Хмельницький національний університет

Факультет програмування

та комп'ютерних і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота

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 ДПППЗ.170110.01.10.ПЗ

Виконав студент IV курсу група ІПЗ-17-1 _____ О. В. Максимів
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ Г. І. Радельчук
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

04 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 ПРОЄКТ (РОБОТУ)

Максимову Олександру Володимировичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота

Керівник проєкту (роботи) Радельчук Галина Іванівна
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання
кандидат технічних наук, доцент

Затверджена наказом ректора університету від 05.02.2021 р. № 11

2. Строк подання студентом проєкту (роботи) на кафедру 01.06.2021 р.

3. Вихідні дані до проєкту (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 проєкту (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 проєкту (роботи)	Строк виконання етапів проєкту (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 _____ О. В. Максимів
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Г. І. Радельчук
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: «Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота».

Автор проекту: Максимів Олександр Володимирович.

Керівник проекту: Радельчук Галина Іванівна.

Пояснювальна записка: 129 с., 34 рис., 6 табл., 4 дод., 22 джерел.

Графічна частина: 16 презентаційних слайдів.

ЧАТ-БОТ, МЕСЕНДЖЕР, REST, WEB API, SPRING WEB, SPRING BOOT, SPRING JPA, SPRING TEST, LOMBOK, GRADLE, MONGODB.

Метою проекту є розробка програмного забезпечення, яке дозволяє оптимізувати та автоматизувати публікацію та пошук оголошень роботи або працівників серед користувачів ПЗ, забезпечує швидкий та зручний їх пошук, а також має сучасний інтуїтивно зрозумілий користувацький інтерфейс.

У дипломному проекті встановлено особливості та визначено специфіку месенджерів, проаналізовано використання реляційних і нереляційних баз даних та порівняно їх переваги і недоліки, виконано порівняння мікросервісної та монолітної архітектури, проведено аналіз та вирішено основні проблеми способів реалізації програмного забезпечення, встановлено особливості та правила для побудови користувацького інтерфейсу.

Для реалізацій програмного забезпечення використано мову програмування Java та фреймворк Spring з його додатковими надбудовами, нереляційну базу даних MongoDB і інтерфейс месенджера Telegram.

В результаті проектування виконана програмна реалізація програмного забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-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.170110.01.10.ПЗ	Пояснювальна записка	129		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні	16		
			слайди			

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.170110.01.10.ВД		
Виконав		Максимів О.В.		1.06	Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота Відомість документів	Літ.	Арк. 1 / Акрушів 1
Керівник		Радельчук Г.І.		1.06			
Н. контр.		Радельчук Г.І.		1.06		ХНУ, ІПЗ-17-1	
Зав. Каф.		Бедратюк Л.П.		1.06			

ЗМІСТ

Перелік скорочень 6
Вступ 7
1 Дослідження предметної області та постановка задачі 9
1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей 9
1.2 Аналіз наявного програмно-технічного забезпечення предметної області 13
1.3 Визначення вимог до програмного забезпечення та розробка технічного завдання 18
2. Проектування програмного забезпеченя 21
2.1 Аналіз клієнт-серверної архітектури та протоколів взаємодії сучасних веб-додатків 21
2.2 Аналіз та вибір архітектури серверної частини ПЗ 25
2.3 Аналіз та вибір типу бази даних 29
2.4 Проектування моделі бази даних 32
2.5 Детальне проектування серверної частини та розбиття її на модулі 35
2.6 Проектування інтерфейсу користувача 40
2.7 Аналіз та вибір технологій і методів реалізації системи 43
3. Програмна реалізація 48
3.1 Реалізація серверної частини ПЗ 48
3.2 Керівництво користувача 61
3.3 Вимоги до технічних та програмних засобів 68
3.4 Розгортання та встановлення системи 68
4. Тестування програмного забезпечення 71
4.1 Вибір та обгрунтування методів тестування додатку 71

4.2 Розробка тестових наборів даних 72
4.3 Аналіз результатів тестування системи 74
Висновки 77
Перелік джерел посилання 79
Додаток А Технічне завдання 81
Додаток Б Діаграми класів програми 88
Додаток В Код (лістинг) програми 91
Додаток Г Презентаційні матеріали 121

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
ВВ	–	варіант використання
ОС	–	операційна система
ПЗ	–	програмне забезпечення
ПК	–	персональний комп’ютер
UML	–	уніфікована мова візуального моделювання
API	–	Application programming interface
BDD	–	Behavior Driven Development
CLI	–	Command-line interface
XML	–	eXtensible Markup Language
HTTP	–	Hypertext Transfer Protocol
IDE	–	Integrated Development Environment
JVM	–	Java Virtual Machine
JRE	–	Java Runtime Environment
REST	–	Representational state transfer
RSS	–	Really Simple Syndication
TDD	–	Test-driven development
TCP	–	Transmission Control Protocol
URI	–	Uniform Resource Identifier
UI	–	User interface
UX	–	User experience
URL	–	Uniform Resource Locator

ВСТУП

Інформація є важливим ресурсом у будь-якій сфері людини. У всесвітній мережі Інтернет останнім часом популярнішим джерелом інформації стають месенджери, в яких можна обмінюватись повідомленнями з друзями, знаходити актуальні новини на потрібну тему та іншу корисну інформацію.

Месенджери, як інструмент обміну повідомленнями, перетворилися у засоби для отримання інформації та у неймовірно потужний маркетинговий інструмент, у яких люди проводять більше часу, ніж на інших Інтернет-ресурсах. Чималу роль у цьому відіграли боти.

Зараз існує безліч їх варіацій – від ботів для отримання RSS-розсилок до ботів для пошуку роботи або замовлення продовольчих товарів. Звісно, таким корисним інструментом цікавляться власники бізнесу, які хотіли б завжди «залишатись поруч» з їх споживачем. І хоча боти уже декілька років активно використовуються за кордоном, 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 враховувати, що для потенційних клієнтів важливим аспектом є досвід користувача при роботі з ботом, який включає те, як бот сприймає введення користувача, наскільки добре він виділяє суть запиту користувача та наскільки зрозумілими і доречними є його відповіді. Всі ці аспекти формують враження від бота та грають важливу роль у подальшому використанні його та рекомендації іншим користувачам.

Актуальність теми полягає у тому, що на сьогодні у мережі Інтернет є багато сайтів для пошуку роботи та пошуку працівників, проте в Україні месенджери, як інструмент для пошуку роботи, не настільки популярний і розвинутий. Що може бути приємніше, ніж замінити нудні вечори та години потраченого часу на пошук потрібного оголошення/працівника, на зручний, швидкий та зрозумілий інтерфейс телеграм-бота. Також досить актуально та перспективно використовувати бота для швидкого розвитку власного бізнесу.

Telegram охоплює не тільки молодіжну аудиторію, а й старше покоління, таким чином дозволяючи охопити досить велику аудиторію і досягти швидкого розвитку ідеї.

Метою проекту є розробка програмного забезпечення, яке дозволяє оптимізувати та автоматизувати публікацію і пошук оголошень роботи серед користувачів ПЗ, забезпечує швидкий та зручний їх пошук, а такаж має сучасний інтуєтивно зрозумілий користувацький інтерфейс.

Для реалізації програмного забезпечення необхідно виконати наступні завдання:

– визначити актуальність теми та встановити практичну користь розроблюваного ПЗ;

– дослідити наявне ПЗ предметної області та виявити недоліки;

– визначити вимоги до розроблюваного ПЗ та функції, які воно має виконувати;

– проаналізувати використання реляційних та нереляційних баз даних при реалізації програмного забезпечення;

– виконати порівняння мікросервісної і монолітної архітектури та вибрати одну з них;

– провести аналіз програмних модулів та встановити зв’язки між ними;

– провести аналіз способів реалізації програмного забезпечення та реалізувати серверну частину ПЗ;

– встановити особливості та загальні правила для побудови користувацького інтерфейсу та створити його;

– провести тестування програмної системи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Месенджер – це спеціальна програма, веб-сервіс або мобільний додаток, який надає користувачам можливість миттєво обмінюватись повідомленнями в режимі реального часу. Повідомлення відправляються співрозмовнику відразу після того, як відправник закінчить введення, редагування і натисне на кнопку відправки. Після відправки повідомлення користувачу, якому воно було адресоване, приходять сповіщення, якщо вони увімкнені. Найчастіше під месенджером розуміють програму, в яку ви пишете повідомлення і де ви їх читаєте. Однак, за кожною такою програмою стоїть мережа обміну повідомленнями, яка теж входить в поняття «месенджер». Це може бути мережа всередині вашої компанії, а може бути глобальна мережа.

Відмінність месенджера від електронної пошти тут в тому, що обмін повідомленнями йде в реальному часі (англ. Instant – миттєво). У попередніх версіях програм все, що друкував користувач, тут же відображалось співрозмовнику. Якщо він робив помилку і виправляв її, це теж було видно. У такому режимі спілкування нагадувало телефонну розмову. У сучасних програмах повідомлення з’являються на моніторі співрозмовника вже після закінчення редагування і відправки повідомлення.

Більшість користувачів бачать тільки клієнтську частину месенджера – це або програма на комп’ютері, або додаток на смартфоні (планшеті). Тим часом, всі месенджери мають власні сервери – там зберігається інформація та обробляються дані, які приходять від користувачів. Тому месенджери і називають клієнтами (клієнтськими програмами). Серверна частина забезпечує безпеку логінів, паролів та повідомлень, вона дозволяє знаходити контакти навіть тоді, коли їх власники не в мережі. Кожен з месенджерів працює за власним протоколом передачі даних, і ці протоколи рідко бувають сумісні, тобто якщо ви і ваш

знайомий маєте два різних месенджери, які не зв’язані між собою, то ви не зможете обмінюватись повідомленнями в них.

Існують месенджери з додатковими можливостями, такі як Telegram. Станом на 2021 рік месенджер Telegram вийшов на 11-е місце в світі серед найпопулярніших месенджерів за кількістю активних користувачів, а саме понад 500 мільйонів [1]. Цей месенджер має основні можливості стандартного месенджера та низку додаткових:

– спілкуватись за допомогою аудіо або відеозв’язку;
– зберігання вивантажених файлів;
– можливість брати участь в групових чатах;
– створювати засекречені чати;
– вбудований медіаплеєр;
– створювати «канали» зв`язку та переглядати або публікувати їх зміст;
– зберігати вивантаженні файли на необмежений термін;
– користуватись ботами та створювати їх [2].

Також існують тематичні месенджери, одним з таких є «Slack». Це також месенджер, але він дає ідеальні можливості та способи, щоб зібрати всю команду разом та обговорювати діяльність вашої компанії, також є можливість спілкуватись в окремих чатах на будь-яку тему. В ньому є ряд додаткових і зручних можливостей, таких як додавання реакції у вигляді прикріплення емодзі на текстові повідомлення, як ваші так і відправника, розширені можливості редагування текстового повідомлення, оформлення блоків програмного коду, додавання посилань на текст, перегляд вкладених файлів просто в чаті та інші функціональні можливості. Також «Slack» виділяється серед інших стандартних месенджерів тим, що в нього є інтеграція з іншими сервісами такими як «GitHub», «Google Drive», «MailChimp», «Google Docs», «Twitter», «Trello», «Google Hangouts», «DropBox» [3] та багатьма іншими.

Бот – це спеціальна комп’ютерна програма, яка виконує автоматично або за заданим сценарієм будь-які дії через інтерфейс месенджера. Боти призначені

для виконання різних функцій: від отримання актуальних новин до пошуку інформації і навіть торгування акціями компаній. Головне завдання бота – це давати автоматичну відповідь після отриманої користувачем команди. При цьому, команди можна вводити безпосередньо в автоматичному режимі як через інтерфейс месенджера, так і через звичайне текстове повідомлення. Використовуючи спеціальний синтаксис «/команда», програма імітує дії живого консультанта або ж робітника та дає відповідь на команду, або дію користувача. За рахунок такого функціоналу користування ботом стає досить зручним і зрозумілим.

Боти діляться на низку напрямків:

– чат-боти;

– боти-інформатори;

– ігрові боти;

– боти-асистенти;

– боти для виконання певних дій.

Чат-боти – це найпростіший чат, який імітує спілкування на конкретну тему, задану користувачем. Боти-інформатори відповідають за надсилання актуальних подій (новин, публікацій, заходів тощо). Ігрові боти дають можливість грати в прості ігри з нескладним інтерфейсом. Боти-асистенти доповнюють основну веб-версію онлайн-сервісу, для виконання певних дій. Боти для виконання певних дій налаштовуються на їх виконання та застосовуються там, де потрібна краща реакція в порівнянні з можливостями людини [4].

Деякі боти можуть містити в собі функціональні можливості декількох типів ботів. З їх допомогою них можна навчатись, тестувати, шукати інформацію та навіть взаємодіяти з глобальними сервісами такими як «Google Calendar», «Google Sheets», «Google Drive» та іншими. Вся функціональність, яка була вказана раніше, дозволяє замінити нудне перелистування сторінок сайту на зручний бот, який завжди під рукою у вашому смартфоні.

Більшість ботів побудовано на так званих станах. Оскільки програмне забезпечення наперед не знає, яку дію вибере користувач, розробником

створюються стани, для того, щоб бот розумів, яку дію виконує користувач та що йому потрібно відповісти. Алгоритм прийому і видачі відповіді бота досить примітивний. На серверну частину програмного забезпечення приходять запити у вигляді повідомлень або ж команд, які проходять низку ітерацій шифрування повідомлення через анонімний сервер месенджера, який і здійснює зворотній зв'язок між ПЗ і користувачем.

Можна виділити такий життєвий цикл відправки повідомлення користувачем: користувач бота надсилає команду у месенджері -> сервер месенджера отримує цю команду або повідомлення та надсилає його на серверну частину бота -> при отриманні повідомлення серверна частина бота взаємодіє з Telegram, обробляє отримане повідомлення від сервера месенджера та віддає відповідь -> сервер месенджера отримує відповідь від сервера бота -> бот виводить відповідь у вигляді повідомлення користувачеві. І цей цикл повторюється раз за разом, коли ви натискаєте на кнопки, вводите текст або команди, тим самим ви взаємодієте з ботом.

На сьогодні більшість людей вже звикла до сучасних технологій і більшість стараються полегшити виконання повсякденних справ за допомогою сучасних програмних забезпечень. Раніше для пошуку роботи люди проводили багато часу, шукаючи оголошення в газетах, рекламних стовпах. На сьогодні ця проблема легко вирішується відвідуванням сайтів для пошуку роботи та наявністю персонального комп'ютера, браузера і підключення до мережі Інтернет. Але все ж таки не в кожного є можливість дозволити собі покупку ПК та використання його в будь-якому місці світу, а ось смартфоном навпаки. Кількість смартфонів в населення України це 55% в загальному та 92% у молоді, вражають та показують, що більшість українців, а головне молодь, яка частіше шукає роботу, можуть дозволити собі смартфон та сім-карту з тарифним планом, який включає в себе доступ до мережі Інтернет [5]. На основі наведених фактів можна вважати, що програмне забезпечення, яке розробляється не тільки для ПК, а ще й для телефонів, охоплює в два або навіть в три рази більшу аудиторію. На

основі проведення дослідження предметної області, співставлення усіх фактів я пропоную реалізувати програмне забезпечення для покращення процесу пошуку працівників/роботи в месенджері Telegram у вигляді бота.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На сьогодні є безліч сайтів для пошуку роботи, що дозволяють публікувати, шукати та подавати заявки на вакансію. Telegram ботів є дуже мало, так як в Україні такий спосіб пошуку роботи непопулярний. Отже, важливо, щоб розроблюване програмне забезпечення було зручне, продуктивне та привабливе для нових користувачів.

Розглянемо деякі популярні веб-сайти для пошуку роботи та telegram-бота, проведемо порівняльну характеристику.

Основними критеріями для оцінювання є:

– функціональність;
– зручність;
– зрозумілість та привабливість користувацького інтерфейсу;
– продуктивність.

Розглянемо один з найпопулярніших сайтів для пошуку роботи в Україні «Work.ua». На рисунку 1.1 зображено головну сторінку сайту. На сайті є швидкий та розширений пошук, також наявний пошук за категоріям та посадами. На головній сторінці сайту розташовані популярні логотипи компаній натиснувши на яких можна побачити вакансії для вибраної компанії. Також на цьому сайті є місце і роботодавцям, їм можна створювати вакансії, вказувати унікальні характеристики для вакансії, що дозволить краще розуміти, що потрібно від претендента на роботу. Контент сайту відображається трьома мовами, а саме: українською, англійською, російською. Деякі вакансії мають логотип компанії,

що привертає увагу. Також на сайті є можливість використовувати мінімальні можливості месенджера.

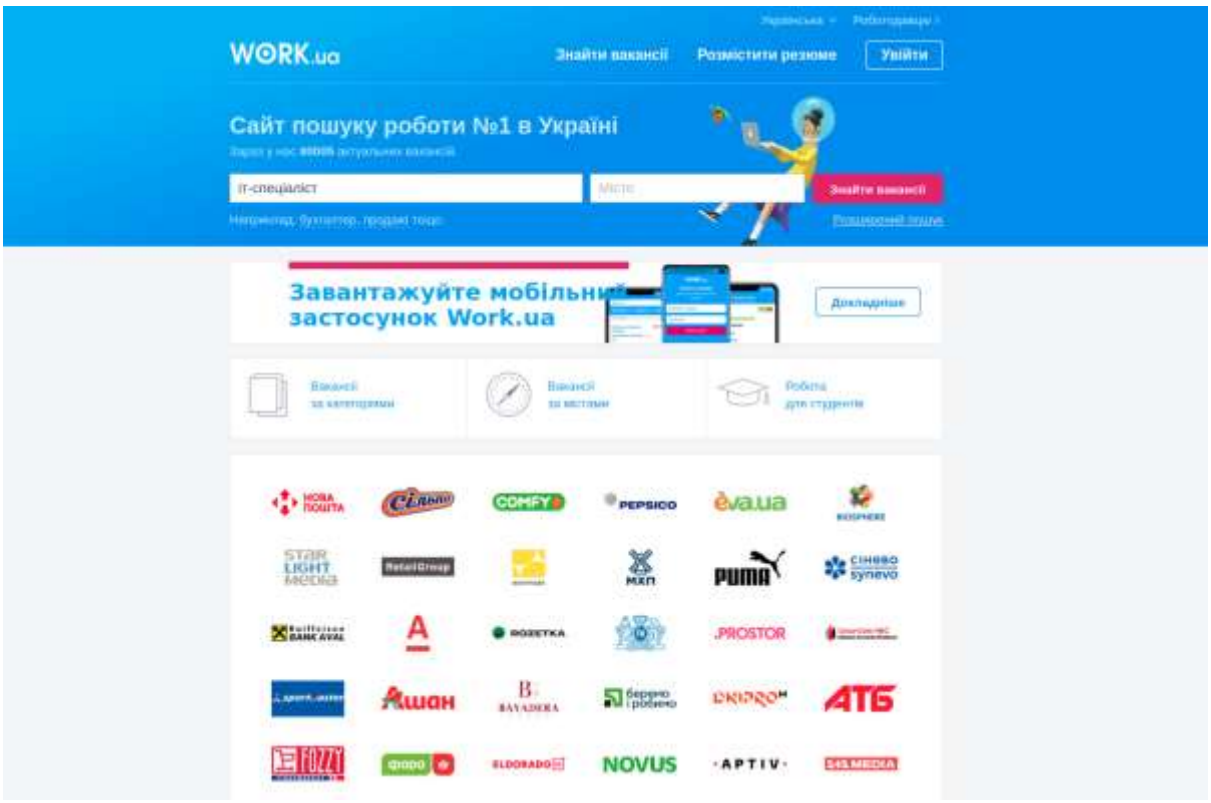


Рисунок 1.1 – Головна сторінка сайту «Work.ua»

Сайт інтуїтивно зрозумілий, швидкодія пошуку вакансій середня. Головні переваги цього сайту – це великі функціональні можливості та інтуїтивно зрозумілий інтерфейс. Мінуси сайту в тому, що потрібно мати два облікових записи на пошук роботи і на пошук працівників, що завдає великих незручностей, також потрібно потратити багато часу на реєстрацію та заповнення розширеного пошук за вакансіями.

На рисунку 1.2 зображено головну сторінку сайту «Olx rabota» для пошуку роботи. Цей сайт орієнтований на розміщення, продаж та пошук різноманітних товарів, але в ньому є ще додаткова сторінка для пошуку працівників/роботи. Інтерфейс мінімалістичний та сучасний, швидкодія пошуку вакансій середня. Контент на сайті відображається на двома мовами: українською та російською. Авторизація доступна через соціальні мережі, що надає додаткову швидкість

реєстрації, також доступна реєстрація особистого кабінету на самому сайті. Нявна можливість пошуку роботи за різними критеріями, але відсутні деякі критерії, такі як пошук за віком, статтю, освітою тощо. Мінус сайту в тому, що нові користувачі можуть заплутатися в пошуку потрібної їм інформації, так як сайт орієнтований на два різні напрямки.

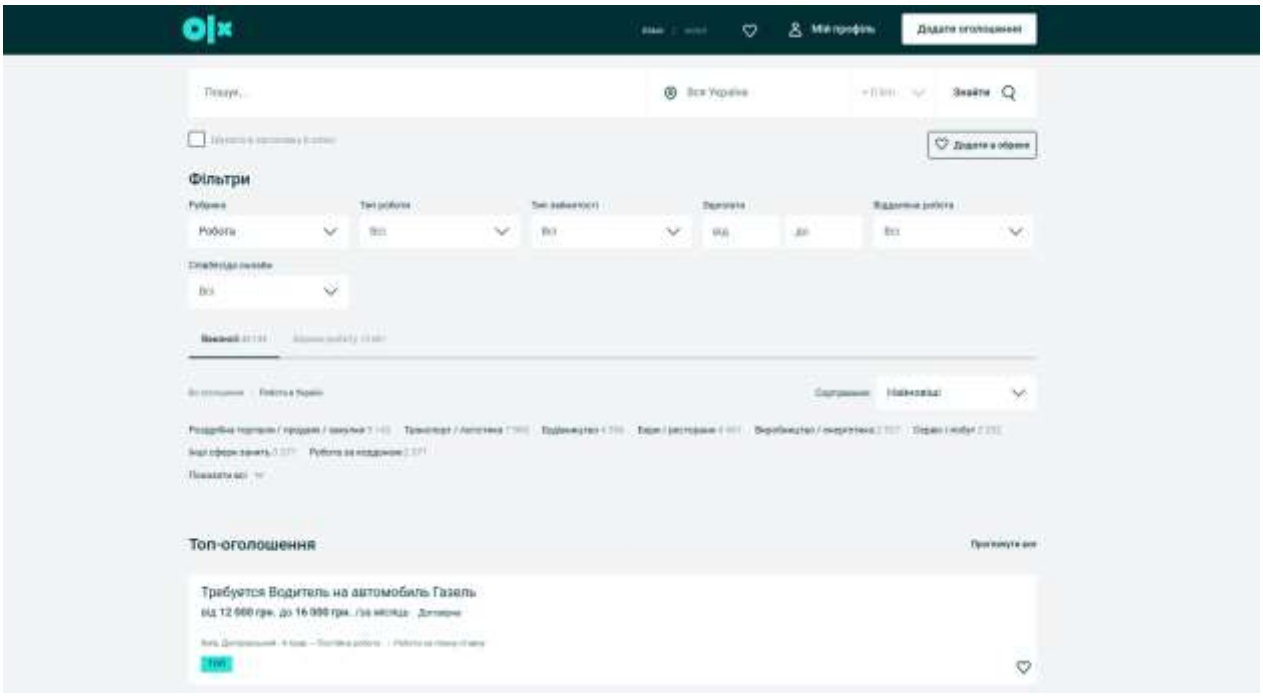


Рисунок 1.2 – Одна зі сторінок сайту «Olx rabota»

На рисунку 1.3 і 1.4 зображено приклад роботи Telegram-бота «@olx_robota_bot». Це бот-асистент, він напряму зв’язаний з сайтом «Olx rabota» та має деякі функціональні можливості, які присутні на сайті. У ньому наявна можливість підписки на вакансії, що дає можливість автоматично відправляти користувачу нову вакансію в конкретній категорії. Інтерфейс інтуїтивно зрозумілий, присутній швидкий вибір параметрів для пошуку вакансій. Контент на боті відображається українською мовою. Відсутня реєстрація клієнта в системі, що не дозволяє створити особисте резюме, відправити та редагувати його. Швидкодія пошуку роботи та роботи самого бота низька. Після заповнення критеріїв пошуку бот автоматично підбирає роботу користувачу та надсилає

результати пошуку, в яких міститься сума заробітної плати та посилання на основний сайт https://www.olx.ua/rabota/ з вакансією, що створює ряд незручностей для користувача. Результат пошуку відображає тільки п’ять вакансій, розподіл на сторінки відсутній.

Основні мінуси цього бота:

– відсутня можливість відправити своє резюме в самому боті;

– відсутня можливість реєстрації користувача в боті;

– відсутня можливість відображення повної інформації про заявку в інтерфейсі бота;

– відсутня можливість спілкування з роботодавцем в боті;

– відсутня можливість виступати в ролі роботодавця;

– відсутній розширений пошук за віком, статтю та іншими критеріями;

– відсутнє пояснення, як працює бот;

– відсутнє розбиття на сторінки результату пошуку.

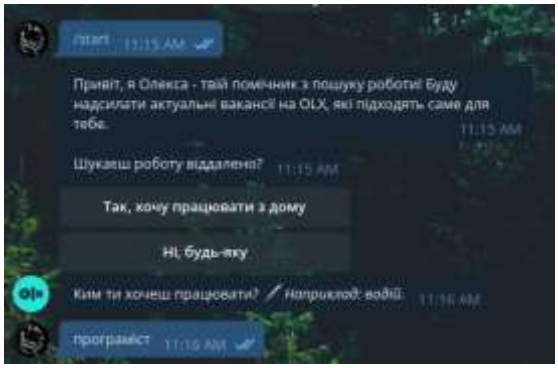


Рисунок 1.3 – Робота telegram-бота «@olx_robota_bot»

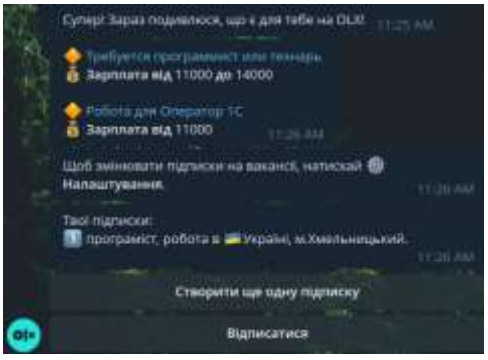


Рисунок 1.4 – Робота telegram-бота «@olx_robota_bot»

На основі отриманих даних складено порівняльну таблицю наявного програмного забезпечення та сайтів (таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика наявного ПЗ

Назва	Тип	Можливість реєстрації	Пошук працівників	Пошук роботи	Можливість спілкування на сервісі	Швидкість виконання запиту	Відображення оголошень в інтерфейсі користувача
«Work. ua»	сайт	Так	Так	Так	Так	середня	Так
«olx rabota.ua»	сайт	Так	Так	Так	Так	середня	Так
«@olx_ robota_bot»	бот	Ні	Ні	Так	Ні	низька	Ні

Проведений аналіз наявного програмного забезпечення показав, що в telegram-ботів відсутня велика кількість функціональних можливостей, які прискорять та спростять процес пошуку працівників/роботи. Провівши аналіз ПЗ, можна виділити основні функціональні можливості для розроблюваного ПЗ:

– можливість публікації оголошень про пошук працівників/роботи;

– сучасний, зручний та інтуїтивно зрозумілий інтерфейс;

– простий набір параметрів для пошуку;

– можливість відображення контактних даних для спілкування в месенджері;

– можливість розбиття на сторінки результату пошуку;

– швидка та зрозуміла відповідь бота.

1.3 Визначення вимог до програмного забезпечення та розробка технічного завдання

Для визначення вимог до розроблюваного програмного забезпечення використаємо UML діаграми. UML – це уніфікована мова візуального моделювання, яка створена, щоб забезпечити стандартний спосіб візуалізації життєвого циклу розроблюваного програмного забезпечення [6]. Ця мова є загальноприйнятим стандартом в графічному описі ПЗ. UML діаграми дають можливість представити систему у такому вигляді, щоб кожен міг швидко зрозуміти основну функціональність розроблюваного ПЗ, а відповідно до можна було без зусиль перевести в програмний код. UML не є методом розробки, в ній не надаються інструкції щодо побудови програмного забезпечення, але ця мова допомагає явно переглядати компоненти системи і полегшує співпрацю з іншими її розробниками. Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість продукту.

Діаграма варіантів використання – це діаграма для представлення взаємодії користувача (актора) із системою, яка показує взаємозв`язок між користувачем та різними випадками використання модулів ПЗ, в яких він бере участь [7]. Також ця діаграма ідентифікує різні типи користувачів системи та різні випадки використання для них.

Складемо описи користувачів системи та необхідних функціональних можливосте для них. У таблиці 1.2 подано опис акторів програмної системи, а в таблиці 1.3 наведені варіанти використання.

Таблиця 1.2 – Опис акторів програмної системи

Актор	Короткий опис
1	2
Незареєстрований користувач (гість)	Має можливість пройти реєстрацію

Кінець таблиці 1.2

1	2
Зареєстрований користувач	Має можливість виступати в ролі роботодавця, підбирати працівників, створювати, редагувати та видаляти свої оголошення, шукати роботу, створювати, редагувати на видаляти свої резюме, також має можливість редагувати обліковий запис.
Модератор	Виконує перевірку на коректність користувацьких публі- кацій, а саме приймає або відхиляє резюме/вакансії з поясненням причини відхилення

Таблиця 1.3 – Опис варіантів використання програмної системи

Актор	Найменування ВВ	Опис ВВ
1	2	3
Незареєстрований користувач	Реєстрація в системі	Користувач може зареєструватися в системі
Зареєстрований користувач	Керування обліковим записом	Користувач може заповнити обліковий запис заново
	Авторизація в системі	Користувач автоматично розпі– знається системою за допомогою його особистого ідентифікатора
	Публікація оголошень та керування ними	Користувач може створювати оголошення для пошуку праці– вників/роботи, редагувати та видаляти, якщо вони ще не підтверджені, також він може відображати список претендентів на вакансію
	Додатково	Користувач може відобразити свій обліковий запис та заповнити його заново, також він має можливість відобразити текст з поясненням, як працює бот
Модератор	Модерація користувацького контенту	Модератор може виконувати модерацію користувацьких ого– лошень, редагувати та видаляти їх

Відповідно до описаних акторів та варіантів використання, побудуємо діаграму варіантів використання (рисунок 1.5):

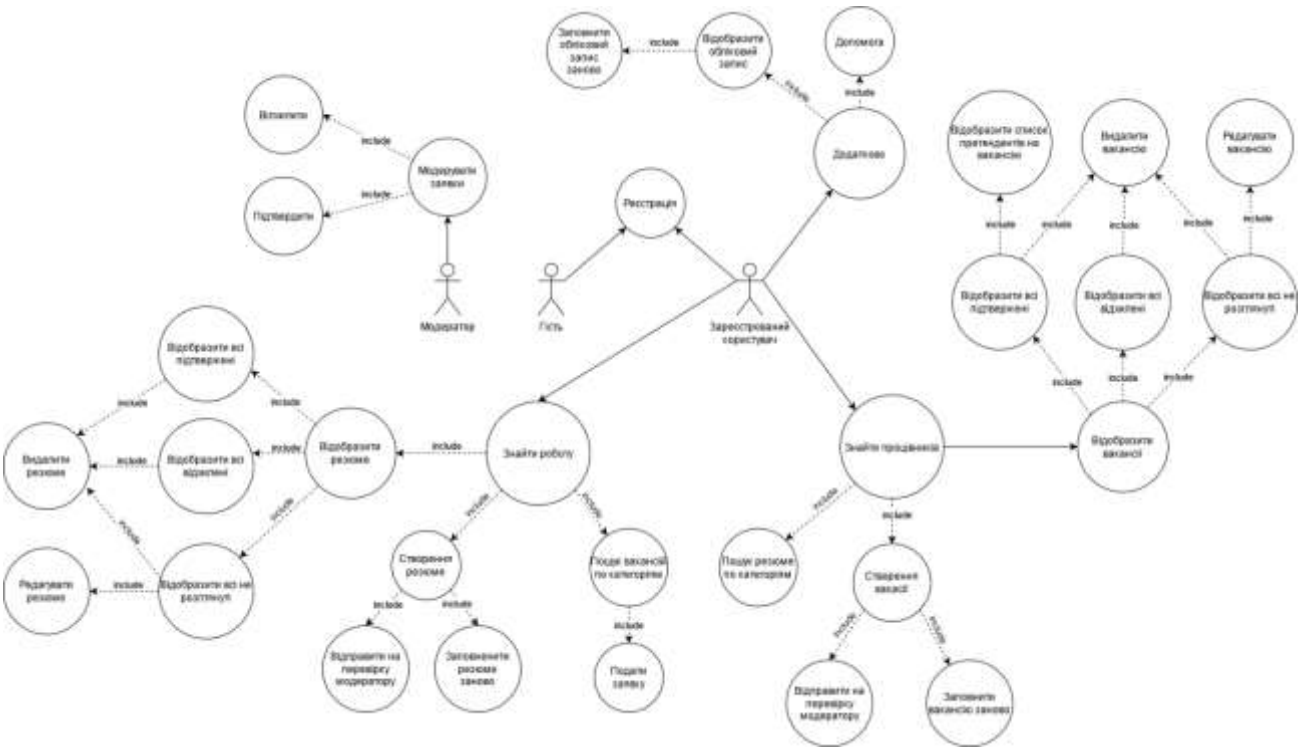


Рисунок 1.5 – Діаграма варіантів використання Telegram бота для пошуку працівників та підбору персоналу

На основі проведеного аналізу вимог до ПЗ було розроблене технічне завдання, яке подано у додатку А.

Отже, було проведено аналіз предметної області та встановлено, що Telegram боти – популярні сервіси, але недостатньо розвинені в Україні, що відіграють значну роль у сучасному житті людини, дають хороші перспективи для швидкого розвитку бізнесу та суттєво зменшують час на рутинні справи. Усі вони повинні бути продуктивними, зручними та легкими у користуванні.

Також було проведено аналіз існуючого програмного забезпечення предметної області, в результаті якого визначено, що існує багато подібних програмних систем і подібний бот, усі вони відрізняються своїм зовнішнім виглядом, типом, функціональними можливостями та зручністю у користуванні. Більшість сайтів мають вже більшість необхідних функціональних можливостей, але Telegram боти відстають в цьому плані. В результаті були визначені функціональні та нефункціональні вимоги до програмного забезпечення та встановлено і описано основні варіанти використання системи.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕНЯ

2.1 Аналіз клієнт-серверної архітектури та протоколів взаємодії сучасних веб-додатків

Відповідно до поставленої задачі, взаємодія з розроблюваним ПЗ буде реалізована ботом у месенджері Telegram через його інтерфейс. Для того, щоб зв’язати користувача бота, та ПЗ, яке відповідає за взаємодію з ботом, буде використано веб API, використовуючи яке ПЗ зможе отримувати нові повідомлення від користувача та надсилати відповідь. Оскільки Telegram бот є інтерфейсом користувача, а розроблюване ПЗ повинно обробляти повідомлення від бота, то така система є реалізацією клієнт-серверної архітектури.

Клієнт-серверна (дволанкова) архітектура – це один з видів архітектури ПЗ, який передбачає створення застосунків, які взаємодіють та обмінюються даними між собою через мережу. В основі такої архітектури лежать два основних компоненти: клієнт та сервер. Зазвичай клієнтом виступає комп’ютерний пристрій, який може надсилати запити через локальну мережу або мережу Інтернет на віддалений комп’ютер, тобто сервер, для отримання певних даних, надсилання запиту виконує спеціальне ПЗ [8]. Сервером виступає комп’ютер або обладнання, на якому встановлене ПЗ, для отримання та обробки вхідних даних та повернення у відповідь результату виконаних дій. На рисунку 2.1 зображена модель клієнт-серверної архітектури.

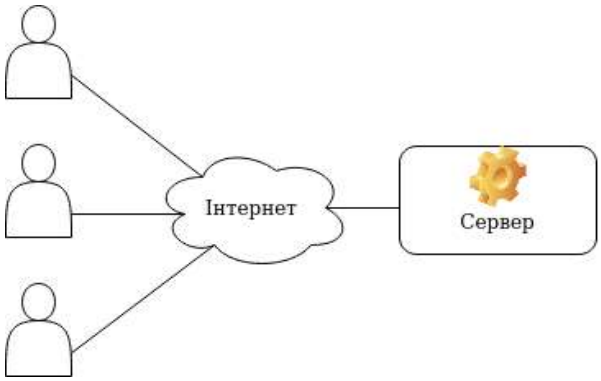


Рисунок 2.1 – Модель клієнт-сервер

Оскільки клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнта і сервера, існує дві моделі клієнта:

– модель тонкого клієнта, яка характеризується зосередженням на стороні клієнта тільки функції рівня представлення, а логіка застосунку та управління даними відбувається на стороні сервера.

– модель товстого клієнта, яка характеризується зосередженням інтерфейсу користувача та логіки обробки інформації на стороні клієнта, а сервер тільки керує даними [9].

Отже, в нашому випадку ми будемо використовувати клієнт-серверну архітектуру та модель тонкого клієнта, тому що в нас комунікація між клієнтом і ПЗ буде відбуватись через інтерфейс месенджера Telegram, а основна логіка обробки вхідних і вихідних даних буде зосереджена на розроблюваному ПЗ.

Для того, щоб клієнт і сервер могли коректно обмінюватись даними, розпізнавати та формувати вихідні дані, буде використано протокол передачі даних (протокол взаємодії). На сьогодні найбільш поширеними актуальними протоколами є HTTP та WebSocket.

WebSocket – це протокол, який призначений для комунікації між клієнтом та сервером через двонаправлений канал зв’язку, який використовує один TCP-сокет. За рахунок того, що канал між клієнтом і сервером відкритий постійно, з’єднання відбувається у режимі реального часу.

HTTP – це протокол, який використовується для передачі даних у вигляді гіпертексту. Для кожного нового клієнта створюється сеанс зв’язку з сервером. Основним призначенням протоколу є передача вебсторінок, але на сьогодні його успішно використовують для передачі файлів і інформації в іншому вигляді.

Кожен запит протоколу HTTP містить у собі заголовки (headers), рядок запиту та необов’язкове тіло запиту (body). Заголовки містять текстову інформацію, що зберігається у вигляді «ключ-значення». Ці заголовки передають основну інформацію, наприклад, який браузер використовує клієнт, тип запиту, які дані запитуються. Приклад заголовків наведено на рисунку 2.2.

Рисунок 2.2 – Приклад заголовків HTTP-запиту

Рядок запиту містить в собі ідентифікатор ресурсу, а також HTTP-метод, який вказує, які дії потрібно виконати з даним ресурсом. Тіло запиту може містити всередині все, що завгодно, наприклад: дані запиту або опис проблеми, якщо запит не виконався успішно. Для того, щоб покращити розуміння системи, а також спростити процес долучання нових розробників у процес розробки, було створено HTTP методи, розглянемо основні з них:

– GET – метод для отримання вмісту вказаного ресурсу;

– POST – метод для передачі даних, який додає новий ресурс;

– PUT – метод для оновлення ресурсу, або завантаження частини ресурсу на сервер;

– DELETE – метод, який видаляє ресурс;

Webhook (також названий зворотнім викликом або HTTP push API) – це концепція API, яка дозволяє надавати програмам інформацію в режимі реального часу. Ця концепція доставляє дані на сервер програми, як тільки відбувається якась дія на стороні клієнта, тобто ви отримуєте дані негайно. На відміну від типових API, де потрібно постійно надсилати запити на отримання нових даних, щоб з’єднання відбулось у реальному часі [10].

Взаємодія з користувачем буде через інтерфейс бота, відповідно ПЗ не потребує взаємодії у режимі реального часу, тому для нашого ПЗ кращим варіантом, буде використати концепцію Webhook для того, щоб в найкоротші терміни отримати та у відповідь надіслати дані користувачу.

Для того, щоб покращити та полегшити створення серверних API,

створюються різні архітектурні підходи. Найпопулярніший з них – REST. Даний підхід до розробки використовується для забезпечення стандартів між комп'ютерними системами в Інтернеті, що спрощує взаємодію систем одина з одною [11]. Системи, які повністю дотримується правил REST, часто називають системами RESTful. REST-сумісні системи, характеризуються тим, що вони не мають стану і розділяють проблеми клієнта і сервера, тобто обробка запитів при такій архітектурі має виконуватись у відповідності до того, яким HTTP методом були надіслані дані. Даний підхід містить п'ять обов'язкових архітектурних правил, шосте є не обов'язковими. Розглянемо правило поділу клієнта і сервера, відсутності стану та шарування компонентів.

Поділ клієнта і сервера – розподіляє відповідальність між компонентами для того, щоб реалізація клієнта і сервера могла виконуватись незалежно одина від одної. Це означає, що код на стороні клієнта можна змінити в будь-який час, не впливаючи на роботу сервера, а код на стороні сервера можна змінити, не впливаючи на роботу клієнта. Поки кожна сторона знає, який формат повідомлень відправляти іншій, вони можуть бути модульними й окремими;

Відсутність стану – це правило позначає, що серверу не потрібно знати про те, в якому стані знаходиться клієнт, і навпаки. Таким чином, і сервер, і клієнт можуть зрозуміти будь-яке отримане повідомлення, не знаючи попередніх повідомлень, тобто це обмеження засноване на використанні ресурсів, а не конкретних команд.

Шарування компонентів системи – вказує на те, що вся система повинна бути поділена на шари. Кожен рівень має певну роль і відповідальність в архітектурі. Наприклад, рівень представлення відповідає за обробку всього призначеного для користувача інтерфейсу і взаємодії з браузером, тоді як бізнес-рівень відповідає за виконання певних бізнес-правил, пов'язаних із запитом. Шари ізольовані один від одного, та не мають інформації про внутрішню роботу інших шарів. Кожен рівень взаємодіє тільки з нижнім шаром. Це означає, що рівень представлення взаємодіє тільки з бізнес-рівнем і не повинен

безпосередньо взаємодіяти з рівнем збереженості. Таким чином, клієнт використовує сервер, як так званий «чорний ящик». Він надсилає запит лише до першого шару сервера, не знаючи скільки шарів буде задіяно для обробки запиту.

2.2 Аналіз та вибір архітектури серверної частини ПЗ

Для реалізації серверної архітектури, виділяють два найбільш поширених архітектурних стилі: монолітний та мікросервісний. Розглянемо кожен з них та проведемо аналіз.

Монолітний архітектурний стиль позиціонує себе як класичний метод розробки, в якому всі функції проекту зібрані в єдиній кодовій базі [12]. Масштабування такого проекту є важким, бо якщо потрібні будь-які оновлення коду, то ці оновлення не можна розмістити паралельно. Розробник повинен використовувати ту ж базу коду, внести необхідні зміни і повторно розгорнути оновлений код у вигляді jar файлу. Таким чином, навіть якщо потрібна одна незначна зміна, вся кодова база зачіпається, зупиняється і повторно розгортається. Проектування цього стилю будується на чотирьох різних рівнях. На рисунку 2.3 зображено схематичний вигляд монолітної архітектури.

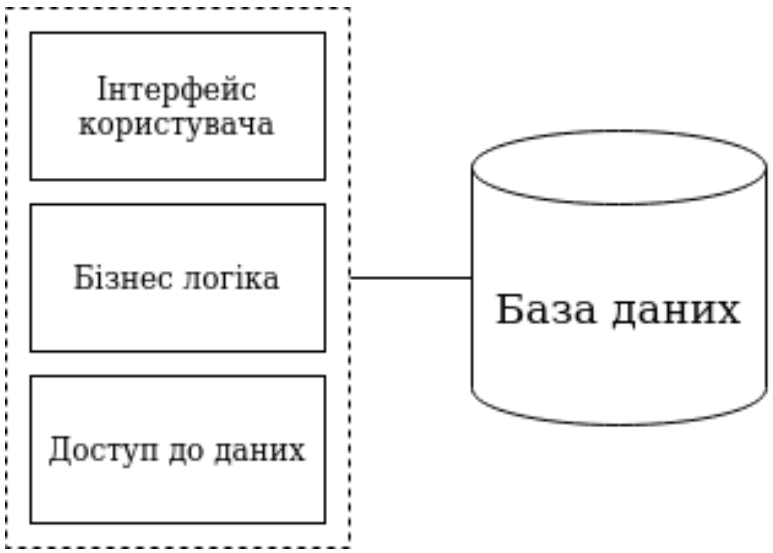


Рисунок 2.3 – Монолітна архітектура

Рівень бізнес-логіки – створює правила та спосіб організації моделей проекту, які описують сутності предметної області відповідно до діяльності в реальному світі, яку підтримує система. Зазвичай цей рівень створює базову ієрархічну структуру класів та їх методів. Модель повинна описувати сутності і відповідати їх реальнії моделі. Тобто коротко можна описати, що бізнес-логіка – це реалізація предметної області в розроблюваній системі, до якої належать моделі та їх поведінка з реального життя [13].

Рівень доступу до сховища даних – цей рівень відповідає за спрощений доступ до даних, які зберігаються у базі даних та реалізує основні операції з нею [13]. Є чотири основні операції, які коротко вміщуються в абревіатурі CRUD. Кожна буква з цієї абревіатури відповідає певній операції над даними:

1) C (Create) – додавання нових даних;

2) R (Read) – отримання інформації;

3) U (Update) – оновлення даних;

4) D (Delete) – видалення даних.

Рівень представлення – відповідає за спрощене та коректне графічне відображення даних через інтерфейс користувача. Скорочено цей рівень прийнято називати UI, також до цього рівня відносять ще UX [13].

Основний плюс монолітної архітектури – це те, що таку систему легше розробляти. Оскільки відображення, обробка та збереження даних знаходяться в одній кодовій базі, таку систему легше створити. Завдяки такому підходу мережеві затримки зводяться до нуля, а відстежувати помилки можна значно швидше. Завдяки цьому, розробка за монолітною архітектурою займає менше часу. Також тестування такої програми стає досить простим завданням, оскільки логіка зосереджена в одному місці. Варто відзначити, що середовища для розробки програм початково проектувалися для розробки ПЗ у вигляді однієї кодової бази, через що на сьогодні розробнику легше розробляти програмне забезпечення в таких IDE, тому що всі інструменти під рукою, і їх можна використовувати максимально ефективно.

Але все ж таки, така на перший погляд прекрасна архітектура має свої мінуси. Головний мінус – це проблема масштабування такого проекту, оскільки при збільшенні одного компоненту ви повинні збільшувати обсяг всієї програми, що з часом переросте в груду незрозумілого та некерованого коду та значно сповільнить час розробки нових компонентів. При отриманні помилки, яка зачіпає одну частину кодової бази, автоматично зачіпаються всі, що повністю зупинить роботу програми і може погано вплинути на репутацію та економіку проекту. Також існують перешкоди для вдосконалення технологій на проекті. Якщо ви хочете використовувати нову технологію або фреймворк, вам доведеться переписувати весь додаток. Отже, таку архітектуру варто використовувати, коли обмаль часу на розробку або якщо додаток простий та не містить великої кількості бізнес-логіки.

Мікросервісна архітектура характеризується розробкою системи у вигляді окремих компонентів (сервісів) невеликого розміру. Кожен з таких сервісів має визначену конкретну задачу, та спілкується з іншими сервісами для передачі даних або вирішення інших задач [14]. Таким чином, кожен із сервісів є самостійним модулем, який мало зв’язаний з іншими, та у випадку необхідності може бути замінений іншим сервісом. Зазвичай, спілкування між сервісами відбувається за допомогою HTTP-запитів в середині їхньої мережі. На рисунку 2.4 зображено схематичний вигляд мікросервісної архітектури.

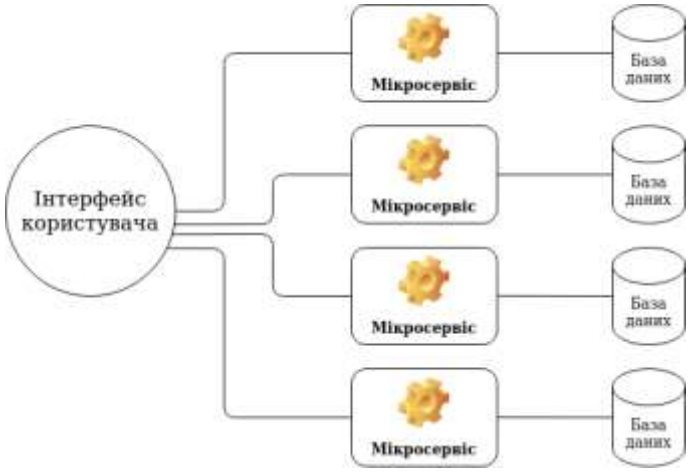


Рисунок 2.4 – Мікросервісна архітектура

Головний плюс такої архітектури – це розробка та оновлення окремих модулів може вестись паралельно без зупинення всієї програмної структури. Кожен з таких сервісів можуть розробляти різні розробники абсолютно різними мовами програмування і через те, що зв'язок між сервісами відбувається через HTTP запити, не важливо, на якій мові і як реалізований всередині сервіс, результат взаємодії буде один і той самий. Виникнення помилок у мікросервісах нагадує гру у морський бій, один з мікросервісів (кораблів) не працює (підбитий), але інші сервіси (частини корабля) продовжують працювати коректно.

При такій розробці у проекті легше реалізовувати нові технології, оскільки старі технології будуть стосуватись лише окремих сервісів. Варто звернути увагу на те, що кожен мікросервіс виконує якусь задачу і може бути використаний в абсолютно іншому проекті. Також при масштабуванні, необхідно оновлювати, редагувати та запускати лише ті сервіси, які потрібні, на відміну від моноліту, де необхідно зупиняти, вносити зміни та запускати всю програму, що може негативно вплинути на використання ресурсів.

Такий підхід до створення сервісів приваблює, але в нього теж є свої недоліки. Основний недолік – це проблеми зі зв'язком між самими сервісами. Так як кожен функціональний елемент ізольований, потрібна особлива ретельність при побудові між ними грамотної комунікації, адже їм в будь-якому випадку доведеться обмінюватися запитами і відповідями один з одним. Зрозуміло, що зі збільшенням кількості сервісів складність в побудові їх повідомлення буде рости, та можуть виникати затримки в отриманні відповідей. Також таку структуру з мікросервісів важко тестувати. На відміну від монолітної архітектури, де тестування відбувається в одній кодовій базі, до всіх мікрісервісів потрібен свій підхід. Також для тестування такої системи потрібна багато ресурсів для запуску всі сервісів, що може збільшити час на тестування. З цієї ж причини, для такої системи важко написати тести, які будуть перевіряти роботу сервісів одночасно.

Порівнюючи ці стилі, їх переваги та недоліки, було вирішено використати монолітну архітектуру, оскільки з огляду на терміни розробки ПЗ та те, що

взаємодія буде відбуватись з месенджером Telegram через Webhook. Обробка інформації буде позиціонуватись як одна задача, тому розділення програмного забезпечення на мікросервіси є недоцільним.

2.3 Аналіз та вибір типу бази даних

Більшість сучасних ботів зберігає інформацію про користувача, та іншу додаткову інформацію в особистому сховищі. Для її коректного та надійного зберігання використовуються бази даних. Основними типами БД, які активно використовуються на сьогодні при розробці ПЗ, є реляційні та нереляційні БД.

Реляційні бази даних характеризуються набором зв'язних таблиць, кожна з яких містить інформацію про об'єкти різних типів. На сьогодні найпопулярніші реляційні БД – MySql, Postgres та Oracle. Дані структуруються двовимірно, тобто кожна таблиця складається з стовпців та рядків [15]. Як правило, кожна таблиця є одиним типом об'єкту (наприклад, користувач або працівник). Рядок таблиці або кортеж є окремим записом, екземпляром об'єкта, який містить елементи одного типу в межах одного стовпця. Стовпець або атрибут таблиці, повинен мати індивідуальну назву та зберігати значення одного типу, в межах однієї таблиці, які властиві конкретному об'єкту. Кожен рядок в таблиці має свій унікальний ідентифікатор – ключ. Рядки в таблиці можна зв'язати з рядками в інших таблицях, додавши стовпець для унікального ключа пов'язаного рядка, такі стовпці називають зовнішнім ключем. Кожна таблиця повинна містити стовпець або сукупність стовпців, які унікально ідентифікують кожен запис в БД, такі стовпці називають первинним ключем.

Одна з основних переваг реляційних БД полягає у простоті та доступності для розуміння користувачем, оскільки дані оперуються в одній інформаційній конструкції – таблиці. Також одна з головних переваг такої моделі БД – це транзакції, які виконують сукупність операцій як одне ціле. Всі операції запису в транзакції слідують одному правилу: «все або нічого», тобто якщо транзакція

завершується успішно, то всі записи вступають в силу, або у випадку помилки база даних не зберігає ніяких записів, які були подані в транзакції. Підтримка транзакцій в базі даних характеризується ACID. Кожна буква з цього скорочення відповідає певним вимогам до бази даних:

1) A (Atomicity) – транзакція не може бути виконана не повністю;
2) C (Consistency) – до початку транзакції та після, система повинна перебувати в несуперечливому стані;
3) I (Isolation) – доки транзакція не завершилась, зміни не будуть доступні;
4) D (Durability) – БД гарантує, що у випадку збоїв під час виконання транзакції дані гарантовано будуть збережені після відновлення системи.

Така модель також не може бути без недоліків. Один з головних недоліків – це те, що БД займає багато зовнішньої пам’яті, яку прийдеться збільшувати пропорційно до напливу користувачів. Оскільки при проектуванні з’являється множина таблиць, це призводить до важкості розуміння структури даних та суттєво зменшує швидкість доступу до даних.

Нереляційні бази даних, які ще називають NoSQL – така БД забезпечує механізм зберігання та пошуку даних, яке моделюється іншими способами, крім табличних відношень, які притаманні реляційним БД [16]. Найпопулярніші приклади таких БД – MongoDB, Apache CouchDB. Зазвичай, дані в такій моделі зберігаються у вигляді документів. Кожен запис в базі даних – це окремий документ з набором полів, які можуть містити різні за типом дані. Документи інкапсулюють і кодують дані в деяких стандартних форматах або кодуваннях. Використовувані кодування включають JSON, XML, YAML, а також двійкову форму для подання простих або складних структур даних – BSON. Документи адресуються в базі даних за допомогою унікального ключа, який представляє цей документ. Такий підхід до збереження даних є досить зручним та гнучким. За рахунок простоти та зручності час на проектування БД зводиться практично до нуля, оскільки відсутня необхідність проектування таблиць та визначення типів даних для полів, замість цього БД просто зберігає вказаний документ. Також така

база даних працює в рази швидше від реляційної за рахунок парсингу та трасиляції SQL запитів, об’єднання таблиць та роботи оптимізатора. На рисунку 2.5. зображено порівняння швидкості реляційної MySQL та нереляційної MongoDB бази даних [17].

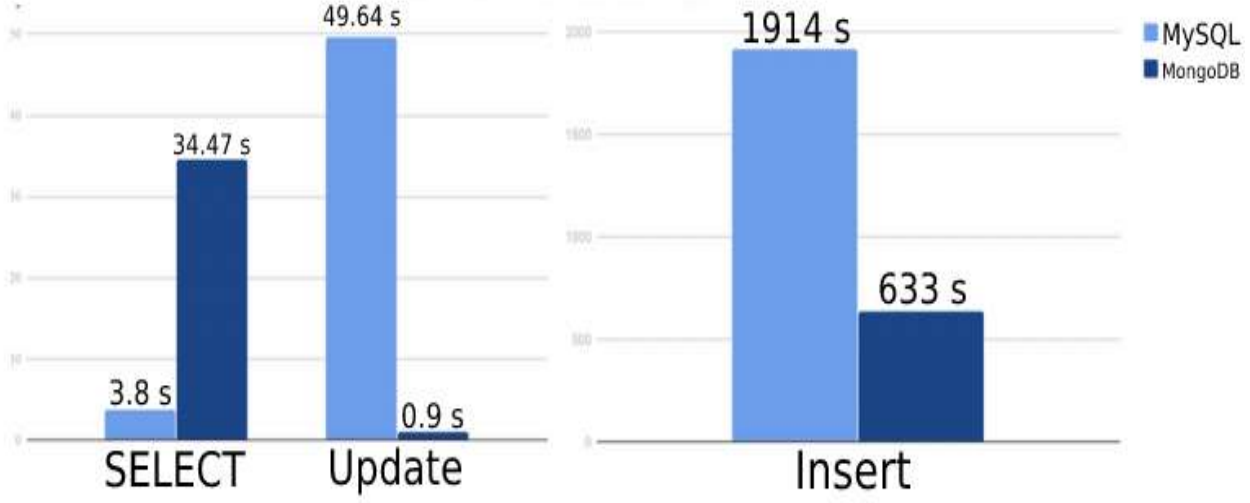


Рисунок 2.5 – Порівняння швидкості обробки запитів select, update, insert реляційної та нереляційної БД

Недоліком таких баз даних є те, що за рахунок гнучкості та високої продуктивності втрачається стовідсоткова цілісність даних. Також такий тип БД повноцінно не підтримує транзакції, атомарні оновлення можна виконувати на рівні одного документа. Транзакцію можна виконувати як семантичну над MongoDB, але це потребує досить складних реалізацій. Також на даний момент база не має вбудованого механізму шифрування даних.

Порівнявши та зваживши всі переваги та недоліки вказаних типів БД, було прийняте рішення обрати для реалізації ПЗ нереляційну бази даних, оскільки для неї не потрібно детально проектувати структури даних, таблиці та їх зв’язки, також втрата даних не стане критичною для роботи системи.

2.4 Проектування моделі бази даних

В системі буде одна база даних під назвою jobalook. Вона буде використовуватись для зберігання усіх вхідних та вихідних даних. Як було вказано вище, основними структурними елементами даних в нереляційних базах даних є документи у вигляді JSON, XML, YAML та BSON. Відповідно до діаграми варіантів використання (рисунок 1.5) можна побачити, що для коректного функціонування програми в базі даних потрібно зберігати користувача, створені резюме та створені вакансії. Відповідно до потреб зберігання в БД буде три колекції під назвами «User», «Resume», «Job», розглянемо їх.

Колекція «User» призначена для зберігання усіх даних про користувача та містить такі поля:

– userName – ім’я в месенджері Telegram;

– userId – унікальний ідентифікатор, який дає Telegram;

– userChatId – унікальний ідентифікатор чату з користувачем, який дає Telegram;

– name – ім’я;

– age – вік;

– gender – стать;

– email – електронна пошта;

– state – конкретний статус бота;

– lastState – попередній статус бота;

– isModerator – є користувач модератором чи ні.

Колекція «Job» призначена для зберігання даних про вакансію, її поля:

– uniqueId – унікальний ідентифікатор ;

– title – заголовок;

– description – опис вакансії;

– salary – об’єкт зарплати, який складається з двох полів from і to;

– userId – ідентифікатор користувача, який створив вакансію;

– category – категорія, до якої відноситься вакансія;

– employment – тип зайнятості;

– experiance – досвід, який потрібен;

– workPlace – місце розташування роботи;

– requested – масив користувачів, які подали заявку;

– approved – показує, чи підтверджена вакансія;

– declined – показує, чи відхилена вакансія;

– declineExplane – причина відхилення;

– checked – показу,є чи була вакансія перевірена модератором.

Колекція «Resume» призначена для зберігання даних про резюме та містить такі поля:

– uniqueId – унікальний ідентифікатор користувача;

– userId – ідентифікатор користувача, який створив резюме;

– profileInfo – містить інформацію про користувача та додаткові дані;

– skills – опис вмінь користувача;

– category – категорія, до якої відноситься резюме;

– employmentHistory – інформація про попередній досвід роботи користувача;

– education – інформація про освіту користувача;

– approved – показує, чи підтверджена вакансія;

– declined – показує, чи відхилена вакансія;

– declineExplane – причина відхилення;

– checked – показує, чи була вакансія перевірена модератором.

Розглянуті документи зберігатимуть велику кількість полів, які необхідні для коректної роботи розроблюваного ПЗ. Для того, щоб не прив’язувати створені резюме або вакансії до документа «User», було вирішено в кожен документ додати унікальний ідентифікатор (userId) для ідентифікації створених користувачем резюме/вакансій та самого користувача. Як видно із документів,

«Job» та «Resume» мають кілька одинакових полів, а саме: «approved», «declinde», «declineExplane», «checked». Всі ці поля дають можливість розділити та спростити вибір для відображення резюме або вакансій за певними категоріями поточних статусів, у яких вони знаходяться.

Всі облікові дані користувача, такі як електронна адреса, вік, стать та інші, будуть зберігатись у документі «User». Також у ньому будуть зберігатись поточний стан бота та попередній. Ці стани слугують позначенням того, в якому стані знаходиться бот, наприклад якщо стан буде «FIND_JOB_TO_DO», а попередній «ASK_TODO», то це буде означати, що користувач знаходиться в меню «Знайти роботу» та вибирає подальшу дію і має можливість повернутись назад до головного меню за використовуючи попередній стан. Користувач може одночасно шукати роботу та працівників і виконувати функції модератора, тобто модерувати контент, для визначення того, чи є користувач модератором, використовуються поле «isModerator». На рисунку 2.6 зображено модель бази даних та дані в форматі JSON в колекції «User».

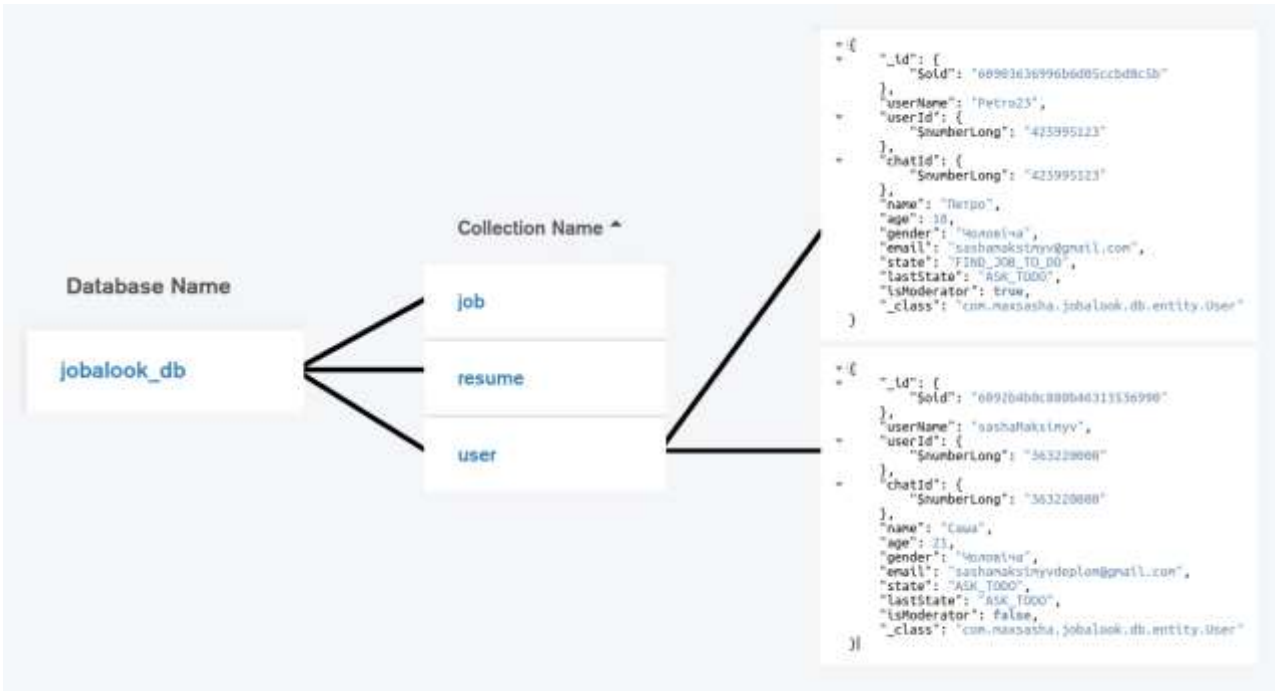


Рисунок 2.6 – Відображення моделі бази даних та даних в форматі JSON

2.5 Детальне проектування серверної частини та розбиття її на модулі

Процес отримання оновлень від користувача ділиться на два типи: ми можемо вручну надсилати запити на отримання оновлень з сервера через надсилання HTTP запитів на Telegram API або за налаштуванням Webhook.

Перший метод називають тривалим опитуванням (Long poling), використовуючи такий спосіб отримання оновлень, сервер запитує інформацію у сервера Telegram точно так само, як при звичайному опитуванні, але сервер може не відповісти негайно. Якщо при надходженні запиту на отримання оновлень сервер не має нової інформації для клієнта, замість надсилання порожньої відповіді, сервер тримає запит відкритим і чекає, коли інформація про відповідь стане доступною. Отримавши нову інформацію, сервер негайно надсилає відповідь клієнту. Цей метод хороший тим, що він гранично простий, але поганий тим, що потрібно самостійно постійно опитувати сервер на предмет оновлень.

Другий метод відрізняється від першого тим, що Telegram дає можливість встановити URL, на який при отриманні оновлень від користувача сам Telegram буде надсилати оновлення на ваш сервер, тобто цей спосіб базується на спрацюванні тригеру та дозволяє не думати про те, як отримати оновлення, а зосередитись на їх обробці та значно зменшити витрату ресурсів.

Серверна частина програмної системи буде побудована на основі шаблону «Repository-Service-Controller». Цей шаблон дозволить розділити бізнес-логіку програми на 3 рівні. Перший рівень це рівень представлення (Controller), він відповідає за отримання даних, перетворення їх у коректний вигляд для подальшої обробки нижніми рівнями та відправлення результату виконаних дій. Наступним рівнем виступає сервіс (Service), який є прошарком між рівнем представлення та рівнем роботи з базою даних. На цьому рівні зосереджена основна логіка взаємодії клієнта з функціональними можливостями програми. Останній рівень, який вважається найнижчим – це рівень роботи з базою даних (Repository). Рівень репозиторія напряму працює із сховищем даних, на цьому

рівні зосереджені вся логіка запитів до бази даних. Важливо наголосити, що кожен репозиторій працює тільки з одним класом моделі [18]. Схема роботи шаблону зображена на рисунку 2.7.

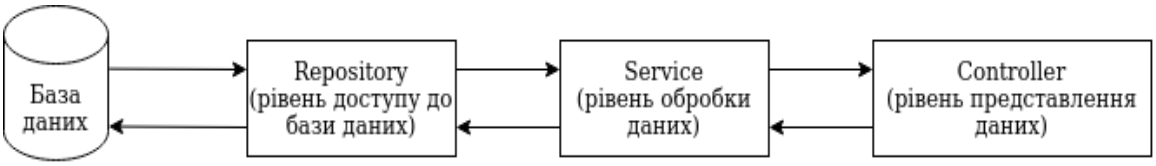


Рисунок 2.7 – Схема шаблону проектування

Розглянемо взаємодію користувача з ПЗ, а саме: реєстрацію користувача, пошук роботи за категоріям, створення вакансії, редагування вакансії, модерацію публікацій. Для відображення взаємодії буде використано діаграми послідовності, які покажуть, у якій послідовності користувач взаємодіє з ПЗ. На рисунку 2.8. зображена діаграма послідовності реєстрації користувача в системі.

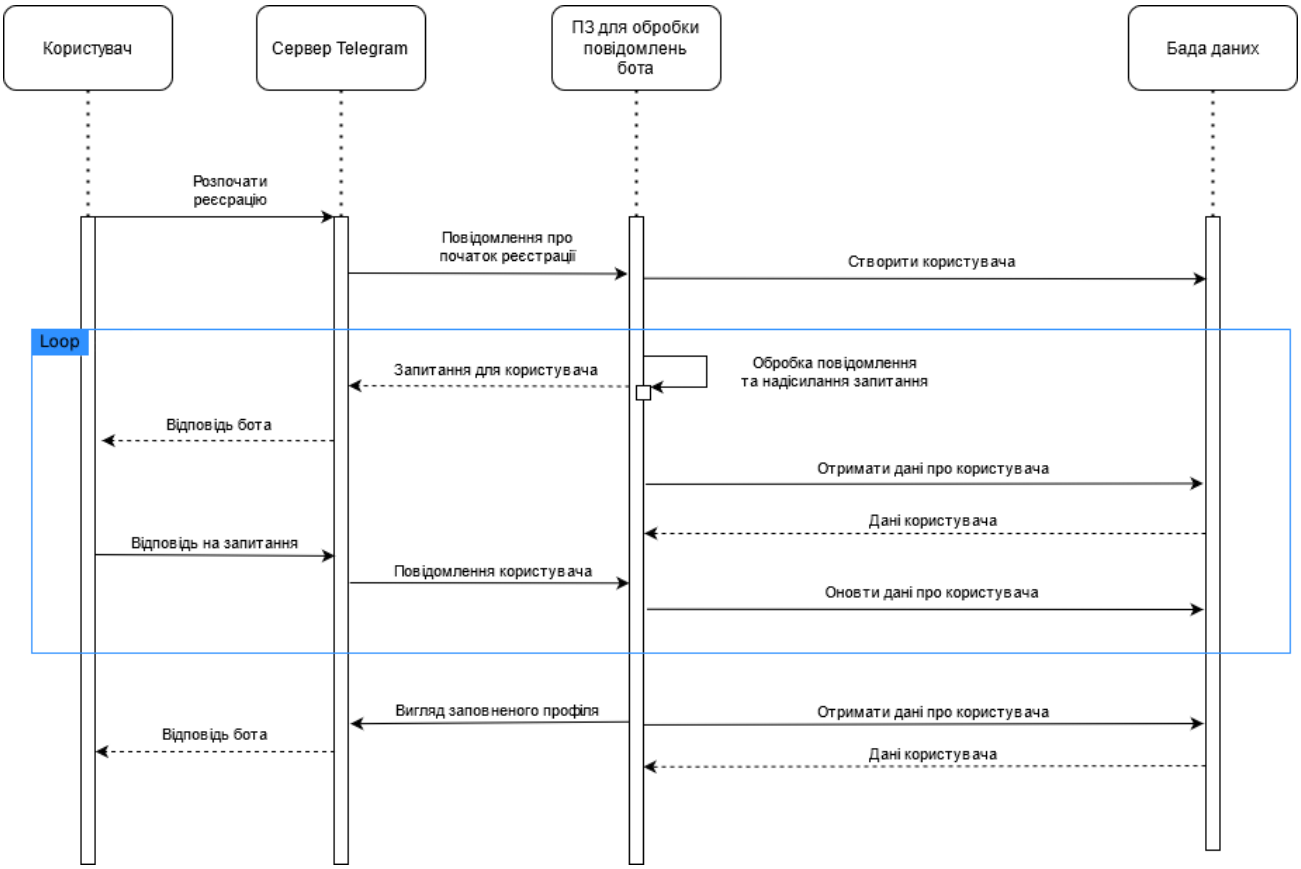


Рисунок 2.8 – Діаграма послідовності реєстрації користувача

Коли користувач уперше розпочинає розмову з ботом, тобто надсилає команду «/start», месенджер Telegram обробляє повідомлення користувача та надсилає повідомлення на наше ПЗ по Webhook, в якому містяться дані про користувача (ідентифікатор користувача та унікальний ідентифікатор чату). Відповідно до отриманого повідомлення ПЗ надсилає привітальне повідомлення і пропонує розпочати реєстрацію в системі. Коли користувач підтвердив початок реєстрації, ПЗ створює нового користувача в базі даних та почергово надсилає запитання для заповнення профілю користувача. Після завершення реєстрації(відповіді користувача на останнє запитання) ПЗ зберігає профіль в базі даних та надсилає його візуальний вигляд у форматі повідомлення користувачу.

Користувач, який пройшов реєстрацію, має можливість розпочати пошуки роботи/працівників. Оскільки сценарії створення резюме і вакансії схожі, розглянемо тільки один з них – створення резюме (рисунок 2.9).

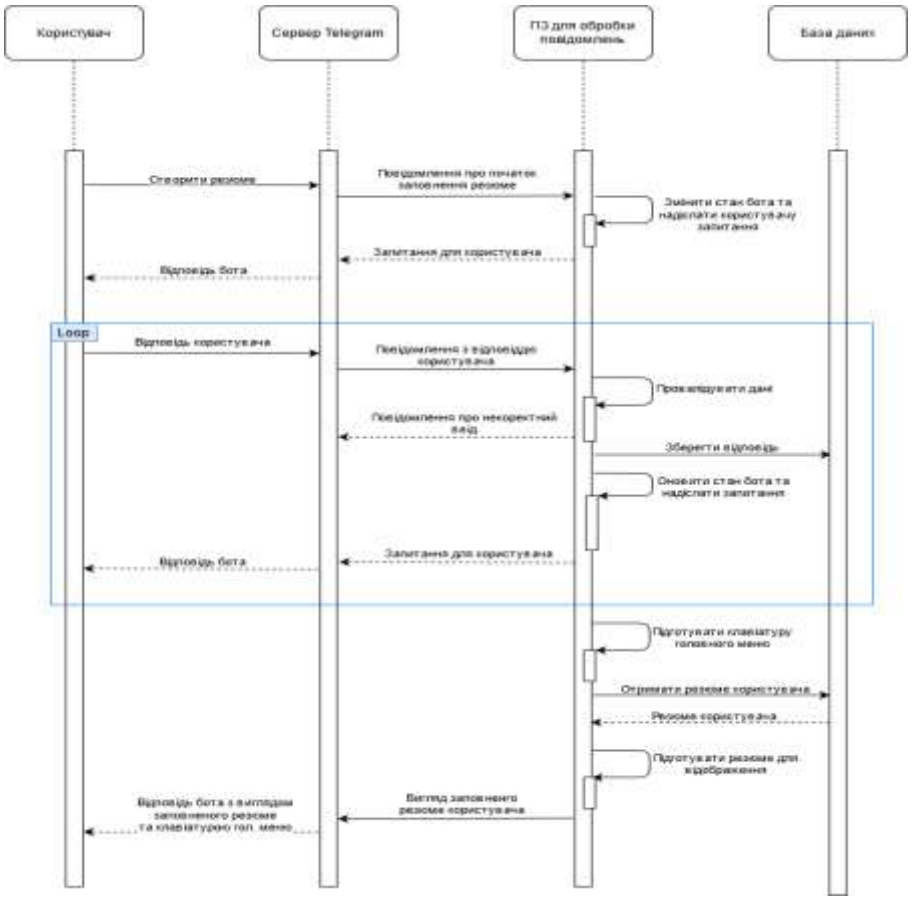


Рисунок 2.9 – Діаграма послідовності створення резюме

Якщо користувач бажає шукати публікації, він може скористатись пошуком. На рисунку 2.10 зображено діаграму послідовності пошуку вакансій.

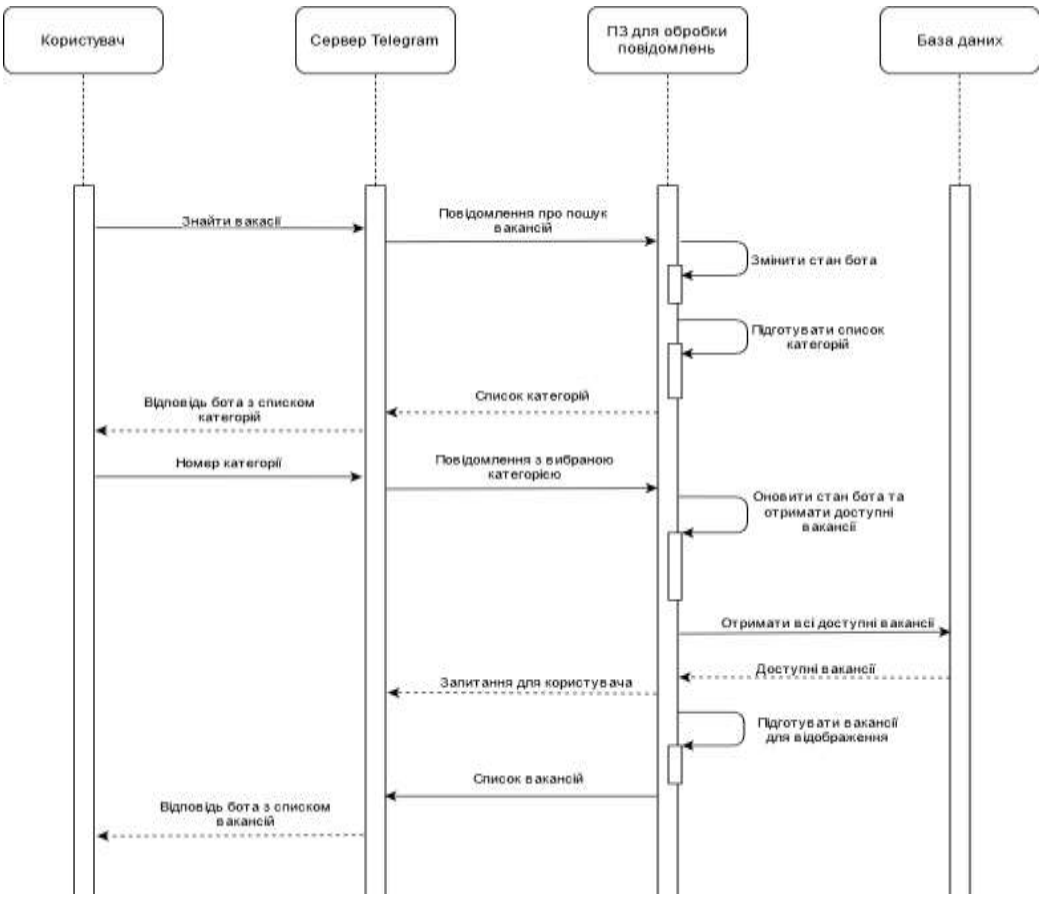


Рисунок 2.10 – Діаграма послідовності пошуку вакансій

Оскільки система відповідатиме шаблону MVC, тобто усі модулі пов’язані між собою та належать до відповідних рівнів монолітної архітектури, відповідати за рівень представлення даних буде модуль під назвою «Controller». Цей модуль має контролер, який відповідає за отримання даних та повернення відповіді, використовуючи Webhook посилання. Вхідні дані, які приходять від користувача, перенаправляються вниз по рівнях та повертаються для відправки користувачу.

Модуль «Bot» – в цьому модулі зберігається основна логіка маніпуляції над станами бота та вхідними і вихідними даними.

Наступним модулем нашої програми являється модуль «Api». Цей модуль належить до рівня обробки даних, тут відбуваються переключення станів бота та

вибір класу обробника відповідно до стану бота.

Основний модуль ПЗ – «Handler». Цей модуль належить до рівня обробки даних. В ньому відбувається основні дії відповідно до стану бота та вхідного повідомлення користувача.

Модуль «Cache» – належить до рівня доступу до даних. Він відповідає за збереження даних, які не потрібно зберігати довговічно, а тільки на певний час, тобто після перезапуску програми або її зупинення дані будуть втрачені.

«DB» – цей модуль належить до рівня роботи з даними і є основним для їх збереження та отримання.

Модуль «Repository» – відповідає за виконання CRUD операції над даними та їх коректне отримання у вигляді об’єктів.

Модуль «Entity» – модуль, який містить моделі, вони відповідають об’єктам, які зберігаються в базі даних та вказують на поля, які будуть зберігатися в БД.

Модуль «Service» – в цьому модулі розташовані класи, які відповідають за допоміжну функціональність таку як: розділення отриманих даних на сторінки, отримання повідомлень, які заздалегідь створені та зберігаються в ресурсах. На рисунку 2.8 зображено взаємодію розглянутих модулів. Діаграми класів для поданих модулів подані на рисунках Б.1 – Б.4 (додаток Б).

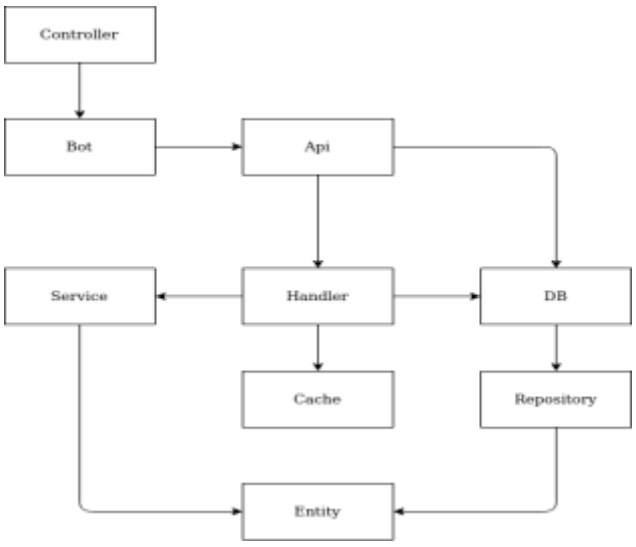


Рисунок 2.8 – Схема взаємодії модулів ПЗ

2.6 Проектування інтерфейсу користувача

Розглянемо приклад інтерфейсу Telegram бота «olx_robota_bot», а саме створення підписки на категорію робіт (рисунок 2.9).

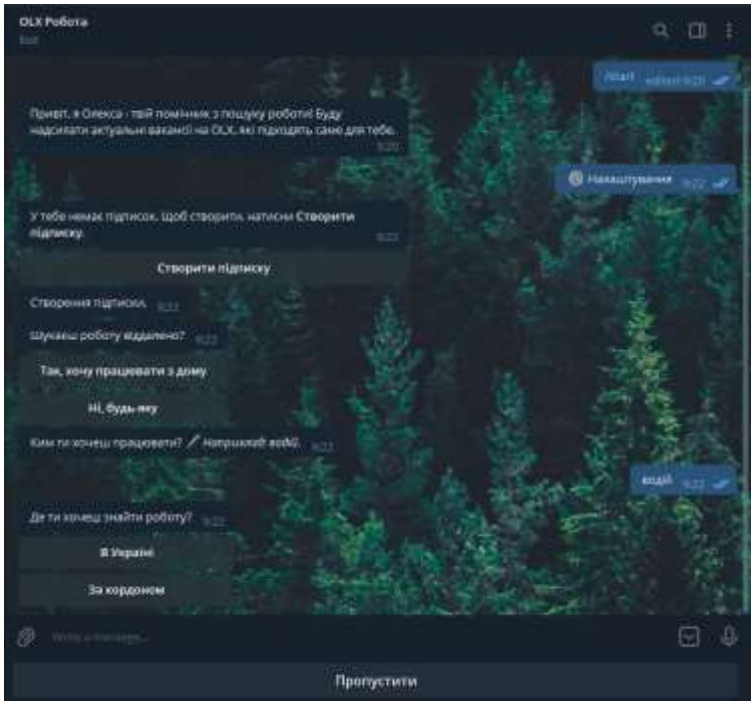


Рисунок 2.9 – Вигляд інтерфейсу бота «olx_robota_bot»

Як бачимо, для заповнення підписки бот отримує команду у вигляді повідомлення «Створення підписки». Взаємодія користувача з ботом відбувається у вигляді текстових повідомлень або кнопок на інтерфейсі. Відрізняються вони тим, що при текстовому вводі бот отримує та перевіряє ввід на коректність, а ввід кнопками дозволяє наперед встановити певні правила відповіді на запитання або команду бота, що не потребує валідації даних, та зменшує ризик введення некоректних даних користувачем. Кнопки бувають двох видів: вбудовані в повідомлення та вбудовані в інтерфейс месенджера. Клавіатура, яка вбудована в повідомлення, зберігається в історії повідомлень користувача, та дозволяє користувачу натискати на них тоді, коли бот не очікує такого вводу користувача, така клавіатура досить зручна, але завдає певних

проблем при розробці ПЗ. На відміну від кнопок, вбудованих у повідомлення клавіатура, вбудована в інтерфейс, вирішує проблему неочікуваного вводу користувача та дозволяє структурувати всі можливі дії користувача та інші додаткові можливості в один загальний інтерфейс вводу. Також можна комбінувати ці клавіатури, як було показано на рисунку, що умовно розділяє інтерфейс на ввід користувача та можливість переходу стану бота (натисканням кнопки «Пропустити»).

Базуючись на діаграмі варіантів використання, поданій у першому розділі, було вирішено створити інтерфейс користувача у вигляді структурованої клавіатури, вбудованої в інтерфейсі месенджера Telegram. Базуючись на діаграмі варіантів використання (рисунок 1.5), розглянемо деякі клавіатури програмного інтерфейсу користувача. Створимо клавіатуру входу в програму після реєстрації, тобто головне меню користувача (рисунок 2.10).

Рисунок 2.10 – Головне меню користувача

Цей макет клавіатури показує стандартне меню бота(зареєстрованого користувача), та дає можливість вибрати дію користувачу. В меню зображено дві основні та одна додаткова кнопки, вони будуть використані для взаємодії користувача з основними функціональними можливостями ПЗ. Пошук роботи буде реалізований пасивним та активним способом. Створення резюме буде відповідати за пасивний пошук, а пошук за категоріям – за активний. На рисунках 2.11 та 2.12 зображено два стани відображення меню користувача для пошуку роботи/працівників з заповненим раніше резюме/вакансією та без них.

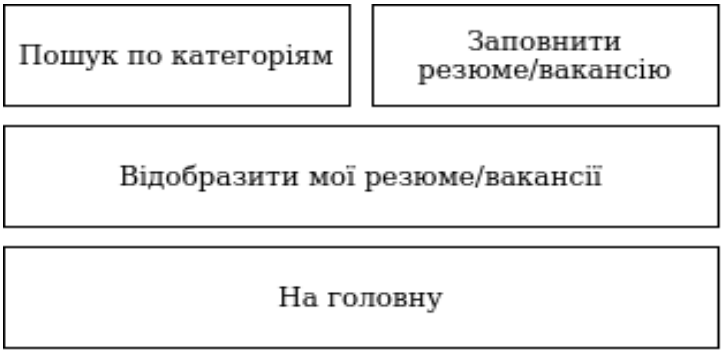


Рисунок 2.11 – Вигляд меню користувача з створеними публікаціями

Рисунок 2.12 – Вигляд меню користувача без створених публікацій

Після натискання на кнопку пошуку «Знайти роботу» або «Знайти працівників» буде відображено меню з вибором подальших дій. Як було вказано раніше, пошук буде відбуватись двома способами (пошук за категоріям та створення резюме/вакансії). Такий макет дозволить користувачу вибирати між швидким пошуком роботи за допомогою кнопки «Пошук по категоріям» та самому подавати заявки на роботу. Взаємодія з користувачем для заповнення резюме/вакансії буде відбуватись за допомогою постановки запитань ботом та відповідей користувача на них. Додаткова клавіатура для заповнення вакансії, а саме вибору виду зайнятості, зображена на рисунку 2.13.

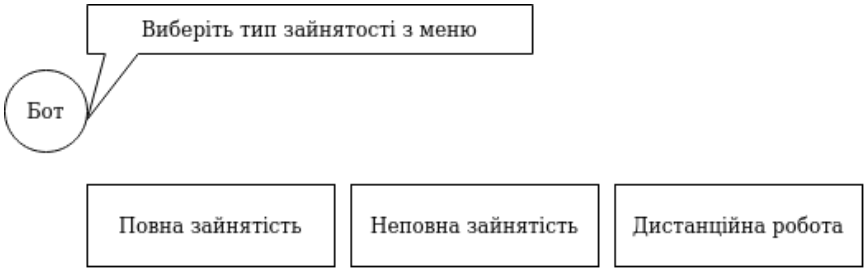


Рисунок 2.13 – Відображення клавіатури для вибору виду зайнятості

Така можливість відповіді користувача за допомогою клавіатури дозволить спростити відповідь боту на запитання та унеможливлює некоректну відповідь. На рисунку 2.14 зображено меню вибору відображення резюме/вакансій.

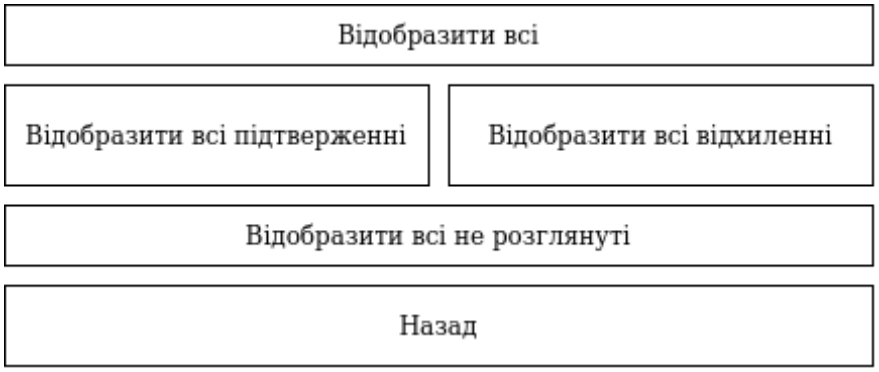


Рисунок 2.14 – Вибір відображення резюме/вакансій

Дана модель клавіатури розділяє статуси резюме/вакансії та полегшує пошук потрібних даних.

2.7 Аналіз та вибір технологій і методів реалізації системи

На відміну від мов програмування, які використовують для розробки користувацького інтерфейсу (frontend), розробка серверної частини ПЗ (backend) несе на собі велику відповідальність за коректне збереження, видачу та якщо потрібно, шифрування даних. Чим більше даних у проекті, а це тексти, різного роду медіа, особисті дані користувачів, різноманітні файли та інші дані, тим стабільнішим і потужнішим повинен бути сервер (backend). Backend повинен видавати у найкоротші терміни підготовану та структуровану інформацію для користувача на його запит. У більшості випадків очікування після надсилання запиту користувачем дуже дратує відвідувачів, тому «двигун» ПЗ повинен працювати достатньо швидко та без затримок обробляти та надсилати потрібну інформацію. Ситуація ускладнюється, коли користувачів стає набагато більше і всі вони очікують на швидку відповідь. Тому потрібно вибрати баланс між якістю

представлення даних та швидкістю їх обслуговування. На сьогодні найпопулярнішими мовами програмування для розробки серверної частити програми є C# та Java.

Дві ці мови програмування з'явилися в різний час. Java була створена задовго до появи C# компанією «Sun Microsystems», в 1995 році була випущена її перша бета-версія. Створення C# було анонсовано в 2000 році та перша версія з'явилась в 2002 році. Оскільки Java розроблювалась, опираючись на досвід мов Objective C і C, то C# базувалась вже на досвіді створення Java і C++. З точки зору розробника, який використовує ці дві мови, вони є являються схожими і ввібрали в себе багато синтаксису C++, але ці мови, на відміну від C++ простіші в освоєнні для новачків у програмуванні. Обидві мови супроводжуються багатими колекціями бібліотек. Обидві мови супроводжуються багатими колекціями бібліотек. Але є в мовах також свої особливості і відмінності, сильні і слабкі сторони, основні відмінності:

– підтримка узагальнень;
– перевірка виключень;
– поліморфізм;
– перерахування.

Підтримка узагальнень покращує перевірку типів з використанням компілятора, який видаляє приведення з вихідного коду. В Java засоби узагальнень здійснюються за допомогою стирань. C# також використовує узагальнення, додаючи його в CLI та додає інформацію про тип під час виконання програми, що дає незначне покращення продуктивності.

Перевірка виключень. Java має два типи винятків – які перевіряються та ті, що неможливо перевірити. C# ж має простіший спосіб, реалізуючи тільки один тип винятку. Можливість відловлювати виключення корисна, але вона також може мати негативний вплив на контроль версій і маштабування проекту.

Поліморфізм. C# і Java використовують різні підходи до реалізації поліморфізму. Java за замовчуванням дозволяє поліморфізм, на відміну від C#,

який повинен використовувати «virtual» в батьківському класі та «override» в дочірньому класі.

Перерахування. В C# перерахування представляють собою прості списки констант та походять від простих цілочисельних типів. Java розглядає перерахування більш детально, як клас, а його значення – як об’єкт. Таке бачення полегшує створення нової поведінки користувача в перерахуваннях [19].

Проаналізувавши ці мови, можна виділити те, що Java дасть можливість запустити розроблений додаток на Windows, Linux, MacOS та інших ОС, дозволить зменшити час на написання програми за рахунок використання фреймворків та інструментів для роботи з месенджером Telegram, тому будемо використовувати дану мову.

Найбільш поширеним фреймворком для побудови серверної частини додатків на Java є Spring. Його можна застосовувати для побудови будь-якого додатку на мові Java на відміну від багатьох інших платформ (таких як Apache Struts, яка обмежена створенням тільки веб-додатків). Цей фреймворк описують як полегшену платформу для побудови Java-додатків. В дійсності полегшена характеристика не дорівнює відношенням простоти написання до кількості класів або розмірів дистрибутива. Завдяки простоті написання ви маєте вносити мінімальні зміни в код програми, якщо ви вирішите змінити фреймворк або перестати його використовувати. Платформа Spring є полегшеної в сенсі використання всіх переваг ядра. Ядро Spring Framework засновано на принципі інверсії контролю (Inversion of Control), який дозволяє проектувати та реалізовувати додатки, використовуючи мало зв’язні окремі компоненти, перекладаючи деяку відповідальність керування нашим кодом на фреймворк [20]. Також даний фреймворк підтримує шаблон впровадження залежностей, при якому програмні компоненти отримують інші об'єкти, від яких вони залежать, названі залежностями, а не створює їх сам. Тобто первинний клас А використовує функціональні можливості класу В, а В є залежністю для класу А, тобто клас А має залежність від В. Такі залежності передаються за допомогою присвоєння

поля класу або конструктора. Spring не дає самостійно створювати додатки, для цього використовуються надбудови.

Spring Boot – це надбудова над фреймворком Spring, яка дозволяє найбільш простим способом розробляти додаток, вимагаючи від розробників мінімум зусиль з його налаштування і написання коду [21]. Ця надбудова володіє великими функціональними можливостями. Найбільш яскравими особливостями є: управління залежностями та автоматична конфігурація Spring, Spring Web.

Spring Web також відомий, як Spring MVC або Spring Web MVC, є модулем, який містить у собі інструменти для створення додатків, які пов'язані з мережею [21]. Цей модуль створює додатки за шаблоном MVC (Model-View-Controller).

Використання Spring Boot і Spring Web дає можливість розміщувати скомпільований код на веб-сервері. Spring Boot має вбудований сервер Tomcat, який слугує для цієї цілі [21].

Ручне складання проектів на мові програмування Java досить трудомістке. Потрібно правильно вказати потрібні бібліотеки, фреймворки та їх версії, від яких він залежить. З часом для автоматизації цього процесу програмісти створювали скрипти, але такий підхід був дуже поганий, так як всі розробники створювали схожі і однакові скрипти велику кількість разів. Для вирішення цієї проблеми на світ з'явлися системи збирання проектів. Ці системи давали можливість застосували більш-менш стандартизовані засоби для збирання та зібрати проект з мінімальним набором тільки потрібних залежностей.

На сьогодні найбільш поширеними збиральниками проектів є Gradle і Maven. Головна відмінність між ними це те, що Maven керує збірками проектів, звітами і документами, і те, що Maven спрощує процес складання проекту і забезпечує простий і прозорий перехід до передових функцій. Gradle, у свою чергу позиціонується, як система, заснована на JVM для автоматизації збірників проектів з відкритим вихідним кодом, орієнтована на продуктивність і гнучкість. Дотримуючись підходу «build-by-convention», Gradle дозволяє моделювати ваше ПЗ за допомогою потужної мови Groovy, замість XML(який використовується в

Maven). Оскільки Gradle оснований на JVM, він дозволяє вам писати призначену для користувача логіку на Java або Groovy. Gradle дозволяє структурувати збірку, крім того він підтримує можливість збірки декількох проектів одночасно, використовуючи простий спосіб міграції [22].

Для розроблюваного ПЗ суттєвої різниці використання того чи іншого інструменту не буде відчутно, оскільки розроблюваний програмний продукт буде написано на мові Java, буде використовувати Gradle для зручності.

Отже, в даному розділі було проведено аналіз сучасних архітектур, розглянуто переваги та недоліки монолітного та мікросервісного стилю побудови додатку та визначено, що монолітний стиль буде кращим вибором, оскільки розроблюване ПЗ не буде розбито на компоненти, а буде розроблено в єдиній кодовій базі. Також було проаналізовано типи баз даних, порівняно їх та вирішено, що для реалізації даного проекту краще використовувати нереляційну базу даних MongoDB через її зручність у використанні та швидкість виконання запитів. Було визначено, що для написання додатку буде використано мову програмування Java. Додатково буде використовуватись фреймворк Spring і його додаткові модулі, а також систему автоматичного складання проектів Gradle.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Реалізація серверної частини ПЗ

Створення та написання коду для розроблюваного ПЗ буде виконано за допомогою інтегрованого середовища для розробки – Eclipse. Як було вирішено, при проектуванні, виборі технологій і методів реалізації системи для стандартизації та складання проекту з мінімальними залежностями буде використано Gradle. При виборі нового проекту «Gradle project» буде створено стандартний шаблон Java проекту з автоматично налаштованим збиральником проектів Gradle.

Для того, щоб розпочати написання коду, потрібно вказати залежності, які ми будемо використовувати для розробки в файл, який відповідає за автоматичне завантаження бібліотек «build.gradle»:

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-
data-mongodb'
    implementation 'org.springframework.boot:spring-boot-starter-
web'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-
starter-test'
    compile group: 'org.telegram', name: 'telegrambots-spring-boot-
starter', version: '4.4.0.2'
}
```

Для зменшення об’єму коду, пришвидшення та спрощення його написання було використано бібліотеку Lombok, яка замінює стандарті методи та інші інструменти анотаціями. Наприклад, для заміни коду методів, які встановлюють значення для приватних полів класу, так звані «set-методи», буде використано анотацію @Setter, аналогічно для «get-методів» буде використана анотація @Getter, @AllArgsConstructor згенерує код конструктора для всіх полів класу.

Також ця бібліотека дозволяє замінити шаблон проектування «Будівельник», однією анотацією @Builder.

Після підготовки всіх залежностей можна розпочинати написання коду. Для взаємодії з ботом потрібно налаштувати конфігурацію для нього:

```
@Bean
public JobalookTelegramBot jobalookTelegramBot(TelegramRequestRoof
telegramRequestRoof) {
    DefaultBotOptions defaultBotOptions =
ApiContext.getInstance(DefaultBotOptions.class);
    JobalookTelegramBot jobalookTelegramBot = new
JobalookTelegramBot(defaultBotOptions, telegramRequestRoof);
    defaultBotOptions.setBotUserName(botUserName);
    defaultBotOptions.setBotToken(botToken);
    defaultBotOptions.setWebhookPath(WebhookPath);
    return jobalookTelegramBot;
}
```

Для отримання оновлень про введене повідомлення користувача було вирішено використати Webhook, встановлення якого відбувається при створенні об’єкта JobalookTelegramBot, також вказується унікальний токен і ім’я бота.

Початок роботи та ініціалізація усіх залежностей фреймворка Spring відбувається при виклику методу main:

```
@SpringBootApplication
public class JobalookApplication {
    public static void main(String[] args) {
        SpringApplication.run(JobalookApplication.class, args);
    }
}
```

Анотація @SpringBootApplication замінює інші анотації, а саме: @Configuration, @EnableAutoConfiguration, @ComponentScan. Для того, щоб фреймворк розумів, де шукати класи з анотаціями, які йому потрібні, використовується @ComponentScan. Після ініціалізації усіх компонентів програми, розпочинається очікування повідомлень від користувача. Початкова точна отримання оновлення від месенджера – WebhookController:

```
@RestController
public class WebhookTelegramController {
    private final JobalookTelegramBot jbtelegramBot;
    public WebhookTelegramController(JobalookTelegramBot
jbtelegramBot)
    {this.telegramBot = jbtelegramBot;
    }
    @RequestMapping(value = {"/"}, method =
{ RequestMethod.POST } )
    public BotApiMethod<?> onUpdateReceived(@RequestBody Update
update) {
        return jbtelegramBot.onWebhookUpdateReceived(update);
    }
}
```

Для того, щоб фреймворк Spring розумів, як правильно створювати об'єкт класу, та розумів, що цей об'єкт потрібно створювати автоматично при запуску програми, над класом встановлюється анотація @RestController, яка вказує, що даний клас є контролером та взаємодія з ним відбувається за допомогою REST. За отримання оновлень користувача відповідає метод onUpdateReceived, який позначений анотацією @RequestMapping, яка вказує на метод HTTP та кінцеву точку в URL. Цей метод отримує оновлення користувача в об'єкті Update та передає його далі по рівнях, а саме в клас обробник – TelegramRequestRoof, в якому розташований метод handeUpdate для обробки об'єкта Update:

```
public BotApiMethod<?> handleUpdate(Update update) {
    log.info("get message {}" , update);
    SendMessage responseMessage = null;
    Message messageHandle = update.getMessage();
    if (message != null && message.hasText()) {
        responseMessage = this.handleInputMessage(message);
    }
    return responseMessage;
}
```

Метод handleUpdate відповідає за фільтрацію порожніх запитів, та слугує для логування вхідного повідомлення за допомогою бібліотеки Slf4j. Ініціалізація логувальника відбувається за допомогою анотації @Slf4j над класом. Ця

бібліотека дозволяє занотовувати помилки, які відбуваються при роботі програми, вказувати, які дані спричинили помилку та зберігати всі ці дані, враховуючи час, коли помилка відбулась, на фізичний диск.

Коли користувач надіслав повідомлення до бота вперше, відбувається створення його та запис в базу даних за допомогою об'єкта UserRepository. Для того, щоб бот розумів, як йому потрібно обробляти введене повідомлення, при його отриманні відбувається перевірка відповідності командам, які переводять бот у певний стан:

```
private SendMessage handleInputMessage(Message message) {
    int userId = message.getFrom().getId();
    BotState botState;
    String inputMessage = message.getText();.
    User updateStatus = repository.findByUserId(userId);
    switch (inputMessage) {
        case "/start":
            log.info("Hello case");
            botState = BotState.HELLO;
            break;
        case "Знайти працівників":
            botserStState = BotState.FIND_EMPLOYEE;
            updateUatus(updateStatus, BotState.FIND_EMPLOYEE);
            break;
        case "Знайти роботу":
            updateUserStatus(updateStatus, BotState.FIND_JOB);
            botState = BotState.FIND_JOB;
            break;
        default:
            User userDefault = repository.findByUserId(userId);
            botState = userDefault.getState();
            break;
        ...
    }
}
```

Повідомлення поділяються на два типи: ті, які змінюють статус бота та ті, які несуть якусь інформацію від користувача. Якщо повідомлення змінює стан бота, то відповідно оновлюються статус бота для конкретного користувача в базі даних, та встановлюють його всередині системи. Відповідно якщо повідомлення

несе якусь інформацію від користувача, наприклад коли користувач заповнює резюме, то стан для бота береться з бази даних.

Після визначення стану бота визначається обробник, який буде використаний для обробки повідомлення та повернення відповіді, за цю функціональність відповідатиме клас BotStateChecker та два методи findHandler і додатковий, який відповідають назві обробника:

```
private MessageHandler findHandler(BotState state) {
    log.info("grabbing handler for user, state: {}", state);
    if (isFillingProfileState(state)) {
        return messageHandlers.get(BotState.FILLING_PROFILE);
    } else if (isFindJobDoState(state)) {
        return messageHandlers.get(BotState.FIND_JOB);
    } else if (isFillingJobState(state)) {
        return messageHandlers.get(BotState.FIND_EMPLOYEE);
    }
    ...
    return messageHandlers.get(state);
}

private boolean isFillingProfileState(BotState state) {
switch (state) {
    case ASK_GENDER:
    ...
    case FILLING_PROFILE:
    case PROFILE_FILLED:
        return true;
    default:
        return false;
    }
}
```

Обробник визначається за конкретним станом бота, якщо стан бота співпадає з переліченням станів, то повертається назва обробника. Після того, як обробник був виявлений та його екземпляр створений, викликається метод handle та передається повідомлення, яке користувач ввів, на подальшу обробку. Для того, щоб стандартизувати обробники, було вирішено, щоб усі вони мали стандартизований набір методів, які вказані в інтерфейсі MessageHandler:

```
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
```

```
public interface MessageHandler {
    SendMessage handleMessage(Message message);
    BotState getName();
}
```

Припустимо, що повідомлення користувача було «/start», тоді для його обробки потрібно використати клас HelloHandler. Для того, щоб Spring розумів, що об’єкт цього класу йому потрібно створити автоматично, над класом потрібно встановити анотацію @Component, також в цьому класі будуть задіяні можливості логування, тому використаємо анотацію @Slf4j для автоматичного створення об’єкта логера. Для ін’єкції залежності репозиторію UserReposiotry буде використано анотацію @Autowired, яка встановлюється над полем класу. Розглянемо метод, який оброблятиме вхідне повідомлення користувача:

```
private SendMessage processUsersInput(Message inputMessage) {
int userId = inputMessage.getFrom().getId();
long userChatId = inputMessage.getChatId();
User user = repository.findByUserId(userId);
log.info("Changing user status: {}" ,user);
user.setState(BotState.FILLING_PROFILE);
log.info("Save user: {} to database" ,user);
repository.save(user);
SendMessageresponseForUser = replyService
    .sendHelloMessage(userChatId,"reply.helloMessage",
    inputMessage.getFrom().getFirstName());
responseForUser.setReplyMarkup(getMainMenuKeyboard());
return responseForUser;
}
```

Для того, щоб бот розумів, який користувач надіслав повідомлення, з об’єкту Message отримується унікальний ідентифікатор користувача та чату з ним. Після отримання усіх потрібних даних для ідентифікації користувача, потрібно оновити його статус на “FILLING_PROFILE”, який вводить бота в стан заповнення профіля користувача. За допомогою об’єкта UserRepository знаходить користувача в базі даних, отримує всі його дані та записує в об’єкт User, після чого оновлює його статус і зберігає його в базу даних. Щоб спростити редагування повідомлень, усі вони були зібрані в файлі ресурсів

«messages_ua_UA.properties», для їх отримання використовується допоміжний клас ReplyService. Фрагмент коду для отримання повідомлення:

```
private final Locale locale;
private final MessageSource messageSource;
public LocaleService(@Value("${localeTag}") String localeTag,
MessageSource messageSource) {
    messageSource = messageSource;
    ocale = Locale.forLanguageTag(localeTag);
}
public String getMessage(String message) {
    return messageSource.getMessage(message, null, locale);
}
public String getMessage(String message, Object... args) {
    return messageSource.getMessage(message, args, locale);
}
```

Після отримання привітання для користувача потрібно в додаток до повідомлення додати клавіатуру, яка буде знаходитись знизу месенджера:

```
private ResponseKeyboard getMainMenuKeyboard() {
    final ResponseKeyboard responseKeyboard = new
ResponseKeyboard();
    responseKeyboard.setSelective(true);
    responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboardRow = new ArrayList<>();
    KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Розрочнемо"));
    keyboardRow.add(row_one);
    responseKeyboard.setKeyboard(keyboardRow);
    return responseKeyboard;
}
```

Для того, щоб уникнути повторного натискання на клавіатуру, в параметр «setOneTimeKeyboard» ставиться значення «true». Далі створюється масив стрічок для клавіатури, оскільки нам потрібно встановити лиш одну кнопку «Розпочати», створюється лиш одна стрічка.

Коли користувач розпочинає заповнення профілю, ПЗ отримує повідомлення, це повідомлення міняє статус бота на «FILLING_PROFILE» та

задає початкове питання користувачу. При відповіді користувача на запитання, система розуміє, що це відповідь на запитання, оскільки статус бота відповідає обробнику заповнення профілю. Запис даних в базу даних відбувається в конкретному обробнику, базуючись на станах бота. Розглянемо логіку отримання відповіді від користувача та запис даних в БД:

```
private SendMessage processUsersInput(Message inputMessage, int
userId) {
log.info("Proccess response user message: {}", inputMessage);
String userResponse = inputMessage.getText();
long userChatId = inputMessage.getChatId();
BotState
botState=jobalookDataCache.getUsersCurrentBotState(userId);
log.info("Proccess BotState: {}", botState);
SendMessage responseForUser = null;
User user = repository.findByUserId((long)userId);
if (botState.equals(BotState.ASK_AGE)) {
    user.setName(userResponse);
responseForUser=messagesService.getReplyMessage(userChatId,"reply.a
skAge");
    user.setState(BotState.ASK_GENDER);
    repository.save(user);
}
...
if (botState.equals(BotState.PROFILE_FILLED)) {
    Pattern emailPattern = Pattern.compile("^[\\w-
\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");
    Matcher matcher = emailPattern.matcher(userResponse);
    if(!matcher.matches()) {
        return new SendMessage(userChatId, "Ви ввели не вірне
значення");
    }
    jobalookDataCache.setuUsersCurrentBotState(userChatId,
BotState.ASK_TODO);
    user.setState(BotState.ASK_TODO); user.setEmail(userResponse);
    repository.save(user);
    responseForUser = new SendMessage(userChatId,
String.format("Твій профіль виглядає так: \n * Твоє
ім'я: %s\n ...", user.getName()...));
    responseForUser.setReplyMarkup(getTODOKeyboard(user));
}
return responseForUser; }
```

Даний метод обробляє вхідне повідомлення користувача, перевіряючи актуальний статус бота, для конкретного користувача. Коли статус бота співпадає

з поточним статусом, відповідь користувача записується в базу даних, як відповідь на попереднє запитання, за допомогою об'єкта UserRepository. Чергуючи запитання та отримання відповіді, користувач заповнює свій профіль. Коли користувач дав відповідь на останнє запитання(яка його електронна адреса), відбувається перевірка на відповідність наданої відповіді й потрібного формату для електронної адреси, використовуючи шаблон. Якщо дана відповідь не відповідає належній, ПЗ не міняє статус бота, а повертає користувачу повідомлення про некоректний ввід. Після отримання коректної відповіді ПЗ надсилає користувачу візуальний вигляд його профілю у вигляді текстового повідомлення, також у додаток до повідомлення надсилає клавіатуру для вибору наступної дії.

При завершенні реєстрації користувачу відкривається доступ до основної функціональності програмної системи, а саме: знайти роботу та знайти працівників. Для пошуку вакансій на роботу користувачу потрібно натиснути на кнопку «Знайти роботу», після того програмна система надсилає йому у відповідь меню з вибором дій для пошуку роботи. Для пошуку роботи/працівників за категоріями користувачу потрібно відобразити список категорій та дати можливість вибрати одну з них. У програмній системі існує тридцять категорій, які пронумеровані від одиниці до тридцяти. За допомогою введення номеру категорії користувач вибирає категорію, а ПЗ надсилає відповідь з їх списком. Фрагмент коду для відображення користувачу списку категорій:

```
public SendMessage createMessageWithCategories(SendMessage
sendMessage) {
SendMessage newSendMessage = sendMessage;
StringBuilder messageBuilder=new StringBuilder
(sendMessage.getText()+"\n              " + "Список
категорій\n**********************************\n");
int i=0;
messageBuilder.append(String.format("№  | Назва\n"));
for (JobCategoryes category : JobCategoryes.values()){
    i++;
    if(i>=10){
        messageBuilder.append(String.format("%s  | %s\n",i ,
category.toString()));
```

```
    } else {
        messageBuilder.append(String.format("%s    | %s\n",i ,
category.toString()));
    }
}
messageBuilder.append("\n***********************************\n");
newSendMessage.setText(messageBuilder.toString());
return newSendMessage;
}
```

Після отримання списку категорій користувач надсилає номер категорії і отримує у відповідь список даних. Щоб спростити відображення для користувача вакансій/резюме та не завдавати незручностей при їх виборі, відображається тільки декілька з них, для цього використовується розбиття на сторінки (Pagination). Розглянемо метод для розбиття списку даних на сторінки:

```
public <T> List<T> paginationList(List<T> tList, int userId) {
     return getCurrentPage(tList,
     jobalookDataCache.getUserCurrentPage(userId));
}
private <T> List<T> getCurrentPage(List<T> classList, Integer
pageNumber) {
     Integer fromIndex = pageNumber * 3;
     Integer to = fromIndex + 3;
     Integer toIndex = to > classList.size() ?classList.size(): to;
     return classList.subList(fromIndex, toIndex);
}
public <T> List<T> paginationListModerator(List<T> tList, int
userId) {
     return getCurrentPageModerator(tList,
     jobalookDataDache.getUserCurrentPage(userId));
}
private <T> List<T> getCurrentPageModerator(List<T> classList,
Integer pageNumber) {
     Integer fromIndex = pageNumber * 1;
     Integer to = fromIndex + 1;
     Integer toIndex = to > classList.size() ?classList.size(): to;
return classList.subList(fromIndex, toIndex);
}
```

Для визначення поточної сторінки та потрібних даних в ній використовується класичний алгоритм поділу списку на підсписок. Для того, щоб розбити сторінки, потрібний масив значень та поточна сторінка. Збереження поточної сторінки відбувається в кеші програми за допомогою хеш-таблиці, дані

в якій зберігаються у вигляді «ключ-значення», де ключем виступає унікальний ідентифікатор користувача, а значенням номер сторінки. Фрагмент коду збереження та отримання даних в кеші програми:

```
Private Map<Integer, Integer> usersCurrentPageSize = Collections.emptyMap();
public void setUsersCurrentPageSize(int userId, Integer currentPage)
{   log.info("Update current page for user: {}", userId);
    usersCurrentPageSize.put(userId, currentPage); }
public Integer getUserCurrentPage(int userId) {
    try {
        Integer userCurrentPage = usersCurrentPageSize.get(userId);
        if(userCurrentPage==null){
            return 0;
        }
        return userCurrentPage;
    } catch (Exception ex) {
    return 0;
    }
}
```

Переключення між сторінками відбувається за допомогою кнопок, які йдуть у додаток з повідомленням, яке містить дані конкретної сторінки. Щоб уникнути помилок при переключенні між сторінками, потрібно відображати кнопки для переключення сторінок у певних ситуаціях, наприклад, якщо поточна сторінка дорівнює одиниці, то кнопку попередня сторінка не потрібно відображати. Псевдокод для відображення кнопок:

```
if (розмір списку !=0) {
    if( розмір списку > поточна сторінка+3) {
         створити нову стріку(new Кнопка("Наступна сторінка"));
         додати до клавіатури(стрічка);
    }
    if(поточна сторінка!=0 && розмір списку !=0) {
```

```
        створити нову стрічку(new Кнопка("Попередня сторінка"));
        додати до клавіатури(стрічка);
    }
}
```

В додаток до кожної вакансії додається його унікальний ідентифікатор, який користувач може надіслати та подати заявку на роботу. Пошук працівників відбувається аналогічно, але при відображенні резюме роботодавцю відображаються контактні дані власника цього резюме для зв’язку.

Інший спосіб пошуку роботи/працівників це створення резюме/вакансій. Щоб розпочати заповнення даних для публікації, користувачу потрібно натиснути на відповідну кнопку в меню месенджера. Після цього бот почне взаємодіяти з користувачем аналогічно до заповнення резюме, також при завершенні заповнення користувачу буде відображено вигляд публікації та дана можливість відправити на перевірку модератору, або заповнити вакансію заново. Автоматично для кожної публікації створюється унікальний ідентифікатор, який складається з шести символів. Фрагмент коду для створення ідентифікатору публікації:

```
private String generateUID() {
UUID uuid = UUID.randomUUID();
String id = uuid.toString();
String[] newId = id.split("-");
String newUUID = newId[newId.length-1].substring(3,9);
log.info("Generate UUID:{} split ID:{}", uuid, newUUID);
return newUUID;
}
```

Коли користувач має в наявності створені публікації, йому відкривається додаткова кнопка для їх відображення. Щоб спросити відображення, також використовується «Pagination» та попередньо дається можливість сортування публікацій за типами: підтверджені, відхилені, не розглянуті. Для отримання з

бази даних тільки потрібних даних використовується відповідні репозиторії з спеціальними методами. Репозиторії позначають спеціальною анотацією @Repository, яка вказує на те, що фреймворк повинен створити об'єкт цього класу в автоматичному режимі. Щоб спростити написання запитів в базу даних, використовують Spring Data JPA. Створення таких запитів не має особливого синтаксису, а базується на автоматичній генерації відповідно до назви методу. Фрагмент коду запитів для відображення резюме в базу даних:

```
@Repository
public interface ResumeRepository extends MongoRepository<Resume,
String> {
    Resume findByUserId(long userId);
    List<Resume> findAllByUserIdAndApprovedIsTrue(int userId);
    List<Resume> findAllByUserIdAndDeclinedIsTrue(int userId);
    List<Resume> findAllByUserIdAndCheckedIsFalse(int userId);
    List<Resume> findAllByCheckedIsFalse();
    List<Resume> findAllByUserId(int userId);
    Resume findByUniqueId(String uniqueId);
    void deleteByUniqueId(String uniqueId);
    List<Resume> findAllByCategoryAndApprovedIsTrue(JobCategoryes
jobCategory);
}
```

При відображенні публікацій до них прикріплюється ідентифікатор, який дає можливість користувачу вибрати додаткову дію видалення або редагування. Редагування публікації доступно тільки тоді, коли вона ще не переглянута модератором, відповідно після модерації публікації її можна тільки видалити. Для роботодавця доступна додаткова функція при відображені всіх підтверджених публікацій – «відобразити список запитів». Фрагмент коду для відображення списку запитів на роботу:

```
if(state.equals(BotState.SHOW_MY_JOBS_REQUESTS)) {
```

```
    jobalookDataCache.setUsersLastState(userId,
BotState.SHOW_MY_JOBS_REQUESTS);
    String lastUniqueId = jobalookDataCache.getLastUniqueId(userId);
    Job requestedJob = repository.findByUniqueId(lastUniqueId);
    List<User> requestedUsers = requestedJob.getRequested();
    log.info("SHOW_REQUESTS lastUniqueId: {}", lastUniqueId));
    if(requestedUsers.size()==0)
    {
     user.setState(BotState.FIND_EMPLOYEE);
     userRepository.save(user);
        SendMessage showRequestedEmpty =
        new SendMessage(userChatId, "Список пустий");
         showRequestedEmpty.setReplyMarkup(
          getJobKeyboardWithShowJobs());
         return showRequestedEmpty;
    }
SendMessage myResumesMessage=messagesService.getReplyMessage(userCh
atId, "reply.sayHowToCheckRequestedUser");
List<User>                        usersPagieable                    =
paginationService.paginationList(requestedUsers, userId);
StringBuilder myUsersBuilder = new StringBuilder();
     for (User item : myUsersPagieable)
     {
         myUsersBuilder.append(prepareUserToShow(item));
     }
myResumesMessage.setText(myResumesMessage.getText()+"\n"+myUsersBui
lder.toString());
myResumesMessage.setReplyMarkup(getPaginationButtons(myUsersPagieab
le, jobalookDataCache.getUserCurrentPage(userId)));
user.setLastState(BotState.FIND_EMPLOYEE);
userRepository.save(user);
return myResumesMessage;
}
```

Код (лістинг) програми подано у додатку В.

3.2 Керівництво користувача

Коли користувач, ще не спілкувався з ботом і вперше відкриває діалог, для ініціації розмови з ботом йому потрібно натиснути кнопку «START». Після ініціації розмови з ботом у відповідь користувачу бот відправить привітальне повідомлення з кнопкою «Розпочати» (рисунок 3.1).

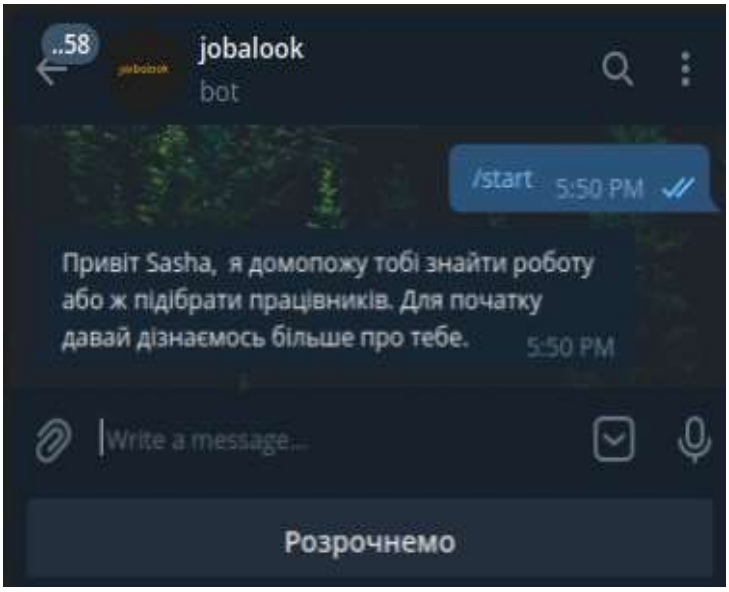


Рисунок 3.1 – Привітальне повідомлення бота

Після натискання на кнопку «Розпочати» бот надсилає запитання для заповнення особистого профілю користувача (рисунок 3.2).

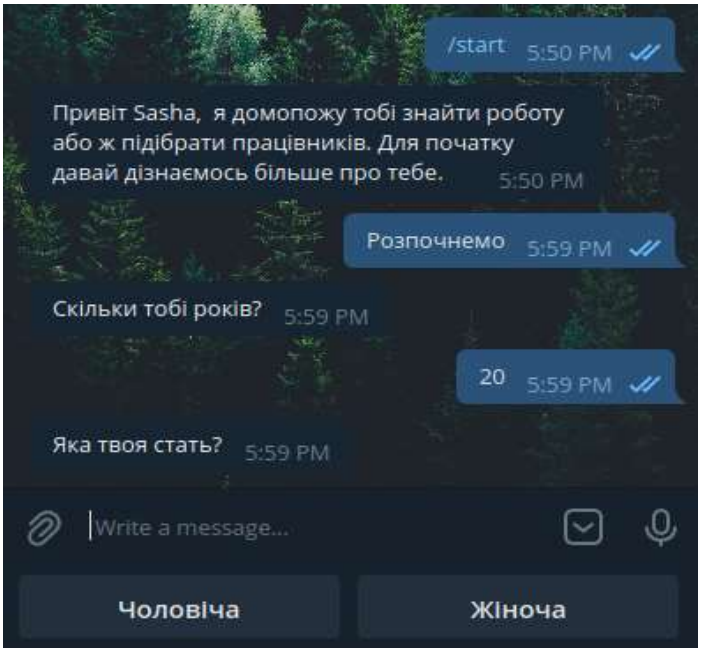


Рисунок 3.2 – Заповнення анкети користувача

Коли користувач дає відповідь на останнє запитання бот у відповідь на нього надсилає відображення його профілю та в додаток до повідомлення клавіатуру головного меню. На рисунку 3.3 зображено головне меню бота.

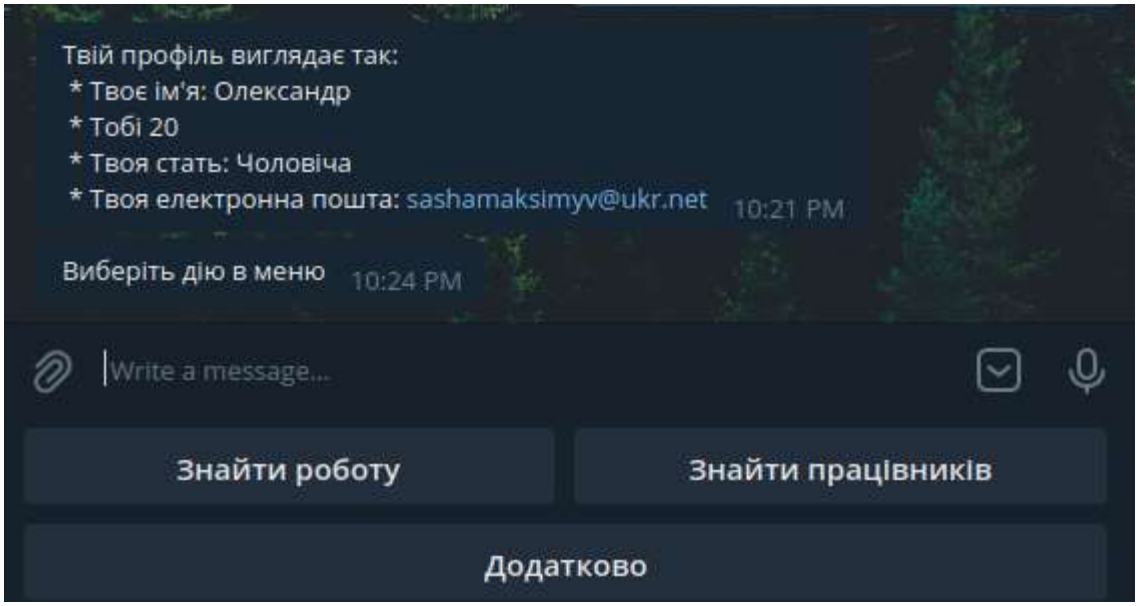


Рисунок 3.3 – Головне меню бота

В головному меню в користувача є три вибори відносно його потреб. Для того, щоб знайти роботу користувачу потрібно натиснути кнопку «Знайти роботу», яка відкриває меню з вибором подальшої дії (рисунок 3.4).

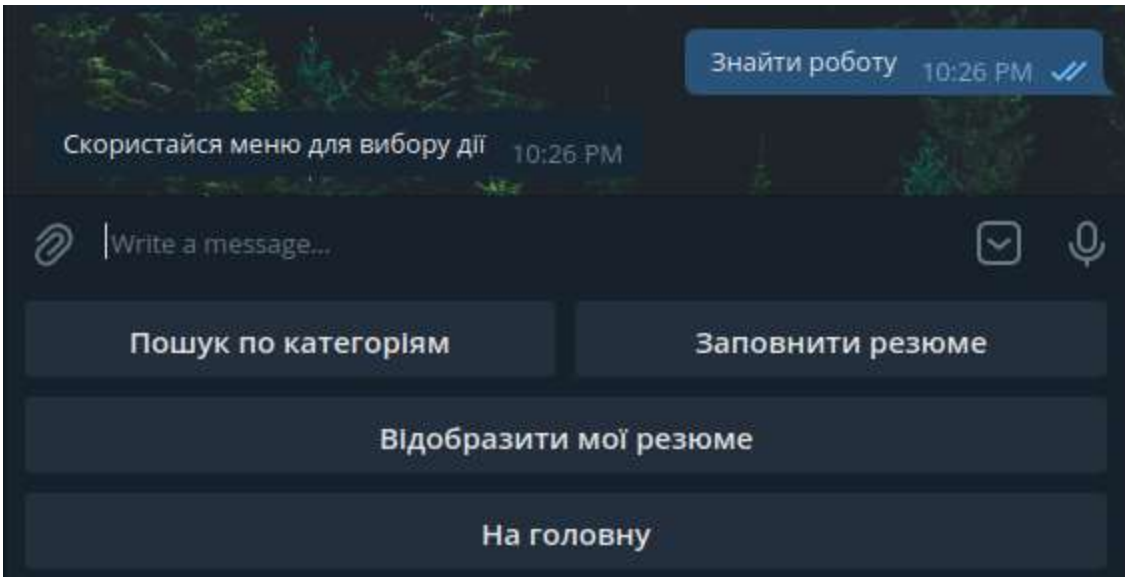


Рисунок 3.4 – Меню вибору для пошуку роботи

Пошук роботи відбувається активним та пасивним способами. Для активного пошуку використовується пошук по категоріям, відповідно для пасивного пошуку – заповнення резюме. Коли користувач визначився з вибором

подальшої дії, він натискає на одну з кнопко меню. Натиснувши на кнопку «Пошук по категоріям» буде відображено список категорій, при виборі однієї з яких буде відображено сторінку у вигляді повідомлення, яка складається з трьох вакансій. Для подачі заявки користувачу потрібно надіслати боту унікальний ідентифікатор вакансії. На рисунку 3.5 зображено приклад вибору вакансії для подачі заявки.

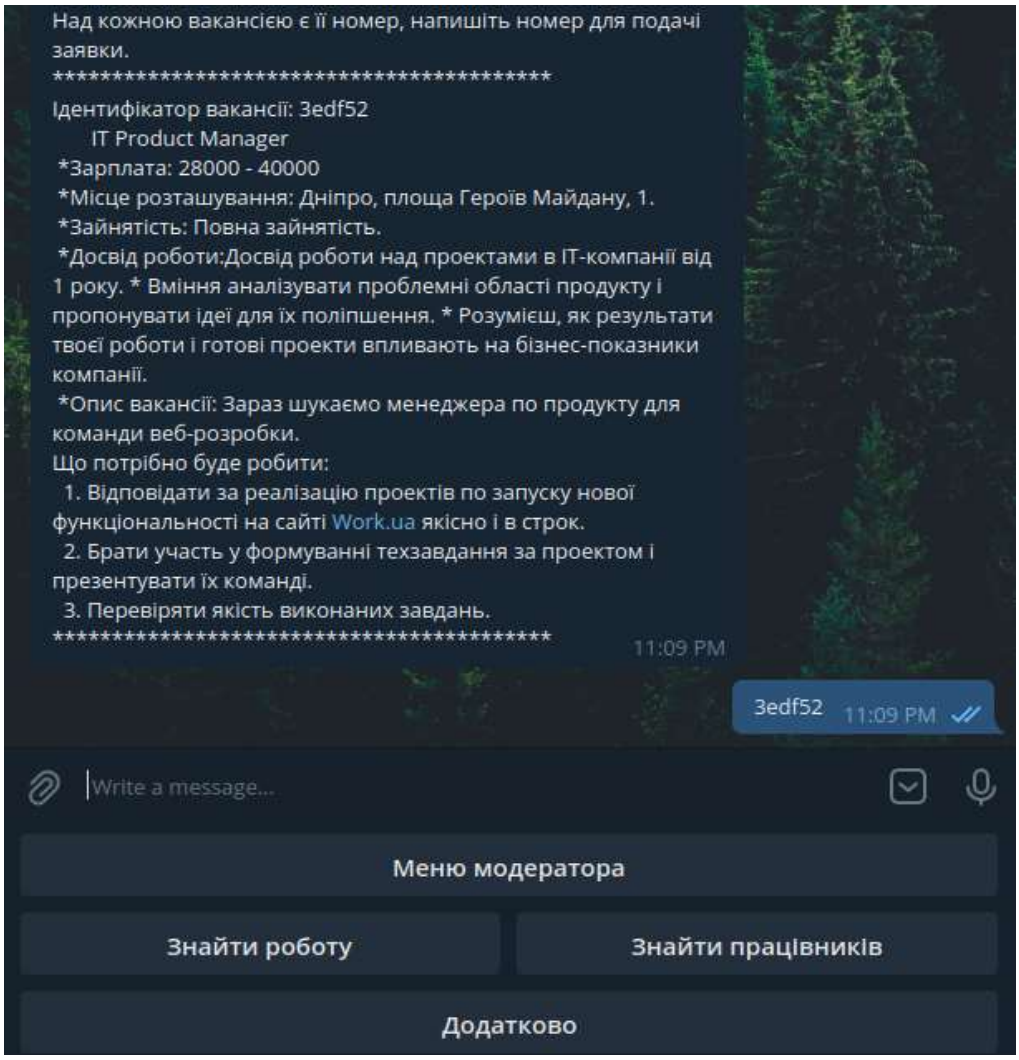


Рисунок 3.5 – Пошук та подача заявки на роботу

Для створення резюме потрібно перейти в меню «Знайти роботу» та натиснути на кнопку «Заповнити резюме». Після цього користувачу бот буде надсилати запитання, а користувачу потрібно буде відповідати на них. На рисунку 3.6 зображено процес заповнення резюме.

Рисунок 3.6 – Процес заповнення резюме

Після завершення запитань бот у відповідь на останнє запитання надішле вигляд вашого резюме та дві кнопки з вибором надіслати ваше резюме на перевірку модератору або ж заповнити його заново без збереження. Якщо користувач не задоволений своїм резюме він може заповнити його заново натиснувши відповідну кнопку. Коли користувач відправив на перевірку резюме, йому відкривається доступ до меню з вибором відображення своїх резюме:

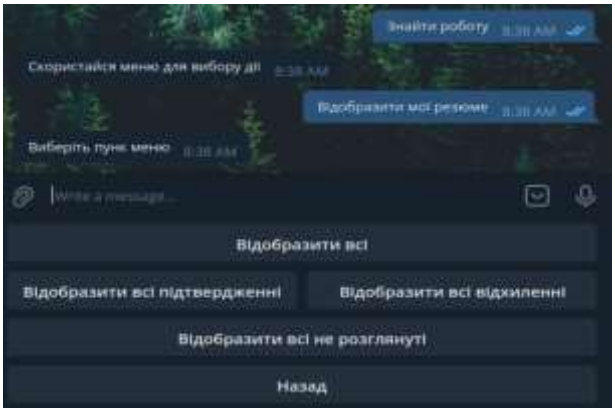


Рисунок 3.7 – Меню вибору відображення резюме

Для того, щоб користувачу відобразити резюме йому потрібно натиснути на відповідну кнопку з меню, назва якої відповідає статусу резюме. Пошук по категоріям, створення вакансії та їх відображення відбувається аналогічним чином. Для полегшення відображення вакансій користувачу використовується розбиття контенту на сторінки (Pagination). Користувачу для переходу між сторінками потрібно натиснути на відповідну кнопку наступної або попередньої сторінки, якщо користувач хоче завершити перегляд публікацій він може натиснути кнопку «Завершити» (рисунок 3.8).

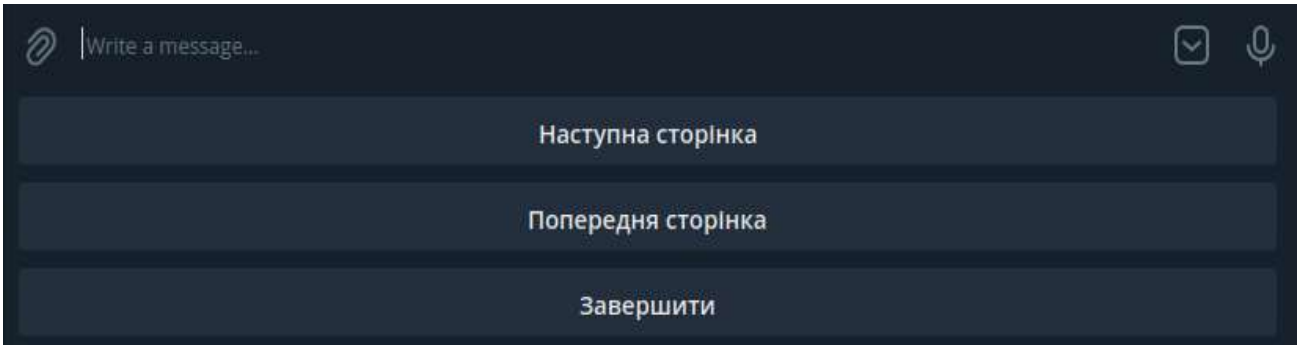


Рисунок 3.8 – Меню кнопок для переключення між сторінками

Додатково до основних функцій бота додається модерація публікацій, яка доступна тільки користувачам з привілегією модератора. Для того, щоб розпочати модерувати вакансії користувачу потрібно перейти в головне меню, перейти в меню модератора натиснувши на кнопку «Меню модератора» та натиснути на кнопку «Модерувати вакансії». Після натискання модератору буде відображено не переглянуту вакансію та надано три кнопки для вибору: «Підтвердити», «Відхилити», «На головну». Якщо модератор вирішить відхилити вакансію йому потрібно натиснути кнопку «Відхилити», після натискання на кнопку потрібно буде ввести причину відхилення, яка буде відображена власнику публікації в відображені всіх відхилених вакансій. На рисунку 3.9 та 3.10 відображено приклад взаємодії з ботом при модерації вакансій та відображення відхиленої вакансії.

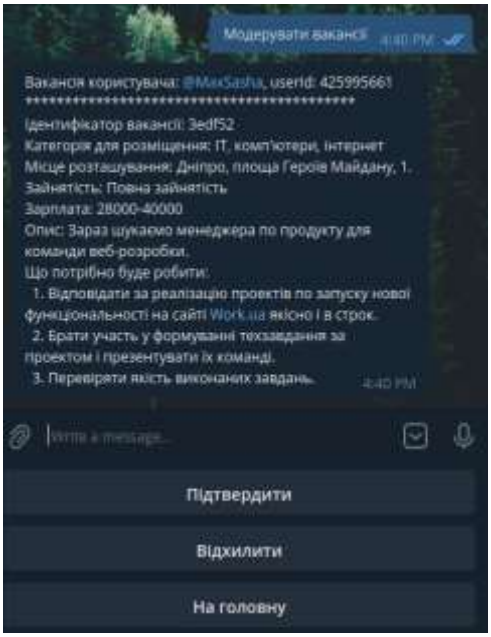


Рисунок 3.9 – Приклад модерації публікацій

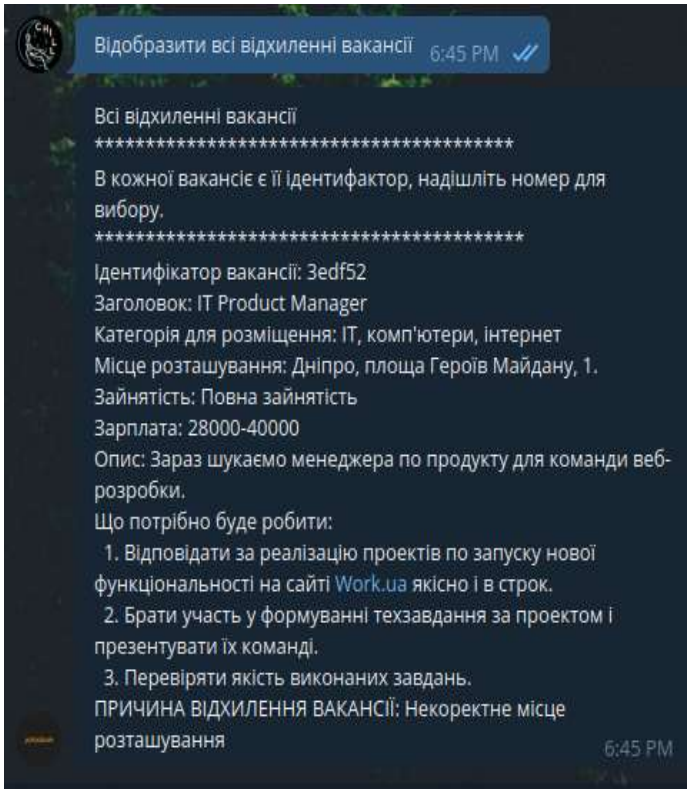


Рисунок 3.10 – Вигляд відображення відхиленої вакансії

Для ознайомлення з функціональними можливостями бота передбачено інструкцію для користування у вигляді тексту. Щоб відобразити його потрібно зайти в головне меню та натиснути на кнопку «Додатково» та вибрати в меню

«Допомога». Також в цьому меню є можливість відобразити свій профіль натиснувши «Відобразити мій профіль», якщо користувач хоче щось змінити в своєму профілі він може натиснути на «Заповнити профіль заново».

3.3 Вимоги до технічних та програмних засобів

Для того, щоб розпочати використовувати бота користувачу потрібен месенджер Telegram версією більшою за 3.2.5 в любому вигляді, будь то браузерна версія, мобільний додаток або комп'ютерна версія.

Програмне забезпечення де буде встановлено та запущено повинно відповідати наступним мінімальним вимогам:

- операційна система Linux Ubuntu 18.04;
- встановлене прогрмане забезпечення ngrok;
- 2-ядерний 64-розрядний процесор з частотою не меньше 2 ГГц;
- 2 Гб оперативноъ пам'яті;
- 20Гб вільного простору на жорсткому диску;
- стабільне Інтернет з'єднання з швидкістю не менше 50Мб/с.

3.4 Розгортання та встановлення системи

Для коректного розгортання системи потрібно встановити наступні програмні засоби: JRE не менше 11 версії та базу даних MongoDb.

Для встановлення необхідно виконати наступні команди в командному рядку слідуючи заданому порядку:

a) sudo apt update;

b) sudo apt-get install openjdk-11-jdk;

c) sudo apt-get install -y mongodb-org.

Щоб зв'язати бота з розробленим ПЗ, потрібно створити його в месенджері

Telegram за допомогою офіційного бота – BotFather. На рисунку 3.11 зображено процес створення бота, встановлення картинки та отримання його конфігурацій.

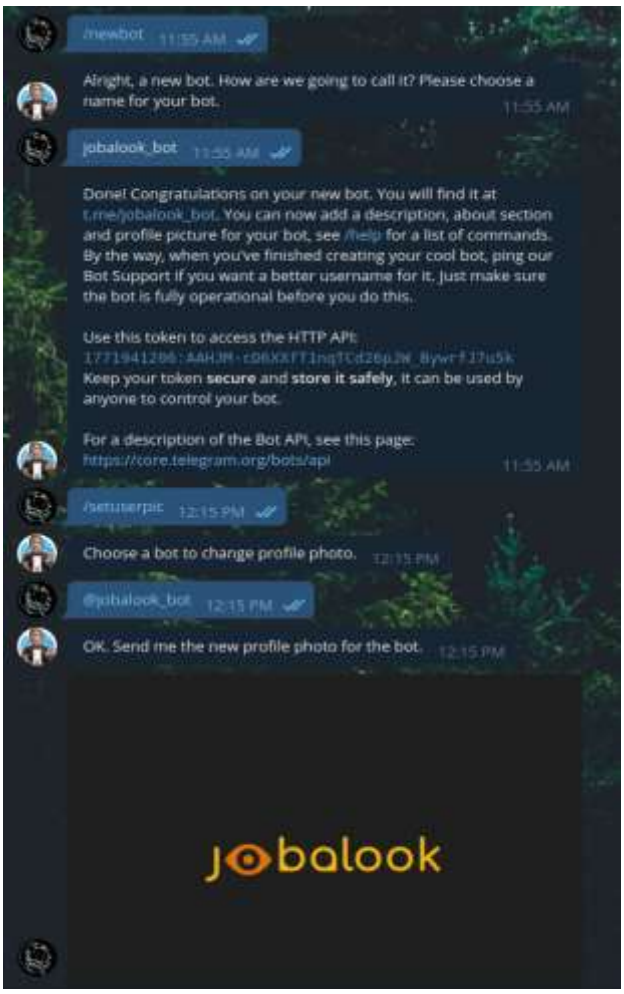


Рисунок 3.11 – Процес створення бота

Після завершення всіх вище перерахованих дій потрібно створити базу даних. Для цього потрібно виконати наступні команди в командному рядку слідуючи заданому порядку:

a) mongo -u root -p root;

b) use jobalook;

c) exit.

В ролі Webhook посилання може виступати ваше особисте або ж автоматично згенероване посилання, яке має відкритий порт 5000. Для

автоматичної генерації посилання потрібно встановити клієнт програми ngrok та запустити його використовуючи наступні команди в заданому порядку:

a) sudo apt-get update -y;

b) sudo apt-get install -y ngrok-client;

c) cd ngrok;

d) ./ngrok http 5000.

Закінчивши отримання необхідних даних, для запуску бота потрібно від–крити файл «application.properties», який знаходиться за шляхом: «jobalook/src/main/resources/» та встановити назву вашого бота в поле «telegrambot.userName», унікальний токен в «telegrambot.botToken» та вебхук адресу в «telegrambot. WebhookPath». Також потрібно перейти за посиланням https://api.telegram.org/bot<token>/setWebhook?url=link, попередньо замінивши поле «<token>» на той, який ви отримали раніше і замість поля «link», встановити згенероване посилання програмою ngrok.

Завершивши всі необхідні дії вам потрібно запустити jar файл програми попередньо відкривши командний рядок в каталозі «jobalook/build/libs» та вико-ристати команду «java -jar jobalook-0.0.1-SNAPSHOT.jar».

Для завершення роботи програми вам потрібно ввести в консольне вікно команду «sudo lsof -t -i:5000» .

У розділі було підготовлено всі залежності для написання коду, виконано практичну розробку програмних модулів і описано їх особливості та функціональні можливості за допомогою фрагментів коду. Було створено інструкцію користувача для взаємодії з інтерфейсом програми, а також описаний процес розгортання, встановлення та зупинення програмного забезпечення.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вибір та обґрунтування методів тестування додатку

Для тестування програмного забезпечення використовуються різні його типи. При виборі методу тестування потрібно врахувати те, що наше ПЗ знаходиться в єдиній кодовій базі. В такому випадку кращим вибором буде модульне тестування. Модульне тестування (Unit) – це метод, за допомогою якого окремі програмні модулі разом з даними і робочими процесами – тестуються, для того, щоб визначити чи коректно вони працюють. В об'єктно-орієнтованому програмуванні модулем, прийнято вважати ту частину програми, яку можна протестувати, це може бути клас, метод класу або ж їх сукупність. Загальноприйнятим правилом вважається, що програміст, який написав певний модуль відповідає за написання тестів для нього, оскільки він знає його очікувану поведінку та вимоги яким він має відповідати. Також для тестування сукупності модулів буде проведено системне тестування.

Взаємодія користувача і ПЗ буде відбуватись через інтерфейс користувача, тому тестування буде відбуватись за принципом «чорного ящику», в якому внутрішні компоненти і процеси не враховуються під час тестування.

Оскільки, реалізацію програмного забезпечення було виконано до написання тестових сценаріїв, то вони будуть написані за методологією BDD. Ця методологія в якої розробка базується на основі поведінки, основною ідеєю якої, являється поєднання в процесі розробки технічних інтересів і інтересів бізнесу, дозволяючи надавати загальну «мову» для програмістів та іншого персоналу.

Для написання модульного тестування на мові програмування Java було використано низку допоміжних бібліотек, а саме: AssertJ, Mockito, Junit, Spring Boot Starter Test. AssertJ – це бібліотека, яка дозволяє просто та швидко писати тести і отримувати достатньо інформації про помилки. Вона володіє багатим «багажом» фукнціональних можливостей, легко інтегрується, а також є плагіни для генерації assertions в класах і інтеграція allure. Mockito – фреймворк, який

дозволяє створювати тестові подвійні об'єкти(заглушки) в модульних тестах, які виконуються автоматично з метою розробки TDD або BDD. Spring Boot Starter Test – це стартовий набір бібліотек, які надають ряд утиліт і анотацій, що дають можливість створення середовища для реалізації тестових сценаріїв.

4.2 Розробка тестових наборів даних

Розпочнемо модульне тестування додатку. В створених тестових сценаріях буде перевірено функціональні можливості програми. Тестові сценарії деяких модульних тестів наведені в таблиці 4.1.

Таблиця 4.1 – Тестові сценарії модульних тестів

Ідентифікатор	Модуль	Вихідні дані	Очікуваний результат
1	2	3	4
J-A-1	Пошук Публікацій	Номер категорії	Список публікацій
J-A-2	Створення користувача	Відсутні	Ідентифікатор користувача, чату та Telegram ім’я користувача
J-A-3	Відображення публікацій	Статус публікації	Список публікацій
J-A-4	Подача заявки на вакансію	Ідентифікатор вакансії	Заявка подана
J-A-5	Модерація вакансії	Вердикт модератора	Вакансія пройшла модерацію
J-A-6	Підготовка тексту повідомлення	Створення та конфігурація тексту, ідентифікатор чату користувача	Сформований текст
J-A-7	Підготовка клавіатури для повідомлення	Створення та конфігурація кнопок та тексту, ідентифікатор чату користувача	Створена клавіатура

Кінець таблиці 4.1

1	2	3	4
J-A-8	Обробка повідомлення	Текстове повідомлення, ідентифікатор користувача, ідентифікатор чату	Отримання стану бота, визначення обробника. Перевірка отриманої інформації та створення вихідного повідомлення
J-A-9	Оновлення стану бота	Стан бота	Отримання попереднього стану бота та оновлення його

Фрагмент коду реалізації тесту J-A-4:

```
@Test
void applyingForVacancyThenSuccessfullyAdded() {
when(jobRepositoryMock.findByUniqueId("34f2s")).thenReturn(job);
String jobId = inputMessage.getText();
Job choosedJob = jobRepositoryMock.findByUniqueId(jobId);
List<User> users = choosedJob.getRequested();
users.add(user);
choosedJob.setRequested(users);
when(jobRepositoryMock.save(choosedJob)).thenReturn(choosedJob);
assertThat(jobRepositoryMock.save(choosedJob).getRequested().contai
ns(user));}
```

Системне тестування буде виконуватись відповідно до інструкції користувача та функціональних можливостей ПЗ. Деякі тестові сценаріїв системного тестування наведені в таблиці 4.2.

Таблиця 4.2 – Тестові сценарії системних тестів

Ідетифікатор	Модуль	Вихідні дані	Очікуваний результат
1	2	3	4
JL-1	Головне меню	1. Натиснути на кнопку «Знайти роботу» 2. Натиснути на кнопку «Знайти працівників» 3. Натиснути на кнопку «Додатково»	1.Відображається меню Знайти роботу 2.Відображається меню Знайти роботу 3.Відображається меню Додатково
JL-2	Пошук публікацій	1.Натиснути кнопку «Пошук по категоріям» 2.Номер категорії	1.Відображається список категорій для вибору 2.Список вакансії вибраної категорії
JL-3	Заповнення резюме	1.Натиснути кнопку «Заповнити резюме» 2.Ввести некоректні дані	1.Запитання для користувача 2.Повідомлення про некоректний ввід
JL-4	Відображення публікацій	1. Натиснути кнопку «Відобразити всі» 2. Натиснути кнопку «Відобразити всі підтверджені» 3.Натиснути кнопку «Відобразити всі відхилені»	1. Список всіх публікацій 2. Список всіх підтведжених публікацій 3. Список всіх відхилених публікацій 3.
JL-5	Модерувати вакансію	1. Натиснути на кнопку «Підтвердити» 3. Натиснути на кнопку “Відхилити»	1. З’являється наступна вакансія 4.З’являється запитання про причину відхилення

4.3 Аналіз результатів тестування системи

Інформацію про правильність виконання тестів можна отримати за допомогою запуску збиральника проектів Gradle, який автоматично запустить тести при складанні проекту або вручну запустити їх на виконання за допомогою

інтегрованого середовища для розробки – Eclipse. Запуск тестів на обробку був виконаний за допомогою IDE та поданий на рисунку 4.1.

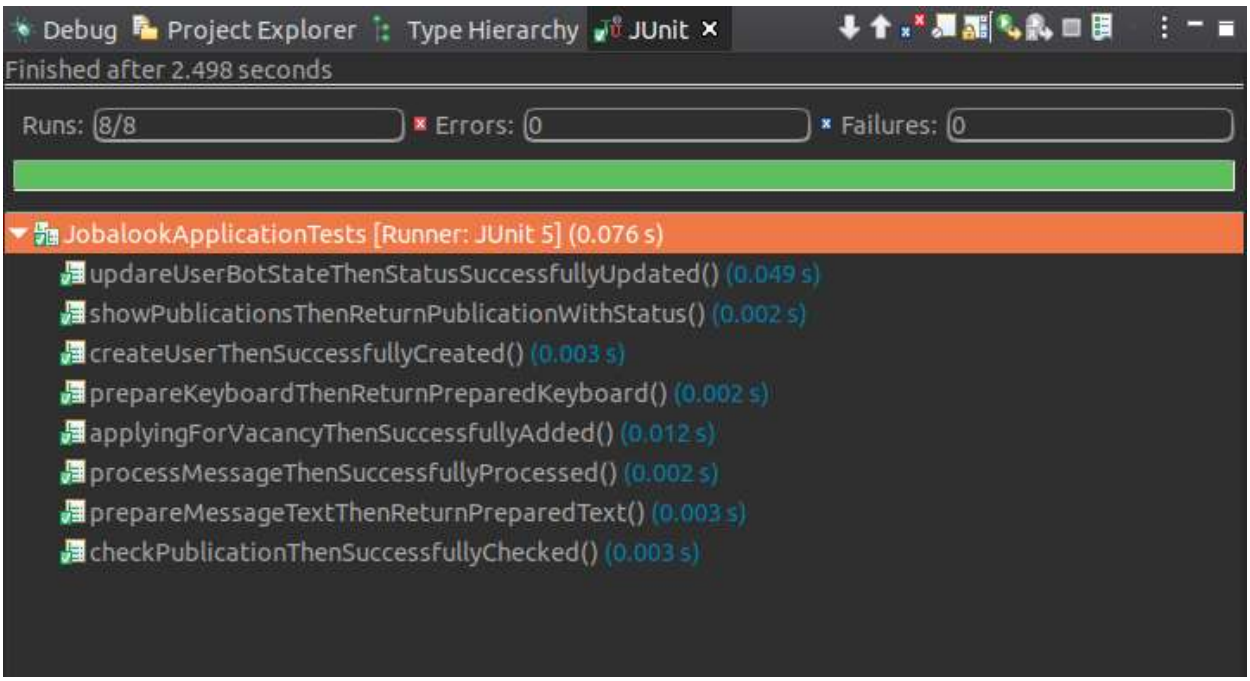


Рисунок 4.1 – Результати виконання модульних тестів

Результати наведених тестових сценаріїв наведені в таблиці 4.3.

Таблиця 4.3 – Результати тестових сценаріїв

Ідентифікатор	Опис	Вхідні значення	Вихідні значення	Результат
1	2	3	4	5
JC-1	Пошук публікацій	1	IT, комп'ютери, інтернет	Правильно
JC-2	Створення користувача	Відсутні	Ідентифікатор користувача та чату	Правильно
JC-3	Відображення публікацій	Підтверджені	Список підтверджених публікацій	Правильно
JC-4	Створення користувача	Відсутні	Ідентифікатор користувача та чату	Правильно

Кінець таблиці 4.3

1	2	3	4	5
JC-5	Відображення публікацій	Підтверджені	Список підтверджених публікацій	Правильно
JC-6	Подача заявки на вакансію	3b4ad	В заявки до вакансіїї додався користувач та був переправлений на головне меню	Правильно
JC-7	Модерація вакансії	Підтвердити	Публікація міняє статус на переглянуту та підтверджену, модератору відображає наступну вакансію	Правильно
JC-8	Підготовка тексту повідомлення	Створення та конфігуряція тексту, 454363	Сформований текст	Правильно
JC-9	Підготовка клавіатури для повідомлення	Створення та конфігурація кнопок та тексту, 3b4ad	Створена клавіатура	Правильно
JC-10	Обробка повідомлення	Текстове повідомлення користувача	Отримання стану бота, визначення обробника. Перевірка отриманої інформації та створення вихідного повідомлення	Правильно

В даному розділі було вибрано підхід до тестування ПЗ. В результаті проведення модульного тестування відповідно до функціональних вимог, було підтверджено коректну роботу усіх модулів ПЗ та те, що програмне забезпечення являється повність працездатним.

ВИСНОВКИ

При виконанні дипломного проекту у першому розділі було проведено аналіз предметної області та встановлено, що Telegram боти – популярні сервіси, але не достатньо розвинуті в Україні, що відіграють значну роль у сучасному житті людини, дають хороші перспективи для швидкого розвитку бізнесу та суттєво зменшують час на рутинні справи. Усі вони повинні бути продуктивними, зручними та легкими у користуванні. Також було проведено аналіз існуючого програмного забезпечення предметної області, в результаті якого визначено, що існує багато подібних програмних систем і подібний бот, усі вони відрізняються своїм зовнішнім виглядом, типом, функціональними можливостями та зручністю у користуванні. Більшість сайтів мають вже більшість необхідних функціональних можливостей але Telegram боти відстають в цьому плані. В результаті були визначені функціональні та нефункціональні вимоги до програмного забезпечення та встановлено і описано основні варіанти використання системи.

У другому розділі було проведено аналіз сучасних архітектур, розглянуто переваги та недоліки монолітного та мікросервісного стилю побудови додатку та визначено, що монолітний стиль буде кращим вибором, оскільки розроблюване ПЗ не буде розбито на компоненти, а буде розроблено в єдиній кодовій базі. Також було проаналізовано типи баз даних порівняно їх та вирішено, що для реалізації даного проекту краще використовувати нереляційну базу даних MongoDB через її зручність у використанні та швидкість виконання запитів. Було визначено, що для написання додатку буде використано мову програмування Java. Додатково буде використовуватись фреймворк Spring і його додаткові модулі, а також систему автоматичного складання проектів Gradle.

У третьому розділі було підготовлено всі залежності для написання коду, виконано практичну розробку програмних модулів і описано їх особливості та функціональні можливості за допомогою фрагментів коду. Було створено

інструкцію користувача для взаємодії з інтерфейсом програми, а також описаний процес розгортання, встановлення і зупинення програмного забезпечення.

У розділі тестування було вибрано підхід до тестування ПЗ. В результаті проведення модульного тестування відповідно до функціональних вимог, було підтверджено коректну роботу усіх модулів ПЗ та те, що програмне забезпечення являється повність працездатним.

Під час використання програмного забезпечення користувач зможе замінити нудні вечори та години потраченого часу на пошук потрібного йому оголошення про роботу або кваліфікованого працівника, на зручний, швидкий та зрозумілий інтерфейс Telegram-бота. Користувачі зможуть користуватись ботом на різноманітних ОС та швидко отримувати доступ до бота за допомогою мобільного телефону, планшету та інших ґаджетів.

Дане програмне забезпечення можна розширити додавши можливість отримання різноманітних інформаційних сповіщень, спілкування в чаті бота, а також створивши систему для розміщення реклами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Лада Шамардина. Telegram впервые обошел по числу пользователей Snapchat и Twitter [Электроный ресурс] / Л. Шамардина // THE BELL – Режим доступа: https://thebell.io/telegram-vpervye-obognal-snapchat-po-kolichestvu-polzovatelej.

2. What is Telegram? What do I do here? [Online] Telegram FAQ / – Available: https://telegram.org/faq#q-what-is-telegram-what-do-i-do-here

3. One platform for your team and your work [Online] / Slack messenger. – Available: https://slack.com/intl/en-ua/features

4. Alexander Meshcheryakov. Боты в Telegram что это и как они работают [Электроный ресурс] / A. Meshcheryakov // Shark Develop. – Режим доступа: https://sharkdevelop.com/boty-v-telegram/

5. Есть ли жизнь без смартфона: мнение украинцев [Электроный ресурс] / Research & Branding Group. – Режим доступа: http://rb.com.ua/blog/est-li-zhizn-bez-smartfona-mnenie-ukraincev

6. Ankit Jain. Unified Modeling Language [Online] / A. Jain // GeeksforGeeks. – Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

7. What is Use Case Diagram? [Online] / Visual Paradigm. – Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/ what-is-use-case-diagram/

8. Client-Server Model [Online] / Researchgate. – Available: https://www .researchgate.net/publication/271295146_Client-Server_Model

9. Thin Client vs Thick Client [Online] / Simplicable. – Available: https:// simplicable.com/new/thin-client-vs-thick-client

10. What’s a Webhook? [Online] / SendGrid. – Available: https://sendgrid .com/blog/whats-Webhook

11. Zell Liew. Understanfing and Using REST APIs [Online] / Z. Liew // Smashing Magazine. – Available: https://www.smashingmagazine.com/2018 /0 1/understanding-using-rest-api

12. Ivy Wigmore. Monolithic architetrue [Online] / I. Wigmore // Whatls. – Available: https://whatis.techtarget.com/definition/monolithic-architecture

13. Concepts of Three-Tier Architecture [Online] / Net Informations. – Available: http://net-informations.com/q/mis/3tier.html

14. Chris Richardson. Microservice Arhitecture [Online] / C. Richardson // Microservices IO. – Available: https://microservices.io/

15. Mark Drake. Understanding Relational Databases / M. Drake // Digital Ocean – Available: https://www.digitalocean.com/community/tutorials/ understanding-relational-databases

16. Dikshay Poojary. Type of nosql databases and its comparison with relational databases [Online] / D. Poojary // ResearchGate. – Available: https://www.researchgate.net/publication/302557703_Article_Type_of_nosql_databases_and_its_comparison_with_relational_databases

17. Vitaliy Ilyukha. How to Choose Between MongoDB vs MySQL? [Online] / V. Ilyukha // Jelvix. – Available: https://jelvix.com/blog/mongodb-vs-mysql

18. Matthew Jones . The Repository-Service Pattern with DI and ASP.NET Core [Online] / M. Jones // Exception not found. – Available: https://exceptionnot found .net/the-repository-service-pattern-with-dependency-injection-and-asp-net-core

19. 10 Key Diffetences between Java and C# [Online] / Guru99. – Available: https://www.guru99.com/java-vs-c-sharp-key-difference.html

20. Spring Framework Overview [Online] / Spring Docs. – Available: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html

21. Spring Projects [Online] / Spring. – Available: https://spring.io/projects

22. Gradle vs Maven Comparsion [Online] / Gradle Build Tool. – Available: https://gradle.org/maven-vs-gradle

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки Telegram-бота для пошуку роботи та підбору персоналу, що дозволяє створювати, шукати та переглядати відповідні оголошення.

1 Підстава для розробки

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

Найменування розробки: «Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота».

2 Призначення розробки

Telegram-бот призначений для пошук роботи та/або підбору персоналу.

Користувачами програмного забезпечення є звичайні користувачі месенджера Telegram.

Функціональне призначення ПЗ передбачає:

− реєстрацію у вигляді заповнення особистих даних, редагування та видалення особистих даних;

− пошук оголошень, попередню фільтрацію та сортування оголошень, створення оголошень;

− публікацію резюме, видалення та редагування створених оголошень/резюме;

− додавання нових розділів до системи, редагування та видалення створених та існуючих;

− попередню модерація вмісту користувацького оголошення та редагування поля «ім’я користувача» (доступно тільки користувачу з правами модератора);

− видалення та блокування користувача (доступно тільки адміністратору).

Експлуатаційне призначення передбачає, що система може використовуватися на будь-якому ПК з попередньо встановленим Telegram, у

будь-якому браузері та у будь-якому смартфоні (з попередньо встановленим Telegram). Додаткових налаштувань додаток не потребує, щоб почати користуватись ботом.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Програмне забезпечення для пошуку роботи та підбору персоналу у вигляді Telegram-бота має виконувати всі функції, які подані нижче.

Функція реєстрації в системі

Новий користувач при запуску бота повинен заповнити свої персональні дані, які в подальшому будуть використанні для публікації оголошення та зв’язку з цим користувачем. Введені дані користувачем проходять етап валідації; у форму реєстрації входять наступні поля:

– ім’я;

– вік;

– стать;

– адреса електронної пошти.

Функція редагування особистих даних

Користувач, який успішно пройшов стадію заповнення особистих даних, може заповнити їх заново.

Функція пошуку оголошення

При пошуку оголошень користувач може знайти оголошення за категоріями.

Функція створення оголошення

Оголошення може бути опубліковане лише тоді, коли користувач пройде стадію заповнення особистих даних та коли модератор попередньо перевірить його контент на коректність. Якщо вміст оголошення порушує зазначені правила публікації, модератор має не допустити оголошення до публікації.

Функція редагування та видалення оголошення

Оголошення, яке ще не пройшло перевірку медератором, може бути відредаговане. Також користувач може видалити оголошення.

Функцію відображення акаунта, який опублікував оголошення

Для того, щоб отримати контакти користувача, який шукає роботу, користувачу який шукає персонал, потрібно вибрати конкретну вакансію та пункт відображення всіх заяв на співбесіду від користувачів, які їх подали; роботодавцю у меню конкретної вакансії відобразиться посилання на Telegram-акаунт користувача та інші контактні дані і він зможе йому написати для обговорення подальшої співпраці.

Повинен бути створений зручний та інтуїтивно зрозумілий інтерфейс користувача, який динамічно буде створювати бот.

Повинен бути створений додатковий інтерфейс для додаткових функцій, таких як модерація оголошення.

3.2 Вимоги до надійності

ПЗ має задовольняти наступним вимогам до надійності:

– користувачі мають мати конкретні права на доступ до певних функціональних можливостей сервісу;

– обробка внутрішньо-програмних помилок та зрозуміле повідомлення про помилку за потреби;

– валідація вхідних та вихідних даних на стороні сервера;

– захист від спам-атаки.

3.3 Умови експлуатації

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 ГОСТ 15150-69.

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

3.4 Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги для функціонування системи на комп’ютері, планшеті чи смартфоні повинні відповідати наступним:

– операційна система: Linux Ubuntu 18.04;

– розрядність: 32біт або 64 біт (x86 або x64);

– процесор: Intel Celeron J1800(2.41 ГГц);

– оперативна пам’ять 512 Мб;

– системна плата: ASRock 775i945GZ;

– жорсткий диск: HDD Samsung 200 Gb 7200 rpm;

– Інтернет: стабільне з’єднання;

– контролер: клавіатура, миша;

– роздільна здатність екрану: SVGA 800x600.

3.5 Вимоги до інформаційної та програмної сумісності

Для створення серверної частини будуть використовуватися технології:

– Java – строго типізована об’єктно-орієнтована мова програмування високого рівня;

– Spring – фреймворк, який виступає альтернативою і доповненням до моделі Enterprise JavaBean;

– Telegram API – призначене використовування функціональних можливостей месенджера Telegram;

– MongoDB – документно-орієнтована система керування базами даних, яка не потребує опису схеми таблиць;

– Docker – інструментарій для управління ізольованими Linux-контейнерами.

3.6 Спеціальні вимоги

Програма повинна мати зручний, сучасний та інтуїтивно зрозумілий для будь-якого користувача інтерфейс.

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 програмного забезпечення для пошуку роботи та підбору персоналу наведено в таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
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
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
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

Кінець таблиці А.1

1	2	3
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 ПЗ і документацію

6 Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами ПЗ, підключеними на етапі його тестування.

ДОДАТОК Б

(обов’язковий)

ДІАГРАМИ КЛАСІВ ПРОГРАМИ

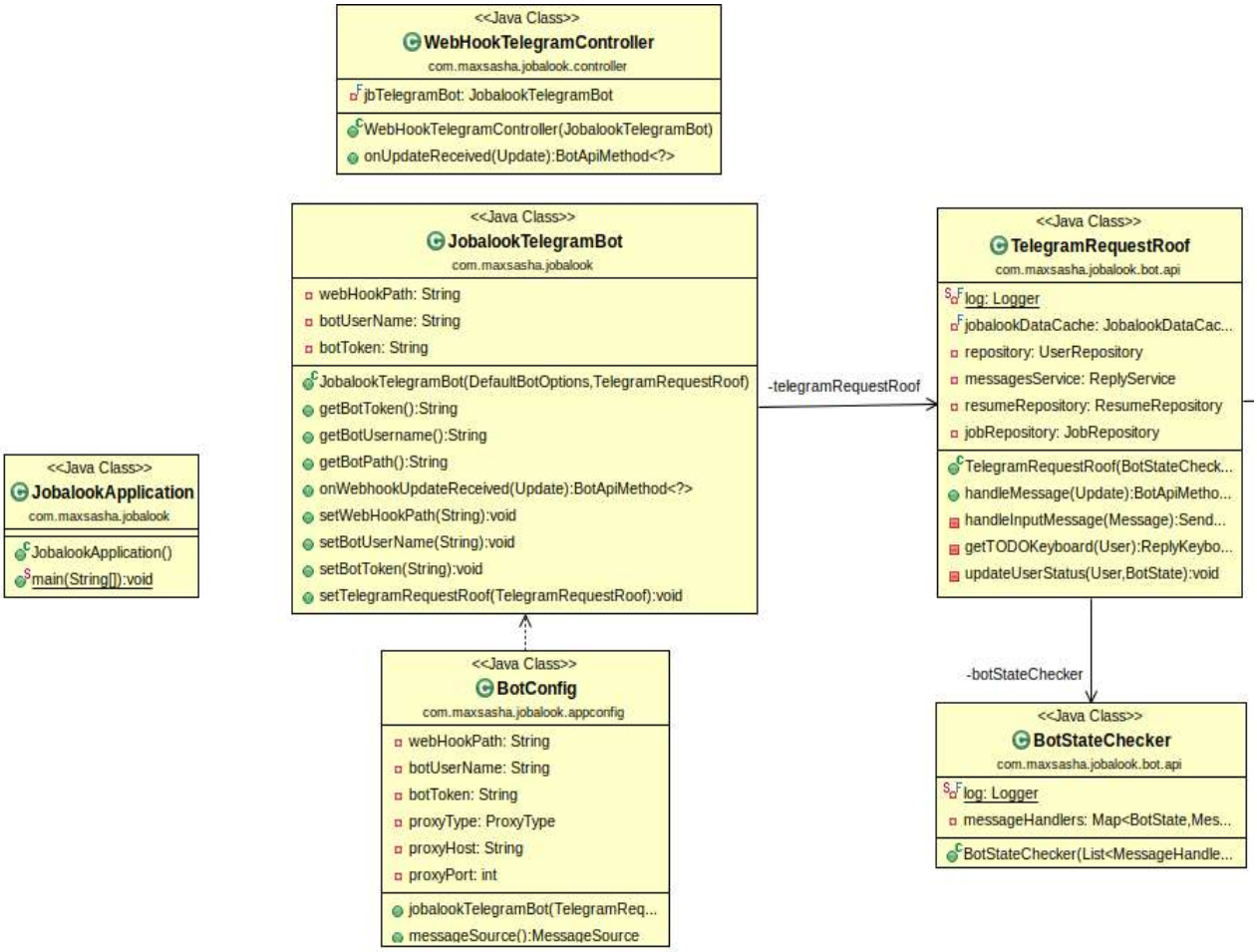


Рисунок Б.1 – Діаграма класів для отримання оновлень та конфігурації бота

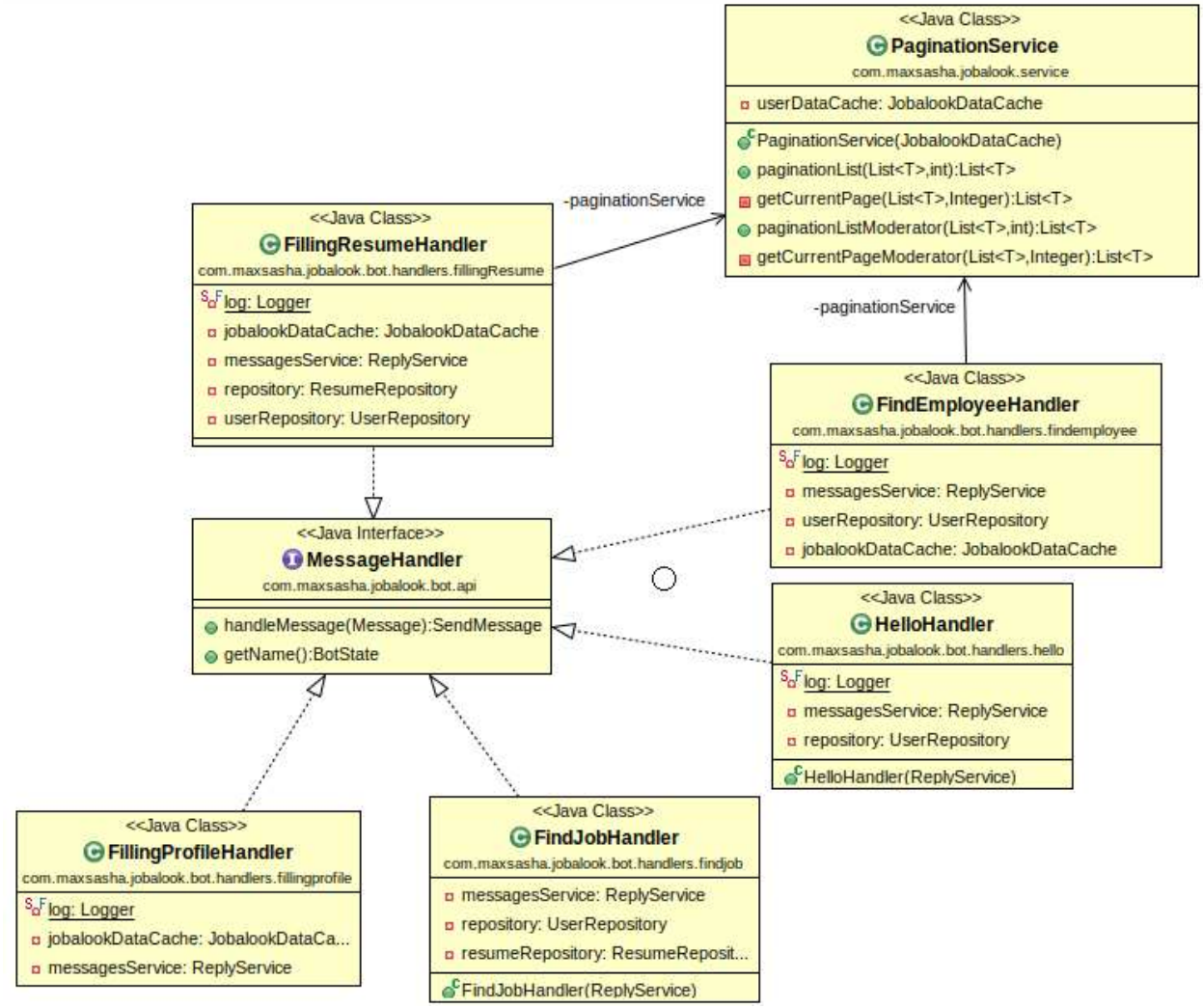


Рисунок Б.2 – Діаграма класів-обробників для повідомлень користувача

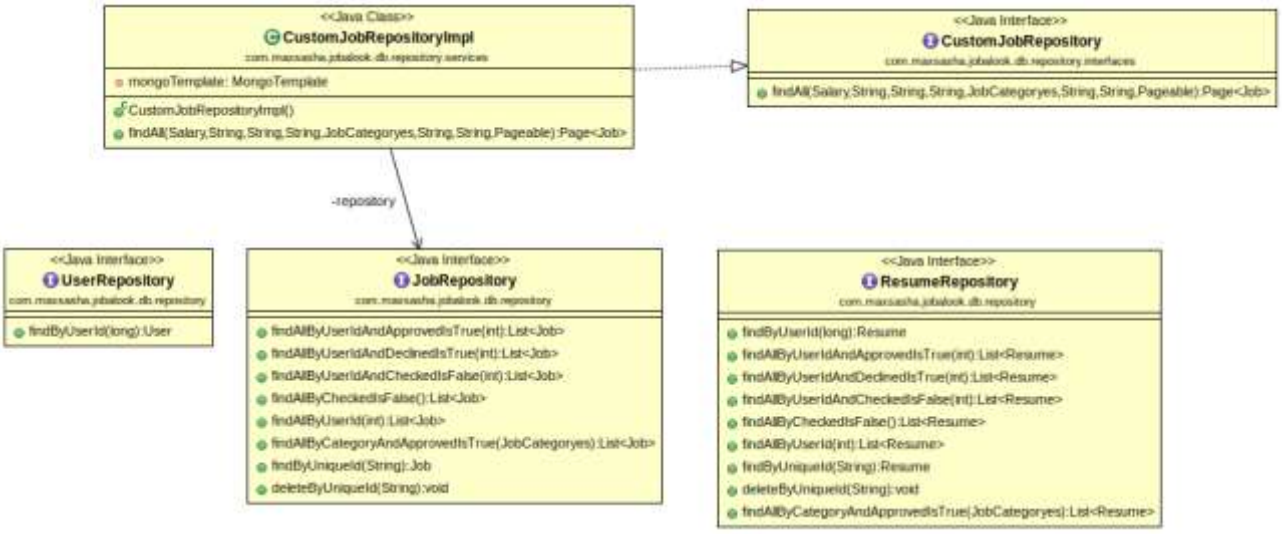


Рисунок Б.3 – Діаграма класів репозиторіїв

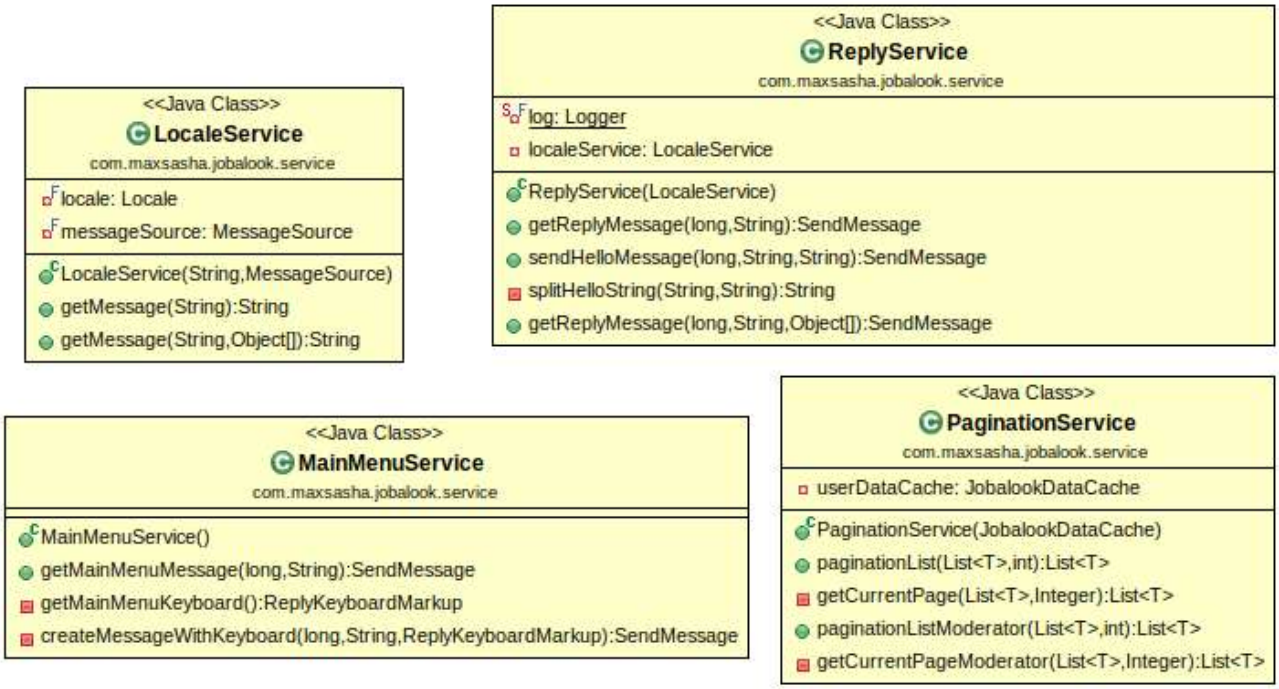


Рисунок Б.4 – Діаграма класів сервісів

ДОДАТОК В

(обов'язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

В.1 Програмний код програмного забезпечення

Клас JobalookApplication

```
package com.maxsasha.jobalook;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class JobalookApplication {
    public static void main(String[] args) {
    SpringApplication.run(JobalookApplication.class, args);}}
```

Клас JobalookTelegramBot

```
package com.maxsasha.jobalook;
//імпортування бібліотек для кофігурації фреймворка Spring
//імпортуваня бібліотек для взаємодії з Telegram
import com.maxsasha.jobalook.bot.api.TelegramRequestRoof;
import lombok.Setter;
@Setter public class JobalookTelegramBot extends TelegramWebhookBot {
private String webHookPath; private String botUserName;
private String botToken; private TelegramRequestRoof TelegramRequestRoof;
public JobalookTelegramBot(DefaultBotOptions defaultBotOptions,
TelegramRequestRoof TelegramRequestRoof) {
super(defaultBotOptions); this.TelegramRequestRoof = TelegramRequestRoof; }
@Override public String getBotToken() { return botToken; }
@Override public String getBotUsername() { return botUserName; }
@Override public String getBotPath() {  return webHookPath; }
@Override public BotApiMethod<?> onWebhookUpdateReceived(Update update) {
final BotApiMethod<?> replyMessageToUser =
TelegramRequestRoof.handleUpdate(update); return replyMessageToUser;}}
```

Клас BotConfig

```
package com.maxsasha.jobalook.appconfig;
//імпортування бібліотек для кофігурації фреймворка Spring
```

```
import
org.springframework.context.support.ReloadableResourceBundleMessageSource;
//імпортуваня бібліотек для взаємодії з Telegram
import com.maxsasha.jobalook.JobalookTelegramBot;
import com.maxsasha.jobalook.bot.api.TelegramRequestRoof; import lombok.Data;
@Data @Configuration @ConfigurationProperties(prefix = "telegrambot")
public class BotConfig {
    private String webHookPath; private String botUserName;
    private String botToken; private DefaultBotOptions.ProxyType proxyType;
    private String proxyHost; private int proxyPort;
    @Bean public JobalookTelegramBot jobalookTelegramBot(TelegramRequestRoof
        telegramRequestRoof) { DefaultBotOptions defaultBotOptions =
        ApiContext.getInstance( DefaultBotOptions.class);
        defaultBotOptions.setProxyHost(proxyHost);
        defaultBotOptions.setProxyPort(proxyPort);
        defaultBotOptions.setProxyType(proxyType);
        JobalookTelegramBot jobalookTelegramBot = new
        JobalookTelegramBot(defaultBotOptions, telegramRequestRoof);
        jobalookTelegramBot.setBotUserName(botUserName);
        jobalookTelegramBot.setBotToken(botToken);
        jobalookTelegramBot.setWebHookPath(webHookPath);
        return jobalookTelegramBot; }
    @Bean public MessageSource messageSource() {
        ReloadableResourceBundleMessageSource resourceMessageSource = new
        ReloadableResourceBundleMessageSource();
        resourceMessageSource.setBasename("classpath:messages");
        resourceMessageSource.setDefaultEncoding("UTF-8"); return
messageSource; } }
```

Клас BotStateChecker

```
package com.maxsasha.jobalook.bot.api;
import java.util.HashMap; import java.util.List; import java.util.Map;
import org.springframework.stereotype.Component;
//імпортуваня бібліотек для повідомлень Telegram
import lombok.extern.slf4j.Slf4j;
@Component @Slf4j public class BotStateChecker {
    private Map<BotState,MessageHandler>messageHandlers=new HashMap<>();
    public BotStateChecker(List<MessageHandler> messageHandlers) {
    messageHandlers.forEach(handler ->
    this.messageHandlers.put( handler.getName(),handler)); }
public SendMessage processMessage(BotState state, Message message) {
    MessageHandler currentHandler = findHandler(state);
    return currentMessageHandler.handle(message); }
private MessageHandler findMessageHandler(BotState state) {
    log.info("grabbing handler for user, state: {}", state);
    if (isFillingProfileState(state)) {
        return messageHandlers.get(BotState.FILLING_PROFILE);}
    else if (isFillingResumeState(state)) {
        return messageHandlers.get(BotState.FILLING_RESUME);}
    else if (isFindingJobState(state)) {
        return messageHandlers.get(BotState.SHOW_CATEGORIES);}
    else if (isFindJobDoState(state)) {
        return messageHandlers.get(BotState.FIND_JOB); }
    else if (isEditResumeState(state)) {
        return messageHandlers.get(BotState.EDIT); }
    else if (isFillingJobState(state)) {
        return messageHandlers.get(BotState.FIND_EMPLOYEE); }
```

```
    else if(isFillingEditState(state)) {
        return messageHandlers.get(BotState.SHOW_SETTINGS_MENU); }
    else if(isModeratorState(state)) {
        return messageHandlers.get(BotState.SHOW_MODERATOR_MENU); }
    return messageHandlers.get(state); }
private boolean isFillingProfileState(BotState state) {
    switch (state) { case ASK_GENDER: case ASK_NAME: case ASK_AGE:
    case ASK_EMAIL: case FILLING_PROFILE: case PROFILE_FILLED: return true;
    default: return false; } } private boolean isModeratorState(BotState
    state) { switch (state) { case SHOW_MODERATOR_MENU: case
    SHOW_VACANCY: case VACANCY_CHOOSING: case VACANCY_APPROVED: case
    VACANCY_DECLINED: case VACANCY_BLOCKING: case VACANCY_FILL_EXPLAINING:
    return true; default: return false; } }
private boolean isFillingEditState(BotState state) {
    switch (state) { case SHOW_HELP_MENU: case SHOW_MY_PROFILE: case
    SHOW_SETTINGS_MENU: return true; default: return false; } }
private boolean isFillingResumeState(BotState state) {
    switch (state) { case ASK_MIDDLE_NAME: case SHOW_MY_RESUMES:
    case SHOW_MY_RESUMES_APPROVED: case SHOW_MY_RESUMES_DECLINED:
    case SHOW_MY_RESUMES_UNCHECKED: case ASK_CAREER_OBJECTIVE:
    case ASK_DESIRED_INCOME_LEVEL_FROM: case ASK_DESIRED_INCOME_LEVEL_TO:
    case ASK_DATE_OF_BIRTH: case ASK_NUMBER: case ASK_SKILL:
    case ASK_EMPOYMENT_HISTORY: case ASK_EDUCATION: case FILLING_RESUME:
    case CHOOSING_RESUME: case RESUME_FILLED: return true; default: return
    false;}  }
private boolean isFillingJobState(BotState state) {
    switch (state) { case SHOW_MY_JOBS: case SHOW_MY_JOBS_APPROVED:
    case SHOW_MY_JOBS_DECLINED: case SHOW_MY_JOBS_UNCHECKED:
    case SHOW_MY_JOBS_REQUESTS: case ASK_PLACING_CATEGORY:
    case ASK_PAYMENT_LEVEL_FROM: case ASK_PAYMENT_LEVEL_TO:
    case ASK_PAYMENT_LEVEL_TO_FILLING: case ASK_TITLE:
    case ASK_DECRIPTION: case ASK_WORK_PLACE: case ASK_EMPOYMENT:
    case ASK_EXPERIANCE: case FILL_JOB: case CHOOSING_JOB:
    case JOB_FILLED: case FIND_EMPLOYEE: return true; default: return false;}}
private boolean isEditResumeState(BotState state) {
    switch (state) {case EDIT_RESUME_SKILLS: case
    EDIT_RESUME_SKILLS_FILLING: case EDIT_RESUME_EMPOYMENT_HISTORY:
    case EDIT_RESUME_EMPOYMENT_HISTORY_FILLING:
    case EDIT_RESUME_EDUCATION: case EDIT_RESUME_EDUCATION_FILLING:
    case EDIT_JOB_SALARY_FROM: case EDIT_JOB_SALARY_FROM_FILLING:
    case EDIT_JOB_SALARY_TO: case EDIT_JOB_SALARY_TO_FILLING:
    case EDIT_JOB_TITLE: case EDIT_JOB_TITLE_FILLING:
    case EDIT_JOB_DESRIPTION: case EDIT_JOB_DESRIPTION_FILLING: case EDIT:
    return true; default: return false;} }
private boolean isFindingJobState(BotState state) {
    switch (state) { case SHOW_CATEGORIES: case FILLING_CATEGORY: case
    SHOW_CATEGORY_JOBS:case FILLING_JOB: return true; default: return false; } }
private boolean isFindJobDoState(BotState state) {
    switch (state) { case FIND_JOB_TO_DO:
    case FIND_JOB: return true; default: return false;} }}
```

Інтерфейс MessageHandler

```
package com.maxsasha.jobalook.bot.api;
//імпортування бібліотек для повідомлень Telegram
public interface MessageHandler { SendMessage handleMessage(Message message);
    BotState getName(); }
```

Клас JobalookDataCache

```
package com.maxsasha.jobalook.cache;
//імпортування стандартних Java утиліт
//імпортування бібліотек для ін’єктії залежностей
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.handlers.fillingprofile.UserProfileData;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.enums.JobCategoryes;
import com.maxsasha.jobalook.transformer.UserTransformer;
import lombok.extern.slf4j.Slf4j;
@Component @Slf4j
public class JobalookDataCache { private Map<Integer, BotState> usersBotStates =
Collections.emptyMap();
private Map<Integer,Integer> usersCurrentPageSize = Collections.emptyMap();
private Map<Integer,JobCategoryes> usersCurrentCategory=Collections.emptyMap();
private Map<Integer, String> usersLastMessage = Collections.emptyMap();
private Map<Integer, BotState> usersLastState = Collections.emptyMap();
@Autowired private UserRepository repository;
public void setUsersCurrentBotState(int userId, BotState botState) {
    usersBotStates.put(userId, botState); }
public BotState getUsersCurrentBotState(int userId) {
    BotState botState=(BotState)this.usersBotStates.get(userId);return
botState; }
public String getLastMessage(int userId) {
    String lastMessage = usersLastMessage.get(userId); return lastMessage; }
public void setUsersLastMessage(int userId, String lastName) {
      usersLastMessage.put(userId, lastName); }
public String getLastUniqueId(int userId) { String lastMessage =
    usersLastMessage.get(userId); return lastMessage; }
public void setUsersLastUniqueId(int userId, String uniqueId) {
     usersLastMessage.put(userId, uniqueId); }
public void setUsersCurrentPageSize(int userId, Integer currentPage) {
      log.info("Update current page for user: {}", userId);
      usersCurrentPageSize.put(userId, currentPage); }
public void setUsersCurrentPageSize(int userId, JobCategoryes category) {
      usersCurrentCategory.put(userId, category); }
public void setUsersLastState(int userId, BotState state) {
      usersLastState.put(userId, state); }
public BotState getUsersLastState(int userId) {
      return usersLastState.get(userId); }
public JobCategoryes getCurrentCategory(int userId) { try{ JobCategoryes
      category = usersCurrentCategory.get(userId);
if(category == null) {return JobCategoryes.IT; } return category; }
catch (Exception ex) { return JobCategoryes.IT; } }
public Integer getUserCurrentPage(int userId) {
try { Integer userCurrentPage = usersCurrentPageSize.get(userId);
    if(userCurrentPage==null){ return 0;}
    return userCurrentPage;
} catch (Exception ex) { return 0; } } }
```

Клас WebHookTelegramController

```
package com.maxsasha.jobalook.controller;
// імпортування бібліотек для роботи з Spring web
```

```
//імпортуваня бібліотек для Telegram API
import com.maxsasha.jobalook.JobalookTelegramBot;
@RestController public class WebHookTelegramController {
private final JobalookTelegramBot jbtelegramBot;
public WebHookController(JobalookTelegramBot jbtelegramBot) {
    this.jbtelegramBot = jbtelegramBot; }
@RequestMapping(value = {"/"}, method = { RequestMethod.POST } )
public BotApiMethod<?> onUpdateReceived(@RequestBody Update update) {
    return jbtelegramBot.onWebhookUpdateReceived(update); } }
```

Клас LocaleService

```
package com.maxsasha.jobalook.service;
import java.util.Locale;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.MessageSource;
import org.springframework.stereotype.Service;
@Service public class LocaleService {
private final Locale locale;
private final MessageSource messageSource;
public LocaleMessageService(@Value("${localeTag}") String localeTag,
MessageSource messageSource) {
    this.messageSource = messageSource;
    this.locale = Locale.forLanguageTag(localeTag);
}
public String getMessage(String message) {
    return messageSource.getMessage(message, null, locale);
}
public String getMessage(String message, Object... args) {
    return messageSource.getMessage(message, args, locale); }
}
```

Клас MainMenuService

```
package com.maxsasha.jobalook.service;
import org.springframework.stereotype.Service;
//імпортування бібліотек Telegram клавіатур
import java.util.ArrayList; import java.util.List;
@Service public class MainMenuService {
public SendMessage getMainMenuMessage(final long userChatId, final String
textMessage) {
    final ResponseKeyboard responseKeyboard = getMainMenuKeyboard(); final
    SendMessage mainMenuMessage =
    createMessageWithKeyboard(userChatId, textMessage, responseKeyboard);
    return mainMenuMessage;
}
private ResponseKeyboard getMainMenuKeyboard() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(false);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
KeyboardRow row_two = new KeyboardRow();
KeyboardRow row_three = new KeyboardRow();
```

```
row_one.add(new KeyboardButton("Знайти роботу"));
row_two.add(new KeyboardButton("Знайти працівників"));
row_three.add(new KeyboardButton("Додатково"));
keyboardRow.add(row_one); keyboardRow.add(row_two); keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard; }
private SendMessage createMessageWithKeyboard(final long userChatId, String
textMessage, final ResponseKeyboard responseKeyboard) {
SendMessage responseMessage = new SendMessage();
responseMessage.enableMarkdown(true); sendMessage.setChatId(chatId);
responseMessage.setText(textMessage); if (responseKeyboard != null) {
responseMessage.setReplyMarkup(responseKeyboard); } return responseMessage;} }
```

Клас PaginationService

```
package com.maxsasha.jobalook.service;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.Job;
import com.maxsasha.jobalook.db.entity.Resume;
import org.jvnet.hk2.annotations.Service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import javax.swing.*; import java.util.List;
@Component public class PaginationService {private JobalookDataDache
jobalookDataDache;
public PaginationService(JobalookDataDache jobalookDataDache)
{ this.jobalookDataDache = jobalookDataDache; }
public <T> List<T> paginationList(List<T> tList, int userId) {
    return getCurrentPage(tList, jobalookDataDache.getUserCurrentPage(userId));
}
private <T> List<T> getCurrentPage(List<T> classList, Integer pageNumber) {
    Integer fromIndex = pageNumber * 3; Integer to = fromIndex + 3;
    Integer toIndex = to > classList.size() ? classList.size() : to;
    return classList.subList(fromIndex, toIndex);
}
public <T> List<T> paginationListModerator(List<T> tList, int userId) {
return
getCurrentPageModerator(tList,jobalookDataDache.getUserCurrentPage(userId));}
private <T> List<T> getCurrentPageModerator(List<T> classList, Integer
pageNumber) {
    Integer fromIndex = pageNumber * 1; Integer to = fromIndex + 1;
    Integer toIndex = to > classList.size() ? classList.size() : to;
    return classList.subList(fromIndex, toIndex);}
}
```

Клас ReplyService

```
package com.maxsasha.jobalook.service;
//імпортування бібліотек для логування
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
@Service @Slf4j
public class ReplyService {
private LocaleService localeService;
```

```
public ReplyService(LocaleService messageService) {
    localeMessageService = messageService;
}
public SendMessage getReplyMessage(long userChatId, String replyMessage) {
return new SendMessage(userChatId,
localeMessageService.getMessage(replyMessage)); }
public SendMessage sendHelloMessage(long userChatId, String replyMessage, String
userFirstName) {
    return new SendMessage(userChatId,
 splitHelloString(localeMessageService.getMessage(replyMessage),userFirstName));
}
private String splitHelloString(String replyMessage, String userFirstName) {
    String[] split = replyMessage.split(","); String splitMessage =
    String.format(split[0] + " " + userFirstName + ", " + split[1]);
    log.info("Split hello message: {}", splitMessage);
    return splitMessage;
}
public SendMessage getReplyMessage(long userChatId, String replyMessage,
Object... args) {
    return new SendMessage(userChatId,
localeMessageService.getMessage(replyMessage,
    args)); }
}
```

Інтерфейс JobRepository

```
package com.maxsasha.jobalook.db.repository;
import com.maxsasha.jobalook.db.entity.Job;
import com.maxsasha.jobalook.db.entity.Resume;
import com.maxsasha.jobalook.enums.JobCategoryes;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface JobRepository extends MongoRepository<Job, String> {
List<Job> findAllByUserIdAndApprovedIsTrue(int userId);
List<Job> findAllByUserIdAndDeclinedIsTrue(int userId);
List<Job> findAllByUserIdAndCheckedIsFalse(int userId);
List<Job> findAllByCheckedIsFalse();
List<Job> findAllByUserId(int userId);
List<Job> findAllByCategoryAndApprovedIsTrue(JobCategoryes category);
Job findByUniqueId(String uniqueId);
void deleteByUniqueId(String uniqueId); }
```

Інтерфейс ResumeRepository

```
package com.maxsasha.jobalook.db.repository;
import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import com.maxsasha.jobalook.db.entity.Resume;
import com.maxsasha.jobalook.enums.JobCategoryes;
@Repository
public interface ResumeRepository extends MongoRepository<Resume, String> {
```

```
Resume findByUserId(long userId);
List<Resume> findAllByUserIdAndApprovedIsTrue(int userId);
List<Resume> findAllByUserIdAndDeclinedIsTrue(int userId);
List<Resume> findAllByUserIdAndCheckedIsFalse(int userId);
List<Resume> findAllByCheckedIsFalse();List<Resume> findAllByUserId(int userId);
Resume findByUniqueId(String uniqueId);void deleteByUniqueId(String uniqueId);
List<Resume> findAllByCategoryAndApprovedIsTrue(JobCategoryes jobCategory); }
```

Інтерфейс UserRepository

```
package com.maxsasha.jobalook.db.repository;
import com.maxsasha.jobalook.db.entity.User;
//імпортування бібліотек для репозиторіїв Spring
import org.springframework.stereotype.Repository;
@Repository
public interface UserRepository extends MongoRepository<User, String> {
User findByUserId(long userId); }
```

Клас JobTransformer

```
package com.maxsasha.jobalook.transformer;
import com.maxsasha.jobalook.db.entity.Job;
import com.maxsasha.jobalook.db.entity.Salary;
import com.maxsasha.jobalook.enums.JobCategoryes;
public class JobTransformer {
public static final Job transformToPreview(Job job) {
return job == null ? null :
Job.builder().title(job.getTitle()).description(job.getDescription()).salary(job
.getSalary()).category(job.getCategory()).employment(job.getEmployment()).experi
ence(job.getExperience()).workPlace(job.getWorkPlace()).build(); }
public static final Job transformToFound(Salary salary, String workPlace, String
experience, String employment, JobCategoryes category, String country, String
city) { return
Job.builder().salary(salary).category(category).employment(employment).experienc
e(experience).workPlace(workPlace).build(); } }
```

Клас UserTransformer

```
package com.maxsasha.jobalook.transformer;
import com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.handlers.fillingprofile.UserProfileData;
import com.maxsasha.jobalook.db.entity.User;
public class UserTransformer {
    public static final User transform(UserProfileData userProfileData) {
        return User.builder()
                .userName(userProfileData.getUserName())
                .userId(userProfileData.getUserId())
                .userChatId(userProfileData.getUserChatId())
                .name(userProfileData.getName())
                .age(userProfileData.getAge())
```

```
                .gender(userProfileData.getGender())
                .email(userProfileData.getEmail()).build();}
    public static final UserProfileData transform(User user) {
        return UserProfileData.builder()
                .userName(user.getUserName()).userId(user.getUserId())
                .userChatId(user.getUserChatId()).name(user.getName())
                .age(user.getAge()).gender(user.getGender())
                .email(user.getEmail()).build(); }
    public static final User transform(String userName, long userId, long
userChatId, BotState state) {
return
User.builder().userName(userName).userId(userId).userChatId(userChatId).state(st
ate).blackList(false).removed(false).build(); } }
```

Клас SettingsHandler

```
package com.maxsasha.jobalook.bot.handlers.settings;
import java.util.ArrayList; import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур
import com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.User;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.service.ReplyService;
@Component public class SettingsHandler implements MessageHandler {
    private JobalookDataCache jobalookDataCache;
    @Autowired private UserRepository repository;
    public SettingsHandler(JobalookDataDache jobalookDataDache)
{ this.jobalookDataCache = jobalookDataCache; }
    @Override public SendMessage handleMessage(Message message)
    { return processUsersInput(message); }
    @Override public BotState getName()
    { return BotState.SHOW_SETTINGS_MENU; }
    private SendMessage processUsersInput(Message inputMessage) {
      int userId = inputMessage.getFrom().getId();
      long userChatId = inputMessage.getChatId();
      User user = repository.findByUserId(userId);
      BotState state = user.getState();
      SendMessage responseForUser = new SendMessage();
      responseForUser.setChatId(userChatId);
        if(state.equals(BotState.SHOW_SETTINGS_MENU)) {
            responseForUser.setText("Виберіть пунк меню");
            responseForUser.setReplyMarkup(getSettingsMenu());}
        if(state.equals(BotState.SHOW_MY_PROFILE)) {
            StringBuilder profileBuilder = new StringBuilder();
            profileBuilder.append("*****************************\n Твій профіль
виглядяє так: \n");
            profileBuilder.append(String.format("Ім'я: %s\n", user.getName()));
            profileBuilder.append(String.format("Стать: %s\n",
user.getGender()));
            profileBuilder.append(String.format("Вік: %s\n", user.getAge()));
            profileBuilder.append(String.format("Електронна
пошта: %s\n",user.getEmail()));
            responseForUser.setText(profileBuilder.toString());
```

```
            responseForUser.setReplyMarkup(getEditProfileMenu());
            jobalookDataDache.setUsersLastState(userId,
BotState.SHOW_SETTINGS_MENU); }
        if(state.equals(BotState.SHOW_HELP_MENU)) {
            responseForUser.setText("Для того, щоб взаємодіяти з ботом потрібно
відправляти відповідь на запитання бота, або ж користуватись кнопками, які
розташовані"+"під полем введення тексту. За допомогою цього бота ти можеш: \n 1.
Знайти роботу"+"\n  * Знайти вакансію самостійно і відправити запит власнику
вакансію, після чого очікувати поки вам не напишуть(власнику вакансії буде
відображені ваші контактні дані)"+"\n  * Створити резюме, та розмістити його для
власників робочих мість(будуть відображені ваші контактні дані)" + "\n  *
Відобразити список ваших резюме(підтверджених, відхилених та неперевірених)" +
"\n 2. Знайти працівників" +"\n  * Знайти претендента самостійно використавши
пошук по категоріям, вам будуть відображені списки резюме та будуть дані
контактні дані для зв'язку" +"\n  * Створити вакансію, та розмістити її для
претендентів, які шукають роботу в конкретній категорії" + "\n  * Відобразити
список ваших вакансій(підтверджених, відхилених та неперевірених). Ви моежете
переглянути підтверджені вакансії, після вибору однієї з них вам" + "буде
відображено меню в якому можна буде натиснути кнопку \"Відобразити список
запитів\", натиснувши на цю кнопку вам буде відображено список претендентів(які
подали заявку) і їх контактні дані " + "для зв'язку з ними" +
"\n\n**********УВАГА*********\nВ даному боті присутня модерація, якщо ви будете
відправляти на перевірку не коректні резюме/вакансії " + "модератор має право на
відхилення даної вакансії з поясненням причини.\n**********УВАГА*********\n" +
"\nКонтактні дані автора бота: \n * Телеграм: @MaxSasha\n * Електронна пошта:
sashamaksimyv@ukr.net"); responseForUser.setReplyMarkup(getSettingsMenu()); }
return responseForUser; }
private ResponseKeyboard getSettingsMenu() {
    final ResponseKeyboard responseKeyboard=new ResponseKeyboard();
    responseKeyboard.setSelective(true);
    responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboardRow = new ArrayList<>();
    KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Відобразити мій профіль"));
    KeyboardRow row_two = new KeyboardRow();
    row_two.add(new KeyboardButton("Допомога"));
    KeyboardRow row_three = new KeyboardRow();
    row_three.add(new KeyboardButton("На головну"));
    keyboardRow.add(row_one);
    keyboardRow.add(row_two);
    keyboardRow.add(row_three);
    responseKeyboard.setKeyboard(keyboardRow);
    return responseKeyboard;
}
private ResponseKeyboard getEditProfileMenu() {
    final ResponseKeyboard responseKeyboard =
    new ResponseKeyboard();
    responseKeyboard.setSelective(true);
    responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboardRow = new ArrayList<>();
    KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Заповнити профіль заново"));
    KeyboardRow row_two = new KeyboardRow();
    row_two.add(new KeyboardButton("Назад"));
    keyboardRow.add(row_one);
    keyboardRow.add(row_two);
    responseKeyboard.setKeyboard(keyboardRow);
    return responseKeyboard;
 }
```

```
}
```

Клас FindByCategoryHandler

```
package com.maxsasha.jobalook.bot.handlers.findByCategory;
//імпортування стандартних Java утиліт
//імпортування бібліотек для ін’єктії залежностей
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур import
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.Job;
import com.maxsasha.jobalook.db.entity.Resume;
import com.maxsasha.jobalook.db.entity.User;
import com.maxsasha.jobalook.db.repository.JobRepository;
import com.maxsasha.jobalook.db.repository.ResumeRepository;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.enums.JobCategoryes;
import com.maxsasha.jobalook.service.PaginationService;
import com.maxsasha.jobalook.service.ReplyService;
@Component public class FindByCategoryHandler implements MessageHandler {
@Autowired private JobalookDataDache jobalookDataDache;
private ReplyService messagesService;
@Autowired private UserRepository repository;
@Autowired private ResumeRepository resumeRepository;
@Autowired private PaginationService paginationService;
@Autowired private JobRepository jobRepository;
public FindByCategoryHandler(ReplyService messagesService)
{ this.messagesService = messagesService; }
@Override public SendMessage handleMessage(Message message)
{ return processUsersInput(message); }
@Override public BotState getName()
{ return BotState.SHOW_CATEGORIES; }
private SendMessage processUsersInput(Message inputMessage) {
int userId = inputMessage.getFrom().getId();
long userChatId = inputMessage.getChatId();
User user = repository.findByUserId(userId);
BotState state = user.getState();
BotState lastState = user.getLastState();
if(state.equals(BotState.SHOW_CATEGORIES)) {
SendMessage responseForUser = messagesService.getReplyMessage(chatId,
"reply.sayHowToChoose");responseForUser =
createMessageWithCategories(responseForUser);
user.setState(BotState.FILLING_CATEGORY);
repository.save(user);
return responseForUser; }
if(state.equals(BotState.FILLING_CATEGORY)){
if(lastState.equals(BotState.FIND_JOB_TO_DO) ||
lastState.equals(BotState.FIND_JOB)) {
    user.setState(BotState.FILLING_JOB);
    repository.save(user); SendMessage responseForUser =
    createMessageWithJobs(userId, inputMessage.getText(), chatId);
    return responseForUser;
}
if(lastState.equals(BotState.FIND_EMPLOYEE)) { SendMessage responseForUser =
    createMessageWithResumes(userId, inputMessage.getText(), chatId);
    return responseForUser; }
```

```
}
if(state.equals(BotState.FILLING_JOB)) {
String jobId = inputMessage.getText();
Job choosedJob = jobRepository.findByUniqueId(jobId);
if(choosedJob==null)
{ return new SendMessage(chatId, "Ви ввели не коректне значення"); }
List<User> users = choosedJob.getRequested();
for (User item : users) {
     if(item.getUserName().equals(user.getUserName())) {
     return new SendMessage(chatId, "Ви вже подали заявку на розгляд");}}
     users.add(user);
     choosedJob.setRequested(users);
     jobRepository.save(choosedJob);
     user.setState(BotState.ASK_TODO);
     repository.save(user);
SendMessage requestMessageTrue = new SendMessage(chatId, "Ви успішно подали
заявку на розгляд"); requestMessageTrue.setReplyMarkup(getTODOKeyboard(user));
return requestMessageTrue; } SendMessage responseForUser =
messagesService.getReplyMessage(chatId, "reply.sayHowToChooseVacancy");
responseForUser = createMessageWithCategories(responseForUser);
return responseForUser; }
public SendMessage createMessageWithCategories(SendMessage responseForUser;) {
SendMessage newSendMessage = responseForUser;;
StringBuilder messageBuilder = new StringBuilder(sendMessage.getText()+"\n
" + "Список категорій\n**********************************\n");
int i=0; messageBuilder.append(String.format("№  | Назва\n"));
for (JobCategoryes category : JobCategoryes.values()){
    i++;
    if(i>=10){
    messageBuilder.append(String.format("%s  | %s\n",i , category.toString()));}
    else {messageBuilder.append(String.format("%s    | %s\n",i ,
category.toString()));
}}
messageBuilder.append("\n**********************************\n");
newSendMessage.setText(messageBuilder.toString()); return newSendMessage; }
private SendMessage createMessageWithJobs(int userId, String categoryNumber,
long chatId) {
SendMessage responseForUser = new SendMessage();
StringBuilder jobMessageBuilder = new StringBuilder();
Integer categoryNumb = getCategoryNumber(categoryNumber);
User currentUser = repository.findByUserId(userId);
if(categoryNumb==-1)
   {sendMessage.setText("Ви ввели не вірний номер категорії");return
sendMessage;}
sendMessage = messagesService.getReplyMessage(chatId,
"reply.sayHowToChooseVacancy");
JobCategoryes jobCategory = getCategory(categoryNumb);
List<Job> jobs = jobRepository.findAllByCategoryAndApprovedIsTrue(jobCategory);
if(jobs.size()==0) {
    SendMessage emptyListMessage = new SendMessage(chatId, "На жаль в цій
    категорії на даний момент немає заявок. ");
       if(resumeRepository.findAllByUserId(userId).size()==0){
emptyListMessage.setReplyMarkup(getResumeKeyboardWithoutShowResumes());}
else {emptyListMessage.setReplyMarkup(getResumeKeyboardWithShowResumes());}
currentUser.setState(BotState.FIND_JOB_TO_DO);
repository.save(currentUser); return emptyListMessage; }
List<Job> paginationJobs = paginationService.paginationList(jobs, userId);
for (Job job : paginationJobs) {
     JobMessageBuilder.append(prepareJobShow(job, job.getUniqueId())); }
     sendMessage.setText(sendMessage.getText()+jobMessageBuilder.toString());
     sendMessage.setReplyMarkup(getPaginationButtons(paginationJobs,
```

```
     jobalookDataDache.getUserCurrentPage(userId)));
     return sendMessage;}
private SendMessage createMessageWithResumes(int userId, String categoryNumber,
long chatId)
{ StringBuilder builder = new StringBuilder();
Integer categoryNumb = getCategoryNumber(categoryNumber);
SendMessage sendMessage = new SendMessage();
if(categoryNumb==-1)
{sendMessage.setText("Ви ввели не вірний номер категорії");return sendMessage;}
sendMessage = messagesService.getReplyMessage(chatId,
"reply.sayHowToChooseResume");
User currentUser = repository.findByUserId(userId);
JobCategoryes jobCategory = getCategory(categoryNumb);
List<Resume> resumes =
resumeRepository.findAllByCategoryAndApprovedIsTrue(jobCategory);
if(resumes.size()==0)
{ SendMessage emptyListMessage = new SendMessage(chatId, "На жаль в цій
категорії на даний момент немає заявок. ");
   if(jobRepository.findAllByUserId(userId).size()==0)
     { emptyListMessage.setReplyMarkup(getJobKeyboardWithoutShowJobs());} else
     { emptyListMessage.setReplyMarkup(getJobKeyboardWithShowJobs());}
     currentUser.setState(BotState.FIND_EMPLOYEE);
     repository.save(currentUser);
     return emptyListMessage;
}
List<Resume> paginationResumes = paginationService.paginationList(resumes,
userId);
for (Resume resume : paginationResumes)
    { builder.append(prepareResumeToShow(resume)); }
    sendMessage.setText(sendMessage.getText()+builder.toString());
    sendMessage.setReplyMarkup(getPaginationButtons(paginationResumes,
    jobalookDataDache.getUserCurrentPage(userId)));
    return sendMessage; }
private JobCategoryes getCategory(int categoryNumber) { int i=0;
for (JobCategoryes category : JobCategoryes.values()){
    i++;if(categoryNumber == i){ return category; }
} return null; }
private Integer getCategoryNumber(String number) {
try{ Integer inputNumber = Integer.parseInt(number);
    if(inputNumber<=30 && inputNumber>=0)
    { return inputNumber; }
} catch (Exception ex)
{ return -1; } return -1; }
private String prepareJobShow(Job job, String ident) {
    StringBuilder jobBuilder = new StringBuilder();
    jobBuilder.append("****************************************\n");
    jobBuilder.append(String.format("Ідентифікатор вакансії: %s\n", ident));
    jobBuilder.append(String.format("       %s\n", job.getTitle()));
    jobBuilder.append(String.format(" *Зарплата: %s - %s\n",
    job.getSalary().getSalaryFrom(), job.getSalary().getSalaryTo()));
    jobBuilder.append(String.format(" *Місце розташування: %s\n",
    job.getWorkPlace()));
    jobBuilder.append(String.format(" *Зайнятість: %s.\n *Досвід роботи:%s\n",
    job.getEmployment(), job.getExperience())); builder.append(String.format("
    *Опис вакансії: %s\n", job.getDescription()));
    jobBuilder.append("****************************************\n"); return
    jobBuilder.toString(); }
private String prepareResumeToShow(Resume resume) {
    StringBuilder resumeBuilder = new StringBuilder();
    resumeBuilder.append("****************************************\n");
    resumeBuilder.append(String.format("Ідентифікатор резюме:%s\n",
```

```
    resume.getUniqueId()));
    resumeBuilder.append(String.format(" %s\n", resume.getProfileInfo()));
    resumeBuilder.append(String.format(" %s\n", resume.getSkills()));
    resumeBuilder.append(String.format(" %s\n", resume.getEmpoymentHistory()));
    resumeBuilder.append(String.format(" %s\n", resume.getEducation()));
    return resumeBuilder.toString(); }
private <T> ResponseKeyboard getPaginationButtons(List<T> classList, int
currentPage)
{ final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
    responseKeyboard.setSelective(true);
    responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboard = new ArrayList<>();
if(classList.size()!=0) { if(classList.size() > currentPage+3) {
    KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Наступна сторінка"));
    keyboard.add(row_one);}if(currentPage!=0 && classList.size()!=0) {
    KeyboardRow row_two = new KeyboardRow();
    row_two.add(new KeyboardButton("Попередня сторінка"));
    keyboard.add(row_two);}}
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("Завершити"));
keyboard.add(row_three);
responseKeyboard .setKeyboard(keyboard);
return responseKeyboard ; }
```

Клас FindJobHandler

```
package com.maxsasha.jobalook.bot.handlers.findjob;
import java.util.ArrayList; import java.util.List;
//імпортування бібліотек для ін’єктії залежностей
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур import import
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.db.entity.Resume;
import com.maxsasha.jobalook.db.entity.User;
import com.maxsasha.jobalook.db.repository.ResumeRepository;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.service.ReplyService;
@Component public class FindJobHandler implements MessageHandler {
private ReplyService messagesService;
@Autowired private UserRepository repository;
@Autowired private ResumeRepository resumeRepository;
public FindJobHandler implements MessageHandler (ReplyService messagesService)
{ this.messagesService = messagesService; }
@Override public SendMessage handleMessage(Message message)
{ return processUsersInput(message); }
@Override public BotState getName()
{ return BotState.FIND_JOB; }
private SendMessage processUsersInput(Message inputMessage) {
int userId = inputMessage.getFrom().getId();
long userChatId = inputMessage.getChatId();
User user = repository.findByUserId(userId);
BotState state = user.getState();
SendMessage responseForUser = messagesService.getReplyMessage(userChatId,
"reply.askToDoJob");
if(state.equals(BotState.FIND_JOB)) {
```

```
    List<Resume> resumes = resumeRepository.findAllByUserId(userId);
    if(resumes.size()==0)
    { responseForUser.setReplyMarkup(getResumeKeyboardWithoutShowResumes());}
    else {responseForUser.setReplyMarkup(getResumeKeyboardWithShowResumes());}
    user.setState(BotState.FIND_JOB_TO_DO);
    repository.save(user);}
if(state.equals(BotState.FIND_JOB_TO_DO)) {
    List<Resume> resumes = resumeRepository.findAllByUserId(userId);
    if(resumes.size()==0) {
    responseForUser.setReplyMarkup(getResumeKeyboardWithoutShowResumes());}
    else {responseForUser.setReplyMarkup(getResumeKeyboardWithShowResumes());}
    user.setState(BotState.FIND_JOB);
    repository.save(user);}
return responseForUser; }
private ResponseKeyboard getResumeKeyboardWithShowResumes() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboard = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Пошук по категоріям"));
row_one.add(new KeyboardButton("Заповнити резюме"));
KeyboardRow row_two = new KeyboardRow();
row_two.add(new KeyboardButton("Відобразити мої резюме"));
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("На головну"));
keyboard.add(row_one); keyboard.add(row_two); keyboard.add(row_three);
responseKeyboard.setKeyboard(keyboard); return responseKeyboard;}
private ResponseKeyboard getResumeKeyboardWithoutShowResumes() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboard = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Заповнити резюме"));
KeyboardRow row_two = new KeyboardRow();
row_one.add(new KeyboardButton("Пошук по категоріям"));
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("На головну"));
keyboard.add(row_one); keyboard.add(row_two); keyboard.add(row_three);
responseKeyboard.setKeyboard(keyboard);
return responseKeyboard;} }
```

Клас FindEmployeeHandler

```
package com.maxsasha.jobalook.bot.handlers.findemployee;
import java.math.BigDecimal; import java.util.ArrayList;
import java.util.Collections; import java.util.List; import java.util.UUID;
//імпортування бібліотек для ін’єктії залежностей
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур import import
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.Job;
import com.maxsasha.jobalook.db.entity.Salary;
import com.maxsasha.jobalook.db.entity.User;
```

```
import com.maxsasha.jobalook.db.repository.JobRepository;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.enums.JobCategoryes;
import com.maxsasha.jobalook.service.PaginationService;
import com.maxsasha.jobalook.service.ReplyService;
import lombok.extern.slf4j.Slf4j;
@Component @Slf4j public class FindEmployeeHandler implements MessageHandler{
private ReplyService messagesService;
@Autowired private UserRepository userRepository;
@Autowired private JobalookDataCache jobalookDataCache;
@Autowired private JobRepository repository;
@Autowired private PaginationService paginationService;
public FindEmployeeHandler(ReplyService messagesService)  {
    this.messagesService = messagesService;
}
@Override public SendMessage handleMessage(Message message) {
log.info("Handle message: {}", message);
int userId = message.getFrom().getId();
long userChatId = message.getChatId();
User user = userRepository.findByUserId(userId);
BotState state = user.getState();
if(state.equals(BotState.FIND_EMPLOYEE)) {
    SendMessage returnFindKeyboard=new SendMessage(userChatId,"Виберіть пункт
    меню");
    List<Job> jobs = repository.findAllByUserId(userId);
    if(jobs.size()==0) {
    returnFindKeyboard.setReplyMarkup(getJobKeyboardWithoutShowJobs());}
    else {
    returnFindKeyboard.setReplyMarkup(getJobKeyboardWithShowJobs());}
    return returnFindKeyboard;
}
if(state.equals(BotState.SHOW_MY_JOBS_REQUESTS)) {
    jobalookDataCache.setUsersLastState(userId, BotState.SHOW_MY_JOBS_REQUESTS);
    String lastUniqueId = jobalookDataCache.getLastUniqueId(userId);
    Job requestedJob = repository.findByUniqueId(lastUniqueId);
    List<User> requestedUsers = requestedJob.getRequested();
    log.info("SHOW_REQUESTS lastUniqueId: {}, job: {}, jobRequestsSize: {}",
    lastUniqueId, requestedJob, requestedUsers.size());
    if(requestedUsers.size()==0) {
        user.setState(BotState.FIND_EMPLOYEE);
        userRepository.save(user);
        SendMessage showRequestedEmpty = new SendMessage(chatId, "Список
        пустий");
        showRequestedEmpty.setReplyMarkup(getJobKeyboardWithShowJobs());
        return showRequestedEmpty;
    }
    SendMessage myResumesMessage = messagesService.getReplyMessage(userChatId,
    "reply.sayHowToCheckRequestedUser");
    List<User> myUsersPagieable =
    paginationService.paginationList(requestedUsers, userId);
    StringBuilder myUsersBuilder = new StringBuilder();
    for (User item : myUsersPagieable) {
         myUsersBuilder.append(prepareUserToShow(item)); }
    myResumesMessage.setText(myResumesMessage.getText()+
    "\n"+myUsersBuilder.toString());
    myResumesMessage.setReplyMarkup(getPaginationButtons(myUsersPagieable,
    jobalookDataCache.getUserCurrentPage(userId)));
    user.setLastState(BotState.FIND_EMPLOYEE); userRepository.save(user);
    return myResumesMessage;
}
```

```
if (state.equals(BotState.SHOW_MY_JOBS) ||
state.equals(BotState.SHOW_MY_JOBS_APPROVED)
|| state.equals(BotState.SHOW_MY_JOBS_DECLINED) ||
state.equals(BotState.SHOW_MY_JOBS_UNCHECKED)) {
    log.info("SHOW_RESUMES states: {}", state);
    List<Job> myJobs = Collections.emptyList();
if(state.equals(BotState.SHOW_MY_JOBS))
    {jobalookDataCache.setUsersLastState(userId, BotState.SHOW_MY_JOBS);
    myJobs = repository.findAllByUserId(userId);
    } else if(state.equals(BotState.SHOW_MY_JOBS_UNCHECKED)) {
    jobalookDataCache.setUsersLastState(userId,BotState.SHOW_MY_JOBS_UNCHECKED);
    myJobs = repository.findAllByUserIdAndCheckedIsFalse(userId);
    } else if(state.equals(BotState.SHOW_MY_JOBS_DECLINED)) {
    jobalookDataCache.setUsersLastState(userId, BotState.SHOW_MY_JOBS_DECLINED);
    myJobs = repository.findAllByUserIdAndDeclinedIsTrue(userId);
    } else if(state.equals(BotState.SHOW_MY_JOBS_APPROVED)) {
    jobalookDataCache.setUsersLastState(userId, BotState.SHOW_MY_JOBS_APPROVED);
    myJobs = repository.findAllByUserIdAndApprovedIsTrue(userId); }
    if(myJobs.size()==0) {user.setState(BotState.FIND_EMPLOYEE);
        userRepository.save(user);
        jobalookDataCache.setUsersLastState(userId, BotState.FIND_EMPLOYEE);
        return new SendMessage(userChatId, "Список пустий"); }
SendMessage myResumesMessage = messagesService.getReplyMessage(userChatId,
"reply.sayHowToChooseJob"); String myResumeText = "";
if(state.equals(BotState.SHOW_MY_JOBS) ||
state.equals(BotState.SHOW_MY_JOBS_APPROVED)) {
    myResumeText = myResumesMessage.getText()+"\n";
}
else if(state.equals(BotState.SHOW_MY_JOBS_DECLINED)) {
    myResumeText = "Всі відхиленні
    вакансії"+"\n**************************************\n" +
    myResumesMessage.getText()+"\n";}
else { myResumeText = "Всі не розглянуті
    вакансії"+"\n**************************************\n" +
    myResumesMessage.getText()+"\n";
}
List<Job> myJobsPagieable = paginationService.paginationList(myJobs, userId);
StringBuilder myResumeBuilder = new StringBuilder(myResumeText);
for (Job item : myJobsPagieable) {
    if(state.equals(BotState.SHOW_MY_JOBS) ||
    state.equals(BotState.SHOW_MY_JOBS_APPROVED) ||
    state.equals(BotState.SHOW_MY_JOBS_UNCHECKED)) {
        myResumeBuilder.append(prepareJobToShow(item));
    }
    else if(state.equals(BotState.SHOW_MY_JOBS_DECLINED)){
        myResumeBuilder.append(prepareDeclinedJobToShow(item));
    }
}
SendMessage myResumesSendMessage =  new SendMessage(userChatId,
myResumeBuilder.toString());
myResumesSendMessage.setReplyMarkup(getPaginationButtons(myJobs,
jobalookDataCache.getUserCurrentPage(userId)));
user.setState(BotState.CHOOSING_JOB);
userRepository.save(user); return myResumesSendMessage; }
if(state.equals(BotState.CHOOSING_JOB)) {
    Job jobChoosed = repository.findByUniqueId(message.getText());
    if(jobChoosed==null)
      {return new SendMessage(userChatId, "Ви ввели не вірне значення");}
       jobalookDataCache.setUsersLastMessage(userId, message.getText());
       SendMessage editResumeMessage = new SendMessage(userChatId, "Виберіть
дію"); editResumeMessage.setReplyMarkup(getEditJob(userId));
```

```
        user.setLastState(BotState.SHOW_MY_JOBS); return editResumeMessage; }
        return processUsersInput(message);}
@Override public BotState getName() { return BotState.FIND_EMPLOYEE;}
private SendMessage processUsersInput(Message inputMessage) {
int userId = inputMessage.getFrom().getId();
long userChatId = inputMessage.getChatId();
String userResponse = inputMessage.getText();
User user = userRepository.findByUserId(userId);
BotState state = user.getState();
log.info("Proccess BotState in EmployeeHandler: {}", state);
SendMessage responseForUser = null;
Job job = repository.findByUniqueId(jobalookDataCache.getLastUniqueId(userId));
if(job == null) {
    job = Job.builder().userId(userId).build(); String uniqueId = generateUID();
    job.setUniqueId(uniqueId); job.setApproved(false); job.setDeclined(false);
    job.setChecked(false); job.setRequested(Collections.emptyList());
    jobalookDataCache.setUsersLastUniqueId(userId, uniqueId);
    repository.save(job); }
if(state.equals(BotState.FILL_JOB)) {
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.chooseCategory");user.setState(BotState.ASK_PLACING_CATEGORY);
    responseForUser = createMessageWithCategories(responseForUser);
    userRepository.save(user); }
if(state.equals(BotState.ASK_PLACING_CATEGORY)) {
    try {
         int categoryNumber = Integer.parseInt(userResponse);
         job.setCategory(getCategory(categoryNumber));
    } catch (Exception ex) {
    return new SendMessage(userChatId, "Ви ввели не вірне значення");
    }
    repository.save(job);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askTitle"); user.setState(BotState.ASK_TITLE);
    userRepository.save(user);
}
if(state.equals(BotState.ASK_TITLE)) {
    job.setTitle(userResponse); log.info("Set title: job: {}", job);
    repository.save(job);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askWorkPlace");
    user.setState(BotState.ASK_WORK_PLACE); userRepository.save(user);
}
if(state.equals(BotState.ASK_WORK_PLACE)) {
    job.setWorkPlace(userResponse);
    log.info("Set workplace: job: {}", job);
    repository.save(job);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askEmployment");
    responseForUser.setReplyMarkup(getEmploymentKeyboard());
    user.setState(BotState.ASK_EMPOYMENT);
}
if(state.equals(BotState.ASK_EMPOYMENT)) {
    SendMessage replyAskEmployment =  new SendMessage(userChatId, "Виберіть
    значення з меню!");
    replyAskEmployment.setReplyMarkup(getEmploymentKeyboard());
    return replyAskEmployment;
}
if(state.equals(BotState.ASK_EXPERIANCE)) {
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askExperiance");
    user.setState(BotState.ASK_PAYMENT_LEVEL_FROM);
```

```
    userRepository.save(user);
}
if(state.equals(BotState.ASK_PAYMENT_LEVEL_FROM))
    { job.setExperience(userResponse);
    log.info("Set experiance: job: {}", job); repository.save(job);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askPaymentLevelFrom");
    user.setState(BotState.ASK_PAYMENT_LEVEL_TO); userRepository.save(user);
}
if(state.equals(BotState.ASK_PAYMENT_LEVEL_TO)) {
    BigDecimal paymentLevelFrom;
    try {
        paymentLevelFrom = new BigDecimal(userResponse);
    } catch (Exception ex) {
        return new SendMessage(userChatId, "Ви ввели не коректне значення");}
        job.setSalary(Salary.builder().salaryFrom(paymentLevelFrom).build());
        log.info("Set salary from: job: {}", job); repository.save(job);
        responseForUser = messagesService.getReplyMessage(userChatId,
        "reply.askPaymentLevelTo");
        user.setState(BotState.ASK_PAYMENT_LEVEL_TO_FILLING);
        userRepository.save(user);
    }
}
if(state.equals(BotState.ASK_PAYMENT_LEVEL_TO_FILLING)) {
     BigDecimal paymentLevelTo;
     try {
          paymentLevelTo = new BigDecimal(userResponse);
     } catch (Exception ex) {
         return new SendMessage(userChatId, "Ви ввели не коректне значення");}
         Salary jobSalary = job.getSalary();
         jobSalary.setSalaryTo(paymentLevelTo); job.setSalary(jobSalary);
         repository.save(job);
         responseForUser = messagesService.getReplyMessage(userChatId,
         "reply.askDecription");
         user.setState(BotState.JOB_FILLED); userRepository.save(user);
     }
}
if(state.equals(BotState.JOB_FILLED)) {
     job.setDescription(userResponse);
     repository.save(job);
     SendMessage responseForUserAsk =
messagesService.getReplyMessage(userChatId,
     "reply.askSendJobVerification");
     responseForUser = new SendMessage(userChatId, String.format("Твоя вакансія
виглядає
     так:\n**********************************\n" + "Категорія для розміщення:
     %s\n" +"%s\n"+Місце розташування: %s\n" +"Зайнятість: %s\n" +
     "Зарплата: %s-%s\n+"Опис: %s\n" +
     "\n**********************************\n"+responseForUserAsk.getText(),
     job.getCategory().toString(), job.getTitle(),job.getWorkPlace(),
     job.getEmployment(),job.getSalary().getSalaryFrom(),
            job.getSalary().getSalaryTo(), job.getDescription()));
     responseForUser.setReplyMarkup(getTODOVereficationKeyboard());}
     return responseForUser;}
private String prepareJobToShow(Job job) {
StringBuilder jobShowBuilder = new StringBuilder();
jobShowBuilder.append("****************************************\n");
jobShowBuilder.append(String.format("Ідентифікатор вакансії: %s\n",
job.getUniqueId()));
jobShowBuilder.append(String.format("Заголовок: %s\n", job.getTitle()));
```

```
jobShowBuilder.append(String.format("Категорія для розміщення: %s\n",
job.getCategory()));
jobShowBuilder.append(String.format("Місце розташування: %s\n",
job.getWorkPlace()));
jobShowBuilder.append(String.format("Зайнятість: %s\n", job.getEmployment()));
jobShowBuilder.append(String.format("Зарплата: %s-%s\n",
job.getSalary().getSalaryFrom(), job.getSalary().getSalaryTo()));
jobShowBuilder.append(String.format("Опис: %s\n", job.getDescription()));
return jobShowBuilder.toString();}
private String prepareDeclinedJobToShow(Job job) {
StringBuilder jobShowBuilder = new StringBuilder();
jobShowBuilder.append("**************************************\n");
jobShowBuilder.append(String.format("Ідентифікатор вакансії: %s\n",
job.getUniqueId()));
jobShowBuilder.append(String.format("Заголовок: %s\n", job.getTitle()));
jobShowBuilder.append(String.format("Категорія для розміщення: %s\n",
job.getCategory()));
jobShowBuilder.append(String.format("Місце розташування: %s\n",
job.getWorkPlace()));
jobShowBuilder.append(String.format("Зайнятість: %s\n", job.getEmployment()));
jobShowBuilder.append(String.format("Зарплата: %s-%s\n",
job.getSalary().getSalaryFrom(), job.getSalary().getSalaryTo()));
jobShowBuilder.append(String.format("Опис: %s\n", job.getDescription()));
jobShowBuilder.append(String.format("ПРИЧИНА ВІДХИЛЕННЯ ВАКАНСІЇ: %s\n",
job.getDeclineExplane())); return builder.toString();}
private String prepareUserToShow(User user) {
StringBuilder userBobBuilder = new StringBuilder();
userBobBuilder.append("**************************************\n");
userBobBuilder.append(String.format("Ім'я: %s\n", user.getName()));
userBobBuilder.append(String.format("Вік: %s\n", user.getAge()));
userBobBuilder.append(String.format("Стать: %s\n", user.getGender()));
userBobBuilder.append(String.format("Контактні дані: \n * email: %s\n *
Телеграм: @%s", user.getEmail(), user.getUserName()));
return userBobBuilder.toString(); }
private JobCategoryes getCategory(int categoryNumber) {
int i=0;
for (JobCategoryes category : JobCategoryes.values()){
    i++;
    if(categoryNumber == i)
    {return category; } }
return null;}
private SendMessage createMessageWithCategories(SendMessage sendMessage) {
SendMessage newSendMessage = sendMessage;
StringBuilder categoryBuilder = new StringBuilder(sendMessage.getText()+"\n
" + "Список категорій\n**********************************\n"); int i=0;
builder.append(String.format("№  | Назва\n"));
for (JobCategoryes category : JobCategoryes.values()){i++; if(i>=10)
    { categoryBuilder.append(String.format("%s  | %s\n",i ,
category.toString()));}
    else { categoryBuilder.append(String.format("%s
| %s\n",i,category.toString())); }}
    categoryBuilder.append("\n**********************************\n");
    newSendMessage.setText(categoryBuilder.toString()); return newSendMessage; }
    public String generateUID(){
    UUID uuid = UUID.randomUUID(); String id = uuid.toString();
    String[] newId = id.split("-");
    String newUUID = newId[newId.length-1].substring(3,9);
    log.info("Generate UUID:{} split ID:{}", uuid, newUUID);
return newUUID; }
private ResponseKeyboard getTODOVereficationKeyboard() {
final ResponseKeyboard responseKeyboard  = new ResponseKeyboard();
```

```
responseKeyboard.setSelective(true);responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Відправити на перевірку"));
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("Заповнити вакансію заново"));
keyboardRow.add(row_one); keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard; }
private ResponseKeyboard getJobKeyboardWithShowJobs() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true); responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Пошук по категоріям"));
row_one.add(new KeyboardButton("Створити вакансію"));
KeyboardRow row_two = new KeyboardRow();
row_two.add(new KeyboardButton("Відобразити мої вакансії"));
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("На головну"));
keyboardRow.add(row_one); keyboardRow.add(row_two); keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard; }
private ResponseKeyboard getJobKeyboardWithoutShowJobs() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Пошук по категоріям"));
KeyboardRow row_two = new KeyboardRow();
row_two.add(new KeyboardButton("Створити вакансію"));
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("На головну"));
keyboardRow.add(row_one); keyboardRow.add(row_two); keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow);return responseKeyboard; }
private ResponseKeyboard getEmploymentKeyboard() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Повна зайнятість"));
KeyboardRow row_two = new KeyboardRow();
row_one.add(new KeyboardButton("Неповна зайнятість"));
KeyboardRow row_three = new KeyboardRow();
row_one.add(new KeyboardButton("Дистанційна робота"));
keyboardRow.add(row_one); keyboardRow.add(row_two); keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow);return responseKeyboard; }
private<T> ResponseKeyboard getPaginationButtons(List<T> classList, int
currentPage) {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
if(classList.size()!=0) {
    int nextPage = currentPage;
    nextPage++;
```

```
    if(classList.size() > nextPage+3) {
        KeyboardRow row_one = new KeyboardRow();
        row_one.add(new KeyboardButton("Наступна
сторінка"));keyboardRow.add(row_one);}
    if(currentPage!=0 && classList.size()!=0) {
        KeyboardRow row_two = new KeyboardRow();
        row_two.add(new KeyboardButton("Попередня
сторінка"));keyboardRow.add(row_two);}
} KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("Завершити"));keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard; }
private ResponseKeyboard getEditJob(int userId) {
final ResponseKeyboard replyKeyboard = new ResponseKeyboard();
replyKeyboard.setSelective(true); replyKeyboard.setResizeKeyboard(true);
replyKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
BotState lastState = jobalookDataCache.getUsersLastState(userId);
if(lastState.equals(BotState.SHOW_MY_JOBS_UNCHECKED)) {
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Редагувати"));keyboardRow.add(row_one);}
if(lastState.equals(BotState.SHOW_MY_JOBS_APPROVED)) {KeyboardRow row_one = new
KeyboardRow();row_one.add(new KeyboardButton("Відобразити список запитів"));
keyboardRow.add(row_one); }
KeyboardRow row_two = new KeyboardRow();row_two.add(new
KeyboardButton("Видалити"));
KeyboardRow row_three = new KeyboardRow();row_three.add(new
KeyboardButton("Назад"));
keyboardRow.add(row_two); keyboardRow.add(row_three);
replyKeyboard.setKeyboard(keyboardRow);
return replyKeyboard; }
}
```

Клас HelloHandler

```
package com.maxsasha.jobalook.bot.handlers.hello;
//імпортування стандартних Java утиліт
//імпортування бібліотек для ін’єктії залежностей
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур import import
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.User;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.service.ReplyService;
import lombok.extern.slf4j.Slf4j;
@Component @Slf4j public class HelloHandler implements MessageHandler {
private ReplyService messagesService;
@Autowired private UserRepository repository;
public HelloHandler(JobalookDataCache jobalookDataCache,
ReplyService messagesService)
{this. jobalookDataCache = jobalookDataCache; this.messagesService =
messagesService; }@Override public SendMessage handleMessage(Message message) {
return processUsersInput(message); }
@Override public BotState getName() {return BotState.HELLO; }
private SendMessage processUsersInput(Message inputMessage) {
int userId = inputMessage.getFrom().getId();
long userChatId = inputMessage.getChatId();
```

```
SendMessage responseForUser =
messagesService.sendHelloMessage(userChatId,"reply.helloMessage",
inputMessage.getFrom().getFirstName());
responseForUser.setReplyMarkup(getMainMenuKeyboard());
User user = repository.findByUserId(userId);
user.setState(BotState.FILLING_PROFILE);
log.info("Save user: {} to database" ,user); repository.save(user);
return responseForUser; }
private ResponseKeyboard getMainMenuKeyboard() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true); responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();row_one.add(new
KeyboardButton("Розрочнемо"));
keyboardRow.add(row_one); responseKeyboard.setKeyboard(keyboardRow);
return responseKeyboard; } }
```

Клас FillingProfileHandler

```
package com.maxsasha.jobalook.bot.handlers.fillingprofile;
import java.util.ArrayList; import java.util.List;
import java.util.regex.Matcher; import java.util.regex.Pattern;
//імпортування бібліотек для ін’єктії залежностей
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур import import
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.User;
import com.maxsasha.jobalook.db.repository.UserRepository;
import com.maxsasha.jobalook.service.ReplyService;
import lombok.extern.slf4j.Slf4j;
@Component @Slf4j public class FillingProfileHandler implements MessageHandler {
@Autowired private JobalookDataCache jobalookDataCache;
@Autowired private ReplyService messagesService;
@Autowired private UserRepository repository;
public FillingProfileHandler() { }
public SendMessage handleMessage(Message message) {
log.info("Handle message: {}", message);
int userId = message.getFrom().getId();
User user = repository.findByUserId(userId);
log.info("Handle bot state\n Old bot state: {}", user.getState());
if (user.getState().equals(BotState.FILLING_PROFILE))
    { jobalookDataCache.setUsersCurrentBotState(userId, BotState.ASK_NAME);
    user.setState(BotState.ASK_NAME);
    log.info("Current bot state: {}", user.getState());
    log.info("Save user: {} to datebase with new state: {}", user,
    user.getState()); repository.save(user); }
return processUsersInput(message, userId); }
public BotState getName(){ return BotState.FILLING_PROFILE; }
private SendMessage processUsersInput(Message inputMessage, int userId) {
log.info("Proccess input message: {}", inputMessage);
String userResponse = inputMessage.getText();
long userChatId = inputMessage.getChatId();
UserProfileData profileData = jobalookDataCache.getUserProfileData(userId);
BotState botState = jobalookDataCache.getUsersCurrentBotState(userId);
log.info("Proccess BotState: {}", botState); SendMessage responseForUser = null;
```

```
User user = repository.findByUserId((long)userId);
if (botState.equals(BotState.ASK_AGE)) {
    user.setName(userResponse);
    responseForUser = messagesService.getReplyMessage(userChatId,
"reply.askAge");
    user.setState(BotState.ASK_GENDER); repository.save(user); }
if (botState.equals(BotState.ASK_GENDER)) {
    try
    {profileData.setAge(Integer.parseInt(userResponse));
    user.setAge(Integer.parseInt(userResponse));
    repository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
"reply.askGender");
    responseForUser.setReplyMarkup(getGenderKeyboard());
    jobalookDataCache.setUsersCurrentBotState(userId, BotState.ASK_EMAIL); }
    catch (Exception ex)
    {responseForUser = new SendMessage(userChatId, "ви ввели не коректне
значення");}}
if (botState.equals(BotState.PROFILE_FILLED)) {
    Pattern emailPattern = Pattern.compile("^[\\w-
    \\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");
    Matcher matcher = emailPattern.matcher(userResponse);
    if(!matcher.matches())
        {new SendMessage(userChatId, "Ви ввели не вірне значення");}
        profileData.setEmail(userResponse);
        jobalookDataCache.setUsersCurrentBotState(userId, BotState.ASK_TODO);
        log.info("Current profile data: Name: {},
        Age: {}, Sex: {}, Email: {}",
        profileData.getName(),
        profileData.getAge(),
        profileData.getGender(),
        profileData.getEmail()); user.setState(BotState.ASK_TODO);
        user.setEmail(userResponse); repository.save(user);
        responseForUser =new SendMessage(userChatId, String.format("Твій профіль
        виглядає так: \n * Твоє ім'я: %s\n * Тобі %s\n * Твоя стать: %s\n * Твоя
        електронна пошта: %s user.getName(),user.getAge(),
        user.getGender(),user.getEmail()));
        responseForUser.setReplyMarkup(getTODOKeyboard(user)); }
this. jobalookDataCache.saveUserProfileData(userId, profileData);
return responseForUser; }
private ResponseKeyboard getTODOKeyboard(User user) {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(false);
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Знайти роботу"));
row_one.add(new KeyboardButton("Знайти працівників"));
if(user.getIsModerator()==true) {
    KeyboardRow row_two = new KeyboardRow();
    row_two.add(new KeyboardButton("Меню модератора"));keyboard.add(row_two); }
KeyboardRow row_three = new KeyboardRow();
row_three.add(new KeyboardButton("Додатково"));
keyboardRow.add(row_one); keyboardRow.add(row_three);
responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard; }
private ResponseKeyboard getGenderKeyboard() {
final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
responseKeyboard.setSelective(true);
responseKeyboard.setResizeKeyboard(true);
responseKeyboard.setOneTimeKeyboard(true);
```

```
List<KeyboardRow> keyboardRow = new ArrayList<>();
KeyboardRow row_one = new KeyboardRow();
row_one.add(new KeyboardButton("Чоловіча"));
row_one.add(new KeyboardButton("Жіноча"));
 keyboardRow.add(row_one);
responseKeyboard.setKeyboard(keyboardRow);
 return responseKeyboard; }
}
```

Клас UserProfile

```
package com.maxsasha.jobalook.bot.handlers.fillingprofile;
//імпортування бібліотеки Lombok
@Data @Builder @AllArgsConstructor @NoArgsConstructor
public class UserProfile {
private String userName; private long userId; private long userChatId;
private String name; private int age; private String gender;
private String email; }
```

Клас FillingResumeHandler

```
package com.maxsasha.jobalook.bot.handlers.fillingResume;
import java.util.ArrayList;  import java.util.Collections;  import
java.util.List; import java.util.UUID;
import java.util.regex.Matcher; import java.util.regex.Pattern;
//імпортування бібліотек для ін’єктії залежностей
//імпортування бібліотек для повідомлень Telegram
//імпортування бібліотек Telegram клавіатур import import
com.maxsasha.jobalook.bot.api.BotState;
import com.maxsasha.jobalook.bot.api.MessageHandler;
import com.maxsasha.jobalook.bot.handlers.fillingprofile.UserProfileData;
import com.maxsasha.jobalook.cache.JobalookDataCache;
import com.maxsasha.jobalook.db.entity.Resume;  import
com.maxsasha.jobalook.db.entity.User;
import com.maxsasha.jobalook.db.repository.ResumeRepository;
import com.maxsasha.jobalook.db.repository.UserRepository; import
com.maxsasha.jobalook.enums.JobCategoryes;
import com.maxsasha.jobalook.service.PaginationService;
import com.maxsasha.jobalook.service.ReplyService; import
lombok.extern.slf4j.Slf4j;
@Component @Slf4j public class FillingResumeHandler implements MessageHandler
{ @Autowired private JobalookDataCache jobalookDataCache;
@Autowired private ReplyService messagesService;
@Autowired private ResumeRepository repository;
@Autowired private UserRepository userRepository;
@Autowired private PaginationService paginationService;  public
FillingResumeHandler() { }
public SendMessage handleMessage (Message message){log.info("Handle message:{}",
message);int userId = message.getFrom().getId();
long userChatId = message.getChatId();
User user = userRepository.findByUserId(userId);
log.info("Handle bot state\n Old bot state: {}", user.getState());
if (user.getState().equals(BotState.FILLING_RESUME)) {
```

```
    user.setState(BotState.ASK_PLACE_CATEGORY);
    log.info("Current bot state: {}", user.getState());
    log.info("Save user: {} to datebase with new state: {}", user,
    user.getState());userRepository.save(user); }
BotState state = user.getState();
if (state.equals(BotState.SHOW_MY_RESUMES) ||
state.equals(BotState.SHOW_MY_RESUMES_APPROVED)
|| state.equals(BotState.SHOW_MY_RESUMES_DECLINED) ||
state.equals(BotState.SHOW_MY_RESUMES_UNCHECKED))
    {log.info("SHOW_RESUMES states: {}", state);
    List<Resume> myResumes = Collections.emptyList();
    if(state.equals(BotState.SHOW_MY_RESUMES))
        { myResumes = repository.findAllByUserId(userId);}
    else if(state.equals(BotState.SHOW_MY_RESUMES_UNCHECKED))
        {jobalookDataCache.setUsersLastState(userId,
        BotState.SHOW_MY_RESUMES_UNCHECKED);
        myResumes = repository.findAllByUserIdAndCheckedIsFalse(userId); }
    else if(state.equals(BotState.SHOW_MY_RESUMES_DECLINED))
        {myResumes = repository.findAllByUserIdAndDeclinedIsTrue(userId);}
    else if(state.equals(BotState.SHOW_MY_RESUMES_APPROVED))
        {myResumes = repository.findAllByUserIdAndApprovedIsTrue(userId);}
    if(myResumes.size()==0)
        {user.setState(BotState.FIND_JOB_TO_DO); userRepository.save(user);
        jobalookDataCache.setUsersLastState(userId, BotState.FIND_JOB_TO_DO);
        return new SendMessage(userChatId, "Список пустий");}
SendMessage myResumesMessage = messagesService.getReplyMessage(userChatId,
"reply.sayHowToChooseShowResume");
String myResumeText = "";
if(state.equals(BotState.SHOW_MY_RESUMES) ||
    state.equals(BotState.SHOW_MY_RESUMES_APPROVED)) {
    myResumeText = myResumesMessage.getText()+"\n";
} else if (state.equals(BotState.SHOW_MY_RESUMES_DECLINED)) {
    myResumeText = "                Всі відхиленні
    резюме"+"\n****************************************\n" +
    myResumesMessage.getText()+"\n";
} else {
    myResumeText = "                Всі не розглянуті
    резюме"+"\n****************************************\n" +
    myResumesMessage.getText()+"\n";}
List<Resume> myResumesPagieable = paginationService.paginationList(myResumes,
userId);
StringBuilder myResumeBuilder = new StringBuilder(myResumeText);
for (Resume item : myResumesPagieable) {
    if(state.equals(BotState.SHOW_MY_RESUMES) ||
        state.equals(BotState.SHOW_MY_RESUMES_APPROVED) ||
        state.equals(BotState.SHOW_MY_RESUMES_UNCHECKED)) {
        myResumeBuilder.append(prepareResumeToShow(item));}
    else if(state.equals(BotState.SHOW_MY_RESUMES_DECLINED))
        {myResumeBuilder.append(prepareDeclinedResumeToShow(item)); }}
SendMessage myResumesSendMessage =  new SendMessage(userChatId,
myResumeBuilder.toString());
myResumesSendMessage.setReplyMarkup(getPaginationButtons(myResumes,
jobalookDataCache.getUserCurrentPage(userId)));
user.setState(BotState.CHOOSING_RESUME); userRepository.save(user);
return myResumesSendMessage;}
if(state.equals(BotState.CHOOSING_RESUME)) {
    Resume resumeChoosed = repository.findByUniqueId(message.getText());
    if(resumeChoosed==null)
    {return new SendMessage(userChatId, "Ви ввели не вірне значення");}
    jobalookDataCache.setUsersLastMessage(userId, message.getText());
    SendMessage editResumeMessage = new SendMessage(userChatId, "Виберіть дію");
```

```
    user.setLastState(BotState.SHOW_MY_RESUMES);return editResumeMessage;}
    return processUsersInput(message, userId); }
public BotState getName(){return BotState.FILLING_RESUME; }
private SendMessage processUsersInput(Message inputMessage, int userId) {
log.info("Proccess input message: {}", inputMessage);
String userResponse = inputMessage.getText();long userChatId =
inputMessage.getChatId();
UserProfileData profileData = jobalookDataCache.getUserProfileData(userId);
SendMessage responseForUser = null; User user =
userRepository.findByUserId(userId);
BotState botState = user.getState();log.info("Proccess BotState: {}", botState);
Resume resume=
repository.findByUniqueId(jobalookDataCache.getLastUniqueId(userId));
if(resume == null) {resume = Resume.builder().userId(userId).build();
    String uniqueId = generateUID();
    resume.setUniqueId(uniqueId); resume.setApproved(false);
    resume.setDeclined(false); resume.setChecked(false);
    jobalookDataCache.setUsersLastUniqueId(userId, uniqueId);
    repository.save(resume);}
if (botState.equals(BotState.ASK_PLACE_CATEGORY)) {
    responseForUser = messagesService.getReplyMessage(userChatId, \
    "reply.chooseCategory");
    responseForUser = createMessageWithCategories(responseForUser);
    user.setState(BotState.ASK_MIDDLE_NAME); userRepository.save(user); }
if (botState.equals(BotState.ASK_MIDDLE_NAME)) {
    try{int categoryNumber = Integer.parseInt(userResponse);
    resume.setCategory(getCategory(categoryNumber));
    } catch (Exception ex)
    { return new SendMessage(userChatId, "Ви ввели не вірне значення");}
    repository.save(resume);
    responseForUser= messagesService.getReplyMessage(userChatId,
"reply.askMiddleName");
    user.setState(BotState.ASK_CAREER_OBJECTIVE);userRepository.save(user); }
if (botState.equals(BotState.ASK_CAREER_OBJECTIVE)) {
    StringBuilder resumeBuilder = new StringBuilder();
    resumeBuilder.append(String.format("%s %s \n\nСтать: %s \n",user.getName(),
    userResponse, user.getGender()));
    resume.setProfileInfo(resumeBuilder.toString());repository.save(resume);
    user.setState(BotState.ASK_DESIRED_INCOME_LEVEL_FROM);
    userRepository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askCareerObjective");}
if (botState.equals(BotState.ASK_DESIRED_INCOME_LEVEL_FROM)) {
    StringBuilder resumeBuilder = new StringBuilder(resume.getProfileInfo());
    resumeBuilder.append(String.format("Бажана посада: %s\n",userResponse));
    resume.setProfileInfo(resumeBuilder.toString());repository.save(resume);
    user.setState(BotState.ASK_DESIRED_INCOME_LEVEL_TO);
    userRepository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askIncomeLevelFrom");}
if (botState.equals(BotState.ASK_DESIRED_INCOME_LEVEL_TO)) {
    int desiredFrom;try{desiredFrom = Integer.parseInt(userResponse);}
    catch (Exception ex) {
    return new SendMessage(userChatId, "Ви ввели не вірне значення");}
    StringBuilder resumeBuilder = new StringBuilder(resume.getProfileInfo());
    resumeBuilder.append(String.format("Бажаний рівень доходу : %s-",
    desiredFrom)); resume.setProfileInfo(resumeBuilder.toString());
    repository.save(resume);user.setState(BotState.ASK_NUMBER);
    userRepository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askIncomeLevelTo");}
```

```
if (botState.equals(BotState.ASK_NUMBER)) {
    int desiredTo;try{desiredTo = Integer.parseInt(userResponse); }
    catch (Exception ex) {
    return new SendMessage(userChatId, "Ви ввели не вірне значення");}
    StringBuilder resumeBuilder = new StringBuilder(resume.getProfileInfo());
    resumeBuilder.append(String.format("%s грн.\n",desiredTo));
    resume.setProfileInfo(resumeBuilder.toString());repository.save(resume);
    user.setState(BotState.ASK_SKILL); userRepository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
"reply.askNumber");}
if (botState.equals(BotState.ASK_SKILL)) {
    Pattern phonePattern = Pattern.compile("^\\+?3?8?(0\\d{9})$");
    Matcher matcher = phonePattern.matcher(userResponse);
    if(!matcher.matches())
        {return new SendMessage(userChatId, "Ви ввели не вірне значення");}
    StringBuilder resumeBuilder = new StringBuilder(resume.getProfileInfo());
    resumeBuilder.append(String.format("Контактна інформація:\n * Телеграм:
    @%s\n * Телефон: %s\n * Електронна пошта: %s\n",
    user.getUserName() ,userResponse, user.getEmail()));
    resume.setProfileInfo(resumeBuilder.toString());repository.save(resume);
    user.setState(BotState.ASK_EMPOYMENT_HISTORY);userRepository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
"reply.askSkill");}
if (botState.equals(BotState.ASK_EMPOYMENT_HISTORY)) {
    resume.setSkills(String.format("Ключові знання і навички: %s \n",
    userResponse));repository.save(resume);
    user.setState(BotState.ASK_EDUCATION); userRepository.save(user);
    responseForUser = messagesService.getReplyMessage(userChatId,
    "reply.askEmploymentHistory");}
if (botState.equals(BotState.ASK_EDUCATION)) {
    resume.setEmpoymentHistory(String.format("Досвід роботи: %s \n",
    userResponse)); repository.save(resume);
    user.setState(BotState.RESUME_FILLED);userRepository.save(user);
    responseForUser= messagesService.getReplyMessage(userChatId,
"reply.askEducation");}
if (botState.equals(BotState.RESUME_FILLED)) {
    resume.setEducation(String.format("Освіта: %s", userResponse));
    repository.save(resume);user.setState(BotState.ASK_RESUME_FILLED);
    userRepository.save(user);
    SendMessage responseForUserAsk = messagesService.getReplyMessage(userChatId,
    "reply.askSendForVerification");responseForUser = new
SendMessage(userChatId,
    String.format(responseForUserAsk.getText()+"\nТвоє резюме виглядає
    так:\n**********************************\n" + "Категорія для
    розміщення: %s\n\n%s\n" +"%s\n" +"%s\n" +"%s\n" +
    "\n**********************************\n\nВаше резюме буде розміщено після
    перевірки модератором на коректність.\nВідправити на перевірку модератору?",
    resume.getCategory().toString(), resume.getProfileInfo(),resume.getSkills(),
    resume.getEmpoymentHistory(),resume.getEducation()));
    responseForUser.setReplyMarkup(getTODOVereficationKeyboard());}
    jobalookDataCache.saveUserProfileData(userId, profileData);return
responseForUser;}
private SendMessage createMessageWithCategories(SendMessage sendMessage) {
SendMessage newSendMessage = sendMessage;
StringBuilder builder = new StringBuilder(sendMessage.getText()+"\n
" +"Список категорій\n**********************************\n");int i=0;
builder.append(String.format("№  | Назва\n"));
for (JobCategoryes category : JobCategoryes.values()){ i++;
    if(i>=10)
        {builder.append(String.format("%s  | %s\n",i , category.toString()));}
    else {builder.append(String.format("%s | %s\n",i , category.toString()));}}
```

```
builder.append("\n***********************************\n");
newSendMessage.setText(builder.toString());return newSendMessage; }
private ResponseKeyboard getTODOVerefication Keyboard() {
    final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
    responseKeyboard.setSelective(true);
    responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboardRow = new ArrayList<>();
    KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Відправити на перевірку"));
    KeyboardRow row_three = new KeyboardRow();
    row_three.add(new KeyboardButton("Заповнити резюме заново"));
    keyboardRow.add(row_one); keyboardRow.add(row_three);
    responseKeyboard.setKeyboard(keyboardRow);return responseKeyboard;}
public ResponseKeyboard getTODORewriteKeyboard() {
    final ResponseKeyboard responseKeyboard = new ResponseKeyboard();
    responseKeyboard.setSelective(true);
    responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboardRow = new ArrayList<>();
    KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Заповнити основні навички, досвід роботи та
    освіту заново")); KeyboardRow row_two = new KeyboardRow();
    row_two.add(new KeyboardButton("Заповнити резюме заново"));
    KeyboardRow row_three = new KeyboardRow();
    row_three.add(new KeyboardButton("Назад"));
   keyboardRow.add(row_one);keyboardRow.add(row_two);keyboardRow.add(row_three);
    responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard;}
private String prepareResumeToShow(Resume resume) {
    StringBuilder resumeShowBuilder = new StringBuilder();
    resumeShowBuilder.append("***************************************\n");
    resumeShowBuilder.append(String.format("Ідентифікатор резюме: %s\n",
    resume.getUniqueId()));
    resumeShowBuilder.append(String.format(" %s\n", resume.getProfileInfo()));
    resumeShowBuilder.append(String.format(" %s\n", resume.getSkills()));
    resumeShowBuilder.append(String.format(" %s\n",
resume.getEmpoymentHistory()));
    resumeShowBuilder.append(String.format(" %s\n", resume.getEducation()));
    return resumeShowBuilder.toString();}
private String prepareDeclinedResumeToShow(Resume resume) {
    StringBuilder resumeShowBuilder = new StringBuilder();
    resumeShowBuilder.append("***************************************\n");
    resumeShowBuilder.append(String.format("Ідентифікатор резюме: %s\n",
    resume.getUniqueId()));
    resumeShowBuilder.append(String.format(" %s\n", resume.getProfileInfo()));
    resumeShowBuilder.append(String.format(" %s\n", resume.getSkills()));
    resumeShowBuilder.append(String.format(" %s\n",
resume.getEmpoymentHistory()));
    resumeShowBuilder.append(String.format(" %s\n", resume.getEducation()));
    resumeShowBuilder.append(String.format("ПРИЧИНА ВІДХИЛЕННЯ РЕЗЮМЕ: %s\n",
    resume.getDeclineExplane()));return resumeShowBuilder.toString();}
public String generateUID(){
    UUID uuid = UUID.randomUUID();
    String id = uuid.toString(); String[] newId = id.split("-");
    String newUUID = newId[newId.length-1].substring(3,9);
    log.info("Generate UUID:{} split ID:{}", uuid, newUUID); return newUUID;}
private ResponseKeyboard getPaginationButtons(List<Resume> resumes, int
currentPage) {
    ResponseKeyboard responseKeyboard = new ResponseKeyboard();
   responseKeyboard.setSelective(true);responseKeyboard.setResizeKeyboard(true);
    responseKeyboard.setOneTimeKeyboard(true);
```

```
    List<KeyboardRow> keyboardRow = new ArrayList<>(); if(resumes.size()!=0) {
    int nextPage = currentPage; nextPage++;
    if(resumes.size() > nextPage*3) { KeyboardRow row_one = new KeyboardRow();
    row_one.add(new KeyboardButton("Наступна сторінка"));
keyboardRow.add(row_one); } if(currentPage!=0 && resumes.size()!=0) {
    KeyboardRow row_two = new KeyboardRow();
    row_two.add(new KeyboardButton("Попередня сторінка"));
keyboardRow.add(row_two);}} KeyboardRow row_three = new KeyboardRow();
    row_three.add(new KeyboardButton("Завершити")); keyboardRow.add(row_three);
    responseKeyboard.setKeyboard(keyboardRow); return responseKeyboard; }
private ResponseKeyboard getEditResume(String message, int userId) {
    final ResponseKeyboard replyKeyboard = new ResponseKeyboard();
    replyKeyboard.setSelective(true);replyKeyboard.setResizeKeyboard(true);
    replyKeyboard.setOneTimeKeyboard(true);
    List<KeyboardRow> keyboardRow = new ArrayList<>();
    if(jobalookDataCache.getUsersLastState(userId).equals(
    BotState.SHOW_MY_RESUMES_UNCHECKED)) {
KeyboardRow row_one = new KeyboardRow();
        row_one.add(new KeyboardButton("Редагувати"));keyboard.add(row_one); }
        KeyboardRow row_two = new KeyboardRow();
        row_two.add(new KeyboardButton("Видалити"));
        KeyboardRow row_three = new KeyboardRow();
        row_three.add(new KeyboardButton("Назад"));
        keyboardRow.add(row_two); keyboardRow.add(row_three);
        replyKeyboard.setKeyboard(keyboardRow); return replyKeyboard;}

private JobCategoryes getCategory(int categoryNumber) {
    int i=0;
    for (JobCategoryes category : JobCategoryes.values()){
        i++;
        if(categoryNumber == i)
        { return category; }
    } return null; }
}
```

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

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

Дипломний проект на тему: «Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота»

Виконав студент Максимів Олександр Володимирович
Керівник: Радельчук Г. І., кандидат технічних наук, доцент

Мета та завдання проекту

Метою проекту є розробка програмного забезпечення (ПЗ), яке дозволяє оптимізувати та автоматизувати публікацію та пошук оголошень роботи або працівників серед користувачів ПЗ, забезпечує швидкий та зручний їх пошук, а також має сучасний інтуїтивно зрозумілий користувацький інтерфейс.

Для реалізації програмного забезпечення необхідно виконати наступні завдання:

1) визначити актуальність теми та встановити практичну користь розроблюваного ПЗ;
2) дослідити наявне програмне забезпечення предметної області та виявити недоліки;
3) визначити вимоги до розроблюваного ПЗ та функції, яке воно має виконувати;
4) проаналізувати використання реляційних, нереляційних баз даних при реалізації програмного забезпечення;
5) виконати порівняння мікросервісної і монолітної архітектури та вибрати одну з них;
6) провести аналіз програмних модулів та встановити зв'язки між ними;
7) провести аналіз способів реалізації програмного забезпечення, та реалізувати серверну частину ПЗ;
8) встановити особливості та загальні правила для побудови користувацького інтерфейсу та створити його;
9) провести тестування розробленого ПЗ.

Актуальність теми

Актуальність теми полягає у тому, що на сьогодні в мережі Інтернет є безліч сайтів для пошуку роботи та пошуку працівників, проте в Україні месенджери, як інструмент для пошуку роботи, не досить популярні та потребують розвитку. На сьогодні "боти" слугують як асистенти до основних веб-сайтів, а не представляють самодостатню систему. Розробивши дане ПЗ, можна буде створити конкурентні вимоги до сайтів і замінити нудні вечори та години потраченого часу на пошук потрібного оголошення або працівника, на зручний, швидкий та зрозумілий інтерфейс Telegram-бота. Також, Telegram охоплює не тільки молодіжну аудиторію, а й старше покоління, таким чином дозволяючи охопити досить широку аудиторію і добитись швидкого розвитку нової ідеї.

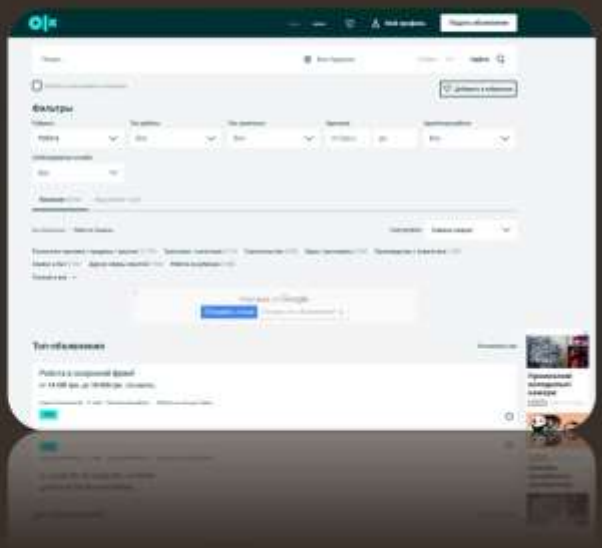
Work ua

На сайті є швидкий та розширений пошук, також присутній пошук за категоріями та посадами. На головній сторінці сайту розташовані популярні логотипи компаній, настиснувши на які можна побачити вакансії для вибраної компанії.

Також на цьому сайті є місце і роботодавцям – вони можуть створювати вакансії та вказувати унікальні характеристики для вакансії, що дозволить краще розуміти, що вимагається від претендента на роботу. Контент сайту представлений трьома мовами, а саме: українською, російською, англійською.

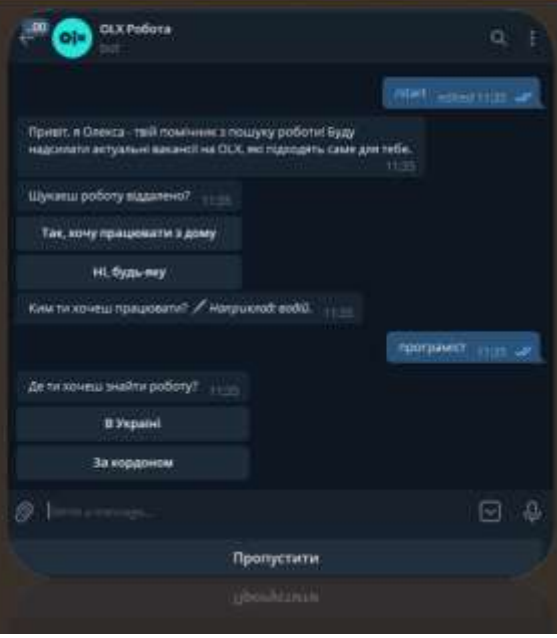
Olx rabota

Цей сайт орієнтований на розміщення на продаж та пошук різноманітних товарів, але в ньому є ще додаткова сторінка для пошуку працівників/роботи. На сайті є можливість пошуку роботи за різними критеріями, але відсутній розширений пошук за віком, статтю, освітою та іншими додатковими критеріями. Недолік сайту полягає в тому, що нові користувачі можуть заплутатися у пошуку потрібної їм сторінки, оскільки сайт орієнтований на різні напрямки.



Olx rabota bot

Це бот-асистент сайту: https://www.olx.ua/rabota/. В ньому присутній інтуїтивно зрозумілий інтерфейс та швидкий вибір параметрів. Відсутня реєстрація клієнта у системі, що не дозволяє створити особисте резюме. Бот автоматично підбирає роботу користувачу та надсилає декілька результатів пошуку, в яких міститься посилання на основний сайт, що створює низку незручностей для користувача.



Порівняння наявного ПЗ

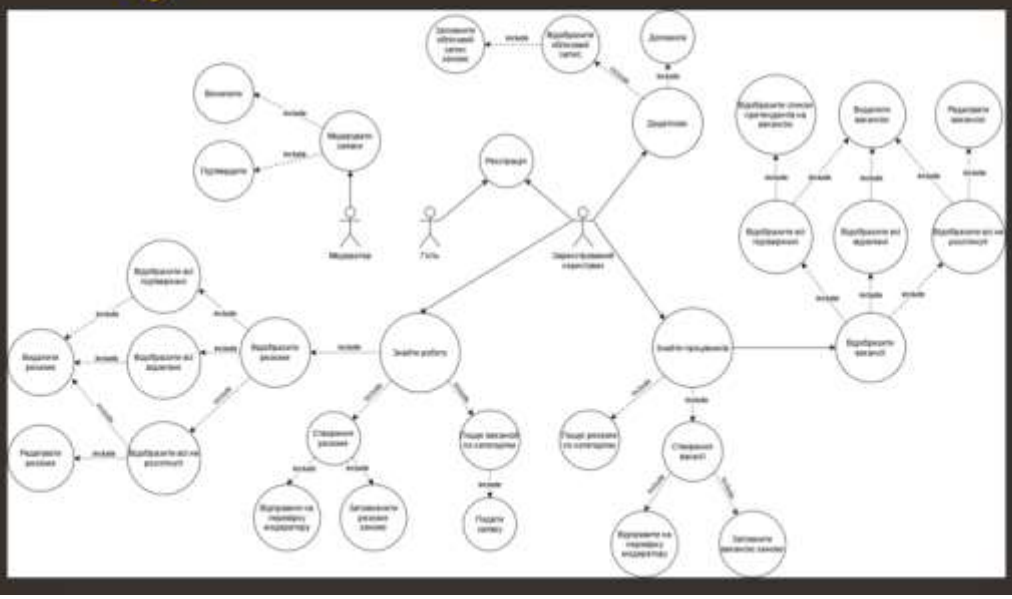
Назва	Тип	Можливість реєстрації	Пошук працівників	Пошук роботи	Можливість спілкування на сервісі	Швидкість виконання запиту	Відображення оголошень в інтерфейсі користувача
Work ua	сайт	Так	Так	Так	Так	середня	Так
olx rabot a ua	сайт	Так	Так	Так	Так	середня	Так
@olx_rob ota_bot	бот	Ні	Ні	Так	Ні	низька	Ні

Аналіз вимог

Проведений аналіз наявного програмного забезпечення показав, що в телеграм-ботів відсутня велика кількість функціональних можливостей, які прискорять та спростять процес пошуку працівників/роботи.
Проаналізувавши аналіз ПЗ, можна виділити основні функціональні можливості для розроблюваного ПЗ:

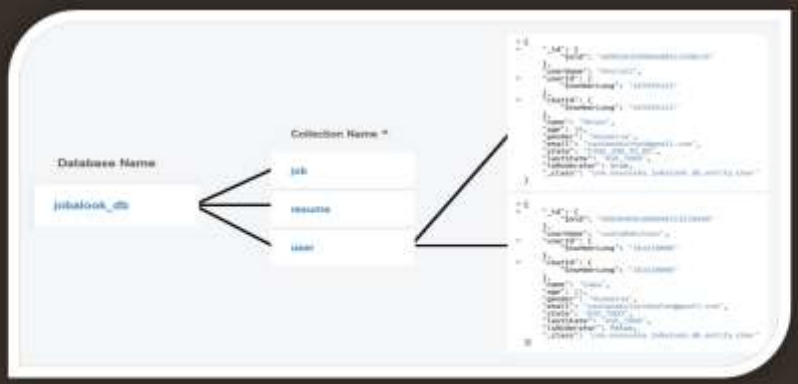
1) можливість публікації оголошень про пошук працівників/роботи;
2) сучасний, зручний та інтуїтивно зрозумілий інтерфейс;
3) простий набір параметрів для пошуку;
4) можливість відображення контактних даних для спілкування в месенджері;
5) можливість розбиття на сторінки результату пошуку;
6) швидка та зрозуміла відповідь бота.

Діаграма варіантів використання



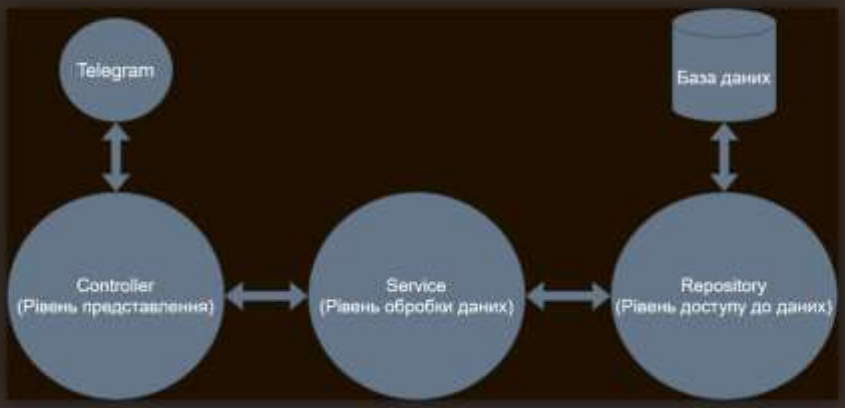
Модель бази даних

При виборі бази даних для серверної частини ПЗ було вибрано нереляційну БД – MongoDB.



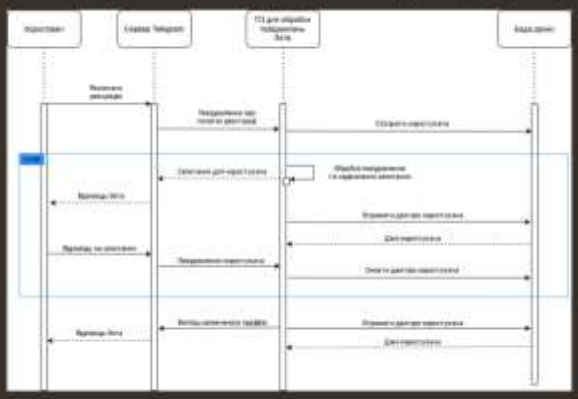
Архітектура програмного забезпечення

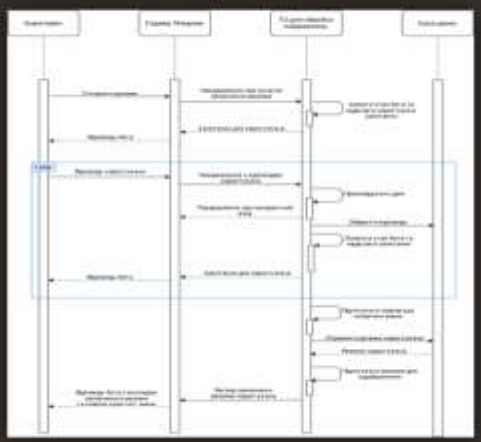
Для реалізації серверної частини програмного забезпечення було вирішено використати монолітний архітектурний стиль. Програмна система поділена на три рівні за допомогою шаблону "Controller-Service-Repository".



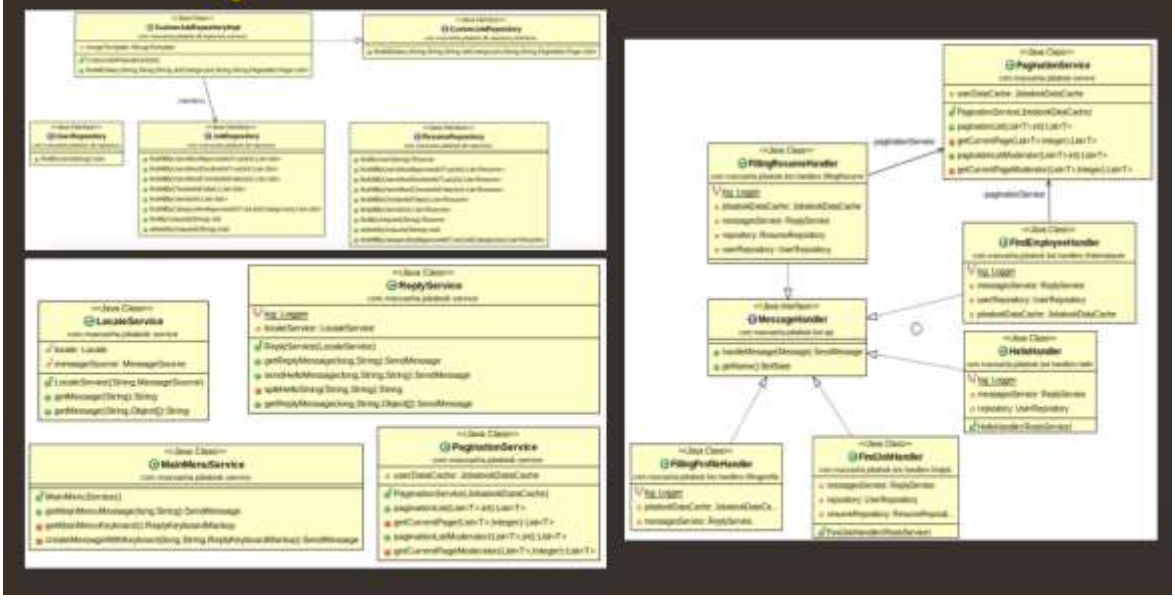
Проектування

Розглянемо взаємодію користувача з ПЗ, а саме: реєстрацію користувача, пошук роботи за категоріями, створення вакансії, редагування вакансії, перевірку публікацій на коректність.





Діаграми класів

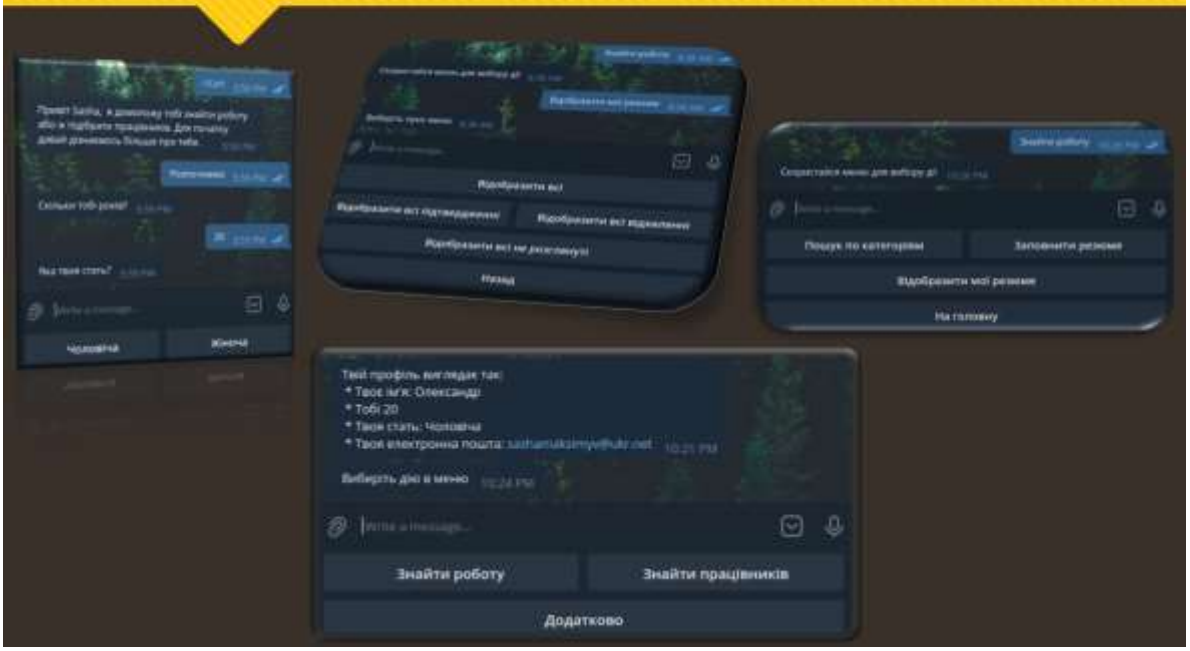


Технології реалізації ПЗ

Для розроблюваного ПЗ було вирішено використати об'єктно-орієнтовану мову програмування Java та низку додаткових інструментів для спрощення написання коду:

1) Spring Test
2) Spring Boot
3) Spring Web
4) Spring Data JPA
5) Lombok
6) Gradle

Вигляд інтерфейсу бота



Висновки

У дипломному проекті досліджено і проаналізовано предметну область, усі функціональні та нефункціональні особливості. Також було виконано аналіз наявного програмного забезпечення, розглянуто його переваги і недоліки та доведено актуальність розробки нового програмного забезпечення для пошуку роботи та персоналу. На основі наявного ПЗ було сформовано та описано основні функціональні вимоги, до розроблюваного ПЗ за допомогою діаграми варіантів використання, розроблено Технічне завдання.

При проектуванні було розглянуто сучасні архітектурні рішення, порівняно їх переваги та недоліки і визначено, що для взаємодії з месенджером Telegram буде використано клієнт-серверний тип архітектури та монолітний стиль для побудови серверної частити ПЗ. Також було проаналізовано типи баз даних, порівняно їх та вирішено, що для реалізації ,проекту краще використовувати нереляційну базу даних MongoDB через її зручність у використанні та швидкість виконання запитів.

Окрім вищезгаданого було спроектовано та розроблено основні елементи користувацького інтерфейсу, вигляд яких був поданий за допомогою макетів.

В результаті виконання дипломного проекту було реалізоване програмне забезпечення, яке забезпечує простий та швидкий пошук роботи або працівників за допомогою месенджера Telegram.

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

здобувача вищої освіти
Максимова О. В.
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 збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.06.2021р
дата

підпис

Sun Jun 06 13:43:39 EEST 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 3.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 22%**

	Документ		Сумарний збіг по Базі Даних	
ID: 92377 Назва: Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота Додано в БД: 2021-06-06 Автора: О. В. Максимів Керівники: Г. І. Радельчук Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	133474	1828	7042 (5%)	111 (6%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1008196325

Дата перевірки:
06.06.2021 14:54:16 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
06.06.2021 15:07:52 EEST

ID користувача:
100005589

Назва документа: **Дипломний проект ІПЗ-17-1 Максимів О.В**

Кількість сторінок: **131** Кількість слів: **25104** Кількість символів: **216733** Розмір файлу: **5.90 MB** ID файлу: **1008272523**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.98%
Схожість

Найбільша схожість: **4.07%** з джерелом з Бібліотеки (ID файлу: **1008265193**)

3.19% Джерела з Інтернету	525	Сторінка 133
7.13% Джерела з Бібліотеки	67	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	29
Підозріле форматування	29 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Максимів Олександр Володимирович

Тема Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 129

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті було досліджено і проаналізовано предметну область, усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Було розглянуто інструменти для реалізації дипломного проекту, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконаний відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломноого проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. В четвертому розділі було виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки на сьогодні в Україні Telegram-боти для пошуку роботи не є достатньо розвинутими та не мають достатньої кількості функціональних можливостей. Також було застосовано новітні технології для побудови програмного продукту та актуальні архітектурні рішеня.

5. Негативні сторони проекту У проекті пошук публікацій був реалізований лише за категоріями – було б доцільно додати розширений пошук. Також краще надсилати повідомлення роботодавцю, коли користувач подав заявку на роботу.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломного проекту та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про дипломний проект в цілому Дипломний проект заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики дипломного проекту. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка дипломного проекту Дипломний проект виконаний у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Мартинюк Валерій Володимирович, доктор технічних наук, професор, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій і телекомунікацій (АКІТ і ТК)

"02" 06 2021 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: «Програмне забезпечення для пошуку роботи та підбору персоналу з реалізацією інтерфейсу у вигляді Telegram-бота»

Автор: Максимів Олександр Володимирович

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»

Науковий керівник: Радельчук Галина Іванівна, канд. техн. 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 у стандартних бланках (титулка, бланк завдання на проектування, відомість документів), у структурі ЗМІСТУ, написах в рамках, назвах розділів/підрозділів тощо) та в назвах переліку джерел посилання;

2) в якості запозичень системою зафіксовано деякі послідовності вихідного коду, які є стандартними мовними конструкціями, спільними для великої кількості задач, і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформлені посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/схожості, складає 8,98% і адресується до 525 першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ Г. І. Радельчук

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк