МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
Н-Н ІНСТИТУТ ПРОСТОРОВОГО ПЛАНУВАННЯ ТА ПЕРСПЕКТИВНИХ ТЕХНОЛОГІЙ

ЗВІТ
до проєкту з дисципліни
«Архітектура клієнт-серверних застосувань»

Виконали:
Студенти гр. КНМС-41
Антощак М. Н.
Басараб Б. Т.
Раделицький В.В.

Прийняла:
Машевська М. В.

Львів 2025

ЗАВДАННЯ НА ПРОЄКТ

Розробити повнофункціональну систему доставки їжі “Food Delivery System”, що включає:

Клієнтську частину:

- Каталог страв із категоріями, пошуком і фільтрацією
- Систему реєстрації та авторизації користувачів
- Кошик з керуванням позиціями та підрахунком суми
- Оформлення замовлень із вибором адреси й часу доставки
- Історію замовлень та відстеження статусів у реальному часі
- Механізм залишення відгуків після доставлених замовлень

Адміністративну панель:

- Управління меню: створення, редагування, видалення страв і категорій
- Управління замовленнями та зміна їх статусів
- Керування даними ресторану (контакти, графік, опис)
- Модерація відгуків
- Управління користувачами та ролями (client / restaurant_admin / system_admin)
- Перегляд статистики продажів і найпопулярніших страв

Backend API:

- RESTful API з повним покриттям клієнтської та адмін-функціональності
- Система автентифікації та авторизації з JWT і role-based access control
- Робота з базою даних SQLite/SQLAlchemy (користувачі, меню, замовлення, кошики, відгуки, платежі)
- Інтеграція з платіжною логікою (обробка онлайн-оплат карткою)
- Інтеграція з Telegram-ботом для push-сповіщень адміністраторам

ЗМІСТ

ВСТУП

Актуальність теми:

У сучасному світі онлайн-замовлення їжі стало невід'ємною частиною повсякденного життя. Пандемія COVID-19 значно прискорила цифровізацію ресторанного бізнесу, і сьогодні послуги доставки їжі є одним з найбільш затребуваних секторів електронної комерції.

Актуальність розробки власної системи замовлення їжі обумовлена такими факторами:

- Зростання ринку: Глобальний ринок онлайн-доставки їжі демонструє стабільне зростання на 10-15% щороку
- Потреба у цифровізації: Малий та середній ресторанний бізнес потребує доступних рішень для виходу в онлайн
- Оптимізація процесів: Автоматизація приймання та обробки замовлень суттєво підвищує ефективність роботи
- Покращення клієнтського досвіду: Зручний інтерфейс та можливість відстеження замовлення підвищують лояльність клієнтів
- Аналітика та прийняття рішень: Система збору даних дозволяє приймати обґрунтовані бізнес-рішення

Мета та завдання проєкту

Мета проєкту: Розробити повнофункціональну веб-систему для онлайн-замовлення їжі з доставкою, яка забезпечує зручний інтерфейс для клієнтів та ефективні інструменти управління для адміністраторів ресторану.

Основні завдання:

- **Аналіз та проєктування:**
- Дослідити предметну область та визначити функціональні вимоги
- Спроєктувати архітектуру системи (клієнт-сервер)
- Розробити схему бази даних
- **Розробка серверної частини:**
- Створити REST API для всіх бізнес-операцій
- Реалізувати систему автентифікації на основі JWT
- Впровадити рольову модель доступу (RBAC)
- **Розробка клієнтських додатків:**
- Створити клієнтський веб-сайт для замовлень
- Розробити адміністративну панель
- **Забезпечення якості:**
- Провести модульне та інтеграційне тестування
- Забезпечити валідацію даних на всіх рівнях

Об'єкт та предмет дослідження

Об'єкт дослідження: Процеси онлайн-замовлення та доставки їжі в системах електронної комерції.

Предмет дослідження: Методи та технології розробки веб-систем для автоматизації процесу замовлення їжі, включаючи проєктування клієнт-серверної архітектури, розробку RESTful API та створення інтерактивних веб-інтерфейсів.

Розділ 1. Аналіз предметної області та проєктування системи

Опис обраної предметної області

Предметна область "Онлайн-замовлення їжі з доставкою" охоплює комплекс процесів взаємодії між клієнтами (споживачами), рестораном та службою доставки. Система автоматизує такі ключові бізнес-процеси:

Основні учасники системи:

Роль	**Опис**	**Можливості**
Клієнт (Client)	Кінцевий користувач, що замовляє їжу	Перегляд меню, формування кошика, оформлення замовлення, відстеження статусу, написання відгуків
Адміністратор ресторану (Restaurant Admin)	Менеджер ресторану	Управління меню, обробка замовлень, перегляд статистики, модерація відгуків
Системний адміністратор (System Admin)	Головний адміністратор	Повний доступ + управління користувачами та призначення ролей

Бізнес-процеси системи:

- Процес реєстрації та автентифікації:
- Реєстрація нового клієнта
- Вхід в систему (отримання JWT токену)
- Перевірка прав доступу на основі ролей

- Процес формування замовлення:
- Перегляд меню з фільтрацією по категоріях
- Додавання страв до кошика
- Редагування кількості товарів
- Оформлення замовлення з вказанням адреси доставки

- Процес обробки замовлення:
- Прийняття замовлення адміністратором
- Зміна статусів: pending → accepted → preparing → ready → delivering → delivered
- Сповіщення клієнта про зміну статусу

- Процес зворотного зв'язку:
- Можливість залишити відгук на доставлене замовлення
- Оцінка від 1 до 5 зірок з текстовим коментарем

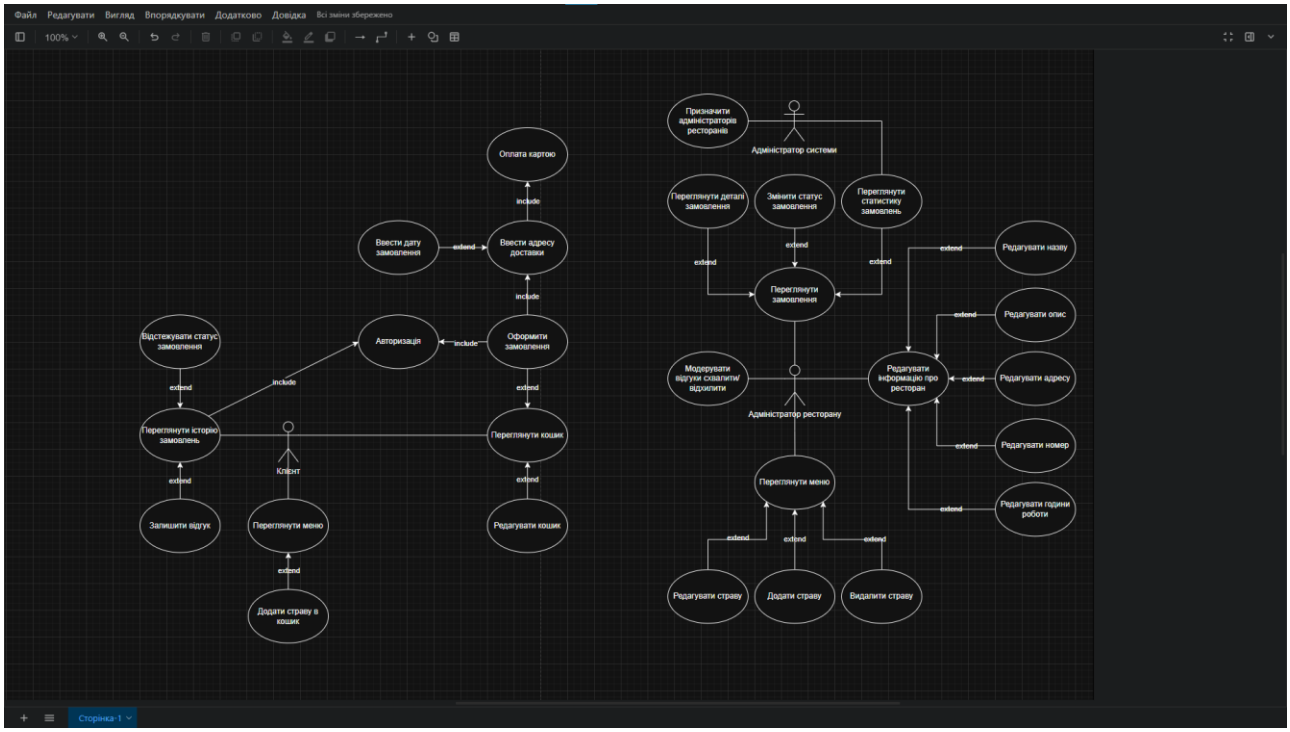
Вимоги до функціоналу

Клієнтська частина має надавати повний цикл самообслуговування: реєстрація, логін, перегляд меню з фільтрами, управління кошиком, оформлення та оплата замовлення, історія та статуси, система відгуків. Адмін-панель повинна забезпечувати CRUD-операції над стравами, категоріями та інформацією про ресторан, повний контроль над замовленнями (перегляд, зміна статусів), перегляд статистики й модерацію фідбеку, управління користувачами та ролями. Бекенд мусить реалізовувати REST API з автентифікацією JWT, ролями, валідацією даних, роботою з БД та інтеграцією з Telegram-ботом і платіжним провайдером.

Обґрунтування вибору технологій

FastAPI та Python 3.14 обрані для бекенду завдяки високій продуктивності, асинхронним можливостям, інтегрованій генерації Swagger-документації та зручній валідації через Pydantic. SQLAlchemy + SQLite забезпечують швидкий старт, ORM-абстракцію і можливість міграції на інші СУБД. React 18 з TypeScript і Vite гарантують компонентну архітектуру, високу продуктивність розробки та типобезпеку як для клієнтського сайту, так і для адмін-панелі. TailwindCSS забезпечує швидке адаптивне оформлення, а React Router 7 та Axios – навігацію й роботу з API. Telegram-бот реалізований через aiogram, що спрощує інтеграцію push-сповіщень.

Діаграма варіантів використання (Use Case Diagram):



На діаграмі показано три актори: Клієнт, Адміністратор ресторану та Адміністратор системи. Для кожного актора виділено ключові сценарії та зв’язки між ними (include/extend), що відображають послідовність дій і залежності.

Клієнт:

- Починає роботу з авторизації; без неї недоступні сценарії кошика, оформлення та перегляду історії (зв’язок include).
- Має базовий сценарій “Переглянути меню”, який розширюється (extend) діями “Додати страву в кошик”.
- Кошик підтримує перегляд і редагування (зміна кількості, видалення).
- Оформлення замовлення включає (include) введення адреси доставки та оплату карткою; додаткова опція – введення бажаної дати доставки (extend).
- Після створення замовлення клієнт може відстежувати статус і переглядати історію; з історії відкривається можливість написати відгук (extend). Усі ці сценарії залежать від авторизованої сесії.

Адміністратор ресторану:

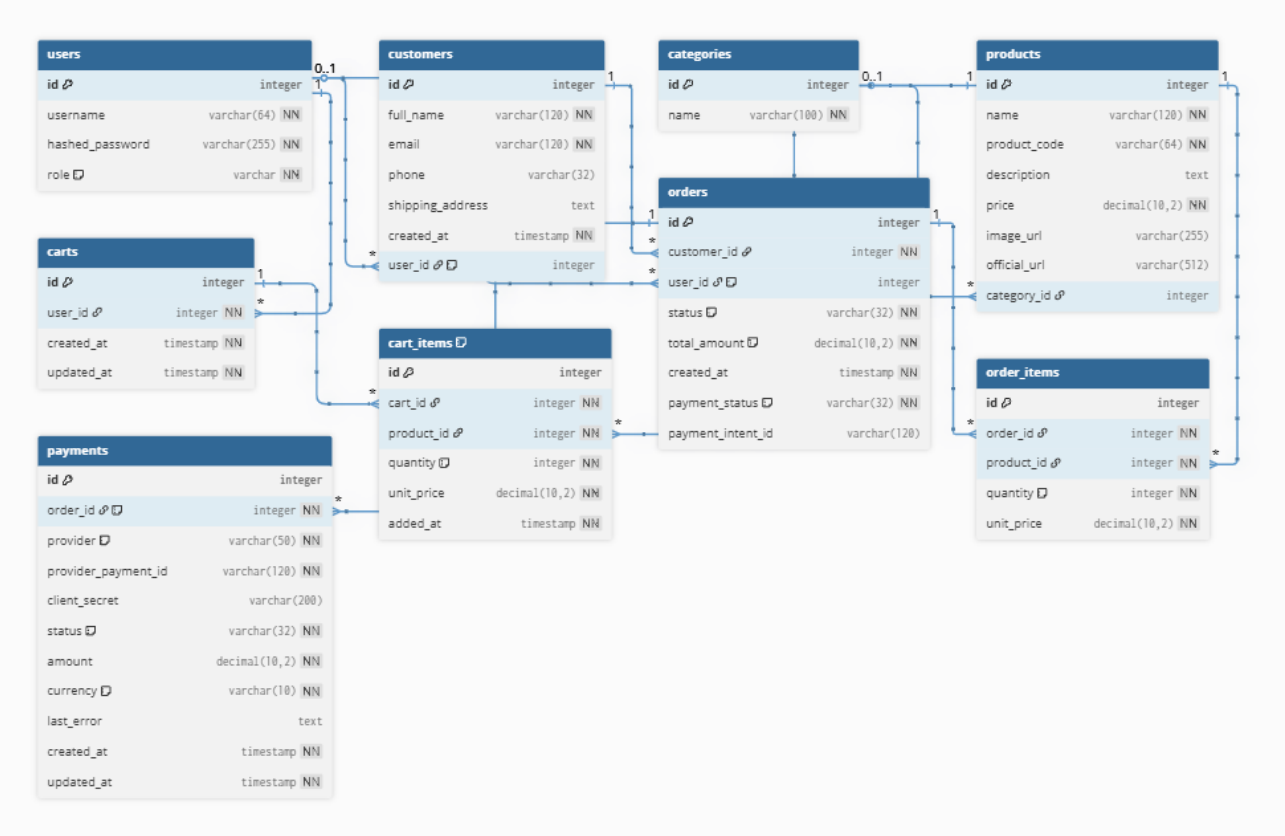
- Центральний сценарій – “Переглянути замовлення”: звідси можна змінювати статуси логістичного ланцюга та переглядати деталі (зв’язки extend).
- Має доступ до модерації відгуків (схвалити/відхилити).
- Працює з меню: сценарії редагування, додавання, видалення страв (extend від “Переглянути меню”).
- Керує інформацією про ресторан – окремі підсценарії редагування назви, опису, адреси, телефону, графіка.

Адміністратор системи:

- Розширює можливості адміністратора ресторану. Окрім перегляду та зміни статусів, має сценарій “Призначити адміністраторів ресторану” і доступ до статистики замовлень.
- Для нього також відкритий сценарій управління користувачами (на діаграмі відображено як частина блоку “Призначити адміністраторів” і “Переглянути замовлення”, що підкреслює контроль над ролями та процесом).

Діаграма демонструє повний життєвий цикл замовлення та засоби керування ним із боку різних ролей, а також відокремлює опціональні та обов’язкові кроки через include та extend.

ER-діаграма бази даних:



Опис основних модулів серверної частини

```
backend/
├── __init__.py
├── main.py                 # Головний файл додатку FastAPI
├── deps.py                 # Залежності (dependency injection)
├── core/
│   ├── config.py          # Конфігурація (JWT_SECRET, тощо)
│   ├── enums.py           # Перерахування (UserRole, OrderStatus)
│   └── security.py        # Хешування паролів, JWT токени
├── database/
│   ├── __init__.py
│   ├── base.py            # Base клас для моделей
│   ├── database.py        # Підключення до БД
│   └── session.py         # Сесії SQLAlchemy
├── models/                 # ORM моделі
│   ├── user.py
│   ├── category.py
│   ├── menu_item.py
│   ├── cart.py
│   ├── order.py
│   ├── order_item.py
│   ├── review.py
│   ├── restaurant.py
│   └── payment.py
├── routers/               # API ендпоінти
│   ├── auth_register.py
│   ├── auth_login.py
│   ├── menu.py
│   ├── cart.py
│   ├── orders.py
│   ├── reviews.py
│   ├── admin_*.py         # Адмін ендпоінти
│   └── ...
├── schemas/               # Pydantic схеми валідації
│   ├── auth.py
│   ├── cart.py
│   ├── order.py
│   └── ...
└── services/
    └── telegram_notifier.py  # Сповіщення через Telegram
```

API Endpoints:

Метод	**Endpoint**	**Опис**	**Авторизація**
POST	/auth/register	Реєстрація користувача	-
POST	/auth/login	Вхід в систему	-
GET	/menu/	Список страв	-
GET	/categories/	Список категорій	-
GET	/cart/me	Отримати кошик	JWT
POST	/cart/me/items	Додати товар	JWT
PUT	/cart/me/items/{id}	Змінити кількість	JWT
DELETE	/cart/me/items/{id}	Видалити товар	JWT
DELETE	/cart/me	Очистити кошик	JWT
POST	/orders	Створити замовлення	JWT
GET	/orders	Мої замовлення	JWT
POST	/orders/{id}/review	Залишити відгук	JWT
GET	/admin/orders	Всі замовлення	Admin
PUT	/admin/orders/{id}/status	Змінити статус	Admin
GET	/admin/stats/overview	Статистика	Admin
GET	/admin/users/	Список користувачів	System Admin

Опис основних компонентів та сторінок клієнтської частини

Структура Client Site:

```
client/src/
├── App.tsx                 # Головний компонент з роутингом
├── main.tsx                # Точка входу
├── index.css               # Глобальні стилі (TailwindCSS)
├── api/
│   └── client.ts           # Axios інстанс для API
├── components/             # Перевикористовувані компоненти
├── contexts/               # React контексти
├── hooks/
│   └── useDebounce.ts      # Хук для debounce пошуку
├── pages/
│   ├── About.tsx           # Про ресторан
│   ├── Cart.tsx            # Сторінка кошика
│   ├── Checkout.tsx        # Оформлення замовлення
│   ├── Login.tsx           # Вхід
│   ├── Register.tsx        # Реєстрація
│   └── Orders.tsx          # Історія замовлень
├── types/
│   └── index.ts            # TypeScript типи
└── utils/
    └── notifications.ts    # Toast сповіщення
```

Структура Admin Panel:

```
admin/src/
├── App.tsx                # Роутинг з PrivateRoute
├── main.tsx
├── index.css
├── api/
│   └── client.ts
├── components/
│   ├── Layout.tsx         # Загальний layout з sidebar
│   ├── Breadcrumbs.tsx    # Навігаційні хлібні крихти
│   ├── KanbanBoard.tsx    # Kanban дошка для замовлень
│   ├── OrdersChart.tsx    # Графік замовлень
│   ├── RatingsSummary.tsx # Зведення рейтингів
│   └── ui/                # UI компоненти
│       ├── Badge.tsx
│       ├── Button.tsx
│       ├── Card.tsx
│       └── Input.tsx
├── pages/
│   ├── Dashboard.tsx      # Головна панель
│   ├── Orders.tsx         # Управління замовленнями
│   ├── Menu.tsx           # Управління меню
│   ├── Categories.tsx     # Категорії
│   ├── Restaurant.tsx     # Налаштування ресторану
│   ├── Reviews.tsx        # Відгуки
│   ├── Users.tsx          # Користувачі (System Admin)
│   └── Login.tsx          # Вхід для адмінів
└── types/
    └── index.ts
```

Сторінки клієнтського сайту:

Сторінка	**Шлях**	**Функціонал**
Home	/	Меню з категоріями, пошук, сортування, додавання в кошик
Login	/login	Форма входу
Register	/register	Форма реєстрації
Cart	/cart	Перегляд кошика, зміна кількості, видалення
Checkout	/checkout	Оформлення замовлення з адресою
Orders	/orders	Історія замовлень, статуси, відгуки
About	/about	Інформація про ресторан

Сторінки адмін-панелі:

Сторінка	**Шлях**	**Функціонал**
Dashboard	/dashboard	Статистика, графіки, топ страви
Orders	/orders	Kanban/Table view, зміна статусів
Menu	/menu	CRUD операції зі стравами
Categories	/categories	Управління категоріями
Restaurant	/restaurant	Редагування інформації
Reviews	/reviews	Перегляд та модерація
Users	/users	Управління користувачами

Опис реалізації ключового функціоналу

1. Система автентифікації та авторизації

JWT (JSON Web Token):

- Токен створюється при успішному логіні
- Містить: username (sub), час створення (iat), час закінчення (exp)
- Термін дії: налаштовується в конфігурації
- Зберігається в localStorage на клієнті

Role-Based Access Control (RBAC):

```
class UserRole(enum.Enum):
    CLIENT = "client"
    RESTAURANT_ADMIN = "restaurant_admin"
    SYSTEM_ADMIN = "system_admin"
```

Захист ендпоінтів:

```
def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
    payload = decode_token(token)
    if not payload:
        raise HTTPException(status_code=401, detail="Invalid token")
    user = db.query(User).filter(User.username == payload["sub"]).first()
    if not user:
        raise HTTPException(status_code=401, detail="User not found")
    return user
def require_admin(current_user: User = Depends(get_current_user)):
    if current_user.role not in [UserRole.RESTAURANT_ADMIN.value, UserRole.SYSTEM_ADMIN.value]:
        raise HTTPException(status_code=403, detail="Admin access required")
    return current_user
```

2. Управління кошиком

Особливості реалізації:

- Кошик створюється автоматично при першому зверненні

- Ізоляція кошиків: кожен користувач бачить тільки свій
- Каскадне видалення: при очищенні видаляються всі cart_items

Процес додавання в кошик:

1. Перевірка автентифікації
2. Отримання/створення кошика
3. Додавання CartItem з прив'язкою до menu_item
4. Збереження ціни на момент додавання (для історії)

3. Обробка замовлень

Процес створення замовлення:

1. Перевірка наявності товарів у кошику
2. Розрахунок загальної суми
3. Валідація адреси та часу доставки
4. Створення Order та OrderItems
5. Очищення кошика
6. Асинхронне сповіщення через Telegram

4. Система відгуків

Бізнес-правила:

- Відгук можна залишити тільки на DELIVERED замовлення
- Один відгук на одне замовлення (unique constraint)
- Рейтинг: 1-5 зірок
- Мінімальна довжина коментаря: 10 символів
- Максимальна довжина: 1000 символів

5. Telegram сповіщення

Реалізація асинхронних сповіщень:

```
async def send_new_order_notification(order_id, total_price, delivery_address, ...):
    """Send notification about new order to admin chat."""
    message = f"""
🆕 **Нове замовлення #{order_id}**

💰 Сума: {total_price/100:.2f} грн
```

📍 Адреса: {delivery_address}

👤 Клієнт: {user_name}

🛒 Позицій: {items_count}

"""

await bot.send_message(ADMIN_CHAT_ID, message, parse_mode="Markdown")

Розділ 3. Інструкція користувача та тестування

Інструкція з розгортання та запуску проєкту

Системні вимоги:

Компонент	**Вимога**
Операційна система	Windows 10+, macOS, Linux
Python	3.13 або новіше
Node.js	18.x або новіше
npm	9.x або новіше
uv	Менеджер пакетів Python

Запуск проєкту:

Автоматичний запуск (Windows):

start-all.bat

Скрипт запустить всі три сервіси одночасно в окремих вікнах.

Ручний запуск:

```
# Terminal 1 - Backend
cd food_delivery
uv run uvicorn backend.main:app --reload
# Terminal 2 - Admin Panel
cd food_delivery/admin
npm run dev
# Terminal 3 - Client Site
cd food_delivery/client
npm run dev
```

Опис основних сценаріїв використання

Сценарій 1: Реєстрація та замовлення їжі (Клієнт)

1. Відкрити сайт http://localhost:5174
2. Перегляд меню:
    - Обрати категорію (Закуски, Основні страви, Десерти)
    - Скористатися пошуком або сортуванням
3. Додати страви в кошик:
    - Натиснути "Додати в кошик"
    - Система попросить увійти
4. Реєстрація:
    - Перейти на /register
    - Ввести username та password
    - Натиснути "Зареєструватися"
5. Вхід:
    - Ввести облікові дані
    - Натиснути "Увійти"
6. Формування кошика:
    - Додати потрібні страви
    - Перейти в кошик (/cart)
    - Відредагувати кількість
7. Оформлення замовлення:
    - Натиснути "Оформити замовлення"
    - Ввести адресу доставки
    - Обрати час доставки (за бажанням)
    - Підтвердити замовлення
8. Відстеження:
    - Перейти в "Мої замовлення"
    - Спостерігати за зміною статусу

Сценарій 2: Обробка замовлень (Адміністратор)

1. Вхід в адмін-панель:

- Відкрити http://localhost:5173
- Ввести логін/пароль адміністратора

2. Перегляд Dashboard:
   - Переглянути статистику
   - Проаналізувати графіки
3. Управління замовленнями:
   - Перейти в Orders
   - Обрати режим перегляду (Таблиця / Kanban)
   - Знайти нові замовлення (pending)
4. Обробка замовлення:
   - Натиснути "View" для деталей
   - Переглянути склад замовлення
   - Змінити статус: pending → accepted → preparing → ready → delivering → delivered
5. Перегляд замовлень у Kanban:
   - Перетягувати картки між колонками
   - Швидко змінювати статуси

Сценарій 3: Управління меню (Адміністратор)

1. Перейти в Menu:
   - Відкрити розділ Menu в бокові панелі
2. Додати нову страву:
   - Натиснути "Add New Item"
   - Заповнити форму (назва, опис, ціна, категорія)
   - Вказати URL зображення
   - Зберегти
3. Редагувати страву:
   - Знайти страву в списку
   - Натиснути "Edit"
   - Внести зміни
   - Зберегти
4. Видалити страву:

- Натиснути "Delete"
- Підтвердити видалення

Сценарій 4: Написання відгуку (Клієнт)

1. Перейти в "Мої замовлення"
2. Знайти доставлене замовлення (статус "delivered")
3. Натиснути "Залишити відгук"
4. Обрати рейтинг (1-5 зірок)
5. Написати коментар (мін. 10 символів)
6. Надіслати відгук

ВИСНОВКИ

У ході виконання курсового проєкту було розроблено повнофункціональну веб-систему замовлення їжі з доставкою "Food Delivery System". Система складається з трьох основних компонентів:

1. Backend API (FastAPI + Python):
    - Реалізовано 20+ REST API ендпоінтів
    - Впроваджено JWT автентифікацію та RBAC
    - Створено 9 моделей бази даних
    - Інтегровано Telegram сповіщення
2. Клієнтський веб-сайт (React + TypeScript):
    - Розроблено 7 сторінок з повним функціоналом
    - Реалізовано адаптивний дизайн
    - Впроваджено пошук, фільтрацію та сортування
3. Адміністративна панель (React + TypeScript):
    - Створено Dashboard зі статистикою
    - Реалізовано Kanban-дошку для замовлень
    - Впроваджено CRUD для всіх сутностей

Можливі напрямки для подальшого розвитку

1. Технічні покращення:
    - Міграція на PostgreSQL для production
    - Впровадження Redis для кешування
    - Docker контейнеризація
    - CI/CD pipeline
2. Функціональні розширення:
    - Мобільний додаток (React Native)
    - Онлайн-оплата (LiqPay, Stripe)
    - Система промокодів та знижок
    - Програма лояльності
3. Інтеграції:

- Google Maps для відстеження доставки
- SMS сповіщення
- Інтеграція з касовими апаратами
- Аналітика (Google Analytics)

4. Масштабування:
    - Підтримка мультиресторанності
    - Мультимовність (i18n)
    - Мікросервісна архітектура

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FastAPI Documentation — https://fastapi.tiangolo.com/ Офіційна документація FastAPI фреймворку.
2. React Documentation — https://react.dev/ Офіційна документація React бібліотеки.
3. TypeScript Handbook — https://www.typescriptlang.org/docs/ Офіційний посібник з TypeScript.
4. SQLAlchemy 2.0 Documentation — https://docs.sqlalchemy.org/ Документація ORM SQLAlchemy.
5. TailwindCSS Documentation — https://tailwindcss.com/docs Офіційна документація TailwindCSS.
6. Pydantic Documentation — https://docs.pydantic.dev/ Документація бібліотеки валідації Pydantic.
7. JWT.io — https://jwt.io/introduction Ресурс про JSON Web Tokens.
8. Python Official Documentation — https://docs.python.org/3/ Офіційна документація Python.
9. Vite Documentation — https://vite.dev/guide/ Документація збирача Vite.
10. React Router Documentation — https://reactrouter.com/ Документація React Router.
11. Axios Documentation — https://axios-http.com/docs/intro Документація HTTP клієнта Axios.
12. pytest Documentation — https://docs.pytest.org/ Документація тестового фреймворку pytest.
13. REST API Design Best Practices — https://restfulapi.net/ Рекомендації з проєктування REST API.
14. OWASP Security Guidelines — https://owasp.org/ Рекомендації з безпеки веб-додатків.

ДОДАТКИ

Додаток А. Лістинг ключових файлів

А.1. Backend: main.py (Точка входу)

```
"""
Main FastAPI application.
"""
import traceback
from fastapi import FastAPI, Request, status
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse

from backend.database import Base
from backend.database.session import engine

# Import models to ensure tables are registered
import backend.models

from backend.routers import (
    auth_register, auth_login, menu, users as users_router,
    restaurants as restaurants_router, admin_restaurants as admin_restaurants_router,
    categories as categories_router, admin_categories as admin_categories_router,
    admin_menu as admin_menu_router, cart as cart_router, orders as orders_router,
    admin_orders as admin_orders_router, payments as payments_router,
    reviews as reviews_router, admin_stats as admin_stats_router,
    admin_users as admin_users_router,
)

# Create FastAPI application
app = FastAPI()
```

```
# Create database tables
Base.metadata.create_all(bind=engine)

# CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
app.include_router(auth_register.router)
app.include_router(auth_login.router)
app.include_router(menu.router)
app.include_router(cart_router.router)
app.include_router(orders_router.router)
# ... інші роутери
```

A.2. Backend: security.py (Безпека)

```
"""
Security utilities for password hashing and JWT token management.
"""
import hashlib
import os
from datetime import datetime, timedelta, timezone
from jose import JWTError, jwt
from backend.core.config import JWT_SECRET_KEY, JWT_ALGORITHM,
JWT_EXPIRES_MINUTES

def hash_password(password: str) -> str:
```

```
    """Hash a password using SHA256 with salt."""
    salt = os.urandom(32).hex()
    pwd_hash = hashlib.sha256((password + salt).encode()).hexdigest()
    return f"{salt}${pwd_hash}"

def verify_password(plain_password: str, stored_password: str) -> bool:
    """Verify a password against its stored hash."""
    try:
        salt, stored_hash = stored_password.split('$', 1)
        pwd_hash = hashlib.sha256((plain_password + salt).encode()).hexdigest()
        return pwd_hash == stored_hash
    except Exception:
        return False

def create_access_token(subject: str) -> str:
    """Create a JWT access token."""
    expire = datetime.now(timezone.utc) +
timedelta(minutes=JWT_EXPIRES_MINUTES)
    to_encode = {"sub": subject, "exp": expire, "iat": datetime.now(timezone.utc)}
    return jwt.encode(to_encode, JWT_SECRET_KEY,
algorithm=JWT_ALGORITHM)

def decode_token(token: str):
    """Decode and validate a JWT token."""
    try:
        payload = jwt.decode(token, JWT_SECRET_KEY,
algorithms=[JWT_ALGORITHM])
        return payload
    except JWTError:
        return None
```

А.3. Backend: orders.py (Управління замовленнями)

```
"""Order management endpoints."""
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from datetime import datetime

from backend.database import get_db
from backend.models.order import Order
from backend.models.order_item import OrderItem
from backend.models.cart import Cart
from backend.deps import get_current_user
from backend.core.enums import OrderStatus, PaymentMethod

router = APIRouter(prefix="/orders", tags=["orders"])

@router.post("", status_code=201)
async def create_order(payload: OrderCreate, db: Session = Depends(get_db),
                       current_user: User = Depends(get_current_user)):
    """Create a new order from cart."""
    cart = db.query(Cart).filter(Cart.user_id == current_user.id).first()
    if not cart or not cart.items:
        raise HTTPException(status_code=400, detail="Cart is empty")

    total_price = sum(item.price * item.quantity for item in cart.items)

    order = Order(
        user_id=current_user.id,
        restaurant_id=DEFAULT_RESTAURANT_ID,
        delivery_address=payload.address,
        payment_method=PaymentMethod.CARD.value,
        total_price=total_price,
```

```
        status=OrderStatus.PENDING.value
    )

    db.add(order)
    db.flush()

    for cart_item in cart.items:
        order_item = OrderItem(
            order_id=order.id,
            menu_item_id=cart_item.menu_item_id,
            quantity=cart_item.quantity,
            price=cart_item.price
        )
        db.add(order_item)

    for item in list(cart.items):
        db.delete(item)

    db.commit()
    return {"message": "Order created", "id": order.id}
```

A.4. Frontend: App.tsx (Клієнтський додаток)

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Login from './pages/Login';
import Register from './pages/Register';
import Cart from './pages/Cart';
import Checkout from './pages/Checkout';
import Orders from './pages/Orders';

function Header() {
  const isLoggedIn = !!localStorage.getItem('client_token');
```

```
  return (
    <header className="glass shadow-sm sticky top-0 z-50">
      <nav className="flex gap-3 items-center">
        <Link to="/">Меню</Link>
        <Link to="/cart">Кошик</Link>
        {isLoggedIn ? (
          <>
            <Link to="/orders">Замовлення</Link>
            <button onClick={handleLogout}>Вихід</button>
          </>
        ) : (
          <Link to="/login">Вхід</Link>
        )}
      </nav>
    </header>
  );
}

function App() {
  return (
    <BrowserRouter>
      <Header />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/cart" element={<Cart />} />
        <Route path="/checkout" element={<Checkout />} />
        <Route path="/orders" element={<Orders />} />
```

```
      </Routes>
    </BrowserRouter>
  );
}
```

A.5. Frontend: Checkout.tsx (Оформлення замовлення)

```
export default function Checkout() {
  const [address, setAddress] = useState('');
  const [deliveryTime, setDeliveryTime] = useState('');
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);

    try {
      const orderData = { address };
      if (deliveryTime) orderData.delivery_time = deliveryTime;

      await api.post('/orders', orderData);
      navigate('/orders');
    } catch (error) {
      showError('Помилка при створенні замовлення');
    } finally {
      setLoading(false);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
```

```
      <input
        type="text"
        required
        value={address}
        onChange={(e) => setAddress(e.target.value)}
        placeholder="Адреса доставки"
      />
      <input
        type="datetime-local"
        value={deliveryTime}
        onChange={(e) => setDeliveryTime(e.target.value)}
      />
      <button type="submit" disabled={loading}>
        {loading ? 'Оформлення...' : 'Підтвердити замовлення'}
      </button>
    </form>
  );
}
```