

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інформаційних систем

Громачденко Єгор Віталійович,
студент групи AI-233

КУРСОВА РОБОТА
з дисципліни
«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

Спеціальність:
122 Комп'ютерні науки

Освітня програма: Комп'ютерні науки

Керівник:
М.А. Годовиченко

Одеса – 2024

ЗМІСТ

ВСТУП.....	3
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.....	4
1.1 Постановка задачі	4
1.2 Аналіз основних бізнес-процесів або сценаріїв	5
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	9
2.1 Проєктування структури даних (сутності, зв'язки, діаграми класів).....	9
2.2 Опис архітектури застосунку	11
2.3 Опис REST API	12
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ	17
3.1 Моделі (Entity-класи, DTO).....	18
3.2 Репозиторії (інтерфейси доступу до даних)	18
3.3 Сервіси (бізнес-логіка).....	19
3.4 Контролери (обробка HTTP-запитів).....	20
3.5. Конфігурація (підключення OAuth2).....	21
4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ.....	23
4.1 Методика тестування	23
4.2 Приклади тестування в Postman.....	23
ВИСНОВКИ	30
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	31
ДОДАТКИ	32

ВСТУП

Сучасні заклади громадського харчування потребують ефективних рішень для автоматизації процесів обробки замовлень, керування меню та обліку персоналу. Саме тому створення інформаційної системи для управління рестораном є актуальним завданням, що підвищує якість обслуговування та зменшує вплив людського фактору.

Метою курсової роботи є закріплення знань з дисципліни «Об'єктно-орієнтоване програмування» та здобуття практичних навичок створення серверної частини прикладного програмного забезпечення мовою Java із застосуванням фреймворку Spring.

У роботі реалізовано серверну частину системи управління рестораном із підтримкою CRUD-операцій для страв, категорій, замовлень, клієнтів та офіціантів. Архітектура додатку побудована на принципах розділення на контролери, сервіси та репозиторії з використанням об'єктної моделі.

Для реалізації використано:

- Java — основна мова програмування;
- Spring Boot — для побудови RESTful API;
- Spring Data JPA + Hibernate — для роботи з PostgreSQL;
- Spring Security + OAuth2 — для Google-автентифікації;
- Postman — для тестування запитів;
- Git + GitHub + Render — для керування кодом та хостингу.

Обрані технології забезпечують масштабованість, зручність у підтримці й відповідають вимогам сучасної backend-розробки. Курсовий проєкт дозволяє практично реалізувати знання з ООП, роботи з базами даних і архітектури вебзастосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі

У сучасних умовах розвитку ресторанного бізнесу особливу увагу приділяють автоматизації процесів обслуговування клієнтів, обліку замовлень, управління меню та адміністрування персоналу. Ресторанна індустрія стикається з необхідністю оптимізувати внутрішні процеси, зменшити витрати часу на виконання рутинних завдань та забезпечити високий рівень обслуговування клієнтів.

З цією метою ставиться задача розробки інформаційної системи управління рестораном, яка дозволяє:

- Реєструвати клієнтів і персонал (офіціантів), розмежовуючи їхні ролі в системі.
- Формувати замовлення, додавати та вилучати страви із замовлення.
- Зберігати інформацію про меню, категорії страв, ціни, популярність.
- Проводити облік обслуговування клієнтів конкретними офіціантами.
- Здійснювати обчислення загальної вартості замовлення.
- Забезпечити автентифікацію користувачів з використанням JWT і Google OAuth2.

Мета розробки: створити RESTful вебзастосунок, який надає зручний і безпечний інтерфейс для управління даними ресторану, враховуючи особливості організації внутрішніх процесів обслуговування клієнтів.

Основні функціональні вимоги:

- Користувач має можливість зареєструватися, увійти до системи та отримати токен доступу.
- Офіціанти можуть створювати, змінювати й завершувати замовлення.
- Адміністратор має змогу керувати стравами, категоріями, користувачами.
- Система повинна фіксувати популярні страви для подальшого аналізу.

- Усі дії мають бути захищені через авторизацію відповідно до ролей (USER/ADMIN).

Ця задача охоплює одразу кілька аспектів інженерії програмного забезпечення: проектування структури даних, визначення архітектури програми, реалізацію рівнів взаємодії (Controller–Service–Repository), а також побудову захищеної системи з використанням сучасних протоколів безпеки.

1.2 Аналіз основних бізнес-процесів або сценаріїв

У системі управління рестораном реалізовано кілька ключових бізнес-процесів, які забезпечують повноцінне функціонування додатку. Оскільки в системі є лише дві ролі (ADMIN і USER), кожен бізнес-сценарій побудовано відповідно до повноважень відповідного користувача.

Сценарій 1: Реєстрація та авторизація користувача

- Учасники: Користувач (USER або ADMIN)
- Опис процесу: Користувач створює обліковий запис шляхом надання унікального імені користувача (username) та пароля. Після цього він може авторизуватись за допомогою введених даних або Google OAuth2.
- Мета: Надати користувачеві безпечний доступ до функціоналу системи з урахуванням його ролі.

Сценарій 2: Перегляд меню та категорій (USER, ADMIN)

- Учасники: Усі зареєстровані користувачі
- Опис процесу: Користувач надсилає GET-запити до /api/dishes для перегляду всіх доступних страв, а також до /api/categories для перегляду категорій.
- Мета: Надати доступ до інформації про меню ресторану.

Сценарій 3: Створення замовлення (USER)

- Учасники: Користувач (USER)
- Опис процесу: Користувач створює нове замовлення шляхом POST-запиту до `/api/orders`, вказуючи ідентифікатор клієнта. Після створення замовлення він може додавати до нього страви.
- Мета: Дозволити клієнту сформувати індивідуальне замовлення.

Сценарій 4: Додавання або видалення страв із замовлення (USER)

- Учасники: Користувач (USER)
- Опис процесу: Користувач надсилає PUT-запити на `/api/orders/{orderId}/add-dish/{dishId}` для додавання та `/api/orders/{orderId}/remove-dish/{dishId}` для видалення страви.
- Мета: Забезпечити гнучке формування замовлення.

Сценарій 5: Підтвердження замовлення (USER)

- Учасники: Користувач (USER)
- Опис процесу: Користувач надсилає PUT-запит на `/api/orders/{orderId}/complete`, після чого замовлення позначається як завершене.
- Мета: Дозволити користувачеві підтвердити виконане замовлення.

Сценарій 6: Управління стравами та категоріями (ADMIN)

- Учасники: Адміністратор (ADMIN)
- Опис процесу: Через POST, PUT і DELETE-запити до `/api/dishes` та `/api/categories`, адміністратор створює, оновлює або видаляє записи.
- Мета: Забезпечити актуальність меню та класифікацію страв.

Сценарій 7: Отримання статистики замовлень (ADMIN)

- Учасники: Адміністратор (ADMIN)

- Опис процесу: GET-запит до `/api/orders/popular` повертає найпопулярніші страви на основі кількості замовлень.
- Мета: Отримати аналітичні дані для прийняття управлінських рішень.

Сценарій 8: Перегляд усіх замовлень (ADMIN)

- Учасники: Адміністратор (ADMIN)
- Опис процесу: GET-запит до `/api/orders` повертає список усіх замовлень у системі.
- Мета: Забезпечити адміністратору повний контроль над активністю користувачів.

1.3 Опис моделі користувача або типової поведінки системи

У системі реалізовано дві ролі користувачів: USER та ADMIN, кожна з яких має чітко визначені повноваження. Модель користувача базується на сутності User, що зберігає основні аутентифікаційні та рольові дані.

Користувач (USER):

Це основний тип користувача системи, який взаємодіє з функціоналом створення та перегляду замовлень. Типова поведінка користувача полягає в наступному:

1. Авторизація: через стандартну форму або за допомогою Google OAuth2;
2. Перегляд меню: доступ до всіх страв і категорій;
3. Формування замовлення:
 - створення нового замовлення;
 - додавання або видалення страв;
 - підтвердження замовлення;
4. Перегляд власних замовлень: можливість переглянути історію замовлень;
5. Без доступу до редагування системи: користувач не має доступу до створення або видалення страв і категорій.

Таким чином, USER — це роль, яка дозволяє використовувати систему для типових завдань клієнта ресторану або замовника.

Адміністратор (ADMIN):

Адміністратор має повний доступ до функціональності системи. Це технічна або адміністративна роль, яка виконує наступні дії:

1. Повне управління користувачами (опціонально): можливість бачити зареєстрованих користувачів;
2. Створення, оновлення та видалення:
 - страв;
 - категорій;
 - замовлень;
3. Отримання статистики:
 - перегляд найпопулярніших страв;
 - перегляд усіх замовлень незалежно від автора;
4. Перевірка стану системи: адміністратор може перевірити, які замовлення були завершені, які — в процесі, та контролювати їх зміст.

Роль ADMIN дозволяє реалізувати повний супровід і технічне обслуговування ресторанної системи.

Типова поведінка системи

Система побудована відповідно до архітектурної моделі MVC (Model–View–Controller) і виконує наступні кроки при типовій взаємодії:

1. Аутентифікація користувача: через форму або Google OAuth2;
2. Авторизація: перевірка ролі, вказаної у JWT-токені;
3. Виконання запиту: залежно від типу користувача й маршруту;
4. Формування відповіді: у вигляді JSON, із відповідним HTTP-статусом;
5. Обробка винятків: у випадку помилок — повернення зрозумілого повідомлення (наприклад, Unauthorized, Forbidden, Not Found).

Це забезпечує чітке розмежування повноважень і стабільну поведінку системи при зміні ролей, запитів або сценаріїв використання.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування структури даних (сутності, зв'язки, діаграми класів)

У процесі розробки інформаційної системи управління ресторанными замовленнями було здійснено детальне проєктування структури даних, що визначає логіку зберігання та взаємодії об'єктів предметної області. Архітектура даних реалізована з використанням об'єктно-реляційного підходу на основі бібліотеки JPA (Java Persistence API). Нижче наведено перелік ключових сутностей, їх атрибути та логічні зв'язки між ними.

Основні сутності:

1. User (Користувач):

Призначений для представлення зареєстрованого користувача системи.

Основні поля:

- id: Long – унікальний ідентифікатор користувача;
- username: String – логін користувача;
- password: String – захищений пароль;
- role: Role – роль користувача в системі (USER або ADMIN).

2. Dish (Страва):

Відповідає окремій одиниці меню.

Основні поля:

- id: Long – унікальний ідентифікатор страви;
- name: String – назва страви;
- price: BigDecimal – ціна страви;
- category: DishCategory – категорія, до якої належить страва.

3. DishCategory (Категорія страв):

Містить назви категорій, що використовуються для групування страв.

Основні поля:

- id: Long – унікальний ідентифікатор категорії;
- name: String – назва категорії (наприклад: «Гарячі страви», «Десерти»).

4. Order (Замовлення):

Представляє процес замовлення страв користувачем.

Основні поля:

- id: Long – унікальний ідентифікатор замовлення;
- customer: User – користувач, який зробив замовлення;
- items: List<Dish> – перелік обраних страв;
- completed: boolean – статус виконання замовлення.

Взаємозв'язки між сутностями:

- Зв'язок між User та Order: один-до-багатьох (*OneToMany*). Один користувач може створити багато замовлень.
- Зв'язок між Order та Dish: багато-до-багатьох (*ManyToMany*). Одне замовлення може містити багато страв, і одна страва може бути частиною різних замовлень.
- Зв'язок між Dish та DishCategory: багато-до-одного (*ManyToOne*). Кожна страва належить до однієї категорії.

Діаграма класів (словесний опис):

Клас User асоційований з множиною об'єктів типу Order. Клас Order містить колекцію типу List<Dish>, що реалізує двосторонній зв'язок між замовленням і стравами. Клас Dish вміщує посилання на об'єкт DishCategory, що дозволяє групувати страви. Таким чином, дані організовані у вигляді цілісної ієрархічної моделі з чітко визначеними асоціаціями.

Програмна реалізація:

Зв'язки реалізовано за допомогою відповідних JPA-анотацій:

- @OneToMany(mappedBy = "customer") у класі User;

- @ManyToMany з таблицею зв'язку між Order та Dish;
- @ManyToOne у класі Dish для визначення категорії страви.

2.2 Опис архітектури застосунку

У даному програмному забезпеченні реалізована багаторівнева архітектура, що чітко розділяє відповідальність між різними компонентами системи. Це забезпечує масштабованість, гнучкість, легкість в тестуванні, підтримці та розширенні функціональності. Архітектура включає три основні рівні:

1. Controller Layer (Рівень контролерів)
2. Service Layer (Рівень сервісів)
3. Repository Layer (Рівень доступу до даних)

1. Controller Layer

Контролери є вхідною точкою до системи. Вони обробляють HTTP-запити, передають дані на рівень сервісів та повертають HTTP-відповіді клієнту. Контролери не містять бізнес-логіки – лише маршалізують запити та відповіді.

Приклади контролерів:

- AuthController – обробляє реєстрацію, авторизацію користувачів.
- OrderController – дозволяє створювати, змінювати та переглядати замовлення.
- DishController – управляє створенням, редагуванням та видаленням страв.
- UserController – (опціонально) управляє користувачами для адміністратора.

2. Service Layer

Сервіси містять бізнес-логіку застосунку. Вони виконують валідацію, обробку даних, приймають рішення, взаємодіють із репозиторіями та обробляють результати.

Приклади сервісів:

- AuthService – реєстрація, логін, хешування паролів, генерація JWT.
- OrderService – додавання/видалення страв, підрахунок загальної суми замовлення, завершення замовлення.

- DishService – прив’язка страви до категорії, оновлення даних.
- JwtService – генерація та перевірка JWT-токенів.

Сервіси використовують `@Service` та `@RequiredArgsConstructor` для автоматичної ін’єкції залежностей через конструктор.

3. Repository Layer

Репозиторії реалізують взаємодію з базою даних через Spring Data JPA. Вони забезпечують CRUD-операції для сутностей.

Приклади репозиторіїв:

- UserRepository – пошук користувачів за ім’ям.
- OrderRepository – знаходження замовлень за клієнтом, офіціантом, а також запит найпопулярніших страв.
- DishRepository – управління сутністю страв.
- DishCategoryRepository – робота з категоріями страв.

Залежності між компонентами:

- Контролери взаємодіють тільки з сервісами.
- Сервіси викликають методи репозиторіїв.
- Репозиторії безпосередньо працюють з базою даних.
- JwtAuthenticationFilter використовує UserDetailsServiceImpl, який у свою чергу – UserRepository.
- Авторизація користувачів заснована на фільтрації запитів за допомогою JWT.

2.3 Опис REST API

REST API (Representational State Transfer Application Programming Interface) — це інтерфейс взаємодії між клієнтом і сервером, що базується на принципах HTTP-протоколу та CRUD-операціях. У рамках даного застосунку реалізовано повноцінний REST API, який надає функціональність для управління користувачами, замовленнями, стравами, категоріями та автентифікацією.

Загальні принципи:

- Запити класифікуються за HTTP-методами: GET, POST, PUT, DELETE.
- Дані передаються у форматі JSON.
- Аутентифікація здійснюється через JWT (Bearer-токен).
- Захищені запити вимагають заголовок Authorization: Bearer <token>.

Таблиця 2.1 – Маршрути автентифікації (/api/auth/)

Метод	Шлях	Призначення	Тіло запиту	Доступ
POST	/api/auth/register	Реєстрація користувача	{ "username", "password" }	ALL
POST	/api/auth/login	Логін та отримання JWT токена	{ "username", "password" }	ALL
DELETE	/api/auth/delete-all	Видалення всіх користувачів		ADMIN

Таблиця 2.2 – Робота із замовленнями (/api/orders/)

Метод	Шлях	Призначення	Аунтифікація
GET	/api/orders	Отримати всі замовлення	USER/ADMIN
GET	/api/orders/{id}	Отримати замовлення за ID	USER/ADMIN
GET	/api/orders/customer/{Id}	Отримати замовлення певного клієнта	USER/ADMIN
GET	/api/orders/waiter/{Id}	Отримати замовлення	ADMIN

		певного офіціанта	
POST	/api/orders	Створити нове замовлення	USER/ADMIN
PUT	/api/orders/{orderId}/add-dish/{dishId}	Додати страву до замовлення	USER/ADMIN
PUT	/api/orders/{orderId}/remove-dish/{dishId}	Видалити страву з замовлення	USER/ADMIN
GET	/api/orders/{orderId}/total	Отримати загальну суму замовлення	USER/ADMIN
PUT	/api/orders/{orderId}/complete	Завершити замовлення	USER/ADMIN
DELETE	/api/orders/{id}	Видалити замовлення	USER/ADMIN

Таблиця 2.3 – Робота зі стравами (/api/dishes/)

Метод	Шлях	Призначення	Аунтифікація
GET	/api/dishes	Переглянути всі страви	USER/ADMIN
POST	/api/dishes	Створити нову страву	ADMIN
PUT	/api/dishes/{id}	Оновити страву	ADMIN
DELETE	/api/dishes/{id}	Видалити страву	ADMIN
GET	/api/dishes/popular	Отримати найпопулярніші страви	ALL

POST	/api/dishes/{dId}/category/{cId}	Призначити категорію для страви	ADMIN
------	----------------------------------	---------------------------------	-------

Таблиця 2.4 – Робота з категоріями (/api/categories/)

Метод	Шлях	Призначення	Аутифікація
GET	/api/categories	Переглянути всі категорії	USER/ADMIN
POST	/api/categories	Створити нову категорію	ADMIN
PUT	/api/categories/{id}	Оновити категорію	ADMIN
DELETE	/api/categories/{id}	Видалити категорію	ADMIN

Таблиця 2.5 – Робота з клієнтами (/api/customers/)

Метод	Шлях	Призначення	Аутифікація
GET	/api/customers	Отримати всіх клієнтів	ADMIN
POST	/api/customers	Зареєструвати нового клієнта	ADMIN
PUT	/api/customers/{id}	Оновити клієнта	ADMIN
DELETE	/api/customers/{id}	Видалити клієнта	ADMIN

Таблиця 2.6 – Робота з офіціантами (/api/waiters/)

Метод	Шлях	Призначення	Аутифікація
-------	------	-------------	-------------

GET	/api/waiters	Отримати список офіціантів	ADMIN
POST	/api/waiters	Додати нового офіціанта	ADMIN
PUT	/api/waiters/{id}	Оновити дані офіціанта	ADMIN
DELETE	/api/waiters/{id}	Видалити офіціанта	ADMIN

1. Додавання клієнта:

POST <https://restaurant-management-8wpo.onrender.com/api/customers>

Тіло:

```
{
  "fullName": "John Doe",
  "phone": "+1234567890"
}
```

2. Додавання офіціанта:

POST <https://restaurant-management-8wpo.onrender.com/api/waiters>

Тіло:

```
{
  "name": "Olga Petrenko",
  "shift": "Morning"
}
```

3. Створення замовлення:

POST <https://restaurant-management-8wpo.onrender.com/api/orders>

Тіло:

```
{
```



```
"customerId": 2,
"waiterId": 1
}
```

Захищені запити потребують JWT. У випадку неавторизованого доступу повертається:

```
"error": "Unauthorized"
```

У разі успішних змін - відповіді у форматі:

```
"message": "Успішно виконано"
```

Щоб авторизуватися як адмін потрібно ввести:

```
{
"username": "supadmin",
"password": "adminpass"
}
```

У курсовій роботі також реалізовано аутентифікацію користувачів через зовнішній провайдер Google за допомогою протоколу OAuth2. Це дозволяє користувачам входити до системи, використовуючи свій обліковий запис Google, без необхідності створювати окремий логін і пароль.

Щоб пройти авторизацію через Google, користувачеві необхідно перейти за наступним посиланням:

<https://restaurant-management-8wpo.onrender.com/oauth2/authorization/google>

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Цей розділ містить опис реалізації основних компонентів системи: моделей, репозиторіїв, сервісів, контролерів і конфігураційних класів. Програмна реалізація виконана з використанням фреймворку Spring Boot та реалізує архітектурну модель MVC (Model-View-Controller). Для збереження та обробки даних використовується база даних PostgreSQL, а взаємодія з клієнтом відбувається через REST API.

3.1 Моделі (Entity-класи, DTO)

Модель даних реалізована у вигляді JPA-сутностей. Всі класи мають відповідній анотації @Entity, @Id, @ManyToOne, @OneToMany, @JoinColumn. Наприклад, клас Dish описує страву, яка належить до певної категорії:

```
@Entity
public class Dish {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private double price;

    @ManyToOne
    @JoinColumn(name = "category_id")
    private DishCategory category;
}
```

Для передачі даних між клієнтом і сервером створені DTO-класи, зокрема RegisterRequest, LoginRequest, AuthResponse. Вони не містять бізнес-логіки, а лише структурують вхідні й вихідні дані.

3.2 Репозиторії (інтерфейси доступу до даних)

Репозиторії реалізовано як інтерфейси, що розширюють JpaRepository. Це забезпечує автоматичне створення стандартних методів доступу до бази даних. Наприклад, інтерфейс DishRepository:

```
public interface DishRepository extends JpaRepository<Dish, Long> {
}
```

Для реалізації складніших запитів використано анотацію `@Query`, як у `OrderRepository`, де реалізовано пошук найпопулярніших страв:

```
@Query("SELECT d.name, COUNT(d) FROM Order o JOIN o.items d GROUP
BY d.name ORDER BY COUNT(d) DESC")
List<Object[]> findMostOrderedDishes();
```

3.3 Сервіси (бізнес-логіка)

Бізнес-логіка реалізована в сервісних класах, які інкапсулюють операції над даними, виклики репозиторіїв, а також валідацію і авторизацію. Наприклад, сервіс `OrderService` дозволяє додати страву до замовлення:

```
public void addDishToOrder(Long orderId, Long dishId) {
    Order order = getById(orderId);
    Dish dish = dishRepository.findById(dishId).orElseThrow();
    order.getItems().add(dish);
    orderRepository.save(order);
}
```

Аутентифікація за допомогою JWT токенів виконується в сервісі `JwtService`, який формує токен із вбудованою роллю користувача:

```
public String generateToken(User user) {
    return Jwts.builder()
        .setSubject(user.getUsername())
        .claim("role", user.getRole().name())
```

```

        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION))
        .signWith(key, SignatureAlgorithm.HS256)
        .compact();
    }

```

3.4 Контролери (обробка HTTP-запитів)

Контролери приймають HTTP-запити, викликають методи сервісів і формують відповіді. Наприклад, DishController обробляє CRUD-операції зі стравами:

```

@RestController
@RequestMapping("/api/dishes")
@RequiredArgsConstructor
public class DishController {

    private final DishService service;

    @PostMapping
    public Dish create(@RequestBody Dish d) {
        return service.save(d);
    }

    @PostMapping("/{dishId}/category/{categoryId}")
    public ResponseEntity<String> assignCategory(@PathVariable Long dishId,
        @PathVariable Long categoryId) {
        service.assignCategory(dishId, categoryId);
        return ResponseEntity.ok("Категорію призначено");
    }
}

```

```
}
```

Для захищених запитів контролери використовують рольову перевірку, яка здійснюється через токен.

3.5. Конфігурація (підключення OAuth2)

Конфігураційні параметри системи містяться у файлі `application.yml`, зокрема для підключення до бази даних PostgreSQL, а також OAuth2:

```
security:
  oauth2:
    client:
      registration:
        google:
          client-id: ${GOOGLE_CLIENT_ID}
          client-secret: ${GOOGLE_CLIENT_SECRET}
          scope:
            - openid
            - email
            - profile
          redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"
          client-name: Google
      provider:
        google:
          authorization-uri: https://accounts.google.com/o/oauth2/v2/auth
          token-uri: https://oauth2.googleapis.com/token
          user-info-uri: https://www.googleapis.com/oauth2/v3/userinfo
          user-name-attribute: sub
```

Конфігураційний клас SecurityConfig встановлює фільтрацію запитів, доступи за ролями, інтеграцію JWT та OAuth2:

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())

        .authorizeHttpRequests(auth -> auth

            // Публічні маршрути
            .requestMatchers(
                "/api/auth/**",
                "/oauth2/**",
                "/login/**",
                "/api/dishes/popular"
            ).permitAll()

            // Доступ лише для ADMIN
            .requestMatchers(
                "/api/dishes/**",
                "/api/waiters/**",
                "/api/categories/**"
            ).hasRole("ADMIN")

            // Доступ для USER та ADMIN
            .requestMatchers(
                "/api/orders/**",
                "/api/customers/**"
            ).hasAnyRole("USER", "ADMIN")
        )
}
```

```
// Усі інші запити (лише для авторизованих користувачів)
.anyRequest().authenticated()
)
```

4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

4.1 Методика тестування

Основна мета тестування — переконатися, що всі реалізовані запити REST API працюють згідно зі специфікацією, забезпечують належну обробку вхідних даних, належну перевірку авторизації й повертають відповідні коди статусу HTTP.

4.2 Приклади тестування в Postman

Ремарка щодо тестування API

Під час розробки та первинного тестування REST API-запити перевірялися на локальному сервері за допомогою інструмента Postman, що дозволило швидко і зручно перевірити працездатність основних маршрутів та логіку обробки запитів на етапі відладки.

Для перевірки функціональності застосунку у розгорнутому середовищі, публічний сервер було розміщено на платформі Render. Усі основні маршрути API доступні за базовим URL-адресом:

<https://restaurant-management-8wpo.onrender.com>

Отже, для виконання тестів через Postman необхідно використовувати вищевказану адресу як базову, додаючи до неї відповідні маршрути, наприклад:

POST <https://restaurant-management-8wpo.onrender.com/api/auth/login>

GET <https://restaurant-management-8wpo.onrender.com/api/dishes>

Також до захищених запитів обов'язково потрібно вказати Bearer Token, який видається після успішної автентифікації.

— Реєстрація користувача рис. 4.2.1, створює нового користувача з роллю USER за замовчуванням.

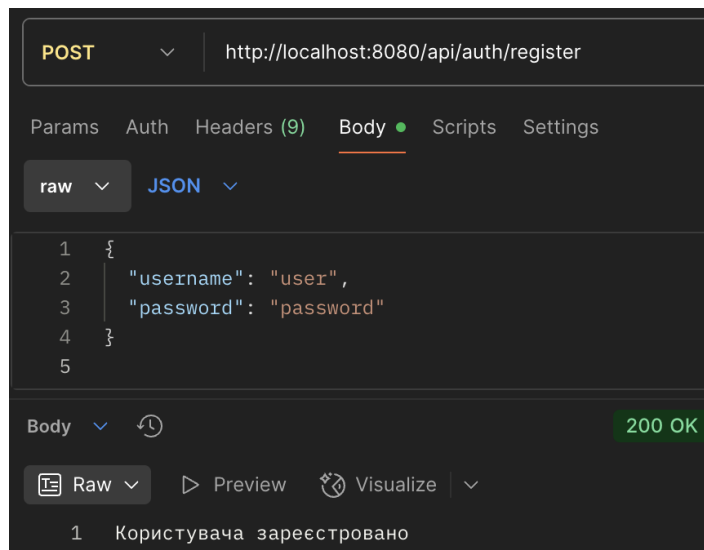


Рис. 4.2.1 – Реєстрація користувача

— Також реалізовано аунтифікацію користувачів через Google за допомогою протоколу OAuth2 рис. 4.2.2, 4.2.3.

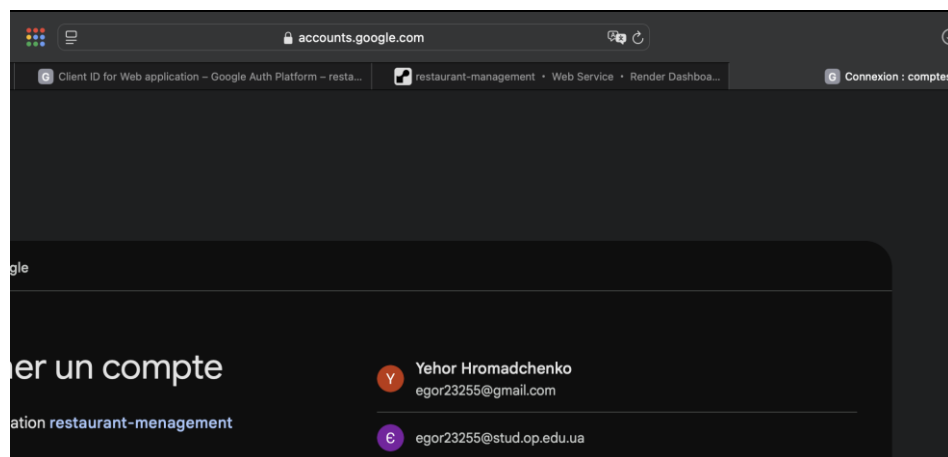


Рис. 4.2.2 – Аунтифікація за допомогою Google аккаунта

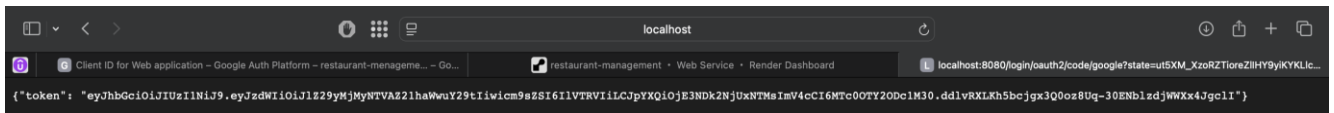


Рис. 4.2.3 – Виведення токену після авторизації

— Вхід користувача рис. 4.2.4, повертає JWT-токен на основі логіну та пароля.

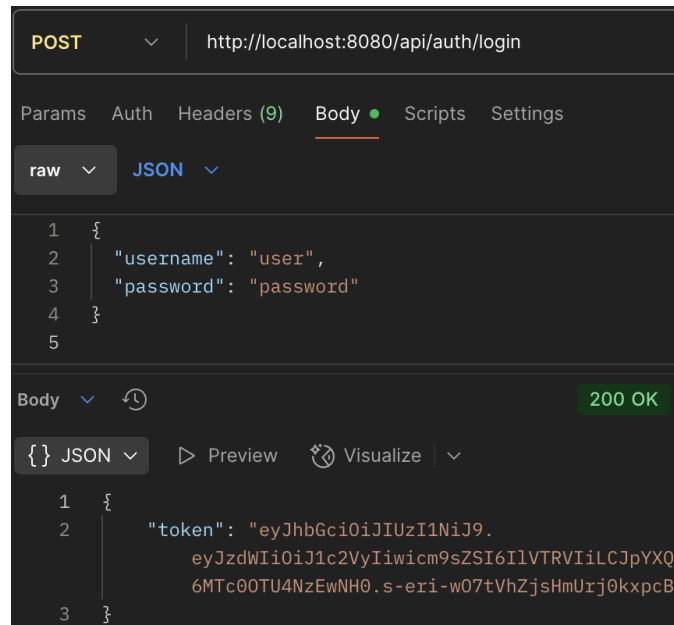


Рис. 4.2.4 – Вхід користувача

— Виведення всіх замовлень рис. 4.2.5, повертає перелік замовлень із БД.

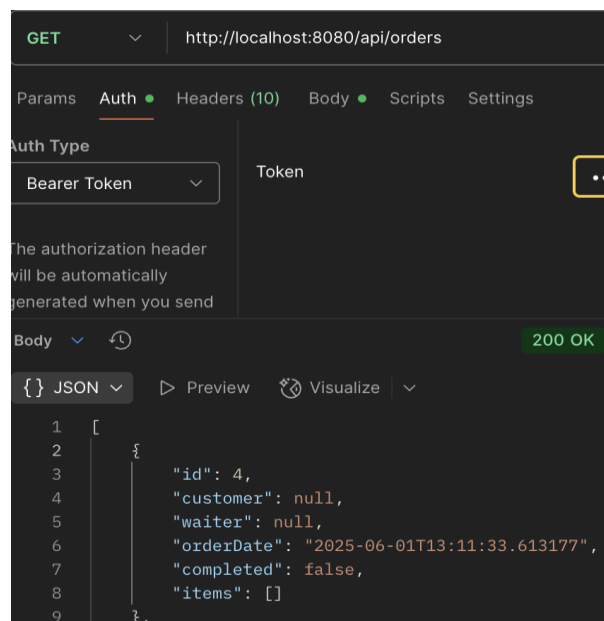


Рис. 4.2.5 – Всі замовлення

— Створення страви як користувач рис. 4.2.6, користувач не має змоги створювати страви закладу, це може зробити тільки адмін рис. 4.2.7.

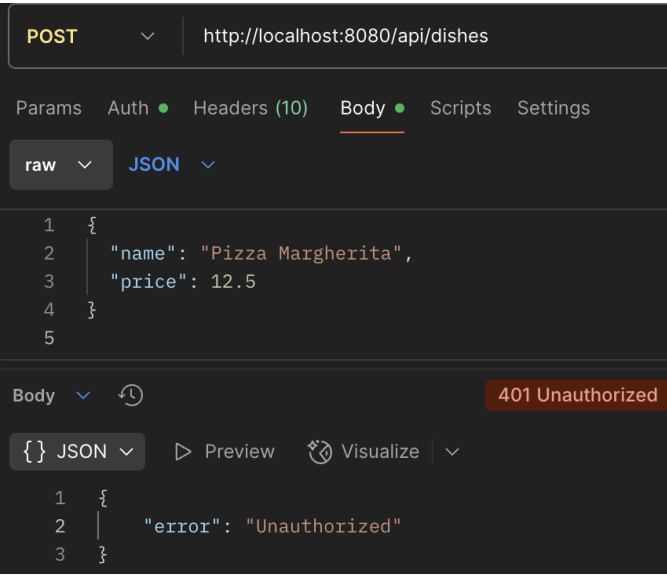


Рис. 4.2.6 – Створення як користувач

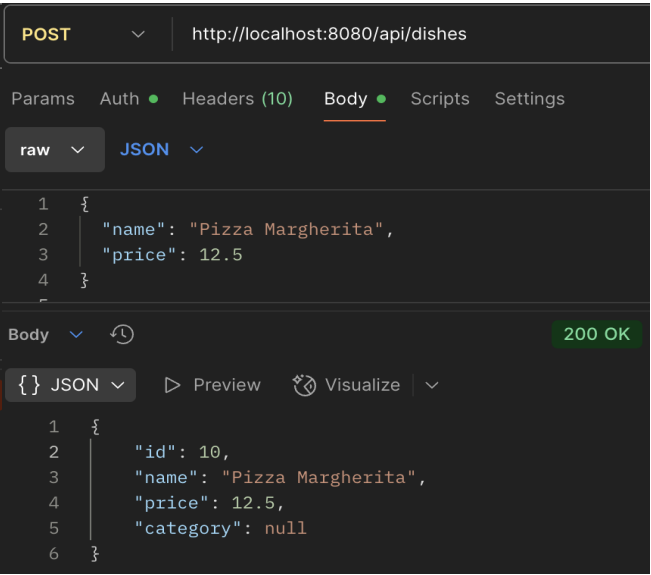


Рис. 4.2.7 – Створення як адмін

— Всі створені страви рис. 4.2.8, повертає перелік страв.

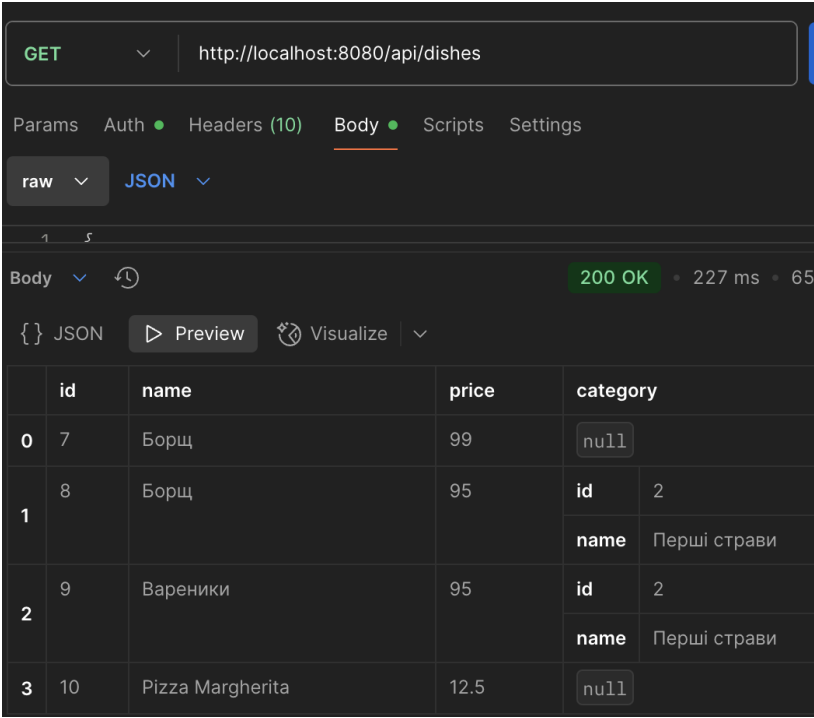


Рис. 4.2.8 – Всі створені страви

— Оновлення страв рис. 4.2.9, дозволяє оновити страви.

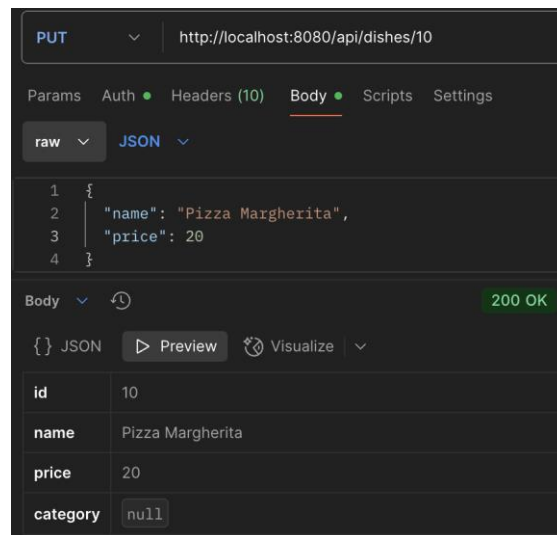


Рис. 4.2.9 – Оновлення страв

— Видалення страв рис. 4.2.10.

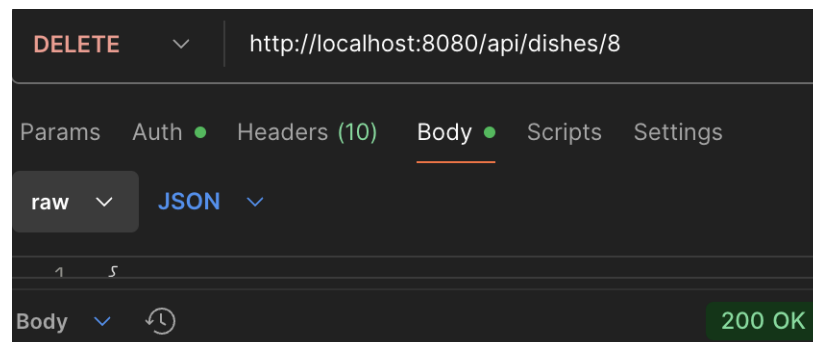


Рис. 4.2.10 – Видалення страв

— Додавання категорії страв рис. 4.2.11, створює нову категорію.

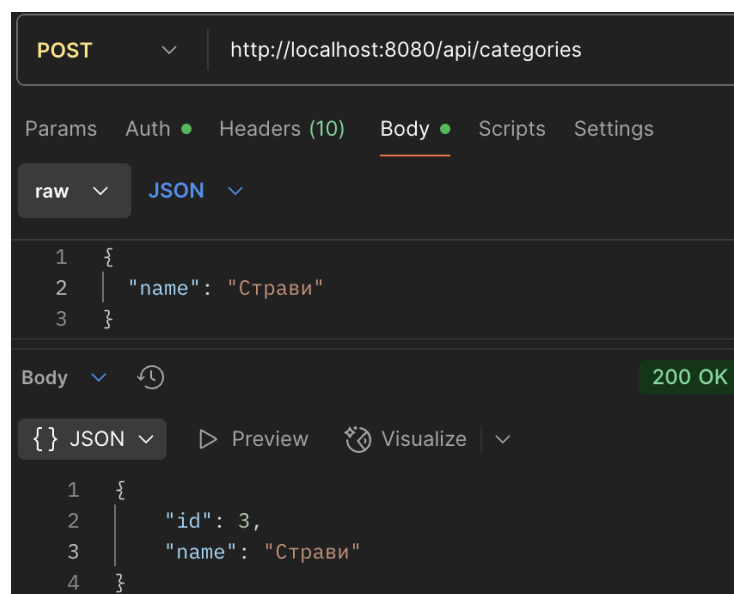


Рис. 4.2.11 – Додавання категорії страв

— Призначення категорії рис. 4.2.12 та рис. 4.2.13, зв’язує страву з певною категорією.

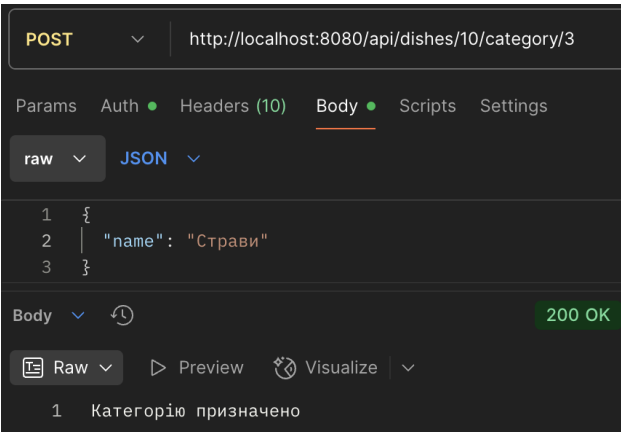


Рис. 4.2.12 – Призначення категорії

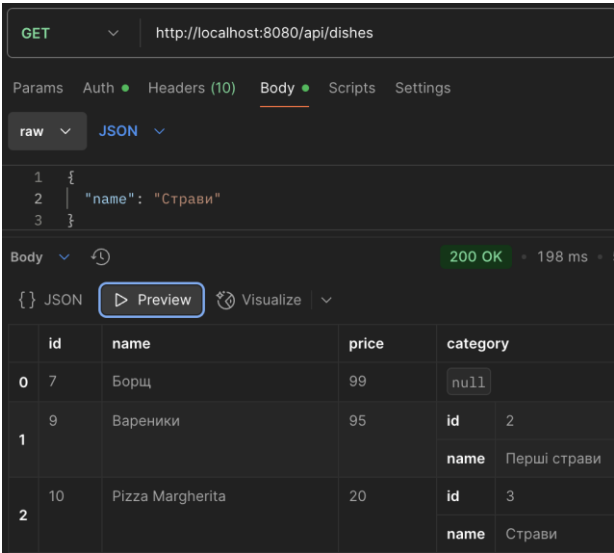


Рис. 4.2.13 – Призначення категорії

— Додавання страви до замовлення рис. 4.2.14, додає обрану страву до певного замовлення.

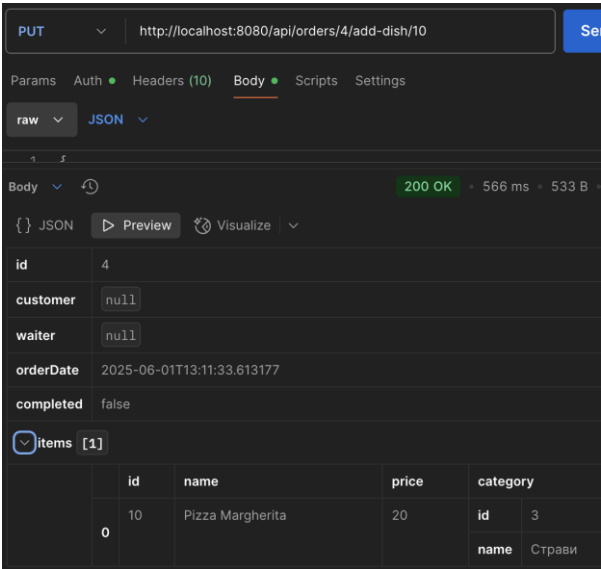


Рис. 4.2.14 – Додавання страви до замовлення

— Завершити замовлення рис. 4.2.15, змінює статус замовлення на completed = true.

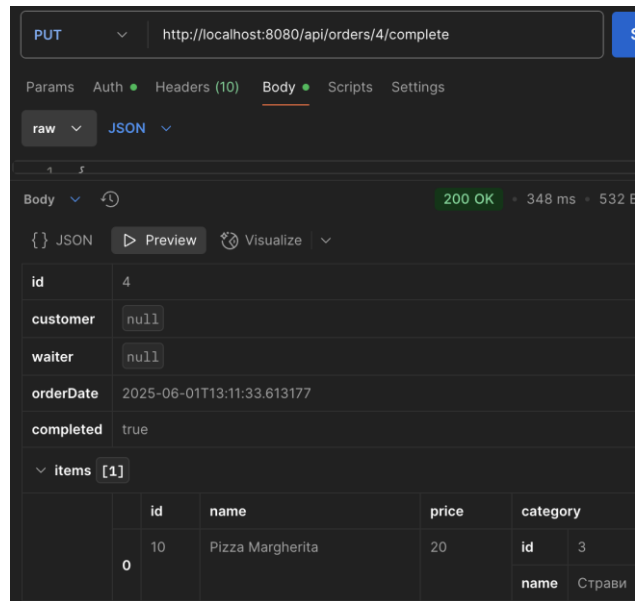


Рис. 4.2.15 – Завершення замовлення

— Найпопулярніші страви рис. 4.2.16, повертає найчастіше замовлені страви.

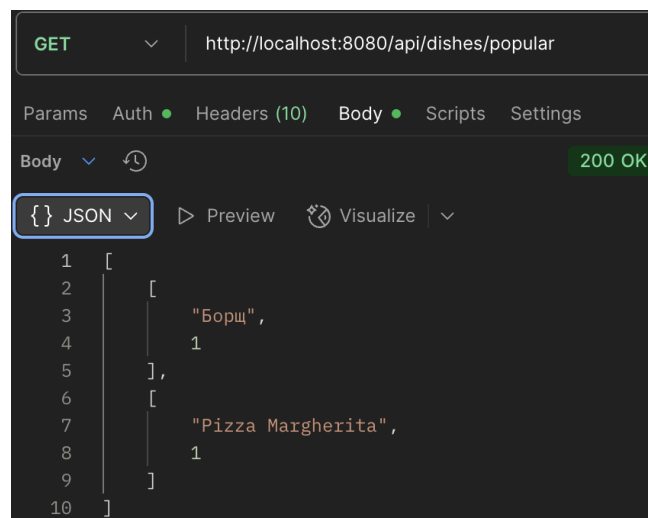


Рис. 4.2.16 – Найпопулярніші страви

У ході розробки програмного забезпечення було здійснено комплексне тестування деяких REST-запитів за допомогою інструменту Postman. Основна мета тестування — перевірка коректності виконання бізнес-логіки, дотримання вимог автентифікації та авторизації, а також відповідності відповіді очікуваному формату. Окрему увагу приділено обмеженню доступу до ресурсів залежно від ролі користувача.

ВИСНОВКИ

У процесі виконання курсової роботи було реалізовано повнофункціональний веб-застосунок для управління ресторанными замовленнями з використанням Java Spring Boot, PostgreSQL та сучасних принципів розробки REST API. Основна мета полягала у створенні зручної та безпечної системи, яка дозволяє обробляти замовлення клієнтів, керувати категоріями страв, додавати нові страви, а також забезпечувати доступ на основі ролей користувачів. Усі ключові функції були успішно реалізовані та перевірені.

У проєкті чітко дотримано архітектурного поділу на рівні Controller–Service–Repository. Такий підхід сприяв підтримуваності та масштабованості системи. Для обробки авторизації та аутентифікації впроваджено два механізми: стандартний вхід за логіном і паролем з видачею JWT-токенів, а також вхід через Google OAuth2, що відповідає сучасним вимогам до зручності й безпеки. Для кожного користувача реалізовано присвоєння ролі (USER або ADMIN), з обмеженням доступу до відповідних REST-ендпоінтів.

Особливу увагу було приділено тестуванню: основні запити перевірено в Postman, налаштовано обробку помилок на рівні HTTP-відповідей, протестовано захищені маршрути, перевірено реакцію системи на неправильну автентифікацію та помилки авторизації. Під час розробки виникали труднощі з конфігурацією OAuth2 та обробкою ролей користувачів, однак усі проблеми були успішно вирішені.

Таким чином, поставлену мету досягнуто повністю. У результаті курсової роботи було отримано практичний досвід побудови веб-сервісу з багаторівневою архітектурою, реалізації безпечної автентифікації, розробки REST API, а також налагодження й тестування функціоналу. Подальший розвиток системи може передбачати створення клієнтського інтерфейсу, генерацію статистичних звітів, підтримку мобільних пристроїв і розширення прав доступу для нових ролей (наприклад, кухарів чи адміністраторів залу).

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Косенко С. І. Spring Boot. Швидкий старт : навч. посіб. Харків : Ранок, 2021. 164 с.
2. Тищенко М. П., Кузьменко Д. В. Розробка вебзастосунків у Java : підручник. Київ : КНУ, 2020. 198 с.
3. Spring Framework Documentation. URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/> (дата звернення: 15.04.2025).
4. Baeldung. Spring Security Tutorial. URL: <https://www.baeldung.com/spring-security> (дата звернення: 12.05.2025).
5. Spring Authorization Server Reference. URL: <https://docs.spring.io/spring-authorization-server/reference/> (дата звернення: 10.05.2025).
6. Postman API Platform. URL: <https://www.postman.com/> (дата звернення: 10.05.2025).
7. OAuth 2.0 для веб-додатків. Google Identity Platform. URL: <https://developers.google.com/identity/protocols/oauth2> (дата звернення: 11.06.2025).

ДОДАТКИ

Перевірка працездатності REST API через Postman

1. Авторизація через логін/пароль

POST /api/auth/login

Тіло запиту (JSON):

```
{  
  "username": "supadmin",  
  "password": "adminpass"  
}
```

У відповідь ви отримаєте:

```
{  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}
```

Скопіюйте його та вставте в Postman у вкладку Authorization → Bearer Token

Права користувача: ADMIN

Цей обліковий запис має повний доступ до всіх маршрутів.

2. Авторизація через Google (OAuth2)

Перейдіть за посиланням:

<https://restaurant-management-8wpo.onrender.com/oauth2/authorization/google>

Після авторизації у браузері буде згенеровано JWT-токен.

Скопіюйте його та вставте в Postman у вкладку Authorization → Bearer Token