Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних та видавничих технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділенням
комп’ютерних та видавничих
технологій

Чубей О.О. /_____/

підпис

«___»_____2022 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту
освітньо-кваліфікаційного рівня «молодший спеціаліст»

зі спеціальності 122 «Комп’ютерні науки»

на тему : «Онлайн помічник користувача сайту

Галицького коледжу імені В’ячеслава Чорновола»

Студент групи КН-41 Демчук А.В. _____
(підпис)

Керівник проекту Кузик В.М. _____
(підпис)

Консультанти:

з техніко-економічного
обґрунтування Меленчук Л.І. _____
(підпис)

нормоконтролер Сиротюк Н.С. _____
(підпис)

Тернопіль – 2022

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних та видавничих технологій

циклова комісія інформатики та комп`ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділенням
комп’ютерних та видавничих технологій

Чубей О.О. /_____/
підпис
«___»_____2022 р.

ЗАВДАННЯ

на дипломне проєктування

на здобуття освітньо-кваліфікаційного рівня «молодший спеціаліст»

студенту

_____
(прізвище, ім’я та по-батькові студента)

1. Тема проєкту

_____
_____
_____
_____

затверджена наказом по коледжу від “____” _____2021 р., № _____

2. Термін здачі студентом завершеного проєкту “_____” _____2022 р

3. Вихідні дані до проєкту _____
_____
_____
_____

4. Перелік питань, які повинні бути розроблені в проєкті: _____

а) основна частина _____
_____
_____
_____

б) техніко-економічного обґрунтування _____
_____
_____
_____
_____

5. Перелік графічного матеріалу_____
_____
_____

6. Консультанти проєкту: _____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН
дипломного проєктування

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми, ознайомлення з вимогами до дипломного проектування.	20.09.21 р.	01.10.21 р.
2.	Огляд типових рішень та написання відповідного розділу ПЗ	02.12.21 р.	20.01.22 р.
3.	Дослідження технологій реалізації та написання відповідного розділу ПЗ	21.01.22 р.	11.02.22 р.
4.	Розробка функціональних вимог до проекту та робота над структурою програмного продукту. Написаннявідповідного розділу ПЗ	11.02.22 р.	06.03.22 р.
5.	Встановлення на налаштування середовища реалізації та написання відповідного розділу ПЗ	06.03.22 р.	12.03.22 р.
6.	Проектування програмного засобу (функціоналу, інтерфейсу, бази даних продукту) та написаннявідповідного розділу ПЗ	12.03.22 р.	15.04.22 р.
7.	Реалізація та налаштування програмного засобу та написання відповідного розділу ПЗ	15.04.22 р.	01.05.22 р.
8.	Доопрацювання модулів	01.05.22 р.	14.05.22 р.
9.	Тестування на налагодження програмного продукту та написання відповідного розділу ПЗ	14.05.22 р.	01.06.22 р.
10.	Опрацювання економічного розділу дипломного проекту та оформлення спеціального розділу	19.05.22 р.	03.06.22 р.
11.	Робота над оформленням пояснювальної записки	03.06.22 р.	14.06.22 р.
12.	Попередній захист дипломного проекту, доопрацювання	15.06.22 р.	
13.	Підготовка до захисту дипломного проекту	15.06.22 р.	23.06.22 р.
14.	Захист дипломного проекту	23.06.22 р.	29.06.22 р.

7. Дата видачі завдання ”____” _____ 2021 р.
Керівник _____ / _____
Завдання прийняв до виконання_____ / _____

Реферат

Дипломний проєкт. Тема: «Онлайн помічник користувача сайту Галицького коледжу імені В’ячеслава Чорновола». 58 сторінок, 29 рисунків, 6 таблиць. Галицький коледж імені В’ячеслава Чорновола.

Об`єкт дослідження – боти, автоматизація, штучний інтелект.

Предметом дослідження є розробка чат-бота, його оформлення, дизайн та функціонал.

Метою дослідження є оцінка актуальності використання чат-ботів в навчальних закладах.

Завданням проєкту є розробка бота, який зможе проконсультувати абітурієнтів. Чат-бот може бути інтегрований у любі сервіси навчального закладу.

Для розробки даного бота було використано мову Python та бібліотеку FastAPI, тому що вже була використана ця бібліотека в інших проєктах. База даних PostgreSQL, з графічним інтерфейсом PgAdmin, в проєкті зв’язок з базою здійснювався за допомогою бібліотеки psycopg2. Для взаємодії з базою даних використовувалась методологія ORM. Авторизація розроблена за схемою JWT токени.

Результат – розроблений чат-бот, який може відповідати на запитання абітурієнтів, здійснювати розсилки та допомагати зв’язуватись з приймальною комісією користувачам.

Abstract

Diploma project. Subject: "Online assistant for Galician college name of Vyacheslav Chornovyl". 58 pages, 29 figures, 6 tables.

Galician college name of Vyacheslav Chornovyl.

Object of research - bots, automation, artificial intelligence.

The subject of the study is the telegram messenger, its design, design and functionality and bots.

The purpose of the study is to assess the relevance of the use of telegrams in educational institutions.

The task of the project is to develop a bot that will be able to advise applicants. Chat bot can be intergraded in all education services.

The Python language and the FastAPI library were used to develop this bot, because this library has already been used in other projects. The PostgreSQL database, with the PgAdmin graphical interface, in the project communicated with them database using the psycopg2 library. For communication with data base was used ORM schema. Authorization developed by JWT tokens.

The result is a developed bot that can answer questions from applicants, make mailings and help users contact the admissions office.

ЗМІСТ

ВСТУП

Оскільки 21 століття це час автоматизації та комп’ютерних наук, це має вагомий вплив на буденне життя людей. Вже зараз ми не уявляємо свого життя без Інтернету, соціальних мереж та електронних джерел інформації. 21 століття відкриває нові можливості у сфері технологій та IT-продуктів. Все це може спрощувати користування потрібними застосунками та сервісами, що дають можливість вільно почуватись у часі комп’ютерних технологій. Важливе місце у сьогоденні зайняли чат-боти.

Чат-бот – це програма що використовує певну послідовність наперед продуманих алгоритмів дій або штучний інтелект, для обміну інформацією між людиною (користувачем).

Сучасна людина не уявляє життя без спілкування, купівель, пошуку інформації в Інтернеті. Сьогоденні технології дозволяють реалізувати IT-продукти для всіх цих потреб – це месенджери. Наприклад, Facebook, Telegram, Skype, Discord, Viber.

Стрімкий розвиток цих платформ, дає не тільки можливість користувачу спілкуватись і дізнаватись щось нове, але й можливість розробнику створювати чат-ботів на цій платформі.

Можливості чат-ботів та їх розвиток зробив їх привабливим як компаній так і простих користувачів. Одні з перших та найпростіших чат-ботів розробляли для сайтів, як інструмент спілкування з відвідувачами. Чат-боти стали важливою частиною багатьох магазинів, навчальних закладів і також державних і приватних бізнесів.

Створення чат-ботів вимагає чіткого розуміння протоколів HTTPS/TCP, принципів розробки RESTFUL API, вивчення API-бібліотек для взаємодії з певною платформою, а також вірний вибір мови програмування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Обґрунтування необхідності створення чат-ботів

Чат-боти в більшості використовуються для індивідуальних або ділових потреб. Таких як, інформаційні системи, продаж товарів, пошук користувачів/інформації, системи технічної підтримки. Чат-боти які реалізовані на комунікаційних платформах (месенджерах) використовуються для вирішення багатьох завдань і функціонал яких не базується на єдиному обміну текстових повідомлень з користувачем: для пошуку інформації, пошуку потрібних товарів, споживання контенту, а також для взаємодії клієнтів з компаніями.

Індивідуальні чат-боти, зазвичай використовуються як помічники людині і можуть виконувати певні функції: пошук інформації, парсинг інформації, управління календарем, зберігання інформації, масові розсилки і виклики, нотатки тощо. Такі чат-боти стають важливим інструментом в обробці інформації і її використанні. В більшості індивідуальні чат-боти створюються для виконання рутинних дій користувача, вони не складні в створенні і не потребують майбутньої підтримки.

Ділові чат-боти створюються з метою масштабного використання багатьма користувачами і призначенні для спілкування, продажу товарів, поширення інформації користувачу, та іншим завданням. Такі чат-боти складні в створені, потребують специфічних знань в цій галузі і подальшої підтримки від розробника.

Процес розробки та використання чат-ботів є обширний, все залежить від задач які чат-бот повинен виконувати і технологій які застосовуються під час його створення.

Чат-боти можуть замінювати певні професії людей і виконувати роботу швидше і якісніше. Розглянемо приклад бізнес чат-бота для продажу товару, такі боти виконують всі основні етапи співпраці з користувачем:

- Пошук товару.

- Реклама товару.
- Перегляд повної інформації про товар.
- Збір інформації користувача.
- Продаж товару.

Це дає змогу компанії не витрачати кошти на робітників, для виконання цих задач, і підтримувати співпрацю з користувачем цілодобово.

Також чат-боти можуть використовувати штучний інтелект, для повноцінної імітації людини при спілкуванні з користувачем. Таких чат-ботів можна розділити на два типи:

- Звичайне спілкування.
- Вузько направлене спілкування.

Чат-боти для звичайного спілкування навчені розуміти діалоги на прості теми, вони можуть підтримувати розмову з користувачем, відповідати на питання і запитувати користувачів про буденні справи. Зазвичай вони не розуміють складних, вузько направлених питань, а також не можуть розбирати граматично не правильні повідомлення.

В той час як вузько направленні чат-боти спроможні відповідати користувачу вузько направленою інформацією. Наприклад такі чат-боти можуть використовуватись в компаніях, які пропонують певні інформаційні продукти, в навчальних закладах для полегшення пошуку інформації про заклад і все що з ним пов’язано.

Тому на сьогоднішній час використання чат-ботів у бізнесі, державних і приватних закладах, а також у персональних цілях, економить час, матеріальні і людські ресурси, а також спрощує процеси взаємодії з користувачем.

1.2 Аналіз існуючих чат-ботів даної тематики

На даний період часу не існує дуже багато ботів з штучним інтелектом для навчальних закладів, оскільки процес створення і підтримки таких ботів, вимагають великих затрат часу і ресурсів.

ОсвітаВсім – це інформаційний телеграм бот, створений міністерством освіти для полегшення вступної компанії для студентів, його адреса «@inforesurs_gov_ua_bot». Цей бот реалізований на платформі Telegram, і використовує функціонал внутрішньої API.

Основа мета цього чат-бота – це допомогти майбутньому студенту оприділитись на який факультатив поступати, які документи потрібні для вступу у навчальний заклад, основні дати вступної кампанії з описом, а також переглянути навчальній заклади і їхні факультети.

За допомогою чат-бота можна отримати таку інформацію:

- Основні дати – інформація про етапи вступної кампанії.
- Відеороз'яснення – відеокурс з вибору спеціальності.
- Інформаційні матеріали – поради з вибору спеціальності.
- Вибір спеціальності – пошук за заданими параметрами.
- Перевірити документи – перевірка власних документів про освіту.
- Нормативні документи – основні законодавчі акти.

Також в чат-боті існує додатковий функціонал, це так звана технічна. підтримка:

- Q&A – популярні питання про вступну кампанію.
- Служба підтримка – контакти для звернення.

Оскільки чат-бот реалізований на платформі (месенджер) Telegram, він використовує вже готовий інтерфейс: reply/inline кнопки, стилізація, (рисунок 1.1) це облегшує розробнику, створення і підтримку бота, але також має свої мінуси, у вигляді обмеження можливостей.

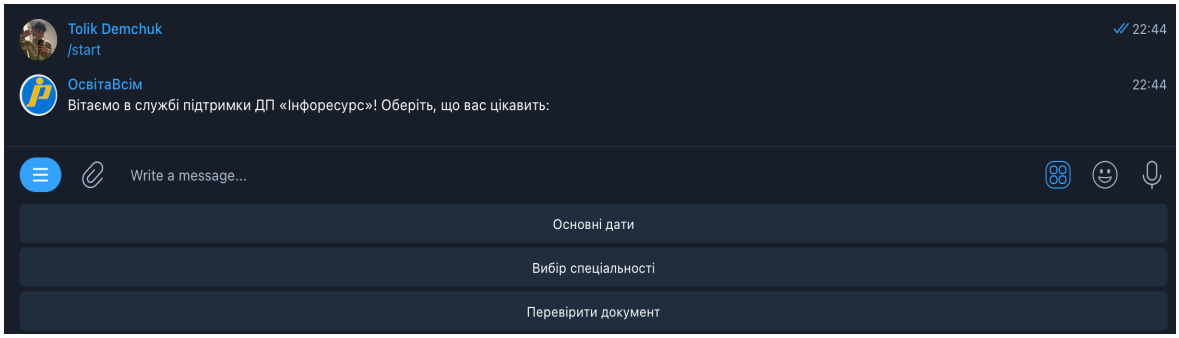

Рисунок 1.1 – Головний інтерфейс чат-бота ОсвітаВсім

При створенні чат-боту використовувалось Public Telegram API для взаємодії з платформою. А також одна з доступних мов програмування: Python, JavaScript, Ruby, C++, яка підтримує Telegram BOT API.

Далі наведений аналіз цього бота.

Плюси:

- Легкий у створенні і підтримки, оскільки створений на платформі.
- Автоматизований парсер вступних компаній навчальних закладів.
- Простота доступу користувача до бота.

Мінуси:

- Обмежений функціонал, оскільки створений на платформі Telegram.
- Обмеженість потоку даних, через використання протоколу https.
- Використовується тільки в певні періоди, під час вступної компанії.

Чат-бот ХНЕУ ЗНО Математика - це бот який був створений для навчальних цілей у університеті ХНЕУ іменні С. Кузнеця, його адреса «@HNEU_ZNO_math_bot». Його основна мета – це підготовка користувача до ЗНО з математики.

Створений чат-бот на платформі (месенджер) Telegram. Чат-бот використовує весь доступний функціонал Telegram API: кнопки, реакції, фото повідомлення (рисунок 1.2).

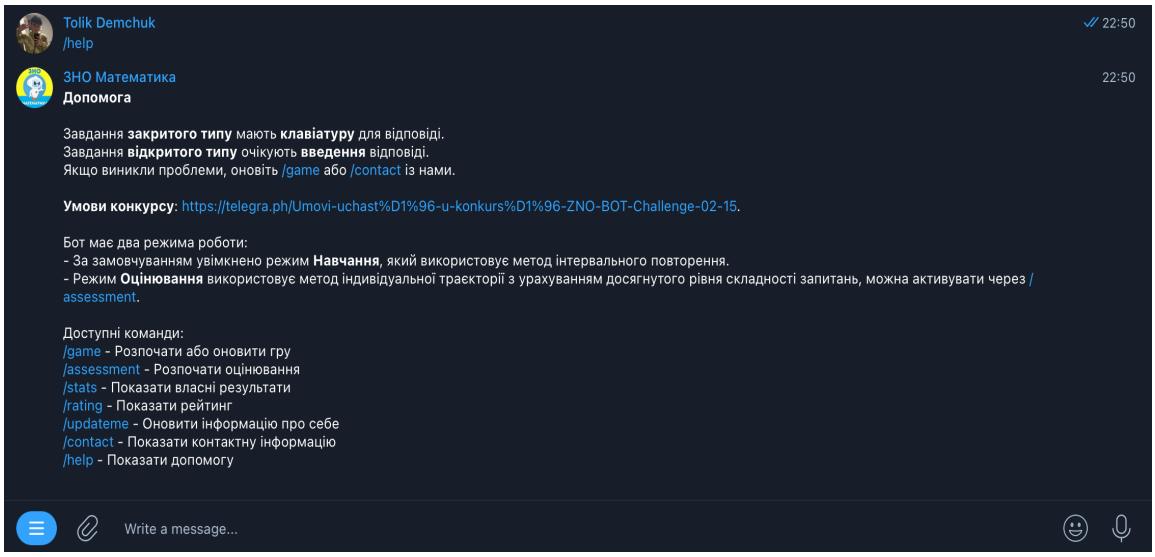

Рисунок 1.2 – Головний інтерфейс і меню чат-бота від ХНЕУ

Основний функціонал чат-боту:

- Надсилання математичних задач.
- Перевірка відповіді користувача.
- Формування рейтингу користувачів.
- Нагородження користувачів.
- Перегляд реєстрів ЗНО задача за певними категоріями.
- Аналіз знань користувача, і створення графіків.

Додатковий функціонал:

- Технічна підтримка чат-бота.

При створенні чат бота використовувались власні парсери, для отримання інформації про ЗНО.

Далі наведений аналіз цього бота:

Плюси:

- Легкий у використанні.
- Реєстр задач.
- Автоматизоване наповнення бота математичними задачами.
- Простота у використанні.

Мінуси:

- Обмежений функціонал, через розробку бота на існуючій платформі.
- Нагруженість чат-бота, через протокол обробки даних.

1.3 Мета та задачі

Основним завдання у дипломному проєкті – є створення чат-бота, який буде розпізнавати запитання користувача про Галицький коледж і відповідати на них. Також одна з головних задач цього бота, це можливість легко інтегрувати його, у наявні сервіси такі як:

- Сайти.
- Месенджери.
- Мудл.

Основна мету цього чат-боту це допомогти студентам і тим хто тільки планує поступати в навчальний коледж, швидко і якісно дізнатись інформацію, яка їх цікавить.

Це означає, що чат-бот повинен мати завжди обновлену інформацію, що до всього, що відбувається у навчальному закладі. Також мета цього бота, бути універсальним, тобто не бути розробленим тільки для одного навчального закладу або ресурсу, він повинен відповідати за велику кількість ресурсів, і мати всю інформацію з ними.

Всі основні задачі, які повинен виконував цей чат-бот:

- Можливість використання на багатьох ресурсах.
- Доступність на любому пристрої.
- Якісний аналіз запитання і вибір правильної відповіді.
- Простота у підтримці чат-бота.
- Швидке і легке доповнення чат-бота новою інформацією.

Додаткові задачі:

- Технічна підтримка.
- Веб-інтерфейс.

Також основна з задач, це легкість в обслуговуванні чат-бота, щоб користувач який не є досвідченим у розробці чат-ботів, міг легко підтримувати і керувати інформацією чат-бота. Організація структури чат-бота, є основною частиною розробки, якісного продукту (рисунок 1.3).

Рисунок 1.3 – Схема організаційної структури

Найголовнішим в ієрархії є адміністратор, він керує всіма проектами. Він має всі можливості у всіх проектах. Далі йдуть сервіси інтеграції, це так звані проекти, всі сервіси у яких є бот. Кожен з цих сервісів має свого бота і свою інформацію з прикладами запитань і відповідей.

Властивості і функції Адміністратора:

- Керування всіма проектами.
- Керування основною системою.

Властивості і функції проектів:

- Керування внутрішніми користувачами.
- Керування системою чат-бота.
- Зберігання всієї інформації для коректної роботи чат-бота.
- Керування базою даних.

Властивості і функції бота:

- Отримання запитань.
- Аналіз запитань і відправлення відповідей.

2 ПРОЄКТУВАННЯ ЧАТ-БОТА

2.1Проєктування структури чат-бота

Чат-боти зараз є стандартними додатками, користувачі використовують їх всюди, на веб-сайтах, у месенджерах, у додатках, а також вбудовані чат-боти у пристроях. Існує багато типів чат-ботів, які мають свої оригінальні функції і виконуються певні задачі. Також існують чат-боти, які є повноцінною системою для вирішення масштабних задач, звісно чат-ботам не вийде повноцінно замінити веб-додатки, додатки на смартфонах або веб-сайти. Тому для певних задач є свох типи ботів для їх вирішення. Ось деякі з них:

- Боти асистенти – це боти створенні для допомоги користувачу у вирішені певних простих однотипних задач. Такі боти можуть бути інтегрованими у системи або пристрої, для швидко використання функцій цієї системи.
- Пошукові боти – це боти створенні для допомоги пошуку користувачу інформації. Ці боти можуть бути двох видів: інтегровані і глобальні. Інтегровані це боти які виконуються пошук по обмеженій інформації у сервісі де цей бот інтегрований. Глобальні це боти які виконують пошук по всі системі і у веб ресурсах.
- Боти візитки – це боти створенні для надання основної інформації про ресурс, який створив цього бота. Це так звані візитки компаній, де вказана вся основна інформація, способи зв’язку і посилання на інші ресурси компанії.
- Боти оператори (технічна підтримка) – це боти створенні для допомоги користувачу з вирішенням його проблеми, основна задача таких ботів, це з’єднання користувача з оператором, і подальший обмін інформації користувача з оператором.
- Боти з штучним інтелектом – це боти створенні для імітації співбесідника для користувача. Такі боти реалізовуються у різному напрямку,

для спілкування з користувачем, для інформування користувача, для вирішення навчальних проблем користувача.

Для виконання дипломного проєкту – був розроблений тип інтегрованого чат-бота з штучним інтелектом, для допомоги користувачу з пошуком інформації про навчальний заклад. Як сказано вище, основна мета – це допомога користувачу з пошуком інформації про пошук інформації про: навчальний заклад, спеціальності, документи, списки, рейтинги, а також допомога з вступною компанією. Чат-ботом можуть користуватись студенти, абітурієнти, і прості користувачі, які планують поступати у навчальний заклад. Етапи створення такого бота можна розбити на багато підпунктів.

Етапи створення чат-бота:

- Проєктування архітектури чат-бота.
- Проєктування бази даних чат-бота (моделі, таблиці, зв’язки).
- Проєктування бота.
- Розробка системи авторизації.
- Розробка системи мікросервісів.
- Розробки системи зберігання і пошуку питань і відповідей.
- Розробка системи редагування інформації для чат-бота.
- Тестування.

Чат-бот допоможе адміністрації навчального закладу, автоматично відповідати користувачам на прості запитання про навчальний заклад, що буде економити час для працівників закладу. Також є можливість просто доповняти чат-бота новою інформацією або змінювати вже існуючу. Цей чат-бота також є візиткою навчального закладу, який демонструє рівень професіоналізму закладу у вивченні і використанні новітніх технологій і повсякденному житті закладу. Завдяки функціоналу цього чат-бота, не потрібно на пряму зв’язуватись з користувачем, що економить багато часу працівникам навчального закладу.

Вимоги до функціоналу чат-бота зображенні на діаграмі IDEF0 (рисунок 2.1), яка дозволяє швидко ознайомитись з функціями чат-бота.

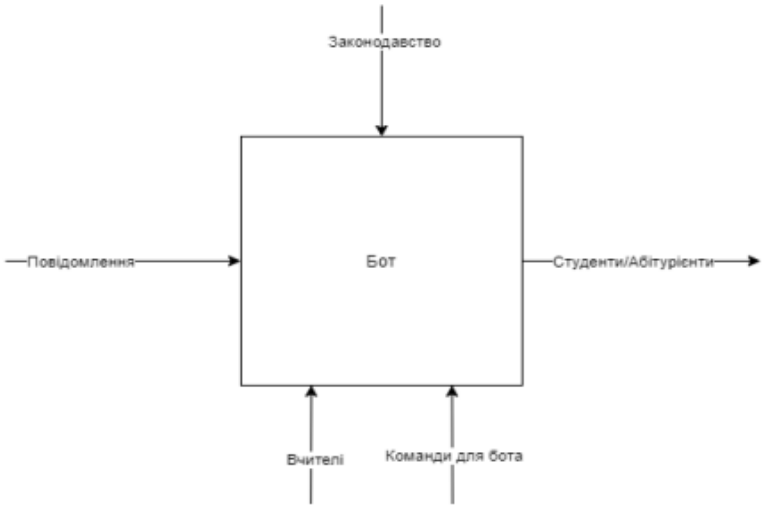

Рисунок 2.1 – IDEF0 схема діяльності чат-бота

2.2 Проєктування логіки чат-бота і бази даних

Логіка чат-бота є найважливішою частиною всього проєкту. Правильно вибрані рішення на етапі проєктування чат-бота впливають на багато факторів роботи бота в майбутньому.

Такі фактори як:

- Швидкодія додатку.
- Важкість підтримки чат-бота.
- Оптимізація чат-бота під всі існуючі платформи.
- Доповнення чат-бота новою інформацією.

Як сказано вище основна з задач даного чат-бота, це можливість бути інтегрованим у будь які сервіси. І виконувати різний функціонал на сервісах. Тому на етапі проектування логіки чат-бота, було прийнято рішення щодо архітектури чат-бота.

Головні етапи архітектури чат-бота:

- Для кожного з сервісів де буде інтегрований цей чат-бот, потрібно створювати окремий проєкт в системі. Цей проєкт буде відповідати за користувачів які, редагують інформацію, яку використовує чат-бот. Також кожен з цих проєктів буде мати власного бота, який створюється автоматично після створення проєкту, і кожен з них має свою особисту інформацію яку

використовує для вірною роботи. Основна властивість цих проєктів, це те що вони не можуть бути пов’язаними між собою.

- Логіка авторизація, є розбитою на два основних види: авторизація адміністратора головного проєкта, авторизація користувачів у кожному мікро проєкті. Тобто це означає, що адміністратори головного проєкта повинні бути унікальними в плані даних для авторизації у систему, в ту чергу як авторизація користувачів у мікро проєктах повинна бути унікальна в межах цього проєкта.
- Логіка управління база даних у проєкті, вимагає використання мікро сервісного виду архітектури бази даних. Оскільки в системі буде існувати багато чат-ботів, то кожен з них повинен мати власну базу даних.

Проєктування база даних – це головний етап проєктування головної системи, який впливає на всю подальшу роботу додатку. Цей етап заключає в себе проєктування таблиць і зв’язків між ними, типи даних які буде зберігати кожна колонка таблиці, а також мета і задачі кожної з таблиць в системі. База даних системи містить наступні основні сутності: Адміністратори, Користувачі, Проєкти, Інформація чат-ботів, Чат-боти.

Для системи інтегрованих чат-ботів потрібно створити базу даних, в якій будуть зберігатись наступні відомості:

- Про адміністраторів.
- Про користувачів.
- Про проєкти.
- Інформація для коректної роботи чат-ботів.
- Системна інформація чат-ботів.
- Історія.
- Системна інформація.

Первині ключі повинні знаходиться в кожній таблиці даної бази даних, зазвичай це колонка “id”, яка буде дозволяти створювати зв’язки між таблицями, також такі поля можна назвати полями ідентифікаторами відомостей в таблиці. В базах даних можу існувати три існуючі види зв’язків.

Такі як:

- One to One (Один до одного) – це зв'язок який, об’єднує одне поле з іншими, тобто в таблиці не може існувати ще один зв’язок з цим самим зовнішнім ключем. Звідси виходить таке правило, що кожне з полів даної таблиці може посилатись тільки на одне унікальне поле з іншої таблиці. Основна структура такого зв’язка це те що батьківська і дочірня таблиця повинні мати унікальні посилання між собою в межах цих таблиць.
- One to Many (Один до багатьох) – це зв'язок який, об’єднує одне поле з головної таблиці з полем в іншій таблиці, також він передбачає умову, що поля в головній таблиці можуть посилатись на одне і теж саме поле з зовнішньої таблиці. Основна структура такого зв’язка це те що дочірня таблиця може мати безліч посилань на один і той самий запис з батьківської таблиці.
- Many to Many (Багато до багатьох) – це зв'язок який, об’єднує одне поле з головної таблиці з полем в іншій таблиці, також він передбачає умову, що поля в дочірній таблиці можуть посилатись на один і той самий запис у батьківській таблиці, і так само може відбуватись з батьківською таблицею.

Під час проєктування бази даних був отримані всі сутності бази даних, які мають наступний структурний вигляд (таблиця 2.1-2.5).

Таблиця 2.1 – Адміністратори

Назва стовбця	Тип даних	Nullable	Unique
username	varchar(256)	False	True
password	bitarray	False	False
email	varchar(256)	True	True
full_name	varchar(256)	True	False
age	int	True	False

Таблиця 2.2 – Користувачі

Назва стовбця	Тип даних	Nullable	Unique
username	varchar(256)	False	True
password	bitarray	False	False
email	varchar(256)	True	True
full_name	varchar(256)	True	False
age	int	True	False
created_at	date	False	False
project_id	Int	False	False

Таблиця користувачів є найбільшою і найважливішою таблицею. В ній є обов’язкові і не обов’язкові стовбці.

Таблиця 2.3 – Проєкти

Назва стовбця	Тип даних	Nullable	Unique
id	int	False	True
name	varchar(256)	False	False
description	text	False	False
disable	boolean	False	False

Також всі таблиці реалізованні на PostgreSQL, на теперішній час це одна з найпопулярніших баз даних у розробці веб-додатків.

Таблиця 2.4 – Інформація для чат-бота

Назва стовбця	Тип даних	Nullable	Unique
id	int	False	True
question	varchar(256)	False	False
answer	varchar(256)	False	False
project_id	int	False	False

Таблиця 2.5 – Чат-бот

Назва стовбця	Тип даних	Nullable	Unique
id	int	False	True
name	varchar(256)	False	False
project_id	int	False	False

2.3 Вибір та опис засобів реалізації чат-бота

Сучасний чат-бот, який не реалізований на платформах, таких як: Telegram, Viber, Messager, реалізується існуючою архітектурою. Структура бота складається з двох основних сторін це – backend і frontend, які реалізовується окремо і незалежно одне від одного. Також для кожної з цих частин є свій набір технологій для реалізації.

Оскільки чат-бот буде мати можливість інтегруватись у існуючі web-ресурси, його можна називати web-додатком. Web-додатки це люба програма, яка складається з двох частин: frontend і backend, яка виконує певний функціонал, і використовує веб-браузер для взаємодії з користувачем. Як правило web-додаток обробляє запити які надходять від браузера користувача і повертає якийсь результат. Запити – “request” це спосіб взаємодії користувача з web-додатком, він підтримує 4 основних методи: GET, POST, DELETE, UPDATE. Кожен з цих методів має свої відмінності від інших і своє призначення. Наприклад POST запит дозволяє відправляти і повертати request body, це дані у форматі JSON, а також POST підтримує шифрування даних в запиті, а GET дозволяє робити простий запит без додаткових даних і шифрування. Все це відбувається через протокол HTTP/HTTPS. Також обмін даними може відбуватись через інші протоколи такі як: TCP/IP, FTP.

Сучасні web-додатки використовують велику кількість технологій які підтримують багату відомих язиків програмування, також web-додатки на пряму працюють з базами даних.

Frontend (клієнтська частина) – це користувацький інтерфейс, який відображається в браузері на web-сторінці. Цей інтерфейс дозволяє виконувати певний функціонал, який працює на стороні користувача. Все що бачить користувач на web-сторінці і все з чим він може взаємодіяти.

Backend (серверна частина) – серверна частина, яка зазвичай називається API або REST API. Це програма яка виконується на сервері, до якої можна звернутись по певному домені і порті. Серверна частина приймає всі запити, які надсилає користувач з клієнтської частити web-додатка, обробляє їх і повертає відповідь клієнту. Також backend виконує всі операції які зв’язані з базами даних, і має пряме підключення з базою даних, яка може знаходить на одно або на іншому сервері ніж backend частина web-додатку.

Оскільки чат-бот інтегрується у всі потрібні ресурси навчального закладу, frontend частину не потрібно реалізовувати, весь функціонал чат-бота знаходиться на backend частині.

Backend частина ділиться на дві відомі архітектури:

- Мікросервісна.
- Модульна.

Мікросервісна архітектура – це одна з самих відомих і популярніших архітектур backend частини. Вона дозволяє розробнику розділяти додаток на малі частини, так звані сервіси. Головна мета цієї архітектури це те, що всі сервіси є незалежними і кожен з них відповідає за свій функціонал, а також має доступ до бази даних, яка не взаємодіє з таблицями в іншому сервісі (рисунок 2.2). В цієї архітектури є свої плюси і мінуси. Також така архітектура займає багато часу на вірне проектування додатку.

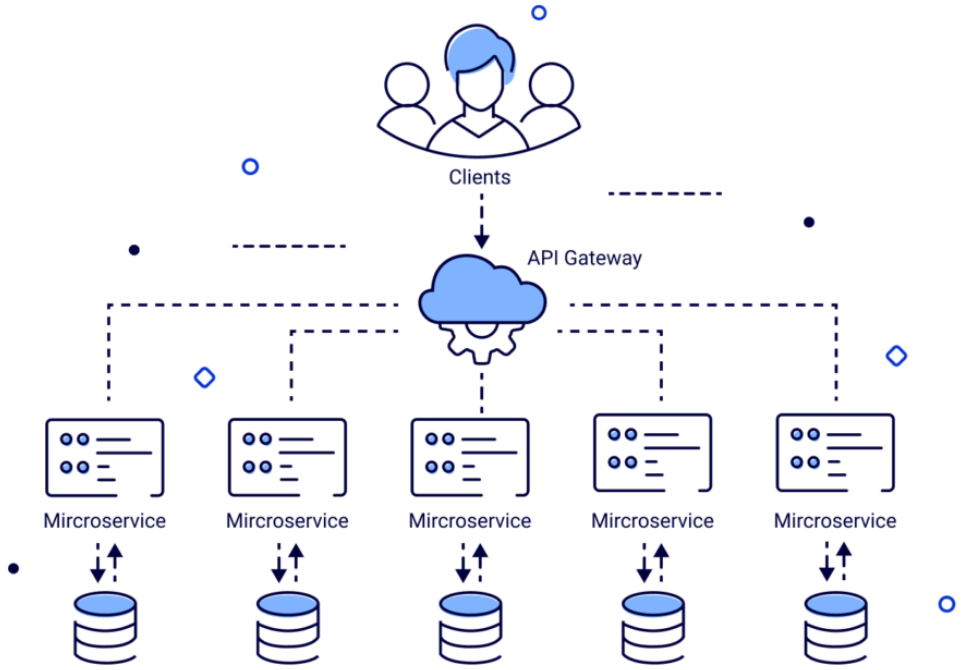

Рисунок 2.2 – Схема роботи мікросервісної архітектури

Плюси мікросервісної архітектури:

- Простота в розробці і подальшої підтримки додатку.
- Функціонал розбитий на сервіси, це рішення уникає сильного навантаження на весь додаток і на базу даних.
- Розподілена база даних, для кожного сервісу своя незалежна база даних.
- Зміна одного функціоналу (сервісу) не впливає на інший сервіс.

Мінуси мікросервісної архітектори:

- Складне і довге проектування додатку.
- Великий за розмірами на диску додаток.
- Велика кількість не пов’язаних між собою баз даних.
- Не підходить для реалізації всіх додатків.

Важливою частиною успішного додатку є вибір вірних технологій для певної задачі. Як сказано вище, основною задачею чат-боту це можливість бути інтегрованим у різні сервіси, це означає що backend частина яка є всієї логічною частиною чат-бота, повинна бути багатопроцесорною системою.

Багатопроцесорність дозволяє додатку виконувати функції не лінійно, як звичайні додатки, а паралельно, тобто чат-бот може опрацьовувати багато запитань одночасно.

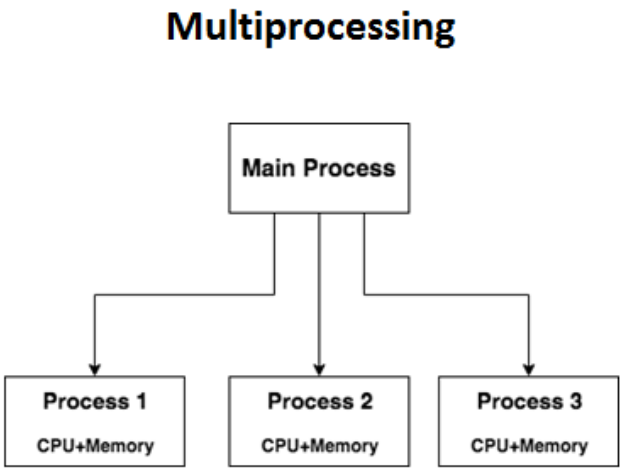

Рисунок 2.3 – Схема роботи багатопроцесорного додатку

Для створення основного API можна використовувати безліч мов програмування: Python, Ruby, Java, JavaScript, Golang, C# (.Net).

Python – це динамічна мова програмування, із автоматичним керуванням пам’яті, орієнтована для збільшення продуктивності. На теперішній час це є одна з найвідоміших мов програмування, яка є легка для початківців. Плюси використання Python:

- Динамічне типізація.
- Підтримка функціонального програмування.
- ООП.
- Легкий рівень для початкового вивчення мов програмування.
- Багато бібліотек, які спрощують і збільшують можливості для створення додатків.

Також Python має багато відомих фреймворків, які використовуються для створення web-додатків, такі як: Django, Flask, FastAPI, Starllete, Pyramid.

Для повноцінного web-додатку потрібно використовувати базу даних, яка буде використовуватись з API. Існують два види баз даних: реляційна база даних, нереляційна база даних. Відомі реляційні СУБД: MySQL, PostgreSQL, SQLite та багато інших.

Переваги використання реляційних баз даних:

- Строгі правила проектування.
- Дані зберігаються в нормалізованих таблицях.
- Простота і доступність для користувача.
- Всі дані нормалізовані.
- Підтримка відношення між даними.
- Єдиний язик для створення запитів, який використовується для всіх відомих СУБД – SQL.

Недоліки використання реляційних баз даних:

- Не під всі web-додатки зручно використовувати таблиці.
- База даних займає багато пам’яті.
- Довге отримування даних з таблиць.
- Нагруженість бази даних.

Для вирішення задач багатопроцесорності підходить Python Framework FastAPI, це швидкий, простий і легкий фреймворк, який підтримує лінійний і багатопроцесорний режим додатку. Також він підтримує всі відомі архітектури backend додатку. Як і у кожної технології, у FastAPI є свої плюси і свої мінуси.

Плюси використання FastAPI:

- Легке масштабування додатка.
- Підтримка лінійного і багатопроцесорного режиму додатка.
- Простота вивчення і використання.
- Надзвичайно швидкий фреймворк.
- Інтегровані засоби валідації даних.
- Підтримка драйвера для багатопроцесорного керування базами даних.

Мінуси FastAPI:

- Лінійний режим додатку, є слабкою частиною фреймворка.
- Для запуску додатка потрібні додаткові Python бібліотеки.

Середовище розробки – це середовище, яке автоматизує певні моменти розробки програмного забезпечення. Воно допомагає розробнику з написанням коду, підтримує багато функціоналу. Проєкт розроблений з підтримкою PyCharm. Це програма відомої компанії JetBrains, яка спеціалізується на розробці застосунків, які допомагають з розробкою додатків.

Для розробки backend використовується PyCharm. Це середовище розробки, спеціально створене для мови програмування Python (рис. 2.4). PyCharm має багато функціоналу для створення додатків, на мові Python.

Підтримка мов програмування і технологій:

- Python.
- Django.
- FastAPI.
- Flask.
- Штучний інтелект.
- Pandas/Modin.

Можливості PyCharm:

- Аналіз коду, допомога з синтаксисом.
- Інструменти для керування базами даних.
- Створення virtualenv для Python проєкту.
- Тестування.
- Інструменти створення бібліотек.

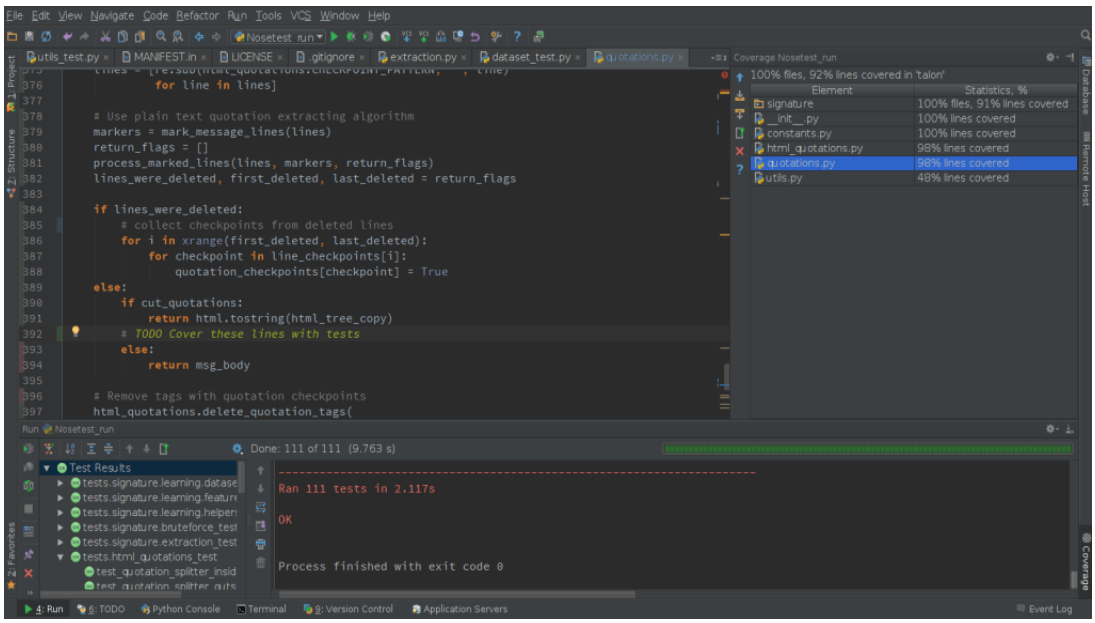

Рисунок 2.4 – Інтерфейс середовища розробки PyCharm

Оскільки проєкт реалізований як web-додаток, він використовує протокол HTTPS, для обміну даними з сервером і користувачем. Цей протокол має свій набір методів, які дають можливість роботи запити до сервера з певними властивостями. Ось деякі з них:

- Get.
- Post.
- Patch.
- Delete.
- Options.

Get і Post запити вважаються найбільш популярнішими серед розробників web-додатків. Вони відрізняються певними властивостями (таблиця 2.6)

Таблиця 2.6 – Порівняння Get і Post запитів

Назва	GET	POST
Кнопка “Назад” / Перезавантаження сторінки	Безпечно	Дані будуть відправлені заново. Браузер попередить про повторну відправку даних
Добавлення в закладки	Так	Ні
Хешування	Так	Ні
Тип кодування	application/x-www-form-urlencoded	application/x-www-form-urlencoded multipart/form-data
Історія	Залишається в історії браузера	Не залишається в історії браузера
Обмеження по довженні	Є обмеження у 2048 символів	Немає обмеження
Безпечність	Менше безпечний метод передачі	Більш безпечний, так як дані не відображаються користувачу

Тому для розробки проєкта використовується запроси Post, оскільки вони наділенні більшим функціоналом.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація основних модулів системи

Реалізація чат-бота була розбита на модулі, які потрібно виконувати поступово. Завершення який означитиме закінчення проекту, і гарантію його функціонування:

- Створення директорії проєкта.
- Створення основного API застосунку.
- Створення бази даних.
- Створення системи авторизації.
- Створення алгоритмів розпізнавання запитань.
- Створення ієрархії проєктів.

Створення REST API це доволі простий етап створення системи, він здійснюється за допомоги готових інструментів фреймворка, FastAPI. Тому першою основною частиною буде створення application. Оскільки в проєкті використовується мікросервісна архітектура, ця функція винесена в окремий файл. Алгоритм створення application зображено на лістингу 3.1.

Лістинг 3.1 – Створення FastAPI application

```
def get_application() -> FastAPI:
    application = FastAPI(
        title=settings.PROJECT_NAME,
        version=settings.PROJECT_VERSION
    )
    application.add_middleware(
        CORSMiddleware,
        allow_origins=settings.ALLOWED_HOSTS or ["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"]
    )
    application.include_router(router)
    return application
```

Цей алгоритм буде повертати application, на якому знаходиться вся система, включаючи бота. FastAPI має встроєний WSGI-сервер для запуску web-додатків. Uvicorn – простий WSGI застосунок, головною задачою якого є запуск web-додатку на сервері. Керувати Uvicorn можна через просту командну строку.

Запуск просто web-додатку зображений на рисунку 3.1.

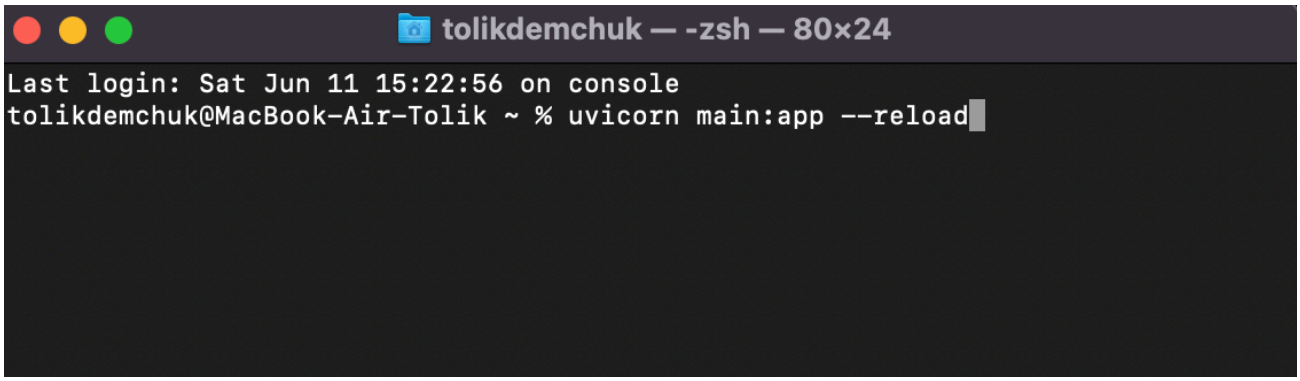

Рисунок 3.1 – Запуск просто web-додатку

Розберемо що саме виконує ця команда:

- uvicorn – викликає веб сервер.
- main:app – вказує шлях до application, де main – це назва файлу, а app – це назва application.
- --reload – це параметр що дозволяє автоматично перезапускати сервер, коли він замітить зміни в коді системи.

Директорія проєкта – це є основна задача на ранній стадії розробки проєкту. Директорія повинна бути створена такою, щоб було просто зрозуміти за що відповідають певні папки і файли в проєкті (рисунок 3.2).

Інтуїтивно зрозуміла директорія реалізації чат-бота, облегшує розробки і подальшу підтримку чат-бота. Також це є основною частиною мікро сервісної архітектури.

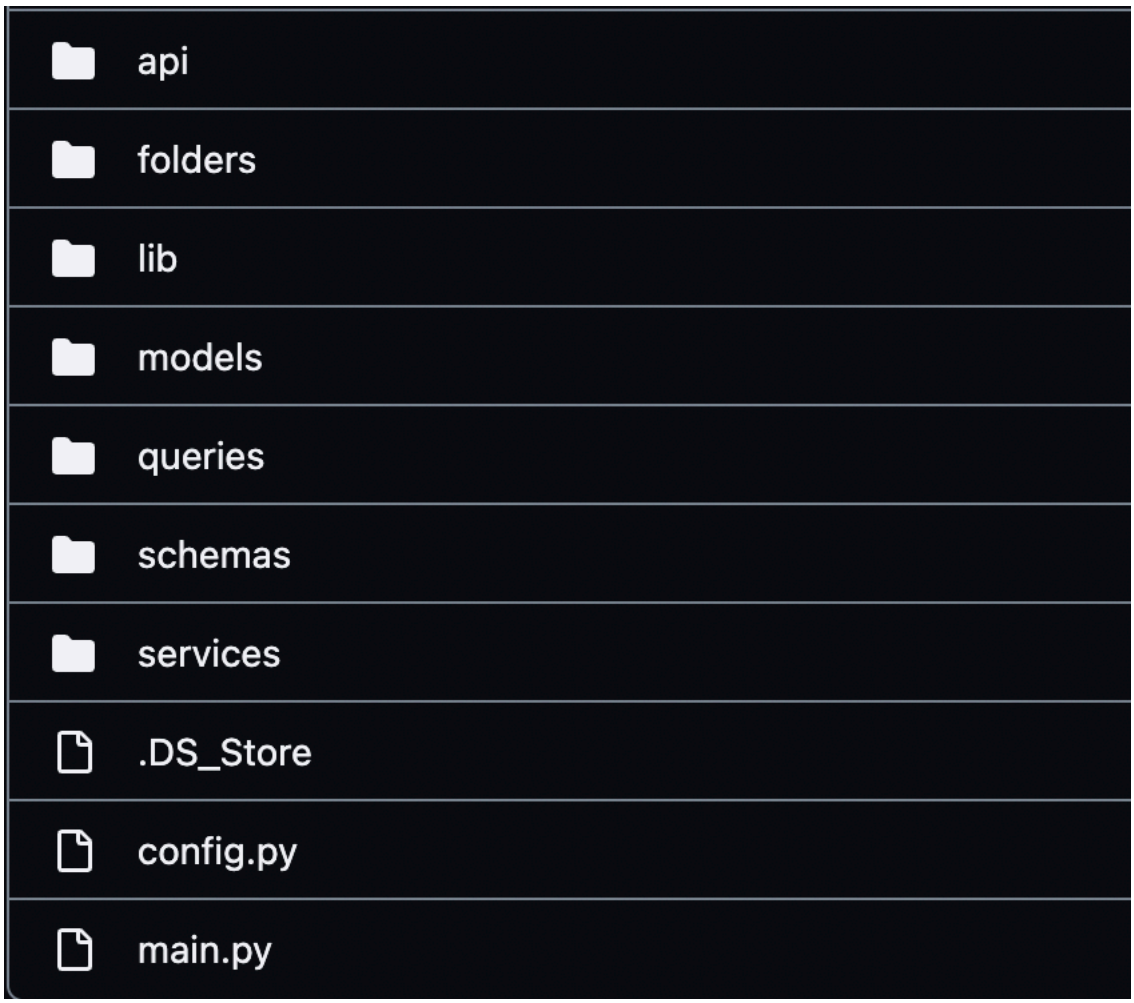

Рисунок 3.2 – Директорія проєкту

Розберемо детальніше за що відповідає кожна з цих папок і файлів:

- main.py – це основний файл, в якому створюється application, в ньому знаходиться єдина функція яка повертає FastAPI application.
- config.py – це файл в якому оприділяється основа конфігурація проєкта, по типу: ip, port, name, headers, methods, description, db_uri.
- services – це папка в які знаходять всі сервіси, це файли які взаємодіють з базою даних для кожного сервісу окремо, і виконують все набір CRUD методів: create, read, update, delete.
- schemas – це так звані FastAPI схеми, які виконують роль валідаторів даних. Вони мають можливість перевіряти дані, як ті що приходять запитом на сервер, так і ті які повертає сервер користувачу.

- queries – це папка яка зберігає в собі всі стандартні запити до бази даних, такі як: select, update, delete, insert. Це зроблено для того щоб, вся основна логіка взаємодії з базою даних находилась в одній категорії.

- models – це каталог, який описує всі сутності бази даних, так званні моделі для бази даних. Кожна з моделей описує певну таблицю, її типи даних, звязки з іншими таблиця і підключення до бази даних.

- lib – це додатковий каталог, який розміщає в собі всі додаткові функції, які не маю прямого відношення до web-додатку. Це можуть бути операції хешування і шифрування для пароля, методи зберігання файлів і каталогів на диску.

- folders – це додатковий каталог, який зберігає в собі інформацію для тестування системи, і приклади її роботи. Він не впливає на функціонування системи, а використовується тільки під час розробки проєкту.

- api – це один з основних каталогів, який зберігає в собі всю логіку застосунку, функції, http-методи і endpoints. Саме в цьому каталозі описаний основний функціонал чат-бота.

Створення сутностей база даних, так званих моделей бази даних, це важливий крок в якому оприділяються всі таблиці, їхні зв'язки між собою, типи даних в стовбцях, назви, і властивості. Ці моделі дозволяють використовувати ORM. Це відомий спосіб взаємодії з базами даних в коді програми. Його основна мета це спростити написання SQL запитів і використовувати зазделагіть підготовленні функції, такі як select, update, delete, insert.

В даному проєктів існують 5 основних сутностей (моделей):

- Administrator – модель відповідає за таблицю Адміністрація і зберігає всі дані про адміністрацію системи.

- Users – модель відповідає за таблицю Користувачі і зберігає всі основні дані про користувача, такі як: ім'я, пароль, пошту, статуси блокування. Пароль зберігається у хешованому вигляді, це зроблено спеціально для захисту персональних даних користувача, в разі якщо хтось отримає

несанкціонований доступ до даних, то він не зможе дізнатись пароль користувача і не зможе зайти в його профіль.

- Projects – модель призначена для зберігання даних про ресурс де інтегрований чат-бот. Також це є батьківською моделю чат-бота, яка створює унікальний чат-бот, для сервіса. Зберігає в собі основну інформацію про сервіс.

- Questions – це основна модель, яка зберігає в собі інформацію про всі можливі запитання і відповіді для певних ботів. Також вона має прямий зв'язок з моделю Projects.

- Bots – модель для зберігання всієї інформації про чат-ботів, про їхній опис, назви, сервіси. Вона посилається на моделі запитань і проєкту. Також вона відповідає за стан самого чат-бота.

На лістингу 3.2 зображена модель Users:

Лістинг 3.2 – Реалізація моделі Users.

```
class Users(Base):
    __tablename__ = "Users"

    id      =      Column(Integer,      Identity(always=True),
primary_key=True)
    username     =     Column(String(63),     nullable=False,
unique=True)
    password = Column(UnicodeText, nullable=False)
    full_name = Column(String(128), nullable=False)
    age = Column(Integer, nullable=True)
    scopes = Column(ARRAY(UnicodeText), default=["*"])
    created_at = Column(DateTime, server_default=func.now())
```

Процес створення бази даних, надзвичайно простий, потрібно мати створенні моделі які унаслідуються від Declarative Base Model, після опису всіх моделей добавити команду, для їх створення, зображено на лістингу 3.3.

Лістинг 3.3 – Створення бази даних

```
Base = declarative_base()
```

```
Base.metadata.create_all(engine)
```

Існує багато методів реалізації системи авторизації користувачів, але кожна з них має свій функціонал і логіку роботи. Ось деякі з них:

- Авторизація на основі сесій.
- Авторизація на JWT токенах.
- Авторизація на основі cookie.
- Авторизація на основі даних про пристрій.

Всі вони мають свої плюси і мінуси, але для реалізації проєкту був вибраний спосіб на основі JWT токенах.

В проєкті авторизація з JWT токенами реалізована за допомогою бібліотеки python-jose. Ця бібліотека дозволяє шифрувати токени і розшифровувати їх. Логіка такої авторизації заключається в тому що користувач при надсиланні запитів на сервер, автоматично надсилає authorization header з вбудованим токеном.

Самий JWT токен – це зашифрований json об’єкт, який зберігає в собі інформацію про користувача, а також час коли був створений цей токен. Це дозволяє роботи тимчасові токени, які є своєрідним захистом від викрадення токена.

Отже під час авторизації користувач отримує новий згенерований токен. І може використовувати його для подальшого доступу до системи. Генерація JWT токена зображена на лістингу 3.4.

Лістинг 3.4 – Генерація нового JWT токена.

```
def create_user_tokens(user: Users):
  access_token_expires =
  timedelta(minutes=settings.SECURITY_ACCESS_TOKEN_EXPIRE_M
  INUTES)
  refresh_token_expires =
  timedelta(minutes=settings.SECURITY_REFRESH_TOKEN_EXPIRE_
  MINUTES)

    # Create access token
```

```
        access_token = create_token(
            data={
                "sub": user.username,
                "scopes": user.scopes
            },
            expires_delta=access_token_expires
        )
        # Create refresh token
        refresh_token = create_token(
            data={
                "sub": user.username
            },
            expires_delta=refresh_token_expires
        )

        return access_token, refresh_token
```

Таким методом користувач отримує новий токен для подальшої роботи в системі, в проєкті система авторизації реалізована так що, звичаний токен користувача може бути дійсним тільки 30 хвилин, після чого користувач може отримати новий токен, завдяки refresh токену як знаходиться в cookie, яку користувач отримав під час успішної авторизації. Алгоритм генерування refresh токена зображений на лістингу 3.5.

Лістинг 3.5 – Алгоритм збереження refresh токена в cookie.

```
    def set_refresh_token_cookie(response: Response, token:
str):
        response.set_cookie(
            key=settings.SECURITY_REFRESH_TOKEN_COOKIE_KEY,
            value=token,

expires=settings.SECURITY_REFRESH_TOKEN_COOKIE_EXPIRES,
            path=settings.SECURITY_REFRESH_TOKEN_COOKIE_PATH,

domain=settings.SECURITY_REFRESH_TOKEN_COOKIE_DOMAIN,
```

```
        httponly=settings.SECURITY_REFRESH_TOKEN_COOKIE_HTTPON,
secure=settings.SECURITY_REFRESH_TOKEN_COOKIE_SECURE,
        samesite=settings.SECURITY_REFRESH_TOKEN_COOKIE_SAMESIT
        )
```

Основна логіка такої системи авторизації – це обмін токенами з сервером і їхня валідація в кожному запиті (рис. 3.3). На даний момент часу це вважається одна з найбільш захищених систем авторизації, але в неї є мінуси. Такі як, відправлення токена в кожному запиті, це означає, що сервіс в якому інтегрований чат-бот, повинен десь зберігати токен і автоматично підставляти його для кожного нового запиту. Що ускладнює інтеграцію чат-бота у різні сервіси.

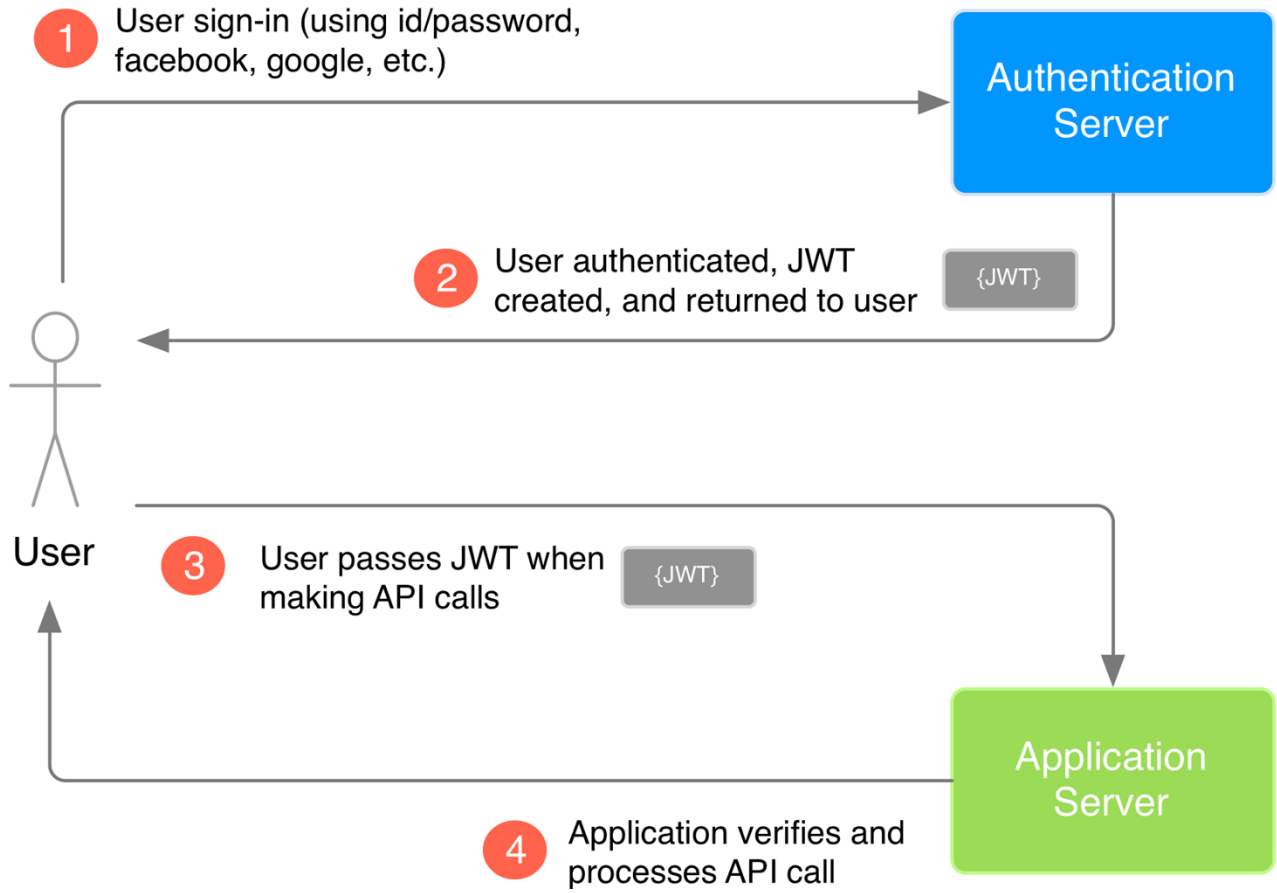

Рисунок 3.3 – Алгоритм роботи JWT токенів

При авторизації користувача в системі, виконується алгоритм валідації даних, таких як username і password, і створення токена для авторизації користувача. Алгоритм авторизації користувача зображений на лістингу 3.6.

Лістинг 3.6 – Алгоритм авторизації користувача.

```
# Generate the access token
@router.post(
    "/login",
    response_model=TokenResponseSchema,
    name="users:login"
)
def login_users(
        request: Request,
        response: Response,
        form_data: OAuth2PasswordRequestForm = Depends(),
        user:                UsersService                =
Depends(get_service(UsersService))
):
    # Check if user data is correct
    user_data = user.get_by_username(form_data.username)
    if            not            (user_data            and
verify_password(form_data.password)):
        raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST)
    access_token,             refresh_token             =
create_user_tokens(user_data)
    # Set cookie
    set_refresh_token_cookie(response, refresh_token)
    return {
        "access_token": access_token,
        "token_type": "bearer"
    }
```

Також під час реєстрації нових користувачів у системі, над паролем відбувається метод хешування pkdef2, який хешує пароль 200 тисяч раз одним і тим самим хешем.

Хешування – це криптографічні функція, яка перетворює дані у бітовий набір символів, хешування це одностороння функція, це означає, що з хешу не можливо дістати початкові дані. Тому коли користувач виконує авторизацію, то система хешує пароль який він увів, після чого зрівнює хеші, і якщо вони одинакові, значить користувач ввів вірний пароль і його можна пропускати. Алгоритм перевірки пароля зображений на лістингу 3.7.

Лістинг 3.7 – Алгоритм перевірки хешів паролів.

```
def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)
```

Алгоритмів розпізнавання запитань є безліч, в проєкті використовується алгоритми Левенштейна. Цей алгоритм зрівнює запитання на процес подібності за допомоги власних алгоритмів, які вже є готові у бібліотеці. Принцип роботи цього алгоритма заключається в тому що він комбінує слова в реченні, а також символи в словах, і визначає процент подібності з іншими запитанням. Таким способом можна дізнаватись що сам питає користувач у чат-бота і дати максимально можливу коректну відповідь (рисунок 3.4).

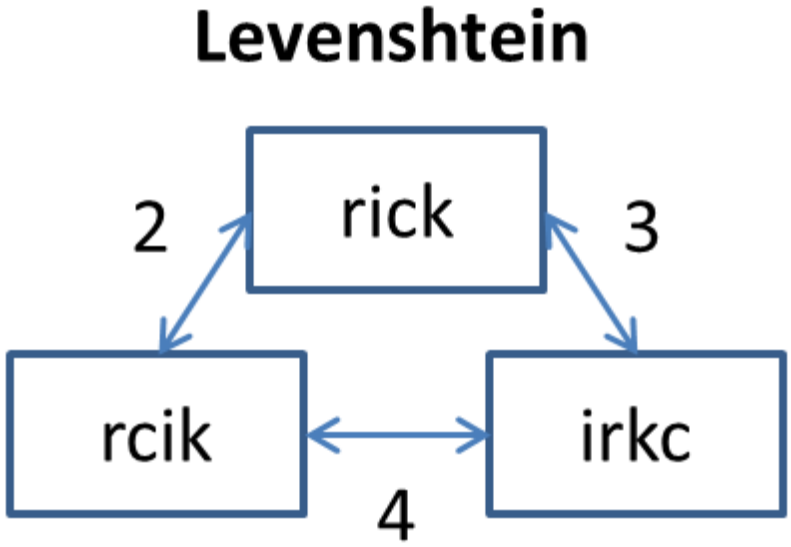

Рисунок 3.4 – Схема роботи алгоритма Левенштейна

Такий спосіб є самим найвірнішим вирішенням задачі бота. Оскільки створювати повноцінний штучний інтелект, в цьому випадку є не практичним,

через те що, штучний інтелект вчиться на певному наборі даних. А оскільки дані які буде використовувати цей чат-бот будуть з часом змінюватись, то його навчання займало б надзвичайно багато часу і ресурсів. Тому оптимальний варіант це використовувати цей алгоритм для розпізнавання запитань і пошуку відповідей. Реалізація алгоритма Левенштейна зображена на лістингу 3.8.

Лістинг 3.8 – Реалізація алгоритма Левенштейна.

```
@lru_cache(None)
    def min_dist(s1, s2):

        if s1 == len(a) or s2 == len(b):
            return len(a) - s1 + len(b) - s2

        # no change required
        if a[s1] == b[s2]:
            return min_dist(s1 + 1, s2 + 1)

        return 1 + min(
            min_dist(s1, s2 + 1),      # insert character
            min_dist(s1 + 1, s2),      # delete character
            min_dist(s1 + 1, s2 + 1),  # replace character
        )
    return min_dist(0, 0)
```

Також у алгоритма Левенштейна є його математична формула (рисунок 3.5). Зазвичай такий алгоритм часто використовується для вирішення задач де немає точних даних, а тільки можливість їх подібності. Але потрібно зрозуміти що цей алгоритм не можу розпізнавати дані на всі сто процентів, завжди є шанс похибки. Тому якість роботи цього алгоритму на пряму залежить від кількості якісної багато варіантною інформації, яку вносить адміністратор цього чат-боту.

Алгоритм визначає “відстань” між реченнями, це значить корегується від 0 до 100, і чим більше число тим більше ці запитання похожі між собою. Також є властивість “trashehold” це чисельний показник, він відкидає всі

запитання які меньші певної “дистанції”, рекомендоване число це 85 процентів.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

Рисунок 3.5 – Математична формула алгоритму Левенштейна

Оскільки проєкт реалізований на мікро сервісній архітектурі, то в ньому є реалізація підтримки сервісів та їх ініціалізація. Як було сказано вище, мікро сервісна архітектура дозволяє розбивати функціонал проєкта на окремі сервіси, кожен з яких виконує свій функціонал і має відповідні таблиці.

В проєкті реалізовані три мікро сервіси:

- Сервіс для адміністрації і користувачів.
- Сервіс для проєктів і інформації.
- Сервіс для чат-ботів.

Для правильно реалізації сервісів, потрібно створити абстрактний сервіс, який буде мати функціонал по інтеграції його у всі можливі функції, і всі сервіси будуть успадковуватись від нього. Реалізація абстрактного сервісу зображена на лістингу 3.9.

Лістинг 3.9 – Реалізація абстрактного сервісу.

```
from sqlalchemy.orm import Session

class BaseService:
    def __init__(self, db_session: Session) -> None:
        self._db_session = db_session
```

```
    @property
    def db_session(self) -> Session:
        return self._db_session
```

Після успадкування батьківського сервісу, всі дочірні сервіси будуть мати можливість відкривати сесію до бази даних, що дає можливість взаємодіяти сервісам з інформацією, яка зберігається в базі даних.

Сервіс для користувачів і адміністрації, один з найбільших сервісів в проєкті, у якому реалізовані всі можливості CRUD систем. Ініціалізація сервісу зображена на лістингу 3.10.

Лістинг 3.10 – Ініціалізація сервісу для користувачів і адміністрації.

```
from . import BaseService

class UsersService(BaseService):
```

Функціонал даного сервісу є дуже обширним:

- Вибірка користувачів за його ім’ям (лістинг 3.11).
- Створення користувачів в базі даних (лістинг 3.12).

Лістинг 3.11 – Select функція у сервісі Users.

```
def get_by_username(
            self,
            username: str
    ) -> Optional[Users]:
        stmt = select_user_q(
        ).where(
            Users.username == username
        )
        result = self.db_session.execute(stmt)
        user = result.scalar_one_or_none()
        return user
```

Лістинг 3.12 – Створення нових користувачів у сервісі Users.

```
def create_user(
            self,
```

```
            username: str,
            password: str,
            full_name: str,
            age: Optional[int] = None
    ):
        # Close started session and start new
        self.db_session.close()
        with self.db_session.begin():
            stmt = insert_user_q(
            ).values(
                username=username,
                password=password,
                full_name=full_name,
                age=age
            )
            self.db_session.execute(stmt)
```

3.2 Типовий сценарій роботи з чат-ботом

У даному чат-боті є три сценарії роботи, кожен з цих сценаріїв має свої функції і можливості, а також свій мікро сервіс.

Головні сценарії в чат-боті:

- Сценарій створення користувачів і авторизації.
- Сценарій створення проєктів, і керування інформацією.
- Сценарій використання чат-бота.

Тестування системи відбувається через програму Postman, оскільки в проєкті не передбачений інтерфейс, то цей сервіс є REST API, для користування яким потрібно надсилати запроси на сервер. Це зроблено для полегшення інтеграції системи на сервіси. Також у фреймворку FastAPI є вбудована автогенерація документації функціоналу який реалізований, ця документація розміщена за посиланням “http://127.0.0.1:8000/docs” (рисунок 3.6).

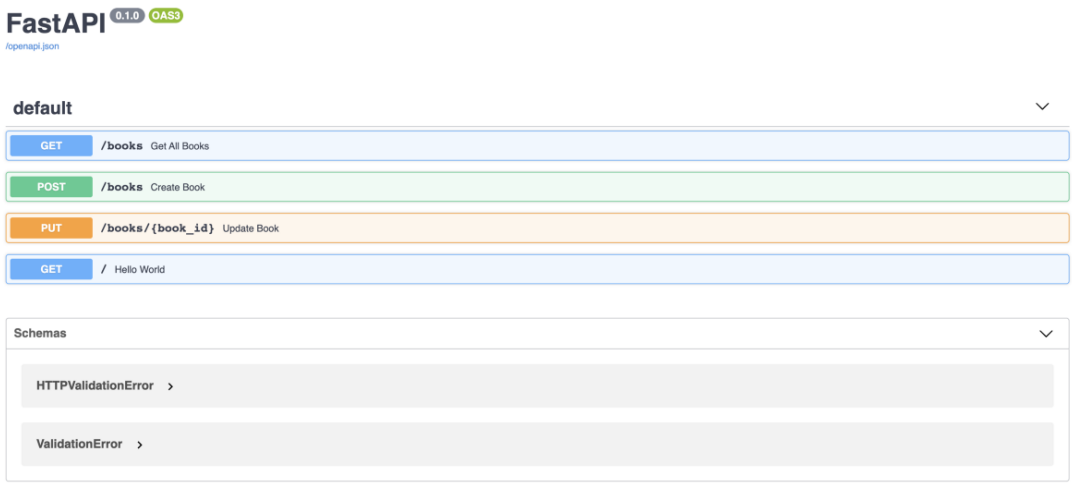

Рисунок 3.6 – Інтерфейс автозгенерованої документації Open API

Сценарій користувачів, передбачає собою дві функцій для, взаємодії з системою:

- Створення користувача – це функція реєстрації розміщена за посиланням “http://127.0.0.1:8000/users/register/”
- Авторизація користувача – це функція для авторизації користувача у системі, розміщена за посиланням “http://127.0.0.1:8000/users/login”.

Для виконування реєстрації потрібно виконати Post запит, і добавити в Body Json масив з даними (рис. 3.6). Для формування json body створена схема для валідації, де є важливі ключі і другорядні.

Розбір схеми валідації для запиту на реєстрацію, всі схеми у системі контролюються бібліотекою PyDantic. Це Python бібліотека для створення схем валідації даних, вона є вбудованою у FastAPI зі старту. Приклад схеми реєстрації зображений на рисунку 3.7.

Обов’язкові ключі схеми:

- username – тип даних str.
- password – тип даних str.
- email – тип даних str.

Не обов’язкові ключі схеми:

- age – тип даних int.
- full_name – тип даних str.

Рисунок 3.7 – Приклад JSON схеми для реєстрації.

Сценарій створення проєктів і редагування інформації передбачає собою, набір CRUD функцій для управління даними про проєкти та інформацію.

Цей сценарій передбачає собою ось такі функції:

- Створення проєкту – ця функція створення проєкту розміщена за посиланням “http://127.0.0.1:8000/projects/create”.

- Редагування проєкту – ця функція редагування проєкту розміщена з посиланням “http://127.0.0.1:8000/projects/edit”.

- Видалення проєкту – ця функція видалення проєкту розміщена за посиланням “http://127.0.0.1:8000/projects/delete”.

- Перегляд проєкту – ця функція перегляду інформації проєкту розміщена за посиланням “http://127.0.0.1:8000/projects/info”.

- Створення нової інформації – ця функція створення нової інформа-ції розміщена за посиланням “http://127.0.0.1:8000/info/create”.

- Редагування інформації – ця функція редагування інформації розміщена за посиланням “http://127.0.0.1:8000/info/edit”.

- Видалення інформації – ця функція видалення інформації розміщена за посиланням “http://127.0.0.1:8000/info/delete”.

Приклад створення нового проєкту зображений на рисунку 3.8.

```
{
    "name": "New project name",
    "description": "Very simple project. Main task is to testing my new chat-bot"
}
```


Рисунок 3.8 – Створення нового проєкту.

Приклад редагування інформації зображений на рисунку 3.9.


```
{
    "id": 1,
    "name": "New Question???"
}
```


Рисунок 3.9 – Редагування інформації

Сценарій чат-бота, передбачає всього лиш одну функцію, для надсилання питань боту. Функція приймає один параметр “question” і повертає користувачу відповідь на його питання. До функція можна звернутись за посиланням “http://127.0.0.1:8000/bot/question” і разом з цим надіслати питання у JSON Body (рисунок 3.10).

Як було сказано вище весь функціонал проєкту можна переглянути і одночасно протестувати у автозгенерованої документації.


```
{
    "project_id": 1,
    "question": "Які є спеціальності у навчальному закладі?"
}
```


Рисунок 3.10 – Надсилання питання чат-боту

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Оскільки чат-ботів зараз надзвичайно швидко розвивається, не залишилось сумнівів того що поступово наступає “ера” чат-ботів. Чат-боти стали для бізнесу, новим етапом розвитку. Вони використовуються у багатьох сферах: реклама, системи керування підприємством, допомога людині у повсякденному житті. Також чат-боти допомагають автоматизувати прості процеси у сферах бізнесу, можуть допомагати обслуговувати клієнтів, проводити аналіз витрат, а також підвищують продуктивність.

В теперішній час чат-боти можуть бути інтегрованими у любі сервіси взаємодії користувача з системою. Чат-боти є основною частиною функціоналу месенджерів, веб-сайтів. Сфера чат-ботів настільки розвинулась за останній час, що зараз на основі чат-ботів створюють інтернет магазини, системи технічної підтримки. Навіть зараз у складний час в Україні, під час війни, існують чат-боти які допомагають нашій армії, збором інформації про ворогів. Чат-боти можуть надавати життєву необхідну інформацію про зелені коридори і розсилати сповіщення про повітряні тривоги. На даний період чта-боти тільки починають захоплювати ІТ ринок України, а це означає що збільшуються робочі місця для спеціалістів в цій сфері і росте економіка держави.

Також в даний період швидкого розвитку електронних грошей, чат-боти дозволяють створювати платіжні системи на основі месенджерів, аналізувати ІТ ринок. Допомагають в пошуку роботи ІТ-фахівцям.

4.2 Розрахунок витрат на проєктування

На даний момент в Україні працює закон про встановлення мінімальної заробітної плати у розмірі 6500 грн. Це означає що працівник не можу отримувати меншу заробітну плату. Грошова оплата повинна систематично відбуватись кожного місяця. Це може бути вся сума на наприкінці поточного місяця, або в половина всередині місяця і інша в кінці.

У створенні інтегрованого чат-боту брали участь такі працівники:

- Проектувальник.
- Бекенд-програміст.
- Тестувальник.
- Проджект менеджер.

Проектувальник отримував щомісячну заробітну плату у розмірі 11500 гривень.

- Обрахунок податку на доходи фізособи: 11500 * 18% = 2070 гривень.
- Військовий збір: 11500 * 1,5% = 172 гривень.
- Один внесок: 11500 * 22% = 2530 гривень.
- Утримання: 2070 + 172 = 2242 гривень.
- До розрахунка: 11500 – 2242 = 9258 гривень.

Бекенд-програміст отримував щомісячні заробітну плату у розмірі 21000 гривень.

- Обрахунок податку на доходи фізособи: 21000 * 18% = 3780 гривень.
- Військовий збір: 21000 * 1,5% = 315 гривень.
- Один внесок: 21000 * 22% = 4620 гривень.
- Утримання: 3780 + 315 = 4095 гривень.
- До розрахунка: 21000 – 4095 = 16905 гривень.

Тестувальник отримував щомісячні заробітну плату у розмірі 15000 гривень.

- Обрахунок податку на доходи фізособи: 15000 * 18% = 2700 гривень.
- Військовий збір: 15000 * 1,5% = 225 гривень.
- Один внесок: 15000 * 22% = 3300 гривень.
- Утримання: 2700 + 225 = 2925 гривень.
- До розрахунка: 15000 – 2925 = 12075 гривень.

Проджект менеджер отримував щомісячні заробітну плату у розмірі 18500 гривень.

- Обрахунок податку на доходи фізособи: 18500 * 18% = 3330 гривень.
- Військовий збір: 18500 * 1,5% = 278 гривень.

- Один внесок: 18500 * 22% = 4070 гривень.
- Утримання: 3330 + 278 = 3608 гривень.
- До розрахунка: 18500 – 3608 = 14892 гривень.

Система матеріально винагородних розробників зведена у таблиці 4.1.

Таблиця 4.1 – Обрахунок матеріальної винагороди.

№	Посада	Оклад Грн./міс	Відрахування, Грн./міс	Кількість	Сума, грн
1	Проектувальник	11500	2070	1	9258
2	Бекенд-програміст	21000	3780	1	67620
3	Тестувальник	15000	2700	1	24150
4	Проджект менеджер	18500	3330	1	29784
		Усього заробітної плати			130812

Кошторисна вартість системи: 5784 + 19500 + 3995 = 29279 гривень.

Податок на додаткову ціну: 29279*20% = 5856 гривень.

Договірна ціна – 5856 + 29279 = 35135 гривень.

Кошторис витрат у проектуванні зведено у таблиці 4.2.

Таблиця 4.2 - Кошторис витрат у проектуванні.

Назва статей витрат	Сума, грн	Обгрунтування
Зарплата розробників	130812	
Відрахування на соціальні питання	19500	
Контрагентські функції	-	
Відрядження	-	
Інші прямі витрати	-	
Усього прямих витрат	19500	

Продовження таблиці 4.2

Накладні витрати	11711	
Планові накопичення	5856	
Загальна, вартісна ціна проекту	29279	
Податок на додаткову вартість	5856	
Загалом, договірна ціна	35135	

4.3 Обґрунтування необхідності розробки

Зважаючи на фактор збільшення популярності чат-ботів у різних сферах життя людини, створення чат-бота для Галицького коледжу допоможе вирішити ряд задач:

- Автоматизація спілкування з вступником.
- Автоматизація діяльності головного сайту.
- Показ вивчення сучасних технологій.

Оскільки ринок чат-ботів розвивається і вже скоро чат-бути будуть на всіх пристроях і в більшій половині додатків, і стануть стандартом для компанії, створення чат-бота, допоможе Галицькому коледжу привернути більше уваги, майбутніх студентів і зарекомендувати себе як професійний навчальний заклад.

ВИСНОВКИ

При реалізації дипломного проєкту, був проведений чіткий аналіз ринку технологій і розвиток чат-ботів у різних напрямках. Під час проєктування було розроблено і узгоджено чат-бота, який буде допомагати студентам і абітурієнтам у вирішенні їхніх інформаційних проблем.

Також був проведений аналіз існуючих чат-ботів навчальних закладів, визначення плюсів і мінусів даних видів ботів, на основі чого було складено план реалізації чат-бота, чітко поставлено задачі і мету даного проєкт, які в результаті розробки було успішно виконано. Був обраний чіткий набір технологій для реалізації даного чат-боту, які інтегрованої системи.

Чат-бот отримав можливості бути інтегрованим у любий сервіс, а також не бути заточеним під виконання певних задач. За допомоги асинхронного програмування чат-бот може виконувати паралельно багато функцій, що дозволяє на одній системи працювати зразу декільком чат-ботам.

Визначено вартість витрат для реалізації чат-бота командою IT-фахівців, проведений чіткий аналіз ринку чат-ботів і їхнє місце в майбутньому. З аналізу ринку було доказано стрімке наростання популярності чат-ботів у різних сферах життя людини.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1.Що таке чат-бот і як він працює? Creativesmm: веб-сайт URL: https://creativesmm.com.ua/shho-take-chatbot-ta-komu-vonu-p/ (дата звернення 09.05.2022).

2.PostgreSQL documentation. PostgreSQL: веб-сайт URL: https://www.postgresql.org/docs (дата звернення 15.05.2022).

3.PgAdmin download. PostgreSQL Admin: веб-сайт URL: https://www.pgadmin.org/docs/pgadmin4/latest/index.html (дата звернення 12.05.2022).

4.FastAPI user guide. FastAPI: веб-сайт URL: https://fastapi.tiangolo.com/tutorial/ (дата звернення 15.05.2022).

5.Python 3.10.5 documentation. Python: веб-сайт URL: https://docs.python.org/3/ (дата звернення 19.05.2022).

6.DialogFlow documentation. DialogFlow: веб-сайт URL: https://cloud.google.com/dialogflow/docs (дата звернення 23.05.2022).

7.Python NLTK documentation. Natural Language Tool Kit: веб-сайт URL: https://www.nltk.org/api/nltk.html (дата звернення 24.05.2022).

8.Python Pydantic models. Pydantic: веб-сайт URL: https://pydantic-docs.helpmanual.io/usage/models/ (дата звернення 26.05.2022).

ДОДАТКИ

Додаток А

Авторизація користувача

user.py:

```
def create_token(data: dict, expires_delta: timedelta):

    data_copy = data.copy()

    expire = datetime.utcnow() + expires_delta

    data_copy.update({"exp": expire})

    encoded = jwt.encode(data_copy,
settings.SECURITY_SECRET_KEY,
algorithm=settings.SECURITY_ALGORITHM)

    return encoded




def create_user_tokens(user: Users):

    access_token_expires =
timedelta(minutes=settings.SECURITY_ACCESS_TOKEN_EXPIRE_MINUTES)

    refresh_token_expires =
timedelta(minutes=settings.SECURITY_REFRESH_TOKEN_EXPIRE_MINUTES
)



    # Create access token

    access_token = create_token(

        data={

            "sub": user.username,

            "scopes": user.scopes

        },

        expires_delta=access_token_expires

    )

    # Create refresh token
```

```
    refresh_token = create_token(
        data={
            "sub": user.username
        },
        expires_delta=refresh_token_expires
    )

    return access_token, refresh_token


def set_refresh_token_cookie(response: Response, token: str):
    response.set_cookie(
        key=settings.SECURITY_REFRESH_TOKEN_COOKIE_KEY,
        value=token,
        expires=settings.SECURITY_REFRESH_TOKEN_COOKIE_EXPIRES,
        path=settings.SECURITY_REFRESH_TOKEN_COOKIE_PATH,
        domain=settings.SECURITY_REFRESH_TOKEN_COOKIE_DOMAIN,
httponly=settings.SECURITY_REFRESH_TOKEN_COOKIE_HTTPONLY,
        secure=settings.SECURITY_REFRESH_TOKEN_COOKIE_SECURE,
        samesite=settings.SECURITY_REFRESH_TOKEN_COOKIE_SAMESITE
    )


# Register new user
@router.post(
    "/register",
    name="users:register"
)
```

```
def register_user(
        user_data: UserRegisterSchema,
        user: UsersService = Depends(get_service(UsersService)),
):
    # Check if user with same username exist
    if user.get_by_username(user_data.username):
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="User with same username exist"
        )
    # Hash password
    hashed_password = get_password_hash(user_data.password)
    # Create new user
    user.create_user(
        user_data.username,
        hashed_password,
        user_data.full_name,
        age=user_data.age
    )
    # Create main root
    user_id = user.get_by_username(user_data.username).id
    create_main_dir(user_id)
    return {"message": "User successfully created"}


# Generate the access token
@router.post(
    "/login",
```

```
    response_model=TokenResponseSchema,
    name="users:login"
)
def login_users(
        request: Request,
        response: Response,
        form_data: OAuth2PasswordRequestForm = Depends(),
        user: UsersService = Depends(get_service(UsersService))
):
    # Check if user data is correct
    user_data = user.get_by_username(form_data.username)
    if not (user_data and verify_password(form_data.password,
user_data.password)):
        raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Bad data!")
    access_token, refresh_token = create_user_tokens(user_data)
    # Set cookie
    set_refresh_token_cookie(response, refresh_token)
    return {
        "access_token": access_token,
        "token_type": "bearer"
    }



# Refresh access token
@router.post(
    "/token/refresh",
    response_model=TokenResponseSchema,
    name="users:refresh-token"
```

```
)
def refresh_token(
        request: Request,
        response: Response,
        token: Optional[str] = Cookie(None,
alias=settings.SECURITY_REFRESH_TOKEN_COOKIE_KEY),
        user: UsersService = Depends(get_service(UsersService))
):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate refresh token"
    )
    if token is None:
        raise credentials_exception
    # Try to get username from refresh token
    try:
        payload = jwt.decode(token,
settings.SECURITY_SECRET_KEY,
algorithms=[settings.SECURITY_ALGORITHM])

        username: str = payload.get("sub")

        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception
    # Get user
    user_data = user.get_by_username(username)

    if user_data is None:
```

```
        raise credentials_exception

    access_token, refresh_token = create_user_tokens(user_data)

    set_refresh_token_cookie(response, refresh_token)

    return {
        "access_token": access_token,
        "token_type": "bearer"
    }


# Logout
@router.post(
    "/logout",
    name="users:logout"
)
def logout(
        response: Response,
        user: UserSchema = Security(get_current_user)
):
    response.delete_cookie(
        settings.SECURITY_REFRESH_TOKEN_COOKIE_KEY,
        settings.SECURITY_REFRESH_TOKEN_COOKIE_PATH,
        settings.SECURITY_REFRESH_TOKEN_COOKIE_DOMAIN
    )
    return {}
```

Додаток Б

Конфігурація проекту

config.py:

```
class Settings(BaseSettings):

    # Project

    PROJECT_VERSION: str = "0.0.1"

    PROJECT_NAME: str = "Test Task"

    PROJECT_IP: Union[IPv4Address, str] = "localhost"

    PROJECT_PORT: int = 8008

    ALLOWED_HOSTS: List[str] = ["*"]


    # Security

    SECURITY_ACCESS_TOKEN_EXPIRE_MINUTES: int = 60  # 1 hour

    SECURITY_REFRESH_TOKEN_EXPIRE_MINUTES: int = 43200  # 30
days

    SECURITY_SECRET_KEY: str =
"ffuwefundomsdngjnjfqekrmi23jiojio5u4358u3fdgjgfh"

    SECURITY_ALGORITHM: Literal["HS256"] = "HS256"

    SECURITY_ACCESS_TOKEN_URL: str = "/login"

    SECURITY_REFRESH_TOKEN_COOKIE_KEY: str = "test_task_cookie"

    SECURITY_REFRESH_TOKEN_COOKIE_EXPIRES: int = 2592000  #
seconds (30 days)

    SECURITY_REFRESH_TOKEN_URL: str = "/token/refresh"

    SECURITY_REFRESH_TOKEN_COOKIE_PATH: str =
SECURITY_REFRESH_TOKEN_URL

    SECURITY_REFRESH_TOKEN_COOKIE_DOMAIN: str = "localhost"

    SECURITY_REFRESH_TOKEN_COOKIE_HTTPONLY: bool = True

    SECURITY_REFRESH_TOKEN_COOKIE_SECURE: bool = True

    SECURITY_REFRESH_TOKEN_COOKIE_SAMESITE: Literal["lax",
"strict", "none"] = "none"
```

```
    # Database

    # DB_USER: str = "tolikdemchuk"

    # DB_HOST: str = "localhost"

    # DB_PORT: int = 5432

    # DB_PASSWORD: str = ""

    # DB_NAME: str = "test_work"

    # SQLALCHEMY_DATABASE_URI: str =
"postgresql://{}:{}@{}:{}/{}".format(

    #     DB_USER,

    #     DB_PASSWORD,

    #     DB_HOST,

    #     DB_PORT,

    #     DB_NAME

    # )


    SQLALCHEMY_DATABASE_URI: str = Field(...,
env='DATABASE_URL')




@lru_cache()

def cached_settings():

    return Settings()




settings = cached_settings()
```