math_2024_053

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Створення системи для проходження навчальних тестів з використанням телеграм-бота

Кваліфікаційна робота

Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент 4 курсу, 407 групи
Зенюк Мирослав Сергійович
Керівник:
доц. Готинчан Т.І.

До захисту допущено
на засіданні кафедри
математичного моделювання
протокол № _17_ від ____6 червня_____ 2024р.
Зав. кафедри _____ проф. Черевко І.М.

Чернівці – 2024

Анотація

У кваліфікаційній роботі описані етапи створення системи для проходження тестів, зокрема з підготовки до співбесіди на здобуття IT-професії, за допомогою мови програмування Python, убудованої бібліотеки Telebot та створення адміністративної панелі засобами мови програмування JavaScript та NodeJS та різносторонніх бібліотек, з використанням бази даних MongoDb.

Ключові слова: Python, Bot, MongoDB, GPT, Chat, Telegram, JS, HTML, NODE JS.

Abstract

The qualification work describes the stages of creating systems for passing tests, in particular, preparing for an interview for an IT profession, using the Python programming language, the built-in Telebot library, and creating the administrative panel using the JavaScript programming language and NodeJS and versatile libraries, using databases MongoDb.

Key words: Python, Bot, MongoDB, GPT, Chat, Telegram, JS, HTML, NODE JS.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____Зенюк М.С

Зміст

Вступ

Актуальність роботи. Проходження тестів може бути корисним етапом як у підготовці до співбесіди в інформаційних технологічних компаніях, так і тестувань при проходженні тем навчальних предметів закладів освіти. Багато іт-компаній використовують технічні тести для оцінки навичок програмування, алгоритмічного мислення та інших технічних аспектів. До співбесід із виявлення технічних знань та навичок треба готуватись. Тестування на допоміжних спеціалізованих ресурсах може допомогти визначити рівень технічних знань. Під час проходження тестів користувач може визначити, на які типи питань йому важко відповідати, і відповідно адаптувати свою стратегію підготовки, звертаючи більше уваги на слабкі місця.

Проведення тестів у навчальних закладах сприяє глибшому розумінню матеріалу, оскільки вимагає від студентів та учнів аналізу, висновків та застосування знань на практиці. Крім того, тести допомагають вчителям та викладачам оцінити ефективність своєї роботи та вчасно виявити проблемні моменти в навчанні, що дозволяє швидко коригувати навчальний процес. Таким чином, використання навчальних тестів є доцільним і важливим етапом у процесі освіти, сприяючи якісній підготовці студентів до подальшого життя і кар'єрного зростання. Викладач підбирає базу питань до тестів та завантажує їх за допомогою адміністративної панелі, реєструє бот у Телеграмі і надає назву студентам чи учням.

В інтернеті є багато платформ для проходження тестів. Можна скористатися вебсайтами або додатками, які надають доступ до тестових завдань. На спеціалізованих ресурсах, зокрема dou.ua, зазначаються рекомендації та питання. Останнім часом набувають популярності чат-боти, за допомогою яких можна проходити тести у меседжерах.

Практичне значення дослідження. У контексті створення телеграм-бота мовою програмування Python для підготовки до співбесід використання бази даних MongoDB та вбудованого чату GPT є ключовими елементами. Обране завдання визначає важливі аспекти технічної реалізації, оскільки потрібно забезпечити ефективну обробку даних, зручний доступ до них через MongoDB та інтерактивний чат, що використовує технологію GPT для покращення взаємодії з користувачем.

У сучасному розробницькому середовищі вибір правильних інструментів відіграє визначальну роль у вдалому втіленні поставленої задачі. Використання мови програмування Python забезпечує зручність у розробці, а база даних MongoDB стає оптимальним рішенням для збереження та обробки інформації про підготовку до співбесід. Додатковим функціоналом у спілкуванні є вбудований чат GPT, який розширює можливості взаємодії, забезпечуючи користувача інтелектуальним та природним спілкуванням.

Мета і завдання дослідження. Це дослідження має на меті створити не лише бота, але інтегровану систему, яка враховує потреби користувача під час підготовки як до співбесід, так і закріплюючи навчальний матеріал з предметів у закладах освіти, забезпечуючи зручний доступ до ресурсів бази даних та інтерактивний досвід спілкування з використанням GPT.

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, списку використаних джерел і додатків. У першому розділі описані технології, що використовуються при розробці функціоналу сервісу. Другий розділ присвячений опису постановки задачі та її програмної реалізації. У додатках зазначені частини коду, що реалізують програмний додаток.

Розділ 1 Використані технології

1.1 Python

Python – це мова програмування, яка останнім часом має широке застосування в інтернет-додатках, розробці програмного забезпечення, а також у галузях науки про дані та машинного навчання (ML). Розробники віддають перевагу Python завдяки його високій ефективності, легкості вивчення та універсальності на різних платформах. Програми, написані на Python, доступні для вільного завантаження, і сумісні з різними операційними системами, що робить їх універсальними і сприяє швидкій розробці проектів.

Python – широковідома мова і вже пройшла довгий шлях розвитку. Перша версія мови вийшла у лютому 1991 року. Поточною версією мови є версія 3.12, яка оновлюється з виходом версії Python 3.0 від 2008 року.

Мова Python унікальна завдяки таким своїм особливостям [1]:

- **Інтерпретована мова.** Python є інтерпретованою мовою, тобто вона виконує код порядково. Якщо в коді програми присутні помилки, вона перестає працювати. Це дає змогу програмістам швидко знайти помилки в коді.
- **Проста у використанні мова.** Мова програмування Python проста своєю особливістю використання слів, аналогічних англійській мові. Особливість полягає у тому, що, на відміну від багатьох інших мов програмування, у Python відсутні фігурні дужки. Натомість для визначення блоків коду використовується відступ, що робить синтаксис мови особливо зрозумілим та зручним. Ця особливість відображається у простоті вивчення та розробки коду на Python, роблячи його відмінним інструментом для програмістів у різних галузях.
- **Мова з динамічною типізацією.** Велика зручність при програмуванні мовою Python полягає у тому, що розробники не зобов'язані передбачати типи змінних при написанні коду. Система типізації, яку використовує

Python, визначає типи змінних під час виконання програми. Ця особливість значно спрощує процес розробки, оскільки програмістам не потрібно витрачати час на явне оголошення типів. Замість цього вони можуть фокусуватися на самому коді, що призводить до швидшого написання програм і покращення продуктивності.

- **Мова високого рівня.** Мова програмування Python вражає своєю природністю, що робить її ближчою до мови розмови, ніж багато інших мов програмування. Це особливо відчутно в тому, що програмістам не потрібно пильно слідкувати за деталями базової функціональності, такими як архітектура та керування пам'яттю. Гнучкість та автоматизація в Python дозволяють розробникам фокусуватися на вираженні своїх ідей і завдань, не докладаючи надмірних зусиль для керування технічними деталями. Це робить мову особливою для швидкого та ефективного написання коду, сприяючи тим самим збільшенню продуктивності розробників.
- **Об'єктно-орієнтована мова.** Python визначається тим, що розглядає усі елементи як об'єкти, проте це не обмежує його сферу застосування лише об'єктно-орієнтованим програмуванням. Одна з сильних сторін Python – його здатність підтримувати різні парадигми програмування, такі як структурне та функціональне програмування.

У Python можна використовувати структурні підходи для організації коду, де акцент робиться на функціях і процедурах, а не об'єктах. Це робить мову гнучкою та адаптованою до різних стилів програмування у залежності від вимог конкретного завдання.

Функціональне програмування також може бути застосоване в Python із використанням функцій вищого порядку, анонімних функцій (lambda-функцій) та інших конструкцій. Це робить Python потужним інструментом для вибору оптимального підходу в залежності від задачі та власних уподобань розробника.

1.2 MongoDb

MongoDB – система керування базами даних, яка працює з документоорієнтованою моделлю даних. На відміну від реляційних СКБД, MongoDB не потрібні таблиці, схеми або окрема мова запитів. Інформація зберігається у вигляді документів або колекцій.

Розробники позиціонують продукт як проміжну ланку між класичними СКБД і NoSQL. MongoDB не використовує схеми, як це роблять реляційні бази даних, що підвищує продуктивність усієї системи.

MongoDB належить до класу NoSQL СКБД і працює з документами, а не із записами. Це кросплатформний продукт, який легко впроваджується у будь-яку операційну систему. Низка унікальних особливостей дає змогу використовувати СКБД під певні завдання, у яких вона забезпечує максимальну продуктивність і надійність [2].

1.3 BotFather

BotFather – це винятковий інструмент у Telegram, який надає користувачам можливість створювати та налаштовувати свої власні чат-боти. Завдяки BotFather створення ботів стає легким та інтуїтивно зрозумілим процесом. Користувачі можуть не лише визначати основні параметри та функціональність свого бота, але й отримувати доступ до різноманітних додаткових функцій та можливостей, які розширюють функціонал їхніх ботів.

BotFather дозволяє власникам ботів індивідуалізувати їх відповідно до конкретних потреб та вимог. Зручний інтерфейс сприяє швидкому створенню та налаштуванню ботів, роблячи цей процес приємним та ефективним.

Основні можливості [3]:

- **Створення ботів**. BotFather відкриває перед користувачами можливість створювати нові боти, які мають своє унікальне ім'я та ідентифікатор. Це дозволяє кожному боту мати індивідуальну ідентичність та чітко визначені параметри. Крім того, користувачі можуть обирати тип свого

бота, залежно від призначення, яке вони мають на увазі. Наприклад, користувачі можуть створити бота для розсилки повідомлень, який спрощуватиме комунікацію зі своїми підписниками. Іншою опцією може бути створення бота для створення опитувань, що дозволить здійснювати збір відгуків чи думок від користувачів. Також може бути обрано бота для обробки замовлень, що надасть можливість автоматизовано обробляти та виконувати різноманітні завдання. Такий гнучкий підхід дозволяє користувачам BotFather адаптувати свого бота до конкретних потреб і завдань, роблячи його більш ефективним та спрямованим.

- **Налаштування ботів.** Після успішного створення бота за допомогою BotFather, користувачам відкривається можливість повністю налаштувати його функції та параметри. BotFather дозволяє здійснити ряд налаштувань, щоб забезпечити боту оптимальне функціонування. Декілька з можливих налаштувань включають:
- **Автоматичні відповіді:** Користувачі можуть визначати автоматичні відповіді, які бот висилатиме на певні команди чи запитання.
- **Генерація ключів API:** BotFather надає можливість генерувати ключі API, які можуть використовуватися для інтеграції бота з іншими сервісами чи додатками.
- **Визначення прав доступу користувачів:** Користувачі можуть контролювати, які дії та функції доступні різним категоріям користувачів, визначаючи права доступу.

Ці налаштування дозволяють користувачам BotFather адаптувати їхнього бота до конкретних потреб, забезпечуючи йому високий рівень персоналізації та відповідність конкретним вимогам.

- **Відкрите API.** BotFather гнучкий та розширюється завдяки наявності відкритого API. Це надає розробникам унікальну можливість створювати нові функції та інтегрувати їх з іншими сервісами чи інструментами. Відкрите API BotFather створює простір для творчості та інновацій,

розширюючи можливості використання чат-ботів у різних сценаріях. Розробники можуть створювати різноманітні розширення для ботів, включаючи нові команди, функції взаємодії з користувачами, або навіть інтеграцію з іншими платформами та сервісами. Це забезпечує більш гнучку інтеграцію та розширює функціональні можливості чат-ботів, роблячи їх більш потужними та адаптованими до різноманітних завдань. Відкритість API BotFather сприяє спільноті розробників, що спільно вдосконалюють та розширюють можливості цього інструменту, роблячи його ще більш цінним для широкого кола користувачів.

1.4 Бібліотека Telobot

Telebot – це бібліотека Python, яка дозволяє розробникам створювати та налаштовувати Telegram-ботів за допомогою Telegram API. Тут представлено два аспекти Telebot як технології.

Позитивні аспекти:

- *простота використання*: Telebot простий у використанні, зручний і зрозумілий. Документація дуже добре написана, а списки функцій і варіантів використання полегшують процес розробки Telebot.
- *широка функціональність*: Telebot має велику кількість функцій, які надають розробникам багато можливостей для розробки Telegram-ботів. Бібліотека підтримує надсилання повідомлень, медіафайлів, голосових повідомлень, стікерів, кнопок, клавіатур, опитувань, груп і каналів, зображень, аудіо, відео та документів.
- *безплатна і відкритий вихідний код*: Оскільки Telebot має відкритий вихідний код, розробники можуть вносити зміни до бібліотеки та адаптувати її до своїх потреб. Крім того, оскільки бібліотека безплатна, розробники не повинні платити роялті.

Негативні аспекти:

- *обмежена функціональність*: Хоча Telebot має багато можливостей, бібліотека поки що не пропонує усю функціональність, доступну в Telegram API. Це може бути обмеженням для розробників, яким потрібна додаткова функціональність.
- *відсутня підтримка асинхронного коду*: Оскільки Telebot базується на бібліотеці запитів, він не підходить для асинхронного коду. Це може бути проблемою для розробників, які працюють з асинхронним кодом і потребують більш ефективного та швидкого рішення.
- *проблеми зі стабільністю:* Хоча Telebot добре задокументований і має багато корисних функцій, можуть виникнути проблеми зі стабільністю, особливо якщо Telegram API змінюється. Це може спричинити проблеми з функціональністю бота.
- *відсутність підтримки віддаленої роботи з базами даних*: Телебот не має убудованого функціоналу для маніпулювання базами даних. Це може стати проблемою для розробників, які хочуть зберігати дані бота та маніпулювати ними віддалено, зокрема на серверах.
- *відсутність документації*: Хоча документація Telebot добре написана, вона не надає повної інформації про всі можливості та функції бібліотеки. Це може бути проблемою для розробників, яким потрібна детальна інформація для розуміння і використання певних можливостей Telebot.

Підсумовуючи, можна сказати, що Telebot – це корисна та зручна бібліотека Python для розробки Telegram-ботів, яка має свої переваги та недоліки. Рекомендується, перш ніж використовувати Telebot, розробникам ретельно проаналізувати свої потреби та вимоги і зробити усвідомлений вибір

1.5 Chat GPT

Chat GPT – це система чат-ботів, яка вміє обговорювати і видавати відповіді на запити, ніби жива людина. Також вона індивідуально підбирає стиль спілкування, виходячи із запитань.

У листопаді 2022 року світ побачив універсального чат-бота, який вразив глобальну аудиторію. Його створено, використовуючи потужний суперкомп'ютер Azure AI, а мовною моделлю виступає технологія від OpenAI – GPT-3.5.

Цей чат-бот пройшов цікавий процес навчання, використовуючи систему взаємодії з людьми та обширний обсяг текстової інформації. Розробники вдосконалювали його шляхом багаторазового переосмислення та перенавчання, використовуючи відповіді самого чат-бота. Цей підхід дозволив досягти високої якості генерації тексту та забезпечити більш "людський" та доступний для взаємодії інтерфейс.

Chat GPT не обмежується лише написанням тексту. Він знаходить застосування у різних областях, таких як фінансовий аналіз, генерація коду, прогнозування, резюме технічних статей, створення персональних порад та формулювання етичних відповідей. Його можна успішно використовувати у великому спектрі завдань, і його застосування продовжує розширюватися. Під час взаємодії з користувачем, навіть у тривалих діалогах, технологія зберігає деталі спілкування, що додає елементи персоналізації.

У майбутньому планується розробка інтерфейсу чат-бота на основі прикладного програмування. Це відкриє нові можливості для розробників, які зможуть легко впроваджувати чат-бота як частину свого додатку або вбудовувати його в структуру свого вебсайту [4].

На сьогоднішній день відомі такі версії ChatGPT як GPT-3, GPT-4, і GPT-Neo. Кожна наступна версія принесла із собою значні покращення у якості генерації тексту, розумінні контексту та здатності взаємодіяти з користувачами на більш глибокому рівні. Вони стали важливим інструментом у багатьох галузях, включаючи дослідження, освіту, медіа, технології та бізнес [5].

1.6 Бібліотеки Python

Бібліотека – це набір корисних функцій та інструментів, який значно полегшує роботу розробників, позбавляючи їх необхідності писати код на Python з нуля.

На сьогодні доступно понад 137 тис. бібліотек, кожна з яких відіграє важливу роль у обробці даних та зображень, візуалізації, створенні програм, машинному навчанні та інших областях [6].

Бібліотека OpenAI Python надає зручний доступ до OpenAI REST API з будь-якого додатку на Python 3.7+. Вона містить визначення типів для усіх параметрів запиту та полів відповіді і пропонує як синхронні, так і асинхронні клієнти на основі

Бібліоткека PyMongo містить інструменти для взаємодії з базою даних MongoDB з мови Python. Пакет pymongo – це власний драйвер Python для MongoDB.

Бібліотека Telebot – це проста та зручна бібліотека для створення Telegram ботів на Python. Вона дозволяє швидко розробляти прості боти, але має обмежений функціонал порівняно з іншими бібліотеками. Бібліотека Telebot для телеграм-ботів на Python надає низку основних функцій: TeleBot(token) – конструктор класу, що приймає токен вашого бота; send_message(chat_id, text) – відправляє повідомлення користувачеві із зазначеним chatid .

1.7 Visual Studio Code

Visual Studio Code – це безплатний, легкий, але потужний редактор вихідного коду, який працює на робочому столі та в Інтернеті і доступний для операційних систем Windows, macOS, Linux та Raspberry Pi OS. Він має вбудовану підтримку JavaScript, TypeScript та Node.js, а також багату екосистему розширень для інших мов програмування (наприклад, C++, C#, Java, Python, PHP та Go), середовищ виконання (наприклад, .NET та Unity),

середовищ (наприклад, Docker та Kubernetes) та хмар (наприклад, Amazon Web Services, Microsoft Azure та Google Cloud Platform) [7].

Окрім легкого та швидкого запуску, Visual Studio Code має функцію завершення коду IntelliSense для змінних, методів та імпортованих модулів; графічне налагодження; лінкування, багатокурсорне редагування, підказки параметрів та інші потужні функції редагування; зручну навігацію та рефакторинг коду; а також убудований контроль вихідного коду, включаючи підтримку Git. Багато з цього було адаптовано з технології Visual Studio.

Підтримка різних мов програмування та їх розширень варіюється від простого підсвічування синтаксису та узгодження дужок до налагодження та рефакторингу.

Останнім часом усе більше розробників віддають перевагу Visual Studio Code у розробці різноманітних додатків і програм.

1.8 JavaScript

JavaScript – це мова сценаріїв або програмування, яка дозволяє реалізовувати складні функції інтерактивності елементів на вебсторінках. Саме з JavaScript співпрацюють браузери при відображенні вебсторінок.

JavaScript є досить поширеною мовою програмування, використовується для розробки різних додатків [8]:

- вебдодатків;
- графічних на основі інтерфейсу;
- меню, що випадають;
- віджетів чату;
- каруселі зображень;
- ігри для браузерів;
- інші бізнес-програми.

Кожного разу, коли вебсторінка робить щось більше, ніж просто відображає статичну інформацію для перегляду, – відображає своєчасні оновлення контенту, інтерактивні карти, анімовану 2D/3D графіку тощо – це означає, що JavaScript, ймовірно, задіяний. Це третій кит стандартних веб-технологій, два з яких HTML і CSS.

Мова JavaScript унікальна завдяки своїм особливостям, які роблять її незамінним інструментом для сучасної веб-розробки:

- **Динамічна типізація**: JavaScript не вимагає явного визначення типів змінних, що дозволяє розробникам працювати більш гнучко та швидко.
- **Функціональна мова програмування**: JavaScript підтримує функціональне програмування, що дозволяє створювати елегантні та масштабовані рішення.
- **Асинхронний код**: Завдяки принципам асинхронного програмування, JavaScript дозволяє створювати динамічні веб-додатки, які взаємодіють з користувачем без перезавантаження сторінки.
- **Багатоплатформовість**: JavaScript може виконуватися не лише у веб-браузерах, але й на серверній стороні за допомогою платформи Node.js, що розширює його можливості.
- **Активна спільнота**: Широка та активна спільнота розробників JavaScript постійно розробляє нові бібліотеки, фреймворки та інструменти, що робить мову завжди актуальною та конкурентоспроможною.

JavaScript – це не лише мова програмування, це екосистема, що постійно розвивається та пристосовується до потреб сучасної веб-розробки.

1.9 Бібліотеки JavaScript

Розглянемо бібліотеки JavaScript, які використовуються при розробці проєкту, описаному в цій роботі [9].

Бібліотека "Split": split() розбиває заданий рядок на впорядкований список підрядків, поміщає ці підрядки у масив і повертає два масиви: масив

підрядків і масив роздільників**.** Поділ виконується шляхом пошуку шаблону, який передається другим параметром у виклику методу.

Бібліотека "fs": Модуль файлової системи js допомагає нам зберігати, отримувати доступ та керувати даними в нашій операційній системі. Найчастіше використовувані функції модуля fs включають fs. readFile для читання даних з файлу, fs. writeFile для запису даних у файл і заміни файлу fs, якщо він вже існує.

Бібліотека "multer": Multer у Node.js – це проміжне програмне забезпечення, яке використовується для легкої обробки багаточастинкових/формових даних, які використовуються при завантаженні файлів. Щоб досягти максимальної ефективності Multer був побудований на вершині busboy модуля node.js, який використовується для обробки вхідних даних HTML-форм у запитах.

Бібліотека "body-parser": Парсер тіла аналізує тіло HTTP-запиту, що зазвичай допомагає, коли потрібно знати більше, ніж просто URL-адресу, за якою виконується запит. Зокрема, в контексті POST, PATCH або PUT HTTP-запитів, де потрібна інформація міститься в тілі. Використання аналізатора тіла дозволяє отримати доступ до req.

1.10 Node JS

Node.js – це потужна технологія, яка змінює підхід до розробки веб-додатків. Вона базується на JavaScript і дозволяє використовувати цю мову програмування для серверної сторони, що відкриває нові можливості для розробників [10].

Однією з ключових переваг Node.js є його асинхронна природа. Замість традиційної синхронної моделі виконання, де кожен запит блокує інші, Node.js використовує неблокуючий ввід/вивід. Це означає, що він може обробляти багато запитів одночасно без затримок, що призводить до більшої продуктивності та швидкодії вебдодатків.

Node.js також зручно використовувати для створення реального часу (real-time) додатків, таких як чати або онлайн-ігри. Його ефективність у роботі з багатьма одночасними підключеннями робить його ідеальним для сучасних веб-застосунків, де важлива реактивність та низька затримка.

Крім того, Node.js побудований на рушії V8 від Google, що робить його дуже швидким у виконанні JavaScript коду. Це дозволяє розробникам забезпечувати ефективну роботу великих та складних веб-проектів.

Отже, використання Node.js розширює можливості розробників, забезпечуючи їм ефективний інструмент для створення швидких, масштабованих та високопродуктивних веб-додатків.

Розділ 2. Програмна реалізація

2.1 Постановка задачі

Завдання кваліфікаційного дослідження полягає у створенні інтегрованої системи, яка надає зручні та ефективні інструменти для закріплення вивченого матеріалу, зокрема, при підготовці до співбесід в інформаційно-технологічних компаніях. Виходячи з актуальності роботи, спеціальна увага приділятиметься проходженню тестів, які можуть бути корисним етапом як у підготовці до співбесід, оцінки технічних знань та навичок програмування, так і для закріплення навчального матеріалу у закладах освіти.

У сучасному інтернет-середовищі існує багато платформ для проходження тестів, які надають доступ до різноманітних тестових завдань. Однак, враховуючи широкий спектр компаній та їх вимоги до кандидатів, створення власного інструменту для підготовки може бути важливим аспектом. У такому контексті, запропоноване дослідження буде спрямоване на розробку системи, яка складається з телеграм-боту, що використовує мову програмування Python та базу даних MongoDB для зберігання та обробки інформації, та і адміністративної панелі, за допомогою якої наповнюється база даних тестовими питаннями.

Отже, основною метою дослідження є створення інтегрованої системи, яка забезпечує зручний доступ до тестових завдань, рекомендацій та інших матеріалів, необхідних для ефективної підготовки. Для досягнення цієї мети, планується застосування інтерактивного чату, що використовує технологію GPT для покращення взаємодії з користувачем як помічника з підготовки до майбутнього тестування, та бази даних MongoDB для зберігання та організації інформації.

Основні завдання дослідження включають:

1. Розробку архітектури та функціоналу телеграм-бота, який надає доступ до тестових завдань та рекомендацій для підготовки до співбесід.

2. Інтеграцію бази даних MongoDB для зберігання та обробки інформації про підготовку до співбесід.
3. Використання технології GPT для створення інтерактивного чату, що допомагає користувачам у проходженні тестів та отриманні рекомендацій.
4. Створення адмін панелі для CRUD операцій із тестовими завданнями.
5. Передбачити завантаження тестових завдань із файлу.

Це дослідження також має практичне значення, оскільки створений програмний продукт може бути використаний як інструмент для навчання та підготовки кандидатів до співбесід. Додатково, він може бути корисним для викладачів у навчальних закладах для підготовки студентів до співбесід та оцінки їхніх технічних знань та навичок програмування.

2.2 Створення макетів системи

Для настільного застосунку був розроблений макет, який є максимально зручним для користувача та приємний для очей. Макет настільного застосунку розділений на 3 частини, див. рис. 2.1.

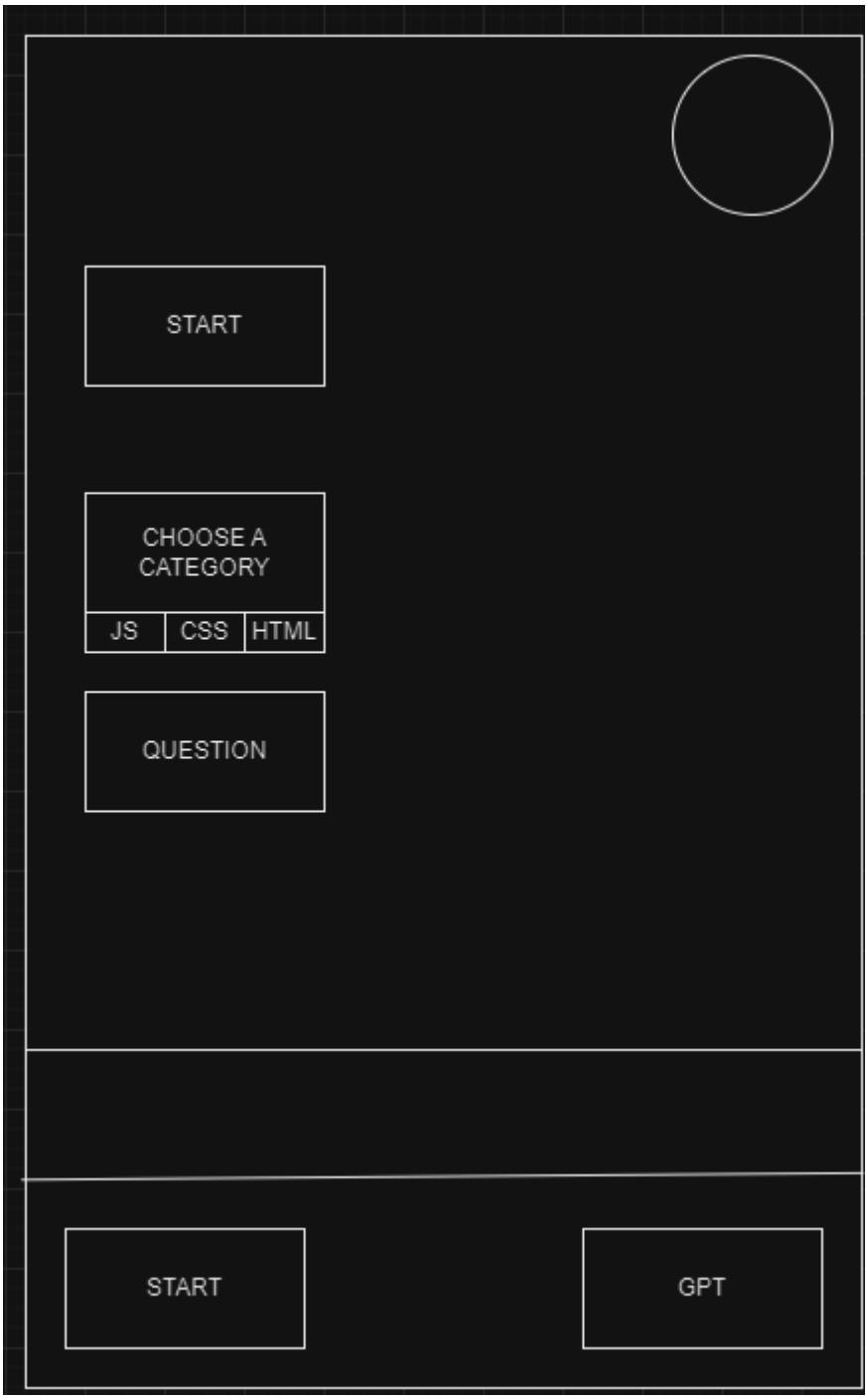

Рис. 2.1 Макет телеграм-бота (Вікторина)

У лівій частині розташована кнопка старту вікторини, праворуч – кнопка старту спілкування з чатом GPT. У центральній частині макету – місце, де виводиться весь текст спілкування користувача з чатом, а також вікторина при виборі пункту Вікторини. На рисунку 2.2 видно макет чату з GPT-ботом.

Рис. 2.2 Макет телеграм-бота (чат GPT)

Для адміністративної панелі також був створений макет, який є максимально зручним для адміністратора та приємним для очей. Макет настільного застосунку, див. рис. 2.3.

Рис.2.3 Макет адмін панелі

У центральній частині макету – місце, де показуються усі питання, є форма для додавання одного елементу, а також кнопка для завантаження файлів. Також у верхній частині є пошукове поле вводу для швидкого пошуку необхідного питання тесту для здійснення CRUD операцій над ним.

2.3 Заповнення бази даних питаннями

Для створення системи навчальних тестів необхідно спочатку підготувати дані, на яких будуватимуться питання для тесту.

Для цього була створена база даних MongoDb з назвою QuizBot і колекціями питань, які були взяті з відкритих джерел як найбільш поширеніші питання про підготовку до співбесіди на вакансію веброзробника. Питання відносяться до різних тем і розкиданні в рандомному порядку. Теми питань: HTML, CSS, JS, Frameworks. Аналогічно можна сформувати банк питань до тестів за будь-якою тематикою.

Уся інформація про питання збережена у колекції. Приклад формату даних зображено на рис. 2.4.

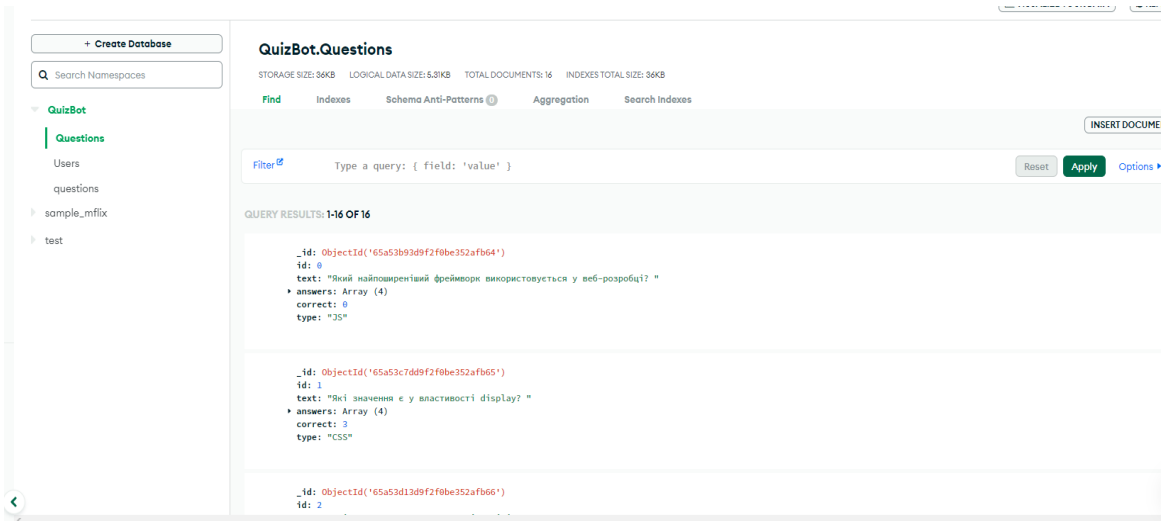

Рис. 2.4 Формат збережених даних

Як ми можемо бачити, структура об’єкта питання доволі проста, це: id самого питання, текст питання, масив відповідей, правильна відповідь і тип. до якого відноситься указане питання. Відповідь зберігається у вигляді індексу з

масиву, а у кінцевому форматі, вона показується користувачу за допомогою відповідного емоджі, як правильна чи хибна.

Для отримування необхідної інформації було використано такі сайти, як https://dou.ua/, https://foxminded.ua.

2.4 Створення телеграм-бота

Бот був створений завдяки батьківському телеграм-боту BotFather. Через API ключ, який був виданий завдяки цьому боту, ми зв'язались з ним у нашому коді. Цей бот надає можливість створювати порожні боти, які в подальшому можна використовувати у своїх цілях. BotFather розуміє декілька мов, дає можливість створити безліч ботів, має різні команди, такі як збереження API ключів, налаштування ботів і так далі.

Телеграм-бот BotFather поданий на рис. 2.5

Рис. 2.5 Інтерфейс телеграм-бота (бібліотеки)

Також узятий API ключ доступу для чату GPT на офіційному сайті, для підключення GPT у чат. Крім того, для цього використовувалась бібліотека OpenAI, про яку було описано у розділі 1.

Ключ GPT чату з офіційного сайту наведено на рис. 2.6.

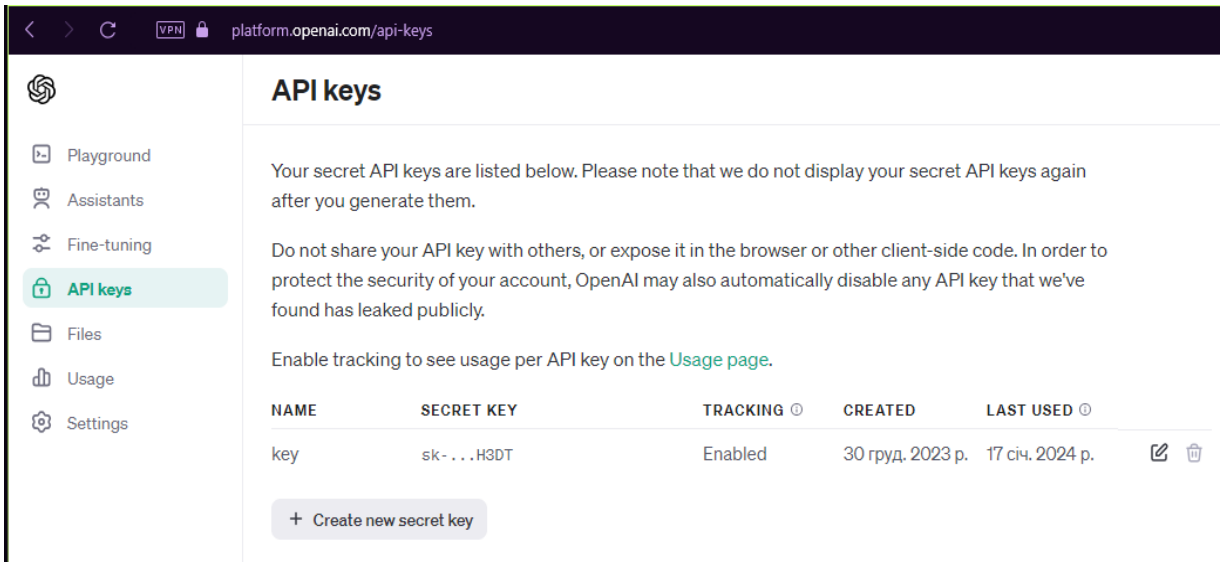

Рис. 2.6 Api-ключі для чату GPT

Для початку був створений клас звернення до бази даних. Звернення до самої бази даних, витяг усіх колекцій, витяг питань з колекції, і їхня рандоматизація, для того щоб кожного разу питання були в рандомному порядку.

Далі був створений інтерфейс самого бота, після чого були написанні функції, які вже відповідають за основний додаток бота. Це такі функції як: вітання бота, вибір опції користувачем, окрема функція на початок роботи чату і спілкування з ним, функція на відповіді на питання, перевірки чи тест був останній, чи потрібно перемикатись далі чи давати користувачеві результат, і так результат підраховується по кількості правильних відповідей на питання, з подальшим виводом відсотків правильних питань.

При розробці телеграм-боту та зв’язку з GPT чатом були написанні такі функції (див. Додаток А):

1) **Database.__init__(self)**: ініціалізує об'єкт бази даних, встановлює з'єднання з MongoDB, отримує колекції користувачів та питань, перемішує список всіх питань.
2) **Database.get_test_user(self, chat_id)**: повертає дані тестового користувача за його chat_id. Якщо користувача не знайдено, створює нового тестового користувача.

3) **Database.test_set_test_user(self, chat_id, update)**: оновлює дані тестового користувача за його chat_id.
4) **Database.get_questions(self, question_type)**: повертає список питань відповідного типу (JS, CSS, HTML або всі питання) з бази даних.
5) **start_handler(message)**: обробник команди **/start**. Надсилає привітальне повідомлення та головне меню користувачу.
6) **start_quiz_handler(message)**: обробник початку вікторини. Починає тестування та надсилає запит на вибір категорії питань.
7) **chat_handler(message)**: обробник вибору опції "Спілкування з GPT". Перевіряє, чи користувач не проходить тестування, та надсилає повідомлення про вибір режиму спілкування.
8) **select_type_handler(message)**: обробник вибору типу питань для тестування. Перевіряє, чи користувач проходить тест, та надсилає перше питання.
9) **menu_handler(message)**: обробник вибору опції "Меню". Надсилає головне меню користувачу.
10) **message_handler(message)**: обробник текстових повідомлень. Використовує GPT для генерації відповіді на користувацький запит.
11) **end_quiz_callback(query)**: обробник завершення тестування. Зупиняє тестування та надсилає повідомлення з підсумками.
12) **answered_handler(query)**: обробник відповідей на питання тесту. Зберігає відповідь користувача та надсилає відповідне повідомлення.
13) **next_handler(query)**: обробник переходу до наступного питання в тесті. Перевіряє, чи користувач пройшов усі питання та надсилає відповідне повідомлення.
14) **get_quiz_message(test_user, questions)**: підготовує повідомлення з питанням для тестування.
15) **get_answer_message(test_user, questions)**: підготовує повідомлення з відповідями на питання для тестування.

2.5 Створення адмін панелі

Сама реалізація зроблена за допомогою мови розмітки HTML, таблиці стилів CSS, а також мови програмування JavaScript. Створено форму додавання питань до колекції, а також можливість додавання файлу з питаннями за допомогою кнопки. Нижче наведений список усіх питань, наявних у цій колекції. Для зручності виведення додана пагінація на сторінці. Також побудовано чотири запити до колекції: перший запит post, який дозволяє нам записувати нові питання до нашої колекції, а інший запит get, який дозволяє отримати усі наші питання і вивести їх. Наступних 2 методи дозволяють адміністратору видаляти та редагувати питання, на самій адміністративній панелі, що є швидко та зручно. Перед цим були встановлені пакети, за допомогою яких ми змогли приєднатися до нашої бази даних і робити відповідні запити. Сам ж вивід питань здійснюється за допомогою методів обходу об'єкта: ми виводимо усю нашу інформацію за допомогою html тегів на нашу сторінку і відображаємо усе. Також була додана можливість пошуку по сторінці для кращого орієнтування на ній, для швидкого пошуку того чи іншого запитання. Користувач також має можливість на вибір завантажувати питання по одному за допомогою форми на сторінці або ж завантажити .txt файл, в якому зберігаються усі питання, які користувач хоче завантажити до бази даних.

При розробці адміністративної частини системи були написанні такі функції (див. Додаток Б):

1) **const express = require("express")**: підключення модуля Express для створення веб-сервера.
2) **const app = express()**: створення екземпляру додатку Express.
3) **const MongoClient = require("mongodb").MongoClient**: підключення MongoDB для взаємодії з базою даних MongoDB.
4) **const mongoose = require("mongoose")**: підключення Mongoose для моделювання даних та спільної роботи з MongoDB.

5) **const bodyParser = require("body-parser")**: підключення bodyParser для обробки тіла запиту.
6) **const ObjectId = require("mongodb").ObjectId**: підключення ObjectId для створення об'єктів ObjectId MongoDB.
7) **const multer = require("multer")**: підключення multer для обробки файлових завантажень.
8) **const upload = multer({ dest: 'uploads/' })**: створення middleware для завантаження файлів у папку "uploads/".
9) **const fs = require("fs")**: підключення модуля fs для роботи з файловою системою.
10) **const split = require('split')**: підключення split для розбиття потоку на рядки.
11) **mongoose.connect()**: підключення до бази даних MongoDB за допомогою Mongoose.
12) **app.use(bodyParser.json())** та **app.use(bodyParser.urlencoded({ extended: true }))**: використання bodyParser для обробки JSON та URL-кодованих даних.
13) **app.get("/")**: маршрут для обробки GET-запитів до кореневого шляху, який надсилає статичний файл index.html.
14) **app.post("/")**: маршрут для обробки POST-запитів до кореневого шляху, який зберігає дані форми у базі даних.
15) **app.get("/getData")**: маршрут для отримання всіх даних з бази даних у форматі JSON.
16) **app.delete("/:id")**: маршрут для видалення документа з бази даних за його ID.
17) **app.put("/:id")**: маршрут для оновлення документа в базі даних за його ID.
18) **app.post('/upload')**: маршрут для завантаження файлу з питаннями, розбиття його на об'єкти та збереження цих об'єктів у базі даних.
19) **app.listen(3000)**: запуск веб-сервера на порту 3000.

20) **escapeHtml(html)**: функція для екранування HTML-сутностей у тексті.
21) **searchItems(searchTerm)**: функція для пошуку елементів на сторінці за введеним терміном пошуку.
22) **handleSearch()**: функція для обробки подання форми пошуку.
23) **displayItemsWithPagination()**: функція для відображення елементів з пагінацією.
24) **handlePagination(direction)**: функція для керування пагінацією вперед або назад.
25) **fetch('/getData')**: запит для отримання даних з сервера.
26) **deleteItem(id)**: функція для видалення елемента за його ID.
27) **editItem(id)**: функція для відображення форми редагування елемента за його ID.
28) **submitEditForm(id)**: функція для відправлення відредагованих даних елемента на сервер.

2.6 Інструкція з використання

При запуску бота, користувач бачитиме повідомлення з привітанням та можливими діями у боті. Є можливість вибрати проходження вікторини або ж спілкування з чатом GPT. На рисунку 2.7 відображене початкове меню бота.

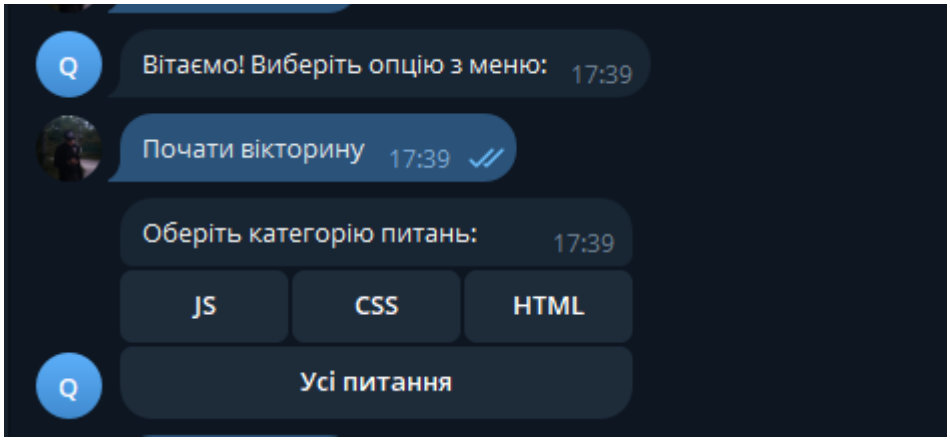

Рис.2.7 Телеграм-бот з початковим меню

При виборі вікторини користувачу буде надана можливість вибрати категорію питання у тесті. Також на рисунку 2.8 наведено приклад питання, з відповідями, а також можливість завершити тест у будь-який момент часу.

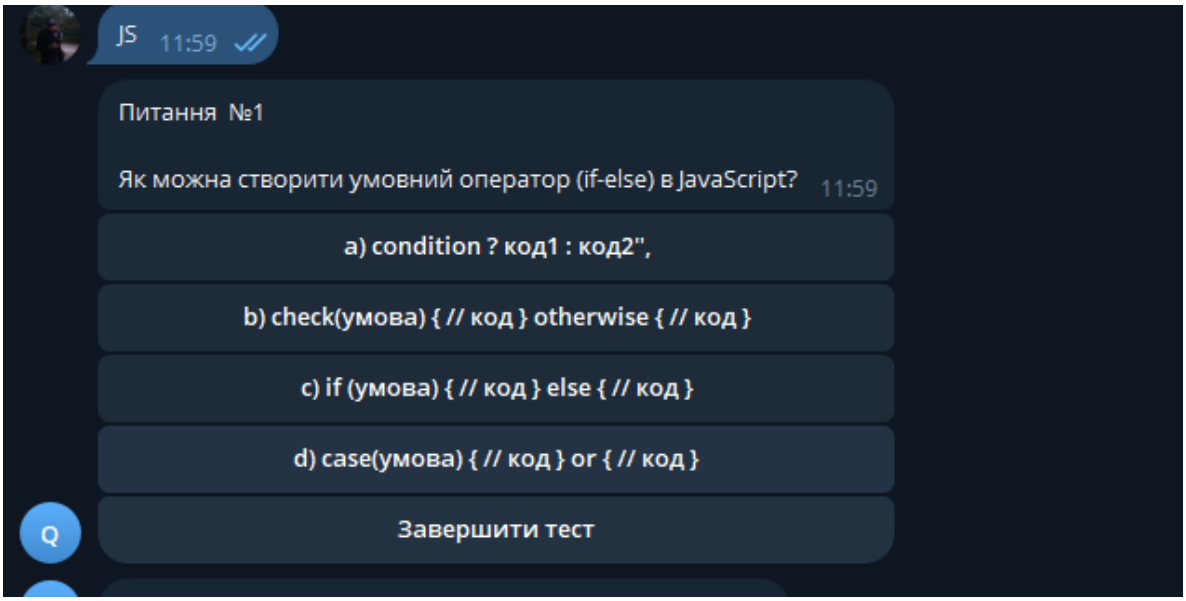

Рис.2.8 Вигляд питання тесту

При проходженні тесту і вибору відповідей, користувач одразу ж бачитиме вибір свого варіанту, і результат – відповідь була правильною чи хибною. На рисунку 2.9 показано як відображатиметься відповідь для користувача. Варто зауважити, що на малюнку також перед показаним питанням можемо бачити, як виглядає спілкування з чатом GPT. Якщо користувач вибрав опцію з спілкування з чатом, було виведено повідомлення про вхід у режим спілкування, після чого очікується питання від користувача, і врешті чого буде видана відповідь від чату GPT.

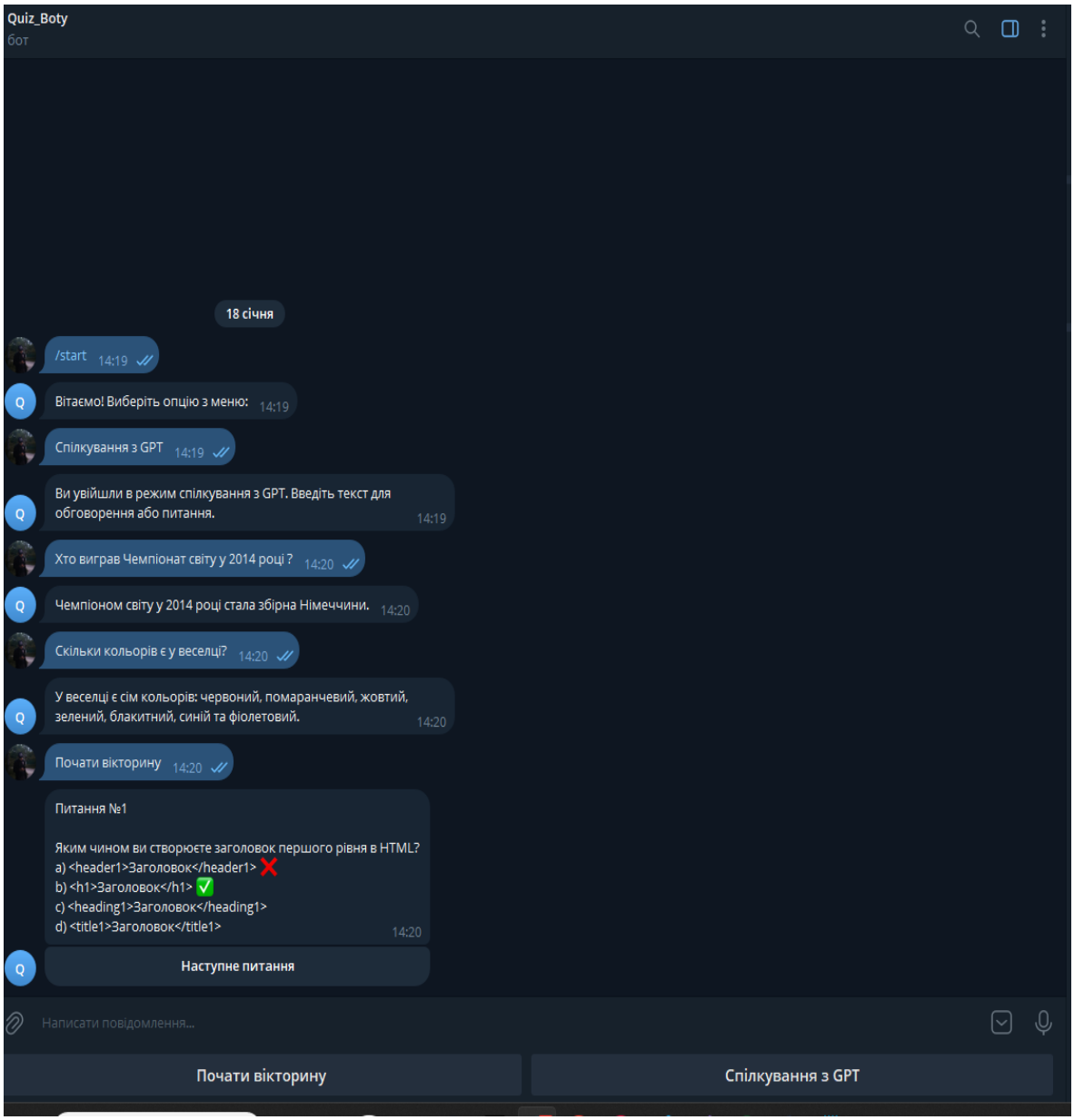

Рис.2.9 Приклади тестів у боті

По завершенню тесту буде показано кількість правильних і хибних відповідей, а також час, за який пройдений тест. Це продемонстровано на рисунку 2.10.

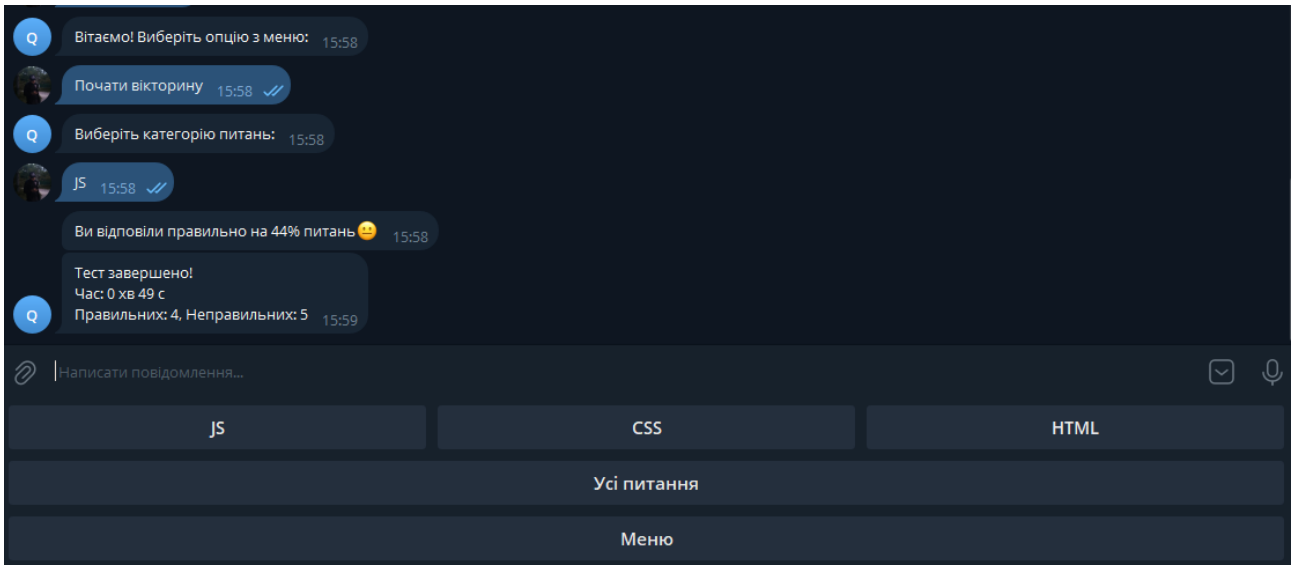

Рис.2.10 Результат тестування

Щодо користування панеллю адміністратора, є можливість пошуку для швидкого знаходження того чи іншого питання. Головними елементами на сторінці є форма додавання питання і кнопка завантаження файлу. Користування формою передбачає заповнення усіх потрібних полів форми і відправку питання по кліку на кнопку. Щодо завантаження файлу питань, вони мають бути у певному форматі. Який саме це має бути формат, подано на сторінці у вигляді прикладу (див. рисунок 2.13), а сам файл має бути в форматі .txt, що є максимально просто і зручно. На рисунку 2.11 зображені форма для відправки і кнопка завантаження файлів.

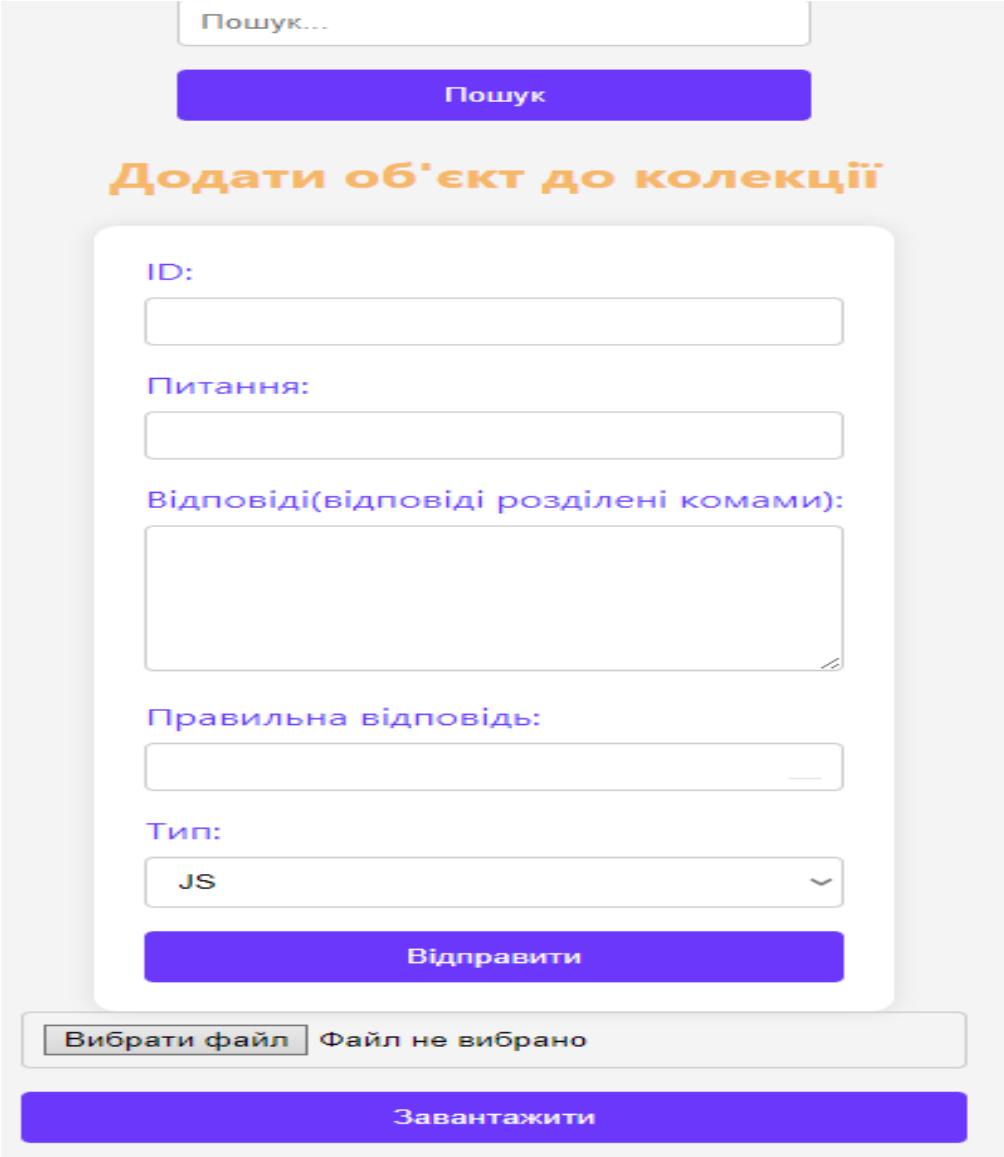

Рис.2.11 Форма для відправки питання

У адміністратора також є можливість редагування і видалення питань на самій адміністративній сторінці. На сторінці також додана пагінація, яка дозволяє відображати наш список питань у кількості по 5 одиниць на сторінці, що також дозволяє покращити функціонал самої адміністративної панелі Ці можливості також дозволяють швидко керувати питаннями і використовувати сторінку адміністратора на повну. На рисунку 2.12 зображене одинарне питання, а також кнопки пагінації.

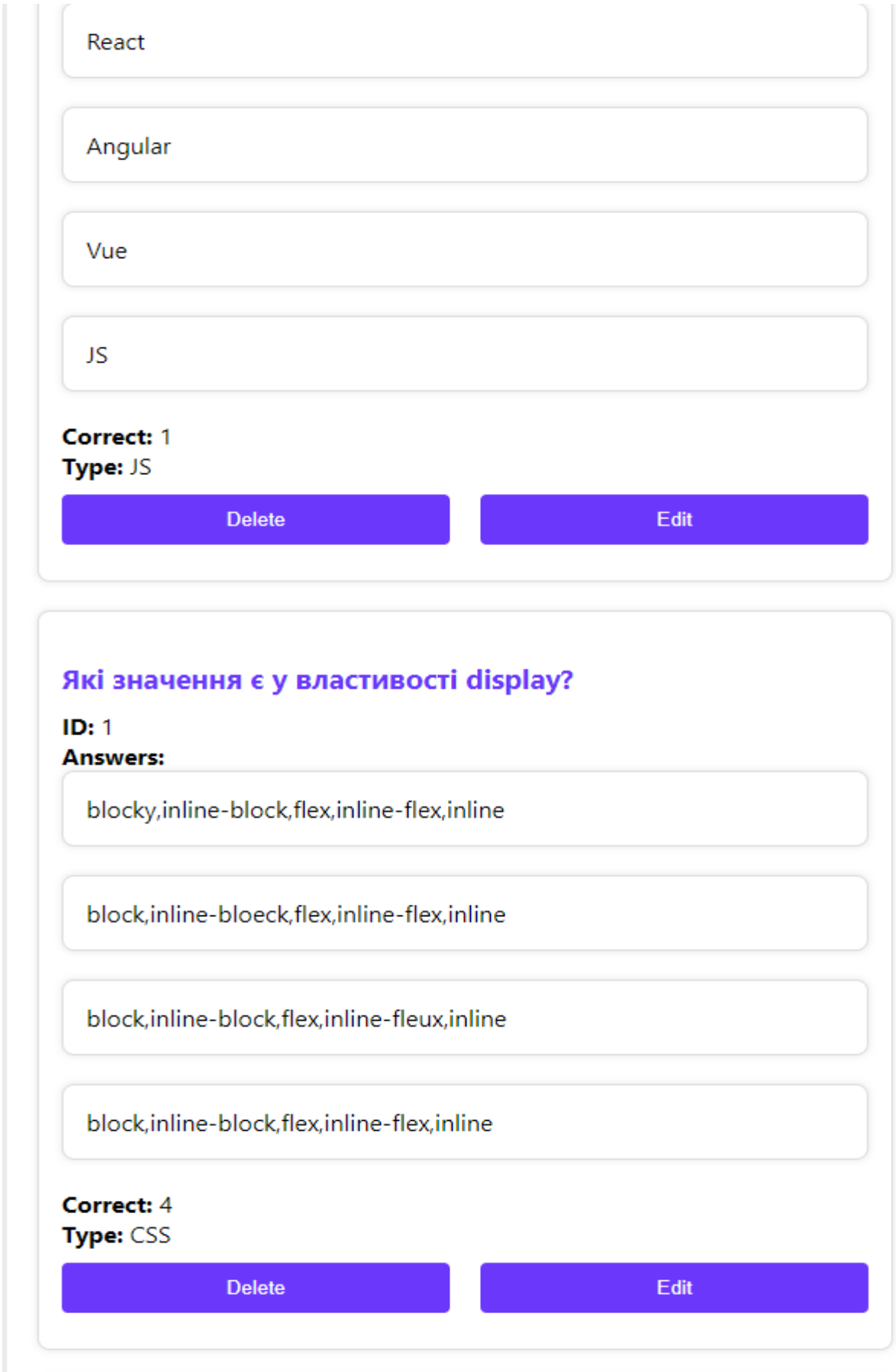

Рис.2.12 Приклад виведення питань

Щодо прикладу питання, які мають бути у файлі, – він має максимально простий і компактний вигляд, що є дуже зручним для використання і заповнення файлу з питаннями.

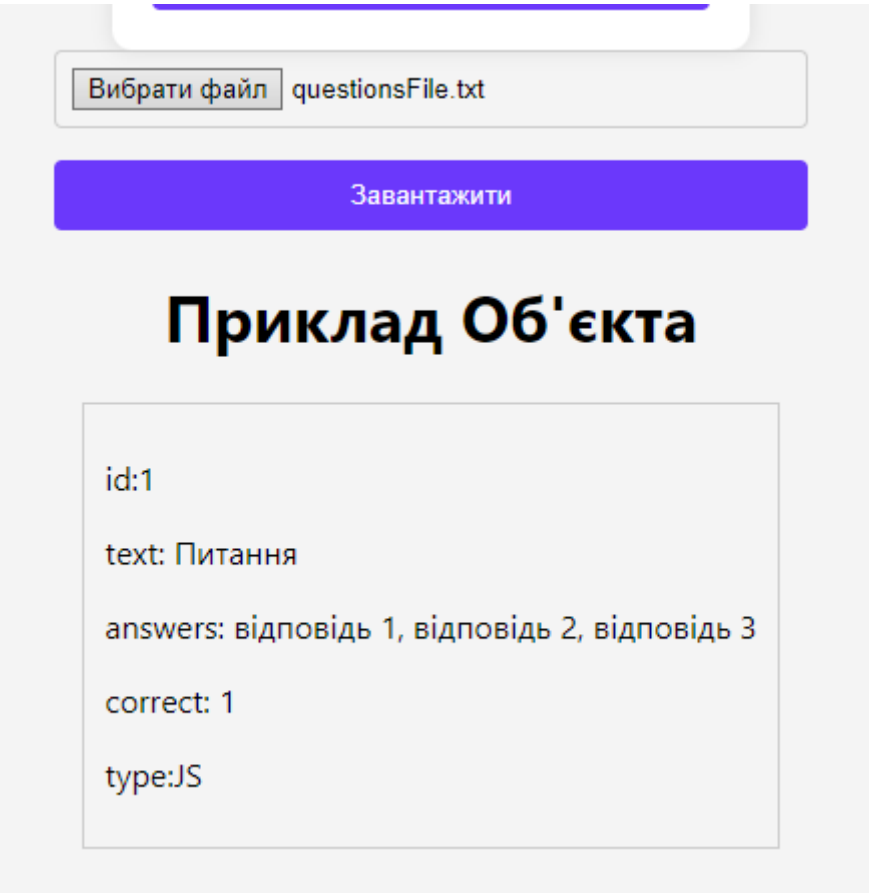

Рис.2.13 Приклад об’єкта у файлі

Висновок

Під час виконання проєкту було розроблено інтегровану систему, яка враховує потреби користувача під час підготовки як до співбесід, так і закріплюючи навчальний матеріал з предметів у закладах освіти, забезпечуючи зручний доступ до ресурсів бази даних та інтерактивний досвід спілкування з використанням GPT. Реалізація цього застосунку вимагала вивчення та впровадження новітніх технології та інструментів, що стосуються обробки даних, роботи з базами даних та інтеграції штучного інтелекту.

Найскладнішим етапом у розробці було створення інтегрованого чату GPT, який дозволяє користувачам взаємодіяти з ботом у природний спосіб. Цей етап передбачав вивчення архітектури GPT, інтеграцію зі зрозумілим інтерфейсом та оптимізацію продуктивності чату для безперервного та інтелектуального спілкування.

Використання мови програмування Python та бази даних MongoDB забезпечило ефективну обробку даних та швидкий доступ до необхідної інформації. Крім того, впроваджено можливості роботи з асинхронними запитами, що сприяє покращенню реакції бота на велику кількість одночасних взаємодій.

А використання мови розмітки HTML і мови програмування JS, дозволило нам зробити адміністративну панель, у якій у нас є змога додавати та переглядати питання до тестів. JavaScript дозволив нам зробити потрібні запити, для покращення роботи адміністратора, та дозволив розширити можливості нашого адміна, для вдосконалення нашого бота.

Отже, реалізована система відповідає поставленим завданням, надаючи користувачам зручну та інтелектуальну підтримку під час підготовки до співбесід і закріплення навчальної інформації.

Робота доповідалась на щорічній університетській студентській конференції [11].

Список використаних джерел

1. Python [Електронний ресурс]. – URL: https://aws.amazon.com/what-is/python/
2. MongoDB [Електронний ресурс]. – URL: https://www.w3schools.com/mongodb/ https://www.techtarget.com/searchdatamanagement/definition/MongoDB
3. BotFather [Електронний ресурс]. – URL: https://gerabot.com/article/botfather_mozhlivosti_ta_funkcional
4. ChatGPT [Електронний ресурс]. – URL: https://gerabot.com/article/what_is_chat_gpt__what_is_it_used_for
5. OpenAI [Електронний ресурс]. – URL: https://openai.com
6. Бібліотеки Python [Електронний ресурс]. – URL: https://pypi.org/project/
7. Visual Studio Code [Електронний ресурс]. – URL: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html
8. JavaScript [Електронний ресурс]. – URL: https://www.semrush.com/blog/javascript/
9. Бібліотеки JavaScript [Електронний ресурс]. – URL: https://www.npmjs.com/package/
10. Node.JS [Електронний ресурс]. – URL: https://www.ukraine.com.ua/uk/wiki/hosting/nodejs/overview
11. Зенюк М. (наук. кер. – Готинчан Т.І.) Створення телеграм-боту для тестів з підготовки до співбесіди на здобуття ІТ-професії // Матеріали студентської наукової конференції Чернівецького національного університету (16 – 18 квітня 2024 року). Факультет математики та інформатики. – Чернівці : Чернівецький нац. ун-т, 2024. – С. 52.

Додатки

Додаток А

```
import telebot
from pymongo import MongoClient
import random
import openai
import time
bot =
telebot.TeleBot("6565045394:AAE262pZg1XWN5s270I8BWl9peWFpVFwfEA")
openai.api_key = "sk-
6sNrrvNXnu8mXC8wl7UhT3BlbkFJJryL1uAvbNJyLvINH3DT"
class Database:
    def __init__(self):
        cluster =
MongoClient("mongodb+srv://user:M1234567@cluster0.7qbzryg.mongodb.
net/")
        self.db = cluster["QuizBot"]
        self.test_users_collection = self.db["Users"]
        self.questions_collection = self.db["Questions"]
        self.all_questions =
list(self.questions_collection.find({}))
        random.shuffle(self.all_questions)
        self.questions_count = len(self.all_questions)
    def get_test_user(self, chat_id):
        test_user =
self.test_users_collection.find_one({"chat_id": chat_id})
        if test_user is not None:
            return test_user
        new_test_user = {
            "chat_id": chat_id,
            "is_passing": False,
            "is_passed": False,
            "test_question_index": None,
            "answers": [],
            "question_type": None,
            "start_time": time.time()
        }
        self.test_users_collection.insert_one(new_test_user)
        return new_test_user
    def test_set_test_user(self, chat_id, update):
        self.test_users_collection.update_one({"chat_id":
chat_id}, {"$set": update})
    def get_questions(self, question_type):
        if question_type == "all":
            return self.all_questions
        return list(self.questions_collection.find({"type":
question_type}))
db = Database()
```

```
main_menu_markup =
telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
main_menu_markup.row("Почати вікторину", "Спілкування з GPT")
select_type_markup =
telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
select_type_markup.row("JS", "CSS", "HTML")
select_type_markup.row("Усі питання")
select_type_markup.row("Меню")
return_to_menu_markup =
telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
return_to_menu_markup.row("Меню")
@bot.message_handler(commands=["start"])
def start_handler(message):
    bot.send_message(message.chat.id, "Вітаємо! Виберіть опцію з
меню:", reply_markup=main_menu_markup)
def start_quiz_handler(message):
    test_user = db.get_test_user(message.chat.id)
    if test_user["is_passed"]:
        bot.send_message(message.from_user.id, "Ви вже проходили
цю вікторину! Другий раз немає можливості це зробити")
        return
    if test_user["is_passing"]:
        return
    # Встановлення часу початку тесту
    db.test_set_test_user(message.chat.id, {"is_passing": True,
"start_time": time.time()})
    bot.send_message(message.from_user.id, "Виберіть категорію
питань:", reply_markup=select_type_markup)

@bot.message_handler(func=lambda message: message.text ==
'Спілкування з GPT')
def chat_handler(message):
    test_user = db.get_test_user(message.chat.id)
    if test_user["is_passing"]:
        db.test_set_test_user(message.chat.id, {"is_passing":
False})
        bot.send_message(message.chat.id, "Вікторина завершена.
Почніть нову вікторину або ж можете поспілкуватися з GPT.",
reply_markup=main_menu_markup)
    else:
        bot.send_message(message.chat.id, "Ви увійшли в режим
спілкування з GPT. Введіть текст для обговорення або питання.")
@bot.message_handler(func=lambda message: message.text == 'Усі
питання' or message.text == 'JS' or message.text == 'CSS' or
message.text == 'HTML')
def select_type_handler(message):
    test_user = db.get_test_user(message.chat.id)
    if test_user["is_passing"] is False:
        return
    if message.text == "Усі питання":
        question_type = "all"
    else:
        question_type = message.text
```

```
    questions = db.get_questions(question_type)
    random.shuffle(questions)
    db.test_set_test_user(message.chat.id, {"test_question_index":
0, "question_type": question_type, "answers": []})
    test_user = db.get_test_user(message.chat.id)
    quiz_message = get_quiz_message(test_user, questions)
    if quiz_message is not None:
        bot.send_message(message.from_user.id,
quiz_message["text"], reply_markup=quiz_message["keyboard"])
@bot.message_handler(func=lambda message: message.text == 'Меню')
def menu_handler(message):
    bot.send_message(message.chat.id, "Виберіть опцію з меню:",
reply_markup=main_menu_markup)
@bot.message_handler(
    content_types=["text"],
    func=lambda message:
db.get_test_user(message.chat.id)["is_passing"] is False,
)
def message_handler(message):
    response = openai.Completion.create(
        model="gpt-3.5-turbo-instruct",
        prompt=message.text,
        temperature=0.5,
        max_tokens=1000,
        top_p=1.0,
        frequency_penalty=0.5,
        presence_penalty=0.5,
    )
    gpt_text = response["choices"][0]["text"]
    bot.send_message(message.chat.id, gpt_text)
@bot.callback_query_handler(func=lambda query: query.data ==
"?end_quiz")
def end_quiz_callback(query):
    test_user = db.get_test_user(query.message.chat.id)
    if test_user["is_passing"]:
        # Завершення тесту
        test_user["is_passing"] = False
        db.test_set_test_user(query.message.chat.id, test_user)
        bot.send_message(query.message.chat.id, "Тест завершено.
Результати не  зберігатимуться.", reply_markup=main_menu_markup)
    else:
        bot.send_message(query.message.chat.id, "Ви не проходите
жодного тесту.", reply_markup=main_menu_markup)

@bot.callback_query_handler(func=lambda query:
query.data.startswith("?ans"))
def answered_handler(query):
    test_user = db.get_test_user(query.message.chat.id)
    if test_user["is_passed"] or not test_user["is_passing"]:
        return
    test_user["answers"].append(int(query.data.split("&")[1]))
    db.test_set_test_user(query.message.chat.id, {"answers":
test_user["answers"]})
```

```
    questions = db.get_questions(test_user["question_type"])
    answer_message = get_answer_message(test_user, questions)
    if answer_message is not None:
        bot.edit_message_text(
            answer_message["text"],
            query.message.chat.id,
            query.message.id,
            reply_markup=answer_message["keyboard"],
        )
@bot.callback_query_handler(func=lambda query: query.data ==
"?next")
def next_handler(query):
    test_user = db.get_test_user(query.message.chat.id)
    if test_user["is_passing"]:
        test_user["test_question_index"] += 1
        db.test_set_test_user(query.message.chat.id,
{"test_question_index": test_user["test_question_index"]})
        questions = db.get_questions(test_user["question_type"])
        quiz_message = get_quiz_message(test_user, questions)
        if quiz_message is not None:
            bot.edit_message_text(
                quiz_message["text"],
                query.message.chat.id,
                query.message.id,
                reply_markup=quiz_message["keyboard"],
            )
        # Перевіряємо, чи користувач пройшов всі питання
        if test_user["test_question_index"] == len(questions):
            # Обчислення часу, який пройшов від початку тесту
            elapsed_time = int(time.time() -
test_user["start_time"])
            minutes = elapsed_time // 60
            seconds = elapsed_time % 60
            time_string = f"Час: {minutes} хв {seconds} с"
            # Підрахунок кількості вірних і невірних відповідей
            correct_answers = sum(1 for i, q in
enumerate(questions) if i < len(test_user["answers"]) and
q["correct"] == test_user["answers"][i])
            incorrect_answers = len(test_user["answers"]) -
correct_answers
            score_string = f"Правильних: {correct_answers},
Неправильних: {incorrect_answers}"
            # Відправлення повідомлення з результатами тесту
            bot.send_message(
                query.message.chat.id,
                f"Тест завершено!\n{time_string}\n{score_string}",
                reply_markup=None
            )
            # Зберігаємо результати тестування у колекції
результатів
            test_results = {
                "chat_id": test_user["chat_id"],
                "time": elapsed_time,
```

```
                "correct_answers": correct_answers,
                "total_questions": len(questions)
            }
            db.db["TestResults"].insert_one(test_results)
            # Починаємо нове тестування для користувача
            db.test_set_test_user(test_user["chat_id"],
{"is_passing": False, "is_passed": False, "test_question_index":
None, "answers": [], "question_type": None, "start_time": None})
    else:
        bot.send_message(query.message.chat.id, "Виберіть опцію з
меню:", reply_markup=main_menu_markup)

def get_quiz_message(test_user, questions):
    if test_user["test_question_index"] == len(questions):
        correct_count = sum(1 for i, q in enumerate(questions) if
q["correct"] == test_user["answers"][i])
        percentage = round(100 * correct_count / len(questions))
        smile = "😕" if percentage < 40 else "😐" if percentage <
60 else "😀" if percentage < 90 else "😎"
        text = f"Ви відповіли правильно на {percentage}%
питань{smile}"
        db.test_set_test_user(test_user["chat_id"], {"is_passed":
True, "is_passing": False})
        return {"text": text, "keyboard": None}
    question = questions[test_user["test_question_index"]]
    keyboard = telebot.types.InlineKeyboardMarkup()
    for answer_index, answer in enumerate(question["answers"]):
        keyboard.row(
            telebot.types.InlineKeyboardButton(
                f"{chr(answer_index + 97)}) {answer}",
                callback_data=f"?ans&{answer_index}",
            )
        )
    # Додаємо кнопку "Завершити тест"
    keyboard.row(
        telebot.types.InlineKeyboardButton("Завершити тест",
callback_data="?end_quiz")
    )
    text = f"Питання  №{test_user['test_question_index'] +
1}\n\n{question['text']}"
    return {"text": text, "keyboard": keyboard}
def get_answer_message(test_user, questions):
    question = questions[test_user["test_question_index"]]
    text = f"Питання  №{test_user['test_question_index'] +
1}\n\n{question['text']}\n"
    for answer_index, answer in enumerate(question["answers"]):
        text += f"{chr(answer_index + 97)}) {answer}"
        if answer_index == question["correct"]:
            text += " ✅"
        elif answer_index == test_user["answers"][-1]:
            text += " ❌"
        text += "\n"
```

```
    keyboard = telebot.types.InlineKeyboardMarkup()
    keyboard.row(
        telebot.types.InlineKeyboardButton("Наступне",
callback_data="?next")
    )
    return {"text": text, "keyboard": keyboard}
bot.polling()
```

Додаток Б

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Add Object to MongoDB Collection</title>
</head>
<style>
    body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
        background-color: #f4f4f4;
        margin: 0;
        padding: 0;
        display: flex;
        justify-content: center;
        align-items: center;
        flex-direction: column;
    }

    h2 {
        color: #f6b768;
        text-align: center;
    }

    #mongoForm {
        max-width: 400px;
        margin: 0 auto;
        background-color: #ffffff;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    label {
        display: block;
        margin-bottom: 8px;
        color: #6b38fb;
    }
```

```
input,
select {
    width: 100%;
    padding: 8px;
    margin-bottom: 16px;
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    background-color: #6b38fb;
    color: #ffffff;
    padding: 10px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
    transition: background 0.3s ease-in-out;
}

button:hover {
    background: #6b58aa;
}

@media (max-width: 600px) {
    #mongoForm {
        width: 90%;
    }
}

ul {
    padding-left: 0px;
}

#mongoDataList {
    list-style-type: none;
    padding: 0;
    margin: 0;
    text-align: left;
    width: 80%;
    max-width: 600px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    background-color: #ffffff;
    border-radius: 10px;
    padding: 20px;
    box-sizing: border-box;
    margin-top: 40px;

}

#mongoDataList li {
    margin-bottom: 20px;
```

```
        padding: 15px;
        border: 1px solid #ddd;
        border-radius: 8px;
        background-color: #fff;
        box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
        list-style: none;
    }

    #mongoDataList li h3 {
        color: #6b38fb;
        margin-bottom: 10px;
    }

    #mongoDataList li p {
        margin: 0;
    }

    img {
        display: block;
        width: 100%;
    }

    textarea {
        width: 100%;
        padding: 10px;
        margin-bottom: 16px;
        box-sizing: border-box;
        border: 1px solid #ccc;
        border-radius: 4px;
        resize: vertical;
        min-height: 100px;
    }

    textarea:focus {
        border-color: #6b38fb;
    }

    button:disabled {
        background-color: #888888;
        color: #ffffff;
    }

    .note {
        border: 1px solid #ccc;
        padding: 10px;
        margin-bottom: 10px;
    }
</style>

<body>
    <form id="searchForm">
        <input type="text" id="searchInput"
placeholder="Пошук...">
```

```
        <button type="submit">Пошук</button>
    </form>
    <h2>Додати об'єкт до колекції</h2>

    <form id="mongoForm" action="" method="post">
        <label for="idNumber">ID:</label>
        <input type="number" id="idNumber" name="id" required><br>

        <label for="textString">Питання:</label>
        <input type="text" id="textString" name="text"
required><br>

        <label for="answers">Відповіді(відповіді розділені
комами):</label>
        <textarea id="answers" name="answers" placeholder=""
required></textarea><br>

        <label for="correctNumber">Правильна відповідь:</label>
        <input type="number" id="correctNumber" name="correct"
required><br>

        <label for="type">Тип:</label>
        <select id="type" name="type" required>
            <option value="JS">JS</option>
            <option value="CSS">CSS</option>
            <option value="HTML">HTML</option>
        </select><br>

        <button onclick="submitForm()"
type="submit">Відправити</button>
    </form>

    <form action="/upload" method="post" enctype="multipart/form-
data">
        <input type="file" name="questionsFile" accept=".txt">
        <button type="submit">Завантажити</button>
    </form>


    <h1>Приклад Об'єкта</h1>
    <div class="note">
        <p>id:1</p>
        <p>text: Питання</p>
        <p>answers: відповідь 1, відповідь 2, відповідь 3</p>
        <p>correct: 1</p>
        <p>type:JS</p>
    </div>

    <!--<img src="example.png"
        style="display: block; margin: 0 auto; opacity: 1; height:
100%; width: 100%; max-width: 450px;" alt="">-->
```

```
    <ul id="mongoDataList">

    </ul>


    <!-- Pagination buttons -->
    <div style="display: flex; margin: 15px; gap: 20px;">
        <button id="prevButton"
onclick="handlePagination('prev')">Попередня</button>
        <button id="nextButton"
onclick="handlePagination('next')">Наступна</button>
    </div>

</body>

</html>

<script>
    // Function to escape HTML entities
    function escapeHtml(html) {
        var text = document.createTextNode(html);
        var p = document.createElement('p');
        p.appendChild(text);
        return p.innerHTML;
    }

    // Функція для виконання пошуку
    function searchItems(searchTerm) {
        const dataList = document.getElementById('mongoDataList');
        const items = dataList.getElementsByTagName('li');
        searchTerm = searchTerm.toLowerCase();
        Array.from(items).forEach(item => {
            const text = item.innerText.toLowerCase();
            if (text.includes(searchTerm)) {
                item.style.display = 'block';
            } else {
                item.style.display = 'none';
            }
        });
    }

    // Функція для обробки подання форми пошуку
    function handleSearch() {
        const searchInput =
document.getElementById('searchInput');
        const searchTerm = searchInput.value.trim();
        searchItems(searchTerm);
    }
```

```
    // Додати обробник події для форми пошуку

document.getElementById('searchForm').addEventListener('submit',
function (event) {
        event.preventDefault();
        handleSearch();
    });




    let currentPage = 1;
    const itemsPerPage = 7;
    let fetchedData;


    function displayItemsWithPagination() {
        const dataList = document.getElementById('mongoDataList');
        dataList.innerHTML = ''; // Clear previous content
        const startIndex = (currentPage - 1) * itemsPerPage;
        const endIndex = startIndex + itemsPerPage;
        const paginatedData = fetchedData.slice(startIndex,
endIndex);
        paginatedData.forEach(item => {
            const listItem = document.createElement('li');
            listItem.setAttribute('id', `item-${item.id}`);
            listItem.innerHTML = `
            <h3>${item.text}</h3>
            <p><strong>ID:</strong> ${item.id}</p>
            <p><strong>Answers:</strong>
                <ul>
                    ${item.answers.map(answer =>
`<li>${escapeHtml(answer)}</li>`).join('')}
                </ul>
            </p>
            <p><strong>Correct:</strong> ${+item.correct + 1}</p>
            <p><strong>Type:</strong> ${item.type}</p>
            <div style="display:flex; margin:10px auto;
gap:20px;">
            <button
onclick="deleteItem(${item.id})">Delete</button>
            <button onclick="editItem(${item.id})">Edit</button>
            </div>
        `;
            dataList.appendChild(listItem);
        });
    }

    // Function to handle pagination navigation
    function handlePagination(direction) {
        const prevButton = document.getElementById('prevButton');
        const nextButton = document.getElementById('nextButton');
```

```
        if (direction === 'prev' && currentPage > 1) {
            currentPage--;
        } else if (direction === 'next' && currentPage <
Math.ceil(fetchedData.length / itemsPerPage)) {
            currentPage++;
        }

        // Update pagination buttons state
        prevButton.disabled = currentPage === 1;
        nextButton.disabled = currentPage ===
Math.ceil(fetchedData.length / itemsPerPage);

        displayItemsWithPagination();
    }

    // Fetch data from the server and update the content
    fetch('/getData')
        .then(response => response.json())
        .then(data => {
            fetchedData = data;
            displayItemsWithPagination();
        })
        .catch(error => console.error('Error fetching data:',
error));

    // Function to delete an item
    function deleteItem(id) {
        fetch(`/${id}`, {
            method: 'DELETE',
            headers: {
                'Content-Type': 'application/json'
            },
        })
            .then(response => {
                if (response.ok) {
                    // Reload the data after successful deletion
                    fetch('/getData')
                        .then(response => response.json())
                        .then(data => {
                            fetchedData = data;
                            displayItemsWithPagination();
                        })
                        .catch(error => console.error('Error
fetching data:', error));
                } else {
                    console.error('Failed to delete item');
                }
            })
            .catch(error => console.error('Error deleting item:',
error));
    }

    // Function to display the edit form for an item
```

```
    function editItem(id) {
        const item = fetchedData.find(item => item.id === id);
        if (!item) {
            console.error('Item not found');
            return;
        }

        const editForm = document.createElement('form');
        editForm.innerHTML = `
            <label for="editId">ID:</label>
            <input type="number" id="editId" name="editId"
value="${item.id}" disabled><br>
            <label for="editText">Text:</label>
            <input type="text" id="editText" name="editText"
value="${item.text}"><br>
            <label for="editAnswers">Answers (comma-
separated):</label>
            <textarea id="editAnswers"
name="editAnswers">${item.answers.join(',')}</textarea><br>
            <label for="editCorrect">Correct:</label>
            <input type="number" id="editCorrect"
name="editCorrect" value="${item.correct}"><br>
            <label for="editType">Type:</label>
            <select id="editType" name="editType">
                <option value="JS" ${item.type === 'JS' ?
'selected' : ''}>JS</option>
                <option value="CSS" ${item.type === 'CSS' ?
'selected' : ''}>CSS</option>
                <option value="HTML" ${item.type === 'HTML' ?
'selected' : ''}>HTML</option>
            </select><br>
            <button onclick="submitEditForm(${item.id})"
type="button">Save Changes</button>
        `;

        // Replace existing item details with edit form
        const itemElement = document.getElementById(`item-${id}`);
        itemElement.innerHTML = '';
        itemElement.appendChild(editForm);
    }

    function submitEditForm(id) {
        const editText =
document.getElementById('editText').value;
        const editAnswers =
document.getElementById('editAnswers').value.split(',').map(answer
=> answer.trim());
        const editCorrect =
document.getElementById('editCorrect').value;
        const editType =
document.getElementById('editType').value;

        fetch(`/${id}`, {
```

```
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                id: id,
                text: editText,
                answers: editAnswers,
                correct: editCorrect,
                type: editType
            })
        })
            .then(response => {
                if (response.ok) {
                    // Reload the data after successful update
                    fetch('/getData')
                        .then(response => response.json())
                        .then(data => {
                            fetchedData = data;
                            displayItemsWithPagination();
                        })
                        .catch(error => console.error('Error
fetching data:', error));
                } else {
                    console.error('Failed to update item');

                }
            })
            .catch(error => console.error('Error updating item:',
error));
    }


</script>
<script src="app.js"></script>

APP.js
const express = require("express");
const app = express();
const MongoClient = require("mongodb").MongoClient;
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const ObjectId = require("mongodb").ObjectId;
const multer = require("multer");
const upload = multer({ dest: "uploads/" });
const fs = require("fs");
const split = require("split");

mongoose.connect(

"mongodb+srv://user:M1234567@cluster0.7qbzryg.mongodb.net/QuizBot"
);
```

```
const uri =
"mongodb+srv://user:M1234567@cluster0.7qbzryg.mongodb.net/QuizBot"
;

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/index.html");
});

app.post("/", async (req, res) => {
  const formData = {
    id: +req.body.id,
    text: req.body.text,
    answers: req.body.answers.split(",").map((answer) =>
answer.trim()), // Split and trim answers
    correct: +req.body.correct - 1,
    type: req.body.type,
  };

  try {
    const client = await MongoClient.connect(uri);
    const database = client.db("QuizBot");
    const collection = database.collection("Questions");

    const result = await collection.insertOne(formData);

    console.log("Document inserted successfully:",
result.insertedId, result.insertedText);

    // Close the MongoDB connection
    client.close();

    res.status(201).send("Form data saved successfully!");
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
});

app.get("/getData", async (req, res) => {
  try {
    const client = await MongoClient.connect(uri);
    const database = client.db("QuizBot");
    const collection = database.collection("Questions");

    const data = await collection.find({}).toArray();

    res.json(data);

    client.close();
  } catch (error) {
```

```
    console.error(error);
    res.status(500).json({ error: "Internal Server Error" });
  }
});

// Delete document by ID
app.delete("/:id", async (req, res) => {
  const id = +req.params.id;
  try {
    const client = await MongoClient.connect(uri);
    const database = client.db("QuizBot");
    const collection = database.collection("Questions");

    // Delete document
    const result = await collection.deleteOne({ id: id });

    // Check if the document was deleted
    if (result.deletedCount === 1) {
      res.status(200).send("Document deleted successfully!");
    } else {
      res.status(404).send("Document not found");
    }

    client.close();
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
});

// Маршрут для оновлення елемента за його ID
app.put("/:id", async (req, res) => {
  const id = +req.params.id; // Отримати ID з параметрів шляху
  const updateData = req.body; // Отримати дані для оновлення з
запиту

  try {
    const client = await MongoClient.connect(uri);
    const database = client.db("QuizBot");
    const collection = database.collection("Questions");

    // Оновити документ
    const result = await collection.updateOne({ id: id }, { $set:
updateData });

    // Перевірити, чи був оновлений документ
    if (result.matchedCount === 1) {
      res.status(200).send("Document updated successfully!");
    } else {
      res.status(404).send("Document not found");
    }

    client.close();
```

```
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
});

app.post("/upload", upload.single("questionsFile"), async (req,
res) => {
  try {
    if (!req.file) {
      return res.status(400).send("No file uploaded.");
    }

    const fsStream = fs.createReadStream(req.file.path, {
encoding: "utf8" });
    const objects = [];

    let currentObject = {};

    fsStream
      .pipe(split(/\n/))
      .on("data", (line) => {
        if (line.trim() === "") {
          if (currentObject) {
            objects.push(currentObject);
          }
          currentObject = {};
        } else {
          const [key, value] = line.split(": ");
          currentObject[key.trim()] = value.trim();
        }
      })
      .on("end", async () => {
        if (currentObject) {
          objects.push(currentObject);
        }
        console.log("Objects from file:", objects);

        try {
          const client = await MongoClient.connect(uri);
          const database = client.db("QuizBot");
          const collection = database.collection("Questions");

          for (const obj of objects) {
            const formData = {
              id: +obj.id,
              text: obj.text,
              answers: obj.answers
                .split(",")
                .map((answer) => answer.trim()), // Split and trim
answers
              correct: +obj.correct - 1,
              type: obj.type,
```

```
            };
            const result = await collection.insertOne(formData);
            console.log("Document inserted successfully:",
result.insertedId);
          }

          client.close();

          fs.unlinkSync(req.file.path);
          console.log("File deleted successfully.");
          res
            .status(200)
            .send("File uploaded and data added to the database
successfully.");
        } catch (error) {
          console.error("Error inserting documents:", error);
          res.status(500).send("Internal Server Error: " +
error.message);
        }
      });
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error: " +
error.message);
  }
});

app.listen(3000, function () {
  console.log("server listening on port 3000");
});
```