрибутів, також при цій НФ відбувається виключення повторювальних гру в певних сутностях інформації, створюються нові сутності для кожного набору зв’язних даних, та кожний набір зв’язних даних ідентифікується за допомогою первинного ключа. Первинний ключ – це один атрибут, що однозначно ідентифікує сутність, тобто робить певний об’єкт повністю унікальним.

Відношення знаходиться в 2НФ лише тоді, якщо воно знаходиться в першій нормальній формі, і при цьому будь-який його атрибут, який не входить до складу

ключа сутності функціонально повністю залежить від кожного потенційного ключа, тобто атрибут повністю залежить від всього складного потенційного ключа, але при цьому не перебуває у функціональній залежності від будь-якої з вихідних в нього частин. Простішими словами, створюються окремі сутності для наборів значень, які застосовуються до кількох записів. Ці сутності зв’язуються за допомогою зовнішнього ключа.

Для того, щоб відношення підходило під умови третьої нормальної форми, відношення вже має знаходитись в другій нормальній формі, і також кожний неключовий атрибут відносин має знаходитись в нетранзитивному зв’язку, залежно від потенційного ключа об’єкта. Говорячи простити словами, при третій НФ повністю виключаються ті поля, що не залежать від ключа. Зазвичай, третьої нормальної форми достатньо для коректної побудови логічної і фізичної моделі відносин між сутностями.

На початковому етапі проектування логічної моделі програмної системи при аналізі предметної області було виділені наступні ключові сутності.

User – об'єкт представляє собою відображення зареєстрованого користувача веб додатку, включаючи в себе всю необхідну інформацію. Атрибути сутності «User»:

- id – ідентифікатор, первинний оригінальний ключ сутності;
- telegram_id – ідентифікатор користувача в системі Telegram;
- telegram_full_name – повне ім’я користувача в системі Telegram;
- telegram_user_name - нікнейм користувача в системі Telegram;
- full_name – ім’я користувача, яке він вказав при авторизації в ПЗ;
- delivery_address – адреса, на яку будуть відправлятись замовлення;
- language – активна мова користувача в веб-додатку;
- chat_id – системне поле, яке вказує на чат з користувачем;
- phone – мобільний номер користувача;
- referral – реферальне посилання користувача в системі;
- admin – флаг, який показує – чи користувач адміністратор;

– support – флаг, показує чи у користувача є роль підтримки;
– super_admin – флаг, чи є користувач глобальним адміністратором;
– create_date – дата додавання користувача в веб-додаток;
– last_modified_date – дата останнього редагування про користувача.

Item – сутність, яка представляє собою опис товару, який продається в магазині. Містить в собі ключову інформацію, за допомогою якої можна описати основні якості будь-якого можливого товару, атрибути цієї сутності:

– id – ідентифікатор предмету;
– name – назва предмету;
– description – опис предмету;
– item_type – тип товару, (годинник, аксесуар);
– image_url – посилання, на зображення для предмета;
– deleted – флаг, який є показує, чи був видалений товар з системи;
– create_date – дата створення товару;
– last_modified_date – дата останньої зміни цього товару.

Inventory – представляє собою сутність, яка є відображенням сховища товарів, які є в наявності в магазині. Містить в собі наступні атрибути:

– id – ідентифікатор сховища;
– item_id – зовнішній ключ, на товар;
– price – ціна товару;
– count – кількість товарів в наявності магазину;
– reserved_count – кількість зарезервованих товарів;
– created_date – дата створення;
– last_modified_date – дата останніх змін інформації про інвентар;

Cart – сутність, яка є відображенням реального замовлення, містить в собі інформацію про користувача, який створив замовлення, товари які купуються і інформацію про статус оплати. Важливим аспектом є те, що веб додаток не працює з платіжною інформацією користувача зовсім, що мінімізує будь-які ризики втрати. Описана сутність містить в собі наступні атрибути:

– id – ідентифікатор замовлення;
– user_id – зовнішній ключ, який веде до користувача;
– address – адреса, на яку відправиться замовлення;
– phone_number – номер, який був вказаний в замовленні;
– full_name – повне ім’я замовника;
– status – статус замовлення;
– payment_id – зовнішній ключ на сутність оплати;
– create_date – дата створення замовлення;
– last_modified_date – дата редагування замовлення.

Payment – ключова сутність, яка описує оплату замовлення. Містить в собі відповідну інформацію, саме: спосіб оплати, статус оплати, ціна замовлення, валюта замовлення, має наступні атрибути:

– id – ідентифікатор оплати замовлення;
– payment_status – актуальний статус замовлення;
– payment_type – спосіб оплати замовлення;
– total_price – ціна замовлення;
– wallet_address – ідентифікатор гаманця;
– create_date – дата створення;
– last_modified_date – дата останнього редагування оплати.

Clock – сутність, яка описує годинник, і його детальні характеристики, містить наступні атрибути:

– id – ідентифікатор годинника в сховищі магазину;
– variation_id – зовнішній ключ, яка описує види годинників;
– manufacturer_id – зовнішній ключ, яка описує виробника;
– waterproof – водостійкість годинника;
– body_material – матеріал, з якого зроблений годинник;
– bracelet_material – матеріал браслета годинника;
– mechanism_type – тип механізму;

– shop_link – посилання на сторінку товару в магазині виробника.

Важливим моментом є те, що у всіх сутностях є додаткові поля, які відповідають за дату створення сутності, і дату останнього редагування сутності. Це дає змогу відслідковувати зміни, а також сортувати товари в пагінації за датою створення сутності в системі веб-додатку.

Отже, після проведення нормалізації даних відповідно до третьої нормальної форми було декомпозовано систему сутності, яка зображена у вигляді ER діаграми (додаток Б, рисунок 6).

2.4 Проектування інтерфейсу користувача

Інтерфейс користувача – (user interface) – сукупність засобів, за допомогою яких користувач спілкується з різними пристроями з різними програмами, веб-додатками, тощо. Як стандарт, інтерфейси максимально пристосовуються для зручності користувача.

Для реалізації інтерфейсу користувача в описаному веб-додатку буде використовуватись функціонал месенджера Telegram. Месенджер надає ряд інструментів, які можна використати для реалізації інтерфейсу магазину, а саме:

– розширені текстові повідомлення;
– підтримка різних шрифтів;
– інтерактивне меню;
– списки;
– випливаючі підказки;
– підтримка файлів мультимедіа;
– інтеграція з іншими додатками.

Для кращого розуміння, розглянемо інтерфейс чат-боту «Укрзалізниці» для перегляду та бронювання білетів, а також перегляду трафіка руху поїзда по номеру. Чат-бот зображений на рисунку 2.12.

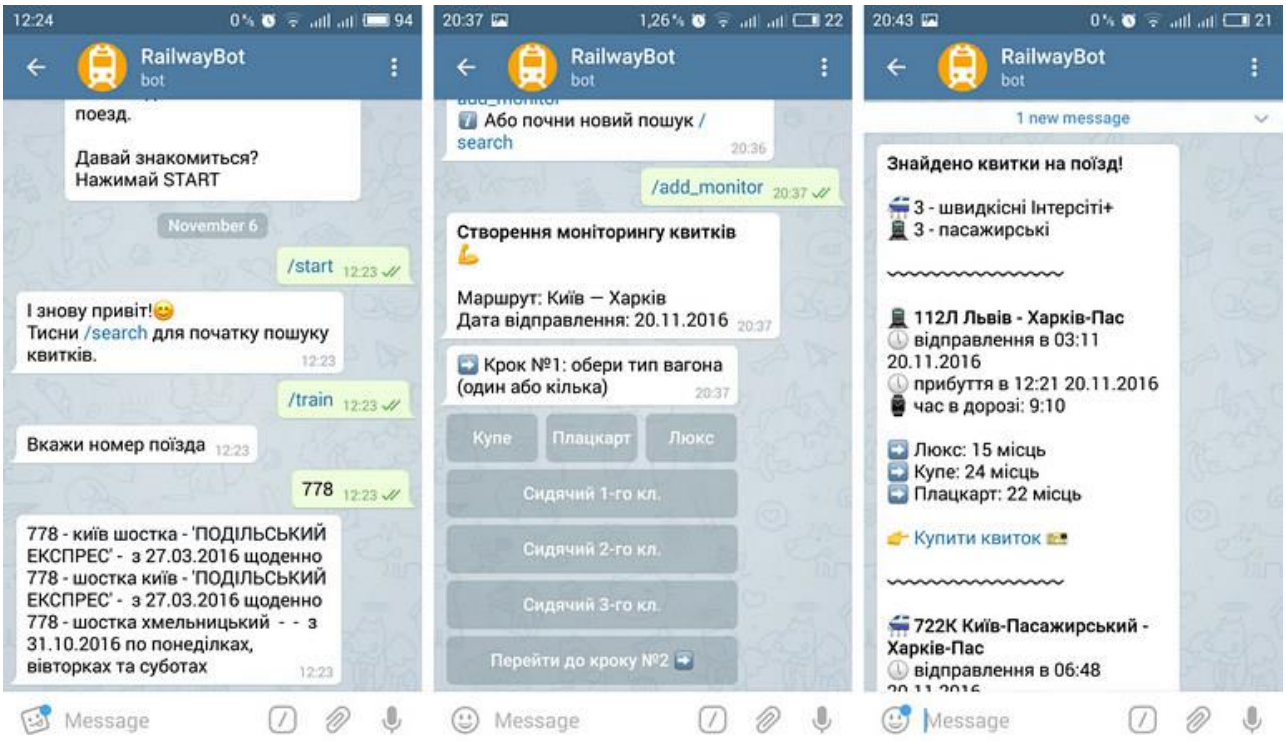


Рисунок 2.12 – Інтерфейс чат-боту Укрзалізниці

З наведеного прикладу графічного інтерфейсу можна виділити основні елементи, а саме: клавіатуру введення, списки кнопок які розподілені по горизонталі і вертикалі, текстові повідомлення з інформаційним вмістом. Наведений в прикладі інтерфейс є доволі простим і в той же час функціональним, такий підхід будемо використовувати при реалізації інтерфейсу нашого чат-боту. Також, не менш важливий елемент, на який варто звернути увагу – вітання бота з користувачем. За правилами системи Telegram, чат-бот не має права писати першим для користувача.

Для того щоб розпочалась розмова з користувачем, клієнт повинен розпочати її стартовою командою, в прикладі це /start, лише після неї бот вітається з користувачем, і відображає основні команди в згенерованому інформаційному повідомленні.

Наступним важливим моментом, на який потрібно звернути увагу – підтримка ресурсів мультимедіа. Користувач може завантажити в чат картинки, відео тощо використовуючи функціонал чату Telegram. В наведеному прикладі відображений інтерфейс з підтримкою двох різних мов, для зручності і кращого

розуміння користувачів потрібно реалізувати зрозумілий інтерфейс, щоб перемикати мову.

Відповідно поставлених задач в технічному завданні додаток Б створимо інтерфейс користувача для першого входу користувача в систему веб-додатку, який зображений на рисунку 2.13.

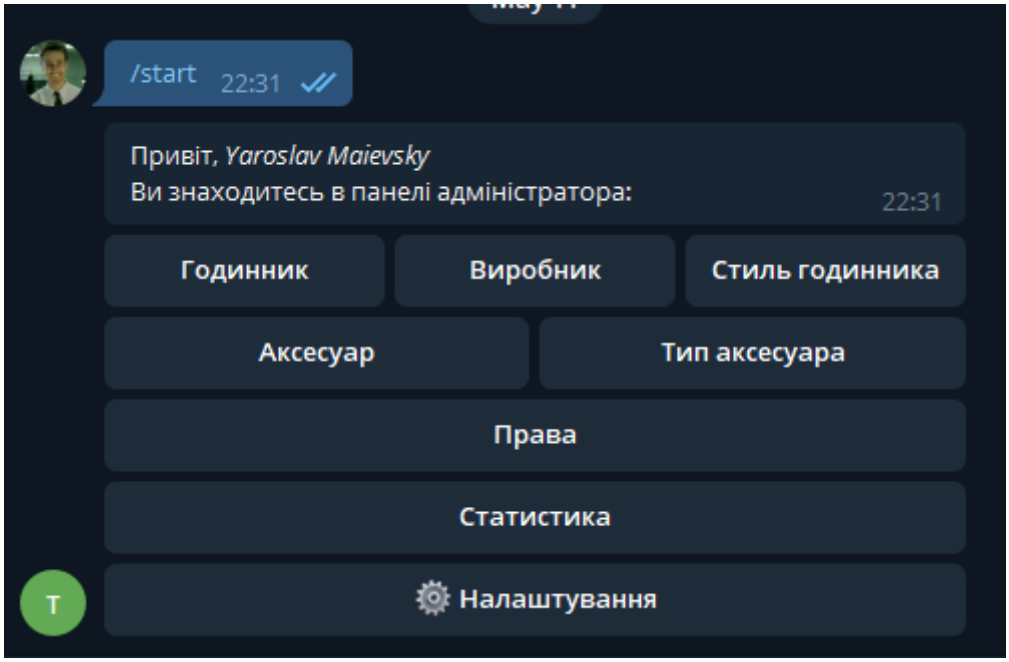


Рисунок 2.13 – Інтерфейс адміністратора чат-боту.

Відповідно задачі, користувач при першій взаємодії з ботом попадає в головне вікно магазину. В інформаційному повідомлені міститься вітання з користувачем і статус сторінки. При натиску на інтерфейс користувача зміниться на вибраний варіант.

Кнопки: переводять користувача на пагінацію на відповідно вибрану сторінку, тобто годинник – сторінка існуючих годинників рис 2.14, виробник – пагінація виробників, аексесуар і тип аксесуара – відповідно. Так як в наведеному прикладі інтерфейс адміністратора магазину, йому доступна кнопки права і статистика, яка веде до пагінації існуючих користувачів в системі. Налаштування є загальним функціоналом для всіх користувачів, надає можливість для зміни мови інтерфейсу чат-бота.

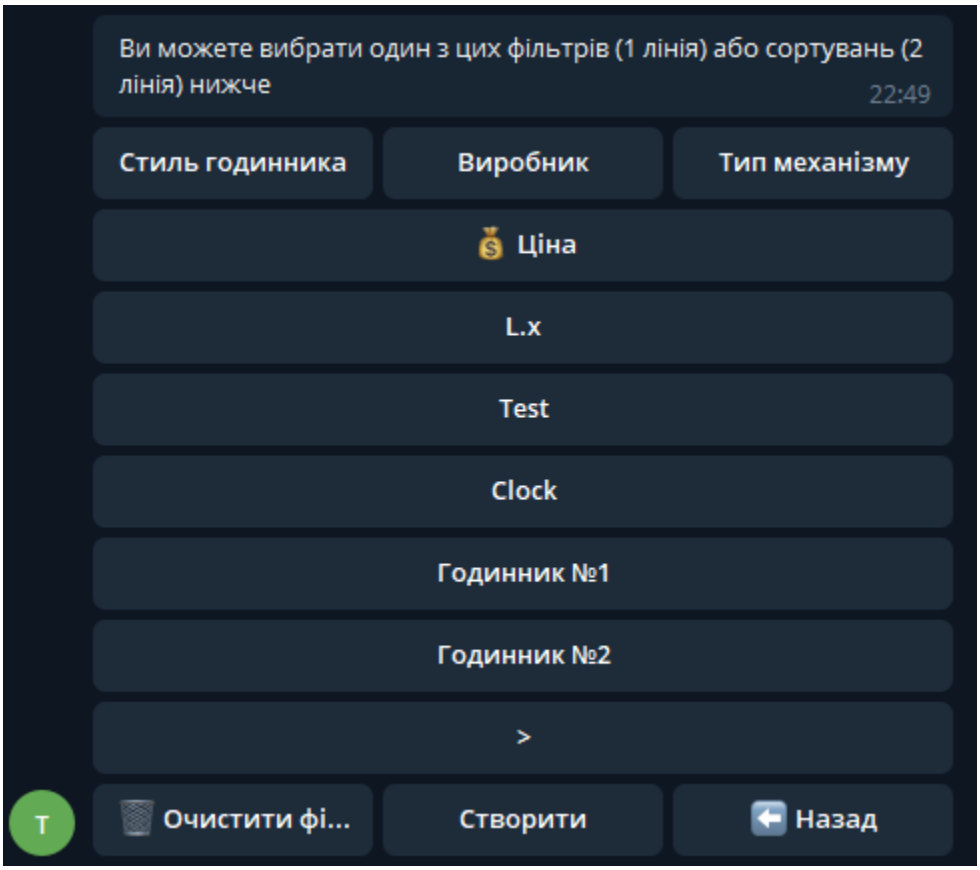


Рисунок 2.14 – Пагінація годинників в чат-боті

Інтерфейс перегляду годинників в магазині є важливим моментом, і тому для максимальної зручності користувачів системи відповідно технічному завданню було реалізована пагінація годинників в системі. На одній сторінці відображається 6 годинників – обмеження по кількості кнопок в одному повідомленні, інакше в іншому випадку всі кнопки не влізли ю на одну сторінку пагінації. В інформаційному повідомленні міститься опис теперішньої сторінки і активні фільтри і сортування товарів. Приклад, з використанням усіх можливих фільтрів і сортувань на цій сторінці зображений на рисунку 2.15.

Для того щоб видалити активні сортування чи фільтрацію, було розроблене відповідно меню, яке зображене на рисунку 2.16. Важливим моментом є те, що для вибору фільтру користувач переходить на міні меню, з вибором варіанта для фільтрів, а для сортування користувачу достатньо натиснути кнопку сортування, для прикладу «Ціна».

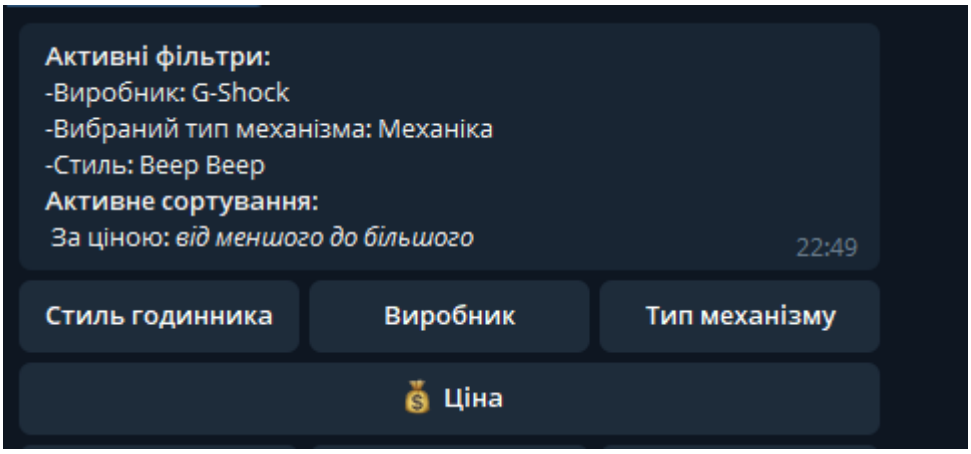


Рисунок 2.15 – Інформаційне меню з активними фільтрами

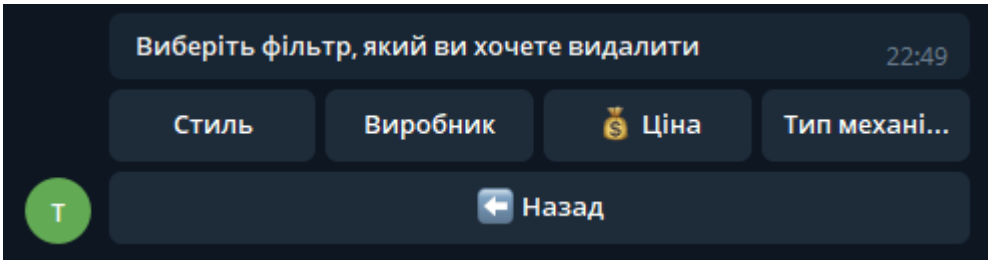


Рисунок 2.16 – Меню видалення активних фільтрів

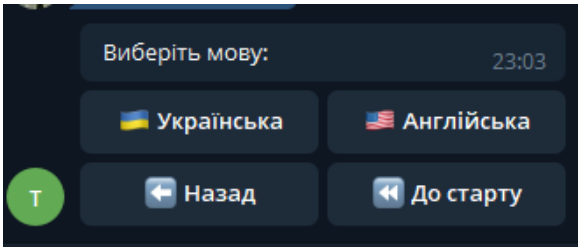


Рисунок 2.17 – Меню налаштування мови інтерфейсу

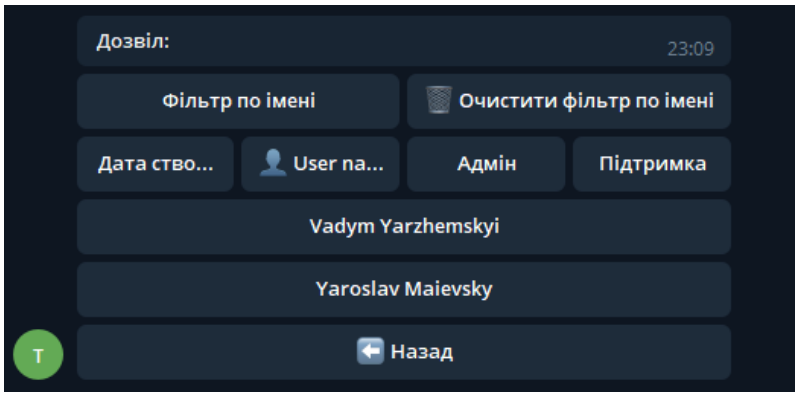


Рисунок 2.18 – Меню керуванням дозволами користувачів

Відповідно технічому завданню було розроблено інформаційне меню для перегляду детального опису годинників. В наведеному прикладі на рисунку 2.19 зображено детальний опис годинників, а саме назву, опис, технічні параметри годинника, посилання на магазин, і використовуючи інструменти телеграма було реалізовано підтримку медіафайлів, а саме. На рисунку 2.19 відображено зображення інформаційного повідомлення з інформацією про годинника.

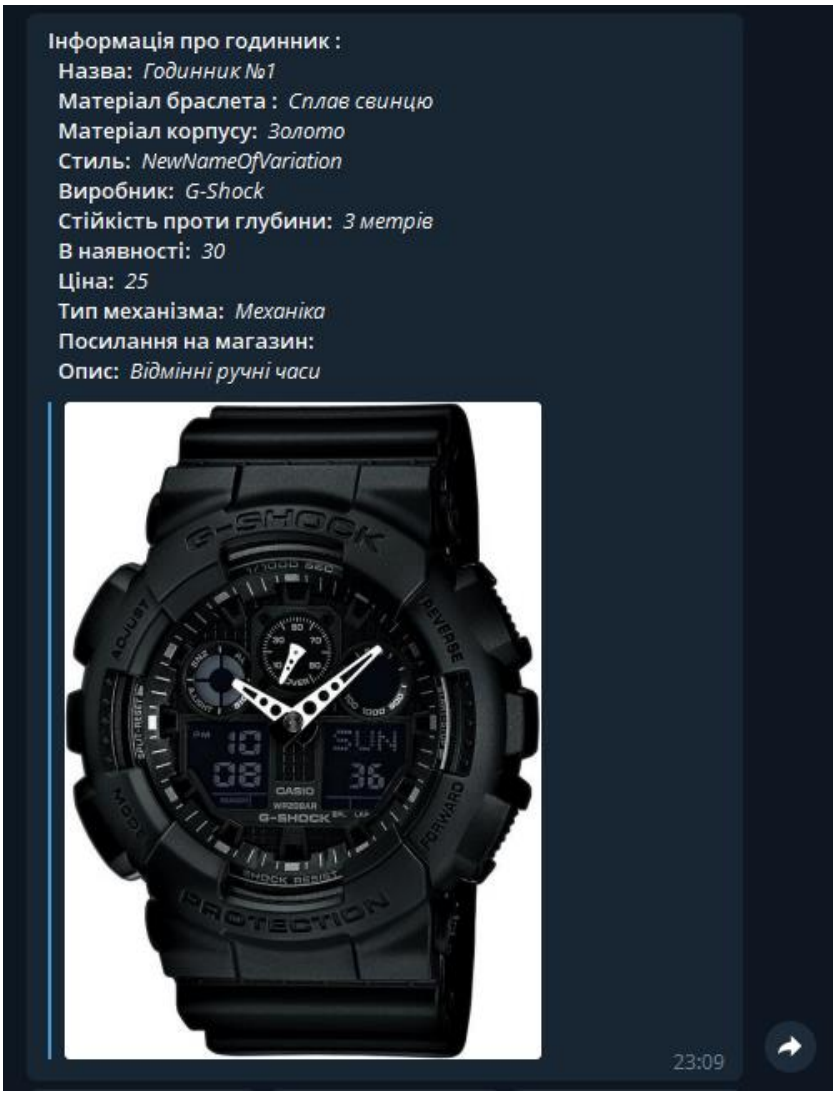


Рисунок 2.19 – Меню керуванням дозволами користувачів

Для меню створення і підтвердження було створено меню, зображене на рисунку 2.20. В інформаційному полі відображаються контактні дані користувача системи, а саме: ПІБ, адреса, мобільний телефон, а також описані товари до сплати, їх ціна і кількість.

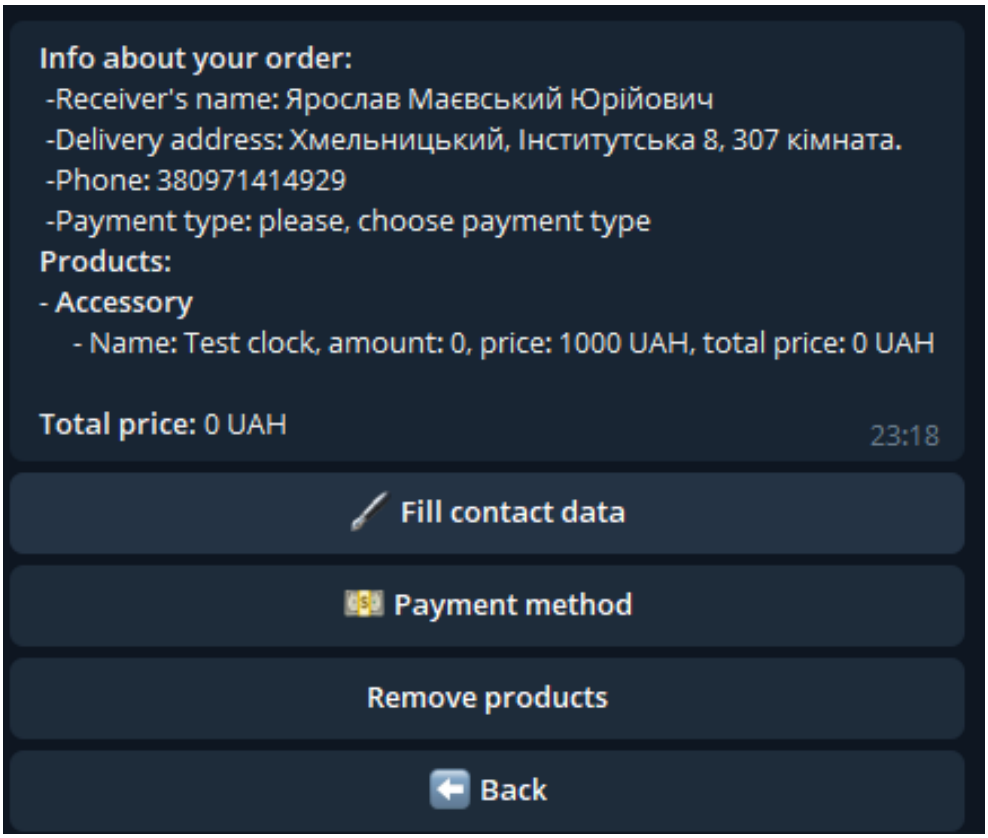


Рисунок 2.20 – Меню перегляду актуального замовлення

Якщо користувач потребує змінити контактні дані, для зручності було реалізована кнопка переходу до персональної інформації користувача. Також в цьому меню користувач може видаляти товари з кошика. Після того як користувач заповнив інформацію про себе, може підтвердити замовлення.

2.5 Аналіз та вибір технологій і методів реалізації системи

Відповідно детального аналізу розроблюваного програмного забезпечення в розділі 2.2 ми обрали серверну архітектуру для нашого веб-додатку, відповідно, технології реалізації обов'язково мають підтримувати передачу даних за допомогою мережевого стеку HTTP, тобто ми реалізовуємо Back-end додаток. Back-end – це рівень, який з'єднує користувацький інтерфейс з базою даних, повертає відповіді від сервера і забезпечує роботу інтерфейсу. Також дослідивши предметну область ми зрозуміли, що розроблюваним продуктом є інтернет

магазин, тому технологія, яку ми будемо використовувати при реалізації має бути безпечною, так як веб-додаток взаємодіятиме з персональними даними клієнтів.

Наступним важливим критерієм є те, що веб-додаток має підтримувати масштабування для розподілу і балансування навантаження на програмне забезпечення, так як навантаження на ПЗ буде зростати відповідно кількості активних користувачів системи. Наступним важливим аспектом є те, що клієнти додатків часто будуть переглядати однакову інформацію, на приклад – наявні годинники в системі, отже технологія має підтримувати системи кешування інформації. Це дасть можливість зменшити навантаження на базу даних. Проаналізувавши вище описані вимоги до технології розроблюваного програмного забезпечення ми можемо їх виділити, а саме:

- розвинута та гнучка система безпеки;
- висока швидкодія;
- підтримка багатопоточності;
- підтримка мережевої взаємодії;
- взаємодія з базами даних;
- підтримка кешування інформації.

Детально проаналізувавши вимоги до технології розробки було обрану мову програмування Java, це строго типізована об'єктно орієнтована мова програмування загального призначення, розроблена компанією Sun Microsystems, а на даний момент підтримується компанією Oracle. Ця мова програмування є універсальною завдячуючи віртуальній машині Java Virtual Machine, вона дозволяє коду на Java однаково працювати на всіх підтримуваних платформах віртуальної машини [7].

JVM – це свого роду обгортка, в якій Java програма перетворюється в код, який може запускатись на будь якій машині. Індекс TIOBE 2021 ставить Java на друге місце серед актуальних технологій для Back-end розробки. Цей рейтинг, зображений на рисунку 2.21 показує, що Java є одним із самих впливових мов програмування на сьогоднішній день.

До головних аргументів в користь описаної мови програмування входять наступні переваги: підтримка масштабування, багатопоточності, високої безпеки, велика кількість бібліотек для рішення різноманітних задач, висока швидкодія, велика спільнота і якісна документація, мультиплатформеність, підтримка ООП, зворотна підтримка старіших версій

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	▲	Python	11.87%	+2.75%
3	2	▼	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%

Рисунок 2.21 – Рейтинг мов програмування в 2021

На теперішній час Java потужна мова, яка надає багато інструментів. Але все ж таки, розробка додатків по типу магазину складно побудувати без додаткових інструментів, бібліотек чи фреймворків. На даний момент, самий популярний і рекомендований Oracle фреймворк для розробки комерційних додатків на Java є Spring. Цей фреймворк створений для того, щоб полегшити процес розробки на Java стеку. На відміну іншим платформам, Spring акцентує увагу на декількох областях функціонування додатку і надає до них широкий спектр додаткових функцій. Однією з таких особливостей є реалізація в фреймворку патерну Dependency Injection (DI, впровадження залежностей). DI допомагає розробникам простіше реалізовувати необхідну додатками функціональність, а також розроблювати слабо зв'язані класи, роблячи їх більш універсальними. Також важливим моментом є те, що Spring Framework фокусується на надані додаткової гнучкості додатку у вигляді різноманітних автокофігурацій і вбудованого сервера. Також перевагами цього фреймворку є: використання POJO при створенні класів, підтримка декларативного програмування, самостійне створення фабрик і сінглтонів, різні способи

конфігурацій, сервіс рівня middleware, вбудований Tomcat, функціональність із коробки і простий процес налаштування.

Розроблюваний додаток має зберігати інформацію про годинники, користувачів, замовлення, тощо. Для рішення цієї задачі в розділі 2.3.1 ми дійшли до висновку, що для підтримки продукту нам потрібні реляційна і нереляційна база даних, отже нам потрібно вибрати які саме бази даних будуть використовуватись для цих задач, почнемо з реляційної бази даних.

Основними вимогами до реляційної бази даних є:

- висока швидкодія;
- підтримка кешування;
- підтримка індексів;
- гнучкість структур для зберігання інформації;
- підтримка нестабільності роботи сервера.

Під описані вимоги вище краще всього підходить реляційна база даних MySQL, вільна реляційна система керування базами даних. Сильними сторонами цієї бази даних є те, що вона підтримує багато поточність що гарантує високу швидкість роботи, вона повністю безкоштовна, має відмінну підтримку транзакцій, підтримує більшість необхідних типів даних. Ця база даних відмічається відмінною надійністю і стабільністю роботи.

У цьому розділі ми визначили, що наша система буде побудована за клієнт-серверною архітектурою. Було проведено аналіз серверних архітектурних підходів, та була обрана мікросервісна архітектура, враховуючи її переваги в зрозумілості компонентів коду перед монолітною. Були виділені 4 мікросервіси та визначені їх взаємодії при виконанні необхідного функціоналу. Був проведений аналіз типів баз даних, та обрано реляційну MySQL і не реляційну MongoDb у зв'язку зручності їх використання, було визначено структуру сутностей для БД. Також ми спроектували користувацький інтерфейс чат бота, а також обрали інструменти для програмної реалізації веб-додатку.

.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування мікросервісів

Наступним важливим кроком є детальне проектування мікросервісів, які ми описали в розділі 2.2. Розпочнемо з того, що потрібно обрати загальний патерн побудови для кожного сервісу, відповідно проаналізувавши існуючі концепції було обрано патерн розбиття сервісів по відповідним шарам відповідальності. Суть цього партерна в тому, що він визначає межі між додатком і шаром сервісів, який утворює набір доступних операцій і управляє результатом додатку в кожній операції. Тобто, цей патерн визначає для додатка набір допустимих операцій з точки зору клієнта. Він інкапсулює бізнес-логіку додатку, керуючи транзакціями і керуючи відповідями в реалізації цих операцій.

Отже, варто виділити основні модулі, які будуть містити в собі усі сервіси розроблюваного магазину, а саме:

- web-api рівень;
- рівень бізнес логіки;
- трансформери;
- обробка помилок;
- інтеграції з іншими ресурсами;
- конфігурація;
- константи;
- рівень взаємодії з репозиторієм;
- util модуль.

Web-API рівень – відповідає за відображення інформації користувачу і інтерпретації запитів клієнтів. Клієнтом може виступати як і користувач системи, так і інший і мікросервіс. Мікросервіс отримує запити за допомогою HTTP, також має підтримувати принципи REST для кращої зручності. REST – набір принципів і обмежень, які в результаті приводять до прискорення швидкодії і спрощення архітектури додатку

Підсумувавши, можна зрозуміти що цей модуль використовується як вхідна точка в мікросервіс, тобто клієнт відсилає запит на ресурс, а web-api рівень його отримує, аналізує вхідні параметри і передає в відповідний сервіс бізнес логіки, з результату якого формує зворотну відповідь, і вертає її клієнту.

Бізнес рівень – це частина корпоративної системи, яка визначає, як оброблюються певні дані, оброблює їх або обраховує. В прикладі розроблюваного нами продукту, логіка за створення нових замовлень або перегляд і редагування існуючих – це бізнес логіка, тобто механізми, які вирішують бізнес задачі додатку.

Репозиторії - компоненти, які надають ряд інструментів для взаємодії з базою даних. Зазвичай, один репозиторій працює тільки з одним типом сутності. Усі методи для пошуку за певними умовами, зберігання, видалення реалізовуються в цих компонентах .

Для того, щоб оброблювати усі можливі помилки в веб додатку створюються відповідні модулі. Зазвичай, в їх задачі входить відловлювання помилки, аналіз при яких умовах вона з’явилась, генерування відповідного повідомлення. Так як наший додаток за типом є серверним, відповідно модуль опрацювання помилок в залежності від ситуації в якій появилась помилка в, має обрати код для мережевої відповіді на запит, в разі помилки.

Для інтеграції з іншими веб-ресурсами створюються окремі модулі, які ще називають communication services (сервіси спілкування). Основним обов’язком такого сервісу є створення нових HTTP запитів до інших ресурсів, а також опрацювання відповіді запиту і обробка помилок зв’язаних з інтеграцією.

Часто в практиці, додатки потрібно конфігурувати в під певні умови. На приклад, потрібно відключити модуль інтеграції з платіжною системою. Для рішення цієї проблеми створюються конфігураційні класи. Зазвичай, вони зчитують параметри налаштування і застосовують на логіку додатку. На приклад: конфігурація запитів: очікування відповіді обмеження розмірів вхідних запитів, підтримуванні розширення ресурсів, інтерсептори при отримуванні запитів.

Усі методи, функції які складно класифікувати за призначенням зберігають в util компонентах. На приклад обрахування або перетворення під певні умови стан певного об’єкта, тощо.

Відповідно патерну, кожний з сервісів має працювати тільки з одним типом даним, на приклад: web з dto об’єктами, бізнес логіка з моделями, рівень взаємодії з репозиторієм працює з entity. Для рішення цієї проблеми створюються трансформери. Трансформер, це компонент додатку, який містить в собі інструменти перетворення об’єкта з одного формат в інший, на приклад : запит в собі містить dto, після валідації контролер за допомогою трансформера перероблює дані в формат моделі, і передає цю модель в бізнес рівень.

Перейдемо до детального проектування окремих мікросервісів, розпочнемо з be-client. Даний мікросервіс містить в собі усі вище описані модулі. В його основні задачі входить отримування запитів від Telegram, аналіз і опрацювання їх. Тобто, цей мікросервіс містить в собі контролери, які отримують запити, в яких зберігається інформація про дії користувачів з ботом. Уся логіка, яка зв’язана з взаємодією з Telegram виокремлена в пакет Engine, а логіка зв’язана з діями користувача в магазині виокремлена в пакет business.

Мікросервіс зберігає в базі даних інформацію про сесію користувача в Telegram, ця логіка зберігається в пакеті db, сутності в пакетах entity, а компоненти для взаємодії в пакеті repository. Відповідно вище описаній архітектурі, цей додаток є проміжним між Telegram і be-shop. Тому, логіка яка зв’язана з інтеграцією з іншими мікросервісами винесена в пакет integration. На приклад, методи для отримання інформації про користувача, отримання наявних товарів в системі, створення нового замовлення і тому подібне.

Також в мікросервісі існують компоненти, які виконуються циклічно за вказаним інтервалом. Такі механізми використовується в цьому мікросервісі для очищення чат-боту від застарілих діалогів, а також для відправки оповіщення користувачам, коли їх сесії застаріли. Ці компоненти зберігаються в пакеті Scheduler. Зазвичай, інтервал в таких компонентах вказується в форматі unix

часових систем, які називаються cron вказівники. Логіка для керуванням сесіями користувачів винесена в SessionService.

Admin-tg мікросервіс призначений для роботи адміністраторів і редакторів з магазином. Цей мікросервіс багато в чому схожий з client-tg, він так само отримує запити з діями користувачів від Telegram, механізми обробки повідомлень і зберігання інформації про сесії користувачів повністю аналогічні.

Але на відміну client-tg цей мікросервіс спроектований для взаємодії адміністратора з магазином, отже бізнес логіка в нього абсолютна інша, а саме: створення нових товарів – ClockActions і AccessoryActions, редагування існуючих, перегляд існуючих користувачів системи і редагування прав користувачів UserActions, перегляд статистики магазину – StatisticsActions.

Be-shop є ключовим мікросервісом. він надає API для client-tg і admin-tg, але також є доступним для інтеграцій з іншими мікросервісами. Цей мікросервіс зберігає усю інформацію магазину в реляційній базі даних, працює з нею за допомогою різних репозиторіїв. Замовлення створюються в сервісах CartActions, взаємодія з годинниками в ClockActions, а з аксесуарами в AccessoryActions. Створення нових товарів реалізовано в InventoryService . Логіка, яка зв’язана з користувачами і їхніми правами зберігається в сервісі бізнес логіки UserActions.

Якщо підсумувати вище сказане, be-shop отримує різні запити в контролери, аналізує і опрацьовує їх, відповідно за призначенням опрацьовує в сервісах бізнес логіки, результат записує в базу даних і вертає відповідь результатом на запит іншого мікросервісу.

Мікросервіс wrapper призначений для взаємодії з платіжною системою GeoPay. Він надає API для створення нових транзакцій, відміни, і перегляду стану існуючих замовлень. Основна логіка цього мікросервіса – за вказаним інтервалом переглядати стан існуючих замовлень в системі. Якщо замовлення застаріло – відмітити його відповідно, якщо замовлення оплачене в платіжній системі – позначити замовлення як оплачене і вказати користувачу, що його замовлення успішно сплачено, і він може отримати замовлення.

3.2 Розробка програмних модулів

Усі мікросервіси були побудовані за допомогою фреймворку Spring, який сильно спростив процес реалізації програмного забезпечення, так як багато задач Spring бере на свою відповідальність. На приклад великі фрагменти коду часто повторюються, і щоб уникнути повторювання – Spring надає стереотипи, анотації які інкапсулюють повторюваний код. Усі додатки потрібно запускати на певному сервері, і конфігурувати його. Фреймворк повністю бере цю задачу на себе, в наведеному далі фрагменті коду використовується стереотип @SpringBootAplication, це означає що фреймворк при запуску програми запустить окремий TomCat сервер, і запустить на ньому програму.

```
@EnableMongoRepositories
@EnableScheduling
@SpringBootApplication
public class ClientTelegramApplication {
    public static void main(String[] args) {
        ApiContextInitializer.init();
        SpringApplication.run(ClientTelegramApplication.class, args);
    }}
```

Цей фрагмент є точкою ініціалізації tg-client. Також можна замітити анотації @EnableMongoRepositories, яка вмикає підтримку роботи з базою даних Mongo, а також анотацію @EnableScheduling, яка вказує що в програмі є класи, логіка яких виконується циклічно з певним інтервалом. В кожному мікросервісі містяться файли властивостей application.yml – вони зберігають в собі конфігураційну інформацію додатку у вигляді ключ-значення, на приклад на якому порту запускається додаток, на якій адресі працює база даних, токен для доступу до телеграм бота, URI інших мікросервісів і багато інших важливих властивостей. Також в кожному проекті є файл з описом залежностей build.gradle, який потрібний для побудови додатку за допомогою Gradle.

Переглянемо далі реалізації основних модулів на прикладі be-shop. Розпочнемо з пакета api який відповідає за опрацювання усіх HTTP запитів до мікросервісу. Пакет містить в собі класи, які за своїм функціоналом є контролером, такі контролери позначаються за допомогою spring анотації @RestController, яка в свою чергу є ідентифікатором для IoC фреймворку. Тобто, коли додаток буде отримувати сторонні запити, фреймворк буде провіряти, чи немає в цих контролерах методів, які підходять під параметри запиту. В пакеті Api містяться контролери для : замовлень, транзакцій, товарів, інвентарю, користувачів, статистики. Для прикладу розглянемо UserController.

```
@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("api/user")
public class UserController {
    private final UserService userService;
    @GetMapping("/{id}")
    public ResponseEntity<UserDto> getById(@PathVariable int id) {
        UserDto byId = UserTransformer.transform(userService.getById(id));

        sout.debug("User {} found", byId);
        return new ResponseEntity<>(byId, HttpStatus.OK);
    }
@PostMapping
public ResponseEntity<UserDto> create(@RequestBody UserDto userDto) {
    userDto.setAdmin(false);
    userDto.setSupport(false);
    UserDto save =
UserTransformer.transform(userService.saveNewUser(UserTransformer.transform(user
Dto)));
    sout.debug("User {} saved", save);
    return new ResponseEntity<>(save, HttpStatus.CREATED);}
    @PutMapping("/{id}")
    public ResponseEntity<UserDto> update(@PathVariable int id, @RequestBody
UserDto user) {UserDto update = UserTransformer.transform(userService.update(id,
UserTransformer.transform(user)));
        sout.debug("User with id:{} updated to {}", id, user);
        return new ResponseEntity<>(update, HttpStatus.CREATED);
    }
```

В наведеному прикладі реалізований функціонал для пошуку користувача по ідентифікатору, і для створення нового користувача в магазині . Можна замітити, що використовується нотація @RequestMapping вказує, по якому URI цей контролер буде отримувати запити. В методи попадають класи, в префіксі яких є приставка DTO – data transfer object. Це свого роду флаг об’єкта, що він

попав в додаток ззовні. В кожному методі використовується метод для трансформації об’єкта в модель, об’єкт з яким працює сервіс. Можна заміти анотацію @Slf4 – вона додає в клас механізм логування, приклад якого можна побачити в методах UserController-a. Такі логи зазвичай записуються в файл, за допомогою якого у випадку помилки в програмі можна буде зрозуміти, в якому місці появилась помилка і чому. В пакеті API є пакет dto – він зберігає в собі об’єкти, які приходять в додаток в http запитах. В http запитах вони відображаються як текст, але коли вони потрапляють в додаток – Spring автоматично за допомогою вбудованих бібліотек сереалізує такі об’єкти з тексту в Java класи. Приклад такого об’єкта наведений далі.

```
@ Data
@ Builderr
@ NoArgsConstructor
@ AllArgsConstructor
public class UserDto {
    private Integer id;
    private Long chatId;
    private Integer telegaId;
    private String telegaFullName;
    private String telegaUserName;
    private String language;
    private String fullName;
    private String phone;
    private String deliveryAddress;
    private Boolean admin;
    private Boolean support;
    private Boolean modified;
    private Boolean created;
    private Boolean superAdmin;}
```

Для анотацій @Data, @Builder, @NoArgsConstructor, @AllArgsContructor – підтримку надає бібліотека Lombok. Ці анотації під час компіляції JVM генерують код. @data генерує методи ToString, геттери і сетери на усі поля класу, а також конструктори для створення об’єкта, і методи для хеш коду, і equals.

@NoArgsContructor генерує конструктор без аргументів, такий клас потрібний для серіалізації JSON в JAVA клас. @Builder реалізує в класі патерн будівник, який можна використовувати як альтернативу конструктора, де кожне поле створюваного класу передавати як виклик нового метода. Наступним важливим компонентом є класи Трансформери, вони містять методи для перетворення DTO в сутності. Приклад такого класу наведений далі.

```
@NoArgsCon3structor(access = AcceessLevel.PRIVATE)
public class UserTransformer {
    public static User transform(UserDto dto) {
        return transform(null, dto);}
    public static User transform(Integer id, UserDto dto) {
        return Optional.ofNullable(dto)
                .map(usr -> User.builder()
                        .id(Optional.ofNullable(id).orElse(dto.getId()))
                        .telegaId(usr.getTelegaId())
                        .telegaFullName(usr.getTelegaFullName())
                        .telegaUserName(usr.getTelegaUserName())
                        .language(usr.getLanguage())
                        .chatId(usr.getChatId())
                        .fullName(dto.getFullName())
                        .phone(usr.getPhone())
                        .deliveryAddress(dto.getDeliveryAddress())
                        .admin(usr.getAdmin())
                        .support(usr.getSupport())
                        .superAdmin(usr.getSuperAdmin()).build()).orElse(null);}
```

В даному фрагменті DTO об’єкт перероблюється в клас сутність, з яким далі може працювати бізнес сервіс або репозиторій. Розглянемо далі пакет config, в якому зберігаються конфігураційні компоненти, на приклад

```
@Configuration
@EnableScheduling
public class Config {
    @Value("${rounding.scale}")
    private Integer scale;
    @PostConstruct
```

```
    public void setUp() {
        PriceTransformer.setScale(scale);}
    @Bean
    public Clock utcClock() {
        return Clock.systemUTC();}}
```

В фрагменті є нотація @Configuration, флаг для Spring, який вказує що клас є конфігураційним, і запускається лише один раз при побудові додатку. @Value є аналогом сетера для поля, який дістає по вказаному шляху потрібне значення з application.properties. Це дає змогу динамічно налаштовувати додаток. В пакеті Constants зберігаються константи, які використовуються в усьому додатку, на приклад тип нотифікацій для бота :

```
public enum NotificationType {
    ADVERTISEMENT,
    NEW_ORDER,
    NEW_PAID_ORDER_SUPPORT,
    NEW_PAID_ORDER_CUSTOMER;}
```

В пакеті cron зберігаються класи, логіка яких виконується періодично за інтервалом, на приклад: видалення застарілих замовлень, відправка нотифікацій застарілих сесіям боту, тощо. Приклад такого класу наведений далі:

```
public class TelegramBotNotificationCron {

    private final NotificationProperties notificationProperties;
    private final NotificationService notificationService;
    private final TelegramBotCommunicationService
telegramBotCommunicationService;

    @Scheduled(cron = "${cron.telegram.bot.story.notification}")
    public void notification() {
        sout.info("Run story notifications for TeelgramBot");
        List<NotificationDTO> notificationDTOs =
NotificationTransformer.transform(notificationService.getNotificationsForSending
(notificationProperties.getCount()));
        Set<String> storyTexts = new HashSet<>();
        notificationDTOs.forEach(notification -> {try {
                List<Integer> userStoryIds = notification.getUserStoryDTOs()
                    .stream()
                    .map(UserStoryDTO::getUserStoryId)
                    .collect(Colectors.toList());
notificationService.updateStatusInNotification(StoryStatus.IN_PROCESS.name(),
userStoryIds);
```

```
notificationService.updateStatusInStory(StoryStatus.IN_PROCESS.name(),
notification.getText());

                telegramBotCommunicationService.publishToTelegram(notification);

notificationService.updateStatusInNotification(StoryStatus.DONE.name(),
userStoryIds);
                storyTexts.add(notification.getText());
            } catch (Exception e) {
                sout.error(e.getMessage());
            }
        });
@Scheduled(cron = "${cron.telegram.bot.story.notification}")
```

Важливим моментом таких класів є те, що щоб увімкнути циклічну роботу, потрібно в анотації вказати інтервал в форматі cron, на приклад: виконати операцію кожних дві години 0 */2 * * *. @RequiredArgsConstructor генерує інкапсульований конструктор.

Далі переглянемо деталі реалізації основних компонентів на прикладі пакета exception, так як він є у всіх мікросервісах має аналогічну реалізацію. Головну задачу, яку він вирішує можна описати так: якщо під час виконання HTTP запита на стороні сервера відбувається якась помилка - цей мікросервіс має згенерувати відповідь на запит клієнта, в результаті якого має зберігатись зрозуміла інформація про помилку. Структура, яка зберігає в собі інформацію про помилку наведена далі:

```
@ !Getter
@ Setter
@ Builderr
@ NoArgsConstructor
@ AllArgsConstructor
public class ErrorInfoDto implements Serializable {
    @NotNull
    private Integer status;
    @NotNull
    private List<String> messages;
    @NotNull
    private String debugMessage;
    @NotNull
    private Integer code;
    private String header;
```

private List<ErrorDetail> errorDetails;}

Значення для code – це оригінальний ідентифікатор для помилки, яка може з’явитись під час виконання запитів в додатку, а messages містять текстовий опис помилки. Усі інформаційні коди помилок зберігаються в enum ExceptionCode в форматі тип помилки, код, опис помилки. Усі помилки, розширюють клас RuntimeException.

```
@ Getter
public class CheckoutException extends BaseException {
    @ Getter
    private final List<ErrorDetail> errorDetails;
    public CheckoutException(String message, List<ErrorDetail> errorDetails) {
        super(ExceptionCode.CHECKOUT_EXCEPTION, message);
        this.errorDetails = errorDetails;
    }
}
```

Для опрацювання помилок використовуються механізми @ControllerAdvice, такі механізми відображають помилки під час виконання запитів до мікросервіса, і в разі якщо з’явиться помилка – цей механізм відловить її і сформує з неї коректне інформативне повідомлення. В прикладі далі наведений фрагмент, який будує відповідь у випадку помилки при оплаті замовлень.

```
    @ExceptionHandler(value = CheckoutException.class)
    public ResponseEntity<ErrorInfoDto>
handleMethodArgumentForCheckoutException(CheckoutException e, HttpServletRequest
request) {sout(request, e);
        ErrorInfoDto errorInfoDto = createErrorDto(e);
        errorInfoDto.setErrorDetails(e.getErrorDetails());
        return new ResponseEntity<>(errorInfoDto,
e.getExceptionCode().getHttpStatus());
:
```

Логіка, яка виконається в разі появи помилки під час процесу виконання запита описана в методах з анотацією @ExceptionHandler, в аргумент якої передається помилка. Значення що повертається з такого метода сереалізується в Http відповідь на запит до мікросервіса.

Інтеграції з іншими мікросервісами зберігаються в пакеті Integration. Для кожного мікросервіса виокремлені пакети під конфігурацію, DTO, властивості, URI посилання. Розглянемо для прикладу інтеграцію з tg-client.

```
@Configuration
public class TelegramBotRestTemplateConfig {
    @Bean(name = "telegramBotRestTemplate")
    public RestTemplate telegramBotRestTemplate(RestTemplateBuilder
restTemplateBuilder,
            TelegramBotProperties telegramBotProperties) {
        restTemplateBuilder =
restTemplateBuilder.rootUri(telegramBotProperties.getUri())
.setConnectTimeout(Duration.ofMillis(telegramBotProperties.getTimeout().getCoenn
ect()))
.setReadTimeout(Duration.ofMillis(telegramBotProperties.getTimeout().getRead()))
;return restTemplateBuilder.build();}}
```

В наведеному фрагменті конфігурується з’єднання з мікросервісом, описується максимальний час на отримування відповіді, і максимальний час на встановлення з’єднання, тощо.

```
@Override
public void userSoutout(Integer id) {
    URI uri = telegramBotUriBuilder.getUserSoutoutUri(id);
    try {
        sout.info("Request to sout out user with id {} on client-tg. ", id);
        telegramBotRestTemplate.exchange(uri, HttpMethod.POST, HttpEntity.EMPTY,
Void.class);
        sout.info("Request to sout out user with id {} on clietn-tg processed.",
id);
    } catch (ResourceAccessException e) {
        String message = String.format("Failed sending soutout request to
client-telegram bot service with error message:%s.", e.getMessage());
        throw new InnerServiceException(message);
    }
}
```

На цьому прикладі зображено запит до client-tg, а саме як відправляються HTTP запити від одного мікросервіса до іншого. Спочатку зчитується з application.properties URI до ресурсу, і вже тоді RestTemplate створює новий запит використовуючи конфігурацію з пакета config. Не складно замітити особливість усіх сервісів, кожний сервіс імплементує свій інтерфейс. Інтерфейси слугують контрактами, які описують методи сервісу, а сервіс вже імплементує такий контракт. Сервіси, які виконують бізнес логіку зберігаються в окремому пакеті service, для прикладу розглянемо один таких як CartProcessingService, логіка якого відповідає за опрацювання замовлень.

```
@Slf4j
@Servce
@RequiredArgsConstructor
public class CartProcessingService {
@Transactional
public void findExpiredCarts() {
    LocalDateTime expiredTime =
LocalDateTime.now().minus(cartProperties.getAmountToSubstract(),
cartProperties.getUnit());
    List<Cart> carts = cartService.findExpiredCarts(expiredTime);
    carts.forEach(c -> {
        rollbackInventoriesReservedCountForCartItems(c.getId());
        c.setStatus(CartStatus.EXPIRED);
    });
    cartService.saveAll(carts);
    List<String> cartsIds = carts.stream().map(cart ->
cart.getId().toString()).collect(Colectors.toList());
    sout.info("Next orders were expired: {}", cartsIds);}
```

Методи, які взаємодіють з базою даних покриті анотацією @Transactional, це означає, що якщо під час виконання цього метода з’явиться помилка – усі дії, які були виконані цим методом відміняться, і сутності з якими взаємодіяв цей фрагмент коду вернуться в стан до початку операції.

Все що зв’язане з взаємодією з базою даних зберігається в пакеті data, а саме інтерфейси, які описують взаємодію з базою даних для певних сутностей, а також відображення об’єктів – entity. Такі сутності є реальним відображенням існуючих таблиць в базі даних. Назви полів в такому класі мають співпадати до назв стовбців в таблиці. Для прикладу, розглянемо сутність яка описує інформацію про користувача

```
@Data
@Entity
@Builder
@NoArgsConstructor
@ AllArgsConstructor
@Table(name = "user")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private Integer telegaId;
    private String telegaFullName;
    private String telegaUserName;
    private String language;
    private Long chatId;
    private Boolean admin;
    private Boolean support;
    private Boolean superAdmin;
    private BigDecimal bonus;
    private String fullName;
    private String phone;
    private String deliveryAddress;}
```

Відповідно до цієї сутності є інтерфейс UserRepository, який описує взаємодію цієї сутності з базою даних. Щоб Spring розумів, що цей контракт описує взаємодію з базою даних, необхідно вказати нотацію @Repository. Зазвичай, немає необхідності імплементувати такі інтерфейси, так як код генерується відповідно сигнатури метода.

```
@Repository
public interface UserRepository extends CrudRepository<User, Integer>,
UserRepositoryCustom {
    User findByTelegaId(Integer telegaId);
```

Але, якщо потрібно написати свою реалізацію взаємодії з базою даних, потрібно імплементувати UserInterfaceCustom, і в ньому описати логіку.

```
@Servce
public class UserRepositoryImpl implements UserRepositoryCustom {

    @PersistenceContext
    private EntityManager entityManager;
@Override
public List<UserChatIdLocale> getUserInfoForAdvertisement() {
    String queryString = "SELECT u.chat_id, u.language FROM user u WHERE u.admin
= 0 AND u.support = 0;";
    return getUserChatIdLocaleInfo(queryString);
}
```

```
@Override
public List<UserChatIdLocale> getSupportInfoForNotification() {
    String queryString = "SELECT u.chat_id, u.language FROM user u WHERE
u.support = 1;";
    return getUserChatIdLocaleInfo(queryString);
}
```

Такий механізм дає нам змогу повністю контролювати взаємодію з БД і використовувати нативні запити до бази даних. Нативні запити часто використовуються в мікросервісах, так як Spring Data погано працює з запитами до багатьох таблиць, що призводить до проблеми N+1, в той час нативні запити дозволяють уникнути таких проблем.

3.3 Керівництво користувача

Перш за все, для того щоб користувач міг працювати з ботом, користувачу потрібно знайти його в Telegram, скористувавшись вбудованим пошуком месенджера, і тоді добавити його в список своїх активних контактів. Взаємодія з чат-ботом розпочинається з активації стартової команди /start користувачем чат-боту. Після цієї команди бот привітається з користувачем системи.

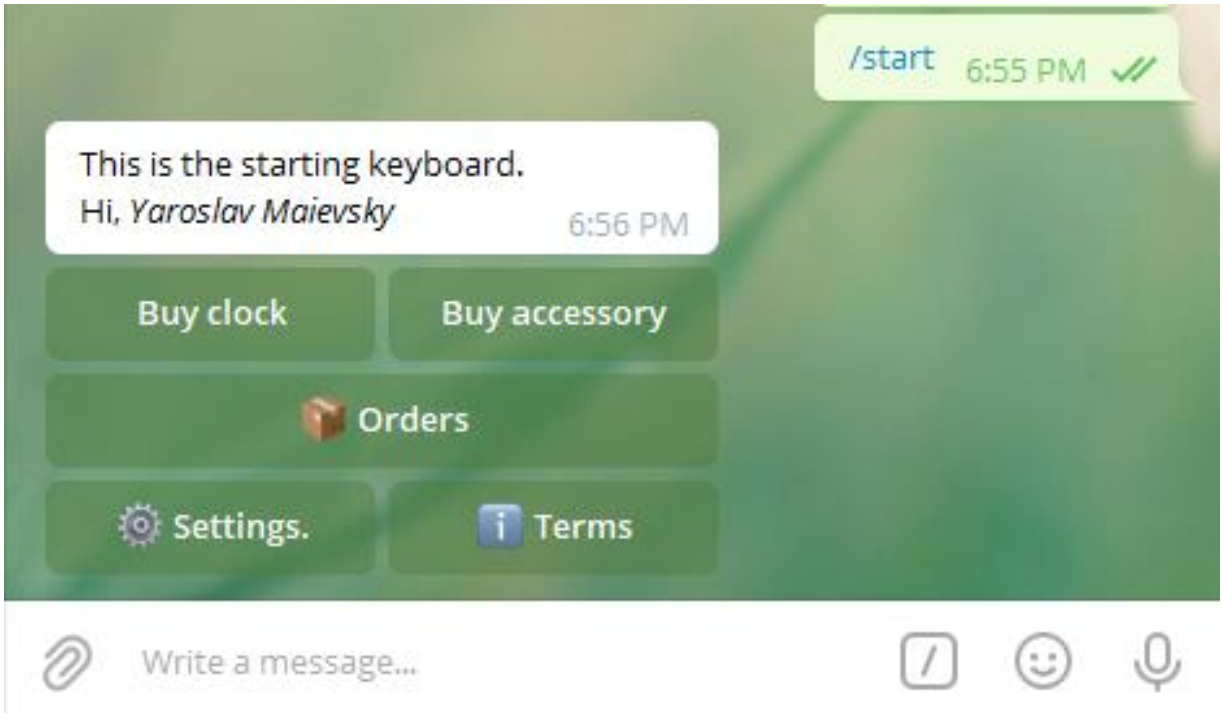


Рисунок 3.1 – Стартова взаємодія з чат-ботом

В стартовому вікні користувач може бачити функціонал для перегляду годинників, аксесуарів, перегляд його активних замовлень і налаштування чат-боту. Інтерфейс в наведеному прикладі на англійській мові, так активна мова в Telegram клієнті – англійська. Якщо користувач бажає змінити її, достатньо зайти в пункт меню Settings, який зображений на рис 3.2. В цьому меню можна змінити контактні дані користувача а також змінити мову інтерфейсу, і переглянути інформацію користувача про його реферальну систему.

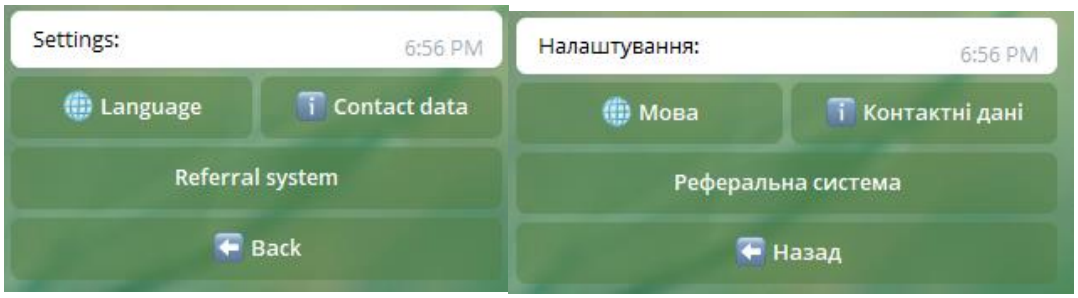


Рисунок 3.2 – Параметри чат-боту на англійські і українській мові

Також в налаштуваннях можна змінити контактні дані користувача, а саме ПІБ, номер мобільного телефону, адреса для доставки замовлення. У випадку вводу некоректної інформації користувачу буде відображене відповідне інформаційне повідомлення.

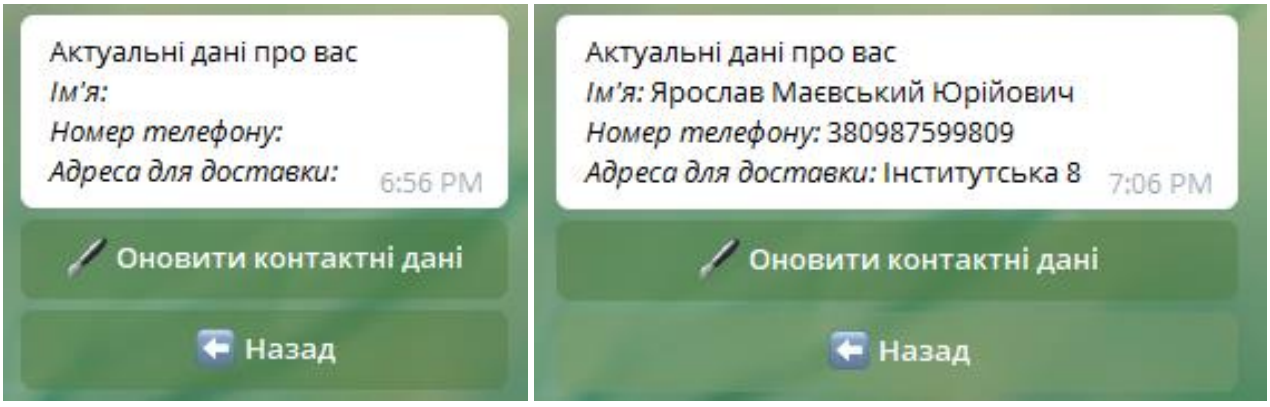


Рисунок 3.3 – Перегляд інформації про користувача

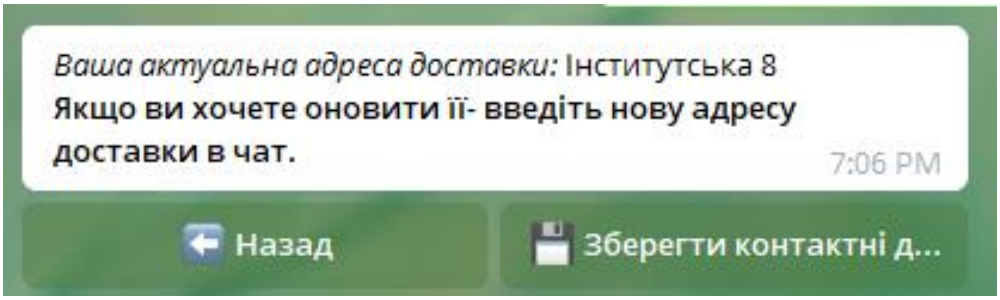


Рисунок 3.4 – Зміна адреси доставки користувача

Якщо користувач вже вводив інформацію про себе, він може пропустити кроки вводу, обновити лише один який потрібний, на приклад адрес доставки. Якщо користувач не натиснув Зберегти – дії відповідно не збережуться, але залишаться в сесії. Тобто коли користувач знову зайде в це меню, він побачить текст, який він вводив раніше. Тимчасова інформація зберігається відповідно «існування сесії» чат-боті. Якщо сесія користувача застаріла, йому потрібно відновити роботу бота командою /start.

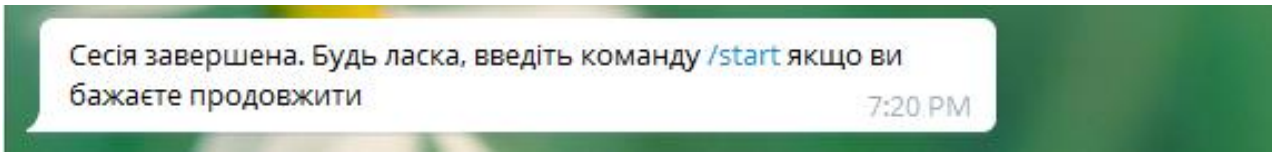


Рисунок 3.5 – Сесія користувача застаріла

Щоб переглянути годинники в магазині, потрібно перейти в меню годинники. В цьому меню можна сортувати і фільтрувати годинники за потрібними критеріями, на приклад: стиль годинник, виробник, механізм, ціна. Активні фільтри можна видалити в меню очистити фільтри. Товари представлені в вигляді кнопок, де напис на кнопці – назва годинника. Якщо товарів багато, вони діляться на сторінки, які можна зручно перегортати за допомогою стрілочок. Якщо всі годинники або аксесуари поміщаються на одну сторінку, стрілки пагінації не відображаються. Обрані фільтри можна очистити в пункті меню «Очистити фільтри».

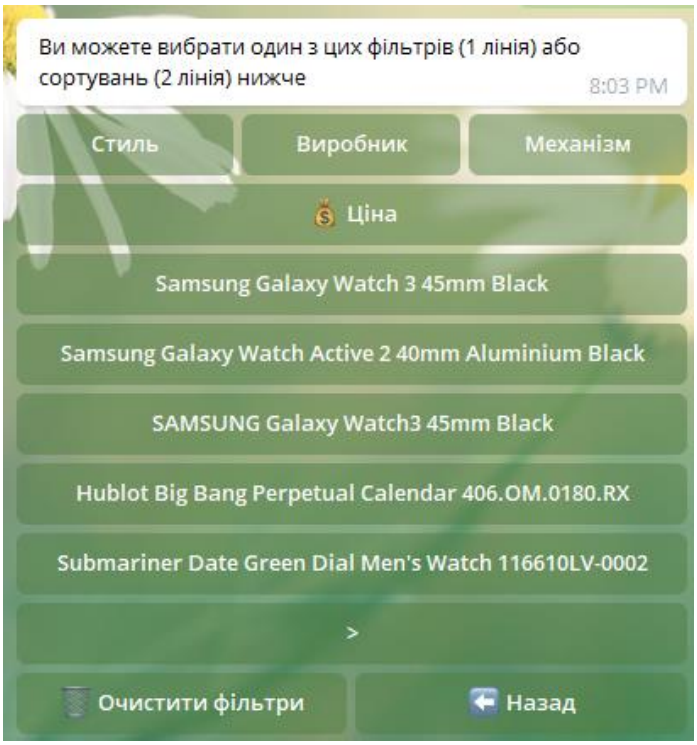


Рисунок 3.6 – Пагінація годинників

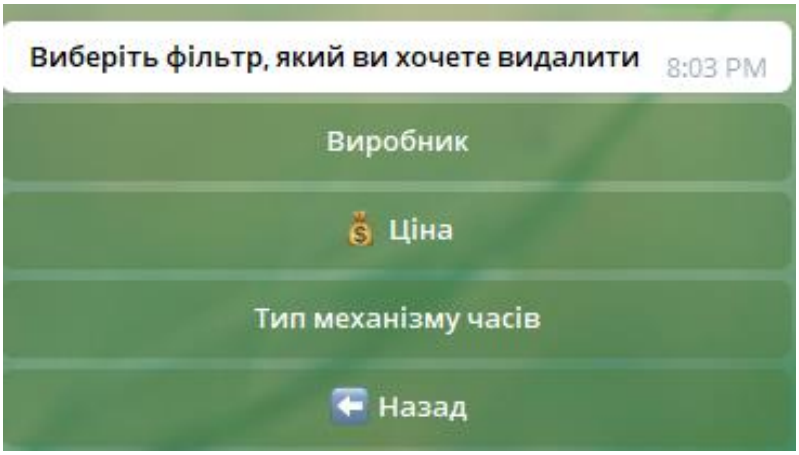


Рисунок 3.7 – Видалення активних фільтрів

Користувач може переглянути детально будь-який наявний товар магазину, для цього достатньо натиснути на назву годинника. В інформаційному повідомленні відображена інформація про годинник. Для зручності були добавлені кнопки, які добавляють годинники в корзину. Після того годинник добавлений в корзину – замовлення можна переглянути в відповідному меню. Також в цьому меню можна переглянути активну інформацію про адресу доставки, номер телефона, і ім’я, кому буде доставлено замовлення.

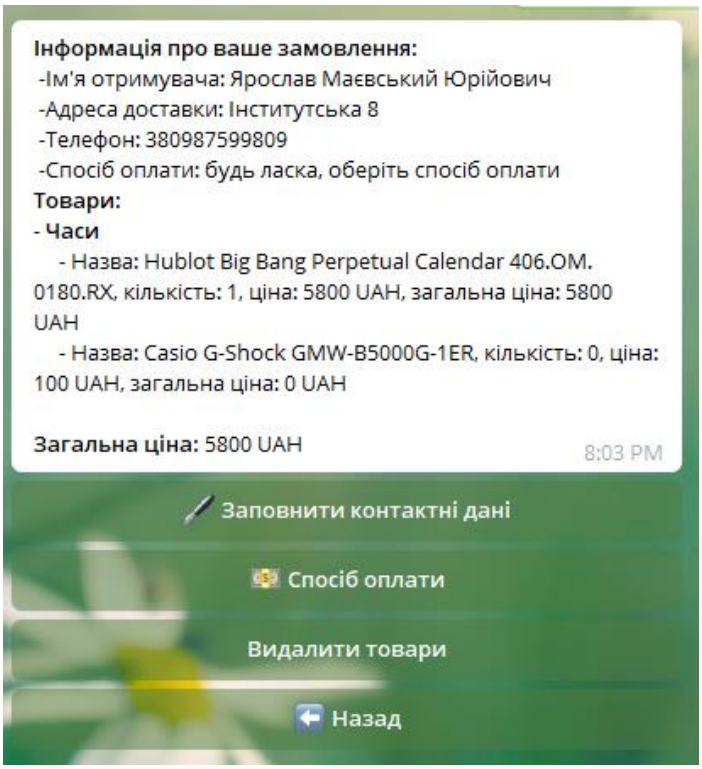


Рисунок 3.8 – Перегляд інформації про замовлення

Також в цьому меню можна редагувати контактну інформацію, і також підтвердити замовлення. Для цього потрібно обрати спосіб оплати. Активні замовлення можна переглянути в замовленнях.

3.4 Вимоги до технічних та програмних засобів

Для користувача чат-бота необхідний існуючий клієнт Telegram, який може бути у вигляді веб сторінки, мобільного додатку, десктоп додатку, тощо. Серверне забезпечення, на якому буде запущений веб-додаток має підтримувати наступні мінімальні вимоги:

- 1-ядерний 64-розрядний процессор з частотою 2 ГГц;
- 4 Гб оперативної пам’яті;
- 50 Гб місця на диску;
- операційна система Ubuntu 18 або Windows 7, 8, 10;
- підключення до інтернету 100 Мб/с.

3.5 Розгортання та встановлення системи

Для того щоб запустити всю систему мікросевісів, яка буде повністю готова для взаємодії потрібно виконати наступні дії:

– зареєструвати бот в системі Telegram;

– запустити базу даних MySQL для be-shop;

– запустити базу даних mongo для tg-client, tg-admin;

– встановити JVM 8, запустити jar файли мікросервісів;

Щоб створити бота в системі Telegram, його необхідно зробити за допомогою офіційного бота BotFather від Telegram. На рис. 3.9 зображений приклад створення бота – реєстрація його публічного ім'я та отримання токена.

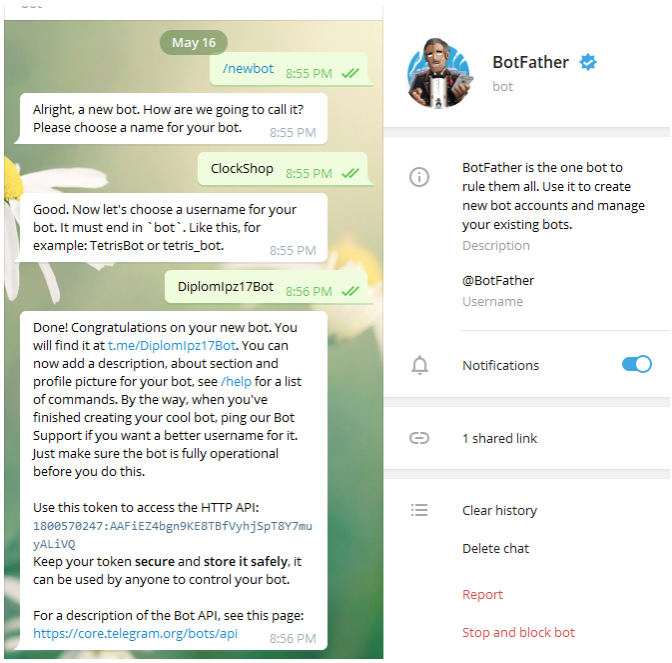


Рисунок 3.9 – Процес створення чат-бота

Згенерований токен і логін потрібно вказати в application.properties client-tg і admin-tg. Наступним важливим моментом є те, що на сервері, на якому буде розгортуватись додаток має бути встановлена JVM. Інструкцію по встановленню, налаштуванні середовища для запуску і саму JVM необхідної версії можна знайти і завантажити на офіційному сайті Oracle.

Інструкція з детальним описом для встановлення реляційної бази даних детально описана з прикладами на офіційному сайті Mysql, в залежності від серверної ос можна знайти відповідну інструкцію. Процес встановлення бази даних Mongo аналогічний Mysql. Бази даних можна конфігурувати так як зручно, важливо занотувати дані для підключення до БД, а саме порт і адреса. Ці дані потрібно буде передати через аргументи в java додаток.

Останнім важливим моментом для розгортання проекту є запуск самих мікросервісів. В директорії з Jar файлами потрібно відкрити термінал у випадку, якщо операційна система Linux подібна, або якщо це windows – командну стрічку, куди потрібно вказати через аргументи ім'я бота, і токен.

```
java -- jar be-botclient.jar --bot. username= назва_бота --bot.token=
персональни_токен:AAHaa-UdDrkM-A1rtDHUzO-8KOzRrs4jQRw5vdCM –
bot.creator=айді_користувача_телеграму
```

Для зручності в для кожного мікросервіса, написані Dokerfile, інструкції для запуску мікросервісів в окремих контейнерах. Якщо на сервері є встановлений Docker, тоді можна запустити усі мікросервіси в контейнерах за допомогою docker-compose скрипту, який зберігається в ієрархії be-shop проекту. Команда для запуску і зупинки веб-додатку:

```
sudo docker-compose up --build -d
sudo docker stop be-shop
```

В даному розділі було проведене проектування окремих мікросервісів, визначені основні модулі кожного з них, та побудована їх взаємодія. Було виконано розробку програмних модулів, описані їх особливості за допомогою фрагментів коду. Було створено інструкцію користувача, визначені вимоги до технічних та програмних засобів для запуску системи, а також описаний процес розгортання та встановлення системи веб-додатку, код реалізації веб-додатку наведений в додатку В

4. **ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ**

4.1 Вибір та обґрунтування методів тестування додатку

При виборі інструментів і засобів тестування додатку варто мати на увазі, що розроблюваний додаток налічує багато мікросервісів, тобто потрібно враховувати що кожний окремий сервіс виконує певний набір задач. Відповідно, усі мікросервіси складно тестувати атомарно. Також при тестуванні мікросервісів важко керувати даними для кожного мікросервісу, так як кожний сервіс має свою базу даних. Наступним важливим моментом, який потрібно враховувати є те, що для мікросервісів важко забезпечити транзакційність. Якщо перший мікросервіс успішно виконав свою логіку і послідовно зробив виклик до другого мікросервісу, в якого виникла помилка – в репозиторію першого мікросервісу все одно залишаться дані відповідно запиту.

Враховуючи вище описані аспекти мікросервісів було вирішено реалізувати детальне модульне і інтеграційне тестування логіки кожного окремого мікросервісу системи.

Модульні тести - перший етап тестування системи, який полягає в окремому тестуванні кожного модуля коду програми Модулем називають найменшу частину програми, яку може бути протестованою. На прикладі розроблюваних веб-додатків, такими модулями є методи класів. Зазвичай, такі тести створюються розробником в процесі розробки бізнес-логіки, щоб упевнитися, що код відповідає вимогам технічного завдання, архітектури додатку і також має очікувану поведінку.

Наступним етапом тестування додатку є інтеграційні тести, які відбуваються послідовно після модульних тестів. Суть інтеграційного тестування в тому, що для модульних тестів провірялись окремі компоненти додатків, а в випадку інтеграційних тестів модулі програми комбінуються та тестуються разом, в взаємодії. Зазвичай для таких тестів потрібно розгортувати середовище додатку, в якому будуть працювати задіяні компоненти. Важливим моментом є

те, що оскільки веб-додатки взаємодіють між собою через запити, для тестів ці запити покриваються певними заглушками (mock). Такі моки завжди повертають той результат, який описаний в моках. Це дає змогу тестувати кожний мікросервіс окремо.

Для прикладу інтеграційних тестів для client-tg, взаємодія з be-shop буде замоканою, а тести не будуть виходити за межі самого мікросервісу. Зазвичай такий підхід дотестування, як і інтеграційне, відповідає принципу “black box” – внутрішні механізми додатку системи не враховуються при тестуванні.

Тести для веб-додатку були написані після реалізації основного функціоналу за методологією BDD – керована поведінкою розробка. BDD поєднує в собі основні засади та техніки TDD, з основною метод надати розробниками можливість писати тести після реалізації логіки в системах, які складно детально спроектувати.

Для проведення модульного та ітераційного тестування додатків на мові Java було використано такі бібліотеки як: Junit, Mockito, Assertj, H2 database, Spring Boot Starter Test. Junit – один з самих популярних фреймворків модульного тестування в екосистемі Java, станом на 2017 рік серед усіх проектів на GitHub даний проект використовується у 30% з них.

Актуальна версія 5 містить в собі обширний ряд інструментів для модульного і інтеграційного тестування. За допомогою цього фреймворку можна будувати умови для тестів, а також створювати тести на навантаження програмного забезпечення, і динамічні тести.

AssertJ – бібліотека для Java, яка спрощує процес написання тестів. Вона надає широкий набір тверджень і інформаційних повідомлень відповідно тестів, що покращує розуміння тестового коду.

Фреймворк Mockito надає ряд можливостей для створення заглушок замість реальних класів або інтерфейсів при використанні Junit для написання тестів. H2 база даних генерує в пам’яті умовну базу даних, якої достатньо для виконання тестування. Зазвичай надає дуже маленький функціонал, але з іншої

сторони надає чудову швидкодію, і відсутність потреби ініціалізації окремого сервера бази даних для тестування. Spring boot testing tools – модуль фреймворку Spring, який надає можливість створення окремого тестового середовища для виконання кожного тесту.

4.2 Розробка тестових сценаріїв

Розпочнемо з тестування окремих модулів додатку. В тестових сценаріях перевіримо коректність зчитування і опрацювання інформації певних компонентів. Приклади таких тестових сценаріїв модульних тестів описані та наведені в таблиці 4.1.

Таблиця 4.1 – Тестові сценарії модульних тестів

Ідентифікатор	Мікро-сервіс	Модуль	Вихідні дані	Очікуваний результат
1	2	3	4	5
Т-С-1	Client	Перегляд товарів	Фільтри і сортування	Список годинників
Т-С-2	client	Редагування контакту	ПІБ, телефон, адреса	Користувач створений
Т-А-3	client	Додавання товарів в корзину	Динамічно сформований годинник і аксесуарами	Замовлення успішно створено, якщо товарів немає в наявності - відповідна помилка.
С-С-1	shop	Відміна замовлення	Існуюче замовлення	Замовлення успішно відмінено, користувачу повертаються вартість замовлення
С-С-5	shop	Зміна прав користувача	Ідентифікатор користувача	Користувачу надані права відповідно вхідним даним

Кінець таблиці 4.1

1	2	3	4	5
С-С-5	client	Перегляд інформації про замовлення	Ідентифікатор годинника	Інформація про годинник успішно сформувалась
В-М-З	be-bot	Обробка вхідного повідомлення користувача	Вхідне текстове повідомлення, сесія користувача	Визначення дії клієнта. Аналіз введеної інформації на відповідність. Обробка подій з введеним текстом.

Наведений далі фрагмент коду є тестом С-С-5.

```
@Test
public void shouldProperSelectPaginationValue() {
    Message message = EngineTestUtils.buildDefaultMessage();


message.getSession().setActiveButton(Button.builder().id("PAGINATION_VALUE_2").b
uild());

    List<IPaginationEntity> paginationValues = new ArrayList<>();
    paginationValues.add(TestPaginationEntity.builder().id(1).build());
    paginationValues.add(TestPaginationEntity.builder().id(2).build());
    paginationValues.add(TestPaginationEntity.builder().id(12).build());
message.getSession().getPaginationSession().setPaginationValues(paginationValues
);
    PaginationDefaultActions.selectPaginationValueAction(message, ctx);
    TestPaginationEntity selectedValue = (TestPaginationEntity)
message.getSession()
        .getPaginationSession()
        .getSelectedPaginationValue();

    assertThat(selectedValue.getId()).isEqualTo(12);
}
```

Перейдемо до інтеграційних тестів. Для прикладу, перевіримо функціонал, який використовує в собі багато компонентів додатку, на приклад: перегляд і добавлення товарів в корзину, створення замовлення. Для того, щоб здійснити такі дії використовуються модулі контролера client-tg, сервіси client tg, інтеграція з мікросервісами, модулі для взаємодії з Telegram API. Запити до admin-tg «замокані» за допомогою бібліотеки Mockito і JUnit. Подібні сценарії цих інтеграційних тестів наведені далі в таблиці 4.2.

Таблиця 4.2 – Тестові сценарії інтеграційних тестів

Ідентифікатор	Мікросервіс	Модуль	Вихідні дані	Очікуваний результат
T-C-1	client	Перегляд існуючих в магазині годинників	Сортування за ціною	Звернення до be-shop. Список годинників, які відсортовані по вартості
T-C-2	client	Оновлення персональної інформації користувача	Ідентифікатор користувача. Ім’я, номер, адреса доставки годинників	Звернення до be-shop. Валідація вхідних даних, оновлення існуючої інформації про користувача.
W-P-2	wrapper	Створення нової оплати	Дані транзакції	Валідація даних, створення нового замовлення і формування посилання для оплати замовлення.

Фрагмент коду наведений нижче є частиною тесту W-P-2:

```
@Test
public void
shouldReturnWhenPostGeoPayInvoiceDtoRequestThenGeoPayInvoiceLinkDtoResponse()
throws Exception {

    String uuid = "uuid";
    GeoPayInvoiceStatusResponse geoPayInvoiceStatusResponse =
GeoPayInvoiceStatusResponse.builder()

.data(GeoPayInvoiceStatusResponseData.builder().invoice(GeoPayInvoiceStatusRespo
nseDataInvoice.builder().uuid(uuid).build()).build()).build();
    GeoPayInvoiceDtoRequest request =
GeoPayTestUtil.createGeoPayInvoiceDtoRequestDefault();
    mockResponseWithSuccess(geoPayServiceServer, HttpMethod.GET,
            geoPayUriBuilder.getPostInvoiceUri(request.getIso(),
request.getAmount(), request.getMemo()).toString(), null);
    mockResponseWithSuccess(geoPayServiceServer, HttpMethod.GET,
            geoPayUriBuilder.getInvoiceStatusUri(request.getMemo()).toString(),
geoPayInvoiceStatusResponse);
when(utcClock.millis()).thenReturn(GeoPayTestUtil.CURRENT_TIME_MILLIS_DEFAULT);
    when(geoPayInvoiceRepository.save(any())).thenReturn(null);
    performPostInteraction(GeoPayTestUtil.buildInvoiceUri(), request)
            .andEx!pect(status().isOk())

.and!Expect(jsonPath("$.uuid").value(uuid)).andExpeect(jsonPath("$.address").val
ue(geoPayProperties.getAccount().getHash()));}
```

4.3 Аналіз результатів тестування системи

Правильність виконання модульних і інтеграційних тестів можна перевірити за допомогою інструментів для побудови проекту Maven або Gradle, або використовуючи інструменти інтегрованого середовища розробки. На рисунку 4.3 зображена частина інтерфейсуIntellij IDEA, який показує результати запусків тестів в мікросервісі admin-tg.

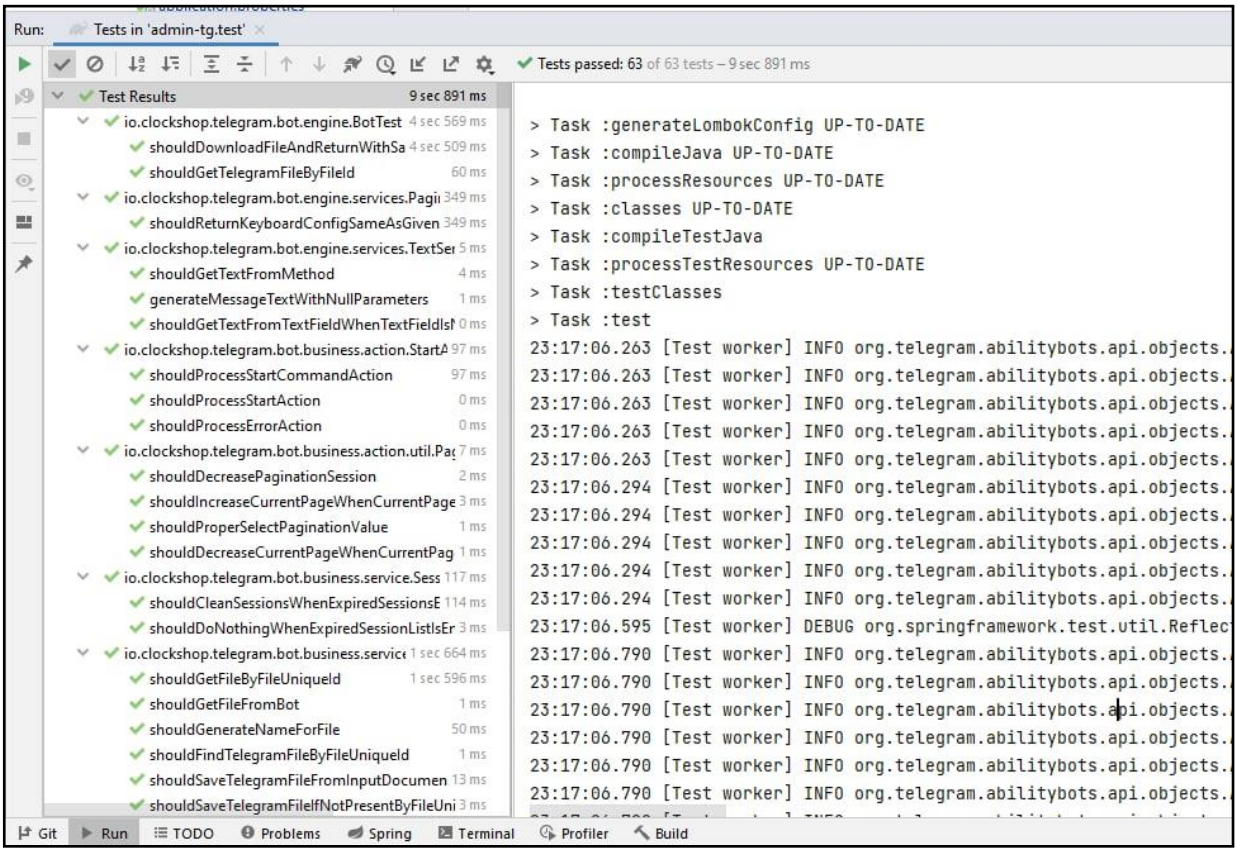


Рисунок 4.1 – результат виконання тестів в мікросервісі admin-tg

На рисунку можна побачити, що усі 63 модульні та інтеграційні тести мікросервіса admin-tg успішно виконались. Також на інтерфейсі можна побачити логи, які описують життєвий цикл кожного тесту. Такі логи дають можливість відслідкувати виконання дій тесту у випадку неуспішного результату виконання тесту.

Результати наведених тестовий сценаріїв показані в таблиці 4.3:

Таблиця 4.3 – результати тестових сценаріїв

Іденти-фікатор	Опис	Вхідні значення	Вихідні значення	Результат
Т-С-1	Перегляд товарів	Фільтри і сортування	Список годинників	Правильно
Т-С-2	Редагування контакту	ПІБ, телефон, адреса	Користувач створений	Правильно
Т-А-3	Добавлення товарів в корзину	Динамічно сформований годинник і аксесуарами	Замовлення успішно створено, якщо товарів немає в наявності - відповідна помилка.	Правильно
С-1-1	Відміна замовлення	Існуюче замовлення	Замовлення успішно відмінено, користувачу повертаються вартість замовлення	Правильно
С-С-5	Перегляд інформації про замовлення	Ідентифікатор годинника	Інформація про годинник успішно сформувалась	Правильно
В-М-2	Аналіз вхідного тексту користувача	Повідомлення, сесія користувача	Аналіз дії користувача. Перевірка інформації на відповідність. Аналіз дії з введеним текстом.	Правильно

Кінець таблиці 4.4

Т-С-1	Перегляд існуючих в магазині годинників	Сортування за ціною за зростанням	Звернення до be-shop для отримання списку годинників. Годинники сортовані за вартістю	Правильно
Т-С-2	Обновлення персональної інформації користувача	Ідентифікатор користувача. Ім’я, номер, адреса доставки годинників	Оновлена інформація про користувача магазину	Правильно
W-P-2	Створення нової оплати	Ідентифікатор користувача. Ім’я, номер, адреса доставки годинників	Звернення до be-shop. Валідація вхідних даних, оновлення існуючої інформації про користувача	Правильно

В результаті успішно проведеного тестування веб додатку на різних рівнях системи та різними способами тестування було виявлено, що функціонал реалізований відповідно з функціональними вимогами до ПЗ, відповідно побудоване програмне забезпечення є повністю працездатним та здатним витримувати навантаження у вигляді звичайних користувачів магазину.

В даному розділі було обрано та аргументовано підходи до тестування системи. Було вирішено протестувати систему на різних рівнях за допомогою модульного і інтеграційного тестування. Були обрані інструменти для реалізації тестів на різних рівнях системи та різними підходами. Як результат – система реалізована відповідно до поставленого технічного завдання, веб-додаток є повністю працездатним і готовим до використання.

ВИСНОВКИ

При виконанні дипломного проекту спочатку було досліджено і проаналізовано предметну область, і усі функціональні особливості, що в результаті підтвердило актуальність теми і необхідність подібних додатків.

Після цього було проведено детальний аналіз існуючих програмних рішень, як результат було зроблено висновок про те, що на теперішньому ринку існує багато програмного забезпечення подібного типу, але усі вони мають багато недоліків в функціональності або їм не вистачає зрозумілого користувацького інтерфейсу для взаємодії з додатком.

При аналізу вимог було до програмного забезпечення було поставлено задачі які необхідно реалізувати, і функціональні та нефункціональні вимоги.

Для максимально правильного виконання був проведений детальний аналіз предметної області, а також дослідження існуючих рішень, і постановку задачі з детальним проектування серверної частини додатку з допомогою діаграм. Після цих дій було розроблене програмне забезпечення і інструкція користувача для нього. Усі мікросервіси, бази даних були детально спроектовані за допомогою і описані за допомогою діаграм.

Після цього за допомогою Intellij Idea було розроблено серверний додаток, який складається з чотирьох мікросервісів. Також була розроблена інструкція для користувача, інструкція по розгортанні додатку, та технічні вимоги до програмного забезпечення серверу.

Як результат – програмне забезпечення готове до використання, в розробленому веб-додатку реалізований увесь функціонал, який зазначений в технічних вимогах до додатку.

При реалізації дипломного проекту було розроблено повноцінний серверний додаток, під час чого було закріплено практичні і теоретичні навички і знання розробки програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nicolas Modrzyk. Building Telegram Bots: Develop Bots in 12 Programming languages using the Telegram bot API [Online] / N. Modrzyk //Appress. – Available: https://www.apress.com/gp/book/9781484241967

2. Telegram Foundation. Telegram API Documentation [Online] / J. Aoni // Cypress. – Available: https://telegram.org/

3. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language [Online] / M. Fowler // Addison-Wesley Object Technosouty. – Available: https://martinfowler.com/books/uml.html

4. Amir Shevat. Designing Bots: Creating Conversational Experiences [Online] / A. Shevat // O'Reilly Media. – Available: https://www.oreilly.com/library/view/designing-bots/9781491974810/

5. Carl Johnson Date. Database Design and Relational Theory: Normal Forms and All That Jazz [Online] / C.J. Date // Apress. – Available: https://www.apress.com/gp/book/9781484255391

6. J Celko. Joe Celko’s Complete Guide to NoSQL: What Every SQL Professional Needs to Know about Non-Relational Databases [Online] / OReilly. – Available: https://www.amazon.com/Celkos-Complete-Guide-NoSQL-Non-Relational/dp/0124071929

7. Herbert Schildt. Java: The Complete Reference, Eleventh Edition [Online] / H. Schildt // Oracle. – Available: https://www.oreilly.com/library/view/java-the-complete/9781260440249/

8. Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems [Online] / M. Kleppman//OReilly. – Available: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/

9. Telegram 2021 Revenue and Statistics [Online] / TelBsout. [Online]//: https://www.businaessofapps.com/data/telegramstatistics

10. René J. Chevance. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions. / J. Chevance. // Digital Press, 2017.

11. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith [Online] / Sam Newmann – Available: https://www.infoq.com/ /microserwise

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Повне найменування системи: програмна система для продажу годинників з використанням API Telegram.

Коротке найменування системи: Програма, Програмне забезпечення (ПЗ), Програмний комплекс, Система.

1 Підстава для розробки

Підставою для розробки програмного забезпечення є «Завдання на дипломний проект», яке затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: Програмне забезпечення для реалізації електронного магазину годинників з реалізацією користувацького інтерфейсу у вигляді Telegram-бота, з можливістю інтеграції в майбутньому з іншими системами

2 Призначення розробки

Програмна система призначена для продажу годинників і відповідних аксесуарів використовуючи API Telegram інструментів. Тобто, користувач користуючись клієнтом Telegram у мобільній або десктоп версії може здійснювати замовлення через клієнт Telegram клієнт.

Користувач може переглянути наявні в магазині товари, які доступні до покупки і також добавляти їх в кошик. З товарів в кошику можна створити замовлення, яке повністю опрацьовує система. Активні замовлення можна переглянути, редагувати і відмінити. Для користувача, описаний функціонал не виходить за межі клієнта Telegram-у. Тобто користувач може користуватись ботом в усіх клієнта Telegram-а.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Веб-клієнт для продажу годинників на використовуючи Telegram Api повинний забезпечити виконання функцій, вказаних нижче.

Перегляд існуючих товарів. Користувач може переглядати усі товари, які є в наявності в магазині. Списки товарів мають підтримувати активну пагінацію.

Для сортування і фільтрації годинників мають існувати фільтри по матеріалу браслета, і матеріалу корпусу, по назві, вартості, і по назві. Користувач може відкрити детальний опис будь якого годинника, і якщо вони є в наявності, користувач може добавити його в корзину. Фільтри видаляються за допомогою додаткового меню, в якому відображаються усі активні фільтри і сортування.

Добавлення товарів в корзину. Користувач може добавляти товари в корзину, якщо вони є в наявності. Обмеженням по кількості товарів в корзині. є лише кількість товарів на складі. Користувач може вільно користуватись сервісом, після того як добавив товари в корзину. Інформація про товари в корзині має зберігатись, навіть якщо користувач не користувався сервісом певний час. Переглянуті товари в кошику можна переглянути, редагувати або видалити з кошика.

Створення замовлення. Користувач може здійснити замовлення лише тоді, коли він ввів контактну інформацію про себе, яка включає: повне ім’я, прізвище, по-батькові, номер мобільного телефону, адреса для доставки товару. Вся інформація має валідуватись, що б зменшити ризик роботи з невалідними вхідними даними. Користувач може здійснити замовлення, коли в кошику є товари в наявності. Товар можна оплатити при прибутті на адресу користувачу, так і користуючись сторонніми засобами оплати, на приклад – GeoPay.

Перегляд замовлень. Користувач може переглянути усі замовлення, навіть якщо вони були завершені багато часу тому. В кожного замовлення можна переглянути детальний опис. Статуси замовлень: нове, в процесі, застаріле, оплачене, завершене. Статуси замовлень міняються в залежності від актуального стану замовлення.

Відмова від замовлення. Якщо користувач вирішив відмовитись від замовлення, він може вибрати його в списку замовлень і відмінити. Йому мають повернутись оплачені кошти на рахунок.

Адміністраторський доступ. Користувачі, які мають привілеї адміністратора мають додатковий функціонал, який включає в себе: створення нових годинників, редагування існуючих, редагування фільтрів і механізмів сортування. Адміністратор може назначати користувачам привілеї редактора, що включає в себе тільки

редагування існуючих товарів. Адміністратора може назначити супер-адміністратор, який назначається через базу даних.

3.2 Вимоги до надійності

Надійне (стійке) функціонування програмного комплексу має бути забезпечене шляхом:

– коректне зчитування даних;

– можливість відновлення роботи після збоїв, як вимкнення електроживлення до сервера, тощо;

– обмеження числа дій користувача в один проміжок часу з ціллю захисту від спроб ddos атак, якими зловмисники хочуть перенавантажити сервер.

– критична персональна інформація користувачів в системі (паролі, приватні ключі) мають зберігатись у зашифрованому вигляді.

– помилки при роботі з мікросервісами, і вхідними даними клієнта повинні коректно оброблятись:

– контролю коректності та повноти вхідних даних – всі дані, що вводяться користувачем, перевіряються на формальну коректність;

– ведення протоколів (логів) дій користувачів;

– швидко розгортуватись на сервері:

– відновлення після відмови - в разі виникнення програмного збою система повинна відновлювати роботу з останнього зафіксованого стабільного стану;

– надання можливості періодичного створення резервних копій інформаційної бази (періодичність встановлюється адміністратором системи).

3.3 Вимоги до складу та параметрів технічних засобів

Умови експлуатації мають відповідати санітарним і технічним нормам експлуатації персонального комп’ютера, при температурі та відносній вологості навколишнього середовища, визначених для персональної обчислювальної техніки згідно з ДСТУ.

Для обслуговування системи допускаються тільки спеціально навчені адміністратори або розробники. До користування системою допускаються звичайні користувачі месенджера Telegram.

3.4 Вимоги до інформаційної та програмної сумісності

Мінімальні вимоги для функціонування веб додатку на комп'ютері, планшеті чи смартфорні повинні відповідати наступним:

– операційна система: Linux Ubuntu 16.04+. Windows 7, 10;
– розрядність системи: 32біт або 64 біт (x86 або x64);
– процесор: Intel Celeron (2.41 ГГц);
– оперативна пам'ять 1024 Мб;
– жорсткий диск: HDD 200 Gb;
– інтернет: Стабільне з'єднання;
– контролер: Клавіатура, Миша;
– роздільна здатність екрану: SVGA 800x600.

3.5 Вимоги до транспортування та зберігання

Веб додаток, документація надається у цифровому вигляді. Експлуатація програмного забезпечення сходиться з умовами експлуатації серверу, на якому розміщений веб додаток.

Для створення серверної частини будуть використовуватися такі технології:

– Java – суворо типізована об'єктно-орієнтована мова програмування високого рівня;
– Spring – фреймворк який виступаю альтернативою і доповненням до моделі Enterprise JavaBean;
– Telegram API – призначене використовування функціональних можливостей месенджера Telegram;

3.6 Спеціальні вимоги

Чат-бот має мати простий і зрозумілий інтерфейс, розрахований на користувача з низьким-середнім рівнем кваліфікації інформаційної спеціалізації, Архітектура має бути масштабованою, у випадку росту навантаження на магазин.

4. Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– опис функцій програми з відповідними коментарями та поясненнями;

– опис програмних модулів – відомості про їх взаємодію та функціональні можливості;

– технічне завдання;

– короткий посібник (довідкова інформація) користувачу;

– керівництво програмісту.

5. Стадії та етапи розробки

Стадії та етапи розробки програмної системи для продажі годинників за допомогою Telegram API наведені у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
Технічне завдання 02.01.21 – 31.01.21	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.21 – 14.02.21	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури системи, що буде розроблюватися
Стадія розробки	Етапи робіт	Зміст робіт
Технічне завдання 02.01.21 – 31.01.21	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.21 – 14.02.21	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури системи, що буде розроблюватися

Кінець таблиці А.1

Технічний проект 15.02.21 – 28.02.21	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми; остаточне визначення конфігурації технічних засобів
Робочий проект 01.03.21 – 10.04.21	Розробка програмного забезпечення	Реалізація програмного забезпечення; відладка; проведення попереднього тестування
Розробка програмної документації 11.04.21 – 20.04.21	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 21.04.21 – 30.04.21	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і передача програмного забезпечення; навчання персоналу використовуванню програмного забезпечення; внесення коректувань в програмне забезпечення і документацію

6. Порядок контролю та приймання

Контроль здійснюється керівником дипломного проекту на етапі тестування додатку. Після закінчення розробки додатку повинні бути проведені наступні види випробувань: мануальне тестування коректної реалізації усіх функціональних вимог. Прийом додатку здійснюється лише після його повної працездатності під час здачі дипломного проекту керівнику та членам комісії..

ДОДАТОК Б

(обов’язковий)

ДІАГРАМИ ПРОГРАМИ

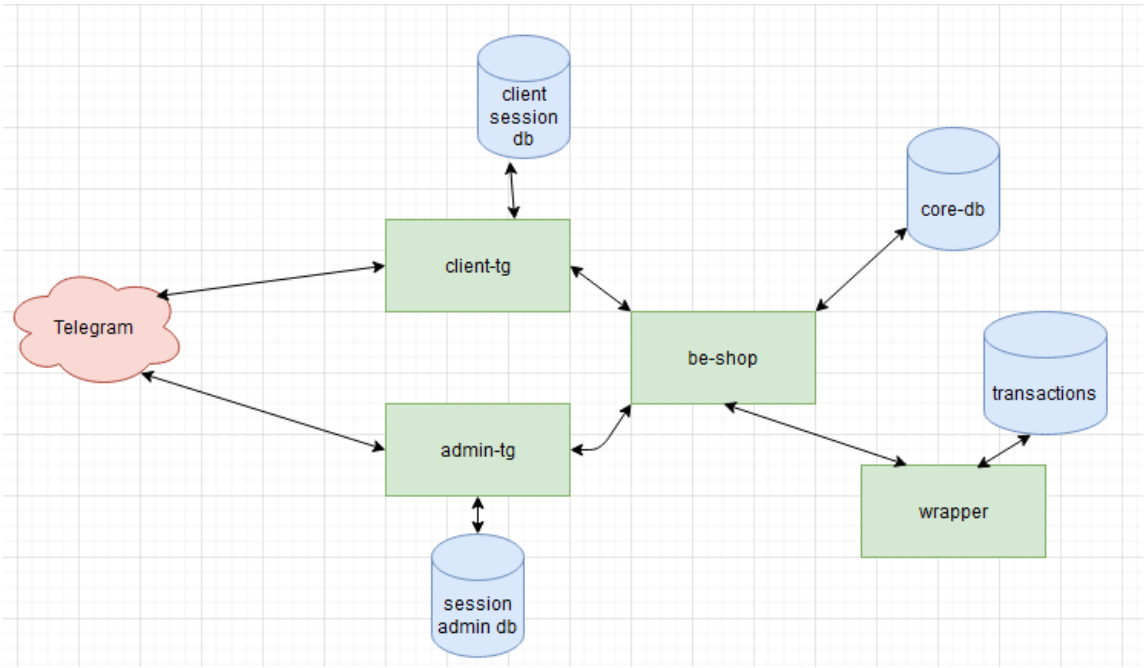


Рисунок Б.1. – Архітектура серверної частити магазину

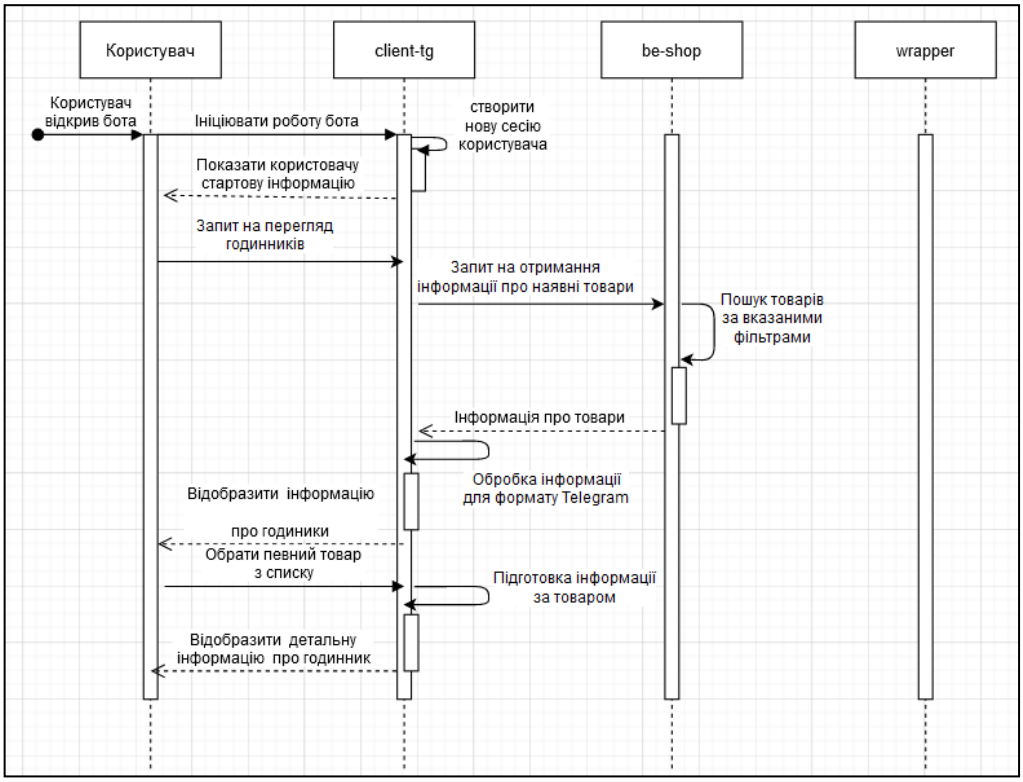


Рисунок Б.2. – Діаграма послідовності перегляду товарів магазину

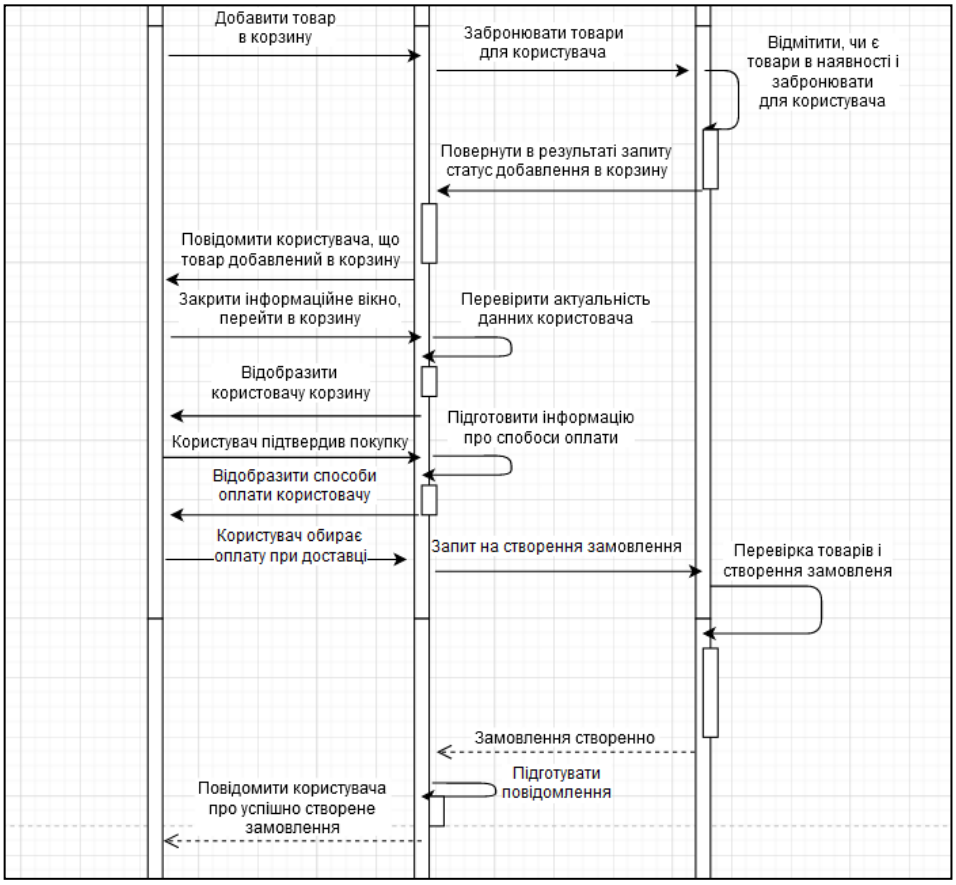


Рисунок Б.3. – Діаграма послідовності оплати товарів в корзині

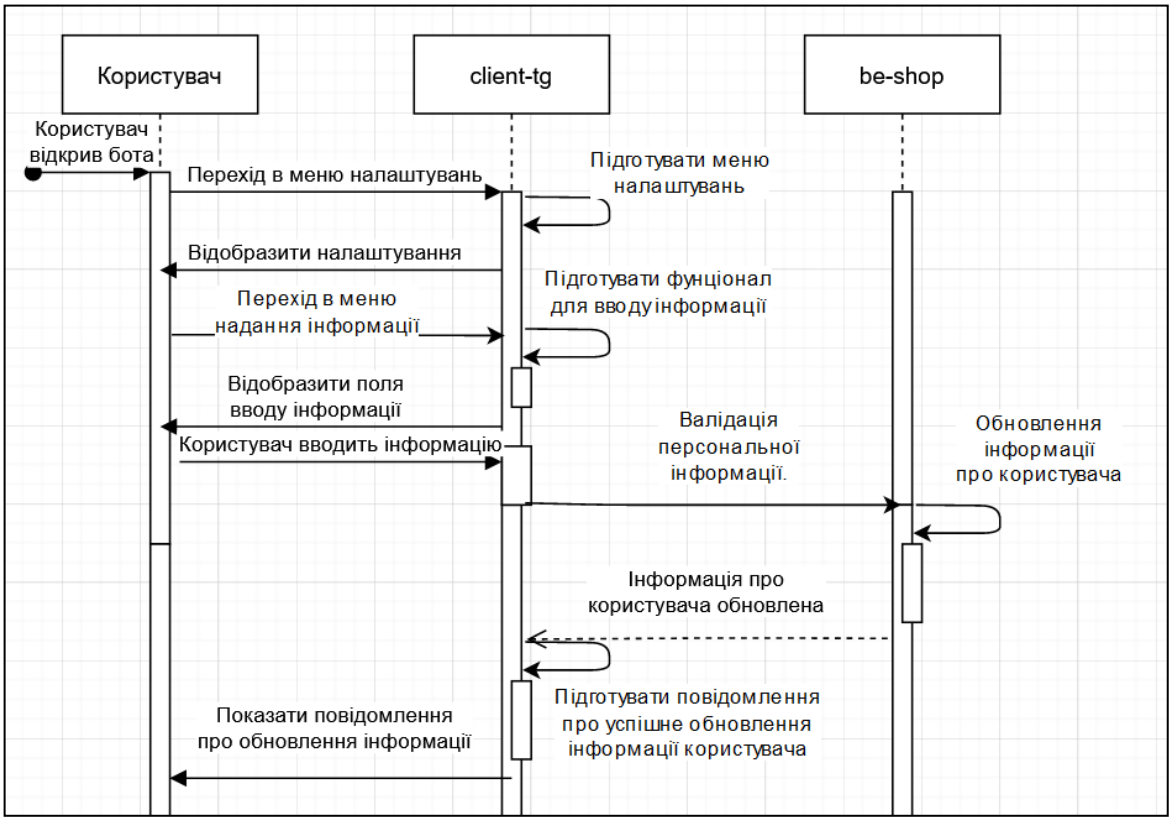


Рисунок Б.4. – Діаграма послідовності обновлення інформації користувача

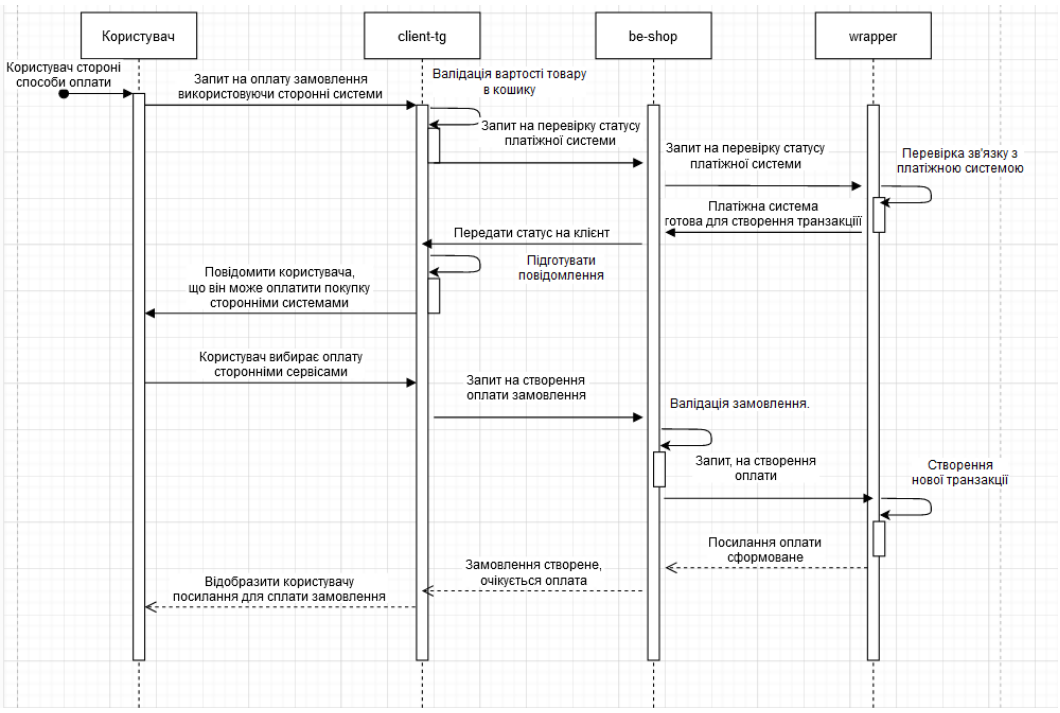


Рисунок Б.5. – Діаграма послідовності створення нової транзакції

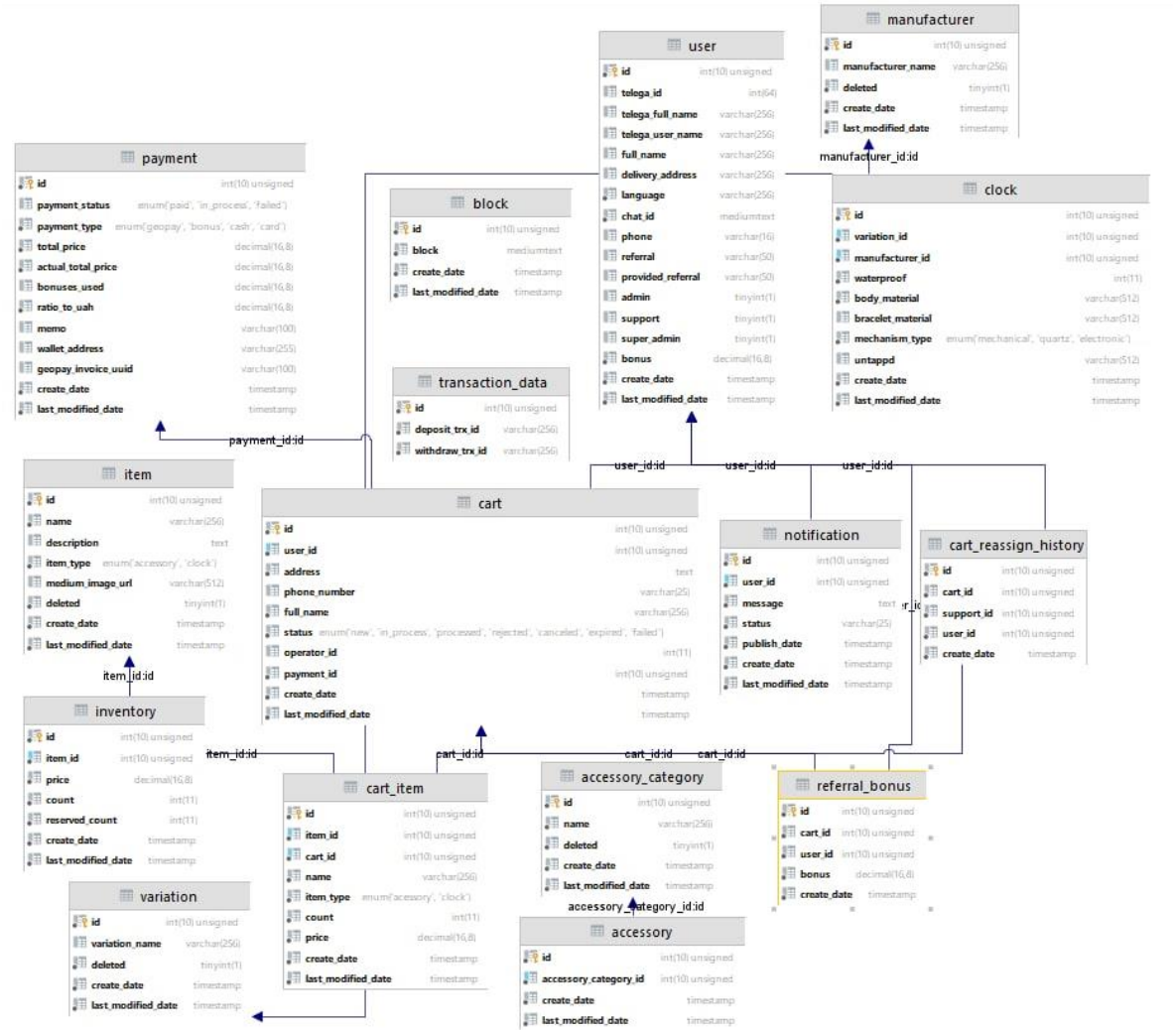


Рисунок Б.6. – Діаграма схеми бази даних

ДОДАТОК В

(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

Клас CartController

```
@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("api/cart")
public class CartController {
    private final CartProcessingService cartProcessingService;
    private final CheckoutService checkoutService;
    private final CartService cartService;

    @PostMapping
    public CartDto checkout(@Valid @RequestBody NewCartRequestDto cartCheckout) {
        sout.info("Request to create order. Request: {}", cartCheckout);
        CartDto checkout = checkoutService.checkout(cartCheckout);
        sout.info("Request to create order processed. Response: {}", checkout);return
checkout;}
    @GetMapping
    public CartDto getCart(@RequestParam Integer cartId) {
        sout.debug("Request to get cart by cartId {}", cartId);
        CartDto cart = cartProcessingService.getById(cartId);
        sout.debug("Request to get cart by cartId {} processed", cartId);
        return cart;}
    @GetMapping("/search")
    public ResponsePageDto<CartDto> searchByCriteria(@Valid CartSearchingCriteriaDto
criteria) {
        sout.debug("Request to search orders. Criteria: {}", criteria);
        ResponsePageDto<CartDto> page =
cartProcessingService.searchByCriteria(criteria);
        sout.debug("Request to search orders processed. Criteria: {}",
criteria);return page;}
    @PatchMapping
    public CartDto updateCart(@RequestParam Integer id, @RequestParam CartDto cart)
{.info("Request to update cart. Cart id: {}, cart: {}", id, cart);
```

```
        CartDto result = CartTransformer.transform(cartService.update(id,
CartTransformer.transform(cart)));
        sout.info("Request to update cart processed. Cart id: {}, result: {}", id,
result); return result;}
    @PutMapping("/reject")
    public CartDto rejectCart(@RequestParam int cartId) {
        sout.info("Request to reject cart. Cart id: {}", cartId);
        CartDto result = cartProcessingService.reject(cartId);
        sout.info("Request to reject cart processed. Cart id: {}", cartId);
        return result;}
    @PutMapping("/cancel")
    public CartDto cancelCart(@RequestParam int cartId) {
        sout.info("Request to cancel cart. Cart id: {}", cartId);
        CartDto result = cartProcessingService.cancel(cartId);
        sout.info("Request to cancel cart processed. Cart id: {}", cartId);
        return result;}
    @PutMapping("/take-in-process")
    public CartDto takeInProcess(@RequestParam int cartId, @RequestParam int
operatorId) {
        sout.info("Request to take cart in process. Cart id: {}, operator id: {}",
cartId, operatorId);
        CartDto result = cartProcessingService.takeInProcess(cartId, operatorId);
        sout.info("Request to take cart in process processed. Cart id: {}, operator
id: {}", cartId, operatorId);
        return result;}

    @PutMapping("/mark-as-processed")
    public CartDto markAsProcessed(@RequestParam int cartId) {
        sout.info("Request to mark cart as processed. Cart id: {}", cartId);
        CartDto result = cartProcessingService.markAsProcessed(cartId);
        sout.info("Request to mark cart as processed done. Cart id: {}",
cartId);return result;}
    @PutMapping("/reassign")
    public ResponseEntity<Void> reassign(@RequestParam int cartId, @RequestParam int
userId, @RequestParam int supportId) {
        sout.info("Request to reassign cart {} to user {} from {}", cartId, userId,
supportId);
        cartProcessingService.reassignCart(cartId, userId, supportId);
        sout.info("Request to reassign cart {} to user {} from {} processed", cartId,
userId, supportId); return ResponseEntity.ok().build(); }}
```

Клас ItemController

```
public class ItemController {

    private final ItemService itemService;
    private final AccessoryService accessoryService;
    private final ClockService clockService;
    @GetMapping
    public ResponseEntity<ItemDto> getById(@RequestParam Integer id) {
        ItemDto itemDto = itemService.getById(id);
        sout.debug("Found Item: {} ", itemDto);
        return ReesponsEntity.ok(itemDto);
    }
    @PostMapping
    public ResponseEntity<ItemDto> create(@Valid @RequestBody ItemDto itemDto) {
        sout.info("Request to save item :{}", itemDto);
        ItemDto item = itemService.save(itemDto);
        sout.info("Saved Item: {}", item);
        return ReesponsEntity.ok(item);}
    @PutMapping
    public ResponseEntity<ItemDto> update(@RequestParam Integer id, @Valid
@RequestBody ItemDto itemDto) {
        sout.info("Request to save by id :{} item :{}", id, itemDto);
        ItemDto item = itemService.update(id, itemDto);
        sout.info("Updated Item: {} with id: {}", itemDto, id);
        return ReesponsEntity.ok(item);}

    @PostMapping("/accessory")
    public ResponsePageDto<ItemDto> searchAccessory(@RequestBody SearchCriteriaDto
searchCriteria,
                                                    Pageable pageable) {
        Page<ItemDto> page = accessoryService.search(searchCriteria, pageable);
        sout.debug("Items(Accessory) found : {}", page);
        return new ResponsePageDto(page.getContent(), page.getTotalElements(),
page.getTotalPages());
    }
    @PostMapping("/clock")
    public ResponsePageDto<ItemDto> searchClock(@RequestBody SearchCriteriaDto
searchCriteria,
                                                Pageable pageable) {
        Page<ItemDto> page = clockService.search(searchCriteria, pageable);
        sout.debug("Items(Clock) found : {}", page);
        return new ResponsePageDto(page.getContent(), page.getTotalElements(),
page.getTotalPages());}}
```

Клас UserController

```
public class User Controller {
    private final UserService userService;
    @GetMapping("/{id}")
    public ResponseEntity<UserDto> getById(@PathVariable int id) {
        UserDto byId = UserTransformer.transform(userService.getById(id));
        sout.debug("User {} found", byId);
        return new ResponseEntity<>(byId, HttpStatus.OK);}
    @GetMapping("/search/{telegaId}")
    public ResponseEntity<UserDto> getByTelegaId(@PathVariable int telegaId)
{byTelegaId = UserTransformer.transform(userService.findByTelegaId(telegaId));
        sout.debug("User with telegaID {}", byTelegaId);
        return new ResponseEntity<>(byTelegaId, HttpStatus.OK);}
    @PutMapping("/telega-name-username/{telegaId}")
```

```
    public ResponseEntity<UserDto> updateFullNameAndUserName(@PathVariable Integer
telegaId, @RequestParam String fullName, @RequestParam String userName) {
        soutsout.iinfo(«Reequest to update user with id {}  telegafullname {}
userName {}", telegaId, fullName, userName);
        UserDto user =
UserTransformer.transform(userService.updateTelegaName(telegaId, fullName,
userName));
        soutsout.iinfo(«Reequest to update user with id {}  telegafullname {}
userName {} processed.", telegaId, fullName, userName);
        return new ResponseEntity<UserDto>(user, HttpStatus.OK);}
    @PutMapping("/language/{telegaId}")
    public ResponseEntity<Void> updateLanguage(@PathVariable int telegaId,
@RequestParam String language) {
        soutsout.iinfo(«Reequest to update user language with id {} to {} received.",
telegaId, language);
        userService.updateLanguage(telegaId, language);
        soutsout.iinfo(«Reequest to update user language with id {} to {}
processed.", telegaId, language);
        return ReesponsEntity.ok().build();    @PostMapping
    public ResponseEntity<UserDto> create(@RequestBody UserDto userDto) {
        userDto.setAdmin(false);
        userDto.setSupport(false);
        UserDto save =
UserTransformer.transform(userService.saveNewUser(UserTransformer.transform(userDto))
);
        sout.debug("User {} saved", save);
        return new ResponseEntity<>(save, HttpStatus.CREATED);}
    @PutMapping("/{id}")
    public ResponseEntity<UserDto> update(@PathVariable int id, @RequestBody UserDto
user) {
        UserDto update = UserTransformer.transform(userService.update(id,
UserTransformer.transform(user)));
        sout.debug("User with id:{} updated to {}", id, user);
        return new ResponseEntity<>(update, HttpStatus.CREATED);
    }
    @PostMapping("/update-role")
    public ResponseEntity updateUserRole(@RequestParam int id, @RequestParam Boolean
admin, @RequestParam Boolean support) {
        sout.info("Request to update roles on user with id {} to  admin: {} and
support {} roles.", id, admin, support);
        userService.updateRole(id, admin, support);
        sout.info("Request to update roles on user with id {} to  admin: {} and
support {} roles processed.", id, admin, support);
        return ReesponsEntity.ok().build
    @PatchMapping("/provided-referral")
    public boolean updateProvidedReferral(@RequestParam int userId, @RequestParam
String providedReferral) {
        sout.info("Request to update provided referral. User id: {}, provided
referral: {}", userId, providedReferral);
        boolean result = userService.updateProvidedReferral(userId,
providedReferral);
        sout.info("Request to update provided referral processed. User id: {},
result: {}", userId, result);
        return result;}
    @PostMapping("/search")
    public ResponsePageDto<UserDto> search(@RequestBody SearchCriteriaDto
searchCriteria, Pageable pageable) {
        sout.debug("Request to search users: {}", searchCriteria);
        ResponsePageDto<UserDto> user = userService.search(searchCriteria, pageable);
        sout.debug("Request to search users processed: {}", searchCriteria);
        return user;}
```

Клас VariationController

```
public class VariationController {

    private final VariationService variationService;
    @GetMapping("/{id}")
    public ResponseEntity<VariationDto> getById(@PathVariable Integer id) {
        VariationDto byId =
VariationTransformer.transform(variationService.getById(id));
        sout.debug("Variation {}.found", byId);
        return ReesponsEntity.ok(byId);
    }
    @PostMapping
    public ResponseEntity<VariationDto> create(@RequestBody VariationDto
variationDTO) {
        VariationDto save = VariationTransformer

.transform(variationService.save(VariationTransformer.transform(variationDTO)));
        sout.debug("Variation {}.saved", save);
        return ReesponsEntity.ok(save);
    }
    @PutMapping("/{id}")
    public ResponseEntity<VariationDto> update(@PathVariable Integer id, @RequestBody
VariationDto variationDTO) {
        VariationDto update = VariationTransformer
                .transform(variationService.update(id,
VariationTransformer.transform(variationDTO)));
        sout.debug("Variation with id:{}. updated to {}.", id, variationDTO);
        return ReesponsEntity.ok(update);
    }
    @PostMapping("/pagination")
    public ResponsePageDto<VariationDto> search(@RequestBody SearchCriteriaDto
searchCriteria, Pageable pageable) {
        Page<Variation> variations = variationService.search(searchCriteria,
pageable);
        List<VariationDto> list = variations.getContent()
                .stream()
                .map(VariationTransformer::transform)
                .collect(Colectors.toList());
        Page<VariationDto> page = new PageImpl<>(list, variations.getPageable(),
variations.getTotalElements());
        sout.debug("Variation {}", variations);
        return new ResponsePageDto(list, variations.getTotalElements(),
page.getTotalPages());}}
```

Клас CartTransformer

```
@NoArgsConst1ructor(access = AcceessLevel.PRIVATE)
public final class CartTransformer {

    public static Cart transform(NewCartRequestDto dto) {

        return Optional.ofNullable(dto)
            .map(c -> Cart.builder()
                .userId(c.getUserId())
                .address(c.getAddress())
                .fullName(c.getName())
                .phoneNumber(c.getPhone())
```

```
                .build())
            .orElse(null);
    public static Cart transform(CartDto dto) {
        return Cart.builder()
            .id(dto.getId())
            .address(dto.getAddress())
            .fullName(dto.getFullName())
            .userId(dto.getUserId())
            .operatorId(dto.getOperatorId())
            .payment(PaymentTransformer.transform(dto.getPayment()))
            .phoneNumber(dto.getPhoneNumber())
            .status(dto.getStatus())
            .lastModifiedDate(dto.getLastModifiedDate())
            .createDate(dto.getCreatedDate())
            .build();    public static CartDto transform(Cart cart) {
        return CartDto.builder()
            .id(cart.getId())
            .address(cart.getAddress())
            .fullName(cart.getFullName())
            .userId(cart.getUserId())
            .operatorId(cart.getOperatorId())
            .payment(PaymentTransformer.transform(cart.getPayment()))
            .phoneNumber(cart.getPhoneNumber())
            .status(cart.getStatus())
            .lastModifiedDate(cart.getLastModifiedDate())
            .createdDate(cart.getCreateDate())
            .build(); }}
```

Клас ItemTransformer

```
@NoArgsConstruc1tor(access = AcceessLevel.PRIVATE)
public class ItemTransformer {


    public static Item transform(ItemDto item) {
        return Optional.ofNullable(item)
            .map(i -> Item.builder()
                .id(item.getId())
                .name(i.getName())
                .description(i.getDescription())
                .mediumImageUrl(i.getMediumImageUrl())
                .deleted(i.getDeleted())
                .itemType(item.getItemType())
                .build())
            .orElse(null);}
    public static ItemDto transform(Item item) {
        return Optional.ofNullable(item)
            .map(i -> ItemDto.builder()
                .id(i.getId())
                .name(i.getName())
                .description(i.getDescription())
                .mediumImageUrl(i.getMediumImageUrl())
                .deleted(i.getDeleted())
                .itemType(item.getItemType())
                .build())
            .orElse(null);
    }
}
```

Клас PaymentTransformer

```
@NoArgsConstruc1tor(access = AcceessLevel.PRIVATE)
public class PaymentTransformer {

    public static Payment transform(PaymentDto dto) {
        return transform(null, dto);

    }
    public static Payment transform(Integer id, PaymentDto dto) {
        return Optional.ofNullable(dto)
            .map(pay -> Payment.builder()
                .id(id != null ? id : pay.getId())
                .status(pay.getStatus())
                .totalPrice(pay.getTotalPrice())
                .actualTotalPrice(pay.getActualTotalPrice())
                .bonusesUsed(pay.getBonusesUsed())
                .cryptoRatioToUah(pay.getCryptoRatioToUah())
                .type(pay.getType())
                .memo(pay.getMemo())
                .address(pay.getAddress())
                .geopayInvoiceUuid(pay.getGeopayInvoiceUuid())
                .build())
            .orElse(null);}
    public static PaymentDto transform(Payment payment) {
        return Optional.ofNullable(payment)
            .map(pay -> PaymentDto.builder()
                .id(pay.getId())
                .status(pay.getStatus())
                .totalPrice(pay.getTotalPrice())
                .actualTotalPrice(pay.getActualTotalPrice())
                .bonusesUsed(pay.getBonusesUsed())
                .cryptoRatioToUah(payment.getCryptoRatioToUah())
                .type(pay.getType())
                .memo(pay.getMemo())
                .address(pay.getAddress())
                .geopayInvoiceUuid(pay.getGeopayInvoiceUuid())
                .build())
            .orElse(null);}}
```

Клас Payment

```
@ Data
@ Entity
@ Builderr
@ NoArgsConstructor
@ AllArgsConstructor
@Table(name = "payment")
public class Payment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "payment_status")
    @Enumerated(EnumType.STRING)
    private PaymentStatus status;
    @Column(name = "payment_type")
```

```
    @Enumerated(EnumType.STRING)
    private PaymentType type;
    @Column(name = "total_price")
    private BigDecimal totalPrice;
    @Column(name = "actual_total_price")
    private BigDecimal actualTotalPrice;
    @Column(name = "bonuses_used")
    private BigDecimal bonusesUsed;
    @Column(name = "ratio_to_uah")
    private BigDecimal cryptoRatioToUah;
    private String memo;
    @Column(name = "wallet_address")
    private String address;
    private String geopayInvoiceUuid;
}
```

Клас User

```
@ Data
@ Entity
@ Builderr
@ NoArgsConstructor
@ AllArgsConstructor
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private Integer telegaId;
    private String telegaFullName;
    private String telegaUserName;
    private String language;
    private Long chatId;
    private String referral;
    private String providedReferral;
    private Boolean admin;
    private Boolean support;
    private Boolean superAdmin;
    private BigDecimal bonus;

    private String fullName;
    private String phone;
    private String deliveryAddress;

}
```

Клас Clock

```
@ NoArgsConstructor
@ AllArgsConstructor
@ Table(name = "clock")
public class Clock {
    @Id
    private Integer id;
```

```
    private Integer variationId;
    private Integer manufacturerId;
    @Enumerated(EnumType.STRING)
    private MechanismType mechanismType;
    private BigDecimal waterproof;
    private String untappd;
    private String bracelet_material;
    private String body_material;
}
```

Клас Item

```
@Table(name = "item")
public class Item {    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String name;
    private String description;
    private String mediumImageUrl;
    @Enumerated(EnumType.STRING)
    private ItemType itemType;
    private Boolean deleted;
}
```

Клас AccessoryRepositoryImpl

```
@Repository
public class AccessoryRepositoryImpl implements AccessoryRepositoryCustom {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public List<Accessory> search(SearchCriteriaDto searchCriteria, Integer
startPosition, Integer pageSize) {
        String whereCondition = getSearchWhereConditions(searchCriteria);
        String orderByCondition = getSearchOrderByConditions(searchCriteria);

        String mainQuery = String.format("SELECT * FROM accessory s JOIN
accessory_category sc ON s.accessory_category_id = sc.id"
                + " JOIN item it ON it.id = s.id JOIN inventory inv ON it.id =
inv.item_id  %s ORDER BY %s LIMIT %d,%d",
                                        whereCondition, orderByCondition,
startPosition, pageSize);
        Query query = entityManager.createNativeQuery(mainQuery, Accessory.class);
        updateQueryWithParameters(query, searchCriteria); query.getResultList();}
    @Override
    public Long getTotalElements(SearchCriteriaDto searchCriteria) {
        String whereCondition = getSearchWhereConditions(searchCriteria);
        String countQuery = String
            .format("SELECT count(*) FROM accessory s JOIN item it ON it.id = s.id
JOIN inventory inv ON it.id = inv.item_id %s ",
                    whereCondition);
        Query query = entityManager.createNativeQuery(countQuery);
```

```
        updateQueryWithParameters(query, searchCriteria);

        return Long.valueOf(query.getSingleResult().toString());
 }    private Query updateQueryWithParameters(Query query, SearchCriteriaDto
searchCriteriaRequestParams) {
(Objects.nonNull(searchCriteriaRequestParams.getAccessoryCategoryId())) {
            query.setParameter("id",
searchCriteriaRequestParams.getAccessoryCategoryId());
        }if (Objects.nonNull(searchCriteriaRequestParams.getAccessoryName())) {
            query.setParameter("name", searchCriteriaRequestParams.getName());
}return query;}
```

Клас UserRepositoryImpl

```
@Servce
public class UserRepositoryImpl implements UserRepositoryCustom {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    @SuppressWarnings("unchecked")
    public List<User> search(SearchCriteriaDto searchCriteria, Integer startPosition,
Integer pageSize) {
        String whereCondition = getSearchWhereConditions(searchCriteria);
        String orderByCondition = getSearchOrderByConditions(searchCriteria);
        String mainQuery = String.format("SELECT * FROM user u %s ORDER BY %s LIMIT
%d, %d",
                whereCondition, orderByCondition, startPosition, pageSize);
        Query query = entityManager.createNativeQuery(mainQuery, User.class);
        return query.getResultList();
    }
    @Override
    public Long getTotalElements(SearchCriteriaDto searchCriteria) {
        String whereCondition = getSearchWhereConditions(searchCriteria);
        String countQuery = String.format("SELECT count(*) FROM user u %s",
whereCondition);
        Query query = entityManager.createNativeQuery(countQuery);
        return Long.valueOf(query.getSingleResult().toString());
    }
```

Клас BlockService

```
@Servce
@RequiredArgsConstructor
public class BlockService {
    private final BlockRepository blockRepository;
    public Block update(Block block) {
        Block findBlock = blockRepository.findFirstByOrderByIdAsc();
        if (findBlock != null) {
            block.setId(findBlock.getId());}
        return blockRepository.save(block);
    }
    public Block getFirst() {
        return blockRepository.findFirstByOrderByIdAsc();}
```

Клас ItemService

```
@Servce
@RequiredArgsConstructor
public class ItemService {
    private final ItemInfoService itemInfoService;
    private final List<ItemProcessingStrategy> strategy;

    public ItemDto save(ItemDto itemDto) {
        return strategy.stream().filter(s -> s.getType() ==
itemDto.getItemType()).findFirst().get().save(itemDto);
    }

    public ItemDto update(Integer id, ItemDto itemDto) {
        return strategy.stream().filter(s -> s.getType() ==
itemDto.getItemType()).findFirst().get().update(id,
                itemDto);}
    public ItemDto getById(Integer id) {
        ItemDto itemDto = ItemTransformer.transform(itemInfoService.getById(id));
        return strategy.stream().filter(s -> s.getType() ==
itemDto.getItemType()).findFirst().get().getById(id);}}
```

Клас ReferralService

```
@Servce
@RequiredArgsConstructor
public class ReferralService {
    private final UserService userService;
    private final CartService cartService;
    private final PaymentService paymentService;
    private final ReferralBonusRepository referralBonusRepository;

    @Value("${bonus.multiplier}")
    private String BONUS_MULTIPLIER;

    public void checkBonusPayment(Cart cart, Payment payment) {
        User user = userService.getByUserIdForUpdate(cart.getUserId());
        BigDecimal priceUah = PriceTransformer.toUah(payment.getTotalPrice(),
payment.getCryptoRatioToUah());
        if (priceUah.compareTo(user.getBonus()) <= 0) {
            user.setBonus(user.getBonus().subtract(priceUah));
            userService.update(user.getId(), user);
            payment.setStatus(PaymentStatus.PAID);
            payment = paymentService.save(payment);
            cart.setPayment(payment);
            cartService.save(cart);
            addBonuses(cart, user);
        } else {
String meessage = Strring.format(«Error when performing checkout for user with id
%s", cart.getUserId());
            throw new CheckoutException(message, List.of(ErrorDetail.builder()
                .messageKey("error.not.enough.bonuses")
                .args(new String[] { priceUah.stripTrailingZeros().toPlainString(),
                        user.getBonus().stripTrailingZeros().toPlainString()
}).build()));}}
    public boolean addBonuses(Cart cart, User user) {
```

```
        if (user.getProvidedReferral() != null) {
            User referralUser =
userService.findByReferralForUpdate(user.getProvidedReferral());
            BigDecimal bonus = calculateBonus(cart.getPayment());
            referralUser.setBonus(referralUser.getBonus().add(bonus));
            userService.save(referralUser);
            ReferralBonus referralBonus = new ReferralBonus(cart.getId(),
user.getId(), bonus);
            referralBonusRepository.save(referralBonus);
            return true;
        }
        return false;
    }
    private BigDecimal calculateBonus(Payment payment) {
        return PriceTransformer.toUah(payment.getTotalPrice(),
payment.getCryptoRatioToUah())
            .multiply(new BigDecimal(BONUS_MULTIPLIER));
    }
}
```

Клас StoryService

```
@Servce
@RequiredArgsConstructor
public class StoryService {
    private final StoryRepository storyRepository;
    private final NotificationRepository notificationRepository;
    public Story save(Story story) {
story.setPublishDate(ConvertFromUkranianToSystemTime.apply(story.getPublishDate()));
        story = storyRepository.save(story);

story.setPublishDate(ConvertFromSystemToUkranianTime.apply(story.getPublishDate()));
        return story;}
    public StoryDto getById(Integer id) {
        StoryDto story = storyRepository.getById(id);

story.setPublishDate(ConvertFromSystemToUkranianTime.apply(story.getPublishDate()));
        return story;}
    public Story update(Integer id, Story story) {
        story.setId(id);
story.setPublishDate(ConvertFromUkranianToSystemTime.apply(story.getPublishDate()));
        story = storyRepository.save(story);

story.setPublishDate(ConvertFromSystemToUkranianTime.apply(story.getPublishDate()));
        return story;}
    public void publish(Integer storyId) {
        StoryDto story = getById(storyId);

story.setPublishDate(ConvertFromUkranianToSystemTime.apply(story.getPublishDate()));
        notificationRepository.insertWithStoryText(story.getText(),
story.getPublishDate());
    }
    public Page<StoryDto> search(SearchCriteriaDto searchCriteriaDTO, Pageable
pageable) {
        Integer startPosition = pageable.getPageNumber() * pageable.getPageSize();
        List<StoryDto> storyDTOs = storyRepository.search(searchCriteriaDTO,
startPosition, pageable.getPageSize());
```

```
        storyDTOs.stream().forEach(s ->
s.setPublishDate(ConvertFromSystemToUkranianTime.apply(s.getPublishDate())));
        Long totalElements = storyRepository.getTotalElements(searchCriteriaDTO);
        return new PageImpl<>(storyDTOs, pageable, totalElements);}
```

Клас VariationService

```
@Servce
@RequiredArgsConstructor
public class VariationService {

    private final VariationRepository variationRepository;

    public Variation save(Variation variation) {
        return variationRepository.save(variation);
    }
    public Variation getById(Integer id) {
        return variationRepository.getById(id);}
    public Page<Variation> search(SearchCriteriaDto searchCriteria, Pageable
pageable) {
        Integer startPosition = pageable.getPageNumber() * pageable.getPageSize();
        List<Variation> variations = variationRepository.search(searchCriteria,
startPosition, pageable.getPageSize());
        return new PageImpl<>(variations, pageable,
variationRepository.countByDeletedIs(Boolean.FALSE));
    }
    public Variation update(Integer id, Variation variation) {
        variation.setId(id);
        return variationRepository.save(variation);}}
```

Клас ErrorInfoDto

```
public class ErrorInfoDto implements Serializable {

    @NotNull
    private Integer status;
    @NotNull
    private List<String> messages;
    @NotNull
    private String debugMessage;
    @NotNull
    private Integer code;
    private String header;
    private List<ErrorDetail> errorDetails;
}
```

Клас GlobalControllerExceptionHandler

```
@Slf4j
@ControllerAdvice
public class GlobalControllerExceptionHandler {
    private static final String MESSAGE_PREFIX = " | Message: ";
```

```
    @ExceptionHandler(BaseException.class)
    public ResponseEntity<ErrorInfoDto> handleBaseException(BaseException e,
HttpServletRequest request) {
        sout(request, e);
        ErrorInfoDto errorInfoDto = createErrorDto(e);
        return new ResponseEntity<>(errorInfoDto,
e.getExceptionCode().getHttpStatus());}
    @ExceptionHandler(value = CheckoutException.class)
    public ResponseEntity<ErrorInfoDto>
handleMethodArgumentForCheckoutException(CheckoutException e, HttpServletRequest
request) {
        sout(request, e);
        ErrorInfoDto errorInfoDto = createErrorDto(e);
        errorInfoDto.setErrorDetails(e.getErrorDetails());
        return new ResponseEntity<>(errorInfoDto,
e.getExceptionCode().getHttpStatus());}
    @ExceptionHandler(ServletException.class)
    public ResponseEntity<ErrorInfoDto> handleServletException(ServletException e,
HttpServletRequest request) {
        soutError(request, e);
        ErrorInfoDto errorInfoDto =
createErrorDto(ExceptionCode.PARAMETERS_ABSENT_OR_INVALID,
                ExceptionCode.PARAMETERS_ABSENT_OR_INVALID
                        .getDebugMessage());
        return new ResponseEntity<>(errorInfoDto, HttpStatus.BAD_REQUEST);}

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorInfoDto>
handleMethodArgumentNotValidException(MethodArgumentNotValidException ex,

HttpServletRequest request) {
        soutError(request, ex);
        List<ErrorDetail> errorDetails = ex.getBindingResult()
                .getFieldErrors()
                .stream().map(FieldError::getDefaultMessage)
                .map(e -> ErrorDetail.builder().messageKey(e).args(new
String[0]).build())
                .collect(Colectors.toList());
        ErrorInfoDto errorInfoDto =
createErrorDto(ExceptionCode.PARAMETERS_ABSENT_OR_INVALID,
ExceptionCode.PARAMETERS_ABSENT_OR_INVALID.getDebugMessage());
        errorInfoDto.setErrorDetails(errorDetails);
        return new ResponseEntity<>(errorInfoDto, HttpStatus.BAD_REQUEST);}
```

Клас ExchangeMarketRestTemplateConfig

```
@Configuration
public class ExchangeMarketRestTemplateConfig {

    @Bean(name = "exchangeMarketRestTemplate")
    public RestTemplate exchangeMarketRestTemplate(RestTemplateBuilder
restTemplateBuilder,
ExchangeMarketProperties exchangeMarketProperties) {
restTemplateBuilder = restTemplateBuilder.rootUri(exchangeMarketProperties.getUri())
.setConnectTimeout(Duration.ofMillis(exchangeMarketProperties.getTimeout().getCoennec
t()))
setReadTimeout(Duration.ofMillis(exchangeMarketProperties.getTimeout().getRead()));
return restTemplateBuilder.build();}}
```

Клас TelegramBotCommunicationServiceImpl

```
@Slf4j
@RequiredArgsConstructor
@Servce
public class TelegramBotCommunicationServiceImpl implements
TelegramBotCommunicationService {
    private final RestTemplate telegramBotRestTemplate;
    private final TelegramBotUriBuilder telegramBotUriBuilder;
    public void publishToTelegram(NotificationDTOB storyDTO) {
        sout.info("Publish to Telegram: {}", storyDTO);
        try {
            HttpEntity<NotificationDTOB> request = new
HttpEntity<>(storyDTO);telegramBotRestTemplate.exchange(telegramBotUriBuilder.getPubl
ishUri(), HttpMethod.POST, request,
                    NotificationDTOB.class);
        } catch (ResourceAccessException e) {
            String message = String.format("Failed sending store to telegram bot
service with message:%s", e.getMessage());
            throw new InnerServiceException(message);
        }
        sout.info("Request to telegram SUCCESS: {}", storyDTO);}
    @Override
    public void sendNotification(List<NotificationDto> notificationDtoList) {
        URI uri = telegramBotUriBuilder.getNotificationUri();
        HttpEntity<List<NotificationDto>> request = new
HttpEntity<>(notificationDtoList);
        try { sout.info("Request to send notifications {}", notificationDtoList);
 telegramBotRestTemplate.exchange(uri, HttpMethod.POST, request, Void.class);
            sout.info("Request to send notification processed.");
        } catch (ResourceAccessException e) {
            String message = String.format("No response from customer telegram bot on
send notifications %s", getNotificationsStringForError(notificationDtoList));
            throw new InnerServiceException(message);}}
```

Клас NotificationTransformer

```
@NoArgsConst2ructor(access = AcceessLevel.PRIVATE)
public class NotificationTransformer {

    public static List<NotificationDTOB> transform(List<Object[]> args) {
        return args.stream().map(a -> {
            NotificationDTOB.NotificationDTOBBuilder builder =
NotificationDTOB.builder();
            builder.text(String.valueOf(a[0]));
            List<UserStoryDTO> dtos = new ArrayList<>();
            Stream.of(String.valueOf(a[1]).split(",")).forEach(e -> {
                String[] elements = e.split("_");
                dtos.add(UserStoryDTO.builder()
                    .chatId(Long.valueOf(elements[0]))
                    .userStoryId(Integer.valueOf(elements[1]))
                    .build());
            });
            builder.userStoryDTOs(dtos);
            return builder.build();}).collect(Colectors.toList());}}
```

Клас TelegramBotUriBuilder

```
@Servce
@RequiredArgsConstructor
public class TelegramBotUriBuilder {

    private final TelegramBotProperties properties;

    public URI getPublishUri() {
        return
UriComponentsBuilder.fromUriString(properties.getUri()).path(properties.getPath().get
Story()).buiild().encoded().toUri();}
    public URI getUserSoutoutUri(Integer id) {
        return UriComponentsBuilder.fromUriString(properties.getUri())
.path(properties.getPath().getUserSoutout()).queryParam("telegramId",id).build().enco
de().toUri();}

    public URI getNotificationUri() {
        return UriComponentsBuilder.fromUriString(properties.getUri())
.path(properties.getPath().getNotification()).buiild().encoded().toUri();}
```

Клас TelegramBotRestTemplateConfig

```
@Configuration
public class TelegramBotRestTemplateConfig {
    @Bean(name = "telegramBotRestTemplate")
    public RestTemplate telegramBotRestTemplate(RestTemplateBuilder
restTemplateBuilder,
            TelegramBotProperties telegramBotProperties) {
        restTemplateBuilder =
restTemplateBuilder.rootUri(telegramBotProperties.getUri())
.setConnectTimeout(Duration.ofMillis(telegramBotProperties.getTimeout().getCoennect()
))
.setReadTimeout(Duration.ofMillis(telegramBotProperties.getTimeout().getRead()));
return restTemplateBuilder.build();}}
```

Клас CartService

```
@Servce
@RequiredArgsConstructor
public class CartService {

    private final CartRepository cartRepository;
    public List<Cart> findExpiredCarts(LocalDateTime expiredTime){return
cartRepository.findAllByCreateDateBeforeAndStatusAndPaymentStatusAndPayment_TypeIn(ex
piredTime, CartStatus.NEW, PaymentStatus.IN_PROCESS,
PaymentType.expiredPaymentTypes);
    }public List<Cart> saveAll(List<Cart>carts) {return
cartRepository.saveAll(carts);}
    public Cart save(Cart cart) {
        return cartRepository.save(cart);
    Transactional
    public Cart update(Integer id, Cart cart) {
        Cart firstById = getByIdForUpdate(id);
        cart.setId(firstById.getId());return cartRepository.save(cart);}
    public Cart getById(Integer id) {
```

```
        return cartRepository.getById(id);}
    public Page<Cart> searchByCriteria(CartSearchingCriteriaDto criteria) {
        Sort sort = Sort.unsorted();
        if (criteria.getStatusOrder() != null)sort =
sort.and(Sort.by(Direction.valueOf(criteria.getStatusOrder()), "status"));
        if (criteria.getCreateDateOrder() != null)
            sort = sort.and(Sort.by(Direction.valueOf(criteria.getCreateDateOrder()),
"createDate"));
        if (sort.isUnsorted()) {sort =
sort.and(Sort.by(Sort.Order.desc("lastModifiedDate")));}
return Optional.ofNullable(criteria.getUserId()).isPresent() ?
cartRepository.findAllByStatusInAndPaymentTypeInAndUserId(criteria.getCartStatus(),
criteria.getPaymentType(), criteria.getUserId(), PageRequest.of(criteria.getPage(),
criteria.getSize(), sort))
:cartRepository.findAllByStatusInAndPaymentTypeIn(criteria.getCartStatus(), criteria
.getPaymentType(), PageRequest.of(criteria.getPage(), criteria.getSize(),
sort));}public Cart getByIdForUpdate(int id) {return
cartRepository.getFirstById(id);}
    public Optional<Cart> findByPaymentMemoAndPaymentTypeForUpdate(String memo,
PaymentType paymentType) {return cartRepository.findByPaymentMemoAndPaymentType(memo,
paymentType);}
    public Optional<Cart> findByPaymentMemoForUpdate(String memo) {return
cartRepository.findByPaymentMemo(memo);}}
```

Клас NotificationService

```
@Servce
@RequiredArgsConstructor
public class NotificationService {

    private final UserService userService;
    private final NotificationRepository repository;
    private final TelegramBotCommunicationService telegramBotCommunicationService;
    private final Executor notificationExecutor = Executors.newCachedThreadPool();

    public void sendNotificationForPaidCart(User cartUser, Cart cart) {
        Runnable task = () -> {
            NotificationDto.Chat userChat = new
NotificationDto.Chat(cartUser.getChatId(), cartUser.getLanguage());
            NotificationDto customerNotification =
NotificationConverter.convert(cart, List.of(userChat), NEW_PAID_ORDER_CUSTOMER);
            List<UserChatIdLocale> supportUsers =
userService.getSupportUserInfoForNotification();
            List<NotificationDto.Chat> supportChatList = supportUsers.stream()
                    .map(NotificationDto.Chat::new).collect(Colectors.toList());
            NotificationDto supportNotification = NotificationConverter.convert(cart,
supportChatList, NEW_PAID_ORDER_SUPPORT);

telegramBotCommunicationService.sendNotification(List.of(customerNotification,
supportNotification));;
        notificationExecutor.execute(task);}
    public void sendNewCartNotification(Cart cart) {
        Runnable task = () -> {
            List<UserChatIdLocale> supportUsers =
userService.getSupportUserInfoForNotification();
            List<NotificationDto.Chat> supportChatList = supportUsers.stream()
                    .map(NotificationDto.Chat::new).collect(Colectors.toList());
```

```
            NotificationDto supportNotification = NotificationConverter.convert(cart,
supportChatList, NEW_ORDER);
telegramBotCommunicationService.sendNotification(List.of(supportNotification));};
        notificationExecutor.execute(task);
    }
    public void updateStatusInNotification(String status, List<Integer> userStoryIds)
{
        repository.updateStatusInNotification(status, userStoryIds);}
```

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний Університет
Факультет програмування та комп'ютерних
і телекомунікаційних систем
Кафедра інженерії прогорамного забезпечення

Дипломний проект, на тему:

«Програмна система для продажу годинників з використанням API Telegram»

Студент: Маєвський Ярослав Юрійович
Керівник: Гурман І. В.. Кандидат технічних наук, доцент

Вступ

- Месенджер – програмний засіб, з допомогою якого користувачі можуть обмінюватись текстовими повідомленнями або іншою інформацією представленню в альтернативному вигляді, в реальному часу. З часом, месенджери еволюціонували з простого інструмента обміну повідомлення в неймовірно потужні комерційні інструменти, в яких люди проводять більше часу, а ніж на будь-яких
- Важливу роль відіграли в цьому боти – спеціальні програми, що виконує автоматично або за певним розкладом запрограмовані дії. Програми, які працюють в месенджері. Така програма може відповідати на запитання, а також самостійно задавати їх. Чат-боти в комерції сильно зменшують витрати, покращують конверсію, і покращують обслуговування клієнтів. Сотні компаній роблять все можливе, щоб досягти вищезгаданих цілей, і для їх досягнення необхідні боти як пріоритетний інструмент інших інтернет ресурсах.

Актуальність та мета дипломного проекту

- Актуальність теми полягає в тому, що сьогодні в мережі інтернет є безліч сайтів, платформ, сервісів для покупки товарів, про те в Україні месенджери як інструмент для продажу товарів не є популярним і розвинутим. Інтеграція бізнесу в межі чат-бота покращить взаємодію підприємства з користувачем, так як месенджери є майже у всіх користувачів на пристроях.
- Метою проекту є реалізація програмного забезпечення, яке дозволить оптимізувати і автоматизувати продаж товарів серед користувачів, також забезпечить швидкий та зручний інтерфейс використовуючи API Telegram.

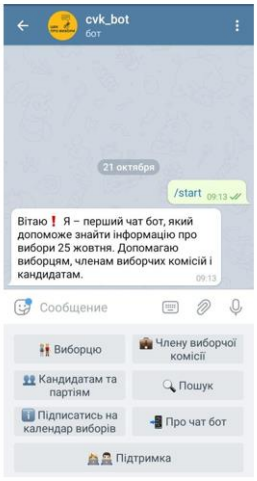
Завдання дипломного проекту

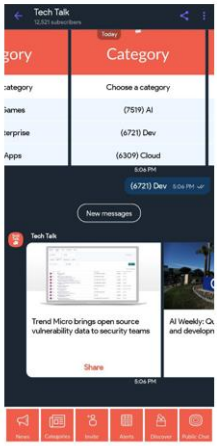
Для вирішення поставленої проблеми потрібно виконати наступні задачі:

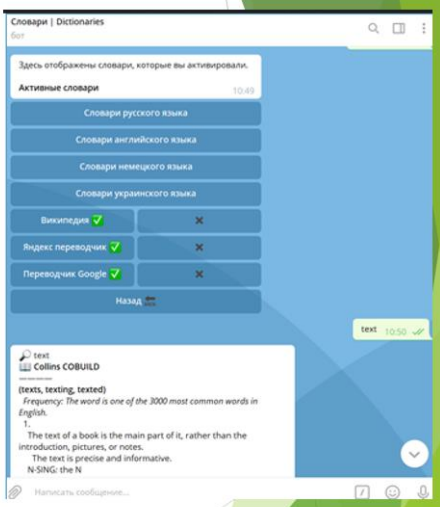
- Проаналізувати існуючі комерційні чат-боти.
- визначити функціональні задачі;
- Розробити технічне завдання;
- Спроектувати серверну архітектуру додатку.
- Спроектувати функціональний інтерфейс чатбота.
- Реалізувати програмну систему
- Провести тестування готового програмного забезпечення.

Наявне програмне забезпечення

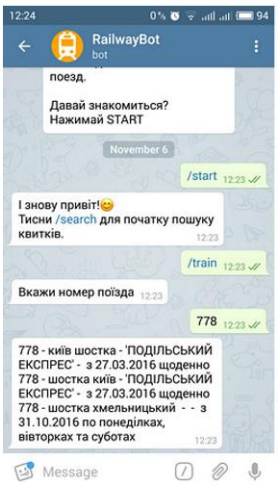
@WeatherBot
@NovaPoshtaBot
@TempMailBot
@ImageSearchBot
@UkrainianNewsBot
@PdaNewsBot
@EbaySearchBot
@KHMRealtorBot
@NationalGeoBot
@FTPBot

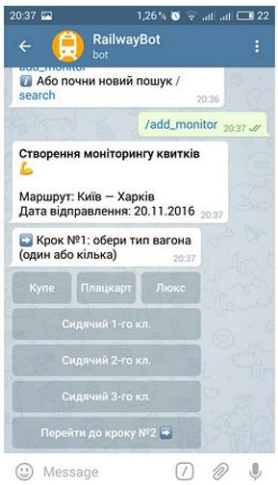


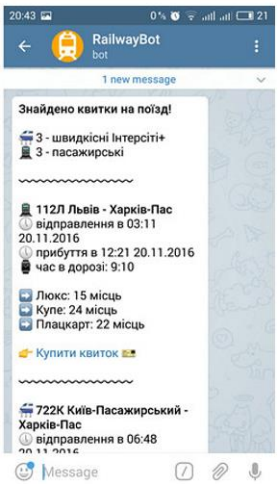




Наявне програмне забезпечення



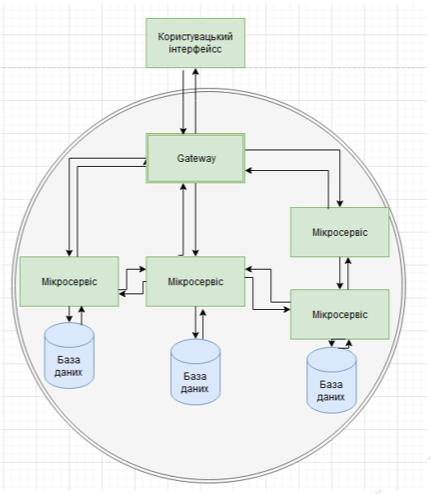




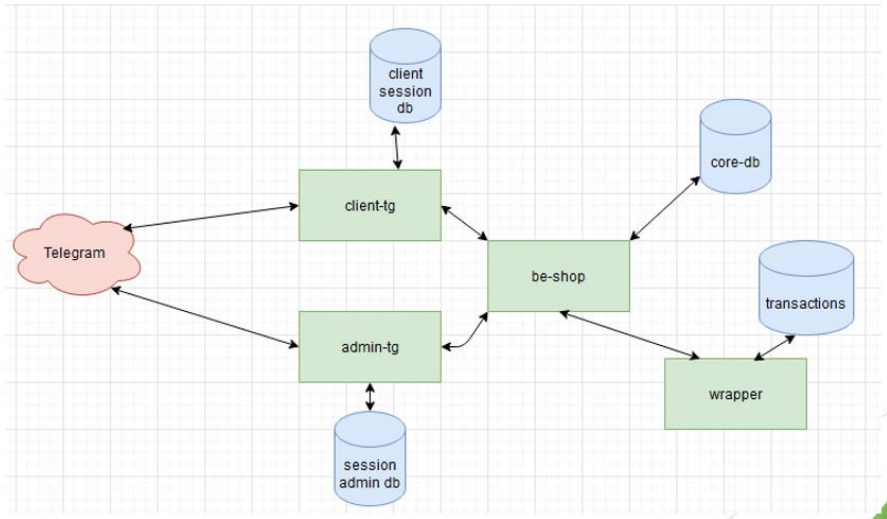
Висновки аналізу наявного програмного забезпечення

- В результаті аналізу існуючого програмного забезпечення було виявлено, що жодна з наведених вище чат-бот реалізацій не вирішує проблеми, які описані в постановці завдання, тому на даний момент у нас немає можливості проаналізувати подібні рішення, так як їх просто немає.
- Відсутність локалізації
- Слабкий функціонал
- Відсутність динамічного інтерфейсу

Проектування архітектури додатку



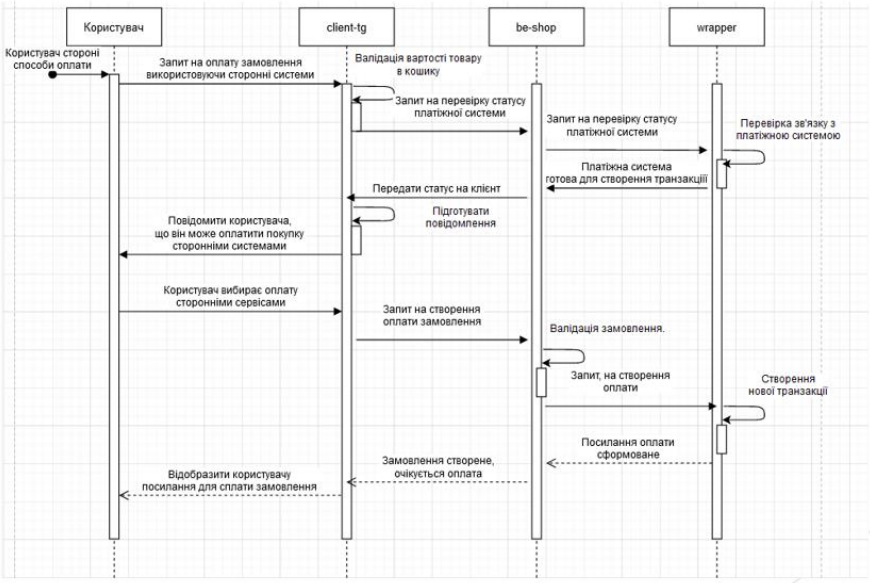
Проектування серверної архітектури



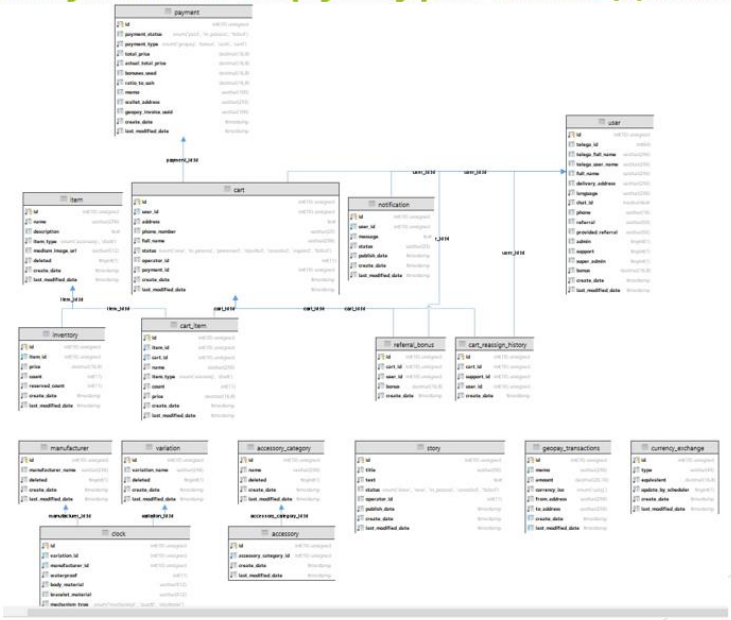
Аналіз та вибір технологій і методів програмного забезпечення

- Java - об'єктно орієнтована строго типізована мова програмування
- Висока та гнучка система безпеки
- Підтримка багатопоточності
- Підтримка мережевої взаємодії
- Взаємодія з базами даних
- Підтримка кешування інформації
- Spring - фреймворк для JavaEE, який значно розробку.
- Core (Context, spEl, IoC)
- Управління ORM
- Підтримка Web (HTTP REST)
- Spring Boot
- Spring Test module

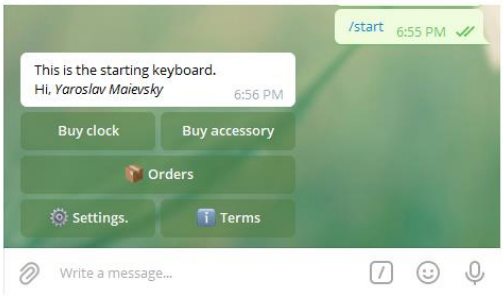
Діаграма послідовностей

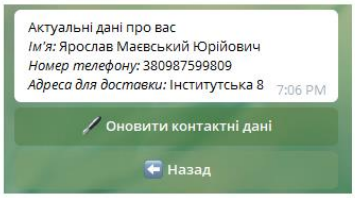


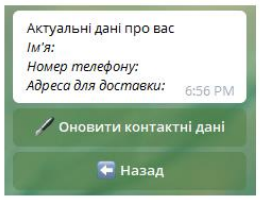
Проектування структури бази даних

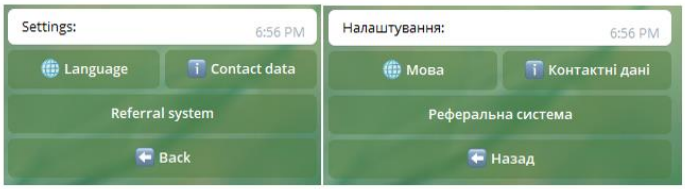


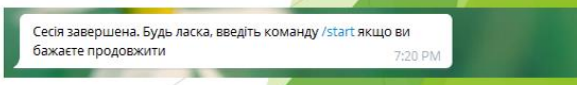
Користувацький інтерфейс чатбота



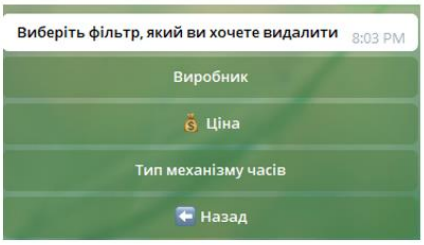


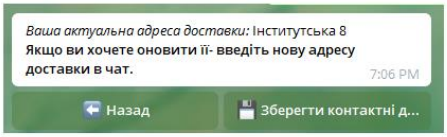


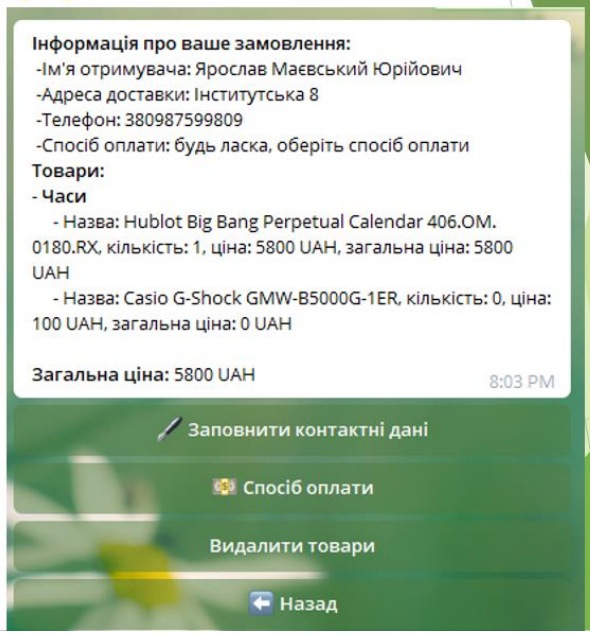




Користувацький інтерфейс

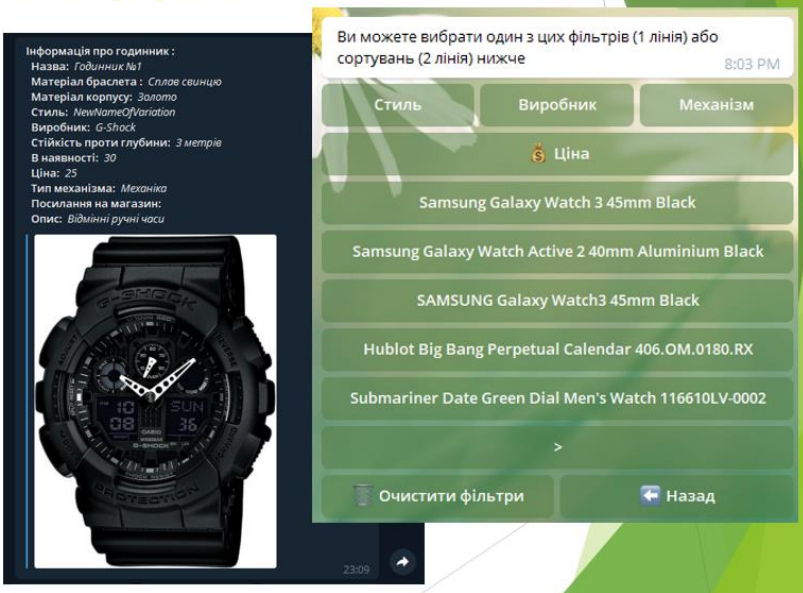




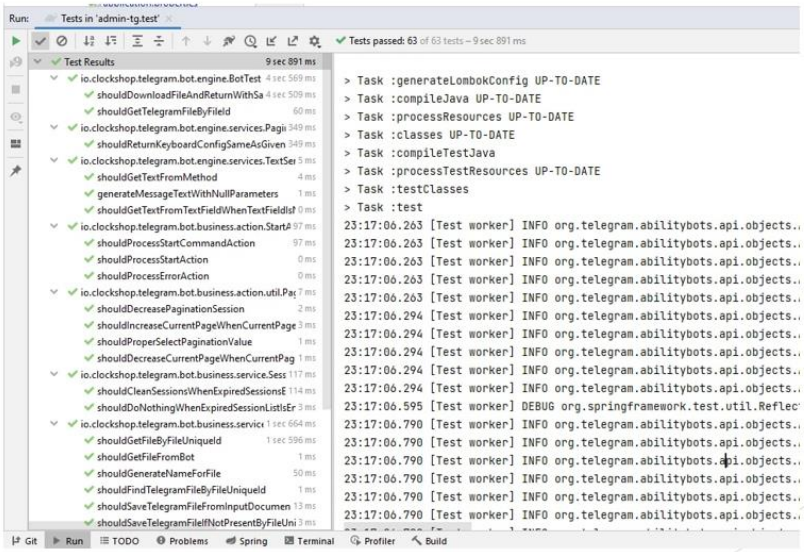


Користувацький інтерфейс

- Активна пагінація
- Динамічні фільтри
- Сортування за параметром
- Підтримка локалізації
- Робота з медіфайлами
- Підтримка HTML розмітки



Тестування програмного забезпечення



Висновки

- При виконанні дипломного проекту спочатку було досліджено і проаналізовано предметну область, і усі функціональні особливості, що в результаті підтвердило актуальність теми і необхідність подібних додатків.
- При аналізу вимог було до програмного забезпечення було поставлено задачі які необхідно реалізувати, і функціональні та нефункціональні вимоги.
- Після цих дій було розроблене програмне забезпечення і інструкція користувача для нього. Усі мікросервіси, бази даних були детально спроектовані за допомогою і описані за допомогою діаграм.
- При реалізації дипломного проекту було розроблено повноцінний серверний додаток, під час чого було закріплено практичні і теоретичні навички і знання розробки програмного забезпечення.

Дякую за увагу!

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Маєвського Я.Ю.
Прізвище, ініціали

факультет ПКТС, 4 курс, група ІПЗ-17-1

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 (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та зберження 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.

01.06.2021
дата

підпис

Fri Jun 04 22:13:50 EEST 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 16%**

	Документ		Сумарний збіг по Базі Даних	
ID: 92364 Назва: Програмна система для продажу годинників з використанням Telegram API Додано в БД: 2021-06-04 Автора: Я. Ю. Маєвський Керівники: І.В. Гурман Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	135584	1997	7868 (6%)	105 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1008188301

Дата перевірки:
05.06.2021 10:49:20 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2021 11:05:37 EEST

ID користувача:
100005589

Назва документа: **Диплом Маєвський ІПЗ-17-1**

Кількість сторінок: **133** Кількість слів: **23068** Кількість символів: **189989** Розмір файлу: **3.91 MB** ID файлу: **1008265193**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.39%
Схожість

Найбільша схожість: **3.73%** з джерелом з Бібліотеки (ID файлу: **1008215685**)

2.54% Джерела з Інтернету	299	Сторінка 135
5.3% Джерела з Бібліотеки	63	Сторінка 137

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.

Замінені символи	10
Підозріле форматування	23 сторінки

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Маєвський Ярослав Юрійович

Тема Програмна система для продажу годинників з використанням API Telegram

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 132

1.Короткий зміст пояснювальної записки та прийнятих рішень. У дипломному проекті було проведено детальний аналіз існуючих комерційних чат-ботів на базі основних месенджерів. Було спроектовано і побудовано серверну архітектуру веб-додатку на основі мікросервісних патернів. Для зберігання інформації використовувались реляційні і не реляційні бази даних. Для реалізації користувацького інтерфейсу було використано API Telegram. Програмне забезпечення протестоване за допомогою різноманітних модульних і інтеграційних тестів.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект освітнього ступеня "бакалавр" у повній мірі відповідає поставленому завданню як у теоретичній, так і в практичній її частині.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність дипломного проекту, детально описано предметну область проблеми, проаналізовано досліджувану проблему, і усі функціональні особливості, що підтвердило актуальність теми і необхідних подібних додатків. Сформульовано цілі і завдання дослідження, описано практична новизна та значимість реалізації. Було доведено переваги такої реалізації з поміж існуючих аналогів на ринку ПЗ. У першому розділі проведений аналіз комерційну область застосування чат-ботів в актуальних месенджерах. Наступні розділи посвячений проектуванню серверної архітектури додатку у вигляді мікросервісів і баз даних, проектування і реалізація користувацького інтерфейсу, а також проведення модульного і інтеграційного тестування веб-додатку.

4. Позитивні сторони проекту Дипломний проект є інноваційним на актуальний момент, аналогів з подібним функціоналом не існує на ринку. Існуючі аналоги чат ботів надають слабкий функціонал, в той час як цей проект надає весь необхідний функціонал онлайн магазину, а саме: особистий аккаунт, перегляд і замовлення товарів, обробка замовлень, панель адміністратора магазину. Серверна архітектура додатку реалізована за допомогою самих передових технологій і архітектурних принципів

5. Негативні сторони проекту Серверна архітектура реалізована за допомогою розподілених мікросервісів, з'являється проблема з розгортання додатку. Також потрібні інструменти для відслідковування статусу кожного з мікросервісів. Для кожного мікросервіса використовується окрема база даних, що ще більше ускладнює процес розгортання.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до теми дипломного проекту з дотриманням вимог стандартів. У загальному графічне оформлення виконане на достатньому рівні. Пояснювальна записка відповідає усім вимогам стандартів до її оформлення.

7. Відгук про дипломний проект в цілому В загальному реалізований дипломний проект заслуговує позитивної оцінки. Матеріал дипломного проекту структурований, чіткий та послідовний. Процес аналізу, проектування, реалізації і тестування описаний відповідно структурі записки. Графічний матеріал відображає актуальність і практичну цінність рішень, які були прийняті за основу для вирішення поставленої задачі.

8. Інші зауваження

9. Оцінка дипломного проекту Розглянувши реалізацію дипломного проекту, а також переваги і недоліки, можна зробити висновок, що робота заслуговує оцінки «відмінно»(4.75/A).

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Нічепорук Андрій Олександрович, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та системного програмування (КІСП) ХНУ

"____" _____ 202__ р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:
Назва: «Програмна система для продажу годинників з використанням API Telegram»

Автор: Маєвський Ярослав Юрійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Гурман Іван Васильович, канд. тех. наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
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 в бланках (титулка, бланк завдання, в структурі підрозділів ВСТУПУ) та в назвах публікацій джерел;
2) В якості запозичень системою було зафіксовано послідовності вихідного коду, які є спільними для великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) усі запозичення фрагментарні, або мають належним чином оформленні посилання.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності / схожості, складає 7,39% і адресується до 362 першоджерела, що з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ І.В. Гурман

Гарант ОП _____ Л.П. Бедратюк

Завідувач кафедри _____ Л.П. Бедратюк