Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)
«___»_____202__р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОНТРОЛЮ ТА ПЕРЕВІРКИ ЗНАНЬ З ДИСЦИПЛІНИ «ШТУЧНИЙ ІНТЕЛЕКТ»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Фісун Денис Анатолійович
_____«___»_____202__р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Чілікіна Тетяна Василівна
_____«___»_____202__р.
(підпис)

Резидент _____

ПОЛТАВА – 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА

«____» _____ 202__р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Створення програмного забезпечення для контролю та перевірки знань з дисципліни “Штучний інтелект”»
зі спеціальності 122 Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавр
Прізвище, ім’я, по батькові Фісун Денис Анатолійович
Затверджена наказом ректора № ____-Н від «___» _____ 202_ р.
Термін подання студентом роботи «___ » _____ 202_ р.
Вихідні дані до кваліфікаційної роботи: публікації з теми, веб-сайти сайтів кафедр.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1 Проблематика створення веб-додатків
2.2 Огляд програмних засобів для створення веб-додатків
2.3 Уведення до тематики штучного інтелекту
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1 Розробка структури проекту
3.2 Обґрунтування вибору вказаних технологій для розробки веб-додатку
3.3 Обґрунтування вибору середовища розробки
3.4 Алгоритм роботи програмного забезпечення
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1 Опис створення програми
4.2 Інструкція користувача
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ ТА ДЖЕРЕЛ
ДОДАТОК А. Код програми
ДОДАТОК Б. Перелік тестових питань
Перелік графічного матеріалу: 25 рисунків

Консультанти розділів кваліфікаційної роботи

Розділ	ПІБ, посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Постанова задачі	Чілікіна Т.В.		
Інформаційний огляд	Чілікіна Т.В.		
Теоретична частина	Чілікіна Т.В.		
Практична частина	Чілікіна Т.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3 .Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6 .Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доопрацювання (за необхідністю), рецезування		

Дата видачі завдання «____»_____ 202__ р.
Здобувач вищої освіти Фісун Денис Анатолійович
Науковий керівник Чілікіна Т.В.

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на_____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____
(підпис) (ініціал та прізвище)

Затверджую
Зав. кафедрою_____
к.ф.-м.н. Олена Ольховська
«____» _____ 202_ р.

Погоджено
Науковий керівник_____
к.ф.-м.н. Чілікіна Т.В.
«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Створення програмного забезпечення для контролю та перевірки знань з дисципліни “Штучний інтелект”»
зі спеціальності 122 Комп’ютерні науки
освітня програма 122 «Комп’ютерні науки»
ступеня бакалавр
Прізвище, ім’я, по батькові Фісун Денис Анатолійович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1 Проблематика створення веб-додатків
2.2 Огляд програмних засобів для створення веб-додатків
2.3 Уведення до тематики штучного інтелекту
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1 Розробка структури проекту
3.2 Обґрунтування вибору вказаних технологій для розробки веб-додатку
3.3 Обґрунтування вибору середовища розробки
3.4 Алгоритм роботи програмного забезпечення
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1 Опис створення програми
4.2 Інструкція користувача
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ ТА ДЖЕРЕЛ
ДОДАТОК А. Код програми

Здобувач вищої освіти _____ _____

(підпис)

«____» _____ 2025 р.

РЕФЕРАТ

Записка: 82 стр., 27 рис., 2 додатки, 30 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, ВЕБ-ДОДАТОК, РОЗГОРТАННЯ FLASK-СЕРВЕРУ

Об'єкт розробки – алгоритмізація та створення працюючого веб-додатку у мережі Internet з перевіркою знань у спосіб проходження тесту на тему «Штучний інтелект».

Тема роботи - створення програмного забезпечення для контролю та перевірки знань з дисципліни «штучний інтелект».

Мета роботи – створення програмного забезпечення для контролю та перевірки знань з дисципліни «штучний інтелект»

Методи дослідження та інформаційне забезпечення – середовище розробки Microsoft Visual Studio Code, мова програмування Python, стандартна бібліотека Python3, фремворк Flask, методи розробки веб-додатків, розгортання бекенд серверу, створення веб-інтерфейсів, а також використання бази даних MySQL для зберігання та управління даними.

Предмет дослідження - процес розробки веб-додатку для тестування знань користувача із теми штучного інтелекту із використанням сучасних веб-технологій.

Результати дослідження – розглянуто основні дії, їх послідовність та етапи створення веб-додатку за допомогою інструментів мови програмування **Python**, його фреймворку для створення веб-додатків **Flask**, базову роботу з базами даних для необхідної для веб-додатку взаємодії інформації між структурою «Клієнт-сервер», мовою SQL для створення запитів до бази даних та роботу з локальною СУБД (Система Управління Базами Даних) SQLite3. Розглянуто переваги мови **Python** та інших використаних інструментів перед альтернативами для написання веб-додатку.

В результаті тестування виявлено, що веб-сайт, POST & GET запити, робота з базами даних, алгоритми роботи та оцінювання тестів працюють коректно.

Ключові слова: Штучний інтелект, веб-додаток, база даних, Python, Flask, алгоритм, блок-схема, SQL, MySQL, Jinja2, Bootstrap, HTML, CSS, JS, бекенд, фронтенд, дизайн, авторизація користувача, шифрування, Json, сервер, користувач, HOST.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ 10
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД 15
2.1 Проблематика створення веб-додатків 15
2.2 Огляд програмних засобів для створення веб-додатків 16
2.3 Уведення до тематики штучного інтелекту 18
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА 20
3.1 Розробка структури проекту 20
3.2 Обґрунтування вибору вказаних технологій для розробки веб-додатку 21
3.3 Обґрунтування вибору середовища розробки 26
3.4 Алгоритм роботи програмного забезпечення 27
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА 31
4.1 Опис створення програми 31
4.2 Інструкція користувача 66
ВИСНОВКИ 70
СПИСОК ЛІТЕРАТУРИ ТА ДЖЕРЕЛ 73
ДОДАТОК А. Код програми 76
ДОДАТОК Б. Перелік тестових питань 77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	**Пояснення умовних позначень, символів, скорочень**
Python	Інтерпретована мова програмування
VS Code	Редактор коду від Microsoft
Фреймворк	Програмне середовище, котре спростовує створення програмного забезпечення
IDE	Інтегроване середовище розробки
ОС	Операційна система
ШІ	Штучний інтелект
ПЗ	Програмне забезпечення
БД	База даних
СУБД	Система управління базою даних

ВСТУП

У сучасному світі стрімкого розвитку цифрових технологій однією з найактуальніших тем є штучний інтелект (ШІ). Сфера його застосування постійно розширюється: від побутових голосових помічників до складних аналітичних систем у медицині, фінансах, транспорті та навіть мистецтві. Сьогодні ШІ є не просто інноваційною галуззю, а стратегічним напрямом розвитку світової науки та економіки. У зв’язку з цим виникає нагальна потреба у підготовці фахівців, які не лише володіють базовими знаннями про принципи роботи ШІ, але й здатні аналізувати, створювати та впроваджувати такі технології на практиці.

Одним із ефективних способів оцінки знань у цій сфері є створення веб-додаток для тестування. Веб-сайти з тестами – це зручний і доступний інструмент для перевірки рівня засвоєння навчального матеріалу, самоконтролю та підготовки до сертифікацій або співбесід. Вони дозволяють охопити широку аудиторію та забезпечити інтерактивну взаємодію користувача з матеріалом. Особливо актуальними такі інструменти є у дистанційному навчанні, яке набуло популярності в умовах пандемії COVID-19 і продовжує активно використовуватися в закладах освіти.

Штучний інтелект уже сьогодні використовується у численних сферах: в охороні здоров’я – для діагностики захворювань, у фінансах – для виявлення шахрайських операцій, у логістиці – для оптимізації маршрутів доставки, в аграрному секторі – для прогнозування врожайності, а також у розважальній індустрії – через рекомендаційні системи. У найближчому майбутньому ШІ все більше впливатиме на прийняття управлінських рішень, розумну автоматизацію виробництв і навіть на розробку законодавчих норм.

Таким чином, формування системи тестування знань з дисципліни «Штучний інтелект» у форматі веб-додатку є не лише актуальним, але й необхідним елементом сучасної освітньої практики. Такий сайт може слугувати

як допоміжний інструмент у навчальному процесі, сприяти самонавчанню та підвищенню кваліфікації фахівців різних напрямів.

Об’єкт розробки – алгоритмізація та створення працюючого веб-додатку у мережі Internet з перевіркою знань у спосіб проходження тесту на тему «Штучний інтелект».

Предмет дослідження - процес розробки веб-додатку для тестування знань користувача із теми штучного інтелекту із використанням сучасних веб-технологій.

Мета роботи – створення програмного забезпечення для контролю та перевірки знань з дисципліни «штучний інтелект».

Актуальність роботи полягає в тому, що сьогодні штучний інтелект – це не просто модне слово, а реальність, яка стрімко змінює практично всі сфери життя: від медицини та освіти до бізнесу та мистецтва. Швидкість розвитку технологій ШІ вражає, але водночас створює виклик: як об’єктивно оцінити реальний рівень розуміння цих технологій? Класичні методи перевірки знань часто відстають від динаміки галузі. Саме тут виникає гостра потреба у спеціалізованих інструментах. Існуючі платформи для тестування часто бувають або занадто загальними, або надто вузькоспеціалізованими, або ж не мають достатньої кількості якісних матеріалів саме українською мовою. Крім того, важливо не просто "зазубрити" термінологію, а зрозуміти принципи роботи алгоритмів, етичні аспекти та практичні можливості застосування ШІ.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Досягнення мети передбачає виконання наступних завдань:

- Аналіз вже існуючих веб-додатків з тестами, щоб мати уявлення про загальний вигляд сайту та його роботи
- Аналіз доступних інструментів які допоможуть в досягненні мети та порівняння з альтернативами
- Вивчення предметної області вибраних інструментів, перегляд офіційної документації потрібних для нас мов, бібліотек та ін. (таких як мова **Python**, фреймворк **Flask**)
- Вибір середовища розробки
- Формування вимог
  - Визначити мету сайту (тестування знань з ШІ).
  - Скласти список функцій: реєстрація, авторизація, проходження тестів, перегляд результатів, особистий кабінет.
- Підготовка середовища
  - Встановити **Python**.
  - Створити віртуальне середовище (venv).
  - Встановити **Flask** та необхідні бібліотеки (Flask-Login, Flask-WTF, **SQLite** тощо).
- Проектування структури
  - Розробити структуру бази даних: користувачі, тести, запитання, результати.
  - Спроектувати маршрути сайту (наприклад: /, /login, /register, /test, /profile).
- Реалізація функціоналу
  - Створити шаблони HTML з **Jinja2**.
  - Реалізувати:
    - Головну сторінку.
    - Сторінку відображення інформації про автора веб-додатку

- Реєстрацію/вхід/вихід.
- Сторінку проходження тесту.
- Збереження результатів.
- Особистий кабінет з історією тестів.

- Тестування
  - Перевірити роботу кожного маршруту та форми.
  - Перевірити валідацію введення.
  - Перевірити збереження результатів у БД.
- Оформлення

1. Додати стилі (CSS/Bootstrap).
2. Додати адаптивність та базову анімацію.

Також визначимо кожен аспект майбутнього сайту: які розділи та сторінки в ньому мають бути, які можуть бути проблеми та «баги» в момент роботи веб-додатку, як має працювати логіка проходження тесту, вигляд особистого кабінету та можливості, які він буде надавати, загальний дизайн сайту, тощо.

Визначимо сторінки, які мають бути на сайті:

1. **Головна сторінка**

   На головній сторінці буде перш за все відображення привітання нового користувача – будуть пояснені основні аспекти сайту, для чого він був створений, що користувач може в ньому зробити і т.д. Також користувач повинен мати можливість попасти на сторінку реєстрації нового облікового запису (або входу до нього, якщо такий запис все існує).

2. **Сторінка реєстрації**

   На сторінці реєстрації користувач буде мати змогу зробити новий обліковий запис. Його наявність необхідна для того, щоб була

можливість зберігати результат усіх спроб користувача пройти тест унікально для кожного. Після вказання логіну, паролю, номеру групи користувача в університеті та імені з прізвищем, користувач потрапить до особистого кабінету.

3. Сторінка особистого кабінету

Ця сторінка призначена для управління можливостями користувача: зміна особистих даних облікового запису, видалення аккаунту, можливість пройти тест, змогу побачити усі свої спроби проходження тесту та загальну оцінку кожної спроби.

4. Сторінка проходження тесту

Безпосередньо сторінка, де користувач може зробити оцінку своїх знань з теми «Штучний інтелект». Кожен раз порядок питань до тесту повинен мати випадкове положення, щоб запобігти банальному запам’ятовування послідовності відповідей. Сторінка тесту повинна мати приємний для ока вигляд і водночас не має відволікати користувача від проходження тесту.

5. Сторінка результатів тесту

Після успішного проходження користувачем тесту, він повинен побачити його результати. Сторінка має бути максимально інформативною по відношенню до відповідей користувача. Сторінка має відображати відповіді, на які користувач відповів правильно, та неправильні відповіді. Якщо користувач не зміг дати вірну відповідь, то під нею має бути підказка, яка допоможе йому в майбутньому дати правильну відповідь. Також повинна бути загальна оцінка з тестування та поради користувачу, як покращити свої знання з теми «Штучний інтелект».

6. Сторінка «про автора»

У випадку, якщо користувачу цікаво знати хто зробив веб-додаток або у нього з’являться питання щодо працездатності веб-додатку чи його покращення, користувач буде мати змогу дізнатись про

автора, контакти для зв’язку з ним та мінімальна біографія щодо шляху в програмуванні.

Дизайн та зовнішній вигляд веб-додатку

Ще одним важливим аспектом сайту є його зовнішній вигляд, і до цього питання треба підійти зі всією відповідальністю. У сучасному цифровому світі веб-сайт є не лише технічним продуктом, а й візитною карткою проекту, компанії чи особистого бренду. Його зовнішній вигляд та дизайн безпосередньо впливають на перше враження користувача, яке формується протягом перших кількох секунд після відкриття сторінки. Саме тому відповідальний підхід до розробки інтерфейсу та візуального стилю є критично важливим етапом створення будь-якого веб-ресурсу. По-перше, естетично привабливий та зручний інтерфейс формує довіру до ресурсу. Якщо сайт виглядає застаріло, перевантажений текстом або має хаотичне розташування елементів, користувач може автоматично зробити висновок про його ненадійність або аматорський рівень. Натомість продуманий дизайн із чіткою структурою, спокійними кольорами та логічною навігацією створює враження професіоналізму та впорядкованості. По-друге, хороший дизайн допомагає фокусувати увагу користувача. Грамотно розставлені акценти – наприклад, кнопки дії, поля введення чи заголовки – допомагають швидко зорієнтуватися на сайті та виконати потрібну дію: пройти тест, зареєструватися, ознайомитися з результатами тощо. Без чіткої візуальної ієрархії користувач може розгубитися і покинути сайт ще до того, як почне ним користуватися.

Крім того, зовнішній вигляд сайту має вплив на емоційне сприйняття. Приємні кольори, якісний шрифт, адаптивність під мобільні пристрої – усе це створює позитивну атмосферу, яка підсвідомо асоціюється з якістю контенту та зручністю користування.

Також важливо дотримуватися принципів доступності – контрастність, розмір шрифтів, достатні відступи. Це робить сайт зручним не лише для широкої аудиторії, а й для людей з порушеннями зору чи моторики.

Таким чином, дизайн – це не лише «красиві картинки», а інструмент комунікації з користувачем, який значною мірою визначає успішність веб-додатку. Відповідальний підхід до цього аспекту є обов’язковою умовою створення якісного та ефективного цифрового продукту.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Проблематика створення веб-додатків

Створення веб-додатків – це складний процес, який потребує врахування цілого ряду технічних, дизайнерських, функціональних та безпекових аспектів. Попри наявність сучасних інструментів розробки, існує низка поширених проблем, з якими стикаються як початківці, так і досвідчені розробники. Розберемо детальніше наведені проблеми та розкриємо їх:

3. **Вибір технологій**

   Одним із перших викликів є правильний вибір мови програмування, фреймворку, бази даних та архітектури застосунку. Помилки на цьому етапі можуть призвести до складної підтримки, обмеженої масштабованості або перевитрати ресурсів

4. **Планування структури сайту**

   Без чіткої структури і логіки навігації користувач може заплутатись у інтерфейсі. Часто недооцінюється важливість попереднього проектування, через що змінювати функціонал під час розробки стає складніше.

5. **Безпека**

   Питання безпеки є критично важливим, особливо коли йдеться про реєстрацію користувачів, збереження персональних даних і взаємодію з базою даних. Вразливості на кшталт SQL-ін’єкцій, XSS або CSRF можуть поставити під загрозу як дані користувачів, так і стабільність сайту.

6. **Адаптивність і кросбраузерність**

   Сайт має коректно працювати на різних пристроях (десктоп, смартфон, планшет) і у різних браузерах. Забезпечення адаптивного дизайну вимагає окремої уваги до верстки та тестування.

7. **Швидкодія та оптимізація**

Повільна робота сайту відлякує користувачів. Оптимізація запитів до бази даних, зображень, коду JavaScript і CSS є важливою частиною ефективної розробки.

8. **Підтримка та масштабування**

Багато веб-проєктів не враховують подальший розвиток. Якщо сайт не побудований з урахуванням масштабування, його буде складно оновлювати або розширювати у майбутньому.

Тож вищенаведені проблеми вимагають від розробника системного підходу, уваги до деталей та постійного вдосконалення технічних навичок. Саме тому створення навіть відносно простого сайту з тестуванням знань – це не лише написання коду, а комплексна інженерна задача.

2.2 Огляд програмних засобів для створення веб-додатків

Сучасна веб розробка має широкий спектр мов програмування, фреймворків та технологій, які дозволяють створювати як прості сайти, так і складні веб-додатки. Вибір програмних засобів залежить від багатьох чинників: складності проекту, швидкості розробки, ресурсоємності, досвіду команди тощо. Тож розглянемо інші альтернативи, які використовуються для написання веб-додатків, наведемо приклад їх використання, переваги та недоліки, і обґрунтуємо вибір **Python** і **Flask** в нашому випадку:

7. **JavaScript (Node.js + Express, React, Vue, Angular)**

JavaScript – основна мова фронтеду, але завдяки платформі Node.js вона використовується і для серверної частини. Express – популярний мінімалістичний фреймворк для Node.js. React, Vue.js та Angular застосовуються для створення інтерфейсів. **Переваги:** висока швидкість роботи, велика спільнота, активний розвиток.

Недоліки: складніша архітектура, потреба в детальному налаштуванні та захисті.

8. PHP (Laravel, Symfony)

PHP традиційно використовується для створення динамічних сайтів. Laravel – сучасний фреймворк із великою кількістю готових рішень.

Переваги: висока продуктивність, простота хостингу, багато прикладів.

Недоліки: менш сучасний підхід до проектування, слабша підтримка новітніх практик.

9. Java (Spring Boot)

Java підходить для масштабних вебзастосунків. Spring Boot забезпечує гнучкість, безпеку та продуктивність.
Переваги: стабільність, масштабованість, використовується у великих проектах.
Недоліки: складність налаштування, довгий час розробки, перевантаженість синтаксису.

10.Python (Django, Flask)

Python – мова з простою синтаксичною структурою та багатою екосистемою. Django – повноцінний фреймворк з великою кількістю готових компонентів, а **Flask** – легкий мікрофреймворк, який дає більше свободи.

Python + Flask як оптимальний вибір для невеликих проектів, тому що:

- **Швидкий старт**: **Flask** дозволяє запустити простий сайт лише за кілька рядків коду.
- **Гнучкість**: **Flask** не нав’язує структуру – розробник сам вирішує, як будувати логіку сайту.

- **Простота навчання**: **Python** – одна з найпростіших мов для новачків, але й потужна для досвідчених розробників.
- **Підтримка базових функцій**: у **Flask** легко реалізувати маршрути, форми, авторизацію, роботу з базами даних.
- **Інтеграція з іншими бібліотеками**: легко підключити Pandas, Scikit-learn, TensorFlow тощо для роботи з даними або ШІ.
- **Активна спільнота та документація**: знайти допомогу або приклад можна дуже швидко.

У порівнянні з **Django**, **Flask** легший і зручніший для невеликих сайтів, де не потрібна складна адміністративна частина або ORM «з коробки». У порівнянні з Express чи Laravel, він простіший в освоєнні, менш «галасливий» та краще підходить для прототипування.

Тож **висновком** являється те, що для реалізації невеликого веб-додатку з тестами, особистим кабінетом і базовою авторизацією ідеально підходить саме зв'язка **Python + Flask**, яка забезпечує баланс між простотою, функціональністю та швидкістю розробки.

2.3 Уведення до тематики штучного інтелекту

Штучний інтелект (ШІ) – це галузь інформатики, що займається створенням систем, здатних імітувати інтелектуальну поведінку людини. Такі системи можуть аналізувати інформацію, приймати рішення, навчатися на основі даних, розпізнавати образи, мову, текст і навіть генерувати нову інформацію.

Головна мета ШІ – створення програм або машин, які здатні виконувати завдання, що традиційно потребують участі людини.

Основою роботи сучасного штучного інтелекту є машинне навчання (machine learning, ML) – підхід, за якого комп'ютерна система не отримує чітко прописаних інструкцій, а самостійно «вивчає» закономірності на основі

наданих прикладів. Наприклад, щоб навчити модель розпізнавати емоції за текстом, їй надають велику кількість фраз, кожна з яких позначена відповідною емоцією (радість, гнів, сум тощо). На основі цих прикладів система вчиться виявляти шаблони та узагальнювати інформацію.

Іншим важливим напрямом є **нейронні мережі**, які імітують структуру та принципи роботи людського мозку. Вони складаються з шарів штучних «нейронів», які отримують, обробляють та передають інформацію. Глибокі нейронні мережі (deep learning) застосовуються в таких складних задачах, як генерація тексту, переклад мов, медична діагностика чи автономне водіння.

Коли штучний інтелект генерує відповідь на запитання, він працює на основі ймовірнісної моделі: система аналізує введене запитання, виділяє ключові слова та контекст, після чого генерує найімовірнішу відповідь, опираючись на ті знання, які були отримані під час навчання на великих масивах текстових або структурованих даних.

Наприклад, мовні моделі, такі як GPT, працюють за принципом передбачення наступного слова у фразі, виходячи з попереднього контексту. Саме завдяки цьому вони здатні відповідати на запитання, писати тексти, резюмувати інформацію та навіть вести діалог.

Таким чином, штучний інтелект є не просто інструментом автоматизації, а потужною технологією, що дозволяє комп’ютерам виконувати творчі, логічні та аналітичні завдання на рівні, близькому до людського.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Розробка структури проекту

Перш за все потрібно визначити план дій та як все буде працювати та взаємодіяти між собою. Визначимо чітку послідовність дій по порядку, для більшого розуміння також намалюємо графічну блок-схему, яка являється візуалізацією взаємодії внутрішніх компонентів веб-сайту та користувача

Щоб мати уяву того як буде виглядати структура проекту та його взаємодія між клієнтом і сервером. Розглянемо наступну блок-схему, на якій чітко зображено приклад того як має виглядати структура та яким чином буде виконуватися обмін даними між клієнтом, сервером та базою даних

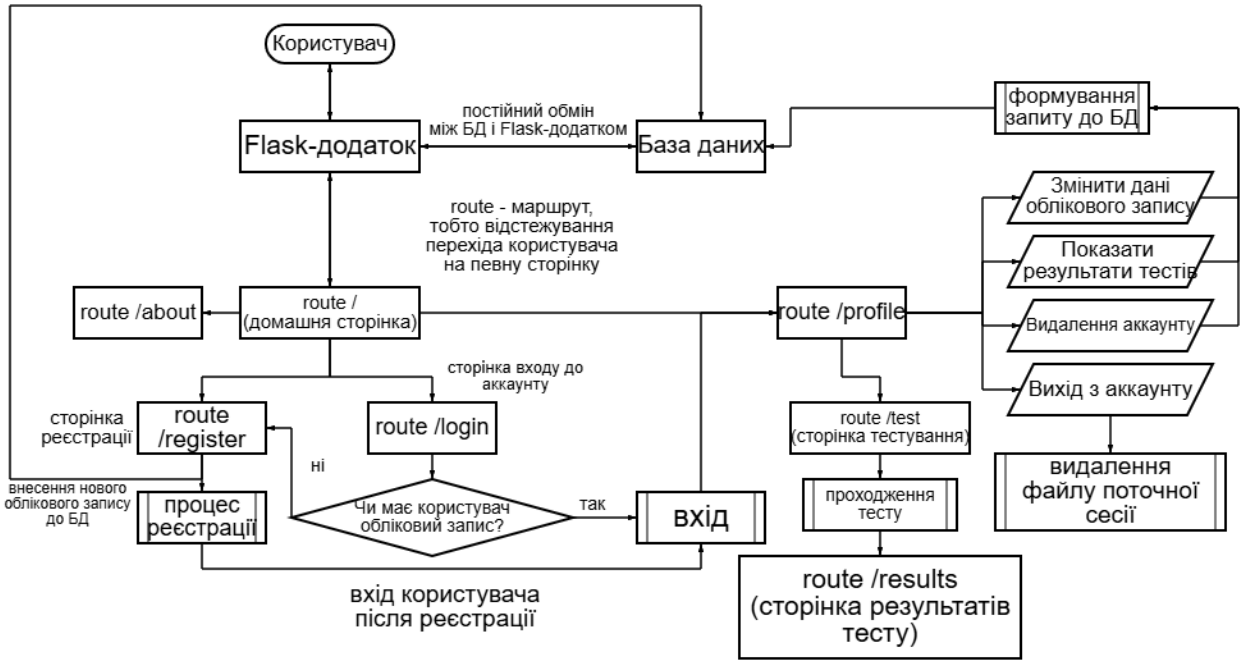


Рисунок 3.1 – Блок-схема роботи взаємодії між клієнтом, сервером та БД

Детально розглянемо, що саме зображено на блок-схемі (див. рисунок 3.1) та як працює взаємодія компонентів у вед-додатку:

Перш за все користувач ініціює запит до серверу (наприклад, перехід до головної сторінки сайту) за допомогою браузера (про це детально описано у блок-схемі далі, див. рисунок 3.2). Далі керування усіма процесами приймає на себе розгорнутий Flask-додаток. Наступним кроком користувач переходить до

головної сторінки сайту, з якої вже можна попасти до інших сторінок. Далі користувач має можливість пройти реєстрацію або увійти вже до створеного облікового запису. Процес реєстрації передбачає внесення нового облікового запису до бази даних, що зображено у блок-схемі. Після усіх процесів створення аккаунту або входу вже до існуючого, користувачу дається змога потрапити до сторінки профілю, де звідти він буде мати змогу керувати своїм обліковим записом або пройти тест та побачити результати усіх своїх тестів. Також зі сторінки профілю користувач може потрапити на сторінку проходження тесту, де після успіху результати будуть записані до бази даних та відображені на сторінці профілю. Детальніше про увесь цей процес да взаємодію буде описано у розділі 3.4.

3.2 Обґрунтування вибору вказаних технологій для розробки веб-додатку

На цьому етапі детально розглянемо вибрані технології та елементи розробки веб-додатку. У межах розробки веб-додатку для тестування знань із дисципліни «Штучний інтелект» було ухвалено рішення використовувати мову програмування **Python** і фреймворк **Flask**. Такий вибір був зумовлений кількома чинниками: простотою у використанні, гнучкістю, зрозумілим підходом до організації логіки програми, а також наявністю великої кількості документації, навчальних матеріалів та активної спільноти.

Чому саме Python?

Обираємо Python – тому що це високорівнева мова програмування з лаконічним синтаксисом і потужною стандартною бібліотекою. Вона активно використовується у сфері науки про дані, машинного навчання, веб-розробки, автоматизації та багатьох інших.

Основні переваги **Python** у контексті веб-розробки:

- **Простий у вивченні та читанні код**: завдяки чистому синтаксису розробник може зосередитися на логіки, а не на технічних деталях.

- **Широка екосистема бібліотек**: зокрема для роботи з базами даних, авторизацією, API, тестуванням тощо.
- **Сумісність із бібліотеками штучного інтелекту**: TensorFlow, scikit-learn, PyTorch, які легко інтегруються у веб-додаток.

Чому Flask?

Flask – це мікрофреймворк для веб-розробки на **Python**, створений з філософією "просте спочатку". Його головна перевага – мінімалізм. **Flask** не нав’язує структуру проекту, дозволяючи розробникові самому визначити архітектуру застосунку. Він ідеально підходить для невеликих та середніх проектів, MVP (мінімально життєздатних продуктів) і систем, де потрібна швидка розробка та кастомізація.

Згідно з офіційною документацією **Flask** (https://flask.palletsprojects.com), його основні риси:

- **Маршрутизація (routing)**: призначення URL-адресам певних функцій.
- **Підтримка шаблонів Jinja2**: дозволяє генерувати HTML динамічно.
- **Можливість підключення розширень**: наприклад, **Flask-Login** (авторизація), **Flask-SQLAlchemy** (база даних).
- **Підтримка REST API**: реалізація клієнт-серверної архітектури.

Як працює Flask-проект:

1. **Користувач (клієнт)** у браузері вводить адресу сайту або натискає кнопку, що надсилає HTTP-запит (наприклад, GET або POST).
2. **Сервер (Flask)** приймає цей запит через визначені маршрути (routes), обробляє його відповідною функцією **Python**.
3. Якщо потрібно, відбувається звернення до **бази даних** (наприклад, **SQLite** чи **PostgreSQL**) для отримання чи збереження інформації.
4. **Flask** формує відповідь (наприклад, HTML-сторінку або **JSON**) і відправляє її назад користувачеві.
5. Браузер клієнта виводить отриману інформацію.

Наприклад:

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        # логіка авторизації
        ...
    return render_template("login.html")
```

У цьому прикладі запит до /login обробляється функцією login(). Якщо метод POST – перевіряються дані форми, якщо GET – виводиться HTML-сторінка.

Як Flask взаємодіє з іншими технологіями

- **HTML/CSS/JavaScript**: **Flask** використовує шаблонізатор **Jinja2**, щоб вставляти динамічні дані у HTML-шаблони.
- **Бази даних**: можна використовувати ORM (Object-Relational Mapping), наприклад, **Flask-SQLAlchemy**, для зручної роботи з SQL-базами.
- **Форми**: розширення **Flask-WTF** дозволяє створювати форми, перевіряти їх правильність (валідація).
- **Авторизація**: за допомогою **Flask-Login** можна реалізувати вхід, вихід, обмеження доступу.
- **API**: **Flask** підтримує створення RESTful API, які можна використовувати для AJAX-запитів або взаємодії з фронтендом.

Чому Flask краще для невеликих проектів:

У порівнянні з Django, Flask:

Критерій	Flask	DJango
Вага	Легкий, мінімалістичний	Важкий, «все включно»
Структура	Гнучка, довільна	Жорстко задана структура
Крива навчання	Пологіша	Стрімкіша
Швидкість старту	Дуже швидка	Більше налаштувань
Підходить для	Невеликих проектів, API, MVP	Великих застосунків, CMS

Що таке фронтенд?

Фронтенд (Front-end) – це клієнтська частина вебсайту, тобто все, з чим взаємодіє користувач напряму:

- Структура сторінки (HTML)
- Стиль оформлення (CSS)
- Логіка поведінки елементів (JavaScript)
- Інтерактивність (анімації, переходи, реакція на кліки, тощо)

Чому фронтенд настільки важливий?

Користувач формує думку про сайт за перші 3–5 секунд. Якщо інтерфейс виглядає застарілим, незрозумілим або незручним – велика ймовірність, що людина його покине. Поганий фронтенд відлякує навіть тоді, коли "начинка" сайту (бекенд) ідеальна. Гарний інтерфейс – це не тільки краса, а логічна структура, зрозумілі кнопки, простий навігаційний шлях. Сучасний дизайн, чіткі шрифти, якісні кольори, відсутність помилок у верстці – усе це формує довіру до сайту. Якщо дизайн виглядає "дешево" або недбало, користувач може вирішити, що і сам продукт ненадійний. Фронтенд має вплив на те, куди дивиться користувач, що помічає першим, а що ігнорує. Коли користувач натискає кнопку, він очікує, що щось відбудеться: сторінка оновиться, з'явиться повідомлення, зміниться вигляд елемента. Якщо цього не відбувається – сайт здається "мертвим". Інтерактивний фронтенд створює ілюзію діалогу між системою і людиною.

Висновок

Flask у зв’язці з **Python** – це перевірене рішення для створення простих і водночас функціональних веб-застосунків. Його гнучкість, простота налаштування, сумісність із бібліотеками машинного навчання роблять його ідеальним вибором для веб-додатку з тестування знань з теми штучного

інтелекту. А завдяки активній спільноті та гарній документації, підтримка й розширення проекту у майбутньому не становитимуть труднощів.

3.3 Обґрунтування вибору середовища розробки

До вибору середовище розробки треба підходити не менш ретельніше ніж до вибору технологій для розробки ПЗ. У світі існує дуже багато середовищ розробки і кожен робить вибір покладаючись на свої переваги. Розберемо найпопулярніші та перевірені часом рішення, які використовують розробники зі всього світу. Торкнемося теми вибору редактору коду або IDE, обґрунтуємо переваги тих чи інших інструментів та зробимо порівняння.

Редактори коду та IDE для Python

1. Visual Studio Code (VSCode)

- **Тип:** редактор коду з розширеннями.
- **Переваги:**
  - Легкий, швидко запускається.
  - Потужні плагіни для **Python**, Git, Docker, Jupyter.
  - Інтеграція з WSL, SSH, Terminal.
- **Недоліки:** не має вбудованого дебагера «з коробки» – потрібні розширення.

2. PyCharm

- **Тип:** повноцінна IDE.
- **Переваги:**
  - Просунутий рефакторінг, перевірка помилок у коді, автодоповнення.
  - Вбудований інструмент для роботи з базами даних.
- **Недоліки:**
  - Важкий, може «гальмувати» на слабких машинах.
  - Повна версія платна.

3. Sublime Text

- **Переваги:** мінімалізм, швидкість.
- **Недоліки:** менше інтеграції з **Python** середовищем.

4. Jupyter Notebook

- Ідеальний для роботи з машинним навчанням, тестуванням гіпотез, візуалізацією.
- Не підходить для повноцінної серверної розробки.

Тож кожен робить вибір в сторону того, що на погляд розробника більше закриває його потреби до середовища розробки ПО та де йому більш зручніше працювати.

3.4 Алгоритм роботи програмного забезпечення

На цьому етапі детально зупинимося на темі взаємодії обміну даними між користувачем та сервером. Розберемо чітку послідовність кроків між кількома рівнями системи – як користувач ініціює запит, відправку HTTP-запиту браузером, обробку запиту **Flask** сервером, надсилання запитів до бази даних, як ОС та **Python** впливає на вищенаведені процеси та яку роль вони виконують, формування HTTP-відповіді та її потрапляння назад у браузер, і підсумковий цикл взаємодії компонентів. Повна картина виглядає наступним чином:

1. **Користувач ініціює запит**

Взаємодія починається з моменту відкривання користувачем браузеру (наприклад, Google Chrome) та вводу URL-адреси сайту. Це може бути як локальна адреса (http://localhost:5000/ або http://127.0.0.1:5000/) так і розгорнутий у хмарі веб-додаток з доменною адресою (наприклад, https://google.com/). На цьому етапі браузер створює HTTP/HTTPS-запит до веб-серверу. Запит проходить кілька етапів:

- Перетворення доменного імені на IP-адресу через DNS-запит
- Встановлення TCP-з'єднання (через триетапне "рукостискання")
- Відправлення заголовків HTTP (метод GET, POST, шлях, Cookies тощо)

2. **Обробка запиту браузером та передача даних на сервер**

Браузер реалізує клієнтську частину (frontend), яка складається з HTML (розмітка сторінки), CSS (стилізації) та JavaScript (динамічної та програмованої поведінки сторінки). У випадку надсилання форми або натискання будь-якої кнопки, JS може ініціювати асинхронний запит до серверної частини (backend), котрий вже потім буде оброблений **Flask** сервером, який оброблює логіку маршрутизації

3. **Обробка запиту Flask сервером**

Flask – це фреймворк для **Python**, який реалізує наступне:

- Вбудований веб-сервер (під час розробки)
- Роутинг (маршрутизація запитів)
- Обробку форм, cookie-файлів та сесій
- Взаємодію з базою даних
- Рендеринг HTML-шаблонів за допомогою шаблонізатору Jinja2

Наприклад:

```
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    ...
    return render_template('profile.html', user=user)
```

На цьому рівні **Flask** виконує такі функції:

- Перевіряє, який саме маршрут викликано (/login, /test, /profile)
- Отримує параметри запиту (форми, JSON, URL-параметри)
- Взаємодіє з базою даних (наприклад, за допомогою **SQLAlchemy**)
- Приймає рішення: віддати HTML-шаблон, JSON або редірект

4. **Взаємодія з ОС**

Flask працює поверх ОС. Під час розробки це часто Windows (або Windows + WSL), Linux, MacOS, тощо. ОС керує:

- Виділенням пам’яті для **Python**-процесу
- Роботою мережевих портів (наприклад, **Flask** за замовчуванням займає порт 5000)
- Роботою з ФС (запис/читання шаблонів, логів, тощо)
- Мережею та міжпроцесною взаємодією

5. **База даних**

Flask часто використовує SQLite як легку вбудовану СУБД, або PostgreSQL/MySQL у більших системах. Використовуючи ORM (Object Relational Mapper, наприклад, SQLAlchemy), сервер виконує:

- Запити на вибірку даних (SELECT REQUEST)
- Вставку нових результатів (INSERT REQUEST)
- Оновлення даних (UPDATE REQUEST)
- Видалення даних (DELETE REQUEST)

Усе це відбувається на у відповідь на дії користувача, тобто кожна дія, яка потребує читання чи модифікацію даних у БД.

6. **Формування та повернення відповіді**

Flask, отримавши дані, передає їх до шаблонізатора **Jinja2**, де відбувається вставка динамічного вмісту у HTML:

<h1>Вітаємо, {{ user.name }}</h1>

Цей HTML-документ повернеться браузеру через HTTP-відповідь (HTTP Response). Браузер, в свою чергу, аналізує відповідь і робить наступне:

- Рендерить сторінку
- Виконує JavaScript код (наприклад, валідація форм або динамічне оновлення)
- Може завантажити додаткові ресурси (зображення, шрифти, стилі, тощо)

Тобто загальна картина виглядає наступним чином:

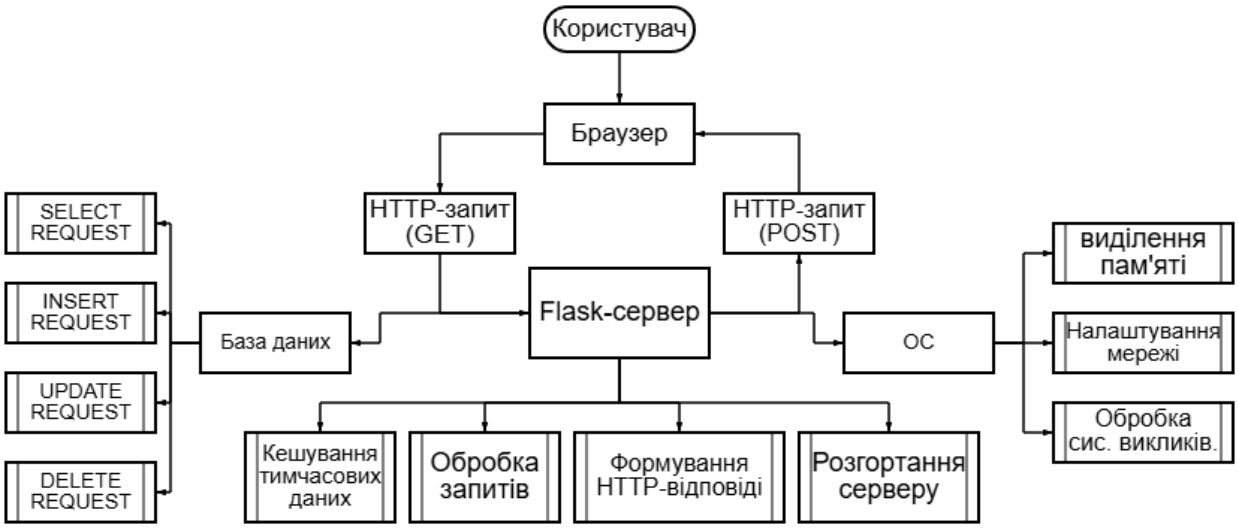


Рисунок 3.2 – Блок-схема роботи розгорнутого Flask-серверу

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Опис створення програми

Структура проекту

Початком кожного проекту повинно бути визначення структури проекту, тож визначаємо наступне:

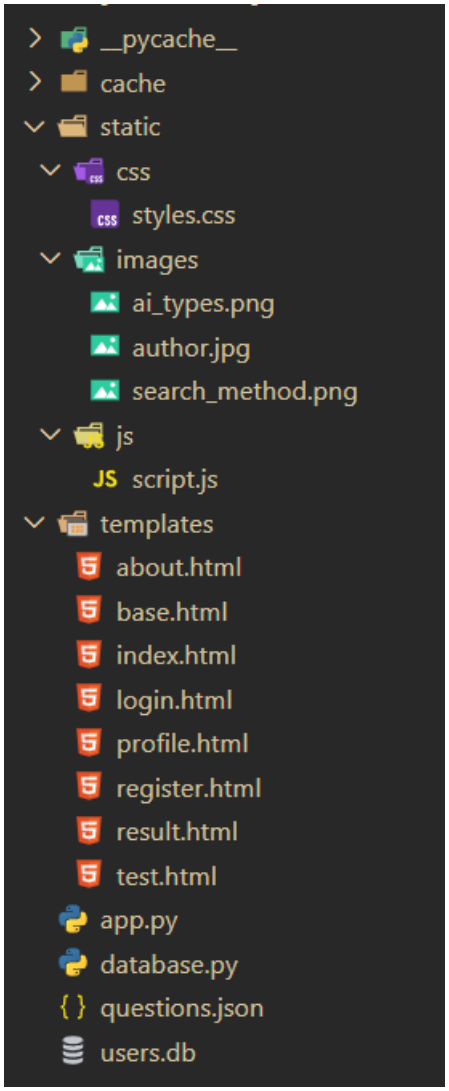


Рисунок 4.1 – Структура проекту

На цьому зображенні ми можемо спостерігати наступну структуру. Розберемо за що відповідає кожен файл та папка:

1. **«__pycache__»**

Це папка, яка створюється автоматично при запуску нашої програми. В ній зберігаються тимчасові дані (кеш), які необхідні для роботи **Python** інтерпретатору

2. **«cache»**

Теж папка для кешу, але в ній зберігається тимчасові дані для VCS

3. **«static»**

У цій папці зберігаються файли, які необхідні для коректного відображення frontend частини сайту. Тобто ми тут можемо спостерігати стилі веб-додатку, додаткові зображення, які будуть відображатися, файл JavaScript, який відповідає за поведінку динамічних елементів сайту

4. **«templates»**

Тут зберігаються HTML шаблони, які **Flask** сервер буде динамічно завантажувати як відповідь на певні дії користувача

5. **«app.py»**

Головний файл в проекті, в якому описана вся логіка нашого веб-додатку: маршрутизація, логіка проходження тестів, логіка реєстрації та входу до аккаунту користувача, генерація HTML-шаблонів

6. «database.py»

Файл, який відповідає за все, що пов’язано з базою даних: створення бази даних, ініціалізація таблиць, формування запитів до БД (створення нового користувача, зміна даних аккаунту, видалення аккаунту, запис результатів тесту, тощо)

7. «questions.json»

Цей файл є JSON-структурую, в якій в спеціальному форматі зберігаються питання до тестів, їх порядок, підказка до правильної відповіді і номер правильної відповіді. В процесі роботи веб-додатку, JSON-структура далі буде перетворена в зрозумілу для **Python** структуру даних «словник» за допомогою вбудованих алгоритмів в МП для роботи з JSON.

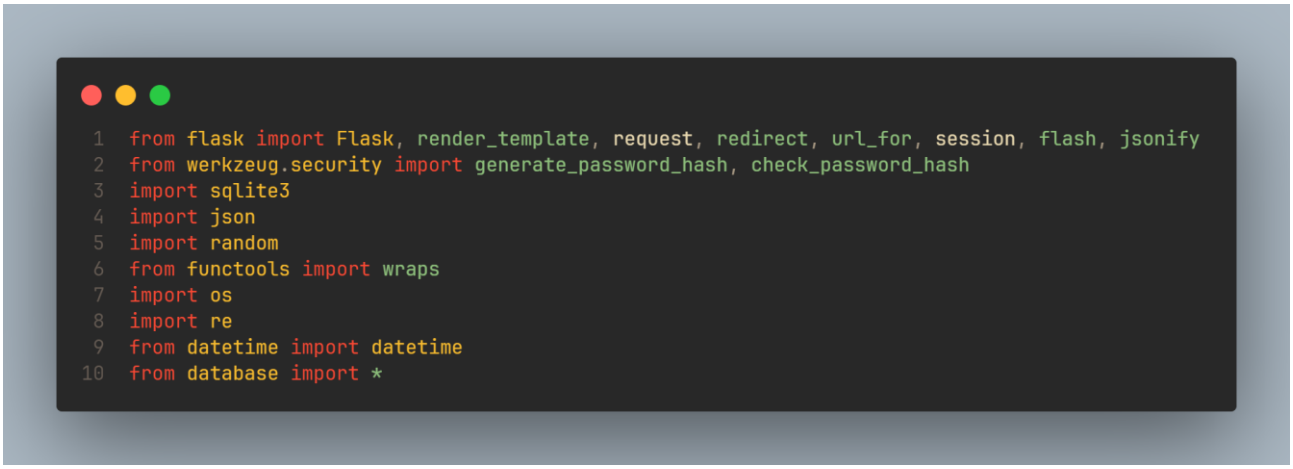
8. «user.db»

Цей файл являє собою безпосередньо базу даних, яка буде створена автоматично при першому запуску веб-серверу

Опис роботи коду Flask-серверу

Переходимо до найважливішої частини проекту – розгляд програмного коду веб-сторінки і поетапний процес створення проекту. На цьому етапі детально розглянемо кожний аспект створення веб-додатку: написання логіки backend`у, взаємодію з базою даних, реалізація дизайну сайту, логіку маршрутизації, тощо.

Перш за все переходимо до головного файлу нашого проекту – «app.py», саме в ньому і буде реалізована уся логіка backend частини веб-додатку. Для початку оголосимо усі імпорти бібліотек та фреймворків



```python
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from werkzeug.security import generate_password_hash, check_password_hash
import sqlite3
import json
import random
from functools import wraps
import os
import re
from datetime import datetime
from database import *
```

Рисунок 4.2 – Імпорти бібліотек

Детально розберемо кожен з імпортів та за що вони відповідають.

import – ключове слово в мові **Python**, яке відповідає за імпорт бібліотек до проекту.

1. **from Flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify**
    - **Flask** – основний клас для створення веб-додатку.
    - **render_template** – дозволяє підключати HTML-шаблони (наприклад, index.html).
    - **request** – об’єкт для роботи з HTTP-запитами (отримання форм, параметрів GET і POST).
    - **redirect** – перенаправлення користувача на іншу сторінку.
    - **url_for** – створює URL-адресу до функції (маршруту) за її назвою.
    - **session** – зберігає дані користувача між запитами (наприклад, чи він увійшов у систему).
    - **flash** – дозволяє відображати тимчасові повідомлення (наприклад, "Пароль невірний").
    - **jsonify** – перетворює дані у формат JSON для передачі на фронтенд.

2. **from werkzeug.security import generate_password_hash, check_password_hash**
    - **generate_password_hash** – створює хеш пароля для безпечного зберігання у базі.
    - **check_password_hash** – перевіряє, чи введений пароль відповідає збереженому хешу.
3. **sqlite3** - Модуль для роботи з вбудованою базою даних SQLite (створення таблиць, виконання запитів тощо).
4. **Json**

   Модуль для серіалізації та десеріалізації об'єктів у формат JSON – часто використовується при збереженні даних або роботі з API.
5. **random**

   Модуль для генерації випадкових чисел або вибору випадкових елементів (наприклад, випадковий порядок питань у тесті).
6. **from functools import wraps**

   wraps – декоратор, який зберігає метадані функцій при обгортанні (часто використовується для декораторів автентифікації).
7. **os**

   Модуль для роботи з операційною системою: шляхи до файлів, змінні середовища, перевірка існування файлів тощо. В проекті використовується тільки для генерації випадкового ключа сесії.
8. **re**

   Модуль регулярних виразів – дозволяє виконувати складні перевірки та пошук у тексті (наприклад, валідація email).
9. **from datetime import datetime**

   datetime – дозволяє працювати з датами та часом (наприклад, зберігати дату проходження тесту).
10. **from database import ***

    Відповідає за імпорт усіх функцій і змінних з модуля **database.py**

Після усіх імпортів почнемо налаштування **Flask** серверу. Перш за все треба створити новий об’єкт класу **Flask** та модифікувати налаштування.

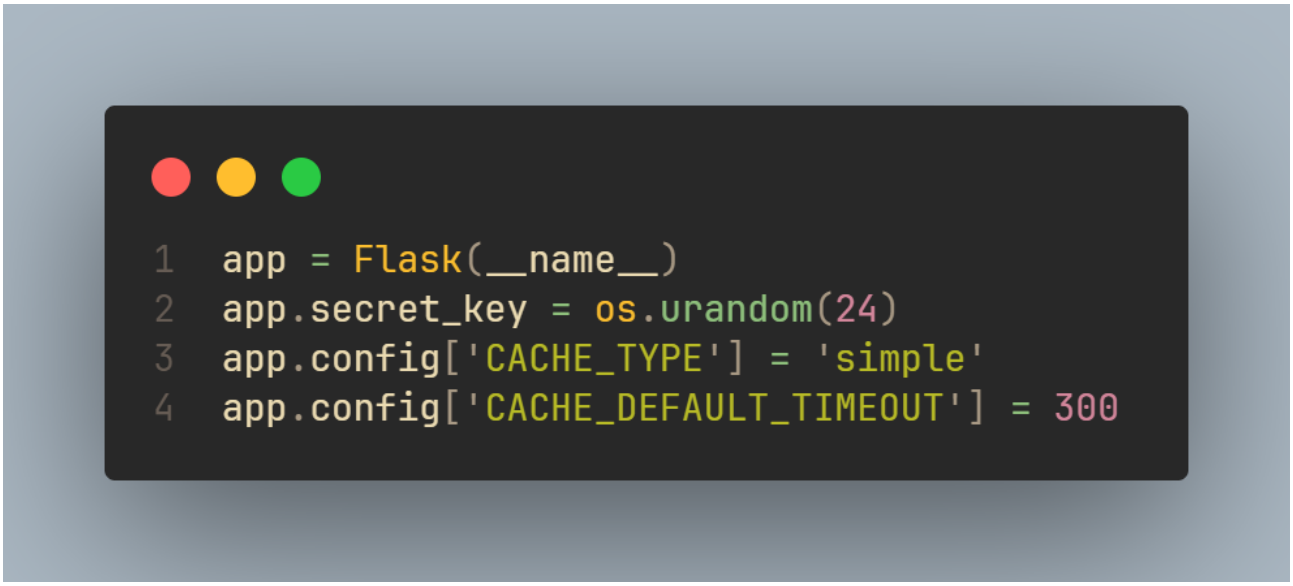


Рисунок 4.3 – Створення та налаштування об’єкту класу Flask

Розберемо, що відбувається в даному коді:

1. Створення екземпляра класу **Flask**

   Flask(__name__) означає, що **Flask** буде шукати ресурси (шаблони, статичні файли) у поточному каталозі, так як ми передали в конструктор класу спеціальну змінну **__name__**, яка вказує на ім’я поточного модуля. **Flask** використовує її, щоб знати, де шукати шаблони і статичні файли.

2. Налаштування секретного ключа для сесій

   secret_key використовується **Flask** для шифрування даних сесій, таких як **session['user_id'].**

   os.urandom(24) генерує випадкову 24-байтову послідовність байтів, яка служить ключем. Це необхідно для безпеки, щоб уникнути підробки сесій сторонніми користувачами.

3. Налаштування типу кешування

   Цей рядок визначає тип кешу для додатку. Значення «simple» означає, що буде використовуватись простий тип кешу, який зберігається у пам’яті (RAM) на стороні сервера. Це підходить для невеликих застосунків або під час розробки.

4. Тривалість кешування за замовчуванням

   Вказує, що збережені у кеші дані будуть зберігатися протягом 300 секунд (5 хвилин), якщо явно не вказано інше.

   Це дозволяє пришвидшити відповіді додатку, зменшити навантаження на базу даних або інші ресурси.

Детальніше про secret_key:

Flask зберігає інформацію про користувача (наприклад, його логін, ID) у сесії. Сесія – це словник session, який прив'язаний до конкретного користувача.

Наприклад, **session[‘user_id’] = 123**

Це означає, що користувачу буде призначено значення user_id = 123, і це збережеться в cookie. Щоб ці дані не можна було підробити, **Flask** шифрує їх за допомогою secret_key. Якщо зловмисник отримає доступ до ключа secret_key, він зможе створювати підроблені сесії, ніби-то він – інший користувач. Тому цей ключ повинен бути секретним і незмінним.

Детальніше про кеш:

Ці параметри налаштовують **Flask-Caching** – розширення, яке дозволяє зберігати дані у кеші, щоб не обчислювати або не запитувати їх кожного разу заново. Уявимо, що у вас є функція, яка повільно працює (наприклад, звертається до бази даних або зовнішнього API). Ви можете зберегти її результат у кеш:

```
from Flask_caching import Cache
cache = Cache(app)

@app.route('/slow')
@cache.cached(timeout=60)
def slow_view():
# тут якийсь довгий процес
return 'Це обчислювалося довго, але кешується!'
```

При першому виклику результат збережеться у кеші, і наступні запити протягом 60 секунд отримають збережену відповідь миттєво.

Для реалізації логічної структури веб-додатку створюємо функцію, яка завантажить наші питання до тесту у форматі **JSON** та завантажить їх до словнику. Але перш за все розглянемо, що таке JSON-структура, як вона виглядає і які її переваги.

JSON – це структура даних по типу «ключ-значення». Тобто всередині структури існує якийсь ключ або «адрес», звернувшись до якої можна отримати значення.

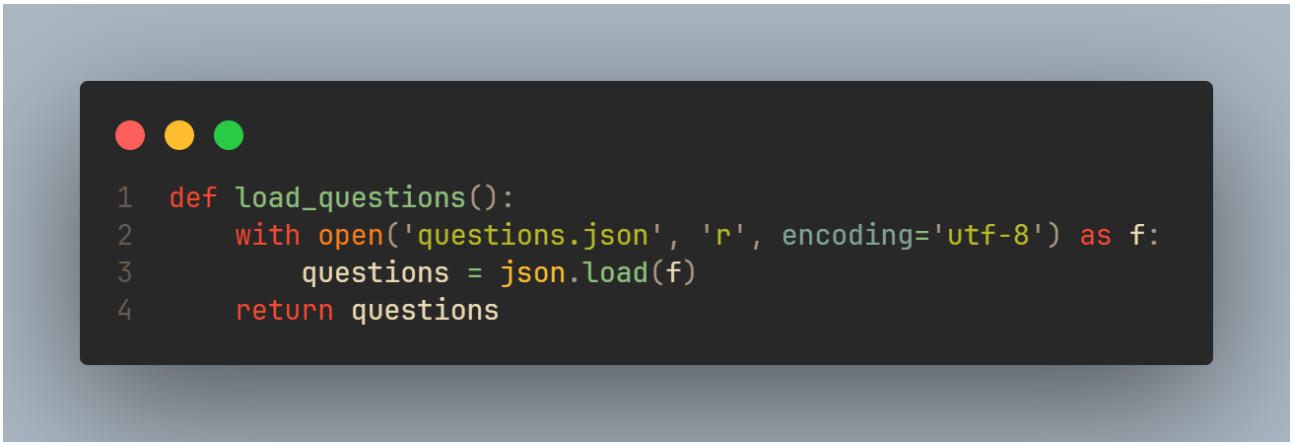
Розберемо на прикладі питань до тесту:

```
[
    {
      "id": 1,
      "question": "Який тип штучного інтелекту зображений на схемі?",
      "image": "images/ai_types.png",
      "options": [
          "Реактивний ШІ",
          "Обмежена пам'ять",
          "Теорія розуму",
          "Самосвідомий ШІ"
      ],
      "correct": 1,
      "explanation": "На зображенні показано ШІ з обмеженою пам'яттю, який використовує минулий досвід для прийняття рішень."
    },
    {
      "id": 2,
      "question": "Який із наведених прикладів є прикладом слабкого ШІ?",
      "options": [
          "Робот, який може самостійно навчатися і адаптуватися до нових ситуацій",
          "Голосовий помічник на кшталт Siri або Google Assistant",
          "ШІ, який володіє свідомістю",
          "Робот з людськими емоціями"
      ],
      "correct": 1,
      "explanation": "Слабкий ШІ спеціалізований на виконанні конкретних завдань і не володіє свідомістю."
    },
    ...
]
```

Рисунок 4.4 – Json-структура питань до тесту

На зображенні наглядно показано як виглядає JSON-структура. Перевагою зберігання подібних даних є те, що це дуже зручно, так як для додавання нових даних чи редагування вже існуючих не треба модифікувати код.

Тож визначимо функцію, яка перетворить цю структуру даних на зрозумілу для **Python**.



```python
def load_questions():
    with open('questions.json', 'r', encoding='utf-8') as f:
        questions = json.load(f)
    return questions
```

Рисунок 4.5 – Реалізація функції конвертації Json-структури в словник

Як працює ця функція:

1. Відкриваємо файл **questions.json** за допомогою функції **open()** та інтерпретуємо його як змінну **f.**
2. За допомогою бібліотеки **json** викликаємо функцію **json.load()**, яка перетворить JSON-структуру в словник та присвоїть його до змінної **questions**
3. Повертає з функції змінну

Таким чином, викликавши дану функцію, можна швидко перетворити JSON-структуру в словник та мати змогу без проблем взаємодіяти з питаннями до тесту.

Наступним кроком буде створення спеціальної функції-декоратора, яка дасть змогу запобігти потрапляння користувачка на ті сторінки, на які не можна заходити без попередньої реєстрації.

```python
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            flash('Будь ласка, увійдіть для доступу до цієї сторінки', 'warning')
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function
```

Рисунок 4.6 – Реалізація функції-декоратору

Данна функція-декоратор перевіряє наявність авторизації користувача на сайті. Якщо користувач не є авторизованим на веб-сторінці (методом перевірки наявності змінної **user_id** в сесії), то ми його перенаправляємо на сторінку авторизації. Якщо є наявність авторизації, то функція-декоратор зробить виклик оригінальної функції **f(),** тобто відобразить захищену сторінку. Декоратор **@wraps(f)** відповідає за збереження інформації і документації оригінальної функції **f(),** щоб **Flask** мав змогу коректно працювати з маршрутом та не забув як функція називалась до моменту декорації.

Далі створимо функцію, яка буде відстежувати маршрутизацію на головну сторінку, тобто до **root (‘/’).** Функція виглядає наступним чином

```python
@app.route('/')
def index():
    return render_template('index.html', logged_in='user_id' in session)
```

Рисунок 4.7 – Реалізація головної сторінки

Розберемо по порядку як вона працює:

1. За допомогою декоратору **@app.route()** відстежуємо перехід користувача на головну сторінку сайту

2. За допомогою функції **render_template()** відображаємо вміст шаблону **index.html**. Насправді існують 2 способи відображення HTML-шаблону за допомогою **Flask**:
    - Як і на прикладі, передати в функцію вже створений шаблон і повернути його з функції
    - Просто передати в **return** рядок, який містить коректний HTML-код. Даний випадок буде корисний у тих випадках, коли код містить не більше 1 рядку. У всіх інших випадках краще всього використовувати перший варіант

Наступним кроком буде реалізація функції, яка відобразить сторінку реєстрації. Вона виглядає наступним чином

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        first_name = request.form['first_name']
        group_number = request.form['group_number']
        password = request.form['password']

        if not re.match(r'^\w{3,20}$', username):
            flash('Логін повинен містити 3-20 символів (літери, цифри, _)', 'danger')
            return redirect(url_for('register'))

        if not re.match(r'^[А-Я]{2}\s\d{2,}-[а-я]$', group_number):
            flash('Невірний формат номеру групи', 'danger')
            return redirect(url_for('register'))

        conn = sqlite3.connect('users.db')
        c = conn.cursor()
        c.execute("SELECT * FROM users WHERE username=?", (username,))
        if c.fetchone():
            flash('Користувач з таким логіном вже існує', 'danger')
            conn.close()
            return redirect(url_for('register'))
        conn.close()

        add_user(username, first_name, group_number, password)
        flash('Реєстрація успішна! Тепер ви можете увійти.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')
```

Рисунок 4.8 – Реалізація сторінки реєстрації

У цій функції виконується дуже багато цікавих моментів, тому зупинимося на ній детальніше. Розберемо її покроково:

1. Вже за допомогою відомого нам декоратору відстежуємо перехід користувача на сторінку реєстрації, але в переданих параметрів функції існує ще 1 аргумент, який дозволяє відстежувати 2 HTTP-методи – GET і POST. GET-запит нам потрібен для запиту сторінки реєстрації, а POST-запит – для обробки цієї самої форми після її заповнення
2. Далі відбувається перевірка того, чи надійшов POST-запит, тобто чи була надіслана форма користувачем. Якщо ж це GET – просто відкривається сторінка реєстрації.
3. Далі починається процес зчитування даних з форми, тобто тих даних, котрі ввів користувач:

```
username = request.form['username']
first_name = request.form['first_name']
group_number = request.form['group_number']
password = request.form['password']
```

4. Потім відбувається процес валідації уведених користувачем даних за допомогою регулярних виразів. Наприклад, паттерн **«r'^\w{3,20}$'»** за допомогою алгоритмів обробки регулярних виразів перевіряє уведений рядок логіну, який повинен містити від 3 до 20 символів, та дозволяє ввести тільки литери, цифри або символ ‘_’. Якщо ж логін буде містити щось інше, то за допомогою функції **flash()** користувач буде сповіщений про то, що уведений ним логін не відповідає вимогам, і буде повернений до сторінки реєстрації. Така ж сама валідація відбувається і з номером групи.
5. Так як може буте таке, що логін був зайнятий другим користувачем, то потрібно зробити перевірку на унікальність логіну. Відбувається це шляхом підключення до бази даних:

```
conn = sqlite3.connect('users.db')
        c = conn.cursor()
        c.execute("SELECT * FROM users WHERE username=?",
(username,))
        if c.fetchone():
            flash('Користувач з таким логіном вже існує', 'danger')
            conn.close()
            return redirect(url_for('register'))
        conn.close()
```

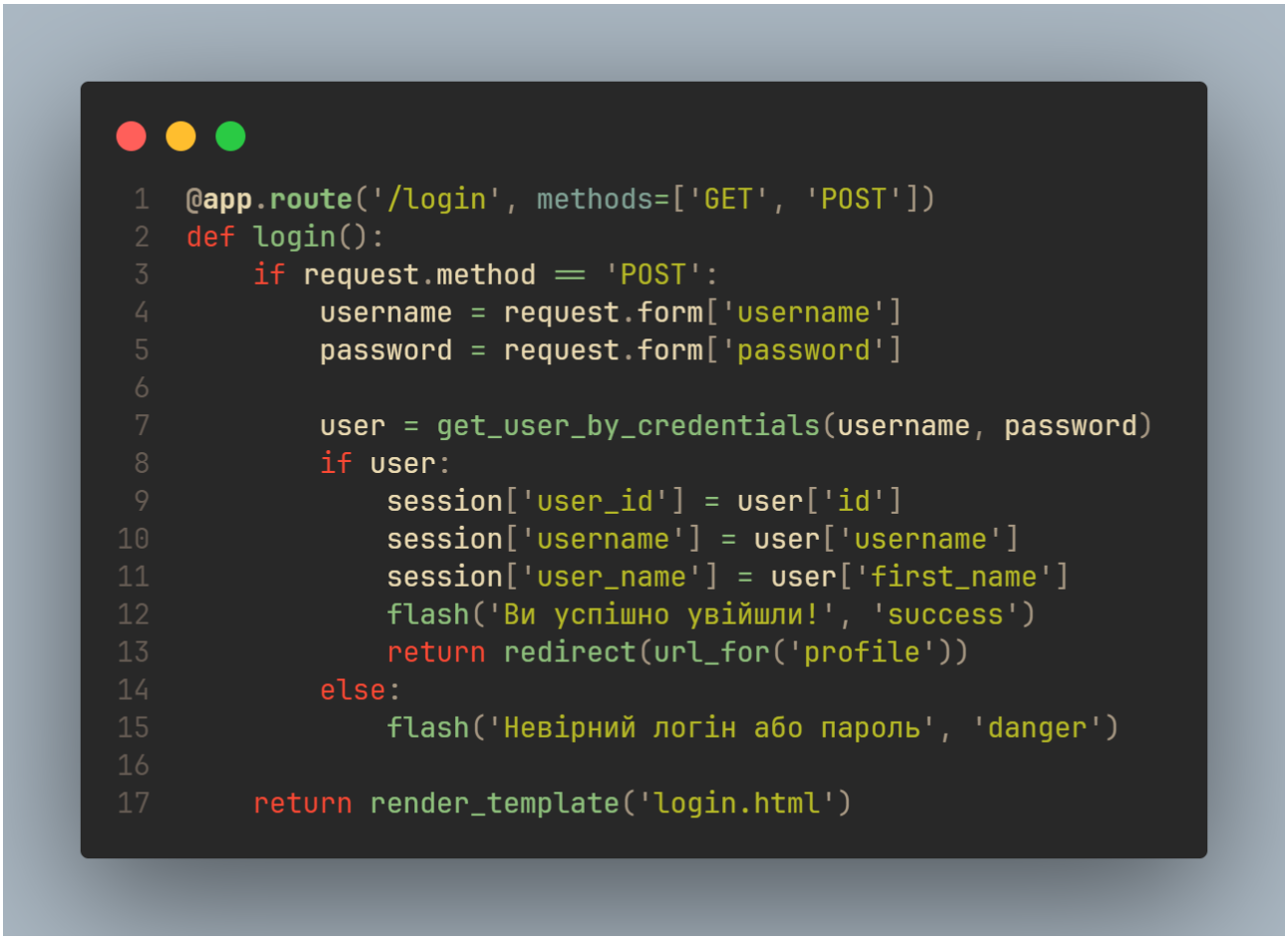
Що відбувається у цій частині коду:

- Підключення до бази даних users.db.
- Перевірка, чи існує користувач з таким логіном у таблиці users.
- Якщо користувач вже є – виводиться повідомлення і реєстрація переривається.
- Після перевірки з’єднання з БД закривається.

6. Наступним шляхом буде відбуватися додавання користувача до бази даних. Відбувається виклик функції **add_user()**, куди передаються введені користувачем дані. Сама функція описана у файлі **database.py**, до якого ми дійдемо потім
7. Після всіх попередніх маніпуляцій, користувачу буде повідомлено, що реєстрація успішна, та переадресовано до сторінки входу **/login**, де він може вже залогуватися з існуючими даними.
8. Якщо HTTP-метод – GET, то користувачу просто відобразить сторінку реєстрації

У результаті таким чином можна просто реалізувати реєстрацію нового користувача в системі веб-додатку.

Тож розберемо функцію, яка дозволить користувачу увійти вже до існуючого аккаунту

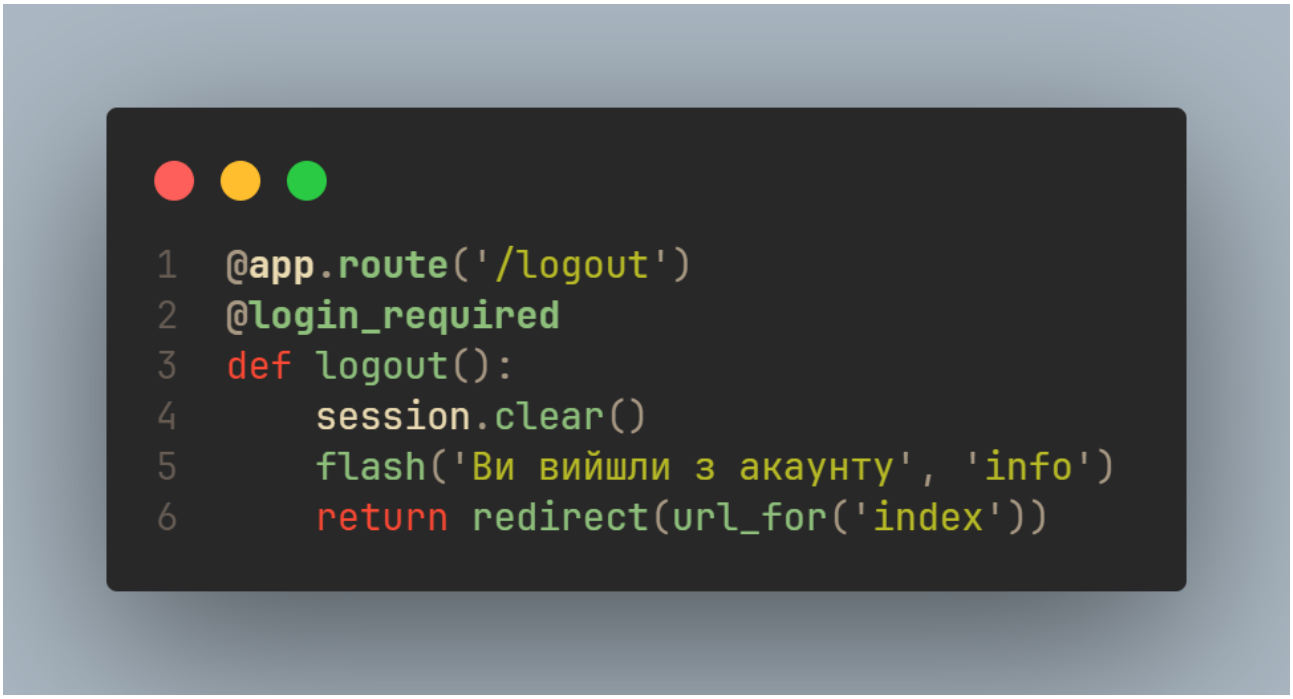


```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = get_user_by_credentials(username, password)
        if user:
            session['user_id'] = user['id']
            session['username'] = user['username']
            session['user_name'] = user['first_name']
            flash('Ви успішно увійшли!', 'success')
            return redirect(url_for('profile'))
        else:
            flash('Невірний логін або пароль', 'danger')

    return render_template('login.html')
```

Рисунок 4.9 – Реалізація сторінки входу до аккаунту

Спочатку функція отримає уведені користувачем дані, далі за допомогою запиту на БД (виклик функції **get_user_by_credentials()**) перевіряємо, чи існує такий користувач у базі даних та записуємо результат до змінної **user**. Якщо так, то у зміну сесії переносимо дані користувача, тим самим вказуємо, що відбувається процес логіну, та переадресовуємо користувача на сторінку профілю. Якщо введені дані невірні, то кажемо про це користувачу.

Далі реалізуємо функції виходу з аккаунту

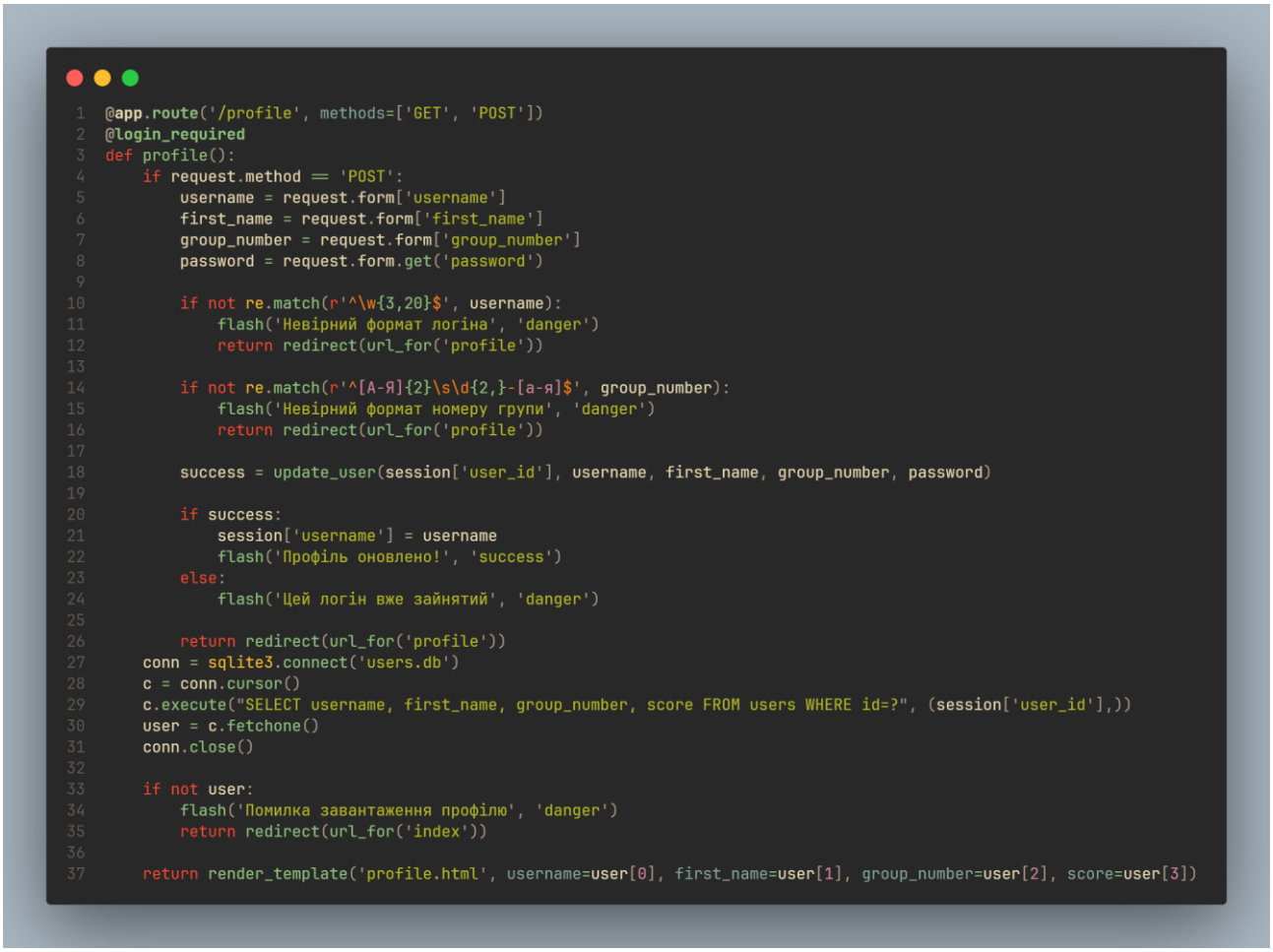


```python
@app.route('/logout')
@login_required
def logout():
    session.clear()
    flash('Ви вийшли з акаунту', 'info')
    return redirect(url_for('index'))
```

Рисунок 4.10 – Реалізація функції виходу з сесії

Ідея полягає в тому, що при потраплянні користувача на сторінку виходу (**/logout**), просто закриваємо поточну сесію та очищаємо її і переадресовуємо користувача на головну сторінку. Але в цій функції ми вперше використовуємо нами написану функцію-декоратор **@login_required**. Без цієї функції користувач міг би потрапити на сторінку виходу з аккаунту попередньо не маючи облікового запису.

Тепер розберемо функцію реалізації перегляду особистого кабінету

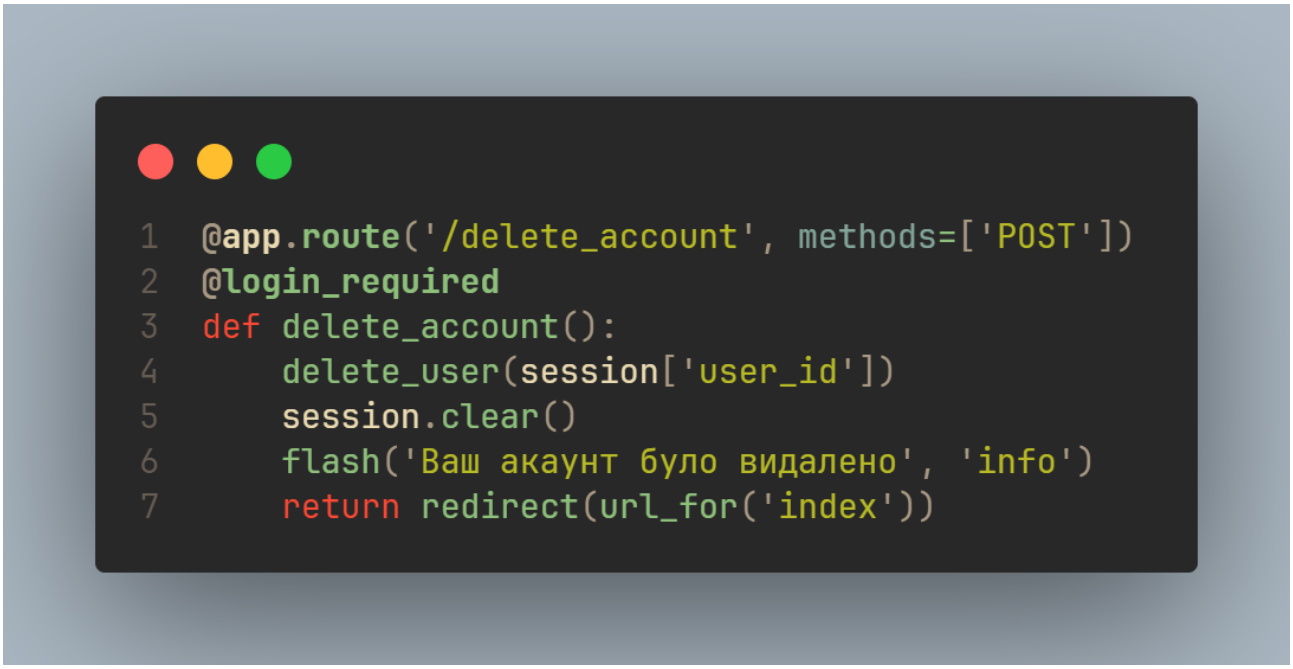


```python
@app.route('/profile', methods=['GET', 'POST'])
@login_required
def profile():
    if request.method == 'POST':
        username = request.form['username']
        first_name = request.form['first_name']
        group_number = request.form['group_number']
        password = request.form.get('password')

        if not re.match(r'^\w{3,20}$', username):
            flash('Невірний формат логіна', 'danger')
            return redirect(url_for('profile'))

        if not re.match(r'^[А-Я]{2}\s\d{2,}-[а-я]$', group_number):
            flash('Невірний формат номеру групи', 'danger')
            return redirect(url_for('profile'))

        success = update_user(session['user_id'], username, first_name, group_number, password)

        if success:
            session['username'] = username
            flash('Профіль оновлено!', 'success')
        else:
            flash('Цей логін вже зайнятий', 'danger')

        return redirect(url_for('profile'))
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute("SELECT username, first_name, group_number, score FROM users WHERE id=?", (session['user_id'],))
    user = c.fetchone()
    conn.close()

    if not user:
        flash('Помилка завантаження профілю', 'danger')
        return redirect(url_for('index'))

    return render_template('profile.html', username=user[0], first_name=user[1], group_number=user[2], score=user[3])
```

Рисунок 4.11 – Реалізація сторінки профілю

За допомогою вже відомих нам методів реалізована і функція особистого кабінету. Виконується перевірка на HTTP-метод, завантажуються особисті дані користувача. Реалізована можливість змінити особисті дані користувача за допомогою виклику функції **update()**, яку ми розберемо у частині, посвяченій роботою з базою даних.

Наступною функцією на розгляді буде видалення аккаунту



```python
@app.route('/delete_account', methods=['POST'])
@login_required
def delete_account():
    delete_user(session['user_id'])
    session.clear()
    flash('Ваш акаунт було видалено', 'info')
    return redirect(url_for('index'))
```

Рисунок 4.12 – Реалізація функції видалення аккаунту

Усередині функції викликається запит до БД про видалення облікового запису та передається зміна сесії, яка містить в собі дані про обліковий запис. Наступним кроком очищається сесія и йде переадресація користувача на головну сторінку.

Далі розберемо функцію, яка відповідає за проходження тестів. Розглянемо її покроково, детально і зрозуміло, як працює цей маршрут:

```python
@app.route('/test', methods=['GET', 'POST'])
@login_required
def test():
    questions = load_questions()

    if request.method == 'POST':
        score = 0
        incorrect_answers = []

        for question in questions:
            q_id = question['id']
            user_answer = request.form.get(f'q{q_id}')
            correct_answer = question['correct']

            if user_answer and int(user_answer) == correct_answer:
                score += 1
            else:
                incorrect_answers.append({
                    'question': question['question'],
                    'correct_option': question['options'][correct_answer],
                    'explanation': question.get('explanation', 'Немає додаткового пояснення')
                })

        update_user_score(session['user_id'], score)

        return render_template('result.html',
                               score=score,
                               total=len(questions),
                               incorrect_answers=incorrect_answers)

    shuffled_questions = random.sample(questions, len(questions))
    for question in shuffled_questions:
        options = question['options']
        correct_index = question['correct']

        correct_answer = options[correct_index]

        random.shuffle(options)

        question['correct'] = options.index(correct_answer)

    return render_template('test.html', questions=shuffled_questions)
```

Рисунок 4.13 – Реалізація логіки тесту

Що відбувається в даній функції:

1. Вже відоме відстежування маршруту до сторінки тестування та обробка HTTP-методів
2. За допомогою створеної функції **load_questions()** завантажуємо наші питання у змінну **questions.**
3. Якщо HTTP-метод – POST, то ініціюємо змінні результатів
    - score – лічильник правильних відповідей

- incorrect_answers - список, куди зберігатимуться всі помилкові відповіді користувача разом з поясненнями.

4. За допомогою циклу **for** перебираємо всі запитання в тесті, де
   - id – унікальний ідентифікаційний номер питання
   - user_answer - ідповідь, яку користувач обрав із HTML-форми (**request.form.get(...)**).
   - correct_answer – правильний варіант (індекс) із JSON-даних
5. Далі відбувається процес порівняння відповідей
   - Якщо відповідь є і вона збігається з правильною – додаємо 1 бал
   - Інакше:
     - Додаємо в список помилок сам текст питання
     - Правильну відповідь
     - Пояснення
6. Далі зберігаємо результат за допомогою функції **update_user_score()** (також буде описано в розділі, присвяченому роботою з базою даних)
7. Повертаємо результат за допомогою **render_template().** Показуємо шаблон **result.html,** куди передаємо к-сть правильних відповідей, загальну к-сть питань та список неправильних відповідей з поясненнями
8. Логіка GET-запиту (тобто коли користувач тільки відкрив сторінку тесту) виглядає наступним чином:
   - За допомогою бібліотеки **random** перемішуємо питання у випадковому порядку для уникнення запам’ятовування послідовності питань
   - Також виконуємо перемішення варіантів відповідей:
     - Для кожного питання перемішуються варіанти

- Запам’ятовується правильна відповідь до моменту перемішування
- Після перемішування оновлюється індекс правильної відповіді. Це важливо, бо інакше після перемішування правильна відповідь вказувала б на неправильний варіант

- Рендериться шаблон **test.html,** який отримує усі питання в перемішаному варіанті

Таким чином ми створили логіку проходження тесту, де ми уникнули можливості користувача просто запам’ятати послідовність відповіді

Перейдемо до останніх двох функцій

```
@app.route('/about')
def about():
    return render_template('about.html')

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Рисунок 4.14 – Реалізація сторінки «Про автора»

Перша функція відстежує перехід користувача на сторінку «Про автора» та відображає її. Код після функції це умовна конструкція, яка перевіряє, чи цей файл запускається безпосередньо (тобто за допомогою команди **Python3 app.py**), а не імпортується з іншого модуля. Ця конструкція є стандартним

способом запуску **Flask**-додатку в середовищі **Python**. Якщо файл імпортується в іншому коді (наприклад, за допомогою імпорту), тоді й тіло усередині блоку **if** не виконується. Це зручно для розділення «запуску додатку» від «імпорту як модуля». Наступним йде виклик функції **init_db(),** яка ініціює базу даних та створює таблиці. Вона гарантує, що база даних підготовлена для використання ще до моменту запуску сервера. Рядок **app.run(debug=True)** відповідає за запуск вбудованого **Flask**-сервера. Аргумент усередині функції означає увімкнення режиму розробника, тобто:

1. Дає змогу автоматичного перезапуску серверу, коли змінюється код, що є дуже зручно під час розробки
2. Виводяться детальні повідомлення про помилки у браузері, тобто реалізується повноцінний traceback.

Але у «продакшн» середовищі режим розробника потрібно вимикати, так як це не є безпечно і може відкрити внутрішні дані сервера.

Опис роботи коду запитів до бази даних

Тепер переходимо до не менш важливою частини проекту – роботою з базою даних. На цьому етапі ми детально розберемо взаємодію **Flask**-сервера з базою даних, розберемо як влаштована СУБД SQLite3, як виконуються запити до неї та детально розберемо, як це влаштовано у проекті.

Почнемо з питання, що таке база даних, мова SQL, СУБД та як вони працюють.

Бази даних – це системи, які дозволяють зберігати, організовувати та ефективно працювати з великою кількістю інформації.

Які бувають бази даних:

1. Реляційні бази даних (RDBMS)

   Дані зберігаються в таблицях, де є стовпці (поля) та рядки (записи). Між таблицями можна встановлювати зв’язки. Найпоширеніший тип

   Приклади: SQLite, MySQL, PostgreSQL, Oracle, MS SQL Server.

2. Документно-орієнтовані бази
   Дані зберігаються у вигляді документів (найчастіше – JSON). Гнучкіші, ніж реляційні
   Приклади: MongoDB, CouchDB.
3. Ключ-значення (Key-Value stores)
   Кожне значення зберігається з унікальним ключем. Дуже швидкі, але не завжди підходять для складних запитів.
   Приклади: Redis, Amazon DynamoDB.
4. Графові бази
   Призначені для зберігання об'єктів і зв'язків між ними. Чудово підходять для соцмереж, рекомендацій.
   Приклад: Neo4j

Що таке SQL:

SQL (Structured Query Language) – це мова для спілкування з реляційними базами даних. За допомогою SQL ми можемо:

1. Створювати таблиці: CREATE TABLE
2. Додавати дані: INSERT INTO
3. Шукати потрібні записи: SELECT
4. Оновлювати інформацію: UPDATE
5. Видаляти дані: DELETE

SQL – це стандарт, який підтримується всіма реляційними СУБД, хоч і з невеликими відмінностями.

Що таке СУБД:

СУБД (система управління базами даних) – це програма, яка «знає», як працювати з базою, зберігати дані, забезпечувати безпеку та доступ до інформації.

СУБД виконує наступні завдання:

1. відповідає за збереження і цілісність даних

2. дозволяє виконувати SQL-запити
3. слідкує, щоби все працювало швидко і безпечно
4. підтримує одночасну роботу багатьох користувачів

Тож чому вибір в проекті пав саме на SQLite? Щоб відповісти на це питання, потрібно дізнатися як вона працює

SQLite – це легка, вбудована СУБД. Вона не потребує окремого сервера: вся база зберігається у звичайному файлі. Вона дуже проста в інтеграції та налаштуванні, саме тому ідеально підходить для маленьких проектів, сайтів, прототипів, навчальних задач. Особливостями SQLite є:

1. Простота: не потрібно нічого встановлювати окремо.
2. Локальне зберігання: все в одному .db файлі.
3. Швидкість: дуже швидка для невеликих обсягів даних.
4. Відкритий код: повністю безкоштовна та кросплатформна.

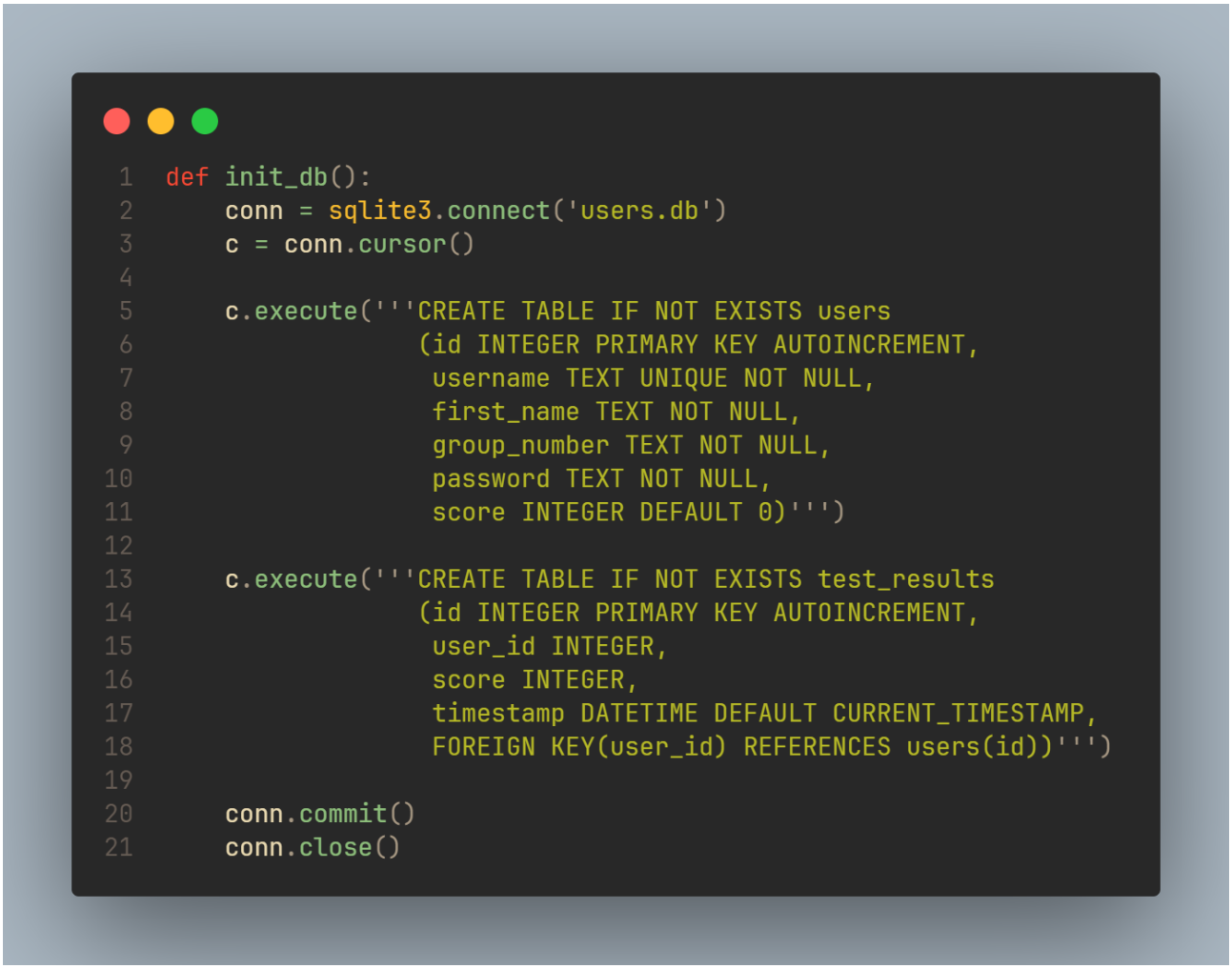
Вибір в сторону SQLite був зроблений, тому що проект має на увазі лише збереження тестів і їх результатів та реєстрацію користувачів, тому для такого завдання підходить щось легке, як, наприклад, SQLite.

Тож почнемо розбирати робото з базами даних на прикладі проекту. Перш за все зробимо усі потрібні імпорти, які знадобляться нам для роботи з SQLite

Рисунок 4.15 – Імпорти для бази даних

Перший імпорт це безпосередньо включення до нашого проекту бібліотеку для роботи з СУБД. Другій імпорт потрібен нам для роботою з алгоритмами шифрування даних, до цього пізніше

Наступним кроком оголосимо основну функцію, яка відповідає за створення бази даних та ініціалізацію усіх потрібних для нас таблиць



```python
def init_db():
    conn = sqlite3.connect('users.db')
    c = conn.cursor()

    c.execute('''CREATE TABLE IF NOT EXISTS users
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  username TEXT UNIQUE NOT NULL,
                  first_name TEXT NOT NULL,
                  group_number TEXT NOT NULL,
                  password TEXT NOT NULL,
                  score INTEGER DEFAULT 0)''')

    c.execute('''CREATE TABLE IF NOT EXISTS test_results
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  user_id INTEGER,
                  score INTEGER,
                  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                  FOREIGN KEY(user_id) REFERENCES users(id))''')

    conn.commit()
    conn.close()
```

Рисунок 4.16 – Реалізація функції ініціалізації бази даних

Що відбувається всередині функції:

1. Виконується підключення до бази даних з назвою **users.db** (ця база даних буде автоматично створена при першому запуску програми, якщо такої не існувало до)
2. **conn** – це об’єкт з’єднання, через який ми можемо надсилати запити до БД. Тобто цей рядок створює об’єкт курсора, який дозволяє виконувати запити.

3. Далі відбувається процес створення першої таблиці **users,** яка буде зберігати особисті дані користувача шляхом SQL-запиту. Наступна таблиця має такі значення:
    - **Id -** Унікальний ідентифікатор користувача. Число, що автоматично збільшується при кожному новому записі.
    - **Username -** Логін користувача. Повинен бути унікальним і обов'язковим.
    - **First_name -** Ім’я користувача. Не може бути порожнім.
    - **Group_number -** Номер групи (наприклад, навчальна група). Обов’язкове поле.
    - **Password -** Хешований пароль. Не зберігається у відкритому вигляді.
    - **Score -** Рахунок або бали за тести. За замовчуванням – 0.

Ключове слово «**CREATE TABLE IF NOT EXIST**» гарантує, що таблиця не буде створена повторно, якщо вона вже є в БД.

Друга таблиця **test_results** містить наступні поля:

1. **Id -** Унікальний ідентифікатор результату.
2. **User_id -** Зв’язок з таблицею users. Вказує, хто проходив тест.
3. **Score -** Кількість балів, які набрав користувач за тест.
4. **Timestamp -** Дата і час проходження тесту. Автоматично встановлюється на момент запису.
5. **FOREIGN KEY (user_id) -** Це зовнішній ключ, який забезпечує зв’язок між результатами та таблицею користувачів. Гарантує, що user_id відповідає існуючому id з таблиці users

Далі йде виклик методу **conn.commit(),** який зберігає всі зміни, внесені через курсор у файл БД. Якщо не зробити цей виклик, то в такому випадку створені

таблиці залишаться лише в оперативній пам’яті і не збережуться. Виклик **conn.close()** закриє зв’язок з БД. Це необхідно робити, щоб звільнити ресурси, уникнути блокування файлу іншими процесами та гарантувати, що все було записано.

Далі оголосимо функцію, що буде заносити нового користувача до БД.

```
def add_user(username, first_name, group_number, password):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    hashed_password = generate_password_hash(password)
    c.execute("INSERT INTO users (username, first_name, group_number, password) VALUES (?, ?, ?, ?)",
              (username, first_name, group_number, hashed_password))
    conn.commit()
    conn.close()
```

Рисунок 4.17 – Реалізація функції додання нового користувача

Так само ми створюємо зв’язок з БД, як і в попередній функції. Наступним кроком буде шифрування заданого користувачем паролю. Це необхідно робити в цілях безпеки. Так як в такому випадку БД буде зберігати тільки хеш-символи, які оброблюються спеціальним алгоритмом бібліотеки. Тобто на сайті користувач вводить свій пароль, який далі передається в бібліотеку шифрування. Далі цей пароль за допомогою спеціального алгоритму (зазвичай це PBKDF2 + SHA256) перетвориться в набір символів та запишеться до бази даних. Якщо користувач захоче увійти вже до існуючого облікового запису, то за допомогою спеціального алгоритму введений користувачем пароль пройде той самий процес, а далі вже буде відбуватися порівняння двох хеш-наборів. Якщо 2 хеш-набори співпадають, то пароль вірний. Таким чином, навіть якщо зловмисник отримає доступ до БД, то він ніяк не зможе дізнатися паролі користувачів. Тож після шифрування та запису пароля, ми створюємо SQL-запит до БД та вставляємо дані нового користувача в таблицю.

Наступним кроком ми розберемо функцію, яка буде виконувати автентифікацію користувача – порівнювати уведені дані.

```python
def get_user_by_credentials(username, password):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute("SELECT * FROM users WHERE username=?", (username,))
    user = c.fetchone()
    conn.close()

    if user and check_password_hash(user[4], password):
        return {
            'id': user[0],
            'username': user[1],
            'first_name': user[2],
            'group_number': user[3],
            'score': user[5]
        }
    return None
```

Рисунок 4.18 – Реалізація функції автентифікації користувача

Що відбувається в функції:

1. Встановлюється зв’язок з БД
2. Робиться SQL-запит на взяття даних користувача по імені
3. Виклик **fetchone()** поверне один результат (рядок у вигляді кортежу), або None, якщо користувача не знайдено
4. Виконується перевірка пароля (процес був описаний в попередній функції)
5. Якщо пароль вірний, то функція переходить до процесу повернення даних користувача у вигляді словника

Таким чином реалізована функція пошуку користувача

Далі розглянемо функцію, яка відповідає за оновлення даних користувача

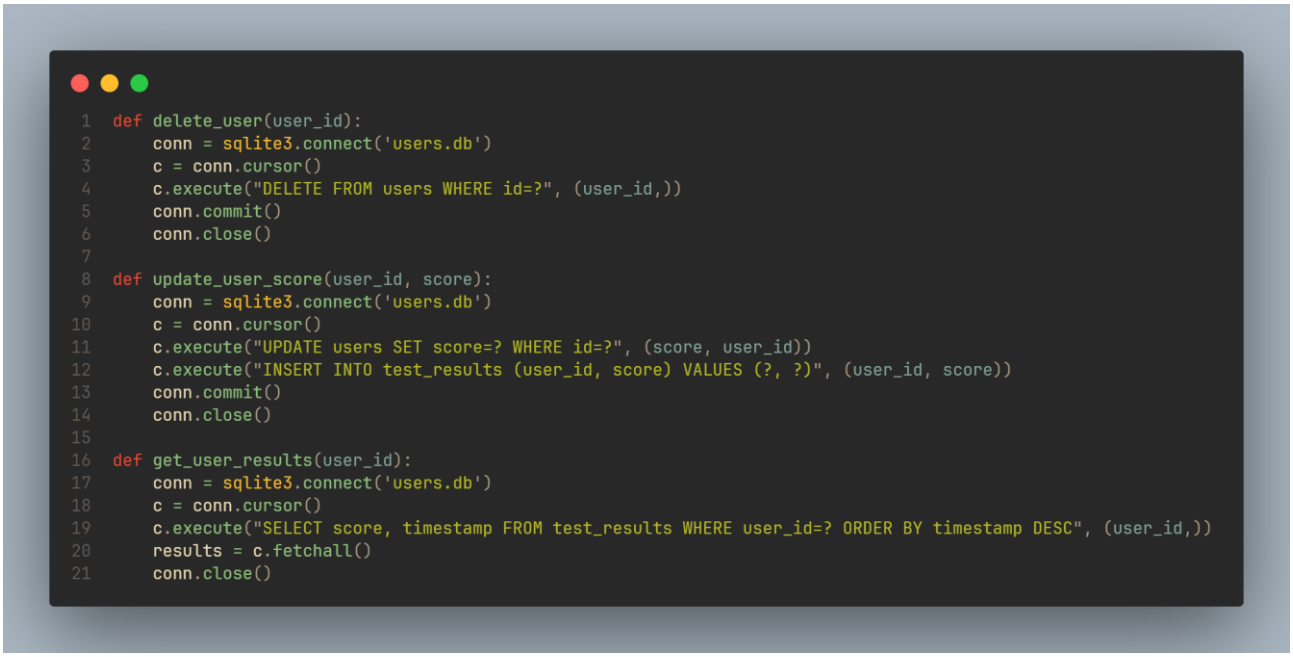
```python
def update_user(user_id, username, first_name, group_number, password=None):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()

    c.execute("SELECT id FROM users WHERE username = ? AND id != ?", (username, user_id))
    if c.fetchone():
        conn.close()
        return False

    if password:
        hashed_password = generate_password_hash(password)
        c.execute("""UPDATE users SET
                    username = ?,
                    first_name = ?,
                    group_number = ?,
                    password = ?
                    WHERE id = ?""",
                (username, first_name, group_number, hashed_password, user_id))
    else:
        c.execute("""UPDATE users SET
                    username = ?,
                    first_name = ?,
                    group_number = ?
                    WHERE id = ?""",
                (username, first_name, group_number, user_id))

    conn.commit()
    conn.close()
    return True
```

Рисунок 4.19 – Реалізація функції оновлення даних користувача

Етапи роботи функції:

1. Встановлення зв’язку з БД
2. Виконання SQL-запиту, який перевіряє, чи існує інший користувач з таким самим логіном
3. Далі є 2 варіанти розвитку подій:
    1. Якщо користувач змінює дані разом з паролем. В такому випадку процес зміни даних також буде включати процес шифрування нового паролю
    2. Якщо пароль не змінюється, то просто оновлюються всі інші дані
4. Запис змін

Наступним кроком розглянемо реалізацію одразу трьох функцій, які відповідають за видалення користувача з БД, оновлення результатів тесту та отримання результатів тесту



```python
def delete_user(user_id):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute("DELETE FROM users WHERE id=?", (user_id,))
    conn.commit()
    conn.close()

def update_user_score(user_id, score):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute("UPDATE users SET score=? WHERE id=?", (score, user_id))
    c.execute("INSERT INTO test_results (user_id, score) VALUES (?, ?)", (user_id, score))
    conn.commit()
    conn.close()

def get_user_results(user_id):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute("SELECT score, timestamp FROM test_results WHERE user_id=? ORDER BY timestamp DESC", (user_id,))
    results = c.fetchall()
    conn.close()
```

Рисунок 4.20 – Реалізація функцій: видалення користувача, оновлення результату тесту та отримання результату

Так як ці функції працюють однаково в сенсі послідовності, то немає сенсу розбирати кожну окрему. У всіх трьох функцій одна логіка:

1. Встановити зв’язок з БД
2. Виконати потрібний SQL-запит
3. Закрити з’єднання з БД

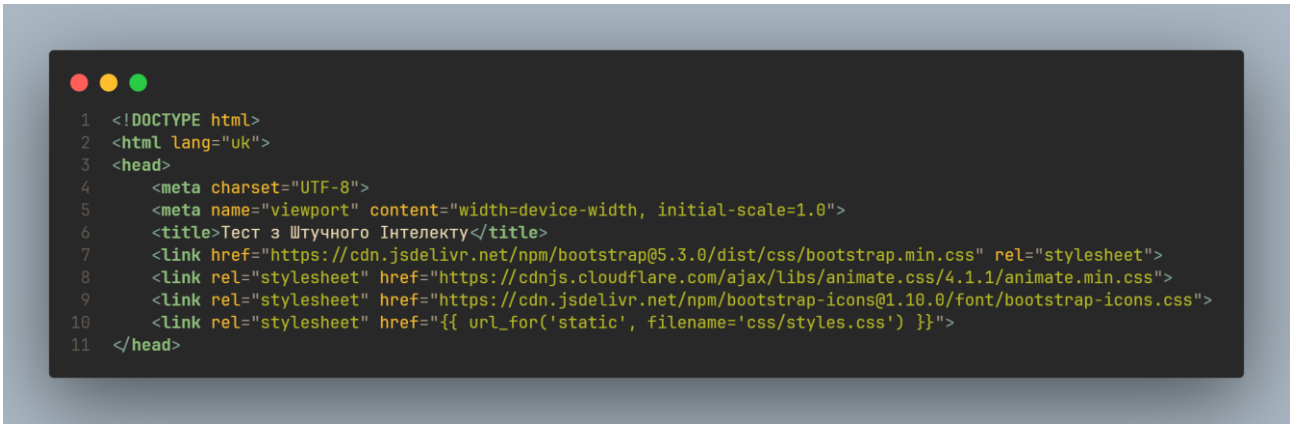
У процесі детального розбору коду, що стосується роботи з базою даних в веб-проекті на **Flask**, ми досягнули таких важливих результатів і зрозуміли, що таке БД, СУБД, SQL, розібрали приклад запитів, створення нових таблиць та їх оновлення на реальному прикладі. Також торкнулися важливої теми шифрування даних та роботу алгоритму шифрування.

Опис роботи front-end частини веб-додатку

У цьому розділі розберемо як була реалізована фронтенд частина веб-додатку.

Для початку розпочнемо огляд того, як реалізовані HTML-шаблони. Перш за все оглянемо шаблон, який є «базовим» для всіх інших. Тобто в ньому будуть присутні елементи, які будуть відображені на всіх інших сторінках сайту. Таким чином від цього шаблону можна буде успадковуватися, щоб уникнути повторення коду.

Так як файл є достатньо великим, будемо розбирати його частинами



```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Тест з Штучного Інтелекту</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.0/font/bootstrap-icons.css">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
```

Рисунок 4.21 – Заголовок HTML-документу

Що містить ця частина HTML-шаблону:

1. **<!DOCTYPE html> -** Це декларація типу документа. Вказує браузеру, що перед ним HTML5-документ. Вона забезпечує правильне відображення сторінки в усіх сучасних браузерах.
2. **<html lang="uk"> -** Початок HTML-документу з атрибутом мови, який встановлює українську мову на веб-додатку
3. **<head> -** початок «голови» документу. Це частина, де підключаються мета-дані, стилі, назва сторінки, скрипти, тощо. Цей вміст не відображається напряму на сторінці, але критично важливий для її роботи

Розберемо, що відбувається усередині **head** блоку:

Спочатку ми встановлюємо кодування символів у форматі UTF-8, завдяки яким можна відображати українські літери та інші національні символи. Далі налаштовуємо адаптивність сайту, тобто даємо змогу сторінці правильно відображатись на різних екранах (наприклад, на екрані телефону чи планшету). Наступним кроком встановлюємо назву сторінки, яка буде відображатися у

вкладці браузера. Також цей елемент використовується пошуковими системами для індексації сторінки в мережі. Далі відбувається дуже важлива річ – підключення стилів і бібліотек, завдяки яким ми швидко та просто зробимо дизайн сайту за допомогою вже написаних елементів. За це відповідає CSS-фреймворк – Bootstrap. Він містить дуже багато вже готових стилів, які треба просто імортувати до проекту та використовувати. Це прискорює розробку дизайну в дуже багато разів.

Тож наступним кроком розберемо шматок коду, який наглядно буде показувати, яким чином в проекті використовується Bootstrap і Jinja2

```
<body class="d-flex flex-column min-vh-100">
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary shadow-sm">
        <div class="container">
            <a class="navbar-brand animate__animated animate__pulse" href="/">
                <i class="bi bi-robot"></i> AI Тест
            </a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav me-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="/"><i class="bi bi-house"></i> Головна</a>
                    </li>
                    {% if 'user_id' in session %}
                    <li class="nav-item">
                        <a class="nav-link" href="/test"><i class="bi bi-question-circle"></i> Тест</a>
                    </li>
                    {% endif %}
                    <li class="nav-item">
                        <a class="nav-link" href="/about"><i class="bi bi-person"></i> Про автора</a>
                    </li>
                </ul>
                <ul class="navbar-nav">
                    {% if 'user_id' in session %}
                    <li class="nav-item dropdown">
                        <a class="nav-link dropdown-toggle" href="#" id="profileDropdown" role="button" data-bs-toggle="dropdown">
                            <i class="bi bi-person-circle"></i> {{ session['user_name'] }}
                        </a>
                        <ul class="dropdown-menu dropdown-menu-end">
                            <li><a class="dropdown-item" href="/profile"><i class="bi bi-person"></i> Профіль</a></li>
                            <li><hr class="dropdown-divider"></li>
                            <li><a class="dropdown-item" href="/logout"><i class="bi bi-box-arrow-right"></i> Вийти</a></li>
                        </ul>
                    </li>
                    {% else %}
                    <li class="nav-item">
                        <a class="nav-link" href="/login"><i class="bi bi-box-arrow-in-right"></i> Увійти</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/register"><i class="bi bi-person-plus"></i> Реєстрація</a>
                    </li>
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>
```

Рисунок 4.22 – Реалізація навігаційної панелі

Почнемо з того, що визначимо, що таке Bootstrap і Jinja2:

Bootstrap - це популярний CSS-фреймворк, який дозволяє швидко створювати красиві, адаптивні й зручні інтерфейси без ручної верстки всіх стилів.

У фрагментах коду видно, що для надання стилів блокам коду використовуються такі елементи в атрибутах, як: navbar, navbar-dark, bg-primary, shadow-sm – це все класи, які вже описані у фреймворку Bootstrap та оформляють навігаційну панель. Тож це дозволяє нам не писати вручну CSS-стилі. Тож завдяки цьому наша навігаційна панель буде виглядати наступним чином:

Рисунок 4.23 – Вигляд навігаційної панелі

І в процесі її розробки ми не написали жодного CSS коду.

Переходимо до Jinja2:

Jinja2 - це шаблонізатор, що дозволяє вбудовувати **Python**-логіку прямо в HTML-код. Він дає змогу динамічно відображати дані, умовно показувати/ховати елементи, циклічно створювати контент тощо.

Розберемо наступний код:

```
{% if 'user_id' in session %}
                    <li class="nav-item">
                        <a class="nav-link" href="/test"><i class="bi bi-question-circle"></i> Тест</a>
                    </li>
                    {% endif %}
```

Рисунок 4.24 – Приклад використання Jinja2

Тут ми можемо спостерігати, що в HTML-коді використовуються конструкції по типу {{}} та {%%} – це і є використання Jinja2. За допомогою цього **Flask**-сервер може динамічно вставляти блоки коду та маніпулювати цим кодом в HTML, який потім буде відображатися користувачу в реальному часі. У цьому прикладі це використано як:

«Якщо у сесії є наявність поля **user_id,** то ми відобразимо користувачу блок коду усередині».

Тож таким чином за допомогою Bootstrap та Jinja2 і була написана уся частина frontend.

Наприклад, наступна сторінка з тестами створена наступним чином:

```
{% extends "base.html" %}

{% block content %}
<div class="d-flex justify-content-center align-items-center">
    <h1 class="text-center">Виберіть тему тесту із запропонованих</h1>
</div>

<div class="row justify-content-center animate__animated animate__fadeIn">
    <div class="row g-4 justify-content-center">

        <div class="col-md-4 animate__animated animate__fadeInLeft">
            <div class="card h-100 shadow-sm">
                <div class="card-body text-center">
                    <i class="bi bi-list-stars feature-icon text-primary"></i>
                    <h2 class="card-title">Тема 1</h2>
                    <p class="card-text">Основні поняття</p>
                    <button type="button" class="btn btn-primary">
                        <i class="bi bi-arrow-return-right"></i> Розпочати
                    </button>
                </div>
            </div>
        </div>

        <div class="col-md-4 animate__animated animate__fadeInDown">
            <div class="card h-100 shadow-sm">
                <div class="card-body text-center">
                    <i class="bi bi-list-stars feature-icon text-primary"></i>
                    <h2 class="card-title">Тема 2</h2>
                    <p class="card-text">Типи подання знань</p>
                    <button type="button" class="btn btn-primary">
                        <i class="bi bi-arrow-return-right"></i> Розпочати
                    </button>
                </div>
            </div>
        </div>

        <div class="col-md-4 animate__animated animate__fadeInRight">
            <div class="card h-100 shadow-sm">
                <div class="card-body text-center">
                    <i class="bi bi-list-stars feature-icon text-primary"></i>
                    <h2 class="card-title">Тема 3</h2>
                    <p class="card-text">Коефіцієнт впевненості</p>
                    <button type="button" class="btn btn-primary">
                        <i class="bi bi-arrow-return-right"></i> Розпочати
                    </button>
                </div>
            </div>
        </div>

        <div class="col-md-4 animate__animated animate__fadeInUp">
            <div class="card h-100 shadow-sm">
                <div class="card-body text-center">
                    <i class="bi bi-award feature-icon text-primary"></i>
                    <h2 class="card-title">Підсумковий тест</h2>
                    <p class="card-text">Збірник питань з усіх тем</p>
                    <button type="button" class="btn btn-primary">
                        <i class="bi bi-arrow-return-right"></i> Розпочати
                    </button>
                </div>
            </div>
        </div>

    </div>
</div>

{% endblock %}
```

Рисунок 4.25 – Реалізація сторінки вибору тестів

і має такий вигляд:

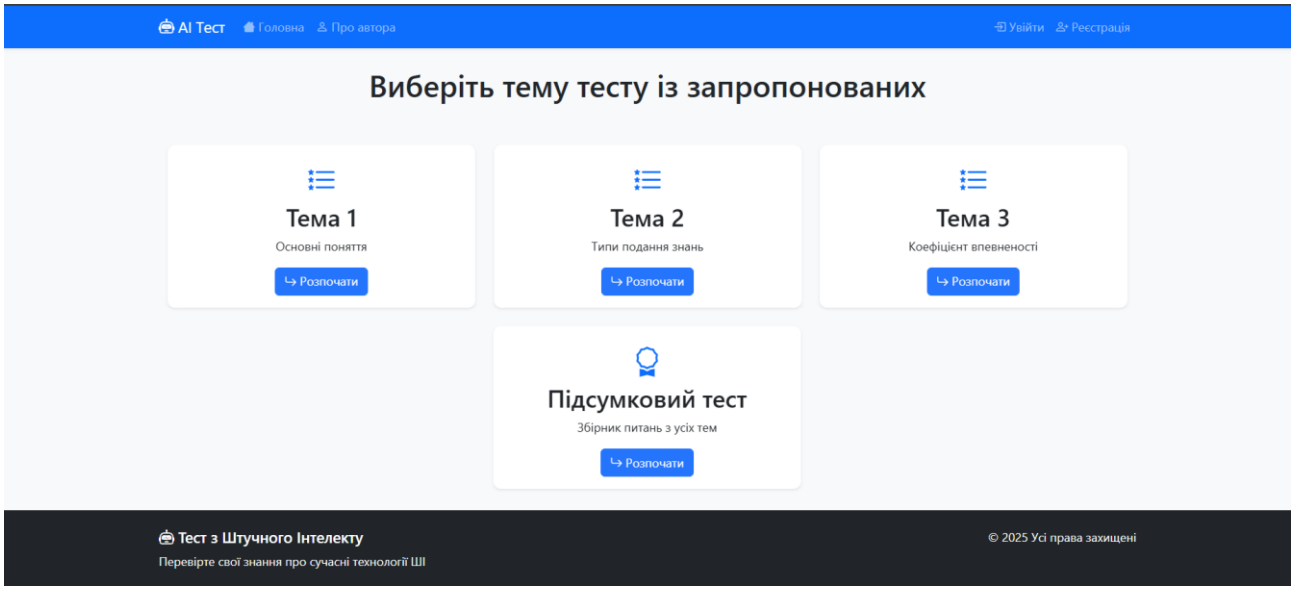


Рисунок 4.26 – Зовнішній вигляд сторінки вибору тестів

JavaScript:

На цьому етапі розберемо реалізацію анімацій, які використовуються на сторінці тестів. Розглянемо наступний JavaScript код:

```
document.addEventListener('DOMContentLoaded', function() {
    const animateOnScroll = function() {
        const elements = document.querySelectorAll('.question-card, .card, .alert');

        elements.forEach(element => {
            const elementPosition = element.getBoundingClientRect().top;
            const screenPosition = window.innerHeight / 1.3;

            if (elementPosition < screenPosition) {
                element.classList.add('animate__fadeInUp');
            }
        });
    };

    window.addEventListener('scroll', animateOnScroll);
    animateOnScroll();

    const questionCards = document.querySelectorAll('.question-card');
    questionCards.forEach(card => {
        const questionText = card.querySelector('.card-title').textContent;

        const hintButton = document.createElement('button');
        hintButton.className = 'btn btn-sm btn-outline-info hint-button';
        hintButton.innerHTML = '<i class="bi bi-lightbulb"></i> Підказка';

        const cardBody = card.querySelector('.card-body');
        cardBody.appendChild(hintButton);

        hintButton.addEventListener('click', function() {
            alert('Підказка: ' + questionText);
        });
    });

    const testForm = document.querySelector('form[action="/test"]');
    if (testForm) {
        testForm.addEventListener('submit', function(e) {
            const unanswered = document.querySelectorAll('input[type="radio"]:checked').length < 15;
            if (unanswered) {
                if (!confirm('Ви відповіли не на всі питання. Продовжити?')) {
                    e.preventDefault();
                }
            }
        });
    }
});
```

Рисунок 4.27 – JavaScript код веб-додатку

Розберемо по порядку, що тут відбувається та як воно працює:

1. Для початку ми чекаємо, поки HTML-документ буде повністю завантажений, перш ніж виконувати JS-код. Це важливо, щоб скрипт не працював із ще неіснуючими елементами. Така поведінка досягається за допомогою вбудованого методу **addEventListener(),** який, сходячи з назви, додає «прослуховувач» подій.
2. Далі всередині лямбда-функції реалізовуємо анімацію при прокрутці. Функція **animateOnScroll()** візьме усі елементи, які відповідають до контейнерів з питаннями до тесту, перевірить, чи елемент вже з’явився в

полі зору користувача та присвоїть елементу клас **animate__fadeInUp** (цей клас знаходиться у файлі **Animate.css,** який ми імпортували)

3. Наступним кроком буде реалізація появлення підказки до кожного питання. Створюємо кнопку підказки зі значком лампочки (він також заходиться в імпортованому фреймворку Bootstrap), додаємо кнопку до класу **card-body,** та виводимо **alert** (функція, яка відповідає за появлення повідомлення), де буде і знаходитись сама підказка.
4. Далі підраховуємо, скільки варіантів відповідей було вибрано. Якщо менше 15 (бо самих питань 15), то відбувається вивід повідомлення, що користувач відповів не на всі питання

Такий JavaScript значно покращує зручність і взаємодію користувача з сайтом, доповнюючи функціональність, яку дає HTML, **Flask** і Bootstrap.

Після всіх цих етапів розглянемо як виглядає зовнішній вигляд веб-додатку:

4.2 Інструкція користувача

1. Розглянемо головну сторінку сайту, де користувач має змогу дізнатися більше про сайт та перейти до таких сторінок, як: реєстрація, авторизація та «Про автора».

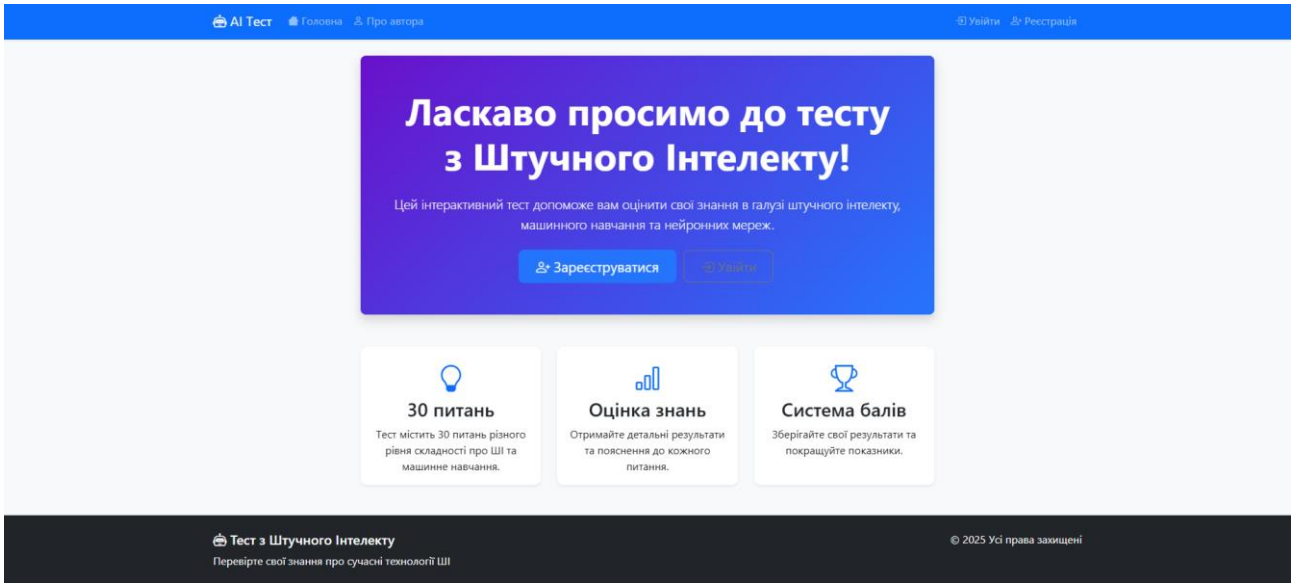


Рисунок 4.28 – Головна сторінка

2. На наступному рисунку зображена сторінка реєстрації нового користувача, де він вводить свої дані (Логін, ім’я та прізвище, номер групи та пароль). Також у користувача є змога перейти на сторінку авторизації, якщо він вже має аккаунт та помилково перейшов до сторінки реєстрації. Після цього етапу користувача перенаправить до сторінки профілю.

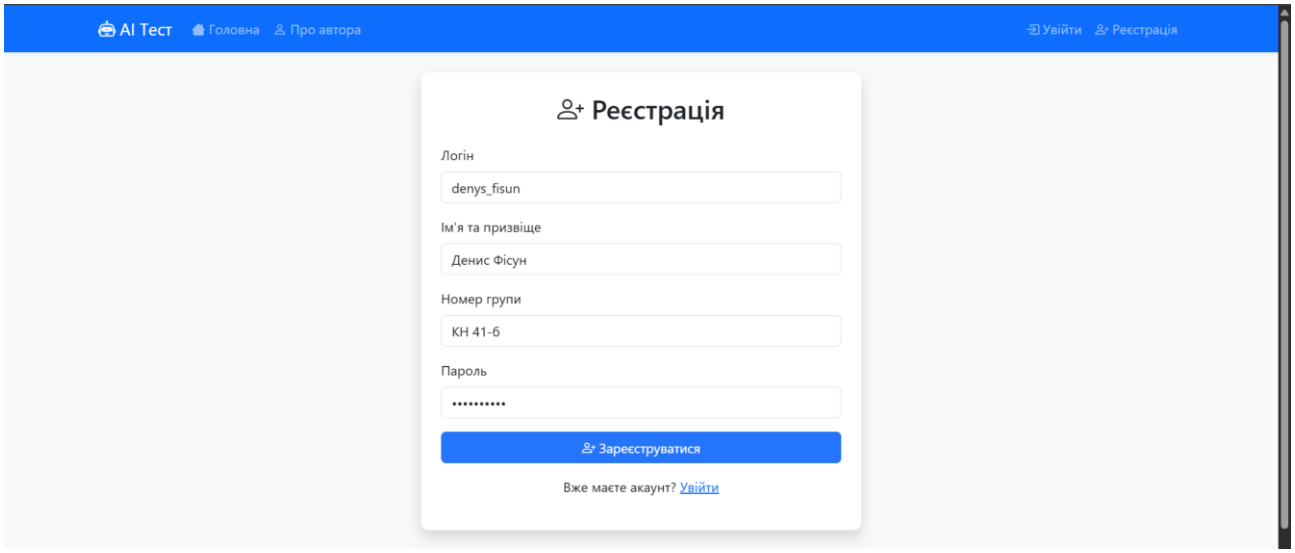


Рисунок 4.29 – Сторінка реєстрації

3. Далі зображена сторінка авторизації, де за допомогою даних, введених при реєстрації, користувач може зайти до свого облікового запису. Так само, як і на попередній сторінці, користувач має змогу перейти до сторінки реєстрації у випадку помилкового натискання кнопки авторизації.

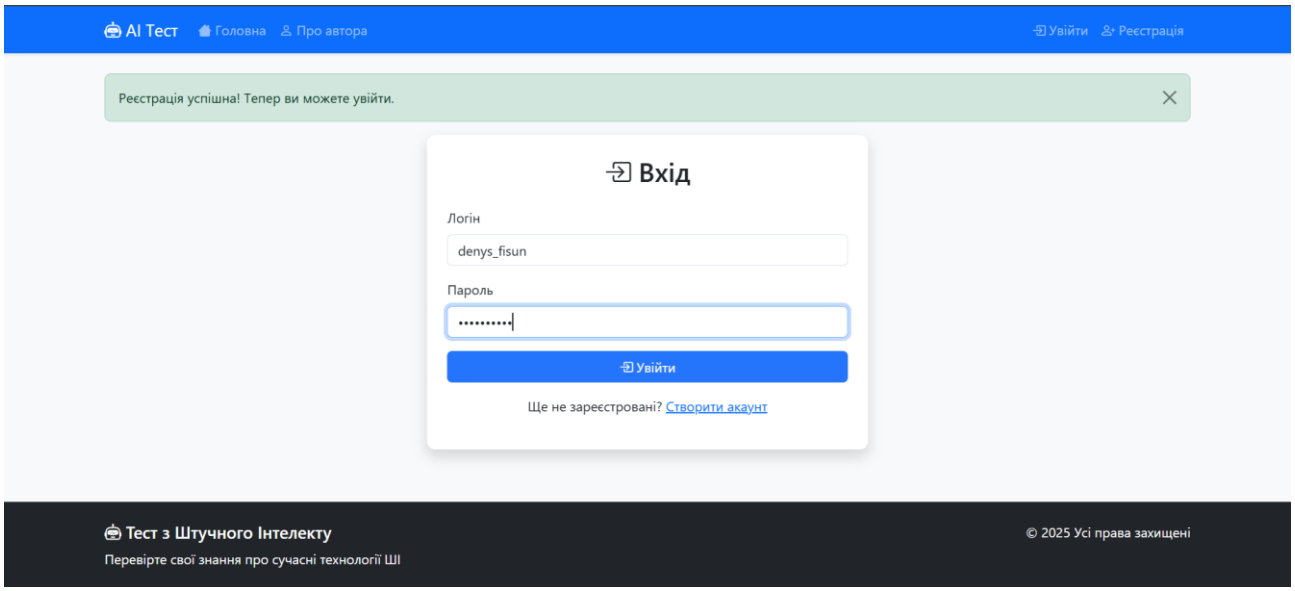


Рисунок 4.30 – Сторінка авторизації

4. Наступною сторінкою є особистий кабінет, де користувач має змогу: змінити дані облікового запису, подивитись результати своїх спроб пройти тест (якщо спроби були) та пройти сам тест.

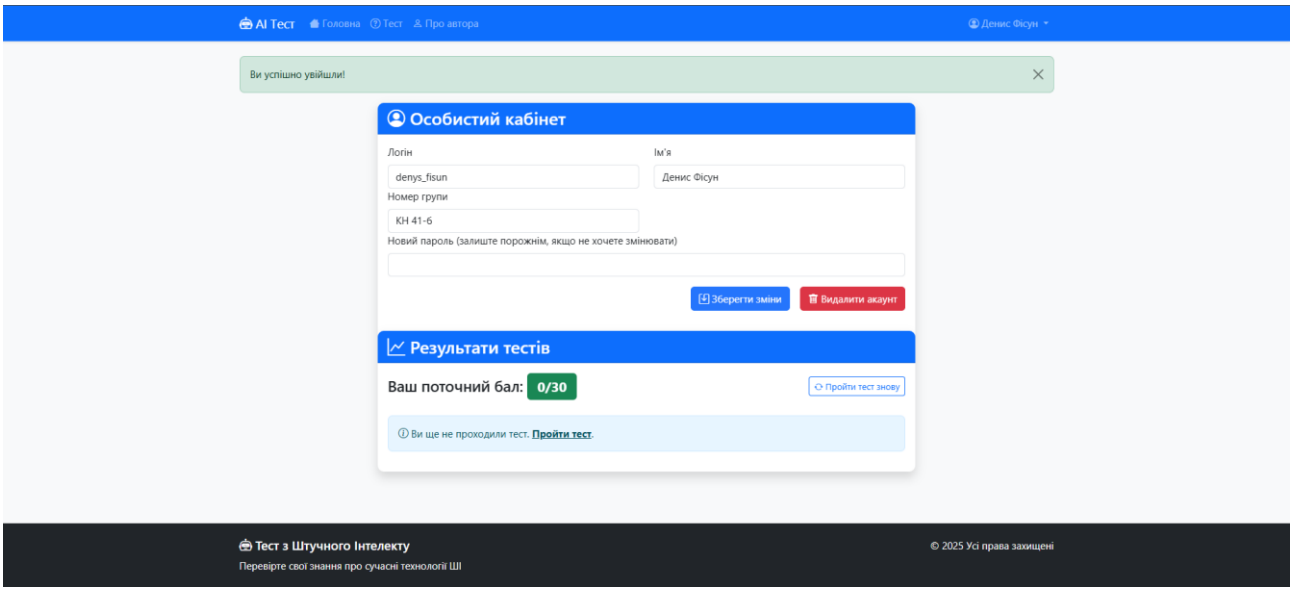


Рисунок 4.31 – Сторінка профілю

5. На наступному рисунку зображена сторінка тестування, де користувачу дається 30 питань з теми «Штучний інтелект». Користувач мусить відповісти на усі питання, щоб побачити результат усього тесту. Кожен раз при спробі пройти тест, питання тесту та відповіді до нього будуть розставлені у випадковому порядку для запобігання запам’ятовування правильних відповідей.

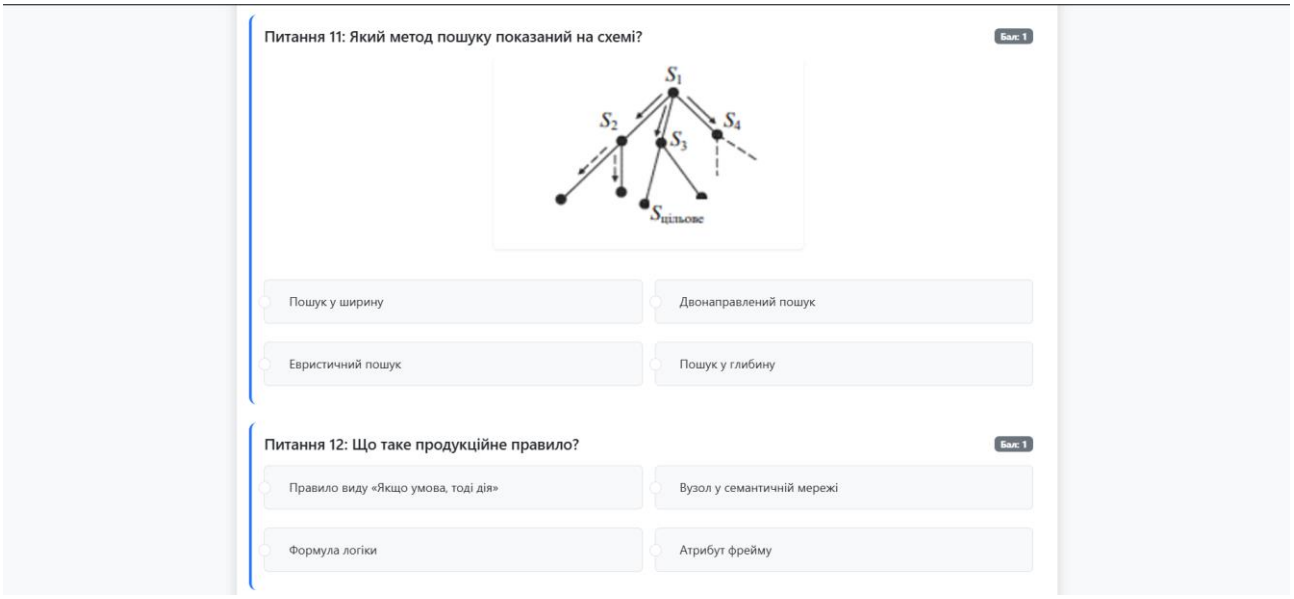


Рисунок 4.32 – Сторінка тестування

6. Наступним чином буде виглядати сторінка результатів тесту, які склав користувач. Сайт оцінить спробу користувача, дасть рекомендацію для

покращення результату та пояснить усі питання та відповіді, на які він відповів неправильно, що допоможе користувачу краще підготуватися та пройти тест ліпше у наступний раз.

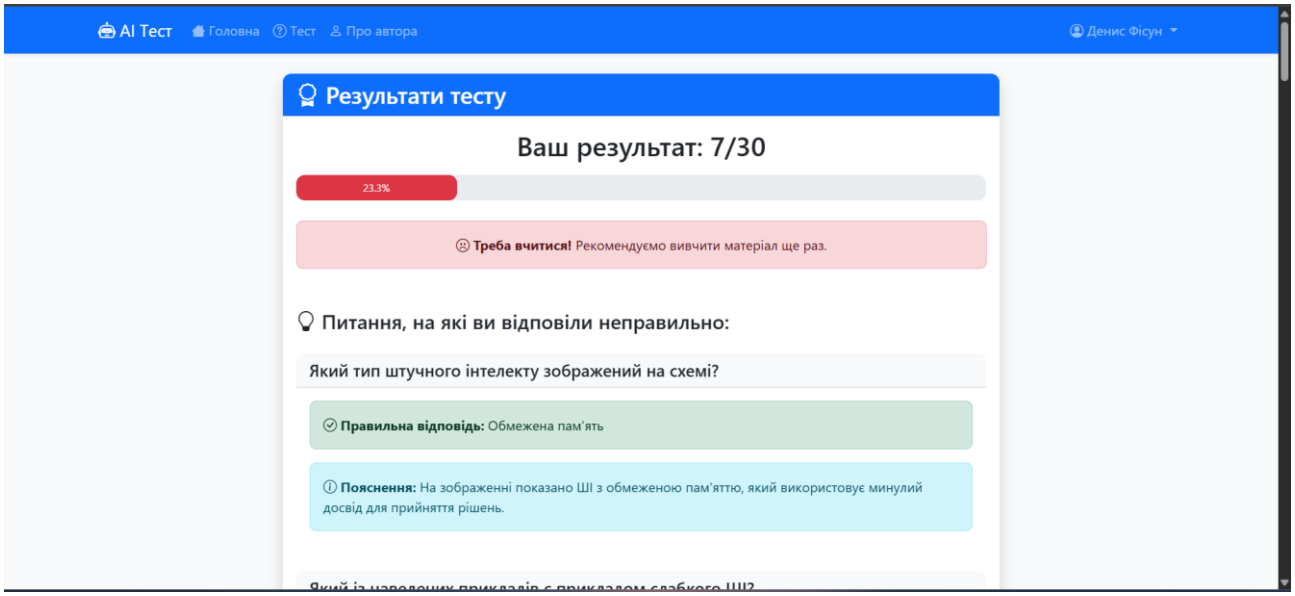


Рисунок 4.33 – Сторінка результату тесту

7. Кожна спроба також буде відображена на сторінці профілю.

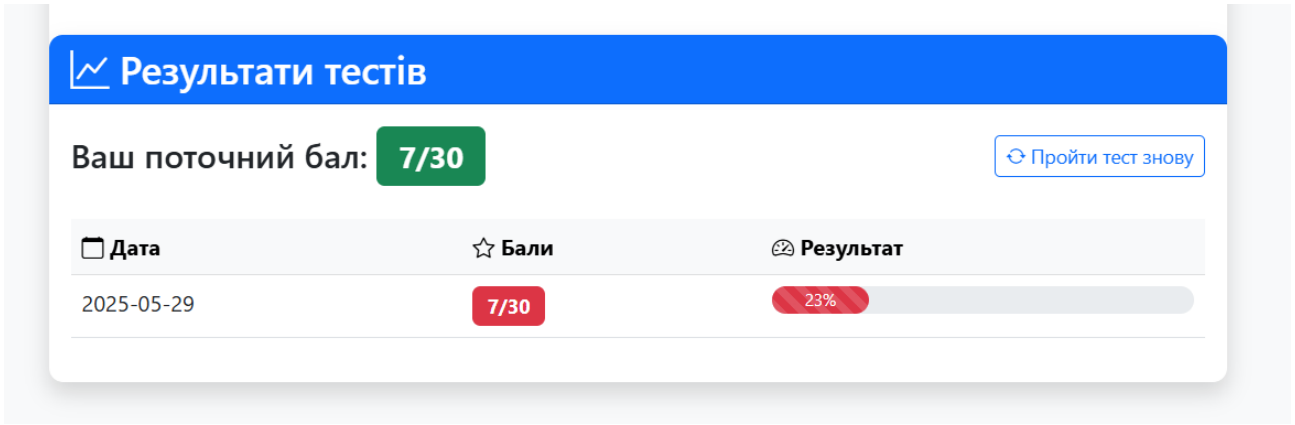


Рисунок 4.34 – Вигляд результатів тестування на сторінці профілю

ВИСНОВКИ

У цьому проекті було реалізовано повноцінну освітню систему, яка не тільки дозволяє перевіряти та контролювати знання зі штучного інтелекту, а його розробка демонструє повний цикл створення веб-додатку – від серверної частини до візуального інтерфейсу.

Було використано **Python** як основну мову програмування і **Flask** як серверний фреймворк. Це дозволило:

- реалізувати маршрутизацію сторінок (@app.route)
- обробити автентифікацію користувачів (реєстрація, вхід, вихід)
- створити тестову систему з оцінюванням
- працювати з сесіями та даними користувача
- забезпечити взаємодію з базою даних через SQLite3

Таким чином, було створено повноцінну логіку бекенду, де кожна дія користувача має результат, записується в БД, та оновлює інтерфейс.

Для створення бази даних, яка містить результати контролю та самі тести, було використано мова структурованих запитів SQL, а саме:

- створено таблиці users та test_results
- збережено логіни, імена, паролі, номери груп, результати тестів
- виконано SELECT, INSERT, UPDATE, DELETE запити
- використано хешування паролів для безпеки

Важливо, що SQLite була обрана через простоту, вбудованість у **Python**та відсутність потреби в окремому сервері – ідеально для навчального проекту.

Було створено веб-сторінки на базі HTML (структура) і CSS (оформлення),підключивши Bootstrap – популярний CSS-фреймворк, що дозволив:

- робити сайт адаптивним (працює на мобільних і ПК)
- створювати кнопки, форми, навігацію, картки

- швидко оформлювати інтерфейс без написання вручну великої кількості стилів

Також було додано Animate.css для анімацій і були створені візуально привабливі сторінки.

Було використано Jinja2 – мову шаблонів, вбудовану у **Flask**, для:

- виведення динамічних даних у HTML ({{ username }}, {{ score }})
- побудови умовних блоків ({% if %}, {% for %})
- повторення елементів списків (наприклад, питань у тесті)
- розділення логіки **Python** і HTML, що забезпечує зручність і безпеку

За допомогою JavaScript було:

- реалізовано анімацію при прокрутці
- додано підказки до кожного питання
- перевірено, чи користувач відповів на всі питання перед надсиланням форми

Це значно покращило користувацький досвід (UX). У проекті враховані базові принципи безпеки:

- хешування паролів через generate_password_hash
- використання сесій замість передачі паролів
- перевірка унікальності логіна перед оновленням профілю
- валідація введення користувача через регулярні вирази

Завдяки цьому проекту було досягнено:

- планування структуру веб-застосунку
- працювання з базами даних та запитами SQL
- використання **Flask** для маршрутизації та логіки
- створено красивий фронтенд за допомогою HTML, CSS, Bootstrap

- підключання JavaScript для динаміки та перевірки даних
- створювання зручний інтерфейс користувача
- розуміння як працює аутентифікація, профілі користувачів, збереження результатів

У процесі створення проекту було ознайомлення з широким спектром інструментів, які дозволяють ефективно та швидко розробляти повноцінні вебдодатки:

- **Python** – як мова з простим синтаксисом та потужними бібліотеками.
- **Flask** – як легкий і зрозумілий вебфреймворк, ідеальний для невеликих проєктів.
- **SQLite** – вбудована, зручна та легка база даних для локальних застосунків.
- **HTML + CSS** – основа структури та стилів вебсторінок.
- **Bootstrap** – фреймворк для створення адаптивного та привабливого дизайну.
- **JavaScript** – мова для інтерактивності та реакції на дії користувача.
- **Jinja2** – шаблонізатор для динамічного генерування HTML з Python.

СПИСОК ЛІТЕРАТУРИ ТА ДЖЕРЕЛ

1. Ольховська О.: Методичні рекомендації до виконання кваліфікаційної роботи. Полтава: РВВ ПУЕТ 2025. 58 с.
2. Федорчук Є. Н. Програмування систем штучного інтелекту. Експертні системи : навч. посіб. / Є. Н. Федорчук ; Нац. ун-т «Львівська політехніка». — Львів : Вид-во Львівської політехніки, 2012. — 168 с. — ISBN 978-617-607-257-7. - https://catalog.lounb.org.ua/bib/393755
3. Федорчук Є. Н. Експертні системи : навч. посіб. / Є. Н. Федорчук ; Нац. ун-т «Львівська політехніка». — Львів : Вид-во Львівської політехніки, 2012. — 168 с.
4. Доусон М. Програмуємо на Python / Майкл Доусон ; пер. з англ. — Санкт-Петербург : Пітер, 2016. — 416 с.
5. Меттіз Е. Python. Курс молодого бійця / Ерік Меттіз ; пер. з англ. — Київ : Видавництво «Діалектика», 2015. — 560 с.
6. «Bootstrap для початківців» – самовчитель в онлайн-ресурсах, окремі pdf
7. Офіційна документація Python 3 [Електронний ресурс]. – Режим доступу: https://docs.python.org/uk/3/ – Назва з екрана.
8. PythonWorld – Курс Python3 для початківців [Електронний ресурс]. – Режим доступу: https://pythonworld.ru/ – Назва з екрана.
9. Python Course – Курси Python 3 [Електронний ресурс]. – Режим доступу: https://python-course.eu/ – Назва з екрана.
10. Real Python – Туторіали з Python [Електронний ресурс]. – Режим доступу: https://realpython.com/ – Назва з екрана.
11. Офіційна документація Flask 2.3 [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/2.3.x/ – Назва з екрана.
12. Stack Overflow – Форум програмістів з теми Flask [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/tagged/flask – Назва з екрана.

13. Habr – Публікації з теми Flask [Електронний ресурс]. – Режим доступу: https://habr.com/uk/hub/flask/ – Назва з екрана.
14. Офіційна документація SQLite3 [Електронний ресурс]. – Режим доступу: https://sqlite.org/docs.html – Назва з екрана.
15. SQLite Language Manual – Методичка SQLite3 [Електронний ресурс]. – Режим доступу: https://sqlite.org/lang.html – Назва з екрана.
16. SQLite Tutorial – Онлайн-посібник з SQLite3 [Електронний ресурс]. – Режим доступу: https://sqlitetutorial.net/ – Назва з екрана.
17. W3Schools – HTML & CSS Tutorials [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/ – Назва з екрана.
18. Mozilla Developer Network – Документація з HTML, CSS, JavaScript [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/uk/docs/web – Назва з екрана.
19. HTML Living Standard – WHATWG [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org/ – Назва з екрана.
20. HTMLBook – Підручник HTML та CSS [Електронний ресурс]. – Режим доступу: https://htmlbook.ru/ – Назва з екрана.
21. CSS-Tricks – Поради з CSS [Електронний ресурс]. – Режим доступу: https://css-tricks.com/ – Назва з екрана.
22. Learn JavaScript – Курси з JavaScript [Електронний ресурс]. – Режим доступу: https://learn.javascript.ru/ – Назва з екрана.
23. Офіційна документація Jinja 3.1 [Електронний ресурс]. – Режим доступу: https://jinja.palletsprojects.com/en/3.1.x – Назва з екрана.
24. Bootstrap – Офіційна документація [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/ – Назва з екрана.
25. W3Schools Bootstrap 5 – Курси Bootstrap [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/bootstrap5/ – Назва з екрана.
26. Google Fonts – Каталог шрифтів [Електронний ресурс]. – Режим доступу: https://fonts.google.com/ – Назва з екрана.

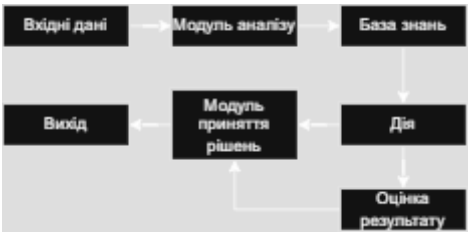
27. Animate.css – Анімаційні стилі [Електронний ресурс]. – Режим доступу: https://animate.style/ – Назва з екрана.
28. Bootstrap Icons – Збірка іконок [Електронний ресурс]. – Режим доступу: https://icons.getbootstrap.com/ – Назва з екрана.
29. CDNJS – Бібліотеки CSS і JavaScript [Електронний ресурс]. – Режим доступу: https://cdnjs.com/ – Назва з екрана.
30. Font Awesome – Каталог іконок та шрифтів [Електронний ресурс]. – Режим доступу: https://fontawesome.com/ – Назва з екрана.

ДОДАТОК А. Код програми

Код усього веб-додатку можна знайти в репозиторії за посиланням:
https://github.com/DisLamb52/flask-web-site.git

ДОДАТОК Б. Перелік тестових питань

1. Який тип штучного інтелекту зображений на схемі?



- Реактивний ШІ
- Обмежена пам’ять (правильна відповідь)
- Теорія розуму
- Самосвідомий ШІ

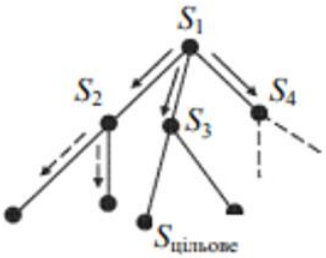
2. Який із наведених прикладів є прикладом слабкого ШІ?

- Робот, який може самостійно навчатися і адаптуватися до нових ситуацій
- Голосовий помічник на кшталт Siri або Google Assistant (правильна відповідь)
- ШІ, що володіє свідомістю
- Робот з людськими емоціями

3. Що таке машинне навчання?

- Процес програмування комп’ютера конкретними інструкціями
- Підмножина ШІ, що дозволяє системам автоматично вчитися на основі даних (правильна відповідь)
- Метод зберігання великих обсягів інформації
- Технологія створення віртуальної реальності

4. Який метод пошуку зображений на схемі?



- Пошук у глибину
- Пошук у ширину (правильна відповідь)
- Двонаправлений пошук
- Евристичний пошук

5. Що таке глибинне навчання (deep learning)?

- Метод навчання, що використовує поверхневі нейроні мережі
- Підхід до машинного навчання, що імітує роботу людського мозку з багатьма шарами нейронів (правильна відповідь)
- Спосіб зберігання даних у хмарних сервісах
- Технологія для створення 3D-графіки

6. Що таке нейрона мережа?

- Фізична мережа комп'ютерів
- Система, що імітує роботу нейронів мозку для обробки інформації (правильна відповідь)
- Соціальна мережа для вчених
- Тип бази даних

7. Який алгоритм часто використовується для класифікації в машинному навчанні?

- Лінійна регресія
- К-найближчих сусідів (KNN) (правильна відповідь)
- Метод бульбашкового сортування
- Алгоритм Дейкстри

8. Що таке надлишкове навчання (overfitting)?

- Коли модель занадто добре працює на тренувальних даних, але погано на нових (правильна відповідь)
- Коли модель навчається занадто повільно
- Коли модель ігнорує важливі ознаки
- Коли модель використовує занадто мало даних

9. Що таке TensorFlow?

- Мова програмування для ШІ
- Бібліотека для машинного навчання від Google (правильна відповідь)
- Тип нейронної мережі
- Апаратне забезпечення для ШІ

10. Що таке комп'ютерний зір?

- Технологія для створення віртуальної реальності
- Галузь ШІ, що допомагає комп'ютерам інтерпретувати зображення (правильна відповідь)
- Метод візуалізації даних
- Тип дисплея з високою роздільною здатністю

11. Що таке обробка природної мови (NLP)?

- Програмування на природній мові

- Галузь ШІ, що займається взаємодією комп’ютерів і людської мови (правильна відповідь)
- Метод стиснення текстових даних
- Технологія перетворення тексту в код

12. Що таке reinforcement learning?

- Навчання через підкріплення, де система вчиться на основі винагороди (правильна відповідь)
- Метод запам'ятовування даних
- Тип нейронної мережі
- Спосіб стиснення даних

13. Що таке GPT у назві ChatGPT?

- General Purpose Technology
- Generative Pre-trained Transformer (правильна відповідь)
- Global Processing Tool
- Graphical Programming Technique

14. Що таке етика ШІ?

- Вивчення того, як програмуються ШІ-системи
- Галузь, що вивчає моральні аспекти розробки та використання ШІ (правильна відповідь)
- Метод покращення продуктивності ШІ
- Тип алгоритмів для ШІ

15. Яка технологія дозволяє ШІ генерувати реалістичні зображення?

- Generative Adversarial Networks (GANs) (правильна відповідь)
- Support Vector Machines (SVM)
- Random Forest
- Linear Regression

16. Що таке експертна система?

- Система, що імітує процес прийняття рішень експертом у певній галузі (правильна відповідь)
- Система для зберігання даних
- Операційна система
- Програма для обробки зображень

17. Що має на увазі "семантична сегментація" в задачах комп'ютерного зору?

- Призначення кожному пікселю зображення певного класу (наприклад, "дорога", "пішохід", "автомобіль", "трава") (правильна відповідь)

- Визначення та локалізація окремих об'єктів на зображенні за допомогою обмежувальних рамок
- Класифікація всього зображення в цілому до однієї категорії (наприклад, "пейзаж", "портрет")
- Розпізнавання тексту, що знаходиться всередині зображення

18. Що таке навчання з підкріпленням?

- Навчання на основі прикладів з правильними відповідями
- Навчання, де агент отримує винагороду або покарання за дії (правильна відповідь)
- Навчання без використання даних
- Навчання за допомогою правил

19. Що таке семантична мережа?

- Графічне представлення математичних функцій
- Мережа, що описує зв’язки між поняттями через вузли та ребра (правильна відповідь)
- Архітектура нейронної мережі для обробки зображень
- Алгоритм стиснення даних

20. Що таке фрейм у контексті подання знань?

- Правило, що описує логічний висновок
- Структура даних, що описує об’єкт через набір атрибутів і значень (правильна відповідь)
- Граф, що з’єднує поняття
- Формула логіки першого порядку

21. Що таке алгоритм?

- Послідовність кроків для розв’язання задачі (правильна відповідь)
- Мова програмування
- Тип даних
- Операційна система

22. Що таке інтервал довіри?

- Діапазон значень, у якому з певною впевненістю знаходиться параметр (правильна відповідь)
- Середнє значення вибірки
- Максимальне значення вибірки
- Мінімальне значення вибірки

23. Що таке продукційне правило?

- Формула логіки

- Правило виду «Якщо умова, тоді дія» (правильна відповідь)
- Вузол у семантичній мережі
- Атрибут фрейму

24. Який тип подання знань використовує предикати і квантори?

- Логічне подання (правильна відповідь)
- Семантична мережа
- Фреймове подання
- Продукційна система

25. Що таке RDF (Resource Description Framework)?

- Мова розмітки для семантичних мереж у Вебі (правильна відповідь)
- Алгоритм сортування даних
- Мова програмування для нейромереж
- Протокол передачі файлів

26. Який інструмент візуалізує семантичні мережі?

- TensorFlow
- Gephi (правильна відповідь)
- Photoshop
- MySQL Workbench

27. Що таке наслідування у фреймовому поданні?

- Процес копіювання правил
- Механізм успадкування атрибутів від батьківського фрейму (правильна відповідь)
- Процес логічного виводу
- Процес створення нових правил

28. Що таке агент у ШІ?

- Програма, що виконує певні дії в середовищі для досягнення мети (правильна відповідь)
- Користувач комп’ютера
- Операційна система
- Тип даних

29. Що таке класифікація в машинному навчанні?

- Розподіл об’єктів на групи за певними ознаками (правильна відповідь)
- Пошук найкоротшого шляху
- Оптимізація функції
- Зберігання даних

30. Що таке "контрприклад" (adversarial example) у контексті машинного навчання?

- Це спеціально підібраний вхідний приклад, який змушує модель дати помилковий результат (правильна відповідь)
- Це приклад, який використовується для навчання моделі правильно класифікувати складні випадки
- Це помилковий результат, отриманий моделлю через недостатнє навчання
- Це алгоритм, який шукає вади в інших моделях машинного навчання.