**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ**

**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ЕНЕРГЕТИЦІ**

КУРСОВА РОБОТА

**з дисципліни:** «Основи Веб-програмування»

**на тему:** «Додаток для доставки їжі»

**Керівник**: **Виконав**:

Гагарін. О.О. Козаченко Олександр Вікторович студент 2 курсу

«Допущено до захисту» групи ТВ-32

(особистий підпис керівника)

« » 20 р.

Захищено з оцінкою (особистий підпис виконавця)

« » 20 р.

(оцінка)

Посилання на **GitHub** репозиторій: <https://github.com/KURYO3/WebCursova>

**Київ – 2025**

**РОЗІДЛ 5. РОЗРОБКА МЕХАНІЗМІВ ОТРИМАННЯ ТА ОНОВЛЕННЯ ДАНИХ**

Цей розділ присвячений детальному розгляду реалізації клієнтської частини веб-додатку, яка відповідає за динамічну взаємодію з даними. Фокус зроблено на використанні JavaScript для ініціювання запитів, обробки відповідей та оновлення користувацького інтерфейсу, забезпечуючи безперервний та чуйний досвід. Серверна логіка цих операцій буде детально описана в Розділі 9.

**5.1. Основи клієнтської взаємодії з даними**

Клієнтська сторона веб-додатку, що функціонує у браузері користувача, є точкою взаємодії з даними. JavaScript виступає як "мозок" фронтенду, який не лише відображає статичний контент, але й активно керує життєвим циклом даних: від запиту до сервера до їх відображення та подальшого оновлення. Ключовими елементами цієї взаємодії є:

* **Асинхронність:** Операції з даними виконуються у фоновому режимі, не блокуючи інтерфейс.
* **Fetch API:** Сучасний інтерфейс для виконання мережевих запитів, що повертає проміси.
* **JSON:** Стандартний формат для обміну даними, який легко парситься JavaScript.
* **DOM (Document Object Model):** Програмний інтерфейс для HTML-документів, що дозволяє JavaScript динамічно змінювати структуру, стиль та вміст сторінки.

**5.2. Асинхронний потік даних та відображення на клієнтській стороні**

Основою динамічного веб-додатку є асинхронна взаємодія з даними. JavaScript, використовуючи механізм промісів (Promises) та Fetch API, дозволяє виконувати мережеві запити у фоновому режимі, не блокуючи основний потік виконання та не заморожуючи інтерфейс користувача. Після успішного отримання даних, JavaScript трансформує їх з формату JSON у зручні для маніпуляцій об'єкти та оперативно оновлює відповідні частини DOM, забезпечуючи миттєве відображення актуальної інформації.

Прикладом реалізації є динамічне завантаження та рендеринг ресторанів (файл restaurants.html). Функція loadRestaurants() забезпечує динамічне завантаження та відображення даних. Цей приклад підкреслює асинхронний характер Fetch API та використання промісів для послідовної обробки відповіді: спочатку перевірка статусу, потім парсинг JSON, і лише потім маніпуляції з DOM. Це забезпечує, що інтерфейс залишається чуйним під час очікування даних, а оновлення відбувається лише після їх успішного отримання та обробки.

**5.3. Валідація та зворотний зв'язок на клієнтській стороні**

Для покращення користувацького досвіду та зменшення навантаження на сервер, значна частина валідації введених даних може бути виконана безпосередньо на клієнтській стороні за допомогою JavaScript. Це дозволяє миттєво повідомляти користувача про некоректні дані (наприклад, невідповідність паролів, недостатня довжина) ще до відправки запиту на сервер. Крім того, JavaScript відповідає за відображення повідомлень про успіх операції або деталізованих повідомлень про помилки, отриманих від сервера, забезпечуючи прозорий зворотний зв'язок. У проєкті для цього використовуються спеціальні div елементи з динамічно доданими класами (success, error) для візуального відображення статусу.

**5.4. Висновок**

Отже, JavaScript є невід'ємною складовою архітектури клієнтської частини, що дозволяє реалізувати динамічні механізми отримання та оновлення даних. Його можливості з асинхронної взаємодії, маніпуляції DOM та обробки подій забезпечують високу чуйність інтерфейсу та ефективну комунікацію з серверною частиною, що є критично важливим для сучасних веб-додатків, орієнтованих на користувача.

**РОЗДІЛ 6. РОЗРОБКА БАЗИ ДАНИХ ДЛЯ ЗБЕРЕЖЕННЯ ТА ОТРИМАННЯ ІНФОРМАЦІЇ**

У цьому розділі розповідається про розробку таблиць за які відповідає Козаченко Олександр.

**1. roles**

Ця таблиця містить типи користувачів системи (наприклад, клієнт, власник ресторану, адміністратор).

* id – первинний ключ (PK), автоінкремент.
* name – унікальна назва ролі (унікальність забезпечує логіку авторизації в системі).

**2. users**

Визначає зареєстрованих користувачів із прив’язкою до ролі.

* id – первинний ключ.
* role\_id – зовнішній ключ (FK) на roles(id), що забезпечує рольову ієрархію.
* email – унікальне поле, яке гарантує, що одна адреса електронної пошти не буде використана кілька разів.
* password\_hash – збережений хеш пароля.
* is\_active, created\_at, updated\_at – технічні поля для контролю активності та часу змін.

**3. restaurants**

Містить інформацію про ресторани, які належать користувачам з відповідною роллю.

* id – первинний ключ.
* user\_id – зовнішній ключ на users(id), що дозволяє відстежити власника ресторану.
* name – назва ресторану.
* description, phone, cuisine\_type – текстові поля для опису, контактної інформації, типу кухні.
* created\_at, updated\_at – контроль часу створення та оновлення.
* Зв'язки: при видаленні користувача, ресторан видаляється також (ON DELETE CASCADE).

**4. categories**

Зберігає категорії страв, які доступні в системі (наприклад, піца, салати, десерти).

* id – первинний ключ.
* name – унікальна назва категорії, що запобігає дублюванню.

**5. dishes**

Представляє меню кожного ресторану з деталями про страви.

* id – первинний ключ.
* restaurant\_id – зовнішній ключ на restaurants(id).
* category\_id – зовнішній ключ на categories(id).
* name, description, price, image\_url – описові атрибути страви.
* is\_available – логічне поле, що показує, чи страва доступна для замовлення.
* Зв'язки: видалення ресторану автоматично видаляє страви з меню; категорії захищені від видалення, якщо до них прив’язані страви.

**РОЗДІЛ 7. ІНТЕГРАЦІЯ СТОРОННІХ СЕРВІСІВ В СИСТЕМУ**

У розробці складних програмних систем часто виникає необхідність використання зовнішніх компонентів або сервісів, які не є частиною основного коду програми, але забезпечують її критично важливий функціонал. У даному проєкті яскравим прикладом такої інтеграції є використання бази даних MySQL, яка функціонує як незалежний сторонній сервіс. Це рішення дозволяє оптимізувати розподіл ресурсів та підвищити надійність системи в цілому.

**7.1. Архітектурне значення зовнішньої бази даних MySQL**

База даних MySQL у цьому проєкті розглядається як окремий, життєво важливий сторонній сервіс. Вона розміщена на віддаленому віртуальному сервері, який знаходиться у датацентрі в Роттердамі, Нідерланди. Технічні характеристики цього сервера включають операційну систему AlmaLinux (64 bit) з панеллю керування Webuzo, 50 GB дискового простору, 4 GB оперативної пам'яті та 2 GB swap-пам'яті.

Таке архітектурне рішення, де база даних відокремлена від основного Flask-додатку, має кілька ключових переваг:

* **Масштабованість:** Дозволяє незалежно масштабувати веб-сервер та сервер бази даних, оптимізуючи використання ресурсів.
* **Надійність:** Вихід з ладу одного компонента (наприклад, веб-сервера) не впливає на доступність даних у базі.
* **Безпека:** Розділення компонентів зменшує поверхню атаки та дозволяє застосовувати специфічні заходи безпеки для кожного сервісу.
* **Оптимізація ресурсів:** Виділені ресурси віртуального сервера повністю використовуються для операцій з базою даних, забезпечуючи високу продуктивність.

З'єднання з цією віддаленою базою даних встановлюється за допомогою спеціалізованого драйвера mysql.connector у Python, який виступає мостом між Flask-додатком та сервером MySQL.

**Приклад реалізації: вхід користувача та отримання даних з віддаленої бази (файл app.py)**

Розглянемо, як відбувається взаємодія з віддаленою базою даних під час процесу входу користувача. Цей процес демонструє отримання даних з MySQL для автентифікації.

|  |
| --- |
| @app.route("/api/login", methods=["POST"])  def login():  email = request.form.get("email")  password = request.form.get("password")  if not email or not password:  return jsonify({"error": "Введіть email і пароль"}), 400  conn = None  cursor = None  try:  conn = get\_db\_connection()  cursor = conn.cursor(dictionary=True)  cursor.execute("SELECT \* FROM users WHERE email = %s", (email,))  user = cursor.fetchone()  if not user:  return jsonify({"error": "Користувача не знайдено"}), 401  if not bcrypt.checkpw(password.encode('utf-8'), user["password\_hash"].encode('utf-8')):  return jsonify({"error": "Невірний пароль"}), 401  session['user\_id'] = user['id']  session['user\_name'] = user['name']  session['user\_role\_id'] = user['role\_id']  except mysql.connector.Error as err:  print(f"Помилка бази даних під час входу: {err}")  return jsonify({"error": "Помилка сервера під час входу"}), 500  finally:  if cursor: cursor.close()  if conn: conn.close()  return jsonify({  "success": True,  "message": "Успішний вхід",  "user": {  "id": user["id"],  "name": user["name"],  "role\_id": user["role\_id"]  }  }) |

**Пояснення:**

* **get\_db\_connection():** Ця функція, як вже згадувалося, є шлюзом до віддаленої бази даних. Вона використовує заздалегідь налаштовані параметри (хост, ім'я користувача, пароль, назва бази даних) для встановлення безпечного з'єднання.
* **cursor.execute("SELECT \* FROM users WHERE email = %s", (email,)):** Цей рядок демонструє, як Flask-додаток відправляє запит на отримання даних до віддаленої бази. Важливо, що замість прямого вбудовування змінних у SQL-рядок, використовується параметризований запит (%s), що є стандартом безпеки для запобігання SQL-ін'єкціям.
* **user = cursor.fetchone():** Отримані дані з віддаленої бази даних потім обробляються у Flask-додатку. У цьому випадку, це запис користувача, який далі використовується для перевірки пароля.
* **Обробка винятків (try...except...finally):** Ця конструкція є критично важливою для забезпечення надійності. Вона дозволяє коректно обробляти потенційні помилки, що можуть виникнути під час взаємодії з віддаленим сервісом (наприклад, проблеми з мережею, недоступність бази даних, помилки автентифікації), забезпечуючи, що додаток не "впаде" і надасть користувачеві відповідне повідомлення. Блок finally гарантує, що ресурси (курсор, з'єднання) завжди будуть звільнені.

**7.2. Висновок**

Інтеграція бази даних MySQL як стороннього сервісу є фундаментальним стовпом архітектури даного веб-додатку. Вона забезпечує надійне та масштабоване сховище для всіх даних, дозволяючи основному додатку зосередитися на бізнес-логіці та інтерфейсі користувача. Завдяки ретельній конфігурації з'єднань та використанню стандартних практик безпеки, ця інтеграція формує міцну та ефективну основу для функціонування всієї системи.

**РОЗДІЛ 8. СТВОРЕННЯ СЕРВЕРНОЇ АРХІТЕКТУРИ ДОДАТКУ**

Цей розділ присвячений формуванню серверної частини веб-додатку, яка є фундаментом для обробки всіх запитів від клієнтської сторони та керування бізнес-логікою. Архітектура сервера побудована на основі мікрофреймворку Flask, що дозволяє гнучко та ефективно реалізувати API для взаємодії з даними. Основна увага приділяється структурі додатку, маршрутизації запитів та механізмам обробки даних на сервері.

**8.1. Вибір та конфігурація Flask як основи серверу**

Для створення серверної архітектури було обрано Python-фреймворк Flask. Його головною перевагою є мінімалістичний підхід ("мікрофреймворк"), що надає розробнику високу гнучкість і контроль над компонентами. Flask ідеально підходить для побудови RESTful API, оскільки він не нав'язує жорстких структур і дозволяє створювати легкі та швидкі сервіси для обробки HTTP-запитів.

Основна конфігурація Flask-додатку включає ініціалізацію об'єкта Flask та встановлення секретного ключа для управління сесіями:

|  |
| --- |
| from flask import Flask, request, jsonify, render\_template, session, redirect, url\_for  import mysql.connector  import bcrypt  import os  from datetime import datetime, timedelta  import secrets  app = Flask(\_\_name\_\_) # Ініціалізація Flask-додатку  app.secret\_key = 'ughsghsighosfhgvoishguoigfo' # Секретний ключ для безпечного управління сесіями |

**8.2. Маршрутизація запитів та обробка HTTP-методів**

Центральним елементом серверної архітектури Flask є система маршрутизації. Вона визначає, як вхідні HTTP-запити (наприклад, GET, POST, PUT, DELETE) зіставляються з конкретними функціями Python, які відповідають за обробку цих запитів. Це реалізується за допомогою декоратора @app.route().

**Приклад реалізації: Маршрут для отримання списку ресторанів (/api/restaurants)**

|  |
| --- |
| @app.route("/api/restaurants", methods=["GET"])  def api\_get\_restaurants():  """  API-маршрут для отримання списку ресторанів.  Підтримує опціональний параметр пошуку.  """  search\_query = request.args.get('search') # Отримання параметра 'search' з URL-запиту  restaurants = get\_restaurants\_from\_db(search\_query) # Виклик функції для отримання даних з БД  return jsonify(restaurants=restaurants) # Повернення даних у форматі JSON |

**Пояснення:**

* **@app.route("/api/restaurants", methods=["GET"])**: Цей декоратор реєструє функцію api\_get\_restaurants як обробник для URL /api/restaurants. Параметр methods=["GET"] вказує, що ця функція буде реагувати лише на HTTP GET-запити.
* **request.args.get('search')**: Об'єкт request надає доступ до даних вхідного HTTP-запиту. request.args є словником, що містить параметри запиту з URL (наприклад, ?search=pizza).
* **get\_restaurants\_from\_db(search\_query)**: Це приклад виклику функції, яка взаємодіє з базою даних для отримання необхідної інформації. (Деталі взаємодії з БД розглядаються в Розділі 9).
* **jsonify(restaurants=restaurants)**: Flask надає зручну функцію jsonify, яка автоматично перетворює словники та списки Python на JSON-відповідь, встановлюючи відповідний HTTP-заголовок Content-Type: application/json.

**8.3. Обробка вхідних даних та управління сесіями**

Серверна архітектура повинна ефективно обробляти різні типи вхідних даних від клієнта (дані форм, JSON-тіла запитів, параметри URL) та керувати станом користувача за допомогою сесій.

**Приклад реалізації: Обробка запиту на скидання пароля (/api/forgot\_password)**

|  |
| --- |
| @app.route("/api/forgot\_password", methods=["POST"])  def api\_forgot\_password():  """  API-маршрут для обробки запиту на скидання пароля.  Генерує токен та симулює надсилання посилання на email.  """  email = request.form.get('email') # Отримання даних з тіла POST-запиту (дані форми)  if not email:  return jsonify({"error": "Будь ласка, введіть ваш email."}), 400  conn = get\_db\_connection()  cursor = conn.cursor(dictionary=True)  cursor.execute("SELECT id FROM users WHERE email = %s", (email,))  user = cursor.fetchone()  cursor.close()  conn.close()  if not user:  # Для безпеки не повідомляємо, чи існує email.  print(f"Запит на скидання пароля для неіснуючого email: {email}")  return jsonify({"success": True, "message": "Якщо ваш email зареєстрований, ви отримаєте посилання для скидання пароля."})  user\_id = user['id']  token = create\_password\_reset\_token(user\_id) # Виклик функції для генерації токена  if token:  reset\_link = url\_for('reset\_password\_page', token=token, \_external=True)  # У реальному застосунку тут буде код для надсилання email  print(f"--- СИМУЛЯЦІЯ НАДСИЛАННЯ EMAIL ---")  print(f"Надіслано на: {email}")  print(f"Посилання для скидання пароля: {reset\_link}")  print(f"---------------------------------")  return jsonify({"success": True, "message": "Якщо ваш email зареєстрований, ви отримаєте посилання для скидання пароля."})  else:  return jsonify({"error": "Не вдалося згенерувати токен для скидання пароля. Спробуйте ще раз."}), 500 |

**Пояснення:**

* **request.form.get('email')**: Для POST-запитів, що надсилають дані форми (як у випадку з FormData на клієнті), request.form є словником, що містить ці дані.
* **session**: Flask надає об'єкт session для зберігання даних, специфічних для сесії користувача. Це дозволяє серверу "пам'ятати" користувача між запитами (наприклад, session['user\_id'], session['user\_role\_id']). Це критично для автентифікації та авторизації.
* **url\_for('reset\_password\_page', token=token, \_external=True)**: Функція url\_for використовується для генерації URL-адрес на основі назв функцій-обробників. \_external=True генерує повний URL, включаючи домен, що необхідно для посилань у email.

**8.4. Висновок**

Серверна архітектура, побудована на Flask, забезпечує надійну та ефективну основу для функціонування веб-додатку. Завдяки чіткій системі маршрутизації, гнучким механізмам обробки вхідних даних та можливості управління станом користувача через сесії, Flask дозволяє розробляти потужні API, які безперебійно взаємодіють з клієнтською частиною та базою даних, забезпечуючи повноцінний функціонал системи.