Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп’ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп’ютерні науки»
зі спеціальності
122 – Комп’ютерні науки

тема роботи:

«Розробка сервісу замовлення їжі у ресторанах міста»

Виконав студент гр. ЗКН-21 _____ Чабанюк Д. В.

Керівник _____ Бугра А. В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студенту групи ЗКН-21 *Чабанюку Денису Володимировичу*

1. Тема кваліфікаційної роботи: *«Розробка сервісу замовлення їжі у ресторанах міста»*

затверджено наказом по університету № *255с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *05.06.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 65 с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Бугра А. В.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.25*
2	*Розділ 1*	*05.04.25*
3	*Розділ 2*	*01.05.25*
4	*Висновки*	*25.05.25*
5	*Оформлення кваліфікаційної роботи*	*28.05.25*
6	*Підготовка презентації та графічного матеріалу*	*20.05.25*
7	*Підготовка доповіді до захисту*	*05.06.25*

6. Дата видачі завдання: *29.01.2025р.*

Керівник _____ / Бугра А. В./

7. Запевнення: *Я, Чабанюк Денис Володимирович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Чабанюк Д. В./

АНОТАЦІЯ

Чабанюк Д. В. Розробка сервісу замовлення їжі у ресторанах міста : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 65 с.

Постійне зростання попиту на дистанційні сервіси харчування та невідповідність глобальних платформ локальним потребам зумовлюють актуальність створення вітчизняного веб-сервісу онлайн-замовлення їжі. Розробка такого рішення сприятиме цифровій трансформації регіонального ресторанного бізнесу та підвищенню якості обслуговування користувачів.

Метою роботи є проєктування та обґрунтування архітектури веб-сервісу замовлення їжі з використанням стеку NestJS + Next.js та бази даних MySQL, що забезпечують високу продуктивність, масштабованість та безпеку.

У вступі обґрунтовано доцільність теми та сформульовано завдання дослідження.

Першій розділ присвячений огляду міжнародних та українських сервісів доставки, визначенню їх переваг та недоліків, формулюванню функціональних вимог та обґрунтуванню вибору технологій NestJS та Next.js.

У другому розділі розроблено багаторівневу архітектуру системи, спроектовано серверну та клієнтську частини, описано модулі аутентифікації, каталогу, кошика, розроблено структуру бази даних, UML-діаграми та користувацькі інструкції.

Практична цінність полягає у створенні масштабованої проектної основи, придатної для подальшої реалізації повноцінного комерційного сервісу, що може бути адаптовано малими та середніми ресторанами для розширення каналів продажу та покращення клієнтського досвіду.

Ключові слова:

ВЕБ-СЕРВІС, ОНЛАЙН-ЗАМОВЛЕННЯ, ПРОГРАМУВАННЯ, ФРЕЙМВОРК, БАЗА ДАНИХ, СУБД, NEXT.JS, NESTJS, PRISMA, OAUTH2

ЗМІСТ

ВСТУП

Сучасний темп урбанізованого життя, зростання електронної комерції та досвід пандемічних обмежень зумовили різке підвищення на дистанційні сервіси харчування. Глобальні платформи, такі як Glovo чи Uber Eats , хоча й задають стандарт швидкого обслуговування, здебільшого не враховують локальні особливості малого та середнього ресторанного бізнесу, а отже не забезпечують повноцінного покриття регіональних потреб. Отже існує актуальна потреба у створенні вітчизняного веб-сервісу, здатного інтегрувати локальні ресторани, оптимізувати логістику доставки та запропонувати користувачам інтуїтивно зрозумілий інтерфейс із функціями реальної години.

Метою даної випускної роботи бакалавра є проєктування та реалізація сервісу онлайн-замовлення їжі для ресторанів міста на базі технологічного стеку Next.js (фронтенд) та NestJS (бекенд) із забезпеченням високої продуктивності, масштабованості та доступності.

Для досягнення поставленої мети необхідно вирішити такі завдання :

– здійснити огляд та критичний аналіз існуючих сервісів, означивши їх функціональні переваги та недоліки;

– сформулювати функціональні та нефункціональні вимоги до розроблюваної системи;

– дослідити можливі технології реалізації та обґрунтувати вибір стеку Next.js + NestJS;

– розробити архітектуру системи та взаємодію її основних компонентів;

– вибрати систему управління базами даних та спроектувати реляційну схему зберігання інформації про користувачів, ресторани, меню та замовлення;

– реалізувати ключовий функціонал клієнтської та серверної частин, включаючи авторизацію, каталог закладів, кошик;

– провести тестування та практичну апробацію сервісу на тестових даних.

Виконання зазначених завдань забезпечити створення конкурентоспроможного веб-сервісу, який гармонійно поєднує локальні потреби ресторанного бізнесу та сучасні очікування користувачів щодо швидкості, зручності та надійності онлайн-доставки. Крім того, очікується, що запропонований підхід сприятиме цифровій трансформації регіонального ресторанного ринку, підвищенню рівня сервісу та зміцненню конкурентоспроможності закладів громадського харчування.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ СЕРВІСУ ЗАМОВЛЕННЯ ЇЖІ ТА ОБГРУНТУВАННЯ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ ПРОЄКТУ

1.1 Аналіз існуючих сервісів замовлення їжі у ресторанах міста

У процесі формування функціональних вимог до сервісу онлайн-замовлення їжі доцільним є проведення аналізу існуючих рішень, що вже функціонують на ринку. Такий аналіз дозволяє не лише визначити ключові можливості подібних сервісів, а й виявити сучасні тенденції в проектуванні користувацьких інтерфейсів, організації взаємодії між клієнтом та рестораном, а також логістики доставки.

Одними з найпоширеніших сервісів у цій галузі є Glovo , Uber Eats та Bolt Food , які активно використовують у великих містах, зокрема в Україні.

Glovo позиціонує собі як мультифункціональна платформа доставки, яка охоплює не лише їжу, але й інші категорії товарів (рис. 1.1).

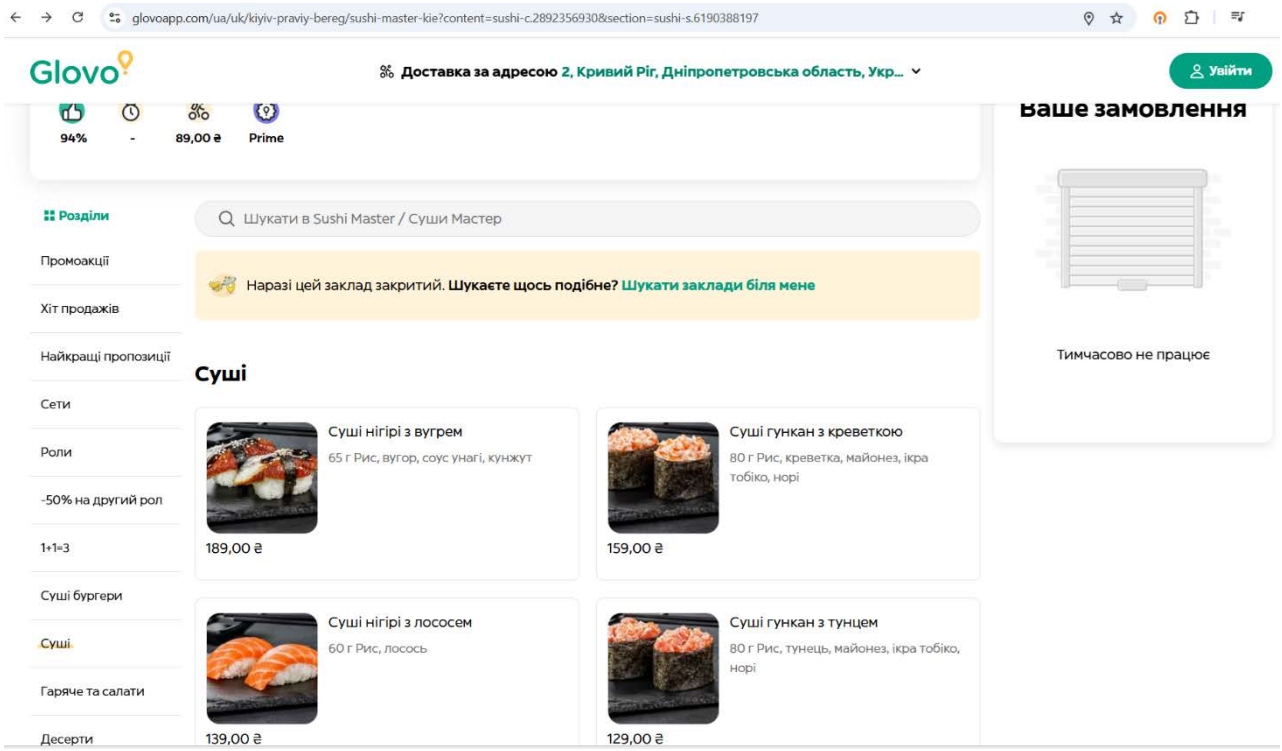

Рисунок 1.1 – Інтерфейс сторінки вибору страви сервісу Glovo

Основні функціональні можливості [1]:

– каталог ресторанів з можливістю пошуку та фільтрації за категоріями, відстанню, ціною та рейтингом;

– підтримка доставки не лише їжі, а й інших товарів (аптеки, супермаркети, документи);

– інтерактивна карта для відстеження замовлення у реальному часі;

– система збереження улюблених товарів та повторних замовлень;

– push-повідомлення про статус замовлення;

– інтеграція з Google Maps.

Основними перевагами платформи Glovo є:

– і багатофункціональність та широкий спектр доставок;

– простий та інтуїтивно зрозумілий інтерфейс;

– висока швидкість обробки замовлень;

– можливість здійснити термінову доставку (кур’єр на вимогу).

Але, можна також виділити і ряд недоліків:

– комерціалізація та висока кількість рекламних елементів;

– часті затримки через перевантаження сервісу у годині пік;

– не завжди коректне відображення часу доставки.

Uber Eats наголошує на персоналізації користувацького досвіду. Користувачам пропонуються детальні профілі ресторанів із фотографіями, меню, рейтингами та відгуками. Додатково, сервіс реалізує систему персональних рекомендацій та програму лояльності, а також дозволяє спілкуватися з кур’єром безпосередньо у застосунку.

Основними функціональними можливостями платформи, що визначають її популярність, є:

– детальні профілі ресторанів з фотографіями страв, описами, рейтингами та відгуками;

– інтелектуальні рекомендації на основі історії замовлень;

– підтримка онлайн-оплати банківською карткою або через мобільні гаманці;

Рисунок 1.2 – Інтерфейс мобільної версії сервісу Uber Eats

– чат з кур’єром для уточнення деталей;

– можливість попереднього замовлення на визначену годину;

– власна програма лояльності (бонуси, скидки, персональні пропозиції).

До основних переваг сервісу можна віднести:

– високу персоналізації взаємодії з користувачем;

– надійність та масштабованість системи;

– зрозумілий та адаптивний інтерфейс;

– великий вибір ресторанів та закладів.

Виконаний аналіз можливостей платформи дозволив також визначити ряд недоліків, зокрема:

– порівняно висока вартість доставки;

– відсутність підтримки готівкової оплати в деяких регіонах;

– повільне реагування служби підтримки.

Bolt Food орієнтований на швидкість обробки замовлень та простоту взаємодії. До переваг сервісу можна віднести мінімалістичний інтерфейс (рис. 1.3), оперативне оновлення статусу доставки, а також підтримку акційних пропозицій та скидок.

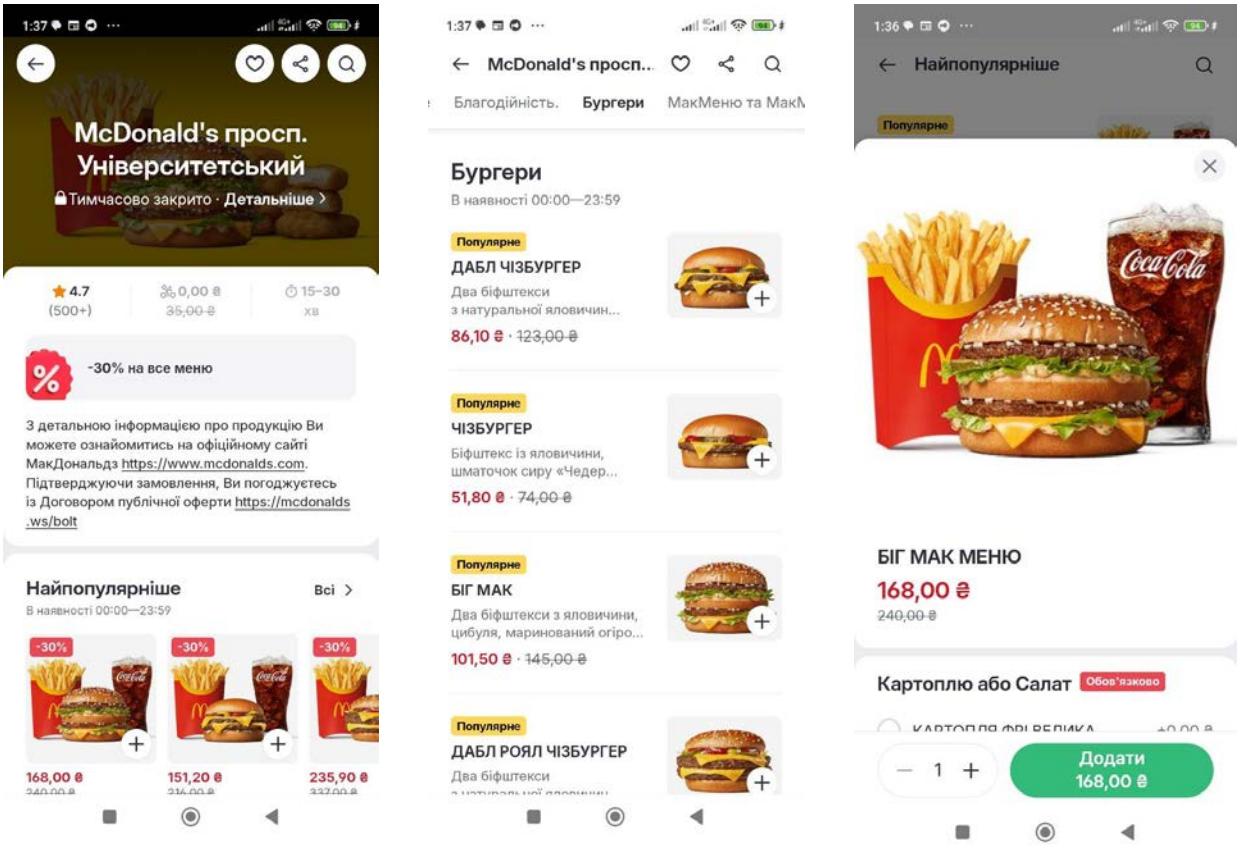

Рисунок 1.3 – Інтерфейс сторінки вибору страви сервісу Bolt Food

До характерних функціональних можливостей можна віднести:

– швидкий перегляд меню з цінами, описом та фото;

– відстеження статусу замовлення (готовиться, в дорозі, доставлено);

– підтримка акцій, промокодів та персональних скидок;

– оплата онлайн чи готівкою;

– простий процес оформлення без необхідності реєстрації (гостьовий режим).

Сервіс має ряд важливих конкурентних переваг на ринку:

– висока швидкість доставки та простота використання;

– часті акції та нижчі ціни порівняно з конкурентами;

– мінімалістичний дизайн, що сприяє зручному користуванню.

До головних недоліків можна віднести:

– обмежений список ресторанів, особливо у малих містах;

– іноді відсутня точна інформація про меню або наявність страв;

– слабка інтеграція з картами (обмежена деталізація маршруту кур'єра).

Серед вітчизняних рішень у сфері онлайн-замовлення їжі вагоме місце посідає сервіс MixFood [2], який функціонує на ринку України з 2010 року. Його діяльність орієнтована переважно на користувачів малих та середніх міст, що дає можливість охопити сегмент, недостатньо представлений у глобальних платформах доставки. Застосунок підтримує як веб-версію, так і мобільне додаток, забезпечуючи доступ до основних функцій без обмежень за платформою.

Основним функціональним блоком MixFood є локалізований каталог ресторанів, піцерій, суші-барів та інших закладів харчування із можливістю сортування за типом кухні, розташуванням чи наявністю доставки (рис. 1.4).

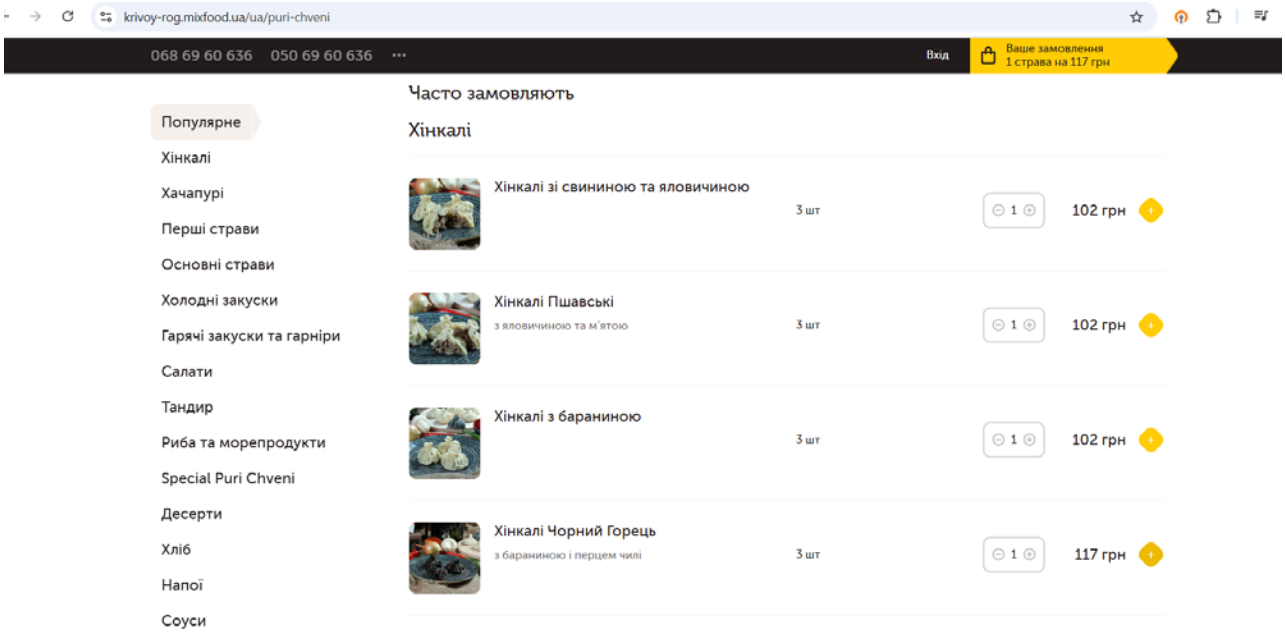

Рисунок 1.4 – Інтерфейс каталогу вибору страв на сайті MixFood

Крім традиційної доставки готових страв, сервіс також підтримує замовлення продуктових наборів, що розширює його цільову аудиторію. Додатковою зручністю є функціонал бронювання столиків, який інтегрований безпосередньо до застосування.Користувачі можуть оформлювати замовлення онлайн, однак у деяких регіонах реалізовано систему зворотного дзвінка (callback), коли після підтвердження замовлення оператор телефонує для уточнення деталей. Такий підхід, хоча і є компромісним рішенням в умовах обмеженої автоматизації, водночас знижує загальну швидкість взаємодії.

Перевагою MixFood є його регіональна адаптація – підтримка локальних закладів та впровадження сервісу в містах, які часто залишаються поза увагою глобальних платформ, таких як Glovo чи Uber Eats. Крім цього, мобільний додаток вирізняється зручним інтерфейсом, наявністю реальних відгуків, а також можливістю зберігати історію замовлень. Відзначається й мультифункціональність сервісу: крім доставки він забезпечує користувачів можливістю бронювання столиків та замовлення продуктів.

Незважаючи на численні переваги, MixFood має і певні недоліки. Зокрема, у низці міст немає повноцінного механізму онлайн-підтвердження замовлення — натомість використовується система зворотного зв'язку з оператором. Також слід відзначити відсутність повноцінного модуля трекінгу замовлення: користувачі не мають змоги візуально відстежувати маршрут доставки в реальному часі. Крім того, в деяких регіонах вибір закладів харчування є обмеженим.

Загалом сервіс MixFood демонструє вдалу адаптацію концепції доставки їжі до умов вітчизняного ринку, зокрема за рахунок охоплення регіонів, де відсутні альтернативні сервіси. Водночас наявні обмеження у функціональності створюють підґрунтя для удосконалення, що може бути враховано під час розробки власного рішення в межах цієї кваліфікаційної роботи.

На підставі проведеного аналізу можна визначити комплекс функціональних вимог, які мають бути реалізовані у розроблюваній інформаційній системі.

1.2 Розробка функціональних вимог до веб-сервісу замовлення їжі у ресторанах міста

В межах розроблюваної інформаційної системи сервісу онлайн-замовлення їжі клієнтська частина повинна забезпечувати користувачам зручний, інтуїтивно зрозумілий інтерфейс для здійснення повного циклу взаємодії з платформою: від реєстрації та пошуку закладів до оформлення та відстеження доставки замовлення. Основні функціональні можливості, що мають бути реалізовані, наведено нижче.

1. Реєстрація та авторизація користувача – користувач повинен мати можливість створити обліковий запис за допомогою електронної пошти, соціальних мереж або сторонніх сервісів авторизації (OAuth). Також має бути реалізований механізм входу до особистого кабінету з перевіркою автентичності.

2. Перегляд списку доступних ресторанів – після входу в систему користувач повинен мати можливість переглянути перелік доступних закладів харчування з можливістю застосування фільтрів за категорією кухні, рейтингом, ціною, місцезнаходженням тощо.

3. Ознайомлення з меню конкретного закладу – користувач може перейти до профілю окремого ресторану, де відображається повне меню із фотографіями, описами страв, можливістю вибору варіантів порцій, додаткових інгредієнтів тощо.

4. Формування та редагування замовлення – користувач має можливість додавати обрані страви до кошика, редагувати кількість позицій, залишати коментарі до замовлення (наприклад, «без солі») та переглядати загальну вартість з урахуванням доставки.

5. Оформлення замовлення – після фіналізації кошика користувач обирає спосіб доставки (кур’єром або самовивіз), вказує адресу вручну або за допомогою геолокації, зазначає бажану годину доставки, а також обирає спосіб оплати (банківська картка, Google Pay, готівка).

6. Оплата замовлення – у разі онлайн-оплати система повинна перенаправляти користувача до захищеного платіжного шлюзу, після чого здійснюється повернення до інтерфейсу застосунку із підтвердженням замовлення.

7. Відстеження статусу замовлення – користувач має змогу в реальному часі відслідковувати зміну статусів замовлення: «прийнято», «готується», «передано кур’єру», «доставлено». У разі потреби – бачити поточне розташування кур’єра на карті.

8. Отримання сповіщень про зміни статусу замовлення – сервіс повинен надсилати push-повідомлення про кожну зміну стану замовлення або затримки доставки, а також про акції та персоналізовані пропозиції.

9. Можливість залишити відгук та оцінку – після завершення доставки користувач може оцінити якість обслуговування, доставку та страви за п’ятибальною шкалою, залишити коментар, який буде відображено у профілі закладу.

10. Перегляд історії замовлень – у профілі користувача має бути доступ до історії попередніх замовлень з можливістю повторного оформлення одного з них у кілька кліків.

11. Керування особистими даними - користувач повинен мати можливість редагувати власні персональні дані (ПІБ, адреси, спосіб оплати), а також керувати налаштуваннями облікового запису.

12. Комунікація з кур’єром – у процесі доставки користувач має змогу зв'язатися з кур'єром через вбудований чат або короткий VoIP-дзвінок (опційно), що дозволяє уточнити деталі.

13. Отримання персоналізованих рекомендацій – на основі історії замовлень та вподобань користувача застосування формує рекомендації щодо нових страв, акцій або ресторанів.

14. Використання промокодів і бонусів – користувач може застосовувати промокоди або бонусні бали, які зменшують загальну вартість замовлення.

Для візуалізації вказаних вимог було розроблено діаграму варіантів використання UML (рис. 1.5).

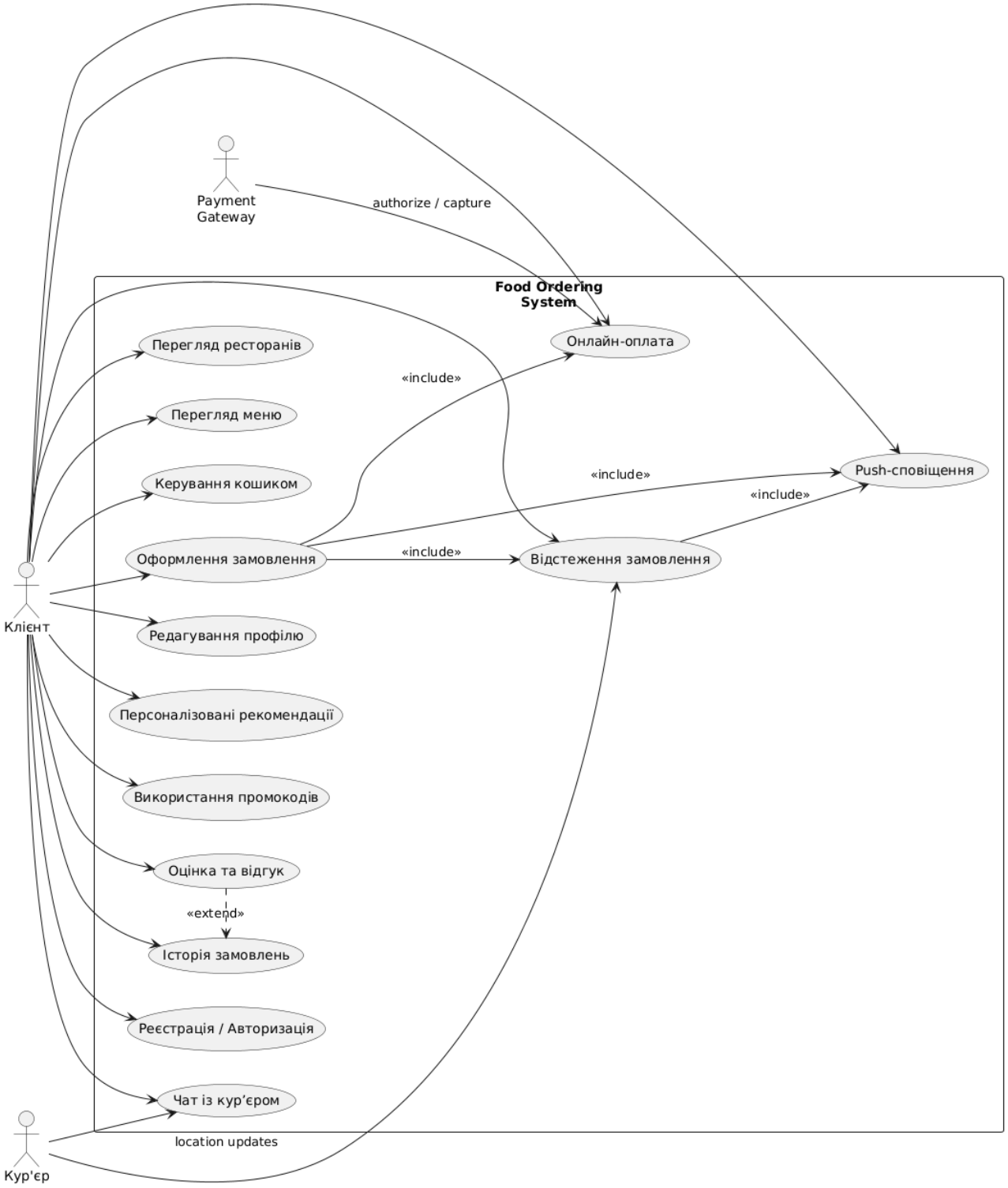

Рисунок 1.5 – Діаграма варіантів використання сервісу доставки їжі

Розглянуті функціональні вимоги користувача також необхідно уточнити з урахуванням особливостей практичної реалізації і перетворення у D-вимоги з відображенням технічних аспектів і пов’язаних нефункціональних вимог. У результаті було сформовано уточнені вимоги до окремих функцій системи, які наведені у табл. 1.1.

Таблиця 1.1 – Уточнені вимоги до функціоналу та особливостям реалізації сервісу замовлення їжі у ресторанах міста

№ з/п	**Функціональна вимога**	**Стислий опис**
1	Каталог закладів	Пошук і фільтрація за типом кухні, ціною, рейтингом, відстанню; сортування за популярністю
2	Інтерактивне меню	Візуальні картки страв, можливість вибору модифікаторів (розмір порції, інгредієнти), відображення алергенів
3	Кошик і повторні замовлення	Додавання/редагування позицій, збереження улюблених наборів, миттєве оновлення вартості
4	Оформлення доставки	Вибір адреси (автозаповнення через Geocoding API), самовивіз або доставка на певний час; поле коментаря
5	Онлайн-оплата	Інтеграція з платіжними шлюзами (банківські картки, Apple Pay, Google Pay); fallback – готівка кур’єру
6	Аутентифікація та ролі	Підтримати реєстрацію через e-mail, соцмережі й OAuth; ролі: клієнт, кур’єр, ресторан
7	Доступність	Контрастні кольори, озвучування для ключових дій, можливість збільшення шрифту
8	Оцінювання та відгуки	Рейтинг закладу й кур’єра (1–5 зірок) + текстовий відгук після завершення доставки
9	Програма лояльності	Система купонів, промокодів, кешбек-балів; персональні «смарт»-пропозиції (рекомендаційний модуль)
10	Безпека та приватність	JWT-сесії, шифрування даних платежу, відповідність PCI DSS; користувацький вибір щодо збору аналітики
Додаткові вимоги		
11	Сповіщення	Push-нотифікації про зміну статусу, акції та персональні рекомендації
12	Комунікація «клієнт - кур’єр»	In-app-чат або VoIP-дзвінок з можливістю обміну фото-/текст-повідомленнями (досвід Uber Eats)
13	Нестабільні мережі	Кеш-first-стратегія (Service Worker) для перегляду меню офлайн; черга відкладених запитів.

Такі вимоги поєднують кращі практики глобальних платформ (глибока персоналізація, трекінг, гнучкі способи оплати) з потребами вітчизняного ринку (регіональна адаптація, підтримка готівки, покриття малих міст) та забезпечують конкурентоспроможність майбутнього сервісу. Частина вимог вказані як додаткові, оскільки їх реалізація не є обов'язковою на першому етапі реалізації проєкту і не чинить значний вплив на основний функціонал, але є бажаною для подальшого розвитку проєкту і забезпечення конкурентних переваг.

1.3 Технічне обґрунтування технологій реалізації сервісу замовлення їжі у ресторанах міста

У процесі проектування інформаційної системи онлайн-замовлення їжі особливу увагу було приділено вибору інструментальних засобів, які зможуть забезпечити ефективну взаємодію між клієнтською, серверною та мережевою складовими системи.

Однією з головних вимог до подібних систем є зручність та швидкодія при взаємодії з користувачем, яка може бути досягнути у першу чергу за рахунок реалізації інтерфейсу з використанням сучасних JavaScript-фреймворків (React, Angular, Vue) разом з WebAPI на стороні сервера. Тому під час вибору засобів реалізації відповідної системи важливо розглядати не лише окремі технології, а й їх взаємодію в контексті архітектурного цілого. З цією метою було виконано порівняльний аналіз кількох популярних стеків для реалізації веб-застосунків, зокрема:

– Next.js + NestJS;

– React SPA + Express.js;

– Vue.js + Laravel;

– Angular + Spring Boot.

Порівняння виконувалось за ключовими технічними та функціональними критеріями, що мають безпосереднє значення для реалізації системи онлайн-

замовлення їжі. Результати проведеного аналізу наведені у табл. 1.2

Таблиця 1.2 – Порівняльний аналіз можливих стеків технологій для реалізації сервісу замовлення їжі у ресторанах міста

Критерій	Next.js + NestJS	React SPA + Express.js	Vue.js + Laravel	Angular + Spring Boot
Тип фронтенду	SSR/SSG/SPA	SPA	SPA	SPA
Серверна архітектура	Модульна, OOP	Мінімалістична, callback-based	MVC (PHP)	MVC, RESTful, OOP
Мова програмування	TypeScript	JavaScript	JS (Vue) + PHP	TypeScript + Java
Повноцінна підтримка SSR	+	–	–	– (частково)
Підтримка WebSocket	+	– (потребує socket.io)	+ (через Laravel Echo)	+
Готовність до масштабування	Висока	Середня	Середня	Висока
Складність навчання	Середня	Низька	Середня	Висока
Документація та спільнота	Активна, зростає	Найбільша (React/JS)	Активна, менш численна	Дуже велика
Придатність до REST API	+	+	+	+
Придатність до Microservices	+	Обмежено	–	+
SEO-оптимізація	+	–	–	Частково
Підтримка типізації	+ (TypeScript)	– (JS by default)	–	+

Як видно з порівняльної таблиці, стек Next.js + NestJS вирізняється збалансованістю між гнучкістю, швидкістю розробки, продуктивністю та готовністю до масштабування [3-12]. Зокрема, Next.js надає повну підтримку SSR/SSG, що критично важливо для SEO та першого завантаження [6, 7], тоді як NestJS дозволяє будувати чітко структуровану серверну логіку, підтримує WebSocket і має високу сумісність з базами даних [4, 12, 13].

Якщо порівнювати з наведеними альтернативами, то можна зробити такі висновки:

– React + Express є зручним стартовим варіантом для невеликих проєктів, однак потребує додаткової конфігурації для SSR і WebSocket-функціоналу [14];

– Vue + Laravel добре підходить для традиційних CRUD-застосунків, але поступається в продуктивності та масштабованості [15, 16];

– Angular + Spring Boot – потужний, але складний стек, який потребує значних ресурсів для розгортання, а також більше часу на вивчення та підтримку [17-19].

Після порівняльного аналізу сучасних технологічних стеків було ухвалено рішення реалізовувати фронтенд-частину системи з використанням Next.js , а серверну логіку – на базі NestJS. Така архітектурна парадигма дозволяє поєднати переваги реактивного користувацького інтерфейсу з масштабованістю та модульністю серверної частини.

Next.js є розширенням бібліотеки React, орієнтованим на розробку повноцінних веб-застосунків із серверним рендерингом (SSR), генерацією статичних сторінок (SSG) та гібридними підходами. Його використання в межах клієнтської частини зумовлене такими характеристиками:

1. Підтримка SSR і SSG дозволяє значно зменшити годину початкового завантаження інтерфейсу, що є критичним для мобільних користувачів і забезпечує кращі показники SEO.

2. Інтегрована маршрутизація на основі файлової структури спрощує організацію сторінок застосування та дозволяє уникнути надмірної конфігурації.

3. API Routes – можливість реалізовувати прості кінцеві точки бекенд (ендпоінти) безпосередньо в межах Next.js, що зручно для обробки локальних запитів (наприклад, попередня валідація) [11].

4. Підтримка TypeScript надає високий рівень типізації, що знижує ризики помилок у великих проектах.

5. Готова оптимізація зображень, шрифтів та кешування сприяє створенню швидких, продуктивних інтерфейсів [9, 10].

З огляду на велику кількість інтерактивних елементів у сервісі замовлення їжі (додавання до кошика, оновлення статусу доставки в реальному часі, push-сповіщення), Next.js є доцільним вибором завдяки підтримці reactive UI та SSR-

комбінованої архітектури.

NestJS є прогресивним фреймворком для створення серверної логіки на базі Node.js. Він поєднує принципи об’єктно-орієнтованого програмування, функціонального підходу та модульної архітектури, що робить його зручним інструментом для побудови масштабованих REST- та GraphQL-інтерфейсів.

До ключових переваг NestJS у контексті вирішення поставленого завдання можна віднести [4]:

1. Чітку модульну структуру, яка дозволяє ізолювати бізнес-логіку, контролери, сервіси та DTO, що позитивно впливає на підтримуваність та тестованість проекту.

2. Вбудовану підтримку TypeScript, що сприяє ранньому виявленню помилок та забезпечує автодоповнення у середовищах розробки.

3. Інтеграцію з популярними ORM (TypeORM, Prisma) [20], що дозволяє гнучко взаємодіяти з реляційними базами даних, що особливо актуально для зберігання інформації про замовлення, меню, користувачів тощо.

4. Підтримка WebSockets дозволяє реалізувати функціонал реального часу, зокрема оновлення статусу доставки, зв’язок між кур'єром та клієнтом.

5. Високий рівень безпеки завдяки механізмам авторизації, валідації, обробці помилок та використанню middleware.

Таким чином, використання NestJS на серверній стороні дозволяє сформувати надійну основу для обробки бізнес-логіки, управління базою даних, реалізації аутентифікації та взаємодії з клієнтською частиною через REST API.

Поєднання Next.js і NestJS дозволяє реалізувати архітектуру frontend–backend separation в якій компоненти взаємодіють через API, а також:

– розмежувати зони відповідальності (інтерфейс та логіка);

– забезпечити кращу масштабованість системи;

– розгортати фронтенд і бекенд незалежно один від одного;

– спростити автоматизоване тестування та CI/CD-процеси.

Завдяки високій сумісності обох фреймворків з TypeScript, забезпечується цілісність типізації на рівні всього проєкту, що знижує ймовірність помилок під

час розробки та підвищує ефективність командної роботи.

Таким чином, поєднання Next.js та NestJS є оптимальним вибором для реалізації сервісу онлайн-замовлення їжі, який передбачає:

– взаємодію в реальному часі (статуси доставки, повідомлення);

– швидке завантаження інтерфейсу;

– можливість масштабування (багато користувачів, ресторанів, кур’єрів);

– зручну підтримку REST-інтерфейсів.

Висновки до розділу:

У рамках першого розділу було виконано комплексний огляд сучасних сервісів онлайн-замовлення їжі, зокрема глобальних платформ Glovo , Uber Eats та Bolt Food , а також вітчизняного рішення MixFood. Для кожного сервісу проаналізовано відповідні функціональні можливості, притаманні переваги та виявлені недоліки, що дало змогу окреслити поточні галузеві стандарти та визначити типові функціональні прогалини.

На підставі проведеного порівняльного аналізу сформульовано вичерпний перелік функціональних вимог до клієнтської частини розроблюваного сервісу. Серед ключових вимог виокремлено: багаторівневу автентифікацію, каталог ресторанів із гнучкою фільтрацією, інтерактивне меню, кошик з можливістю редагування, багатоваріантне оформлення та оплату замовлень, трекінг доставки в реальному часі, push-сповіщення, систему відгуків, програму лояльності та забезпечення доступності інтерфейсу.

Додатково розглянуто ряд альтернативних технологічних стеків (React SPA+Express.js, Vue+Laravel, Angular+Spring Boot) та проведено їх порівняння за критеріями продуктивності, масштабованості, підтримки SSR, WebSocket-функціоналу, складності навчання та готовності до мікросервісної архітектури. На основі отриманих результатів обґрунтовано доцільність використання стеку Next.js (frontend) + NestJS (backend) , який забезпечує:

– повноцінний серверний та статичний рендеринг для швидкого завантаження та SEO-оптимізації;

– типізований, модульний та тестований серверний код з зручною підтримкою WebSocket;

– високу сумісність компонентів завдяки єдиній мовній базі (TypeScript) та готовність до горизонтального масштабування.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СЕРВІСУ ЗАМОВЛЕННЯ ЇЖІ У РЕСТОРАНАХ МІСТА

2.1 Проєктування архітектури сервісу замовлення їжі

Розроблювана інформаційна система онлайн-замовлення їжі базується на сучасній багаторівневій архітектурі, що забезпечує чітке розмежування відповідальностей між клієнтською, серверною та інфраструктурною частинами. Такий підхід гарантує масштабованість, розширюваність та високу підтримуваність рішення.

Архітектурна модель системи реалізована відповідно до трирівневої концепції (рис. 2.1):

1. Презентаційний рівень представлений фреймворком Next.js, що дозволяє реалізовувати гібридні застосунки з використанням серверного рендерингу (SSR), генерації статичних сторінок (SSG) та класичного підходу SPA. Користувацький інтерфейс побудований на базі React-компонентів, а також включає Service Worker для підтримки офлайн-режиму та push-сповіщень. Крім того, Next.js використовується для реалізації початкових маршрутів авторизації та взаємодії з провайдерами OAuth.

2. Логічний рівень реалізовано за допомогою фреймворка NestJS, який забезпечує чітку модульну структуру бекенд-частини. Основні програмні модулі включають: AuthModule (автентифікація, авторизація, OTP, OAuth), UsersModule (керування користувачами), RestaurantsModule та MenuModule (каталог), OrdersModule (обробка замовлень та їх статусів), PaymentsModule (інтеграція з платіжними шлюзами), NotificationsModule (обробка push-повідомлень та WebSocket). Обробка подій у реальному часі (наприклад, зміна статусу замовлення або чат із кур’єром) реалізується через WebSocket, а проміжне кешування та асинхронні черги — через Redis.

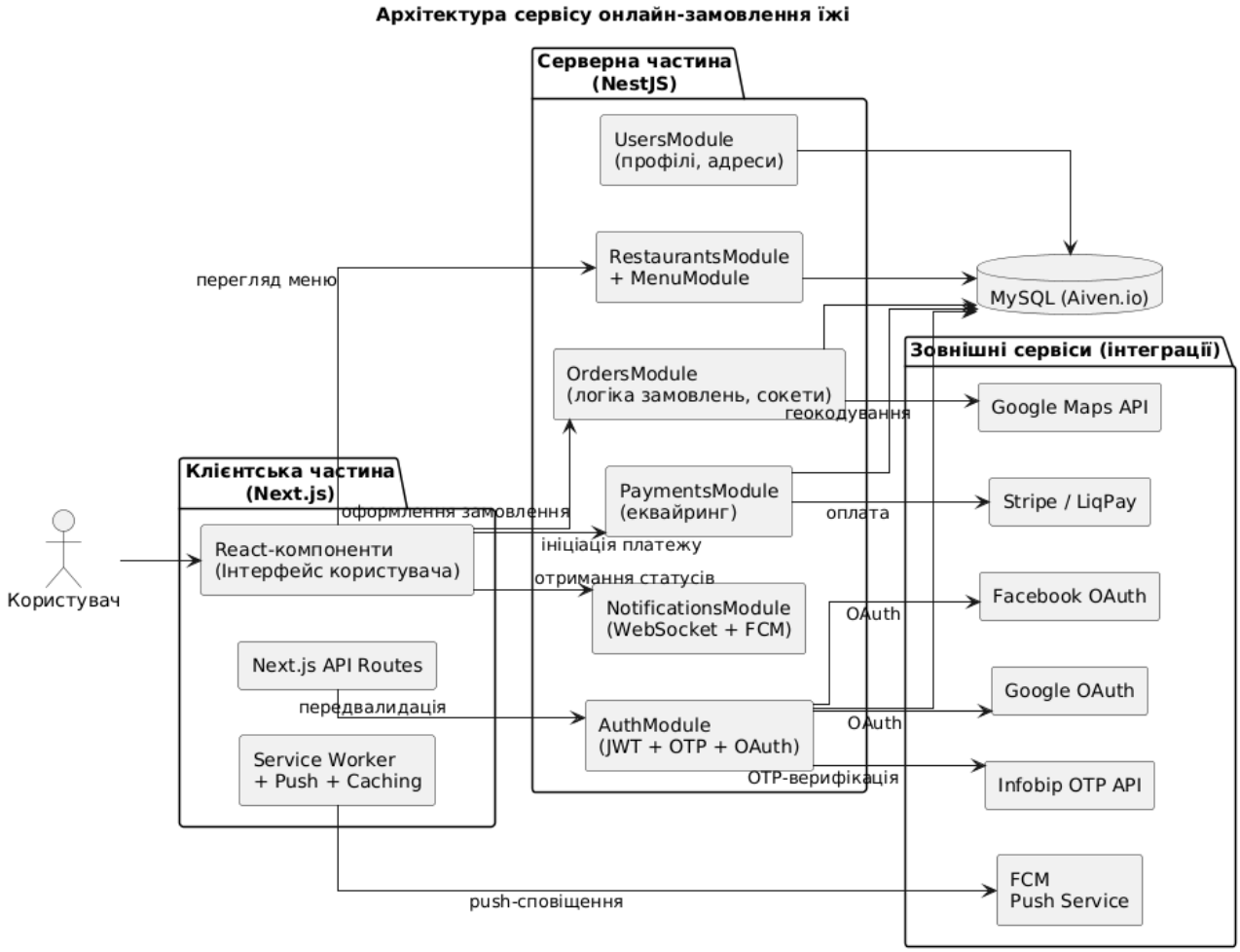

Рисунок 2.1 – Узагальнена архітектура сервісу замовлення їжі

3. Рівень даних побудовано на основі керованого хмарного сервісу Aiven.io, який надає кластер бази даних MySQL з підтримкою автоматичного резервного копіювання, масштабування та високої доступності. Основними сутностями бази даних є: користувачі, ресторани, меню, замовлення, платіжні транзакції та відгуки. Для кешування популярних запитів, зберігання тимчасових сесій та реалізації черг повідомлень використовується Redis.

У межах системи реалізовано мультифакторну автентифікацію. Зокрема, для підвищення безпеки процесу реєстрації та підтвердження входу використовується сервіс Infobip, що забезпечує надсилання одноразових SMS-кодів (OTP) за допомогою верифікаційного API. На додаток до класичної реєстрації за електронною поштою, користувачеві доступна авторизація через зовнішніх провайдерів: Google OAuth 2.0 та Facebook OAuth 2.0. Реалізація цих

сценаріїв у NestJS виконується з використанням відповідних стратегій бібліотеки Passport.js.

Крім автентифікації та авторизації, система інтегрується з рядом зовнішніх сервісів. Зокрема:

– Stripe – для здійснення онлайн-оплати замовлень;

– Firebase Cloud Messaging (FCM) – для реалізації push-сповіщень про зміну статусу замовлення або нові повідомлення від кур’єра.

Зовнішні сервіси логічно згруповані в окремий підрівень інтеграції, який взаємодіє з відповідними модулями NestJS. Таке групування дозволяє ізолювати залежності від сторонніх API, полегшуючи тестування та обслуговування системи.

Загалом, обрана архітектура демонструє високу гнучкість, розширюваність і здатність до масштабування. Поєднання Next.js на стороні фронтенду та NestJS на стороні бекенду забезпечує уніфіковану мовну базу (TypeScript) та спрощує командну розробку. Використання керованої інфраструктури (MySQL на Aiven.io, Infobip, Firebase) знижує операційні ризики, а підтримка мультифакторної автентифікації та реального часу підвищує як рівень безпеки, так і якість користувацького досвіду.

2.2 Проєктування бази даних сервісу замовлення їжі

Структура бази даних сервісу онлайн-замовлення їжі реалізована за допомогою інструменту Prisma ORM [20-24], що забезпечує типізований доступ до даних та зручну інтеграцію з NestJS. Основу логічної моделі становлять сутності, що відображають ключові об’єкти предметної області: користувачів, ресторани, продукти, замовлення, категорії та промоакції. Усі таблиці зберігаються у реляційній базі даних MySQL, розгорнутій на платформі Aiven.io.

Розроблена структура БД з відображенням зв’язків між таблицями наведена на рис. 2.2. Для побудови використано сервіс Prisma ERD [25].

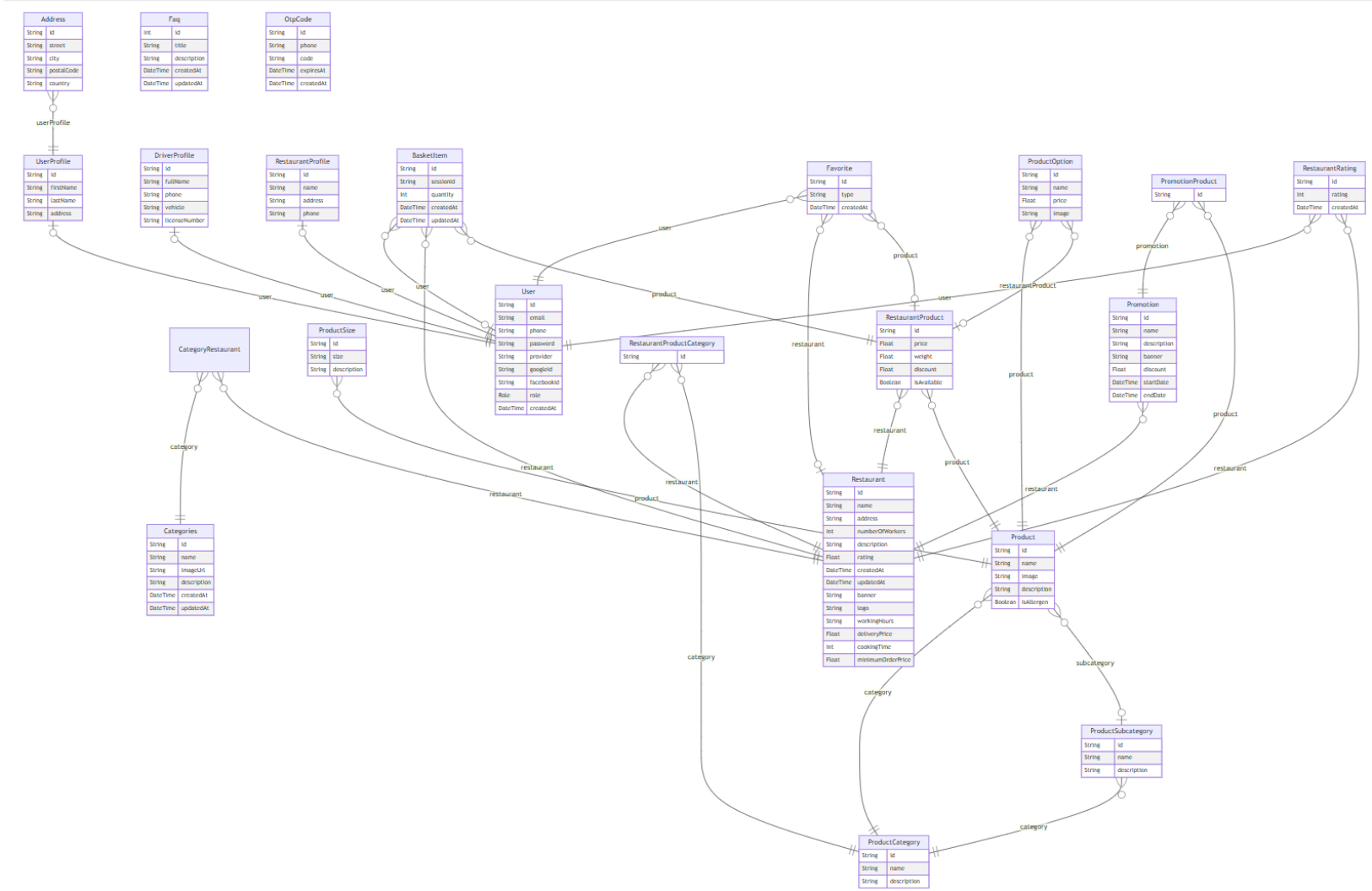

Рисунок 2.2 – Діаграма компонентів для веб-сервісу візуалізації даних екологічного моніторингу

Структура БД складається з наступних основних складових:

1. Користувачі та профілі. Базова сутність User описує обліковий запис користувача системи. Вона містить поля електронної пошти, номера телефону, пароля, а також ідентифікатори для соціальних входів (Google/Facebook). Роль користувача визначається через перелічення Role, що дозволяє розмежовувати звичайних користувачів, кур'єрів, представників ресторанів та адміністраторів.

Кожен користувач може мати один із трьох типів профілю:

– UserProfile – для клієнтів, містить персональні дані та пов'язані адреси (Address);

– DriverProfile – для кур'єрів, з полями про транспортний засіб і ліцензію;

– RestaurantProfil – для представників ресторанів, що містить назву, адресу та контактні дані закладу.

2. Продукти та меню. Модель Product представляє базову продуктову одиницю (наприклад, піца), що належить до певної категорії (ProductCategory) та може мати підкатегорію (ProductSubcategory). Продукт підтримує множинні розміри (ProductSize) та додаткові опції (ProductOption), що забезпечує гнучке формування страв.

Модель RestaurantProduct є проміжною і визначає, які саме продукти доступні в конкретному ресторані, з зазначенням ціни, знижки та доступності.

Додатково реалізовано таблицю RestaurantProductCategory, що дозволяє ресторанам створювати власну ієрархію категорій на основі глобальних категорій.

3. Ресторани та категорії

Сутність Restaurant містить загальні дані про заклад: назву, опис, контактну інформацію, час приготування, рейтинг, логотип, банер тощо. Один ресторан може бути пов'язаний із багатьма категоріями через зв'язок CategoryRestaurant, що реалізує зв'язок N:M.

Сутність Categories є глобальним переліком типів кухонь або страв, кожен з яких може бути прив'язаний до кількох ресторанів.

4. Замовлення, кошик та обране

Модель BasketItem описує позиції в кошику користувача. Вона дозволяє прив’язувати об’єкти як до авторизованого користувача (userId), так і до анонімної сесії (sessionId), забезпечуючи гнучкість у користуванні.

Сутність Favorite дозволяє зберігати улюблені продукти або ресторани. Вона підтримує унікальність на рівні користувача й об’єкта, що запобігає дублюванню вподобань.

Для оцінювання закладів передбачено модель RestaurantRating, де зберігаються числові рейтинги, прив’язані до конкретного користувача.

5. Акції та промоції

Сутність Promotion описує знижки або маркетингові кампанії, які прив’язані до конкретного ресторану. Окрема таблиця PromotionProduct реалізує N:M зв’язок між акцією та продуктами, які беруть у ній участь.

6. Додаткові сутності:

– OtpCode – тимчасова таблиця, що зберігає одноразові коди підтвердження для 2FA через SMS, інтегровані з Infobip;

– Faq – таблиця частих запитань та відповідей, що може використовуватись у довідковому розділі сайту.

На рис. 2.3 наведено зразок опису моделі сутності «Ресторан» у схемі ORM-бібліотеки Prisma. Таблиця містить як обов’язкові поля (id, name, address), так і необов’язкові (такі як description, rating, banner, logo). Як видно з моделі, за допомогою атрибуту @@map(“restaurants”) назва моделі прив’язується до таблиці restaurants. Для поля id з використанням атрибуту @id задано, що поле буде використовуватися як первинний ключ таблиці. Атрибут @default визначає значення за замовчанням, яке для даного поля генерується з використанням функції атрибуту uuid(). Також за допомогою атрибуту @default визначено автоматичне заповнення часу створення запису (поле createdAt) з використанням функції атрибуту now(). Для поля updatedAt атрибут @updatedAt автоматично заповнює час при змінах [22].

prisma > schema > restaurants > restaurants.prisma

```
model Restaurant {
  id                String   @id @default(uuid())
  name              String
  address           String
  numberOfWorkers   Int
  description       String?
  rating            Float?
  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt
  banner            String?
  logo              String?
  workingHours      String?
  deliveryPrice     Float?
  cookingTime       Int?
  minimumOrderPrice Float?

  favorites Favorite[]
  ratings   RestaurantRating[]

  promotions                 Promotion[]
  categoryRestaurants        CategoryRestaurant[]
  restaurantProducts         RestaurantProduct[]
  restaurantProductCategories RestaurantProductCategory[]
  basketItems                BasketItem[]

  @@map("restaurants")
}
```

Рисунок 2.3 – Опис моделі сутності «Ресторан» у схемі Prisma

У табл. 2.1 наведено перелік основних сутностей бази даних, їх основні поля, атрибути, типи даних та характер зв’язків, реалізованих за допомогою Prisma ORM. Дана таблиця не містить повного опису Це дозволяє краще уявити логічну модель системи.

Таблиця 2.1 – Основні таблиці, їх атрибути, типи даних та характер зв’язків у базі даних сервісу замовлення їжі

Назва атрибута	**Тип даних**	**Опис / зв’язок**
Таблиця User		
id	UUID	Первинний ключ
email	String	Унікальний (nullable)
phone	String	Унікальний (nullable)
role	Enum (Role)	USER / DRIVER / RESTAURANT / ADMIN
Відношення		1:1 з UserProfile, DriverProfile, RestaurantProfile; 1:N з Rating, Favorite
Таблиця UserProfile		
id	UUID	Первинний ключ
firstName	String	Ім’я користувача
address	String	Основна адреса
userId	UUID	Зовнішній ключ на User
Відношення		N:1 з User, 1:N з Address
Таблиця Restaurant		
id	UUID	Первинний ключ
name	String	Назва ресторану
address	String	Адреса
rating	Float	Середній рейтинг
Відношення		1:N з Product, Rating, Promotion, CategoryRestaurant
Таблиця Product		
id	UUID	Первинний ключ
name	String	Назва продукту
categoryId	UUID	Зовнішній ключ на ProductCategory
Відношення		1:N з ProductSize, ProductOption, RestaurantProduct
Таблиця RestaurantProduct		
id	UUID	Первинний ключ
productId	UUID	Зовнішній ключ на Product
restaurantId	UUID	Зовнішній ключ на Restaurant
price	Float	Ціна у конкретному закладі
Відношення		N:1 з Restaurant та Product
Таблиця OtpCode		
id	UUID	Первинний ключ
phone	String	Унікальний номер телефону
code	String	OTP-код
expiresAt	DateTime	Час закінчення дії

Керування життєвим циклом БД сервісу замовлення їжі здійснювалось з використанням механізму міграцій [21]. Для початкового заповнення ресурсу даними були використані сідери (seeders) Prisma. На рис. 2.4 наведено приклад коду сідера для початкового заповнення категорій страв.

prisma > seed > TS product-category.seed.ts > ...

```
1  import { Prisma, PrismaClient } from '@prisma/client';
2
3  const prisma = new PrismaClient();
4
5  async function seedProductCategory() {
6      const productCategories: Prisma.ProductCategoryCreateInput[] = [
7          {name: 'Піца'},
8          {name: 'Бургери'},
9          {name: 'Суші'},
10         {name: 'Напої'},
11         {name: 'Десерти'},
12         {name: 'Фрі'},
13         {name: 'Салати'},
14         {name: 'Снеки'},
15         {name: 'Курячі крильця'},
16         {name: 'Нагетси'},
17         {name: 'Роли'}
18     ];
19
20     for (const productCategory of productCategories) {
21         await prisma.productCategory.create({ data: productCategory });
22     }
23
24     console.log('✅ Категорії продуктів успішно додано!');
25 }
26
27 seedProductCategory()
28     .catch((e) => {
29         console.error(e);
30         process.exit(1);
31     })
32     .finally(async () => {
33         await prisma.$disconnect();
34     });
```

Рисунок 2.4 – Лістинг функції початкового заповнення БД інформацією про категорії страв

Отже, запропонована структура бази даних є повноцінним відображенням логіки предметної області. Вона враховує потреби різних типів користувачів (клієнтів, кур'єрів, закладів), підтримує персоналізацію меню, модульність структури продуктів, обробку замовлень у реальному часі, реалізацію промоакцій і програм лояльності. Завдяки чіткому визначенню зв'язків і типів сутностей, система є масштабованою, зручною для подальшого розширення та забезпечує стабільну основу для реалізації бізнес-логіки сервісу онлайн-замовлення їжі.

2.3 Проєктування каркасу серверної частини веб-сервісу замовлення їжі

Серверна частина розробленої інформаційної системи реалізована з використанням фреймворку NestJS, який базується на концепціях модульності, інверсії керування та строгої типізації за допомогою мови TypeScript. Такий підхід забезпечує високу масштабованість, підтримуваність та зручність у супроводі проєкту.

Основна логіка застосунку розміщена в директорії src/, яка містить низку модулів, згрупованих за функціональною ознакою (рис. 2.5). Нижче наведено опис основних структурних одиниць, що входять до складу серверної частини:

– auth – модуль аутентифікації користувачів. Містить класи DTO (Data Transfer Object), що описують структуру даних, необхідних для виконання операцій входу, реєстрації та верифікації користувачів. Зокрема, враховано інтеграцію з зовнішніми провайдерами автентифікації (Google, Facebook) та OTP через Infobip.

– basket – модуль, відповідальний за обробку кошика користувача. Реалізовано сервіс для взаємодії з даними кошика, контролер для обробки відповідних HTTP-запитів, а також класи DTO для оновлення вмісту кошика. Крім того, включено інтерфейс для роботи з анонімними сесіями, що дозволяє обробляти дії неавторизованих користувачів.

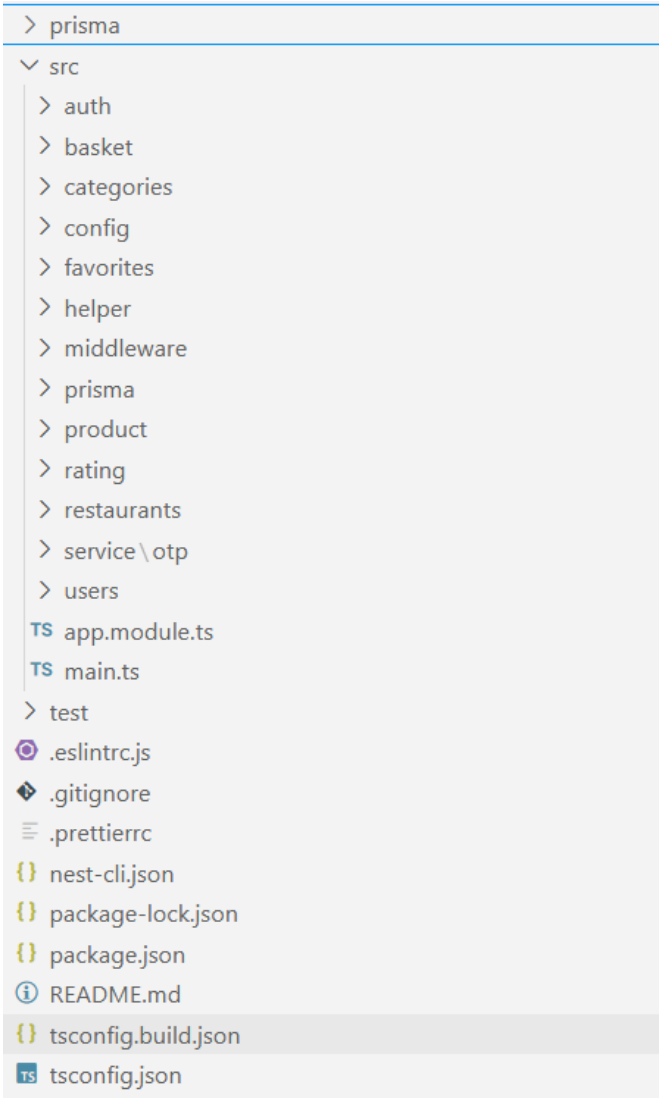

Рисунок 2.5 – Загальна структура серверної частини сервісу замовлення їжі

– categories – модуль для керування категоріями страв та ресторанів. Містить сервіс, який реалізує бізнес-логіку, та контролер для обробки запитів клієнтської частини.

– config – конфігураційний модуль, що зберігає параметри для взаємодії із зовнішніми сервісами, зокрема Infobip. Забезпечує централізовану конфігурацію та гнучке налаштування API-ключів і базових URL.

– favorites – модуль, призначений для управління списками улюблених ресторанів і страв користувача. Структура модуля містить сервіс, контролер та відповідні класи DTO, які забезпечують передачу необхідних параметрів для додавання або вилучення об’єктів з обраного.

– helper – утилітний модуль, у якому розміщено клас ResponseHelper, що стандартизує структуру відповідей HTTP для всіх REST-контролерів застосунку. Це дозволяє досягти узгодженості форматів відповіді у випадках успішного або помилкового завершення запитів.

– middleware – модуль, який реалізує проміжне програмне забезпечення (JWTMiddleware) для верифікації та декодування JWT-токенів у вхідних запитах. Застосовується для захисту приватних маршрутів API та авторизації користувачів.

– product – модуль для керування продуктами (стравами). Містить основну бізнес-логіку, відповідні контролери та DTO-класи. Забезпечує CRUD-операції над сутністю продукту, враховуючи такі параметри як розміри, опції та категорії.

– rating – модуль для реалізації механізмів оцінювання користувачами як окремих продуктів, так і ресторанів. Містить сервіс для збереження, обробки та агрегування оцінок, а також DTO-класи для передачі вхідних параметрів оцінювання.

– restaurants – модуль, який інкапсулює логіку керування ресторанами. Реалізовано методи для реєстрації ресторану, оновлення профілю, отримання списків, фільтрації за категоріями, рейтингу тощо.

– service – модуль сервісної інтеграції із зовнішнім постачальником OTP-кодів (Infobip). Містить класи для генерації, зберігання та верифікації кодів одноразового доступу при реєстрації або вході користувача.

– users – модуль керування даними користувачів. Реалізовано сервіси для створення профілю користувача, оновлення особистих даних, отримання списку користува–

main.ts – точка входу у застосунок, у якій виконується ініціалізація NestJS-додатку, підключення глобальних модулів, реєстрація middleware та запуск HTTP-сервера.

– app.module.ts – кореневий модуль застосунку, який об’єднує всі функціональні модулі, визначає загальні провайдери та реєструє глобальні

конфігурації. Перший проєкт – це бібліотека класів AsekmDomainClassLibrary, яка визначає доменну модель системи. Він містить такі основні компоненти, Department (структурні підрозділи підприємства), Unit (технологічні агрегати, що належать певному підрозділу), Parameter (екологічні показники, що контролюються на агрегатах), Measurement (конкретні значення параметрів).

У архітектурі NestJS усі функціональні одиниці організовано у вигляді модулів, кожен з яких об’єднує відповідні сервіси, контролери, DTO та, за потреби, інтерсептори, middleware або провайдери. Типовий модуль реалізується за принципами інверсії керування та залежностей, що забезпечує високу модульність, масштабованість та повторне використання коду.

Розглянемо приклад модуля роботи зі стравами (ProductModule), який містить такі основні компоненти:

1. Контролер ProductController відповідає за обробку HTTP-запитів до ресурсу products. У контролері визначено маршрути типу GET, що дозволяють:

– отримати повну інформацію про страву за її ідентифікатором (/products/get-product/:id);

– отримати список категорій ресторанів, у яких представлено цю страву (/products/get-product-restaurants-category/:id);

– здійснити фільтрацію страв за категорією та підкатегорією у межах ресторану (/products/get-product-restaurants/:id).

2. Сервіс ProductService реалізує бізнес-логіку обробки запитів. До його функцій входить:

– отримання інформації про страву з бази даних;

– обробка фільтрів за категоріями та підкатегоріями;

– взаємодія з іншими модулями, наприклад, модулями категорій або ресторанів, за допомогою залежностей.

3. DTO-класи, зокрема GetProductRestaurantDto, використовуються для типізованого прийому параметрів запитів з клієнтської частини. Вони дозволяють забезпечити перевірку структури та коректності вхідних даних, що є важливою умовою безпечного та стабільного функціонування API.

4. Інтеграція з іншими модулями: модуль продуктів взаємодіє з модулями ресторанів, категорій та кошика, реалізуючи повноцінну логіку подання даних на клієнтський інтерфейс.

Загалом, така організація модуля забезпечує чітке розділення відповідальностей:

– контролер приймає HTTP-запит, викликає метод сервісу та формує відповідь.

– сервіс оперує логікою обробки даних та взаємодіє з базою даних через ORM (наприклад, Prisma).

– DTO гарантує безпеку типів та валідацію параметрів запитів.

На рис. 2.6 наведено лістинг класу GetProductRestaurantDto.

src > product > dto > TS get-product-restaurant.dto.ts > GetProductRestaurantDto > categoryId

```
1  import {IsNumber, IsOptional, IsString} from "class-validator";
2  import {Type} from "class-transformer";
3
4  export class GetProductRestaurantDto {
5      @IsOptional()
6      @Type(() => Number)
7      @IsNumber()
8      page?: number;
9
10     @IsOptional()
11     @Type(() => Number)
12     @IsNumber()
13     limit?: number;
14
15     @IsOptional()
16     @Type(() => String)
17     @IsString()
18     categoryId?: string
19
20     @IsOptional()
21     @Type(() => String)
22     @IsString()
23     subcategoryId?: string
24
25     @IsOptional()
26     @Type(() => String)
27     @IsString()
28     size?: string
29
30     @IsOptional()
31     @Type(() => String)
32     @IsString()
33     sessionId?: string
34 }
```

Рисунок 2.6 – Лістинг класу GetProductRestaurantDto

На рис. 2.7 наведено фрагмент лістингу сервісу ProductService, зокрема конструктор, що з приймає з використанням механізму впровадження залежностей сервіс PrismaService для спрощення роботи з ORM та ResponseHelper для спрощення формування HTTP-відповідей.

```
import {Request} from "express";
import {GetProductRestaurantDto} from "./dto/get-product-restaurant.dto";

@Injectable()
export class ProductService {
    constructor(
        private readonly prisma: PrismaService,
        private readonly responseHelper: ResponseHelper
    ) {}

    async findProductRestaurantCategoriesWithProducts(restaurantId: string) {
        const categories = await this.prisma.restaurantProductCategory.findMany({
            where: { restaurantId },
            include: {
                category: {
                    include: {
                        products: {
                            where: {
                                restaurantProducts: {
                                    some: {
                                        restaurantId,
                                    },
                                },
                            },
                            include: {
                                sizes: true,
                                subcategory: true,
                            },
                        },
                    },
                },
            },
        });

        const formattedCategories = categories.map(({ category }) => ({
            id: category.id,
            name: category.name,
            description: category.description,
            products: category.products.map(product => ({
                ...product,
                image: product.image ? `${process.env.FILE_BASE_URL}/${product.image}` : null,
            })),
        }));
```

Рисунок 2.7 – Фрагмент лістинг коду сервісу ProductService для роботи зі стравами

На рис. 2.8 наведено лістинг контролера страв. Як видно, до класу контролера застосовано декоратор @Controller, у який передається префікс

маршруту “products” для звернення до відповідних кінцевих точок для отримання інформації про страви. Також до конструктора контролера з використанням механізму впровадження залежностей передається ProductService, який реалізує логіку формування запитів до серверу через механізм API.

```
6   @Controller('products')
7   export class ProductController {
8       constructor(private readonly productService: ProductService) {}
9
10      @Get('get-product/:id')
11      async findOne(@Req() req: Request,
12      @Param('id') id: string,
13      @Query() sessionId: string) {
14          return await this.productService
15          .findProductById(req, id, sessionId);
16      }
17
18      @Get('get-product-restaurants-category/:id')
19      findProductRestaurantCategory(@Param('id') id: string) {
20          return this.productService
21          .findProductRestaurantCategory(id);
22      }
23
24      @Get('get-product-restaurants/:id')
25      findProductByCategoryAndSubcategory(@Req() req: Request,
26      @Param('id') id: string,
27      @Query() getProductRestaurantDto: GetProductRestaurantDto) {
28          return this.productService
29          .findProductByCategoryAndSubcategory(
30              req,
31              id,
32              getProductRestaurantDto);
33      }
34  }
```

Рисунок 2.8 – Лістинг контролера страв ProductController

Перед кожним методом контролера, що використовується для обробки запитів до визначених кінцевих точок застосовано декоратор, що відповідає типу метода HTTP (наприклад, @Get(‘get-product/:id’)).

До параметрів методів дій застосовані наступні декоратори:

– @Req() – для отримання інформації з описом HTTP-запиту, що надійшов;

– @Param() – для автоматичного отримання параметрів запиту;

– @Query() – для отримання інформації про параметри рядку запиту, зокрема прив’язки до класів DTO та ідентифікатора сесії.

На рис. 2.9 наведено діаграму послідовності UML для запиту GET /products/get-product/:id, яка демонструє взаємодію клієнта з модулем Product (контролер → сервіс → база даних) у NestJS-застосунку.

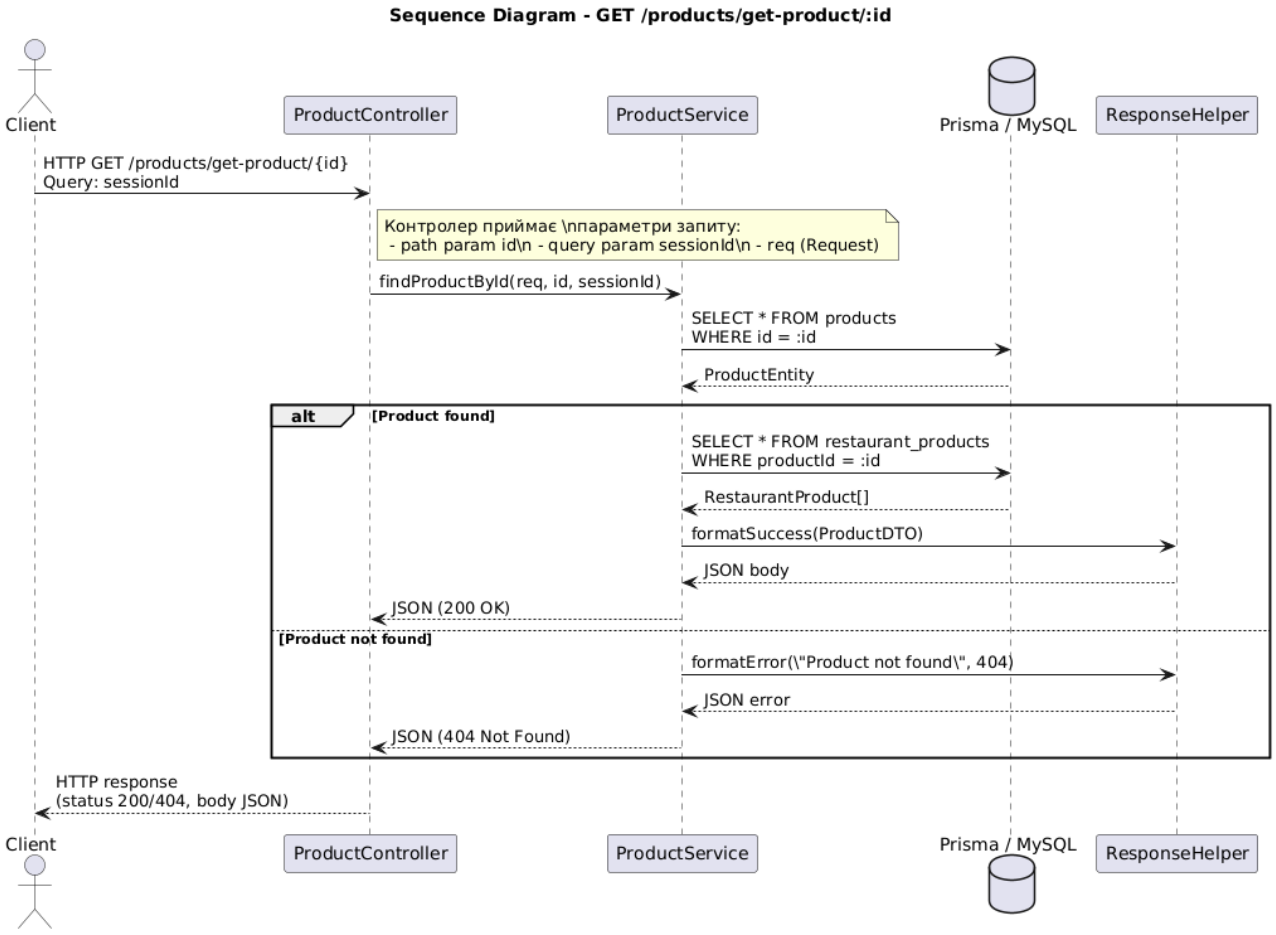

Рисунок 2.9 – Діаграма послідовності UML методу контролера ProductController для отримання страви по id

Дана діаграма описує послідовність обробки HTTP-запиту методом дії контролера:

1. Клієнт ініціює HTTP-запит до ProductController.

2. ProductController передає керування сервісу ProductService, інкапсулюючи всі параметри (req, id, sessionId).

3. ProductService звертається через Prisma до бази MySQL, отримує сутність (страву) і додатково витягує дані про наявність страви у ресторанах.

4. Залежно від результату служба формує відповідь за допомогою ResponseHelper і повертає її контролеру.

5. ProductController відправляє сформовану JSON-відповідь клієнту з відповідним HTTP-статусом.

Цей сценарій відображає типову схему взаємодії NestJS-модуля «Продукти», демонструючи чітке розділення обов’язків між рівнями контролера, сервісу й доступу до даних.

2.4 Реалізація логіки аутентифікації та авторизації у серверній частині сервісу замовлення їжі

Функції аутентифікації та авторизації на стороні сервера виділені у окремий модуль, який має назву AuthModule. Модуль AuthModule виконує функцію централізованого управління всіма процесами, пов’язаними з аутентифікацією користувачів у системі. Його структура відповідає архітектурному стилю модульного розподілу, який характерний для фреймворку NestJS і сприяє високої розширюваності та ізольованості логіки аутентифікації від інших частин застосування. Лістинг коду модуля наведено на рис. 2.10.

Основними компонентами модуля є:

1. Сервіс AuthService реалізує бізнес-логіку, пов’язану з аутентифікацією, OTP-верифікацією, входом через соціальні мережі та сторонні сервіси (Google, Facebook), генерацією JWT-токенів та обробкою refresh-токенів. Він інкапсулює взаємодію з базою даних через PrismaService, надсилання SMS-кодів через OtpService, а також використання допоміжного класу ResponseHelper для формування стандартизованих відповідей.

src > auth > TS auth.module.ts > AuthModule

```
1   import { Module } from '@nestjs/common';
2   import { AuthService } from './auth.service';
3   import { AuthController } from './auth.controller';
4   import { PrismaService } from '../prisma/prisma.service';
5   import { PassportModule } from '@nestjs/passport';
6   import { JwtModule } from '@nestjs/jwt';
7   import { ResponseHelper } from 'src/helper/response.helper';
8   import { OtpService } from "../service/otp/otp.service";
9
10  @Module({
11    imports: [
12      PassportModule,
13      JwtModule.register({
14        secret: process.env.JWT_SECRET,
15        signOptions: { expiresIn: '1h' },
16      }),
17    ],
18    controllers: [AuthController],
19    providers: [AuthService, PrismaService, ResponseHelper, OtpService],
20    exports: [AuthService],
21  })
22  export class AuthModule {}
23
```

Рисунок 2.10 – Лістинг модуля AuthModule, відповідального за аутентифікацію та авторизацію

2. Контролер AuthController відповідає за обробку вхідних HTTP-запитів, пов’язаних із реєстрацією, входом, соціальною аутентифікацією та оновленням токенів. Він делегує логіку сервісу, дотримуючись принципу розділення відповідальностей (Separation of Concerns).

3. Підключення модулів PassportModule та JwtModule забезпечує вбудовану підтримку механізмів автентифікації на базі стратегій (зокрема, JWT) [28].

4. OtpService забезпечує взаємодію із зовнішнім сервісом надсилання одноразових кодів (Infobip), що дозволяє реалізувати просту та безпечну реєстрацію за номером телефону [29].

5. PrismaService виконує роль ORM для доступу до бази даних [21], що забезпечує безпечну по відношенню до типів та ефективну роботу з сутностями користувачів.

6. ResponseHelper – допоміжний клас, який стандартизує формат вихідних відповідей до клієнта, зокрема повідомлень про помилки чи успішні операції.

Модуль AuthModule також експортує сервіс AuthService, що дозволяє повторно використовувати логіку аутентифікації в інших модулях застосування.

Для візуалізації компонентів модуля AuthModule побудовано діаграму компонентів UML (рис. 2.11), яка ілюструє інкапсульовану структуру модуля аутентифікації та його залежності, відповідаючи принципам NestJS-модульності та чистої архітектури.

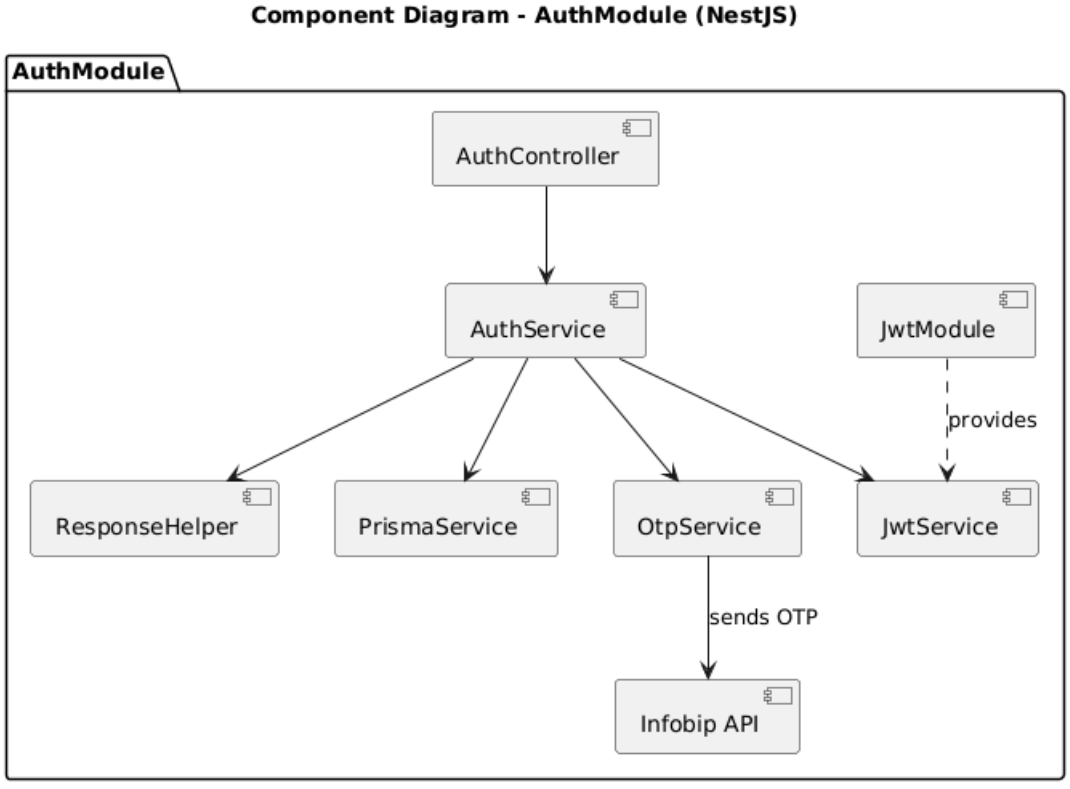

Рисунок 2.11 – Діаграма компонентів модуля AuthModule, відповідального за аутентифікацію та авторизацію

Загалом, модуль реалізований відповідно до кращих практик NestJS, таких як інверсія залежностей, модульна ізоляція та використання стратегій

аутентифікації, що забезпечує надійну та масштабовану інфраструктуру для роботи з користувачами.

Сервіс AuthService є центральним компонентом системи, відповідальним за реалізацію механізмів аутентифікації та авторизації користувачів у клієнтському застосунку (рис. 2.12). Він інтегрується з іншими сервісами, такими як PrismaService (доступ до бази даних), JwtService (генерація та верифікація JWT-токенів), OtpService (відправлення OTP-кодів через Infobip) та ResponseHelper (формування стандартизованих HTTP-відповідей).

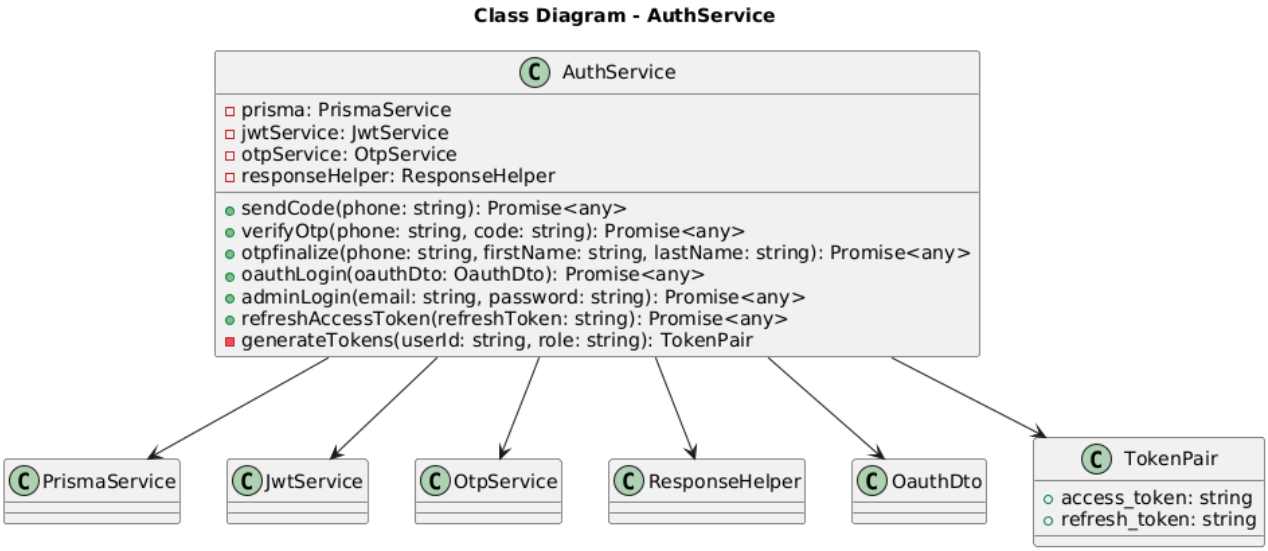

Рисунок 2.12 – Діаграма класів UML, що відображає структуру класу AuthService

Основні функціональні можливості AuthService включають [28, 30]:

1. Надсилання OTP-кодів на телефон (sendCode) – для реалізації швидкої реєстрації або входу користувача за номером телефону. Код OTP створюється випадково, зберігається у базі даних із часовим обмеженням на валідність (5 хвилин), після чого надсилається користувачеві за допомогою зовнішнього провайдера SMS (Infobip).

2. Підтвердження OTP-коду (verifyOtp) – перевірка правильності та актуальності OTP, видалення використаного коду з бази, створення нового користувача в разі першого входу, а також генерація JWT-токенів (access та refresh). У разі першої реєстрації також створюється запис профілю користувача.

3. Остаточне заповнення профілю (otpfinalize) – оновлення імені та прізвища користувача після підтвердження особи через OTP. Це дозволяє забезпечити мінімально потрібну ідентифікацію користувача після швидкої реєстрації.

4. Аутентифікація через соціальні мережі (Google та Facebook) (oauthLogin) – реалізовано підтримку OAuth-провайдерів. Якщо користувача не існує, система створює нового з відповідним унікальним ідентифікатором (googleId або facebookId) та асоційованим профілем.

5. Адміністративний вхід (adminLogin) – окремий механізм авторизації для адміністратора системи. Передбачає перевірку електронної пошти та пароля за допомогою bcrypt-хешування. Успішна аутентифікація дозволяє лише тим користувачам, чия роль у базі даних встановлена як ADMIN.

6. Обновлення токена доступу (refreshAccessToken) – оновлює access-токен на основі переданого refresh-токена. Верифікація токена відбувається через JwtService, після чого генерується новий токен з коротким рядком дії.

7. Генерація токенів (generateTokens) – приватний метод, що створює JWT access- та refresh-токені з ідентифікатором користувача (sub) та його роллю (role) Access-токен діє протягом 15 хвилин, а refresh-токен – 7 діб.

Для демонстрації логіки роботи методів сервісу наведемо код методу oauthLogin для входу з використанням OAuth2-провайдерів аутентифікації (рис. 2.13).

Метод oauthLogin реалізує функціональність входу користувача до системи з використанням облікових записів зовнішніх провайдерів аутентифікації, таких як Google та Facebook. Цей підхід відповідає сучасним вимогам щодо зручності, безпеки та швидкості входу користувачів до веб-ресурсів.

Під час виклику методу контролером AuthController, на вхід надходитиме об’єкт OauthDto, який містить такі параметри: ідентифікатор користувача у сторонньому сервісі (providerId), тип провайдера (provider) та, за наявності, електронну пошту користувача (email).

```
92      async oauthLogin(oauthDto: OauthDto) {
93          let user: { id: string;
94              phone: string | null;
95              createdAt: Date;
96              email: string | null;
97              password: string | null;
98              provider: string | null;
99              googleId: string | null;
100             facebookId: string | null;
101             role: $Enums.Role; };
102
103         if (oauthDto.provider === "google") {
104             user = await this.prisma.user.findUnique({
105                 where: { googleId: oauthDto.providerId },
106             });
107         } else if (oauthDto.provider === "facebook") {
108             user = await this.prisma.user.findUnique({
109                 where: { facebookId: oauthDto.providerId },
110             });
111         }
112         if (!user) {
113             const dataToCreate: any = {
114                 email: oauthDto.email,
115                 role: Role.USER,
116             };
117
118             if (oauthDto.provider === "google") {
119                 dataToCreate.googleId = oauthDto.providerId;
120             } else if (oauthDto.provider === "facebook") {
121                 dataToCreate.facebookId = oauthDto.providerId;
122             }
123             user = await this.prisma.user.create({
124                 data: dataToCreate,
125             });
126             await this.prisma.userProfile.create({
127                 data: {
128                     userId: user.id,
129                 },
130             });
131         }
132         return this.generateTokens(user.id, user.role);
133     }
```

Рисунок 2.13 – Лістинг методу oauthLogin сервісу AuthService

Залежно від зазначеного провайдера:

– для Google здійснюється пошук користувача в базі даних за полем googleId;

– для Facebook – за полем facebookId.

Якщо користувач не знайдений у системі, створюється новий запис у таблиці, при цьому роль користувача встановлюється як USER, а ідентифікатор провайдера зберігається відповідно до джерела.

Після цього генеруються JWT-токені доступу (access_token, термін дії – 15 хвилин) та оновлення (refresh_token, термін дії – 7 днів), які формуються за допомогою сервісу JwtService. Обидва токени додаються у відповідь, що повертається клієнту за допомогою ResponseHelper у стандартизованому форматі.

Для ілюстрації логіки роботи методу побудовано діаграму послідовності UML (рис. 2.14).

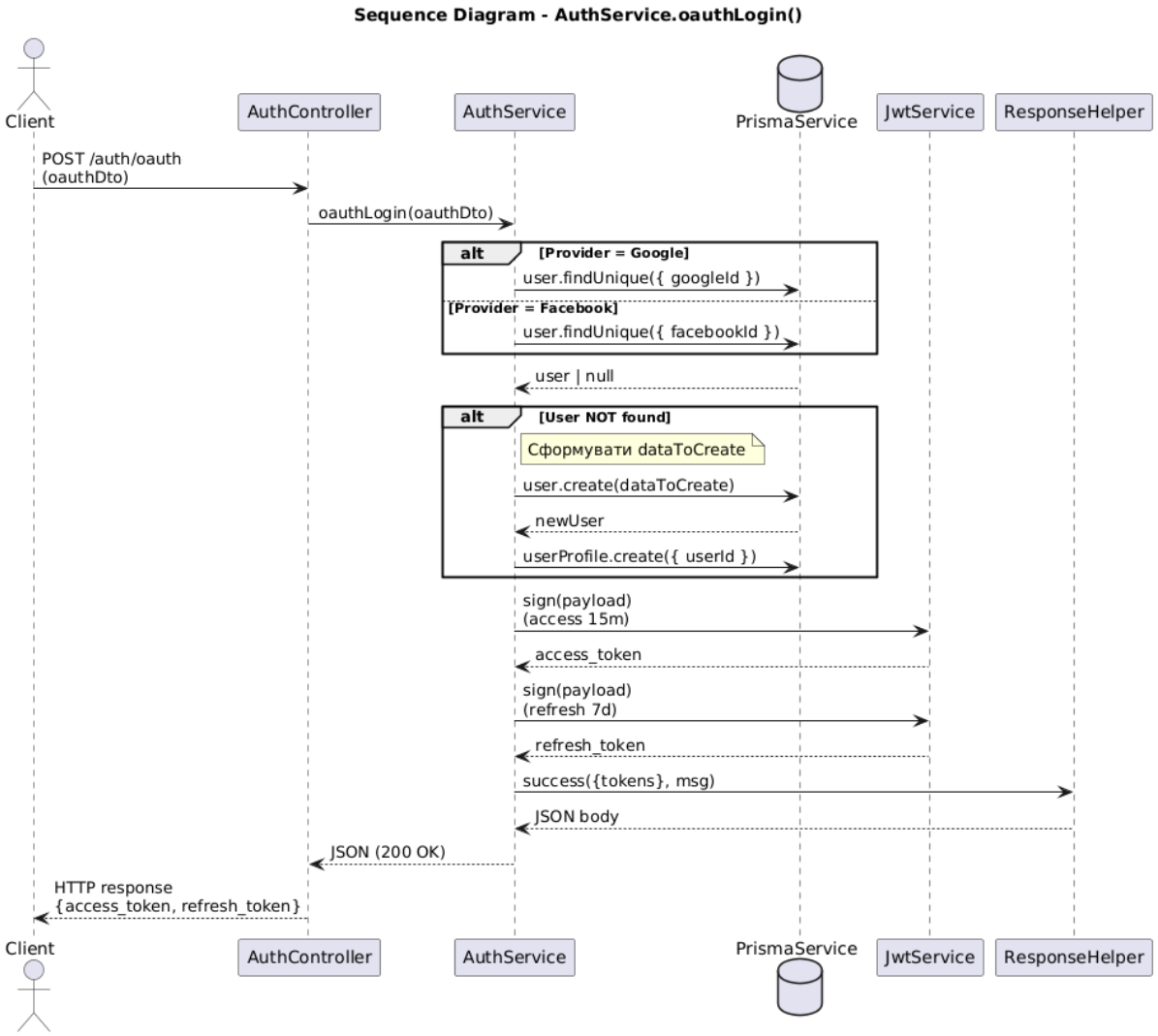

Рисунок 2.14 – Діаграма послідовності UML для методу аутентифікації з використанням сторонніх провайдерів OAuth2

Загалом метод oauthLogin демонструє ефективне поєднання інтеграції із зовнішніми сервісами та адаптації внутрішньої структури користувача в межах єдиної системи, зберігаючи при цьому відповідність принципам безпеки та зручності користувача.

У цілому структура модуля AuthModule, сервісу AuthService та контролеру AuthController відповідає принципам SOLID, забезпечує розділення обов'язків, повторне використання коду та інтеграцію з іншими модулями системи. Це дозволяє ефективно масштабувати функціонал, підвищити безпеку процесів аутентифікації та забезпечити зручність для кінцевого користувача.

2.5 Проєктування клієнтської частини сервісу замовлення їжі

Клієнтська частина інформаційної системи реалізована за допомогою фреймворку Next.js (версія з підтримкою App Router), який забезпечує гібридний рендеринг (SSR/CSR/ISR), покращену продуктивність і сучасний підхід до архітектури React-додатків. Структура проєкту побудована за принципом модульності, що полегшує масштабування, підтримку та супровід розробленого рішення.

Основні структурні частини проєкту (рис. 2.15):

1. app – основний маршрутний рівень (App Router) клієнтського застосунку. Містить дві окремі області:

– main – інтерфейс для клієнтів (користувачів);

– admin – адміністративна частина для керування контентом.

Цей каталог відповідає за визначення сторінок, маршрутизацію та підключення layout-компонентів.

2. assets – каталог статичних ресурсів, включає зображення, логотипи, іконки, банери тощо. Забезпечує централізоване зберігання медіа-вмісту, що використовується на різних сторінках.

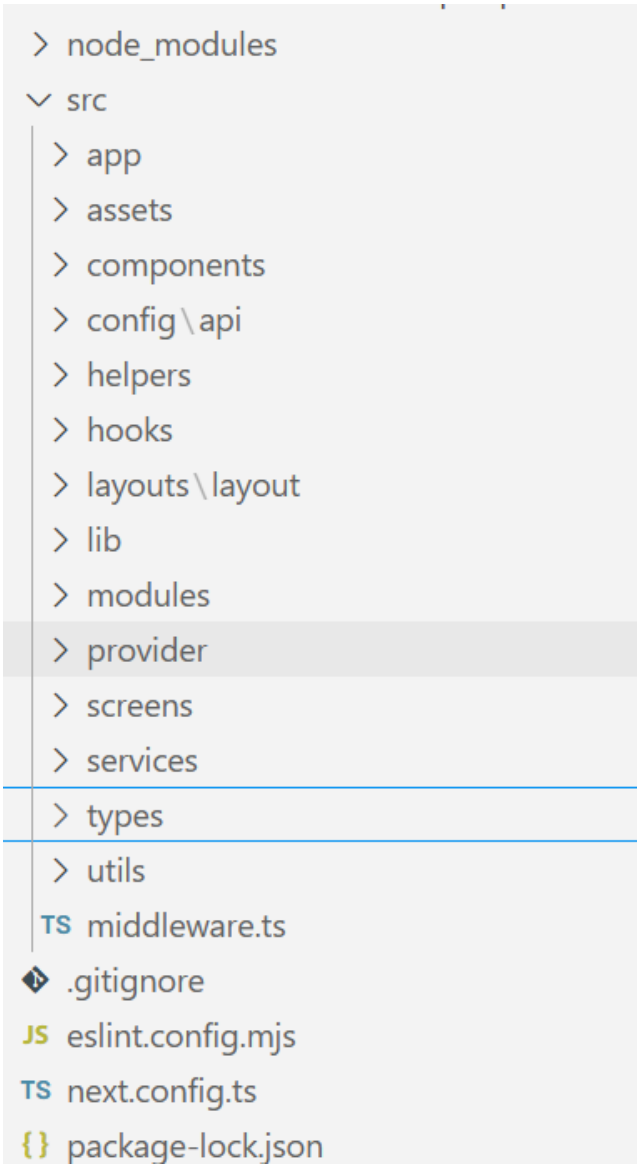

Рисунок 2.15 – Загальна структура клієнтської частини сервісу замовлення їжі

3. components – каталог повторно використовуваних компонентів інтерфейсу. Структуровано за функціональними ознаками: списки страв і ресторанів, сайдбари, модальні вікна (вхід, меню, кошик), а також локальні кастомні хуки, що обслуговують логіку, пов’язану з корзиною, OTP-підтвердженням тощо.

4. config – файл конфігурації, який містить визначення кінцевих точок бекенду (REST API). Зосередження всіх URL-адрес у централізованому місці забезпечує простоту модифікації і тестування.

5. hooks – локальні React-хуки, зокрема хук для підрахунку кількості товарів у кошику. Дозволяє розділити логіку управління станом від візуальних компонентів.

6. layouts – шаблони-обгортки для сторінок. Визначено окремі layout-компоненти для основного клієнтського інтерфейсу та адміністративної частини, що забезпечує єдину структуру та стиль оформлення контенту.

7. lib – бібліотечні функції для роботи з аутентифікацією (JWT), управління токенами доступу, збереженням та отриманням даних із cookies або localStorage. Відокремлення цієї логіки дозволяє забезпечити безпечну обробку конфіденційних даних [30-32].

8. modules – компоненти макета сайту, такі як хедер (верхня навігаційна панель) та футер (нижня панель). Ці елементи повторюються на багатьох сторінках та є частиною загального інтерфейсу користувача.

9. providers – реалізовані провайдери контексту (Context API) для забезпечення глобального стану додатку: зокрема для аутентифікації користувача, управління кошиком тощо.

10. screens – сторінки, що відповідають визначеним маршрутам у app. Включають реалізацію головної сторінки, сторінки ресторану, меню, профілю користувача, а також адміністративні представлення.

11. services – модулі, що інкапсулюють логіку взаємодії з API. Містять функції для надсилання запитів до серверної частини щодо операцій із ресторанами, стравами, кошиком, улюбленим, OTP, профілем користувача тощо. Забезпечують чітке розділення мережевої логіки від компонентів інтерфейсу.

12. types – інтерфейси та типи, що описують структуру даних, які передаються з API або використовуються в компонентах. Забезпечує типобезпечність на рівні всієї кодової бази.

13. utils – утилітні функції для роботи з сесією користувача: створення нової сесії, отримання ідентифікатора поточної сесії, збереження/читання даних

сесії. Використовується для відстеження дій користувача до моменту автентифікації.

Ця структура дозволяє розробити гнучкий, масштабований та підтримуваний інтерфейс клієнтської частини сервісу замовлення їжі, що забезпечує ефективну взаємодію з бекендом, зручність навігації, повторне використання компонентів та відповідність сучасним стандартам веб-розробки.

Для ілюстрації частини виконаної роботи на рис. 2.16 наведено частину коду розробленого компоненту Card, що широко використовується у проєкті для спрощення відображення інформації у вигляді картки на сітці.

Модуль Card, що забезпечує візуальне структурування вмісту інтерфейсу відповідно до принципів UI/UX-дизайну. Цей компонент є універсальним шаблоном картки, який застосовується для подання різноманітної інформації (наприклад, опис страв, ресторанів, оглядів тощо).

Компонент Card реалізовано як реф-орієнтований функціональний компонент із використанням React.forwardRef, що забезпечує підтримку референсів (ref) для зовнішнього керування елементами DOM.

Структурно компонент складається з наступних складових:

1. Card – базова обгортка, що надає стилізацію контейнеру через CSS-класи використаної бібліотеки TailwindCSS [26, 27], зокрема rounded-lg, border, shadow-sm та застосовує кольорову схему bg-card та text-card-foreground;

2. CardHeader– секція для заголовка, що застосовує вертикальне групування елементів з відступами (space-y-1.5, p-6);

3. CardTitle– елемент заголовка, реалізований як HTML-елемент h3з великим розміром шрифту та напівжирним накресленням;

4. CardDescription– допоміжний текст або підзаголовок, стилізований як елемент різ дрібнішим шрифтом та приглушеним кольором;

5. CardContent – основна секція вмісту картки, що надає внутрішні відступи (padding) для розміщення вкладених елементів;

6. CardFooter– нижній блок картки, призначений для розміщення інтерактивних елементів (наприклад, кнопок), вирівняних по осі X.

components > ui > card > Card.tsx > ...

```tsx
import * as React from 'react';
import { cn } from '@/lib/utils';

const Card = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn(
      'rounded-lg border bg-card text-card-foreground shadow-sm',
      className
    )}
    {...props}
  />
));
Card.displayName = 'Card';

const CardHeader = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn('flex flex-col space-y-1.5 p-6', className)}
    {...props}
  />
));
CardHeader.displayName = 'CardHeader';

const CardTitle = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLHeadingElement>
>(({ className, ...props }, ref) => (
  <h3
    ref={ref}
    className={cn(
      'text-2xl font-semibold leading-none tracking-tight',
      className
    )}
    {...props}
  />
));
CardTitle.displayName = 'CardTitle';
```

Рисунок 2.16 – Фрагмент лістингу компоненту Card

Кожен підкомпонент підтримує параметр className, що дозволяє гнучко розширювати стилізацію. Для об’єднання CSS-класів використовується утиліта cn() з модуля /lib/utils.

Таким чином, модуль Card дозволяє реалізовувати погоджену та адаптивну структуру відображення даних в межах різних частин додатку, сприяючи як візуальній цілісності, так і повторному використанню коду.

2.6 Апробація та опис методики застосування розробленого сервісу замовлення їжі

Після переходу за адресою веб-застосунку користувач потрапляє на головну сторінку (рис. 2.17), де відображено лого та слоган сервісу, список кнопок з популярними категоріями, які працюють як фільтр, список рекомендованих закладів, підменю з можливістю обрати страву за її категорією. У верхній частині сторінки розміщено меню навігації, кнопки входу/реєстрації/перегляду акаунта, посилання на корзину та список улюблених страв.

Для зручного пошуку ресторанів за типом кухні реалізовано фільтрацію за категоріями (рис. 2.17). Після вибору категорії (наприклад, «Українська кухня» або «Азіатська кухня») на головній сторінці залишаються лише ті заклади, які відповідають вибраній категорії.

На рис. 2.18 наведено продовження головної сторінки, де відображені рекомендовані страви. Існує можливість фільтрувати страви за категорією (бургери, піца, десерти тощо). При виборі категорії страви завантажуються динамічно з використанням AJAX-запитів, без перезавантаження сторінки (рис. 2.19). Як видно з рис. 2.18, 2.19, існує додаткова можливість фільтрації страв за популярністю, а також за ціною (від дорогих до дешевих і від дешевих до дорогих), а також можливість змінити формат відображення (кількість страв по ширині сторінки).

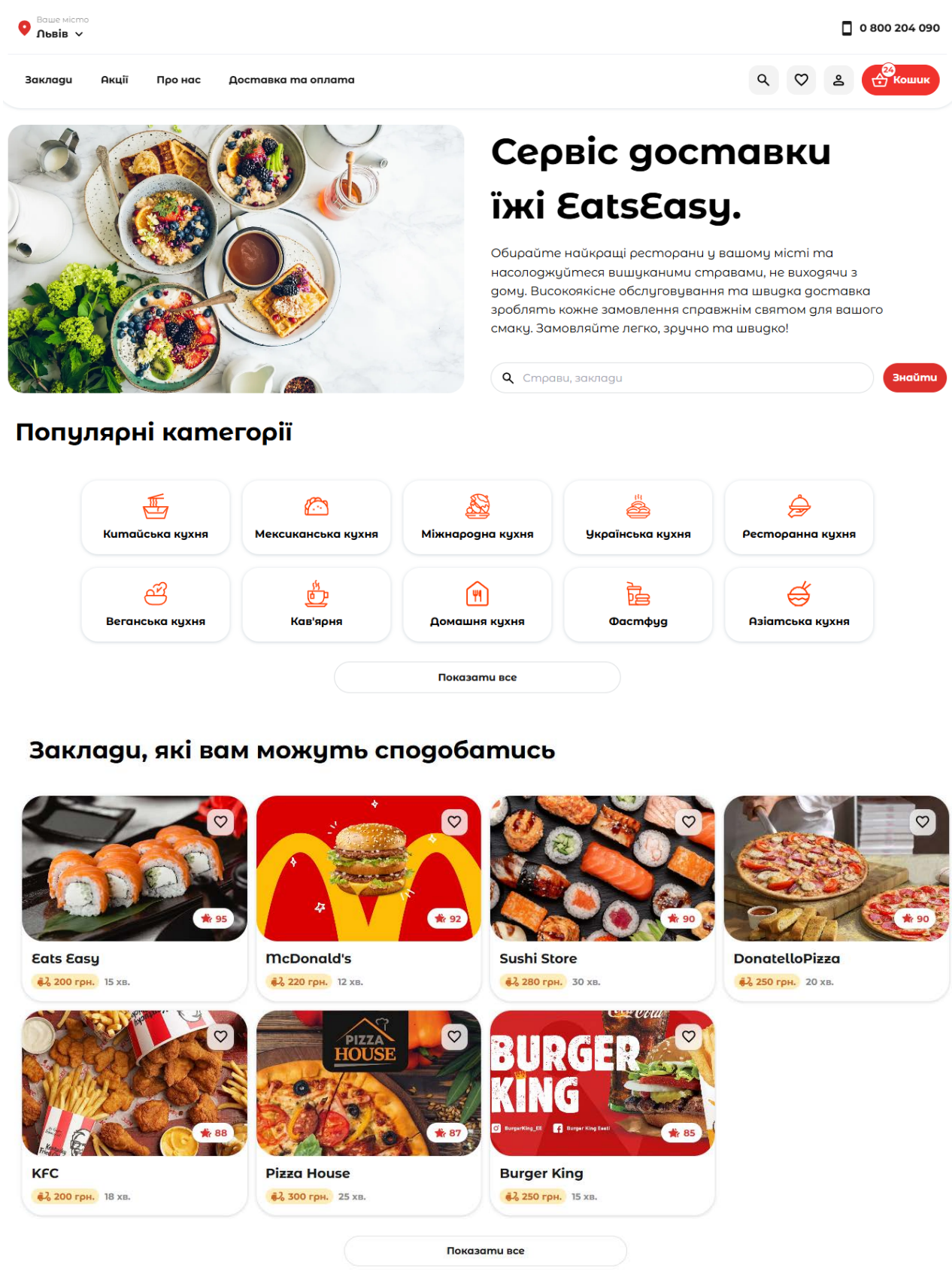

Рисунок 2.17 – Головна сторінка розробленого сервісу

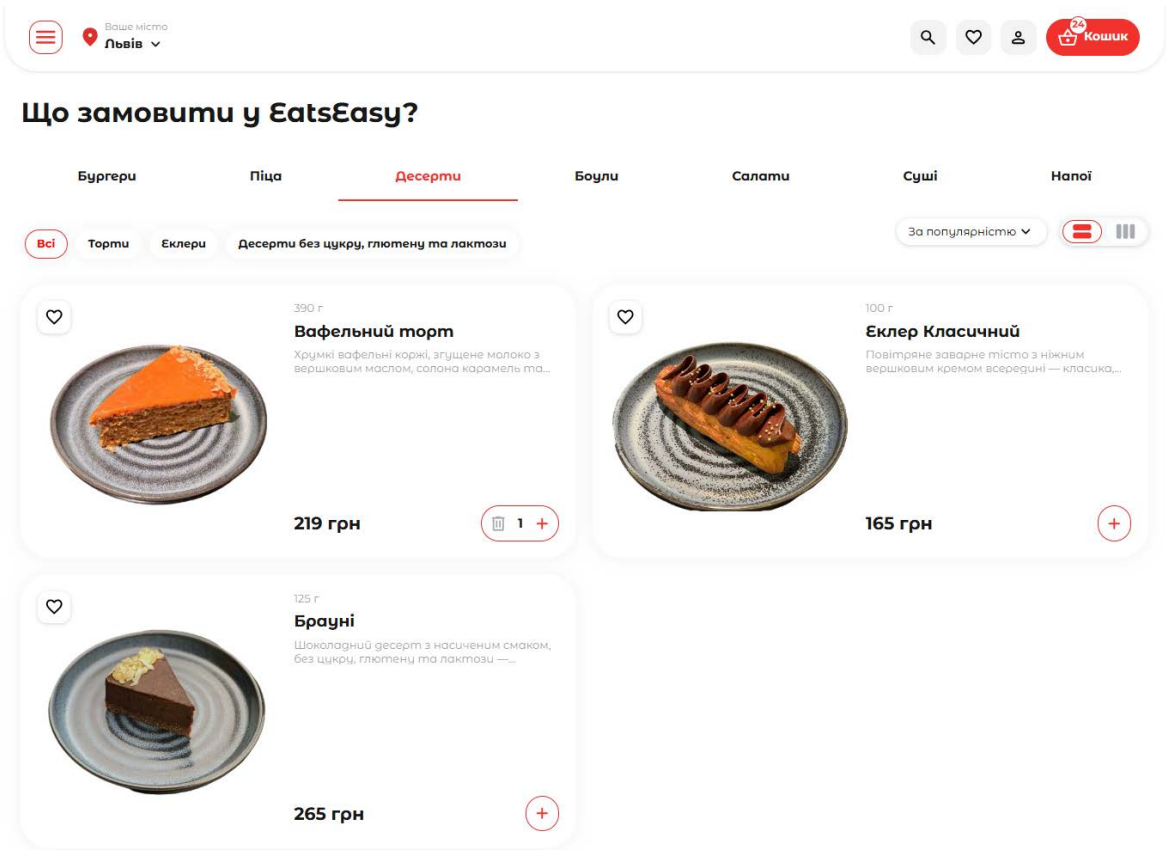

Рисунок 2.18 – Відображення на головній сторінці рекомендованих страв

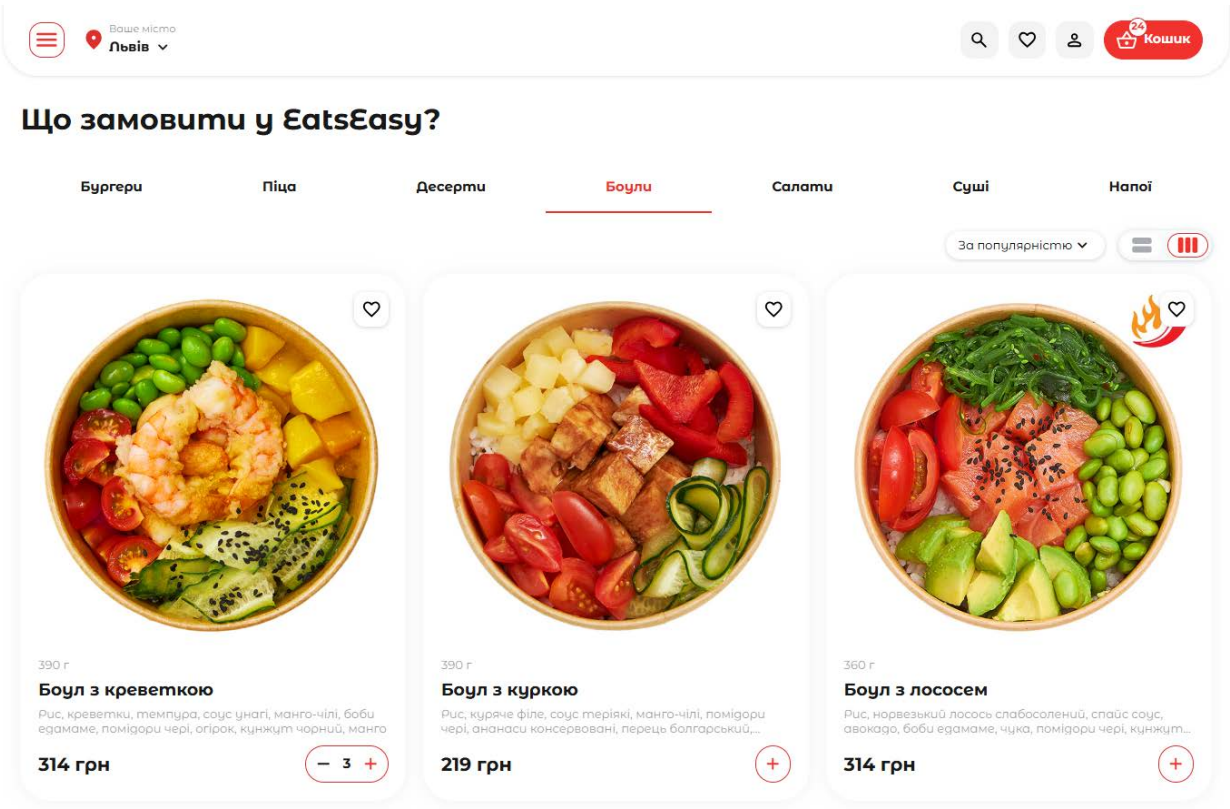

Рисунок 2.19 – Динамічне завантаження страв при виборі іншої категорії

Після вибору ресторану відкривається сторінка з його меню (рис. 2.20). Вона містить загальну інформацію про заклад (рейтинг за відгуками, середня вартість чеку, середня тривалість приготування страви) та перелік доступних страв. Кожна страва супроводжується зображенням, назвою, описом, ціною та кнопкою додавання до кошика.

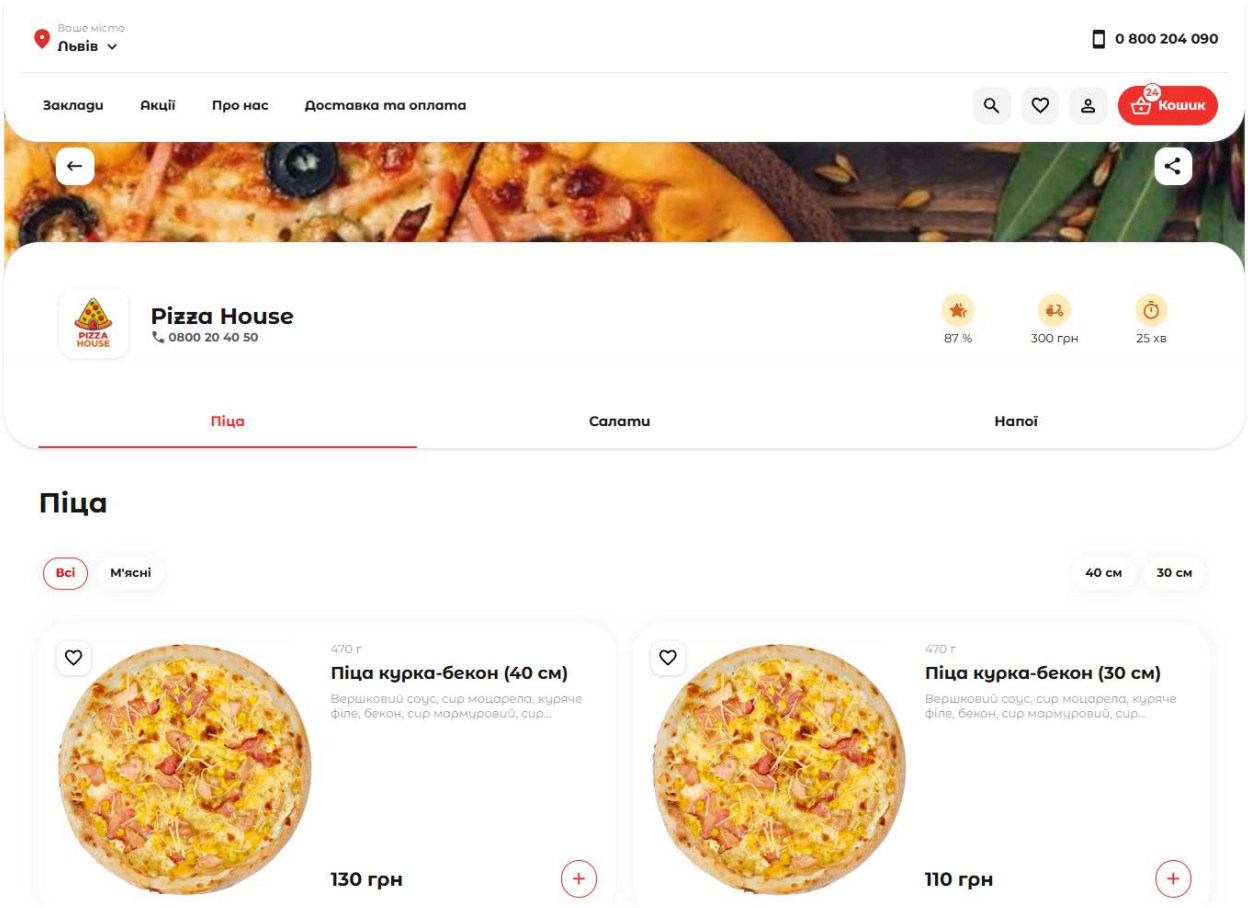

Рисунок 2.20 – Перегляд меню конкретного ресторану

Страви на сторінці структуровані по категоріям страв. Також і меню ресторану структуровано за категоріями (наприклад, «Бургери», «Піца», «Десерти», «Боули», «Салати» тощо). Користувач може обрати потрібну категорію для швидкого переходу до відповідної частини меню (рис. 2.21). Також для різних категорій є додаткові фільтри, які залежать від конкретної категорії. Так, на рис. 2.21 для піци можна застосувати фільтри «Всі», «М’ясні», а також обрати бажаний розмір («30 см», «40 см»). На рис. 2.22 продемонстровано перехід до страв категорії суші на сторінці ресторану.

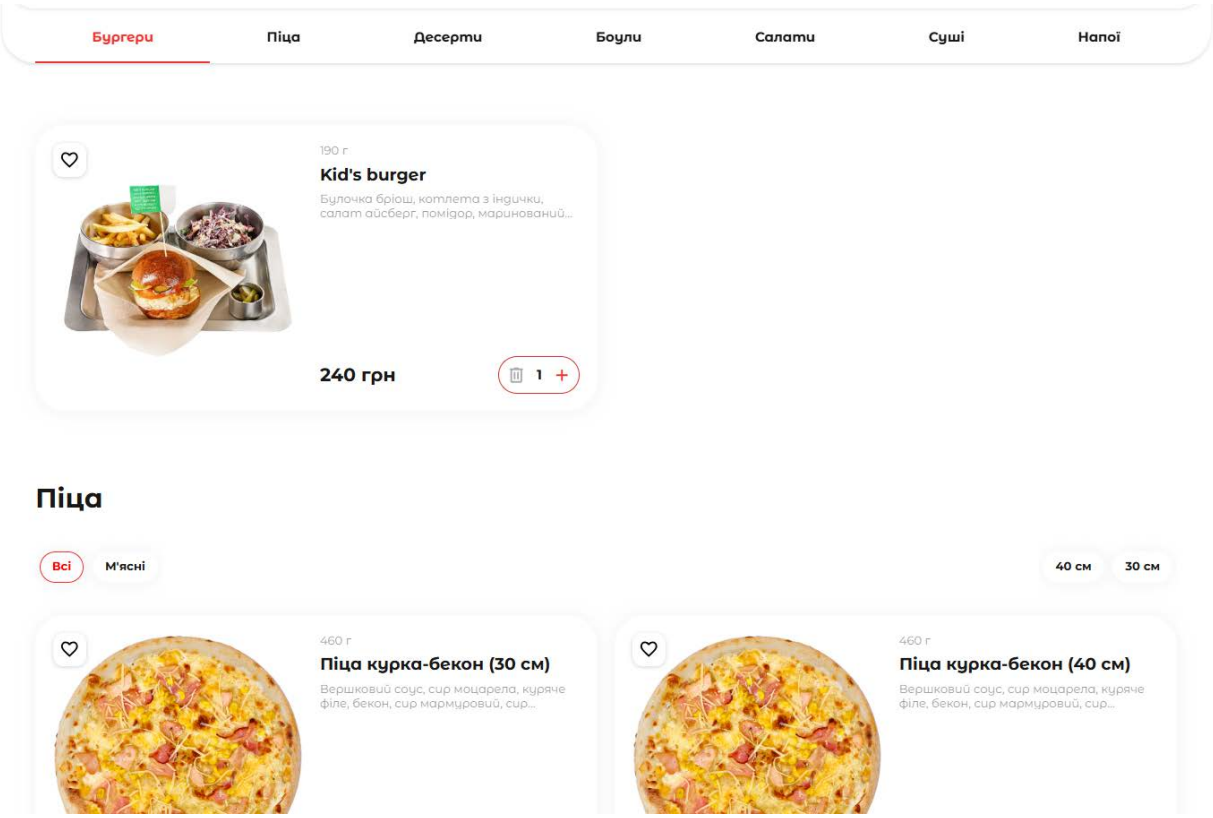

Рисунок 2.21 – Перегляд страв за категоріями

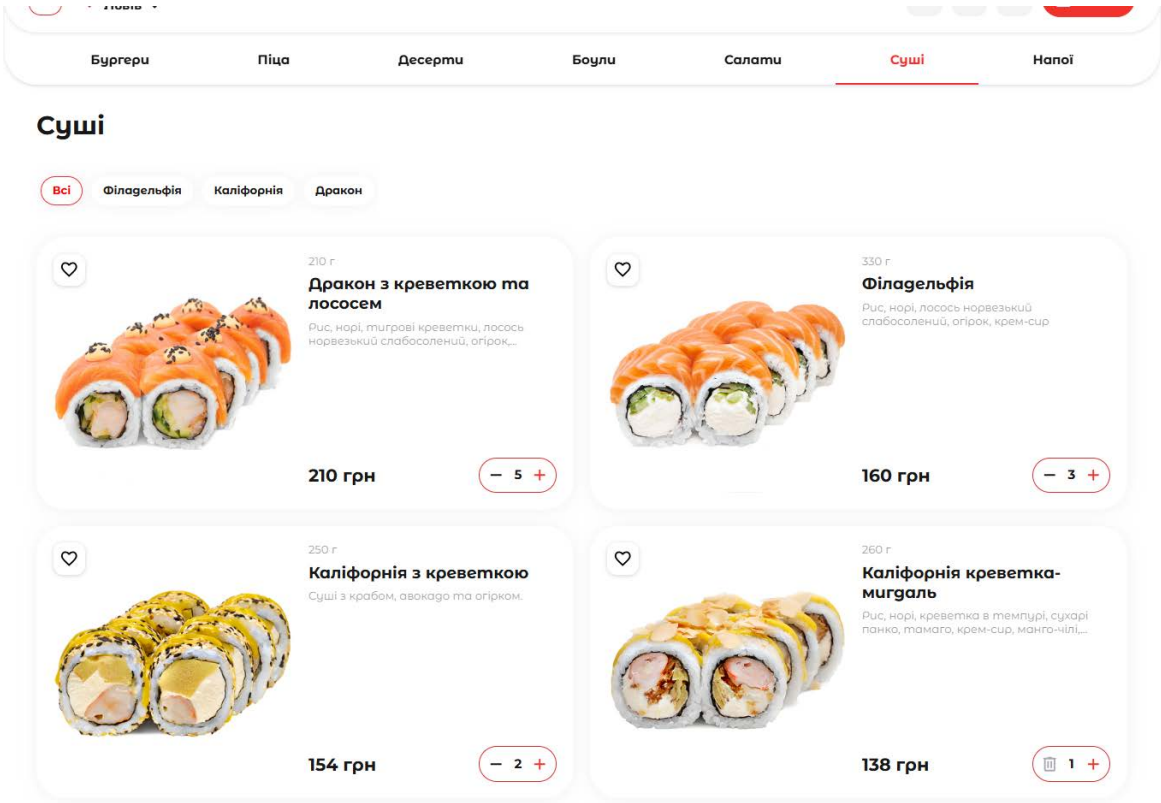

Рисунок 2.22 – Ілюстрація можливості переходу до заданої категорії страв на сторінці ресторану

Після додавання страв у кошик користувач може перейти до перегляду замовлення (рис. 2.23). Кошик відображається як модальне вікно, що спливає у правій частині вікна сервісу.

У кошику відображаються всі обрані позиції, їх кількість, вартість та загальна сума. Перед оформленням замовлення користувач має можливість змінювати кількість страв або вилучати окремі позиції.

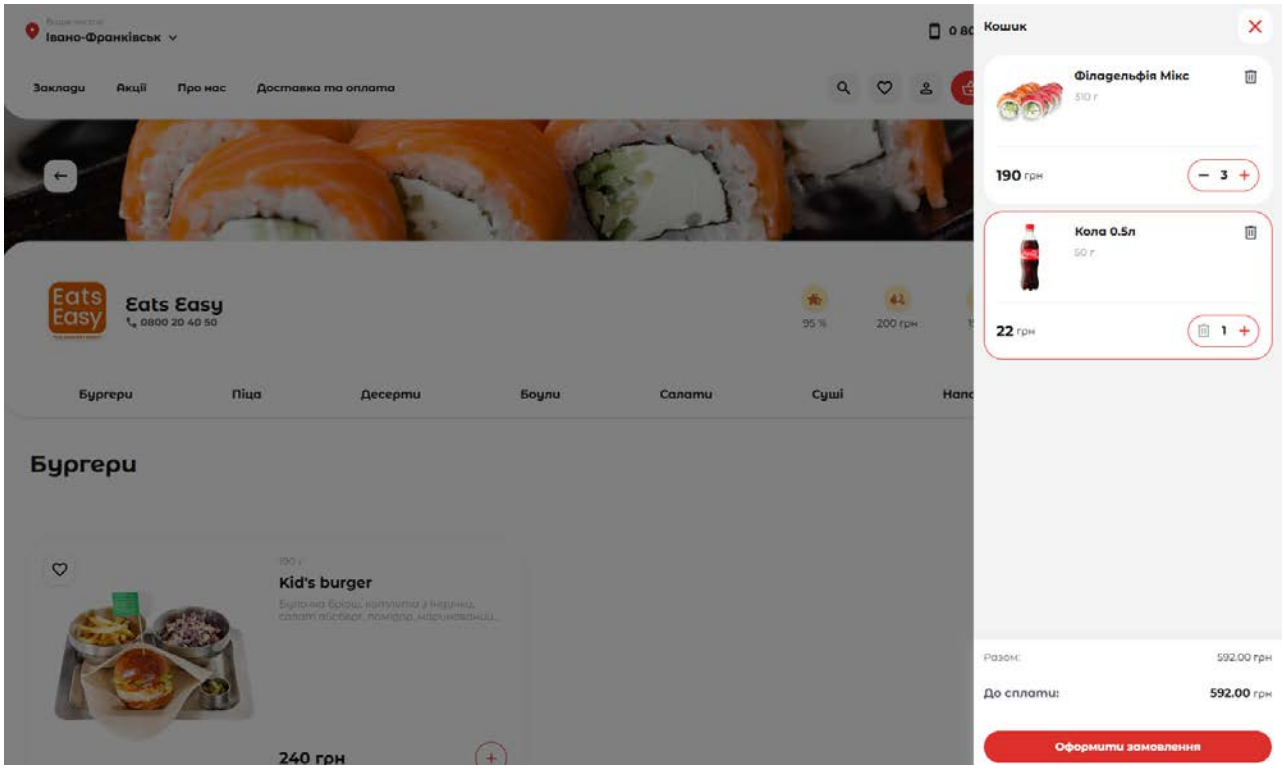

Рисунок 2.23 – Кошик із доданими стравами

У результаті тестування функціональності розробленого сервісу замовлення їжі було підтверджено відповідність реалізованого функціоналу визначеним вимогам. Зокрема, всі основні сценарії використання – перегляд списку ресторанів, фільтрація за категоріями, перегляд меню, взаємодія з кошиком, а також авторизація – працюють коректно та стабільно.

Інтерфейс користувача виявився інтуїтивно зрозумілим і зручним для навігації. Під час тестування жодних критичних помилок або збоїв не виявлено. Усі компоненти клієнтської частини коректно відображають динамічні дані, взаємодія із сервером виконується у режимі реального часу. Окремо варто відзначити коректну роботу механізмів додавання до кошика, зміни кількості

товарів та обчислення загальної вартості замовлення.

Таким чином, система демонструє готовність до практичного використання в умовах реального середовища та подальшого масштабування.

Висновки до розділу:

В другому розділі було здійснено ґрунтовне проектування архітектури веб-сервісу для замовлення їжі, що стало основою для реалізації функціонально цілісної системи. Побудовано компонентну архітектуру з чітким розмежуванням відповідальностей між клієнтською частиною, серверною логікою та зовнішніми інтеграціями. Серверна частина реалізована на основі фреймворку NestJS, що забезпечує модульність, інжекцію залежностей та зручну структуру коду. Детально описано основні модулі, сервіси та контролери, а також представлено діаграми послідовності та компонентів для наочності взаємодії між складовими системами.

Клієнтська частина, створена з використанням Next.js (App Router) має чітко структуровану файлову архітектуру, що передбачає повторне використання компонентів, розмежування логіки за екранами та організацію взаємодії з API.

ВИСНОВКИ

У ході виконання бакалаврської роботи комплексно досліджено та реалізовано веб-сервіс онлайн-замовлення їжі, що відповідає сучасним вимогам масштабованості, зручності користування й безпеки. На етапі постановки задачі обґрунтовано доцільність розробки, виходячи з актуального попиту на дистанційні послуги харчування та недостатньої адаптації глобальних платформ до особливостей локального ресторанного ринку. Проведено огляд провідних міжнародних та вітчизняних рішень (Glovo, Uber Eats, Bolt Food, MixFood), визначено їхні ключові переваги (широка база закладів, персоналізовані рекомендації, програми лояльності) й недоліки (комерціалізація, обмежене покриття малих міст, нестача трекінгу в реальному часі). На основі цього аналізу сформовано функціональні та нефункціональні вимоги до власного сервісу: багаторівнева автентифікація, інтерактивний каталог ресторанів, кошик із підтримкою анонімних сесій, модуль трекінгу доставки, система відгуків, push-сповіщення та програма лояльності.

У першому розділі виконано порівняльний аналіз можливих технологічних стеків (React SPA + Express, Vue + Laravel, Angular + Spring Boot) і обґрунтовано вибір зв’язки NestJS (backend) + Next.js (frontend), що забезпечує єдину мовну базу TypeScript, підтримку SSR/SSG, WebSocket-комунікації й високий потенціал масштабування.

У другому розділі розроблено компонентну архітектуру системи, що складається з презентаційного, логічного та даних рівнів. Виконано проєктування серверної частину на NestJS із модулями аутентифікації (OTP + OAuth), роботи з кошиком, категоріями, стравами, рейтингами та адміністративним інтерфейсом. Для клієнтської частини на Next.js визначено структурну організацію директорій, правила повторного використання компонентів, кастомних хуків і сервісів API. Розгорнуто базу даних MySQL на платформі Aiven.io та сформовано ER-модель, що підтримує гнучкі зв’язки між користувачами, ресторанами, продуктами й акціями. Додатково розроблено

методику використання сервісу: інструкції для реєстрації, оформлення замовлень, відстеження доставки та керування профілем.

Отже, поставлені у вступі мета й завдання виконано повністю: здійснено аналіз предметної області, сформульовано вимоги, обґрунтовано вибір технологій і створено ґрунтовну архітектурну основу для подальшої реалізації та апробації сервісу в реальних умовах експлуатації.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 Доставка Glovo в місті Кривий Ріг. Замовляйте онлайн їжу, продукти, парафармацевтику та багато чого іншого. URL: https://glovoapp.com/ua/uk/kriviy-rig/ (дата звернення: 18.04.2025).

2 Сервіс доставки піци, суши, їжі з ресторанів у Кривому Розі - MixFood. URL: https://krivoy-rog.mixfood.ua/ua/ (дата звернення: 18.04.2025).

3 Documentation | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/ (дата звернення: 23.04.2025).

4 Khan F. A. Nest Js. Medium. URL: https://medium.com/@faz.pak/nest-js-7deb4f80838e (дата звернення: 23.04.2025).

5 Vercel. Next.js Docs | Next.js. Next.js by Vercel - The React Framework. URL: https://nextjs.org/docs (дата звернення: 23.04.2025).

6 Next.js best practices in 2025: Mastering modern web development. August Infotech. URL: https://www.augustinfotech.com/blogs/nextjs-best-practices-in-2025 (дата звернення: 23.04.2025).

7 Up P. Mastering Next.js: Best Practices for Clean, Scalable, and Type-Safe Development. Medium. URL: https://medium.com/@PedalsUp/mastering-next-js-best-practices-for-clean-scalable-and-type-safe-development-626257980e60 (дата звернення: 18.06.2025).

8 Moretti F. Next.js Authentication Best Practices in 2025. Francisco Moretti. URL: https://www.franciscomoretti.com/blog/modern-nextjs-authentication-best-practices (дата звернення: 24.04.2025).

9 Dwyer J. 15 Next.JS Best Practices To Implement For Clean & Responsive Apps. Frontend Monitoring made Easy with Alerty AI. URL: https://alerty.ai/blog/next-js-best-practices (дата звернення: 24.04.2025).

10 Essential Do's and Don'ts in Next.js: Code Like a Pro and Avoid Common Pitfalls!. DEV Community. URL: https://dev.to/aurangzaibramzan/essential-dos-and-donts-in-nextjs-code-like-a-pro-and-avoid-common-pitfalls-4jf6 (дата звернення: 24.04.2025).

11 Vercel. Data Fetching: Data Fetching Patterns and Best Practices | Next.js. Next.js by Vercel - The React Framework. URL: https://nextjs.org/docs/14/app/building-your-application/data-fetching/patterns (дата звернення: 24.04.2025).

12 Design principles, architectural smells and refactorings for microservices: a multivocal review / D. Neri та ін. SICS Software-Intensive Cyber-Physical Systems. 2019. Т. 35, № 1-2. С. 3–15. URL: https://doi.org/10.1007/s00450-019-00407-8 (дата звернення: 28.04.2025).

13 Di Francesco P., Lago P., Malavolta I. Architecting with microservices: A systematic mapping study. Journal of Systems and Software. 2019. Т. 150. С. 77–97. URL: https://doi.org/10.1016/j.jss.2019.01.001 (дата звернення: 28.04.2025).

14 Jha A. How to Build an SPA with React and Node.js. Medium. URL: https://javascript.plainenglish.io/building-a-spa-with-react-and-node-cef18dccef17 (дата звернення: 29.04.2025).

15 The Ultimate Guide for Using Vue.js with Laravel - Vue School Articles. Vue School | The #1 source for learning Vue.js from experts. URL: https://vueschool.io/articles/vuejs-tutorials/the-ultimate-guide-for-using-vue-js-with-laravel/ (дата звернення: 29.04.2025).

16 Використання Vue в Laravel: способи та переваги. FoxmindEd. URL: https://foxminded.ua/vykorystannia-vue-v-laravel/ (дата звернення: 29.04.2025).

17 Harsard. Effortless Full-Stack Deployment: Spring Boot (Java 17) Meets Angular 19 -Part-1. Medium. URL: https://medium.com/@harsard/spring-boot-with-angular-java-17-angular-19-bbbeb6f5eb26 (дата звернення: 29.04.2025).

18 Build a Beautiful CRUD App with Spring Boot and Angular. Auth0 - Blog. URL: https://auth0.com/blog/spring-boot-angular-crud/ (дата звернення: 18.06.2025).

19 Manbar M. How to Build a Full Stack Java Angular Application: Step-by-Step. Medium. URL: https://medium.com/@MohamedManbar/how-to-build-a-full-stack-java-angular-application-step-by-step-d53a17bdcba7 (дата звернення: 18.06.2025).

20 Get started with Prisma | Prisma Documentation. Prisma | Instant Postgres plus an ORM for simpler db workflows. URL: https://www.prisma.io/docs/getting-started (дата звернення: 02.05.2025).

21 Kemal T. Learn Prisma. Medium. URL: https://medium.com/@kemaltf_/learn-prisma-81a019e90b4f (дата звернення: 02.05.2025).

22 Boduroğlu B. Prisma Part 1: Your Easy Tutorial to Set up Prisma. DEV Community. URL: https://dev.to/burakboduroglu/prisma-part-1-your-easy-tutorial-to-set-up-prisma-4p1 (дата звернення: 03.05.2025).

23 Getting Started with Prisma ORM for Node.js and PostgreSQL | Better Stack Community. Better Stack - Radically better observability stack. URL: https://betterstack.com/community/guides/scaling-nodejs/prisma-orm (дата звернення: 03.05.2025).

24 DBML generator for Prisma. notiz. URL: https://notiz.dev/blog/prisma-dbml-generator (дата звернення: 04.05.2025).

25 Prisma ERD. URL: https://prisma-erd.simonknott.de/ (дата звернення: 04.05.2025).

26 Framework guides - Installation. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs/installation/framework-guides (дата звернення: 07.05.2025).

27 GeeksforGeeks. Tailwind CSS Tutorial - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/css/tailwind-css (дата звернення: 07.05.2025).

28 Barracha A. NestJS Authentication with OAuth2.0: Configuration and Operations. DEV Community. URL: https://dev.to/tugascript/nestjs-authentication-with-oauth20-configuration-and-operations-41k (дата звернення: 10.05.2025).

29 Lavagnino F. Google OAuth2 Authentication with NestJS explained. Medium. URL: https://medium.com/@flavtech/google-oauth2-authentication-with-nestjs-explained-ab585c53edec (дата звернення: 10.05.2025).

30 OAuth | NextAuth.js. URL: https://next-auth.js.org/configuration/providers/oauth (дата звернення: 12.05.2025).

31 Udassi P. Using Google OAuth 2.0 with a custom backend in Next.js. DEV Community. URL: https://dev.to/udassi/using-google-oauth-20-with-a-custom-backend-in-nextjs-596d (дата звернення: 18.06.2025).

32 How to implement Social Login (OAuth) in Next.js. Corbado - Simplify passkeys for large-scale applications. URL: https://www.corbado.com/blog/nextjs-login-oauth (дата звернення: 18.06.2025).

33 Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

34 ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

35 ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

36 ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).