Прикарпатський національний університет імені Василя Стефаника
Факультет математики та інформатики
Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
І-го рівня освіти
на тему:
“Розробка чат-бота для пошуку роботи і написання резюме”

Виконав: студент IV курсу гр.
ІСТ-41
спеціальності 126 “Інформаційні системи та технології”
Худицький О.В.

Науковий керівник: наук. ступінь,
доц. к.т.н. Превисокова Н.В.

Рецензент:
доц. к.т.н. Ровінський В.А.

м. Івано-Франківськ – 2025

АНОТАЦІЯ

У бакалаврській кваліфікаційній роботі представлено розробку Telegram-бота для автоматизації взаємодії між шукачами роботи та роботодавцями. Основні функції системи: створення резюме, пошук вакансій, подання відгуків та їх обробка в середовищі месенджера Telegram.

Об’єкт дослідження — процес автоматизації комунікації у сфері працевлаштування, предмет — чат-бот, що реалізує відповідний функціонал. Мета роботи — створення програмного забезпечення, що забезпечує автоматизовану взаємодію між кандидатами та роботодавцями з можливістю формування резюме.

Результатом є Telegram-бот з функціями створення й редагування резюме, пошуку вакансій, подачі відгуків та комунікації між кандидатами й роботодавцями.

ABSTRACT

The bachelor's thesis presents the development of a Telegram bot designed to automate interaction between job seekers and employers. The system's main functions include resume creation, job search, response submission, and processing within the Telegram messenger.

The research object is communication automation in employment; the subject is a chatbot that implements the relevant functionality. The goal is to develop software that facilitates automated interaction with resume generation features.

The result is a Telegram bot supporting resume creation/editing, job search, response submission, and interaction between candidates and employers.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації процесів пошуку роботи виникає потреба у розробці автоматизованих засобів взаємодії між кандидатами та роботодавцями. Серед таких засобів окрему категорію становлять чат-боти, які дозволяють користувачу ініціювати пошук вакансій, отримати рекомендації, сформувати та надіслати резюме безпосередньо через інтерфейс месенджера. Відомі реалізації подібних систем зазвичай виконують обмежений набір функцій або не враховують контекст персональних даних користувача при формуванні резюме. Це обумовлює доцільність створення рішення, яке інтегрує функціональність пошуку вакансій, збереження резюме та керування відгуками у межах одного інтерфейсу.

Автором роботи виконано повний цикл дослідження, проєктування та реалізації програмного забезпечення, що включає інтеграцію з базою даних, реалізацію логіки взаємодії з користувачем та формування персоналізованих резюме.

Об’єктом дослідження є процес автоматизації комунікації між здобувачами та роботодавцями у сфері працевлаштування. Предметом дослідження є інформаційна система у вигляді чат-бота, що реалізує функціонал пошуку вакансій, збереження анкет користувача та створення резюме.

Метою роботи є створення програмного забезпечення, яке забезпечує автоматизовану взаємодію між шукачами роботи та роботодавцями через чат-бот з можливістю формування резюме.

Для досягнення поставленої мети необхідно виконати такі завдання:

– визначити функціональні та нефункціональні вимоги до системи;

– реалізувати програмну логіку взаємодії з користувачем у рамках Telegram-бота;

– розробити та впровадити модуль створення резюме з параметрів, отриманих від користувача;

На момент написання роботи в інформаційно-аналітичних джерелах представлені окремі рішення з використанням ботів для автоматизації процесів пошуку вакансій, однак здебільшого такі реалізації мають обмежений функціонал, не підтримують повний цикл роботи з резюме або не надають можливості зберігати історію взаємодій користувача.

Методи дослідження включають моделювання логіки чат-бота, використання структурованого підходу до розробки, застосування принципів об’єктно-орієнтованого програмування, використання реляційної моделі даних та тестування працездатності реалізованих модулів.

Практичне значення отриманих результатів полягає у можливості використання розробленого чат-бота для автоматизації базових процесів працевлаштування. Результати можуть бути адаптовані для подальшого впровадження в онлайн-платформи працевлаштування, HR-сервіси або інтегровані у внутрішні системи підбору персоналу.

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел, до якого включено 35 позицій, а також містить 1 таблицю, 18 рисунків, 11 лістингів та 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Принципи роботи чат-ботів у сфері працевлаштування

Чат-бот є програмним інструментом, який автоматизує комунікацію з користувачами у текстовому або голосовому форматі. У сфері працевлаштування чат-боти використовуються для обробки запитів кандидатів, автоматизації рекрутингових процесів, збору інформації, попереднього аналізу резюме та надання рекомендацій. Їх основна функція полягає у симулюванні розмови з користувачем для забезпечення доступу до сервісів без участі людини.

Основу роботи чат-бота становлять алгоритми обробки природної мови (Natural Language Processing, NLP), які дозволяють інтерпретувати запити користувача. У більшості випадків застосовуються бібліотеки або платформи, що забезпечують парсинг вхідного повідомлення, його класифікацію за намірами (intents) та виділення сутностей (entities), що дозволяє визначити зміст запиту.

Залежно від ступеня складності чат-боти поділяються на скриптові та розмовні. Скриптові чат-боти працюють за наперед заданим алгоритмом і використовують фіксовані сценарії діалогів. Розмовні чат-боти мають можливість адаптуватися до різних варіантів запитів користувача, застосовують машинне навчання для побудови більш природної комунікації [11].

У сфері працевлаштування чат-боти можуть виконувати функції відбору кандидатів. Наприклад, вони можуть запитувати про досвід роботи, навички, побажання щодо посади, графіку чи місця праці. Отримані дані зберігаються у структурованому вигляді та використовуються для подальшого аналізу.

Однією з функціональних можливостей чат-ботів є попередня перевірка відповідності кандидатів до вимог вакансії. На основі

критеріїв, таких як освіта, досвід, мова програмування чи місце проживання, бот може зробити висновок про релевантність резюме.

Чат-боти також використовуються для автоматичного складання резюме. Користувач надає інформацію у вигляді відповідей на запитання, після чого система формує структурований документ у форматі, прийнятому для рекрутингу.

При реалізації чат-бота необхідно враховувати архітектуру взаємодії з базою даних. У випадку з пошуком роботи база даних зберігає інформацію про кандидатів, вакансії, компанії та результати взаємодії ботів з користувачами. У процесі розробки особливу увагу приділяють захисту персональних даних.

Інтерфейс чат-бота зазвичай реалізується у месенджерах або через вебінтерфейси. Найпоширенішими платформами є Telegram, Facebook Messenger, WhatsApp та вбудовані віджети на сайтах компаній [1].

Для реалізації сценаріїв поведінки чат-ботів у сфері працевлаштування можуть застосовуватись платформи автоматизації, зокрема Microsoft Bot Framework, Google Dialogflow або open-source рішення на базі Python, такі як Rasa чи Botpress.

У роботі чат-ботів необхідним є збереження контексту діалогу. Це дає змогу будувати розмову у кілька кроків, враховуючи попередні відповіді користувача, та адаптувати подальші запитання відповідно до отриманої інформації.

Особливістю чат-бота для працевлаштування є наявність функцій як з боку пошукачів, так і з боку роботодавців. Для перших забезпечується збір інформації, пошук вакансій, зберігання резюме. Для других — отримання анкет кандидатів, сортування, аналіз та ініціація контактів.

Окремим функціональним блоком є фільтрація вакансій. Чат-бот, отримавши параметри від користувача, виконує запит до бази вакансій і повертає список, що відповідає заданим критеріям. Це дозволяє автоматизувати процес пошуку без перегляду великої

кількості нерелевантних пропозицій.

У розробці таких ботів часто використовується концепція finite-state machine (кінцева автоматна модель) [19], яка дозволяє визначити стани діалогу та переходи між ними в залежності від дій користувача.

Для покращення взаємодії з користувачем застосовуються шаблони відповідей та попередньо сформовані варіанти вибору (кнопки, меню), що мінімізує помилки введення та пришвидшує взаємодію.

В контексті працевлаштування чат-бот також виконує роль персонального асистента. Він може надсилати повідомлення про нові вакансії, нагадувати про необхідність оновлення профілю або рекомендувати покращення у резюме [7].

Інформаційна архітектура системи з чат-ботом включає рівень взаємодії (інтерфейс користувача), логічний рівень (обробка запитів), рівень даних (зберігання) та зовнішні сервіси (API, ML-сервіси). Таке розділення забезпечує гнучкість у розширенні та супроводі застосунку.

Таким чином, принципи роботи чат-ботів у сфері працевлаштування охоплюють обробку природної мови, збереження контексту, взаємодію з базами даних, генерацію документів, класифікацію запитів та побудову діалогу з користувачем. Їх реалізація базується на технічних рішеннях, які забезпечують масштабованість, стійкість та відповідність функціональним вимогам.

1.2. Огляд існуючих рішень з автоматизації пошуку роботи

Існують сервіси, що пропонують більш прості та швидкі рішення для пошуку роботи, крім великих платформ і соціальних мереж. Вони дозволяють кандидатам оперативно знаходити відповідні вакансії, а роботодавцям — підбирати підходящих кандидатів.

Joblist є онлайн-ресурсом для пошуку роботи, який дозволяє кандидатам швидко знаходити вакансії на основі заданих фільтрів. Інтерфейс цього сервісу простий: користувач вказує основні параметри пошуку, такі як професія, рівень зарплати та місце роботи, і сервіс автоматично підбирає відповідні вакансії. Joblist надає спрощений механізм подачі заявки без необхідності реєстрації чи додаткових кроків [14]. Користувач може подати заявку безпосередньо через сайт. Інтерфейс цього сервісу, де відображаються вакансії після пошуку за параметрами, наведено на рисунку 1.1.

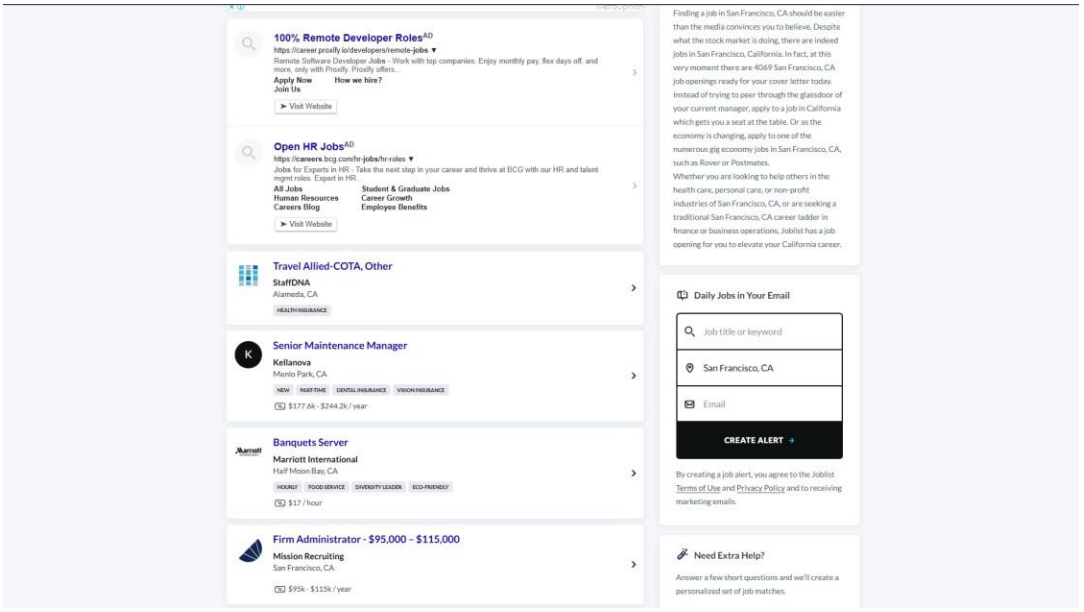

Рисунок 1.1 – Інтерфейс Joblist

Xpert є ще одним рішенням для автоматизації пошуку вакансій. Цей сервіс дозволяє створювати персоналізовані профілі для користувачів, включаючи їх навички та досвід роботи, і на основі цього автоматично отримувати пропозиції від роботодавців. Платформа використовує алгоритми машинного навчання для покращення відповідності вакансій до запитів користувача. Інтерфейс сервісу дозволяє кандидату швидко переглядати вакансії та подавати заявки без необхідності додаткових реєстрацій чи перевірок [15]. Головну сторінку цього сервісу наведено на рисунку 1.2.

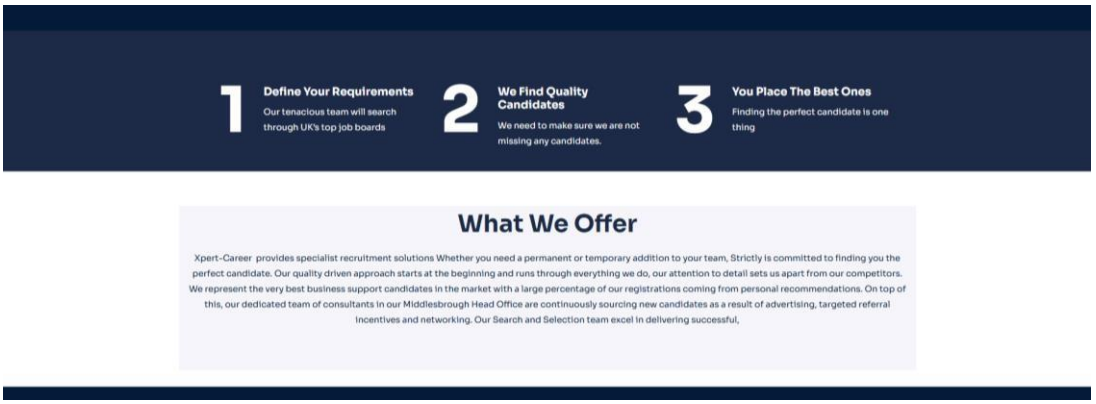

Рисунок 1.2 – Головна сторінка сервісу Xpert

JobBot є чат-ботом в Telegram, що надає можливість користувачам шукати вакансії, застосовуючи прості фільтри. Для пошуку роботи користувачам потрібно лише вказати основні параметри вакансії, після чого бот автоматично надасть список відповідних пропозицій. Користувач може подати заявку на вакансію без необхідності переходити на інші платформи чи реєструватися в додаткових системах. Бот підтримує інтеграцію з кількома джерелами вакансій і забезпечує спрощений процес пошуку роботи.

Ці три рішення мають схожі характеристики: вони автоматизують процес пошуку роботи, дозволяючи користувачам шукати вакансії за допомогою простих параметрів та отримувати пропозиції відповідно до своїх вподобань. Всі ці інструменти пропонують мінімальний набір функцій, що дозволяє кандидату швидко знайти роботу, не витрачаючи багато часу на реєстрації чи додаткові налаштування.

1.3. Визначення вимог до функціоналу чат-бота

Проєктування чат-бота для автоматизації пошуку роботи потребує попереднього визначення вимог до функціоналу системи. Формулювання вимог виконується на основі аналізу предметної області, типових сценаріїв використання, а також врахування потреб кінцевих користувачів. До системи ставляться як загальні вимоги, пов’язані з обробкою запитів і взаємодією з користувачем, так і специфічні вимоги, що визначають набір функцій чат-бота.

Основною групою користувачів є особи, які шукають роботу. Вони взаємодіють із чат-ботом через інтерфейс месенджера. Для реалізації взаємодії необхідно забезпечити можливість ініціалізації діалогу, надання запиту на пошук вакансії, фільтрацію за галуззю, посадою, місцем розташування, типом зайнятості та іншими параметрами. Кожен із параметрів має бути підтриманий як окремий елемент логіки.

Система має зберігати мінімальний обсяг інформації про користувача. Зберігання персональних даних обмежується унікальним ідентифікатором, ім’ям або псевдонімом, а також опціонально контактними даними, якщо це передбачено логікою відгуку на вакансію [9].

Після задання параметрів пошуку користувач повинен отримувати релевантний список вакансій. Результати мають бути представлені у структурованому вигляді з коротким описом, найменуванням компанії, посадою, місцем роботи та способом зв’язку. При потребі надається можливість перегляду детальнішої інформації про вакансію через кнопку або текстову команду [6].

Окрім перегляду вакансій, бот повинен дозволяти надсилати резюме або контактну інформацію у відповідь на вакансію. Для цього передбачається механізм створення резюме або його прикріплення. Система повинна забезпечувати обробку відповіді та передавання її

відповідальному за вакансію користувачеві з боку роботодавця.

Система має використовувати інформацію з резюме для автоматичного підбору відповідних вакансій. Користувач отримує список позицій, які відповідають зазначеним характеристикам. Це дозволяє скоротити обсяг взаємодії через чат та зменшити кількість необхідних кроків при пошуку.

Чат-бот повинен підтримувати обробку помилок. У випадку некоректного запиту система має надати користувачеві зрозуміле повідомлення з описом можливих дій. Це дозволяє уникнути збоїв при взаємодії з ботом і забезпечити стабільність виконання запитів [8].

Окрім пошуку вакансій, бот може надавати користувачам довідкову інформацію. Зокрема, перелік доступних команд, опис процесу створення резюме або правила подання заявки. Ця інформація має бути доступною за запитом.

З боку адміністратора або роботодавця система повинна підтримувати функцію розміщення вакансій. Для цього необхідно реалізувати інтерфейс введення вакансії із зазначенням обов’язкових параметрів, таких як назва, опис, місце роботи, тип зайнятості, контактна особа. Дані зберігаються в базі і стають доступними для пошуку.

Роботодавець має мати можливість переглядати відгуки на вакансії, що були опубліковані ним. Для цього система повинна забезпечити доступ до відповідних даних у структурованому вигляді. У разі необхідності можливе видалення вакансії або її редагування [34].

Вимоги до адміністрування чат-бота включають функції моніторингу, перегляду активності користувачів, кількості запитів, а також обробки технічних збоїв. Це дозволяє підтримувати працездатність системи та виконувати аудит звернень.

З точки зору безпеки, система не повинна зберігати конфіденційні дані без належного шифрування або відповідного механізму захисту. Доступ до адміністративних функцій повинен

обмежуватись автентифікацією через токен або пароль.

Інформаційна архітектура бота передбачає наявність бази даних, яка зберігає вакансії, відгуки, користувачів та їхній статус у діалозі. Структура бази повинна бути уніфікованою та придатною для масштабування у разі розширення функціоналу [13].

Функціонал надсилання повідомлень передбачає використання API месенджера для відправки тексту, кнопок, зображень або документів. Надсилання повідомлень здійснюється відповідно до поточного стану діалогу та заданих параметрів користувача.

Кожен діалог із користувачем розглядається як окрема сесія. У рамках сесії зберігаються обрані параметри, поточний крок взаємодії та ідентифікатор користувача. Це дозволяє реалізувати персоналізований сценарій взаємодії.

Для підтримки масштабування та подальшої інтеграції із зовнішніми ресурсами (наприклад, базами вакансій) система повинна мати модульну архітектуру з чітко визначеними інтерфейсами обміну даними.

Механізм логування подій дозволяє фіксувати дії користувачів, помилки та запити до API. Журнали подій зберігаються в базі даних або у вигляді лог-файлів і можуть бути використані для діагностики або аналітики [32].

Для забезпечення безперервності роботи чат-бота передбачається використання черг обробки запитів або асинхронного виконання задач. Це дає змогу уникати блокування при великій кількості користувачів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ЧАТ-БОТА ДЛЯ ПОШУКУ РОБОТИ

2.1. Вибір архітектурного підходу та технологій реалізації

Реалізація чат-бота для пошуку роботи базується на використанні Telegram API, Python-бібліотеки aiogram, ORM-фреймворку SQLAlchemy та системи керування базами даних MySQL. Застосування вказаних технологій обумовлене вимогами до асинхронної взаємодії з користувачем, підтримки розгалуженої бізнес-логіки та роботи з реляційною структурою даних.

Telegram API є програмним інтерфейсом, що дозволяє здійснювати обмін повідомленнями між користувачами та ботами. Telegram надає RESTful API, що підтримує надсилання та отримання повідомлень, опрацювання подій, взаємодію з кнопками, інлайн-режимами та іншими функціональними можливостями. API дозволяє розробникам створювати ботів, які працюють з повідомленнями в реальному часі, надсилають медіа, приймають відповіді користувача через клавіатури або callback-події. Telegram API не потребує окремого схвалення, крім авторизації токеном, що видається через BotFather [33].

Обмін даними між сервером застосунку та інфраструктурою Telegram здійснюється одним з двох способів: webhook або довгим опитуванням (long polling). У реалізованому рішенні використано long polling. Це означає, що бот-застосунок самостійно періодично надсилає запити до API Telegram з метою отримання нових повідомлень.

Для побудови серверної частини застосовано бібліотеку aiogram [25]. Вона є Python-фреймворком, що дозволяє реалізовувати асинхронних Telegram-ботів із використанням asyncio. Бібліотека містить засоби для реєстрації хендлерів, опрацювання команд,

callback-даних, формування відповіді користувачу та організації маршрутизації повідомлень. Aiogram підтримує модульну структуру, що дозволяє логічно розділити код на незалежні блоки. Обробники подій можуть бути організовані в окремі файли або пакети за типом сценарію. Aiogram має вбудовану підтримку FSM (машини станів), яка застосовується для реалізації сценаріїв з кількома етапами взаємодії [4].

Асинхронність реалізації дозволяє уникнути блокування потоку при зверненнях до зовнішніх ресурсів, зокрема Telegram API або бази даних. Це забезпечує низькі затримки обробки запитів навіть при високому навантаженні.

Для взаємодії із СКБД MySQL використано SQLAlchemy. Це ORM-фреймворк для Python, який дозволяє здійснювати роботу з базою даних за допомогою Python-об’єктів, не використовуючи сирі SQL-запити. SQLAlchemy підтримує генерацію схем, автоматичну міграцію, зв’язки між таблицями та транзакційну обробку запитів. У реалізованому проєкті застосовується декларативний підхід — опис таблиць виконується у вигляді Python-класів із полями, що відповідають стовпцям бази даних. Для асинхронної взаємодії з MySQL використано драйвер asyncmy, який забезпечує неблокуючий доступ до бази даних [30].

ORM-рівень дозволяє ізолювати логіку доступу до даних від основного функціоналу бота. Кожна таблиця бази даних має відповідну модель у Python-коді. Це дозволяє централізовано керувати збереженням, оновленням, видаленням та вибіркою даних. Зв’язки між сутностями, наприклад між користувачами та їхніми резюме або відгуками, також описано у вигляді реляційних зв’язків моделей.

База даних MySQL містить чотири основні сутності: користувачі, вакансії, резюме, відгуки. Кожна сутність реалізується як окрема таблиця з первинним ключем, а також зовнішніми ключами для реалізації зв’язків. Наприклад, резюме містить зовнішній ключ на користувача, якому воно належить. Відгуки містять посилання на

вакансію та резюме, що було надіслано у відповідь.

У проєкті реалізовано розділення коду на функціональні модулі. Усі команди, callback-події та інші обробники згруповано в окремі файли відповідно до їх призначення. Така структура спрощує обслуговування та модифікацію логіки взаємодії з користувачем. Наприклад, команди загального доступу, доступні всім користувачам, розміщено в одному модулі, а callback-події для роботодавця або кандидата — в інших.

Aiogram дозволяє використовувати Dispatcher — механізм маршрутизації, який обирає потрібний хендлер залежно від типу вхідного повідомлення. Dispatcher також може застосовувати middlewares — проміжні обробники, які виконуються до та після основного хендлера. Це дає змогу реалізувати, наприклад, логування, авторизацію або перевірку ролей користувача.

Ініціалізація бази даних виконується окремим методом init_models, який створює всі необхідні таблиці за описаними ORM-моделями. Для керування сесіями бази даних використовується об'єкт async_session, який дозволяє створювати та завершувати асинхронні транзакції з MySQL.

У боті реалізовано рольову модель. При взаємодії з користувачем перевіряється його тип (роботодавець або кандидат), після чого надається відповідний доступ до функціоналу. Це реалізовано як у вигляді логіки маршрутизації повідомлень, так і на рівні перевірки доступу при виконанні операцій над базою даних.

Для зберігання стану взаємодії з користувачем застосовується FSM, яка дозволяє контролювати етапи створення резюме, пошуку вакансій або надсилання відгуків. Кожен стан має окрему логіку обробки вхідних повідомлень.

Таким чином, вибір технологій реалізації забезпечує асинхронну обробку запитів, логічну структуру коду, зручну роботу з реляційною базою даних та можливість масштабування системи.

З урахуванням описаних особливостей, нижче подано загальну

архітектуру застосунку, яка поєднує Telegram API, серверну частину на базі aiogram та СКБД MySQL.

У цій архітектурі бот виконує роль проміжного рівня між інтерфейсом користувача та серверною логікою. З одного боку, він приймає запити від клієнта Telegram і надсилає відповіді через API Telegram. З іншого боку, він підключається до серверної частини застосунку, де реалізовано бізнес-логіку та взаємодію з базою даних. Застосунок використовує асинхронну взаємодію з базою даних для зчитування та збереження інформації про користувачів, резюме, вакансії, відгуки.

Застосовано архітектурний підхід із чітким поділом на рівні: представлення (інтерфейс Telegram), логіки (бот-застосунок), даних (база даних). Така побудова забезпечує ізольованість компонентів, що спрощує підтримку, масштабування та повторне використання компонентів у майбутньому.

Серверна частина реалізована з використанням Python-фреймворку aiogram, що підтримує асинхронну обробку повідомлень Telegram API. Для взаємодії з базою даних застосовано SQLAlchemy у поєднанні з драйвером asyncmy [17] для MySQL. Такий стек дозволяє реалізувати неблокуючу обробку запитів, зменшуючи затримки при виконанні операцій читання та запису.

У базі даних зберігаються сутності, що описують користувачів, вакансії, резюме та відгуки. Для роботи з ними використано ORM-модель, що забезпечує зв’язок між таблицями бази даних та об’єктами Python-коду.

Уся логіка бота реалізована у вигляді окремих модулів, які розміщено у спеціальній структурі каталогів. Такий підхід дозволяє організовано підтримувати різні сценарії взаємодії користувача з ботом. Кожен модуль відповідає за окрему категорію функціоналу, наприклад, створення резюме, перегляд вакансій або відгуки на них.

Таким чином, архітектура проєкту складається з трьох основних компонентів: клієнт Telegram (через який здійснюється взаємодія

користувача), сервер Telegram (який перенаправляє повідомлення) та сервер бота, що реалізує бізнес-логіку і взаємодіє з базою даних.

На рисунку 2.1 наведено загальну схему архітектури чат-бота.

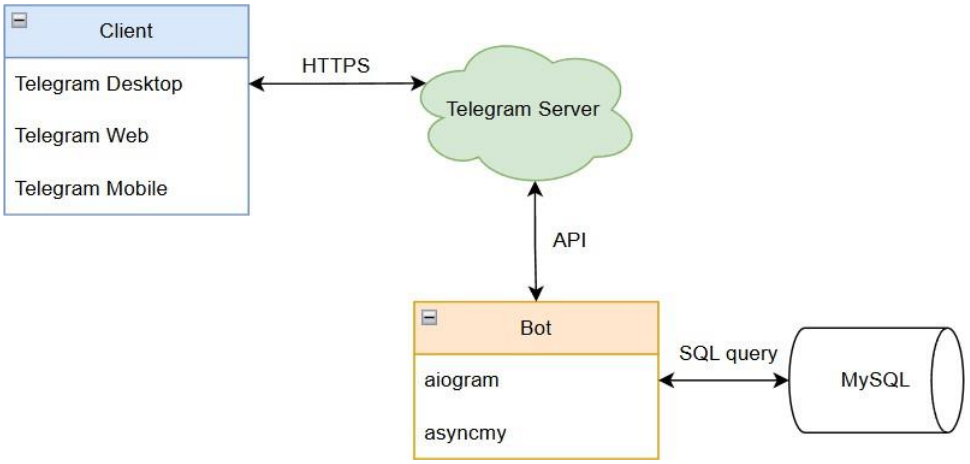

Рисунок 2.1 – Архітектура чат-бота для пошуку роботи

2.2. Структура зберігання даних та модель бази даних

У розробленому чат-боті використовується реляційна модель бази даних для зберігання інформації про користувачів, резюме, вакансії, відгуки та супутні категорії. Структура побудована з урахуванням того, що користувач може бути як кандидатом, так і роботодавцем, оскільки Telegram не передбачає розмежування типів акаунтів. Відповідно, кожен користувач має змогу створити одне резюме, а також публікувати одну або декілька вакансій.

Базова логіка взаємозв'язків між сутностями відображена на ERD-діаграмі класів [2], наведеної на рисунку 2.2. Сутність User пов'язана з Resume відношенням один до одного, оскільки в системі передбачено створення одного резюме для кожного користувача. Зв'язок User з Vacancy реалізується як один до багатьох, тобто один користувач може створити кілька вакансій. Відгук (Response) представляє зв'язок між резюме та вакансією, і є посередньою сутністю у відношенні багато-до-багатьох між цими двома сутностями. Для кожного відгуку також зберігається статус, який

фіксується в окремій таблиці ResponseStatus.

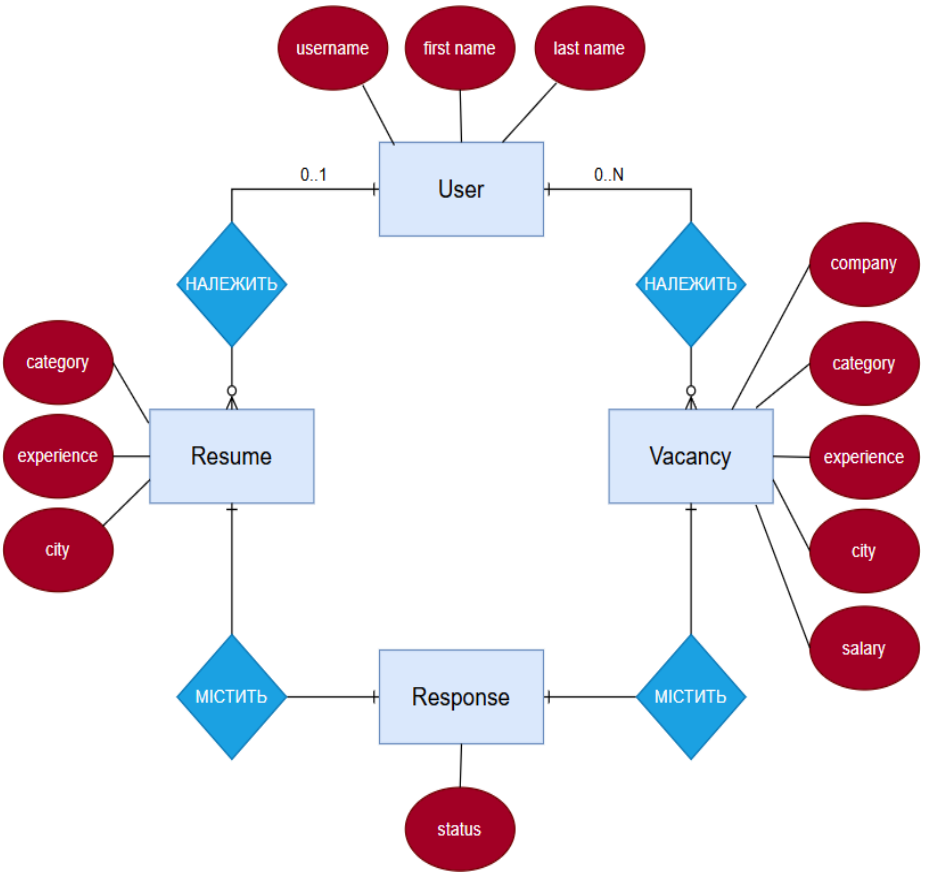

Рисунок 2.2 — ERD-діаграма сутностей чат-бота для пошуку роботи

Для реалізації зберігання даних використовується MySQL. Усі таблиці взаємопов’язані зовнішніми ключами. У логічній структурі бази даних кожна таблиця виконує окрему функцію: зберігання користувачів, вакансій, резюме, відгуків і категорій. Додатково, таблиця response_statuses дає змогу задати статус для кожного відгуку: наприклад, "новий", "переглянутий", "відхилений".

Таблиця users містить Telegram-ідентифікатор, який є унікальним і використовується для зв’язку з профілем користувача. Таблиця resumes зберігає характеристики резюме: позицію, досвід, навички, освіту, місто, бажаний тип зайнятості та очікувану заробітну плату. Кожне резюме пов’язане з категорією.

Таблиця vacancies реалізована аналогічно до resumes, однак містить додаткову інформацію про компанію та вимоги. Як резюме, так і вакансії можуть бути прив’язані до певної категорії через

зовнішній ключ category_id.

Таблиця responses реалізує зв’язок між резюме та вакансією. У цій таблиці зберігається час відгуку, статус перегляду та статус з таблиці response_statuses. Для уникнення дублювання передбачено обмеження у вигляді унікальної комбінації resume_id і vacancy_id.

Логічна структура таблиць [3] представлена на на рисунку 2.3. Діаграма ілюструє основні зв’язки між таблицями, типи зв’язків (один-до-багатьох, багато-до-одного) та зовнішні ключі.

Додатково, таблиця categories використовується для класифікації резюме та вакансій. Вона дозволяє організувати зміст системи за напрямами діяльності або спеціалізаціями. Зв’язок з категоріями реалізується за допомогою зовнішнього ключа category_id у відповідних таблицях.

Зовнішні ключі встановлені з обмеженням ON DELETE CASCADE для забезпечення консистентності при видаленні записів. Наприклад, при видаленні користувача автоматично видаляються його резюме, вакансії та пов’язані з ними відгуки. Для category_id у таблиці resumes задано ON DELETE SET NULL, що дозволяє зберегти резюме навіть у разі видалення категорії.

Усі часові поля створені з типом TIMESTAMP із значенням за замовчуванням CURRENT_TIMESTAMP, що дозволяє фіксувати дату створення кожного запису без додаткових дій зі сторони застосунку.

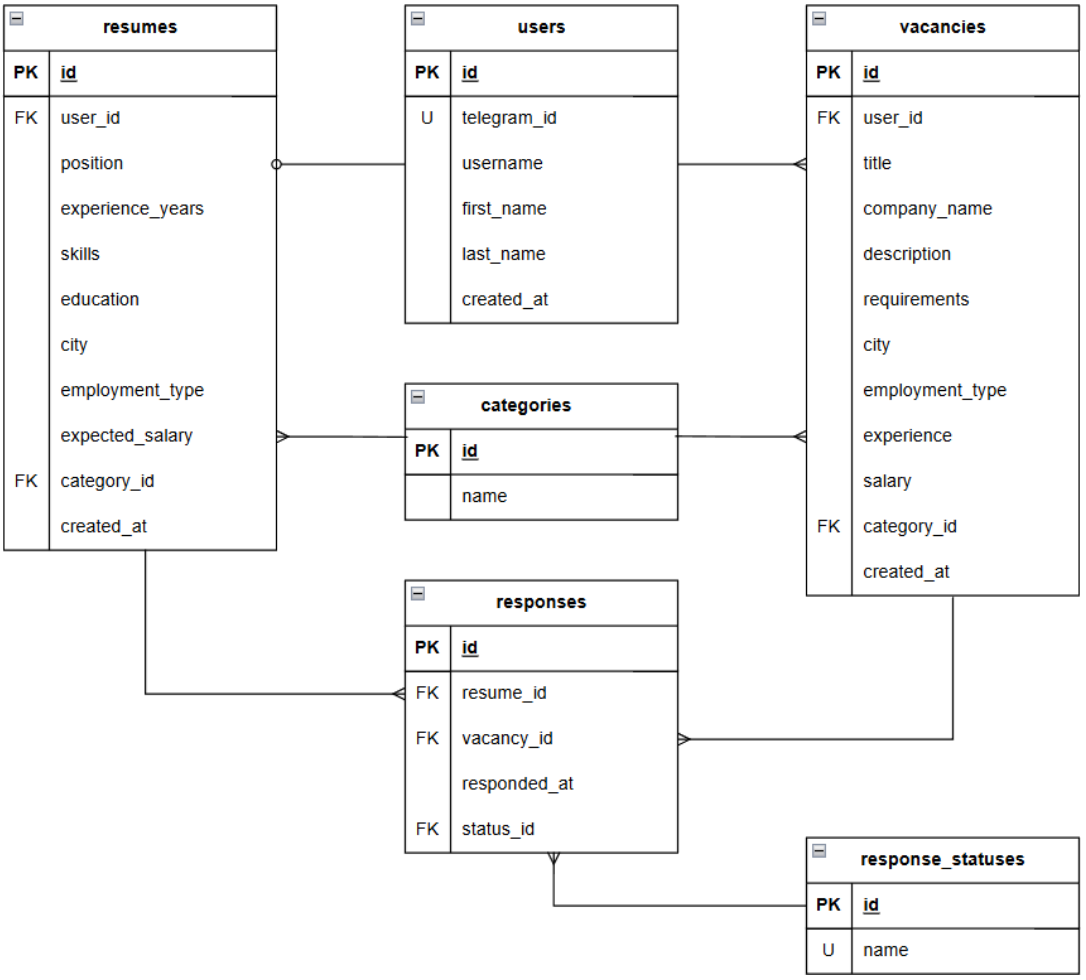

Рисунок 2.3 — Логічна структура таблиць бази даних чат-бота

Фізична структура бази даних відповідає реалізованій у MySQL схемі. Для її побудови використано інструмент зворотного проектування в MySQL Workbench [27], що дає змогу візуалізувати реальні назви таблиць, поля, типи даних і обмеження. Ця діаграма наведена на рисунку 2.4.

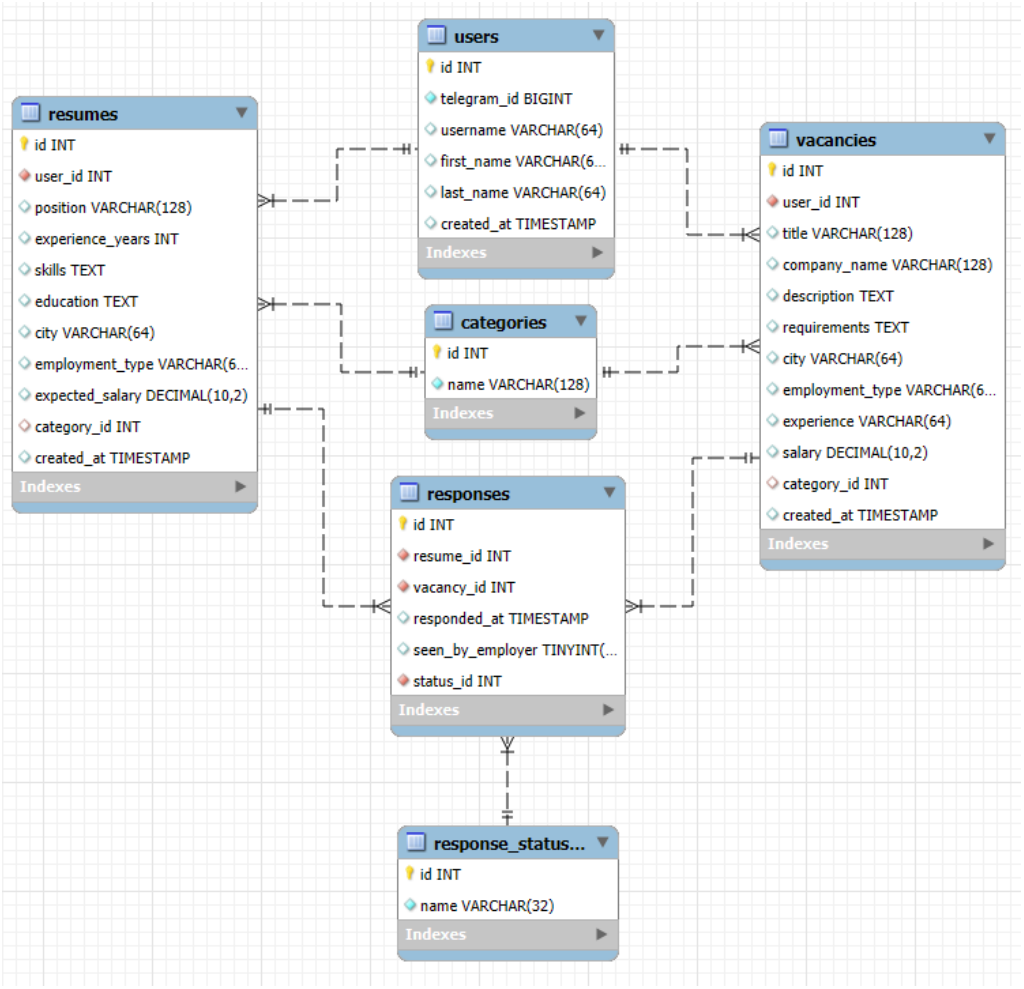

Рисунок 2.4 — Фізична структура бази даних

Сукупність сутностей та зв’язків забезпечує достатню гнучкість для реалізації основної логіки чат-бота: реєстрації користувача, створення резюме та вакансій, а також взаємодії між ними через відгуки.

2.3. Розробка алгоритмів взаємодії з користувачем

На рисунку 2.5 наведено діаграму послідовностей [21], яка ілюструє алгоритм взаємодії між основними учасниками системи: кандидатом, роботодавцем, чат-ботом та базою даних. Діаграма демонструє порядок викликів, що відбуваються під час створення резюме, перегляду вакансій, подачі відгуку, створення вакансій і зміни

статусів у системі.

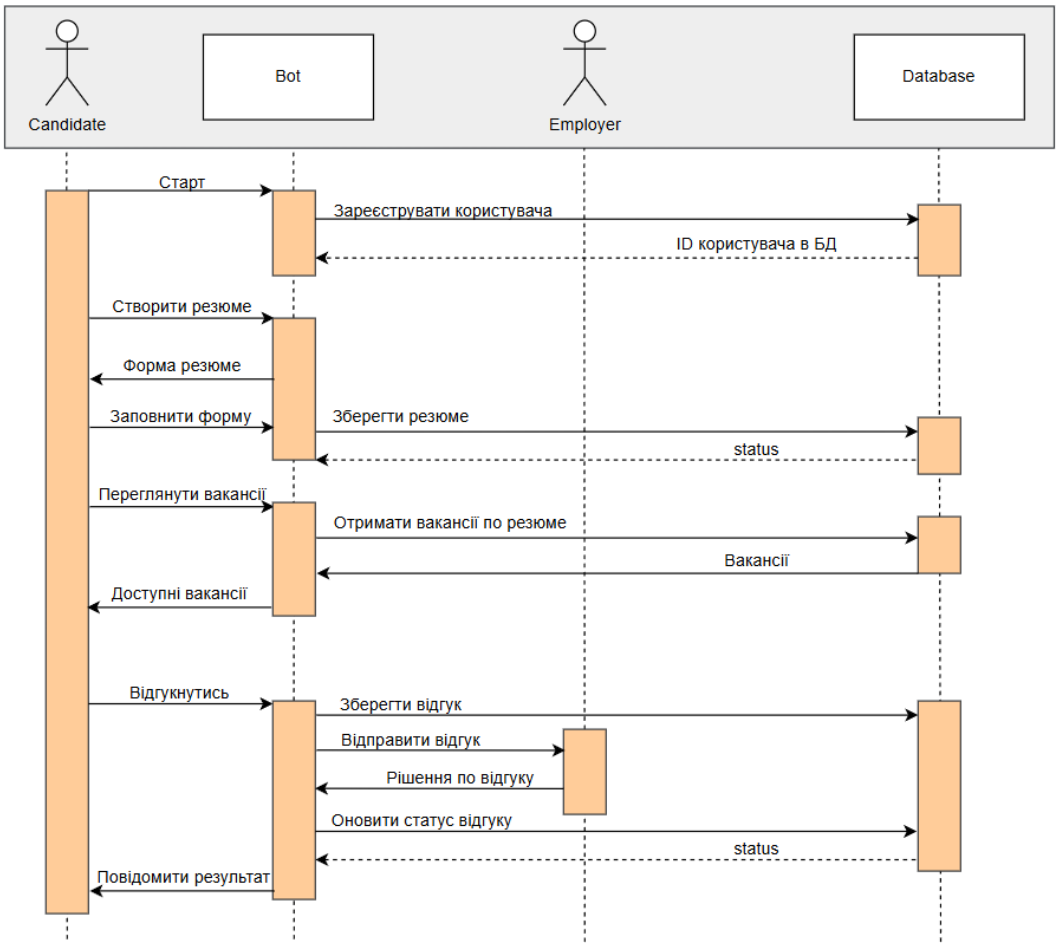

Рисунок 2.5 – Діаграма послідовностей

Взаємодія з користувачем у чат-боті реалізується як послідовність функціональних етапів. Основна роль відведена кандидату, який здійснює ініціацію дій через інтерфейс бота. Першою дією є створення резюме. Кандидат вводить персональні дані, включаючи назву бажаної посади, досвід, навички, освіту. Введена інформація перевіряється на наявність помилок, після чого зберігається у таблиці бази даних resumes. Запис резюме пов’язується з конкретним користувачем через поле user_id.

Після завершення формування резюме кандидат отримує доступ до перегляду вакансій. Бот здійснює вибірку з таблиці vacancies відповідно до параметрів, визначених користувачем, таких як категорія або географічна локація. Результати виводяться у вигляді повідомлень з інтерактивними елементами. Кожна вакансія містить

ідентифікатор для подальшої взаємодії.

У разі зацікавленості кандидат може сформувати відгук на вакансію. Система створює запис у таблиці responses, що включає зв’язок між резюме користувача та вибраною вакансією. Зберігається ідентифікатор вакансії, ідентифікатор резюме, дата подачі та поточний статус. Кандидат отримує підтвердження про реєстрацію відгуку.

На стороні роботодавця реалізовано можливість додавання вакансій, що супроводжується введенням назви, опису, вимог. Кожна вакансія записується у таблицю vacancies з прив’язкою до категорії через зовнішній ключ. Роботодавець також має можливість перегляду відгуків, що містяться у таблиці responses, та перегляду резюме, що пов’язані з конкретними відгуками.

Оновлення статусу відгуків реалізується шляхом модифікації значення поля status_id у таблиці responses. Це дозволяє фіксувати результат розгляду заявки роботодавцем. Статуси визначаються таблицею status та асоціюються з відповідним записом через зовнішній ключ.

Архітектура обміну даними базується на взаємодії чат-бота з Telegram API та базою даних MySQL. Збереження даних виконується через SQL-запити з використанням асинхронного підключення [5]. Модель бази даних містить таблиці для зберігання інформації про користувачів, категорії, вакансії, резюме, відгуки, статуси. Структура передбачає наявність зовнішніх ключів між сутностями для забезпечення логічної цілісності.

Усі дії користувача ініціюються через бот, який є точкою входу до системи. Через бот кандидат виконує послідовні дії з введення, перегляду та взаємодії з вакансіями, що забезпечує централізований підхід до обробки інформації без залучення інших клієнтських інтерфейсів.

Окрім наведеної раніше діаграми послідовностей, для уточнення логіки реалізації користувацької взаємодії побудовано ще

дві допоміжні діаграми: блок-схему [16] процесу створення резюме та діаграму станів (state diagram) [31] об'єкта відгуку.

На рисунку 2.5 зображено блок-схему процесу створення резюме кандидатом.

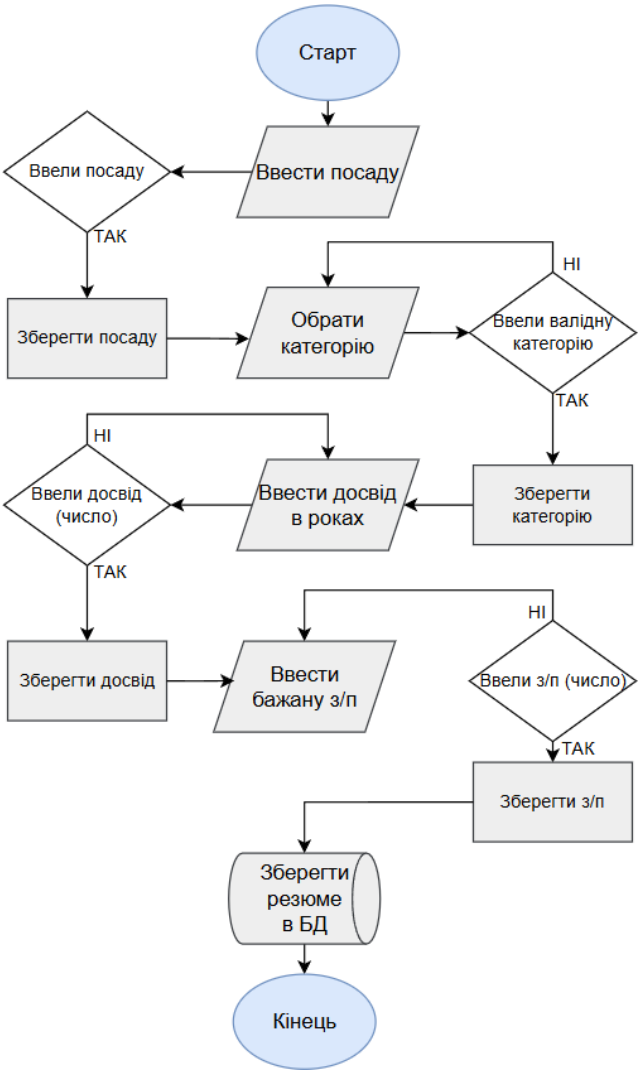

Рисунок 2.5 – Блок-схема процесу створення резюме

У даному алгоритмі реалізовано покрокове опитування користувача з перевіркою введених даних. На кожному етапі бот очікує введення окремого атрибута — посади, категорії, досвіду роботи у роках, бажаної заробітної плати. Після кожного введення виконується перевірка на відповідність вимогам до формату або наявності даних. У випадку невалідного введення бот повторює запит. У разі успішного заповнення — дані зберігаються, і користувач переходить до наступного етапу. Після завершення всіх етапів

формується об’єкт Resume, який зберігається у базі даних. Такий підхід дає змогу реалізувати обробку помилок введення без потреби повертатися до попередніх кроків або перезапускати процес з нуля.

На рисунку 2.6 представлено діаграму станів для об’єкта Response, що виникає при поданні відгуку кандидатом на вакансію.

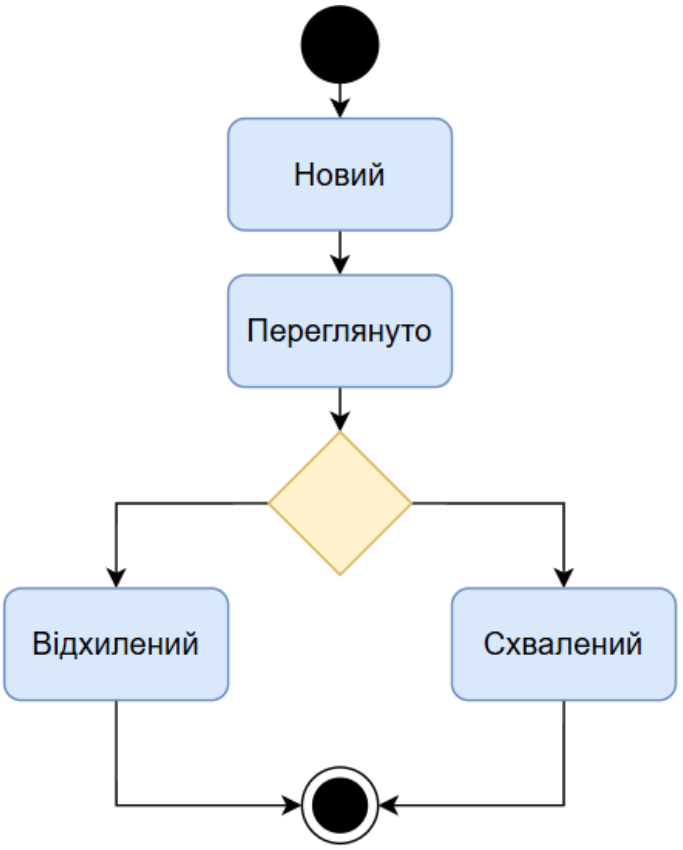

Рисунок 2.6 – Діаграма станів для об’єкта Відгук

Початковий стан об’єкта — «Новий». Після того, як роботодавець переглядає відгук, відбувається перехід у стан «Переглянуто». Далі, в залежності від дій роботодавця, стан змінюється або на «Схвалено», або на «Відхилено». Таким чином, визначено чітку модель життєвого циклу кожного відгуку з фіксацією поточного статусу для подальшої аналітики та сповіщення користувача.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЧАТ-БОТА

3.1. Розгортання серверної частини

Серверна частина чат-бота реалізована з використанням фреймворку aiogram та представлена у вигляді модульної структури, що забезпечує розподіл функціональності за призначенням. Архітектура програми передбачає логічне групування команд, обробників callback-запитів [12], а також додаткових допоміжних компонентів, необхідних для повноцінного функціонування системи.

Основна точка входу до застосунку реалізована у файлі main.py. У цьому файлі виконується ініціалізація диспетчера Dispatcher, підключення маршрутизатора router [28] з пакету routers, ініціалізація схеми логування, запуск підключення до бази даних та старт процесу опитування Telegram-сервера.

Маршрутизатори згруповано в окрему директорію routers/. Цей пакет містить файл __init__.py, у якому до об’єкта Router включаються три незалежні підмаршрутизатори:

- BaseCommands — містить базові текстові команди;
- Callbacks — містить обробники callback-запитів;
- CustomCommands — містить команди, доступні після вибору ролі користувача.

Пакет Callbacks/ містить три окремі підпакети, які відповідають за callback-обробку для різних груп користувачів або загальних сценаріїв: Common, Employer, Candidate. Кожен підпакет містить власний маршрутизатор, у який включаються відповідні обробники з файлів common_callbacks.py, employer_callbacks.py та candidate_callbacks.py.

Пакет CustomCommands/ також містить окремі підпакети CandidateCommands та EmployerCommands, а також модуль

role_selector.py, який відповідає за початковий вибір ролі користувача. У підпакеті CandidateCommands реалізовано такі підмодулі: history.py, view_vacancies.py, manage_resume.py, кожен з яких містить окремий маршрутизатор. Всі маршрутизатори включаються у відповідному порядку до єдиного об’єкта router пакету CustomCommands.

Окрім маршрутизаторів, структура проєкту містить допоміжні модулі:

– config.py — зберігає параметри конфігурації системи, включаючи токен бота та налаштування БД;

– models.py — містить опис ORM-моделей для роботи з базою даних за допомогою SQLAlchemy;

– states/global_states.py — визначає стан машини станів для покрокової взаємодії з користувачем.

На рисунку 3.1 наведено логічну структуру директорій та модулів серверної частини застосунку. Вона ілюструє модульність реалізації та залежності між основними компонентами системи.

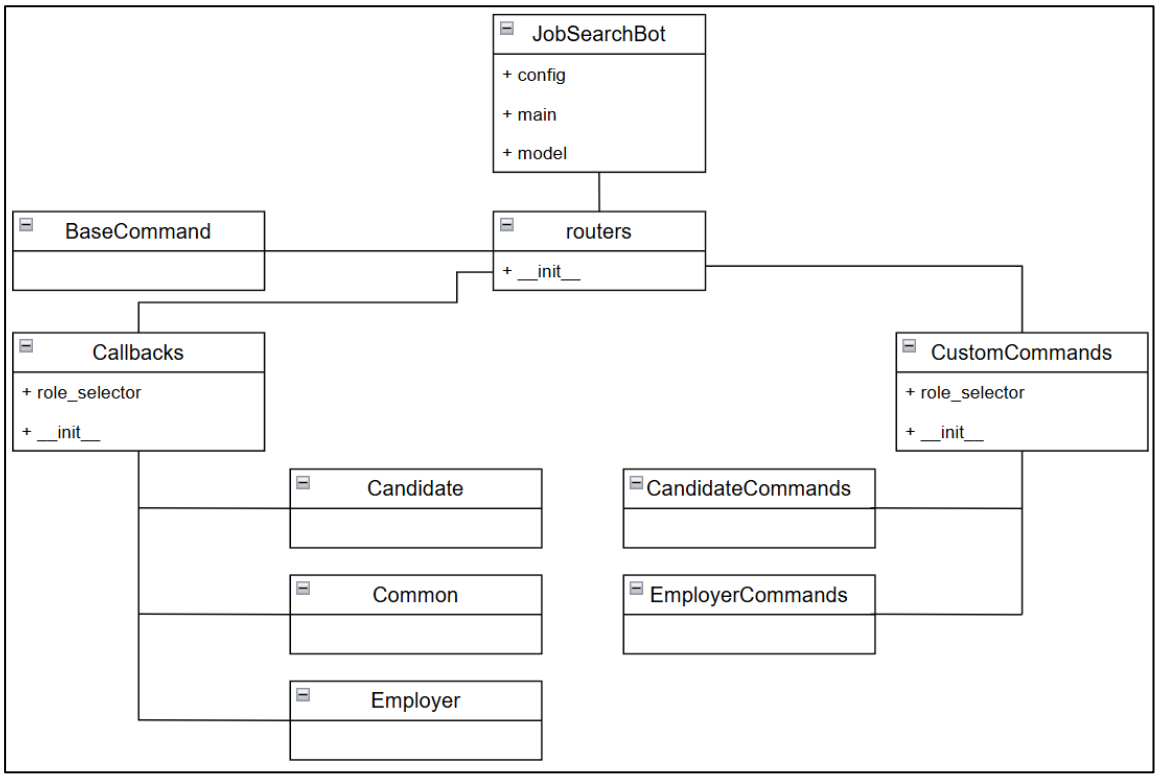

Рисунок 3.1 – Структура серверної частини Telegram-бота

Диспетчер повідомлень відіграє роль центрального елемента,

що забезпечує обробку вхідних подій у Telegram-боті. У контексті реалізації серверної частини, саме диспетчер (Dispatcher) реєструє основний маршрутизатор, який у свою чергу агрегує вкладені маршрутизатори, що обробляють різні типи команд та callback-запитів від користувачів.

Схема побудови маршрутизаторів є ієрархічною. На верхньому рівні знаходиться головний маршрутизатор, оголошений у модулі routers/__init__.py. Цей маршрутизатор включає в себе три підмаршрутизатори:

– base_command_router — обробляє загальні текстові команди;

– callbacks_router — відповідає за callback-запити, що виникають під час взаємодії з інлайн-кнопками;

– custom_command_router — містить функціонал, специфічний для ролей користувачів, які взаємодіють із ботом.

Маршрутизатор callbacks_router є контейнером для обробників трьох категорій: Common, Employer, Candidate. Вони розміщені у відповідних модулях і відповідають за обробку подій callback типу, що стосуються відповідно загальної логіки, логіки для роботодавців та кандидатів. Усі ці обробники зареєстровані у вигляді окремих маршрутизаторів, які включаються у відповідному порядку.

У свою чергу custom_command_router включає три компоненти:

– role_selector_router — реалізує логіку початкового вибору ролі користувача;

– candidate_router — містить набір команд для користувачів з роллю кандидата;

– employer_router — містить набір команд для користувачів з роллю роботодавця.

Пакет CandidateCommands, як частина custom_command_router, містить маршрутизатори, що відповідають за перегляд доступних вакансій (view_vacancies_router), управління резюме (manage_resume_router) та перегляд історії взаємодії з ботом

(candidate_router у history.py).

Кожен маршрутизатор оголошується за допомогою об'єкта Router з модуля aiogram. Його обробники регіструються відповідно до логіки взаємодії користувача з ботом, а сам маршрутизатор передається у структуру включення до маршрутизатора вищого рівня через метод include_routers(...).

У результаті, всі маршрутизатори в кінцевому вигляді реєструються у диспетчері через головний маршрутизатор, що забезпечує централізовану обробку повідомлень, callback-запитів та інших подій Telegram API.

Для забезпечення взаємодії з базою даних у серверній частині Telegram-бота реалізовано набір ORM-моделей з використанням бібліотеки SQLAlchemy [10]. Об’єкти моделей відповідають таблицям у реляційній базі даних і слугують для збереження та обробки структурованої інформації, пов’язаної з користувачами, резюме, вакансіями та відповідями.

Кожна модель створюється як клас, що наслідується від базового класу Base [29], який генерується за допомогою методу declarative_base() або DeclarativeBase у сучасному варіанті. Атрибути класу визначають поля таблиці, типи даних, обмеження та зв’язки між таблицями через механізми mapped_column, ForeignKey, relationship.

Реалізація моделі User наведена на лістингу 3.1.

Приклад коду 3.1 – Реалізація ORM моделі User

```
     class User(Base):
           tablename   = "users"

          id:    Mapped[int]    =    mapped_column(Integer,
primary_key=True, autoincrement=True)
          telegram_id:             Mapped[int]             =
mapped_column(BigInteger, unique=True, nullable=False)
          username: Mapped[str] = mapped_column(String(64))
          first_name:              Mapped[str]             =
mapped_column(String(64))
          last_name: Mapped[str] = mapped_column(String(64))
          created_at: Mapped[str] = mapped_column(TIMESTAMP,
server default=func.current timestamp())

          resumes          =          relationship("Resume",
back_populates="user", cascade="all, delete")
```

```
        vacancies = relationship("Vacancy",
back_populates="user", cascade="all, delete")
```

Модель User містить інформацію про користувачів Telegram. Поле telegram_id використовується для ідентифікації користувача у Telegram. Зв’язок один-до-багатьох реалізовано через relationship для пов’язаних резюме та вакансій, що належать даному користувачеві.

Реалізація моделі Resume наведена на лістингу 3.2.

Приклад коду 3.2 – Реалізація ORM моделі Resume

```
    class Resume(Base):
        __tablename__= "resumes"

        id: Mapped[int] = mapped_column(Integer,
primary_key=True, autoincrement=True)
        user_id: Mapped[int] =
mapped_column(ForeignKey("users.id", ondelete="CASCADE"),
nullable=False)
        position: Mapped[str] = mapped_column(String(128))
        experience_years: Mapped[int] =
mapped_column(Integer)
        skills: Mapped[str] = mapped_column(Text)
        education: Mapped[str] = mapped_column(Text)
        city: Mapped[str] = mapped_column(String(64))
        employment_type: Mapped[str] =
mapped_column(String(64))
        expected_salary: Mapped[float] =
mapped_column(DECIMAL(10, 2))
        created_at: Mapped[str] = mapped_column(TIMESTAMP,
server_default=func.current_timestamp())

        category_id: Mapped[int | None] =
mapped_column(ForeignKey("categories.id", ondelete="SET
NULL"), nullable=True)

        user = relationship("User",
back_populates="resumes")
        category = relationship("Category",
back_populates="resumes")
        responses = relationship("Response",
back_populates="resume", cascade="all, delete")
```

Модель Resume зберігає інформацію про резюме користувача: позиція, стаж, навички, освіта, тип зайнятості та бажана заробітна плата. Зв’язки реалізовано з таблицями users, categories та responses.

Аналогічним чином реалізовані моделі:

– Category — категорії вакансій;

– Vacancy — вакансії, створені роботодавцями;

– ResponseStatus — статуси відповідей кандидатів;

– Response — відповіді на вакансії.

Кожна модель забезпечує двосторонні зв’язки через атрибути relationship, що дозволяє формувати запити з урахуванням пов’язаних об’єктів.

Для запуску Telegram-бота необхідно виконати декілька процедур ініціалізації, які забезпечують коректну роботу системи обробки повідомлень, взаємодію з базою даних, зберігання стану та реєстрацію маршрутів обробки.

Як було зазначено раніше, усі маршрутизатори (роутери), які описують обробку команд, callback-подій або повідомлень користувача, повинні бути зареєстровані у головному об'єкті Dispatcher. Об'єкт Dispatcher фактично наслідує та розширює базовий клас роутера, виступаючи центральним елементом маршрутизації у додатку. Для зберігання стану finite-state machine використовується об’єкт MemoryStorage [35], який забезпечує зберігання FSM-сесій у оперативній пам’яті.

Для майбутнього тестування, відлагодження та моніторингу роботи застосунку ініціалізується базовий рівень логування з рівнем INFO за допомогою стандартного модуля logging [22].

Перед запуском бота необхідно створити всі таблиці у базі даних. Для цього реалізовано асинхронну функцію init_models (лістинг 3.3), яка відкриває підключення до бази даних через об’єкт engine та виконує команду створення всіх описаних ORM-моделей.

Приклад коду 3.3 – Реалізація функції init_models

```
async def init_models():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)
```

Функція відкриває асинхронний контекст з’єднання, через який викликає метод run_sync, що синхронно виконує SQL-команди створення таблиць на основі визначених моделей, зв’язків і метаданих Base.metadata.

Загальний запуск Telegram-бота реалізовано у модулі main.py. Його вміст подано у лістингу 3.4.

Приклад коду 3.4 – Вміст файлу main.py

```
import asyncio
import logging
from aiogram import Bot, Dispatcher
from aiogram.enums import ParseMode
from aiogram.client.default import
DefaultBotProperties
from aiogram.fsm.storage.memory import MemoryStorage
from config import BOT_TOKEN
from routers import router
from models import init_models

storage = MemoryStorage()
dp = Dispatcher(storage=storage)
dp.include_router(router)

async def main():
    logging.basicConfig(level=logging.INFO)
    await init_models()
    bot = Bot(
        token=BOT_TOKEN,

default=DefaultBotProperties(parse_mode=ParseMode.HTML)
    )
    await dp.start_polling(bot)

if __name__== "__main__":
    asyncio.run(main())
```

Основна функція main() виконує кілька послідовних дій:

1. Ініціалізація логування.
2. Ініціалізація бази даних через init_models.
3. Створення об’єкта Bot з токеном та параметрами форматування повідомлень.
4. Запуск циклу обробки подій через метод start_polling.

Таким чином, при запуску main.py бот підключається до Telegram API, створює необхідні таблиці, реєструє всі обробники та очікує вхідних повідомлень для подальшої обробки.

Бібліотека aiogram підтримує систему так званих магічних фільтрів [24], які дозволяють визначати обробники повідомлень за

допомогою спеціальних умов. Одним із базових фільтрів є CommandStart(), який спрацьовує на повідомлення користувача з командою /start. Кожен Telegram-бот має реалізовувати обробку цієї команди, тому після реалізації відповідного обробника серверну частину бота можна вважати розгорнутою та здатною до взаємодії з клієнтом. У реалізованому проєкті обробка команди /start реалізована у функції cmd_start, наведеної на листингу 3.6.

Приклад коду 3.6 – Обробка стартової команди та ініціалізація користувача

```
    @router.message(CommandStart())
    async def cmd_start(message: types.Message, state:
FSMContext):
        user_id = message.from_user.id
        username = message.from_user.username or ""
        first_name = message.from_user.first_name or ""
        last_name = message.from_user.last_name or ""
        async with async_session() as session:
            stmt = select(User).where(User.telegram_id ==
user_id)
            result = await session.execute(stmt)
            user = result.scalar_one_or_none()

            if not user:
                new_user = User(
                    telegram_id=user_id,
                    username=username,
                    first_name=first_name,
                    last_name=last_name
                )
                session.add(new_user)
                try:
                    await session.commit()
                except IntegrityError:
                    await session.rollback()
        full_display_name = f"{first_name}
{last_name}".strip()

        keyboard = types.ReplyKeyboardMarkup(
            resize_keyboard=True,
            keyboard=[
                [types.KeyboardButton(text="Я
роботодавець 🏢")],
                [types.KeyboardButton(text="Я кандидат
💼")]
            ])
        await message.answer_photo(
            photo="https://cdn-icons-
png.flaticon.com/512/3062/3062634.png",
            caption=(
```

```
                    f"👋 Привіт, {hbold(full_display_name)}!\n\n"
                    "Цей бот допоможе знайти або розмістити вакансії.\n"
                    "Вибери, хто ти:"),
                reply_markup=keyboard)
            await state.set_state(GlobalState.waiting_for_role)
```

У першому рядку функції використано декоратор @router.message(CommandStart()), що вказує на обробку повідомлення, яке задовольняє фільтр CommandStart [18]. Цей фільтр активується для вхідної команди /start, яка ініціює сценарій взаємодії користувача з ботом.

Об'єкт message містить дані про повідомлення Telegram, а state є об'єктом контексту керування станами (FSMContext), необхідного для реалізації finite-state машини.

Далі з повідомлення зчитуються дані користувача: Telegram ID, username, ім’я та прізвище. Усі поля нормалізуються для подальшого використання.

Після цього виконується підключення до бази даних за допомогою асинхронного менеджера контексту async with async_session() as session. Це дозволяє створити сесію взаємодії з базою даних у межах однієї транзакції. Використовується запит select() для перевірки наявності користувача в таблиці users за Telegram ID. Якщо запис не існує, створюється новий екземпляр моделі User, який містить відповідні атрибути. Новий користувач додається до сесії через session.add(new_user), після чого виконується спроба збереження транзакції командою await session.commit(). У разі виникнення винятку типу IntegrityError виконується відкат транзакції через await session.rollback().

Після цього формується ім’я користувача для виводу у вітальному повідомленні. Користувачеві надсилається повідомлення з фото та клавіатурою, яка містить два варіанти вибору ролі: роботодавець або кандидат. Для цього використовується

ReplyKeyboardMarkup [26].

На завершення функції викликається метод set_state з передачею нового стану GlobalState.waiting_for_role, який вказує на те, що бот очікує від користувача вибору ролі. Це дозволяє реалізувати подальшу логіку взаємодії залежно від вибраного варіанта.

3.2. Реалізація модуля генерації резюме на основі введених даних

Перед формуванням фінального представлення резюме у вигляді повідомлення, що буде відправлено користувачеві, необхідно отримати повний набір даних. З огляду на це, процес створення резюме поділяється на два етапи: заповнення користувачем відповідних полів та подальша генерація узагальненої інформації на основі введених значень. Таким чином, реалізація модуля генерації резюме передбачає попереднє послідовне введення даних, які зберігаються у контексті сесії, а вже після завершення цього процесу — побудову текстового подання, що структуровано репрезентує зміст резюме.

Для реалізації поетапного введення даних від користувача доцільно використовувати підхід на основі скінченного автомату станів (Finite State Machine, FSM) [23]. Такий підхід дозволяє організувати послідовну взаємодію з користувачем, де кожен етап діалогу відповідає окремому стану. Це забезпечує контрольований перехід між етапами, запобігає втраті даних та дозволяє валідувати введену інформацію на кожному кроці.

FSM у бібліотеці aiogram реалізовано за допомогою механізмів StatesGroup та State, які визначають множину станів, у яких може перебувати користувач протягом сесії взаємодії з ботом. Кожен стан відповідає очікуванню певного типу даних від користувача. Перехід між станами виконується програмно після отримання та обробки відповідного повідомлення користувача.

На лістингу 3.7 наведено визначення класу ResumeCreationState, який реалізує скінченний автомат для створення резюме. Кожен атрибут цього класу є екземпляром класу State і представляє один із кроків заповнення структури резюме.

Приклад коду 3.7 — Клас ResumeCreationState для поетапного створення резюме

```
class ResumeCreationState(StatesGroup):
    category = State()
    position = State()
    experience_years = State()
    skills = State()
    education = State()
    city = State()
    employment_type = State()
    expected_salary = State()
```

Стан category відповідає введенню або вибору категорії вакансії. Наступний стан — position — передбачає введення бажаної посади. Стан experience_years відповідає введенню кількості років досвіду. У стані skills користувач зазначає володіння конкретними навичками, у education — надає інформацію про освіту. У стані city вводиться місто проживання або бажаної зайнятості. Далі, у стані employment_type, зазначається бажаний тип зайнятості. Останній стан — expected_salary — відповідає за введення очікуваного рівня заробітної плати.

Ініціація процесу заповнення резюме відбувається у відповідь на повідомлення користувача з текстом «📝 Створити резюме». Дана дія можлива лише за умови, що у поточного користувача ще не збережено жодного резюме. У протилежному випадку користувач може лише редагувати вже наявне резюме, при цьому сам процес редагування є ідентичним до первинного створення. За обробку повідомлення «📝 Створити резюме» відповідає функція start_add_resume, наведена в листингу 3.8.

Приклад коду 3.8 — Ініціація процесу створення резюме

```
    @router.message(GlobalState.user_as_candidate, lambda
msg: msg.text == "📝 Створити резюме")
    async def start_add_resume(message: types.Message,
state: FSMContext):
```

```
        await message.answer("Введіть бажану посаду:",
reply_markup=types.ReplyKeyboardRemove())
        await
state.set_state(ResumeCreationState.position)
```

Обробник start_add_resume активується лише у випадку, якщо користувач перебуває у глобальному стані GlobalState.user_as_candidate, що вказує на попередній вибір ролі "кандидат" у структурі керування станами. Після отримання відповідного повідомлення бот надсилає запит на введення бажаної посади у вигляді тексту, одночасно деактивуючи клавіатуру за допомогою ReplyKeyboardRemove. Далі виконується перехід у новий стан ResumeCreationState.position, який відповідає за обробку введеної посади. Наступний обробник process_position, представлений у листингу 3.9, відповідає за прийом та обробку цього значення.

Приклад коду — Обробка введеної посади та запит категорії

```
    @router.message(ResumeCreationState.position)
    async def process_position(message: types.Message,
state: FSMContext):
        await state.update_data(position=message.text)

        async with async_session() as session:
            result                =                await
session.execute(select(Category))
            categories = result.scalars().all()

        keyboard = []
        row = []
        for i, category in enumerate(categories):

row.append(InlineKeyboardButton(text=category.name,
callback_data=str(category.id)))
            if len(row) == 3:
                keyboard.append(row)
                row = []
        if row:
            keyboard.append(row)
```

```
        await message.answer("Оберіть категорію:",
reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard
))
        await
state.set_state(ResumeCreationState.category)
```

Даний обробник працює в контексті стану ResumeCreationState.position, який встановлюється автоматом керування станами ResumeCreationState. Після отримання текстового повідомлення від користувача введене значення зберігається у внутрішньому сховищі через виклик state.update_data(position=message.text). Далі виконується асинхронний запит до бази даних з використанням async_session для вибірки всіх записів з таблиці Category. Для цього використовується вираз select(Category) з подальшим отриманням результатів через scalars().all(). Отримані об’єкти категорій використовуються для динамічного формування інлайн-клавіатури. Кнопки формуються з назв категорій, а їх callback_data відповідає ідентифікатору категорії в базі. Кнопки групуються по три в рядок, після чого формується інтерфейс у вигляді клавіатури InlineKeyboardMarkup [20].

Після надсилання повідомлення з клавіатурою користувачу виконується перехід до наступного стану ResumeCreationState.category, який відповідає за обробку вибору категорії. Це завершення поточного кроку заповнення та перехід до наступного етапу.

Повна реалізація послідовного процесу заповнення резюме на основі введених користувачем даних наведена в додатку А.1.

Завершальним етапом процесу заповнення резюме є обробник повідомлення у стані ResumeCreationState.expected_salary. У цьому методі обробляється введене користувачем значення очікуваної заробітної плати, перевіряється його відповідність числовому формату, після чого зчитуються всі проміжні дані, зібрані протягом попередніх етапів. Для цього використовується виклик await state.get_data(), який повертає словник значень, що були записані в

кожному з попередніх станів автомата ResumeCreationState.

Далі здійснюється асинхронний запит до бази даних для пошуку користувача за Telegram-ідентифікатором. У разі успішного пошуку формується об'єкт типу Resume, якому присвоюються відповідні значення з внутрішнього сховища автомата. Зокрема, це позиція, категорія, кількість років досвіду, навички, освіта, місто, тип зайнятості та очікувана заробітна плата. Об’єкт зберігається в базі даних через session.add(resume) з подальшим виконанням session.commit() для фіксації транзакції.

Після створення резюме користувачу відправляється повідомлення із клавіатурою для переходу до подальшої взаємодії з ботом. Завершується метод очищенням поточного стану (await state.clear()) та встановленням глобального стану GlobalState.user_as_candidate, який дозволяє здійснювати доступ до функціоналу користувача-кандидата.

Фрагмент реалізації процесу створення об'єкта резюме на основі даних зі сховища автомата наведено в лістингу 3.10.

Приклад коду 3.10 — Формування об'єкта резюме та збереження до бази даних

```
    data = await state.get_data()
    user_id = message.from_user.id

    async with async_session() as session:
        result = await session.execute(select(User).where(User.telegram_id == user_id))
        user = result.scalar_one_or_none()
        if user is None:
            await message.answer("Користувача не знайдено.")
            await state.clear()
```

Продовження прикладу коду 3.10

```
            return

        resume = Resume(
            user_id=user.id,
            position=data["position"],
            category_id=data["category_id"],
            experience_years=data["experience_years"],
            skills=data["skills"],
            education=data["education"],
```

```
            city=data["city"],
            employment_type=data["employment_type"],
            expected_salary=salary
        )

        session.add(resume)
        await session.commit()
```

В наведеному фрагменті використовується метод get_data() з об'єкта FSMContext, який повертає словник із усіма зібраними даними, що були поетапно збережені в межах автомата ResumeCreationState. Далі з об'єкта повідомлення message отримується унікальний ідентифікатор користувача, який ініціював процес створення резюме.

У межах асинхронного контекстного менеджера async with async_session() as session виконується запит до таблиці User для отримання об'єкта користувача, який відповідає зазначеному telegram_id. У разі відсутності відповідного користувача обробка переривається, автомат очищується за допомогою await state.clear(), та відправляється повідомлення про помилку.

У разі успішного отримання користувача формується об'єкт Resume, у який передаються дані зі сховища автомата: назва посади, ідентифікатор обраної категорії, кількість років досвіду, навички, освіта, місто, тип зайнятості, очікувана заробітна плата. Також встановлюється зовнішній ключ user_id на відповідного користувача.

Створений об'єкт додається до поточної сесії SQLAlchemy методом session.add(resume), після чого зміни фіксуються в базі даних викликом await session.commit().

Перехід до реалізації модуля відображення резюме виконується після завершення формування відповідного об'єкта в базі даних. Для відображення резюме користувача використовується форматоване текстове повідомлення, яке містить основні атрибути, що були введені під час заповнення. Запуск механізму перегляду здійснюється у відповідь на повідомлення користувача з текстом «✎ Моє резюме». Відповідальність за обробку даного повідомлення покладено на метод handle_manage_resume, реалізація якого наведена у лістингах 3.11 та

3.12.

Приклад коду 3.11 – Обробка запиту на перегляд резюме

```
    @router.message(GlobalState.user_as_candidate, lambda
msg: msg.text == "✏ Моє резюме")
    async        def        handle_manage_resume(message:
types.Message):
        async with async_session() as session:
            result = await session.execute(
                select(User).where(User.telegram_id    ==
message.from_user.id)
            )
            user = result.scalar_one_or_none()

            if not user:
                await message.answer("Обліковий запис не
знайдено.")
                return

            result = await session.execute(
                select(Resume).where(Resume.user_id    ==
user.id).limit(1)
            )
            resume = result.scalar_one_or_none()

            if resume is None:
                reply_kb = ReplyKeyboardMarkup(
                    keyboard=[[KeyboardButton(text="📝
Створити резюме")]],
                    resize_keyboard=True
                )
                await   message.answer("У   вас   немає
резюме.", reply_markup=reply_kb)
                return

            category   =   await   session.get(Category,
resume.category_id)
```

У першій частині методу (лістинг 3.11) використовується дескриптор @router.message(GlobalState.user_as_candidate, lambda msg: msg.text == "✏ Моє резюме"), що обмежує обробку лише у разі, якщо користувач перебуває у глобальному стані user_as_candidate. Це запобігає помилковій активації функціоналу для користувачів, які ще не завершили формування резюме.

У межах асинхронного контекстного менеджера async with async_session() as session відбувається послідовне виконання SQL-запитів до таблиць User та Resume. За telegram_id користувача

витягується об'єкт User. У разі його відсутності метод повертає повідомлення про помилку і завершує роботу. Якщо користувача знайдено, виконується вибірка одного запису резюме (limit(1)) за полем user_id. У разі відсутності резюме користувачу надсилається повідомлення про необхідність створення резюме, разом із клавіатурою, яка ініціює цей процес.

Після цього за category_id, збереженим у полі резюме, додатково отримується об'єкт Category. Це дозволяє підставити повну назву категорії в текст повідомлення. Завершення першої частини методу передбачає наявність об'єктів user, resume та category, необхідних для подальшого форматування інформації.

Приклад коду – Формування повідомлення з даними резюме та надсилання

```
            resume_text = (
                f"📌 Посада: {resume.position}\n"
                f"🗂 Категорія: {category.name if
category else 'Невідомо'}\n"
                f"☑ Досвід: {resume.experience_years}
років\n"
                f"🛠 Навички: {resume.skills}\n" f"🎓
                Освіта: {resume.education}\n" f"🔑
                Місто: {resume.city}\n"
                f"🕘 Тип зайнятості:
{resume.employment_type}\n"
                f"💰 Очікувана зарплата:
{resume.expected_salary} грн"
            )

            inline_kb = InlineKeyboardBuilder()
            inline_kb.button(text="Редагувати резюме",
callback_data=f"edit_resume_{resume.id}")
            await message.answer(resume_text,
reply_markup=inline_kb.as_markup())
```

У другій частині методу (лістинг 3.12) формується рядок resume_text, до якого вставляються значення полів об'єкта Resume. Результат подається у вигляді форматованого повідомлення, що містить послідовно: посаду, категорію, досвід, навички, освіту, місто, тип зайнятості та очікувану заробітну плату. Якщо категорія не знайдена, підставляється службове значення «Невідомо».

Після побудови тексту створюється об'єкт типу InlineKeyboardBuilder, у якому додається одна кнопка з підписом «Редагувати резюме» та callback_data, що містить ідентифікатор резюме. Це забезпечує можливість переходу до редагування за допомогою окремого модуля з обробниками callback-запитів. Форматоване повідомлення разом із клавіатурою надсилається користувачеві методом await message.answer(...).

3.3. Тестування та демонстрація функціональності чат-бота

Для перевірки функціональної придатності реалізованого Telegram-бота було проведено послідовне тестування основних сценаріїв взаємодії з користувачем. Першим етапом тестування є перевірка реакції бота на натискання кнопки /start. Результат виконання команди наведено на рисунку 3.2. Бот надіслав привітальне повідомлення із зазначенням імені користувача, що підтверджує коректну ініціалізацію діалогу.

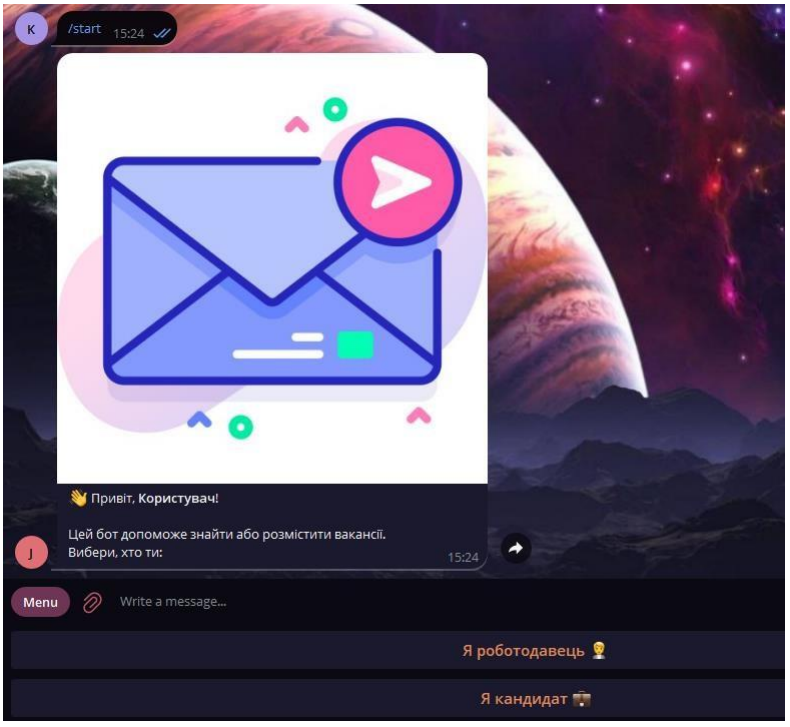

Рисунок 3.2 – Вітальне повідомлення бота після натискання кнопки

«Start»

Далі перевірено функціонал створення резюме. Для цього було вибрано роль кандидата через команду «Я кандидат», після чого надіслано повідомлення «✏ Моє резюме». Бот повідомив про відсутність резюме та запропонував його створити. Процес заповнення резюме здійснювався через послідовну взаємодію із ботом. Кожен етап заповнення супроводжувався відповідними запитами на введення даних. Результат створення резюме зафіксовано на рисунках 3.3 та 3.4.

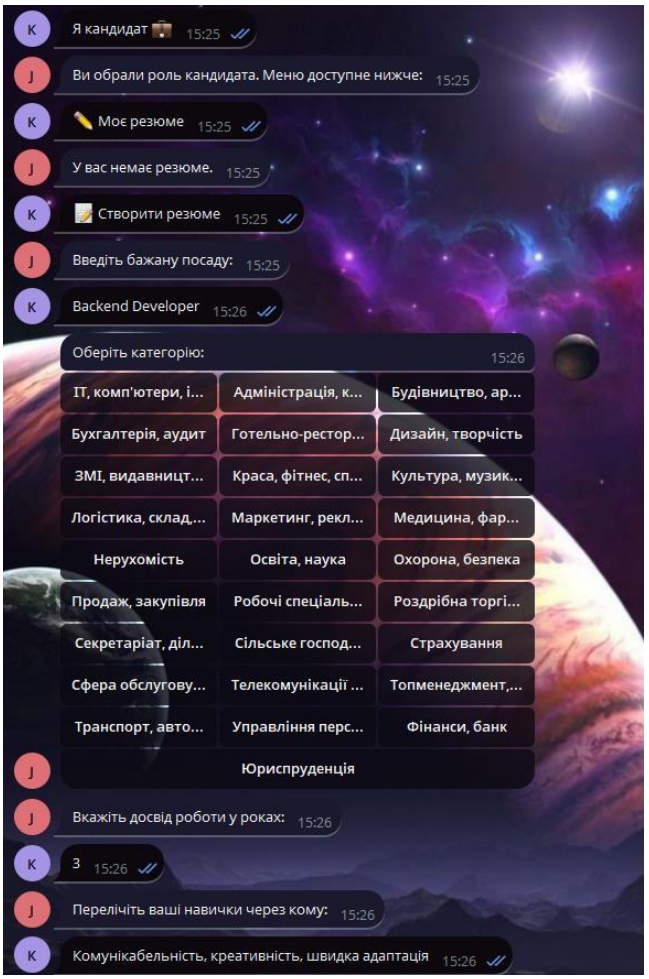

Рисунок 3.3 – Процес створення резюме (1)

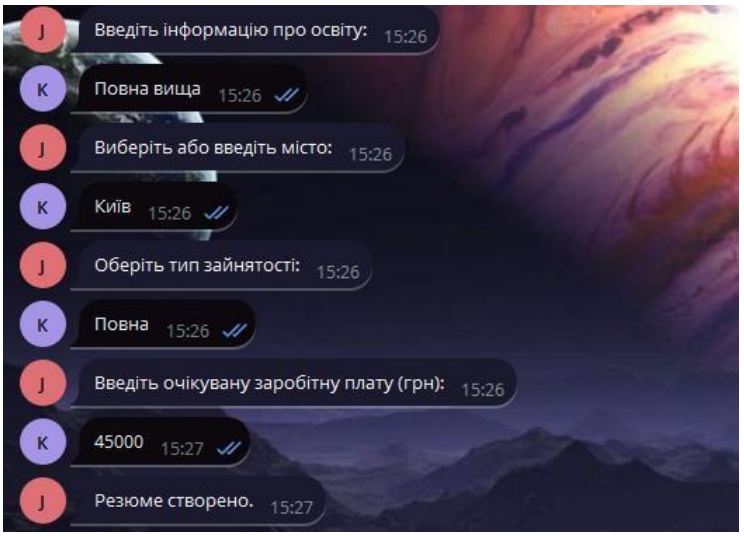

Рисунок 3.4 – Процес створення резюме (2)

Після завершення процесу створення резюме було повторно перевірено відображення введених даних. Знову натиснуто кнопку «✏ Моє резюме», після чого бот сформував форматоване повідомлення з актуальними даними резюме, введеними під час попереднього етапу. Вивід інформації наведено на рисунку 3.5.

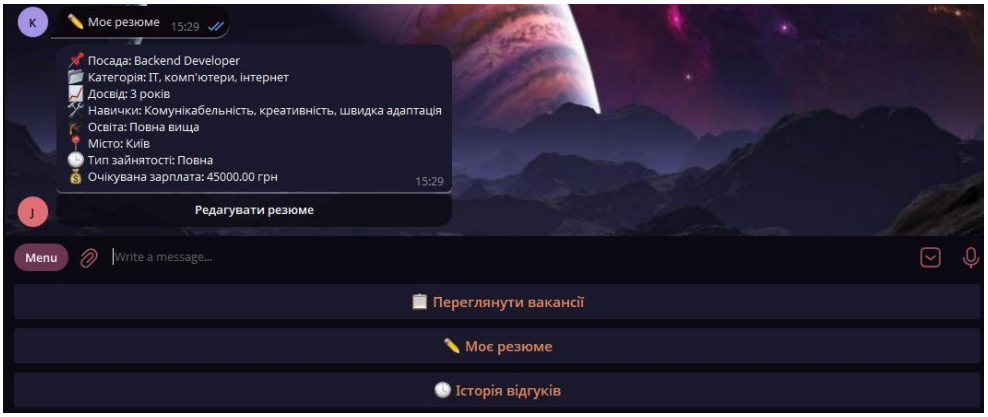

Рисунок 3.5 – Відображення сформованого резюме в боті

Далі було обрано кнопку «📋 Переглянути вакансії». У результаті бот відобразив дві вакансії: одну, що відповідає вказаному в резюме місту, та іншу з віддаленим форматом працевлаштування. Порядок виводу вакансій відповідає закладеній логіці: спочатку — вакансії в заданому місті, потім — віддалені пропозиції. Відповідне повідомлення представлено на рисунку 3.6.

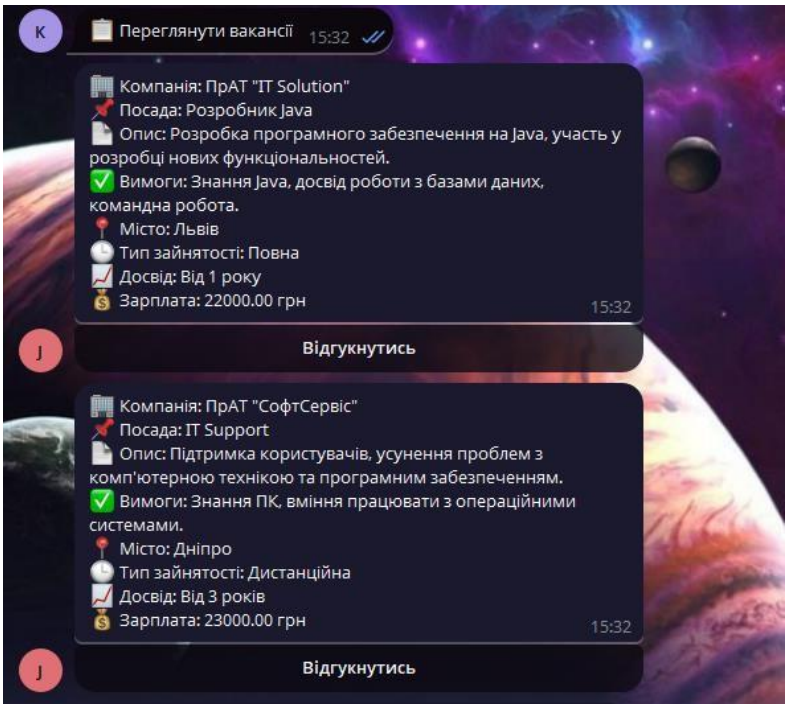

Рисунок 3.6 – Пошук вакансій на основі резюме

Далі перевіряється функціональність подачі відгуків на вакансії. Для цього кандидат обирає одну з вакансій і натискає кнопку «Вігукнутись». Після цього кандидат отримує сповіщення «Відгук враховано», а роботодавець у розділі «Нові відгуки» бачить цей відгук з інформацією, наданою кандидатом у його резюме , як це наведено на рисунку 3.7.

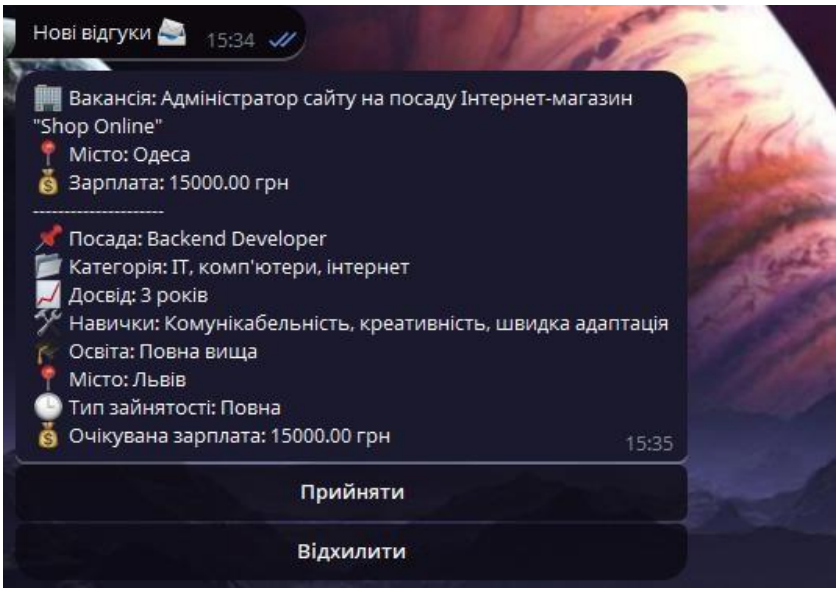

Рисунок 3.7 – Процес подачі відгуку кандидата на вакансію

У випадку, якщо роботодавцю підходить кандидат, він може натискати кнопку «Прийняти», що дозволяє йому отримати контактні дані кандидата (рисунок 3.8).

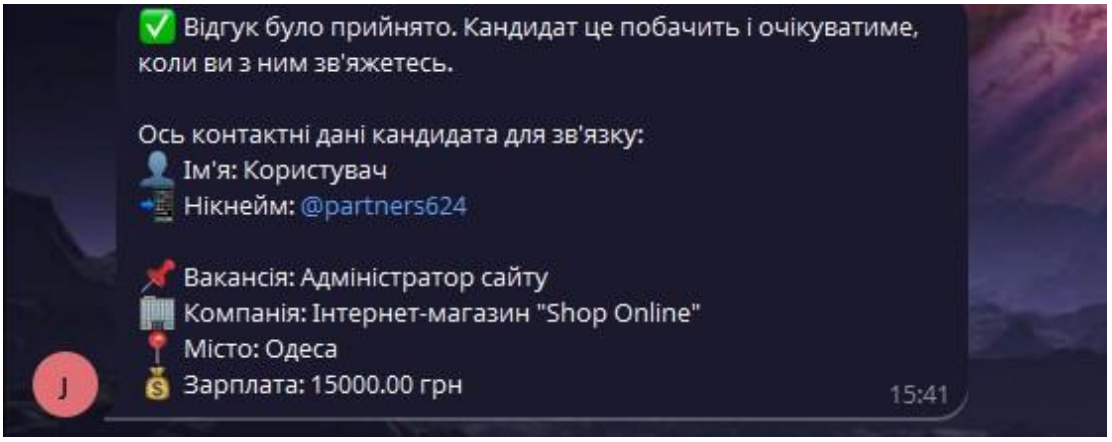

Рисунок 3.8 – Підтвердження прийняття кандидата роботодавцем

Кандидат, у свою чергу, отримає сповіщення про те, що його кандидатуру було схвалено, та буде просити очікувати на зв'язок з роботодавцем (рисунок 3.9).

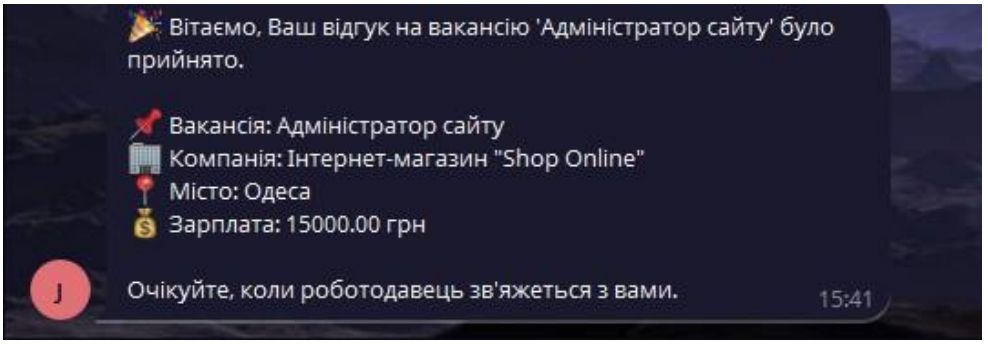

Рисунок 3.9 – Сповіщення для кандидата про прийняття його кандидатури

ВИСНОВКИ

У результаті виконання бакалаврської роботи була досягнута основна мета — розроблено програмне забезпечення у вигляді Telegram-бота, яке автоматизує взаємодію між кандидатами та роботодавцями, забезпечуючи функціонал створення резюме, пошуку вакансій, а також обміну відгуками. Це рішення дозволяє значно спростити процес працевлаштування, автоматизувавши основні етапи взаємодії через інтерфейс месенджера, що забезпечує доступність і зручність для користувачів.

У рамках дослідження було виконано детальний аналіз наявних підходів до автоматизації процесу пошуку роботи за допомогою чат-ботів. Було визначено функціональні та нефункціональні вимоги до системи, що лягли в основу розробленої архітектури програмного забезпечення. Зокрема, для реалізації основної логіки було обрано використання бібліотеки aiogram для створення чат-бота, ORM-фреймворку SQLAlchemy для роботи з базою даних, а також технології finite state machine (FSM) для керування станами чат-бота.

Розробка була орієнтована на забезпечення зручної взаємодії користувачів із системою. Це включає створення резюме з параметрів, введених користувачем, збереження даних у базі даних, а також автоматичне відображення сформованого резюме. Для кожного кандидата реалізовано можливість подавати відгуки на вакансії, що дозволяє роботодавцю приймати рішення щодо кандидата на основі наданої інформації.

Під час тестування було підтверджено працездатність основних функцій бота, зокрема створення резюме, його відображення, пошук вакансій відповідно до зазначених даних, а також процес подачі відгуків на вакансії. Бот правильно обробляє введені дані, формує резюме, відображає вакансії з урахуванням географічного розташування користувача і дозволяє здійснювати відгуки на вакансії з подальшою обробкою роботодавцем.

У процесі розробки були виявлені певні проблеми та обмеження, які можуть виникнути при подальшій експлуатації системи. Однією з таких проблем є обмеженість функціоналу форматування повідомлень у Telegram, що може ускладнити реалізацію більш складних форматів для відображення даних.

З огляду на отримані результати, можна рекомендувати подальший розвиток системи у кількох напрямках. По-перше, доцільно реалізувати механізм автоматичного рекомендованого пошуку вакансій на основі профілю кандидата та його історії взаємодій з ботом. По-друге, можна розширити функціональність бота шляхом додавання можливості для користувачів редагувати свої резюме та зберігати кілька варіантів.

Для подальшого розвитку бота можна також розглянути інтеграцію з іншими платформами для підбору персоналу або онлайн-ресурсами, що дозволить розширити функціональність чат-бота і зробити його більш універсальним інструментом для пошуку роботи. Інтеграція з HR-системами або платформами для автоматизації подачі вакансій дозволить ще більше спростити процес взаємодії між роботодавцями та кандидатами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андрієвська В., Слісаренко М. Особливості роботи чат-ботів на платформі “Telegram” // Наумовські читання : матеріали XXI Всеукр. наук.-метод. конф. здобувачів вищ. освіти та молод. вчених, присвяч. 100-річчю до дня народж. І. О. Наумова, м. Харків, 23–24 листоп. 2023 р. / Харків. нац. пед. ун-т ім. Г. С. Сковороди ; за заг. ред. О. А. Жерновникової. – Харків, 2024. – С. 325–326.

2. Браун Ф. Модель діаграми зв’язків сутностей (ER). Guru99. – URL: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html (дата звернення: 18.04.2025).

3. Зосім М. Моделювання даних / М. Зосім. Maxym Zosym. URL: https://www.maxzosim.com/data-modelling/ (дата звернення: 22.04.2025).

4. Кінцевий автомат (FSM). aiogram. – URL: https://docs.aiogram.dev/uk-ua/latest/dispatcher/finite_state_machine/index.html (дата звернення: 22.04.2025).

5. Міщенко А. А. Інструменти розробника реляційної бази даних / А. А. Міщенко. URL: https://ela.kpi.ua/server/api/core/bitstreams/51e65e0f-e6ab-4012-9add-4bdebe552470/content (дата звернення: 20.04.2025).

6. Павленко Ю. С. Пошукова оптимізація, технології та сервіси веб-аналітики: конспект лекцій / Ю. С. Павленко. URL: https://evnuir.vnu.edu.ua/bitstream/123456789/21864/1/SEO.pdf (дата звернення: 22.04.2025).

7. Рейтблат О., Жуковська В. Особливості застосування чат-ботів в HR // II Міжнародна науково-практична конференція «Глобалізація та розвиток інноваційних систем: тенденції, виклики, перспективи», м. Харків, 14–15 берез. 2024 р. – С. 359-361. – URL: https://repo.btu.kharkov.ua/bitstream/123456789/51090/1/conf-14-15-03-24-mater-360-362.pdf (дата звернення: 20.04.2025).

8. Скальський Ю., Дендюк М. Розроблення інтелектуального чат-бота "Вступник" // Комп'ютерне моделювання та інформаційні технології. – 2023. – С. 86–91. – URL: https://conf.nltu.edu.ua/index.php/conf1/article/view/28/23 (дата звернення: 18.04.2025).

9. Соколов В. П. Автоматизована система обслуговування клієнтів з використанням чат-бота для банківських систем. – URL: https://elibrary.kdpu.edu.ua/bitstream/123456789/10248/1/КРБ_Крестьянов_А_Д_.pdf (дата звернення: 19.04.2025).

10. Сучасне та ефективне управління базами даних у Python. JS Communities. URL: https://javascript.org.ua/sqlalchemy-orm-suchasne-ta-efektivne-upravlinnya-bazami-danih-u-python/ (дата звернення: 22.04.2025).

11. Ушакова І. О. Підходи до створення інтелектуальних чат-ботів / І. О. Ушакова // Системи обробки інформації. – 2019. – Вип. 2. – С. 76–83. – URL: http://nbuv.gov.ua/UJRN/soi_2019_2_12 (дата звернення: 17.04.2025).

12. Фабрика міток зворотнього виклику та фільтрування. aiogram. URL: https://docs.aiogram.dev/uk-ua/latest/dispatcher/filters/callback_data.html (дата звернення: 22.04.2025).

13. Чупріна М. О., Жалдак Г. П. Особливості HR-менеджменту в умовах діджиталізації бізнесу / М. О. Чупріна, Г. П. Жалдак // Сучасні підходи до управління підприємством. – 2020. – Вип. 5. – С. 107–119.

14. About. Joblist. – URL: https://www.joblist.com/about (дата звернення: 15.04.2025).

15. About. Xpert Career. – URL: https://xpert-career.com/about/ (дата звернення: 13.04.2025).

16. Activity diagrams. GeeksforGeeks. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/ (дата звернення: 22.04.2025).

17. Asyncmy. PyPI. – URL: https://pypi.org/project/asyncmy/0.1.6/ (дата звернення: 15.04.2025).

18. Filtering events. aiogram. URL: https://docs.aiogram.dev/en/latest/dispatcher/filters/index.html (дата звернення: 22.04.2025).

19. Finite state machine. aiogram. – URL: https://docs.aiogram.dev/en/latest/dispatcher/finite_state_machine/index.html (дата звернення: 14.04.2025).

20. InlineKeyboardMarkup. aiogram. URL: https://docs.aiogram.dev/uk-ua/latest/api/types/inline_keyboard_markup.html (дата звернення: 22.04.2025).

21. Lee S. Y. UML sequence diagram for reusability of data components / S. Y. Lee // Journal of System and Management Sciences. – 2019. – Vol. 9, No. 3. – P. 65–77. URL: https://www.aasmr.org/jsms/Vol9/2019no.3.4.pdf (дата звернення: 17.04.2025).

22. Logging facility for Python. Python documentation. URL: https://docs.python.org/3/library/logging.html (дата звернення: 22.04.2025).

23. Lteif G. An introduction to fsms and their role in computer science / G. Lteif. SoftwareDominos. URL: https://softwaredominos.com/home/software-engineering-and-computer-science/finite-state-machines-an-introduction-to-fsms-and-their-role-in-computer-science/ (дата звернення: 22.04.2025).

24. Magic filters. aiogram. URL: https://docs.aiogram.dev/en/v3.15.0/dispatcher/filters/magic_filters.html (дата звернення: 22.04.2025).

25. Post documentation. aiogram. – URL: https://docs.aiogram.dev/en/v3.20.0.post0/ (дата звернення: 20.04.2025).

26. ReplyKeyboardMarkup. aiogram. URL: https://docs.aiogram.dev/uk-ua/latest/api/types/reply_keyboard_markup.html (дата звернення: 22.04.2025).

27. Reverse engineering a live database. Dev MySQL. URL: https://dev.mysql.com/doc/workbench/en/wb-reverse-engineer-live.html (дата звернення: 14.04.2025).

28. Router. aiogram. URL: https://docs.aiogram.dev/en/latest/dispatcher/router.html (дата звернення: 22.04.2025).

29. Shil P. SQLAlchemy Basics / P. Shil. Medium. URL: https://medium.com/@prosenjeetshil/sqlalchemy-basics-2-2-the-declarative-mapping-way-c9729c102f91 (дата звернення: 22.04.2025).

30. SQLAlchemy. DataScientest. – URL: https://datascientest.com/en/sqlalchemy-what-is-it-whats-it-for (дата звернення: 17.04.2025).

31. State machine diagrams. GeeksforGeeks. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/ (дата звернення: 22.04.2025).

32. Szőke J., Lakosy D. Chatbot as a corporate communication tool: best practice of a Hungarian HR services company // Journal of Ecohumanism. – 2024. – Vol. 3, No. 6. – P. 849–858. – URL: https://www.ceeol.com/search/article-detail?id=1274819 (дата звернення: 13.04.2025).

33. Telegram APIs. Telegram. – URL: https://core.telegram.org/ (дата звернення: 16.04.2025).

34. Wang Y., Kim S., Rafferty A., Sanders K. Employee perceptions of HR practices: A critical review and future directions // The International Journal of Human Resource Management. – 2020. – Vol. 31, No. 1. – P. 128–173. – URL: https://www.tandfonline.com/doi/epdf/10.1080/09585192.2019.1674360?needAccess=true (дата звернення: 21.04.2025).

35. What is the key aspect of a finite state machine (FSM) in terms of its memory. EITCA Academy. URL: https://eitca.org/cybersecurity/eitc-is-cctf-computational-complexity-theory-fundamentals/finite-state-machines/introduction-to-finite-state-machines/examination-review-introduction-to-finite-state-machines/what-is-the-key-aspect-of-a-finite-state-machine-fsm-in-terms-of-its-memory/ (дата звернення: 22.04.2025).

ДОДАТКИ

Додаток А

Код програми

Лістинг А.1 – Код циклу заповнення резюме

```
    @router.message(GlobalState.user_as_candidate, lambda
msg: msg.text == "📝 Створити резюме")
    async def start_add_resume(message: types.Message,
state: FSMContext):
        await message.answer("Введіть бажану посаду:",
reply_markup=types.ReplyKeyboardRemove())
        await
state.set_state(ResumeCreationState.position)

    @router.message(ResumeCreationState.position)
    async def process_position(message: types.Message,
state: FSMContext):
        await state.update_data(position=message.text)

        async with async_session() as session:
            result = await
session.execute(select(Category))
            categories = result.scalars().all()

        keyboard = []
        row = []
        for i, category in enumerate(categories):

row.append(InlineKeyboardButton(text=category.name,
callback_data=str(category.id)))
            if len(row) == 3:
                keyboard.append(row)
                row = []
        if row:
            keyboard.append(row)

        await message.answer("Оберіть категорію:",
reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard
```

```
))
        await
state.set_state(ResumeCreationState.category)

    @router.callback_query(ResumeCreationState.category)
    async def process_category_callback(callback_query:
types.CallbackQuery, state: FSMContext):
        category_id = int(callback_query.data)
        await state.update_data(category_id=category_id)
        await callback_query.answer()
        await      callback_query.message.answer("Вкажіть
досвід роботи у роках:")
        await
state.set_state(ResumeCreationState.experience_years)

    @router.message(ResumeCreationState.experience_years)
    async      def      process_experience_years(message:
types.Message, state: FSMContext):
        try:
            years = int(message.text)
        except ValueError:
            await message.answer("Введіть кількість років
у числовому форматі.")
            return
        await state.update_data(experience_years=years)
        await  message.answer("Перелічіть  ваші  навички
через кому:")
        await state.set_state(ResumeCreationState.skills)

    @router.message(ResumeCreationState.skills)
    async   def   process_skills(message:   types.Message,
state: FSMContext):
        await state.update_data(skills=message.text)
        await   message.answer("Введіть   інформацію   про
освіту:")
        await
state.set_state(ResumeCreationState.education)
```

```
    @router.message(ResumeCreationState.education)
    async def process_education(message: types.Message,
state: FSMContext):
        await state.update_data(education=message.text)

        city_keyboard = ReplyKeyboardMarkup(
            resize_keyboard=True,
            one_time_keyboard=True,
            keyboard=[
                [KeyboardButton(text="Київ"),
KeyboardButton(text="Львів"),
KeyboardButton(text="Харків")],
                [KeyboardButton(text="Дніпро"),
KeyboardButton(text="Одеса"),
KeyboardButton(text="Запоріжжя")],
                [KeyboardButton(text="Вінниця"),
KeyboardButton(text="Івано-Франківськ"),
KeyboardButton(text="Чернівці")]
            ]
        )
        await   message.answer("Виберіть   або   введіть
місто:", reply_markup=city_keyboard)
        await state.set_state(ResumeCreationState.city)

    @router.message(ResumeCreationState.city)
    async def process_city(message: types.Message, state:
FSMContext):
        await state.update_data(city=message.text)

        employment_keyboard = ReplyKeyboardMarkup(
            resize_keyboard=True,
            one_time_keyboard=True,
            keyboard=[
                [KeyboardButton(text="Повна"),
KeyboardButton(text="Неповна")],
                [KeyboardButton(text="Підробіток"),
KeyboardButton(text="Дистанційна")]
            ]
```

```
        )
        await message.answer("Оберіть тип зайнятості:",
reply_markup=employment_keyboard)
        await
state.set_state(ResumeCreationState.employment_type)

    @router.message(ResumeCreationState.employment_type)
    async def process_employment_type(message:
types.Message, state: FSMContext):
        await
state.update_data(employment_type=message.text)
        await message.answer("Введіть очікувану заробітну
плату (грн):")
        await
state.set_state(ResumeCreationState.expected_salary)

    @router.message(ResumeCreationState.expected_salary)
    async def process_expected_salary(message:
types.Message, state: FSMContext):
        try:
            salary = float(message.text)
        except ValueError:
            await message.answer("Введіть коректне числове
значення зарплати.")
            return

        data = await state.get_data()
        user_id = message.from_user.id

        async with async_session() as session:
            result = await
session.execute(select(User).where(User.telegram_id ==
user_id))
            user = result.scalar_one_or_none()
            if user is None:
                await message.answer("Користувача не
знайдено.")
                await state.clear()
```

```
                return

            resume = Resume(
                user_id=user.id,
                position=data["position"],
                category_id=data["category_id"],
experience_years=data["experience_years"],
                skills=data["skills"],
                education=data["education"],
                city=data["city"],
                employment_type=data["employment_type"],
                expected_salary=salary
            )

            session.add(resume)
            await session.commit()

        keyboard = types.ReplyKeyboardMarkup(
            resize_keyboard=True,
            keyboard=[
                [types.KeyboardButton(text="📋
Переглянути вакансії")],
                [types.KeyboardButton(text="✏️ Моє
резюме")],
                [types.KeyboardButton(text="🕓 Історія
відгуків")]
            ]
        )

        await message.answer("Резюме створено.",
reply_markup=keyboard)
        await state.clear()
        await
state.set_state(GlobalState.user_as_candidate)
```