Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА СЕРВІСУ ДЛЯ БРОНЮВАННЯ СТОЛИКІВ В УНІВЕРСИТЕТСЬКОМУ КАФЕ З ВИКОРИСТАННЯМ ANGULAR, NODE.JS ТА MYSQL

DEVELOPMENT OF A SERVICE FOR TABLE RESERVATIONS AT THE UNIVERSITY CAFE USING ANGULAR, NODE.JS AND MYSQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Пронін Владислав Олександрович

Керівник:
Вознюк Анастасія Вадимівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Пронін Владислав Олександрович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка сервісу для бронювання столиків в університетському кафе з використанням Angular, node.js та MySQL».*
Керівник роботи: *асистент Вознюк А.В.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02.*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Angular 13+, TypeScript, RxJS, Node.js 16+, Express 4+, MySQL (InnoDB), Docker Compose, JWT / jsonwebtoken, bcrypt, Nodemailer, express-validator, Docker, Git / GitHub Actions, Cypress (E2E), Jest (Unit), Jasmine/Karma (Angular), HTML5 / CSS3 / SCSS, Angular Material, VS Code, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз предметної області та існуючих рішень; формалізація функціональних і нефункціональних вимог; проєктування клієнт-серверної архітектури та бази даних; обґрунтування технологічного стеку; налаштування середовища й реалізація ключових модулів; інтеграція адмін-панелі, обробка зображень та email-сповіщень; модульне, інтеграційне та E2E-тестування з налагодженням; забезпечення інформаційної безпеки та підготовка релізу.*
5. Перелік графічного матеріалу: *14 рисунків.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Вознюк А. В.*		
Специфікація вимог до розробленої системи	*Вознюк А. В.*		
Розробка об`єкта проектування	*Вознюк А. В.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,65 %		
Академічна доброчесність	*Вознюк А. В.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об`єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об`єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об`єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об`єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти Владислав ПРОНІН

Керівник кваліфікаційної роботи Анастасія ВОЗНЮК

АНОТАЦІЯ

Пронін В. О. Розробка сервісу для бронювання столиків в університетському кафе з використанням Angular, node.js та MySQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз предметної області бронювання столиків у студентських кафе, досліджено традиційні й сучасні цифрові рішення, визначено функціональні та нефункціональні вимоги і сформульовано завдання кваліфікаційної роботи. У другому розділі проведено модельне проєктування: побудовано UML-діаграми прецедентів, класів, компонентів і розгортання, спроєктовано архітектуру клієнт-серверної системи з Angular, Node.js/Express і MySQL/InnoDB та описано схему даних і міграції. У третьому розділі реалізовано практичну частину: налаштовано Docker-середовище, створено фронтенд на Angular із модулями Auth, User та Admin, розроблено REST-API на Node.js із JWT-автентифікацією, реалізовано CRUD-інтерфейси для кафе, столиків і бронювань, інтегровано email-сповіщення через Nodemailer, побудовано реактивні форми й валідацію, проведено модульні, інтеграційні та E2E-тести і впроваджено заходи безпеки (HTTPS, helmet, express-validator, rate-limiting). У висновках підкреслено, що виконані роботи забезпечили створення безпечного, продуктивного та масштабованого сервісу бронювання столиків.

Ключові слова: бронювання столиків, Angular, Node.js, MySQL, Docker, JWT, REST-API, Nodemailer, тестування, безпека.

ABSTRACT

Pronin V. Development of a Service for Table Reservations at the University Cafe Using Angular, node.js and MySQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

In the first chapter, an analysis of the domain of table reservations in university cafés was conducted, existing traditional and modern digital solutions were examined, functional and non-functional requirements were defined, and the objectives of the bachelor's qualification work were formulated. In the second chapter, model-based design was carried out: UML diagrams of use cases, classes, components, and deployment were created; the client-server architecture using Angular, Node.js/Express, and MySQL/InnoDB was designed; and the data schema and migrations were described. In the third chapter, the practical part was implemented: a Docker-based development environment was configured; a frontend was built in Angular with Auth, User, and Admin modules; a REST API was developed in Node.js with JWT authentication; CRUD interfaces for cafés, tables, and reservations were implemented; email notifications via Nodemailer were integrated; reactive forms and validation were set up; unit, integration, and end-to-end tests were executed; and security measures (HTTPS, Helmet, express-validator, rate-limiting) were applied. The conclusions emphasize that the completed work resulted in a secure, efficient, and scalable table reservation service.

Keywords: table reservation, Angular, Node.js, MySQL, Docker, JWT, REST API, Nodemailer, testing, security.

ЗМІСТ

ВСТУП.....7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ.....9
1.1 Аналіз сучасного стану проблеми.....9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра.....12
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОГО СЕРВІСУ ДЛЯ БРОНЮВАННЯ СТОЛИКІВ В УНІВЕРСИТЕТСЬКОМУ КАФЕ..... 13
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення.....13
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання... 22
РОЗДІЛ 3 РОЗРОБКА СЕРВІСУ ДЛЯ БРОНЮВАННЯ СТОЛИКІВ В УНІВЕРСИТЕТСЬКОМУ КАФЕ.....28
3.1 Практична реалізація об’єкта проектування.....28
3.2 Тестування та налагодження інформаційно-комп’ютерної системи.....43
3.3 Розробка бази даних.....45
3.4 Захист інформаційно-комп’ютерної системи..... 47
ВИСНОВКИ.....49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....51

ВСТУП

Актуальність теми. У сучасних умовах інтенсивного ритму навчання та постійного завантаження студентів особливого значення набувають цифрові сервіси, що здатні автоматизувати рутинні процеси та підвищити якість обслуговування в університетській спільноті. Однією з таких сфер є харчування: довгі черги в корпусних кафе забирають час, знижують продуктивність і створюють додатковий стрес для молоді. Впровадження онлайн-сервісу для резервування столиків у студентському кафе дозволяє не лише розвантажити заклад у пікові години, а й оптимізувати роботу персоналу, планувати закупівлі та скоротити втрати продуктів.

Студенти можуть заздалегідь перевірити доступність вільних місць, обрати зручний час та гарантовано отримати столик без очікування в черзі. Для адміністрації це також означає точний облік відвідувачів, оперативний аналіз навантаження та можливість оперативно реагувати на нестандартні ситуації (спецзаходи, змагання, зміни розкладу занять).

Цифровий сервіс для бронювання столиків у університетському кафе сприяє підвищенню якості життя в кампусі та формуванню комфорту студентів, що, у свою чергу, відображається на їхньому навчальному процесі та загальному задоволенні від навчання. Використання сучасних технологій (Angular для фронтенду, Node.js для серверної частини та MySQL для надійного зберігання даних) забезпечує масштабованість, швидкість реакції інтерфейсу та безпеку взаємодії – критично важливі характеристики для сервісу, який розрахований на щоденне використання сотнями користувачів.

Мета кваліфікаційної роботи бакалавра – створення вебсервісу для бронювання столиків в університетському кафе, який базується на сучасних технологіях: Angular для клієнтської частини, Node.js для серверної логіки та MySQL для зберігання даних.

Об’єкт кваліфікаційної роботи бакалавра – вебдодаток, що забезпечує користувачам можливість перегляду наявних столиків та попереднього

резервування в університетському кафе, а адміністраторам – управління списками кафе, конфігурацією столиків і обробкою бронювань.

Предмет кваліфікаційної роботи бакалавра – практична реалізація інтерфейсу на Angular, REST-API на Node.js і схеми даних у MySQL для забезпечення коректної взаємодії клієнтської та серверної частин сервісу.

Для досягнення поставленої мети були сформульовані такі завдання:

– проаналізувати предметну область та існуючі рішення;

– визначити та формалізувати функціональні і нефункціональні вимоги;

– спроєктувати архітектуру системи та модель даних;

– обґрунтувати вибір технологічного стеку;

– розробити середовище та реалізувати ключові функціональні модулі;

– інтегрувати адміністративну панель, обробку медіа та розсилки;

– провести тестування та налагодження;

– забезпечити безпеку та підготувати реліз.

Таким чином, впровадження веб-сервісу бронювання столиків у університетському кафе не лише підвищить комфорт і безпеку студентів, а й дозволить адміністрації ефективно планувати навантаження та оптимізувати витрати ресурсів. Використання Angular гарантує швидкий та інтуїтивний інтерфейс, Node.js забезпечує масштабовану серверну логіку, а MySQL – надійну роботу з даними. Розробка такого рішення сприятиме модернізації кампусної інфраструктури та створенню більш комфортного середовища для навчання й відпочинку.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У більшості університетських кафе досі застосовується класичний «ручний» спосіб резервування: студенти або співробітники приходять у заклад, займають місце в черзі або залишають свої дані на папері біля входу. Іноді адміністратори приймають дзвінки або повідомлення в месенджерах, записуючи час та ім’я клієнта у блокноті чи електронній таблиці. Такий підхід працює лише на малих об’ємах відвідувань і не дає жодної гарантії точності – людині доводиться постійно перевіряти журнал, щоб упевнитися у відсутності конфліктів бронювань, а студент, у свою чергу, не отримує офіційного підтвердження й часто змушений приходити завчасно або стояти в черзі «на всяк випадок».

Крім того, ручні методи практично не залишають слідів: немає єдиної бази даних, що ускладнює аналіз пікових годин, планування закупівель продуктів та оцінку навантаження на персонал. Адміністратори витрачають значну частину свого робочого часу на обробку дзвінків і виправлення неточностей у записах, а студентам доводиться постійно телефонувати або навідуватися особисто, щоб переконатися в наявності вільного столика.

Відсутність електронного підтвердження й автоматичних нагадувань призводить до частих «но-шоу», коли заброньоване місце залишається вільним, а кафе втрачає потенційний дохід. У комплексі це створює високий рівень невизначеності та незадоволення для обох сторін – і саме тому університетам потрібен сучасний онлайн-сервіс, який забезпечить прозорий обмін інформацією, автоматичний розподіл столиків і своєчасні повідомлення про стан бронювання.

У глобальному масштабі одним із найрозповсюдженіших рішень для онлайн-бронювання є OpenTable, що пропонує повний набір інструментів:

керування резервуваннями, автоматичне призначення столиків, інтеграцію з POS-системами, а також детальну аналітику та звіти в реальному часі. Платформа об’єднує найбільшу мережу закладів і гостей, забезпечує репутаційний менеджмент і цифровий маркетинг, проте працює за комерційною моделлю щомісячної підписки або комісії з кожного бронювання, що робить її малоефективною для бюджетних закладів освіти [1].

Resy спеціалізується на гнучкому керуванні столиками: автоматичних чергах очікування, миттєвих SMS-підтвердженнях, оптимізації розсадки в реальному часі та розширених аналітичних інструментах. Платформа надає можливість адаптувати доступність столиків під окремі події та є популярною серед ресторанів середнього та преміального сегментів, проте її тарифи та функціонал не передбачають глибокої кастомізації під академічний розклад і внутрішні системи університету [2].

Yelp Reservations (нині Yelp Guest Manager) інтегрує бронювання, керування чергою та takeout-замовленнями, дозволяє надсилати автоматичні нагадування через email і SMS і синхронізується з профілем закладу на Yelp і Google. Ця система також працює за підпискою, пропонує простий інтерфейс для користувачів і власників бізнесу, але не підтримує єдину авторизацію за студентським обліковим записом та не враховує пікові години за розкладом занять [3].

Усі вищезгадані сервіси демонструють високий рівень стабільності й багатий функціонал для комерційних ресторанів, але їхня вартість, відсутність інтеграції зі студентськими системами та недостатня гнучкість у плануванні з урахуванням академічного розкладу створюють прогалини для університетського середовища. Саме тому виникає потреба в розробці спеціалізованого рішення, яке поєднувало б переваги сучасних платформ бронювання з можливістю безшовної інтеграції в інформаційну інфраструктуру вищого навчального закладу.

У процесі розробки сервісу бронювання столиків виникає низка технічних і організаційних викликів, без вирішення яких неможливо гарантувати коректність і стабільність роботи системи.

По-перше, це проблема одночасного доступу декількох користувачів до одних і тих самих ресурсів (столиків) у пікові години. Щоб уникнути «гонок» за одне й те саме бронювання, серверна логіка на Node.js повинна працювати із транзакціями в MySQL або застосовувати механізми блокувань (наприклад, SELECT … FOR UPDATE). Це забезпечує атомарний «захват» обраного столика і виключає ситуацію, коли двоє студентів одночасно підтверджують бронювання одного й того ж місця.

По-друге, для забезпечення швидкої реакції інтерфейсу й відображення актуального стану вільних місць потрібно реалізувати механізм оновлення даних у реальному часі. У нашому випадку клієнтська частина на Angular опитує REST-ендпоінти або використовує WebSocket-з’єднання для отримання оновлень статусу столиків без перезавантаження сторінки. Це дозволяє миттєво блокувати вже заброньовані місця й показувати нові вікна доступного часу.

Третьою важливою задачею є коректна робота з часовими зонами й академічним розкладом: система мусить пропонувати тільки ті часові слоти, які не конфліктують із заняттями студентів, святковими днями та іншими подіями в університеті. Це вимагає інтеграції з внутрішньою системою розкладу або налаштування власного модуля календаря з можливістю задавати корпоративні артифіційні періоди «закриття» кафе.

З організаційної точки зору, впровадження нового сервісу потребує злагодженої роботи IT-відділу та адміністрації кафе. Необхідно чітко прописати правила бронювання (наприклад, максимальна тривалість сесії, час затримки після неявки), навчити персонал користуватися адмін-панеллю та підготувати технічні інструкції для підтримки безперебійної роботи.

Лише з урахуванням цих технічних і організаційних аспектів можна створити надійний, зручний і безпечний сервіс, який відповідатиме потребам студентів та персоналу університетського кафе.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи бакалавра є створення вебсервісу для бронювання столиків в університетському кафе, який базується на сучасних технологіях: Angular для клієнтської частини, Node.js для серверної логіки та MySQL для зберігання даних.

Для досягнення поставленої мети в роботі було сформульовано такі завдання:

– проаналізувати предметну область та існуючі рішення;

– визначити та формалізувати функціональні і нефункціональні вимоги;

– спроєктувати архітектуру системи та модель даних;

– обґрунтувати вибір технологічного стеку;

– розробити середовище та реалізувати ключові функціональні модулі;

– інтегрувати адміністративну панель, обробку медіа та розсилки;

– провести тестування та налагодження;

– забезпечити безпеку та підготувати реліз.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОГО СЕРВІСУ ДЛЯ БРОНЮВАННЯ СТОЛИКІВ В УНІВЕРСИТЕТСЬКОМУ КАФЕ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Успішна реалізація будь-якого програмного рішення неможлива без чіткого й вичерпного опису того, що воно має робити та в яких умовах експлуатуватися. Саме на етапі аналізу вимог виявляються всі зацікавлені сторони, їхні потреби й очікування, узгоджуються обмеження й пріоритети. Коректно сформульовані функціональні та нефункціональні вимоги виступають своєрідною «дорожньою картою» для всіх наступних етапів – від проектування архітектури до тестування й експлуатації, дозволяють уникнути множинних переробок і гарантують, що отриманий продукт відповідатиме реальним запитам користувачів та адміністрації.

У рамках даної роботи вимоги до системи бронювання столиків в університетському кафе розподілені на дві категорії:

Функціональні вимоги:

1) реєстрація та автентифікація користувачів: можливість створити обліковий запис зі студентським email та увійти в систему через JWT;

2) перегляд переліку кафе: відображення списку всіх доступних закладів зі стислими характеристиками (назва, локація, години роботи);

3) перегляд вільних столиків: вибір конкретного кафе й отримання актуального розкладу доступності столиків по часовим слотам;

4) створення бронювання: резервування обраного столика на вказаний інтервал часу з підтвердженням успіху або повідомленням про конфлікт;

5) керування бронюваннями користувачем: перегляд історії власних бронювань, можливість скасувати або змінити час до настання правила dead-line;

6) адміністрування: для ролі «Адміністратор» – CRUD-операції над кафе (додати/редагувати/видалити), столиками та перегляд/підтвердження/скасування бронювань інших користувачів;

7) нагадування та сповіщення: надсилання email-повідомлень про успішне бронювання, наближення часу або скасування.

Нефункціональні вимоги:

1) продуктивність: час відповіді серверу на REST-запит не більше 200 мс за типового навантаження (50 одночасних з'єднань);

2) масштабованість: можливість горизонтального розширення серверної частини (Node.js кластеризація) та розподілу навантаження через load-balancer;

3) надійність та стійкість: гарантія цілісності даних при конкурентному доступі (транзакції MySQL або блокування рядків), резервування й автоматичне відновлення після збоїв;

4) безпека: шифрування всіх з'єднань через HTTPS, захист від SQL-ін'єкцій, XSS, CSRF-атак, надійне зберігання паролів у хешованому вигляді (bcrypt);

5) юзабіліті: адаптивний інтерфейс Angular з чіткою навігацією, інтуїтивно зрозумілою формою бронювання та дотриманням Material Design Guidelines;

6) підтримуваність: чиста модульна структура коду, використання TypeScript із строгими типами, докладна документація API та схеми бази даних;

7) доступність: час безвідмовної роботи не менше 99,5 % на місяць; логування всіх критичних подій і моніторинг через зовнішні сервіси (наприклад, Prometheus/Grafana).

Побудова UML-діаграм є важливим кроком у процесі проєктування програмного забезпечення, оскільки дозволяє візуалізувати ключові аспекти системи на різних рівнях абстракції. Діаграма прецедентів допомагає чітко окреслити всі сценарії взаємодії користувачів і адмінів із сервісом, клас-діаграма формалізує структуру даних і зв'язки між сутностями, діаграма послідовностей ілюструє порядок обміну повідомленнями під час виконання

бізнес-процесів, активності розкривають логіку виконання операцій, а компоненти та розгортання демонструють архітектуру модулів та їх фізичне розміщення. Завдяки такому поетапному моделюванню створюються чіткі вимоги, спрощується комунікація в команді розробки та з адміністрацією закладу, а також закладається міцна основа для подальшої реалізації, тестування й масштабування рішення.

Діаграма прецедентів ілюструє ключові сценарії взаємодії користувача та адміністратора із системою бронювання столиків (рис. 2.1).

Рисунок 2.1 – Діаграма прецедентів

У цій діаграмі представлені дві ролі – «Користувач» і «Адміністратор» – та групи функціональних можливостей системи. «Користувач» спочатку проходить реєстрацію або авторизацію, після чого може переглядати список кафе, дивитися доступність столиків у потрібний часовий проміжок, створювати нові бронювання й керувати власними замовленнями (переглядати історію, змінювати чи скасовувати бронювання). Роль «Адміністратор» охоплює внутрішню панель, де доступні CRUD-операції над сутностями кафе, столиків та бронювань інших користувачів. Така модель прецедентів допомагає чітко сфокусуватися на реальних сценаріях взаємодії та визначити набір функціональних вимог для подальшого проектування й реалізації системи.

Клас-діаграма ілюструє основні сутності предметної області – користувача, кафе, столик та бронювання – та їхні взаємозв'язки (рис. 2.2).

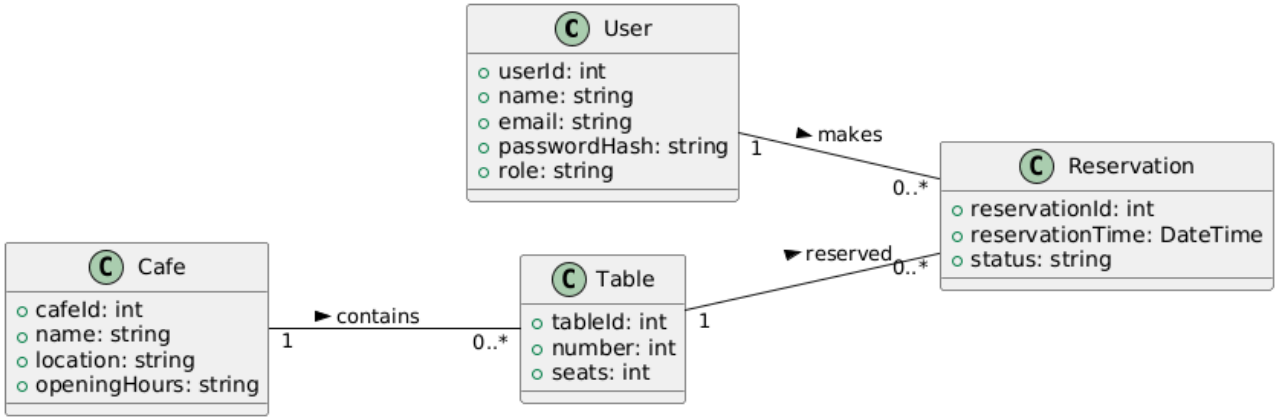

Рисунок 2.2 – Клас-діаграма

У цій діаграмі:

– User містить базові атрибути облікового запису (userId, name, email, passwordHash) та поле role для розрізнення звичайного користувача та адміністратора;

– Cafe описує заклад із cafeId, назвою, місцем розташування і графіком роботи (openingHours);

– Table – це окремий столик із унікальним tableId, номером і кількістю місць (seats);

– Reservation фіксує факт резервування: має власний reservationId, час бронювання (reservationTime) та status (наприклад, «підтверджено», «скасовано»).

Зв’язки моделі відображають бізнес-логіку:

– User → Reservation: один користувач може зробити багато бронювань, але кожне бронювання належить саме одному користувачеві;

– Cafe → Table: кафе володіє набором столиків – від нуля до багатьох;

– Table → Reservation: кожен столик може кілька разів резервуватися в різний час, але кожне бронювання прив’язане до одного столика.

Такий огляд дає чітке уявлення про структуру даних і дозволяє побудувати базу MySQL із таблицями users, cafes, tables та reservations з необхідними зовнішніми ключами та обмеженнями цілісності.

Діаграма послідовностей демонструє послідовність обміну повідомленнями між клієнтом і сервером під час створення нового бронювання (рис. 2.3).

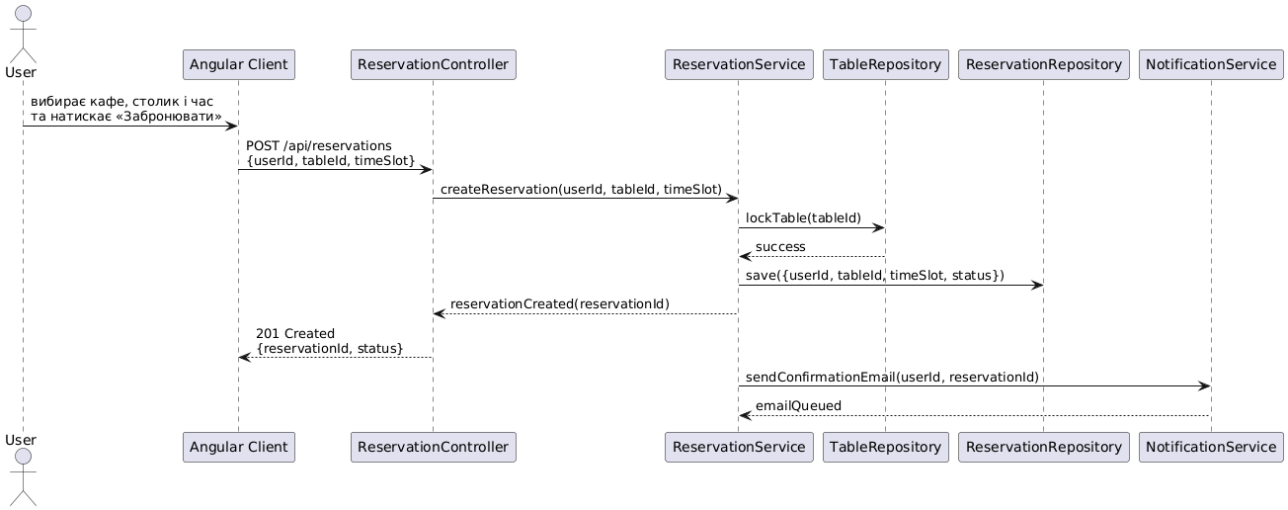

Рисунок 2.3 – Діаграма послідовностей

У цьому сценарії спочатку користувач у клієнтському інтерфейсі Angular обирає конкретний кафе, столик і часовий слот та відправляє POST-запит на контролер резервацій. ReservationController передає отримані дані до бізнес-логіки в ReservationService. Сервіс блокує столик через TableRepository,

щоб уникнути одночасних бронювань, і зберігає новий запис у таблиці бронювань за допомогою ReservationRepository. Після успішного збереження контролер повертає клієнту відповідь із кодом 201 і деталями нового бронювання. Нарешті, паралельно до відповіді клієнту, ReservationService викликає NotificationService для відправки підтверджувального email, який ставиться в чергу на асинхронне виконання. Така послідовність гарантує цілісність даних, швидкий відгук інтерфейсу та своєчасне інформування користувача.

Діаграма активності відображає основний бізнес-процес створення бронювання від початку вибору до завершення підтвердження (рис. 2.4).

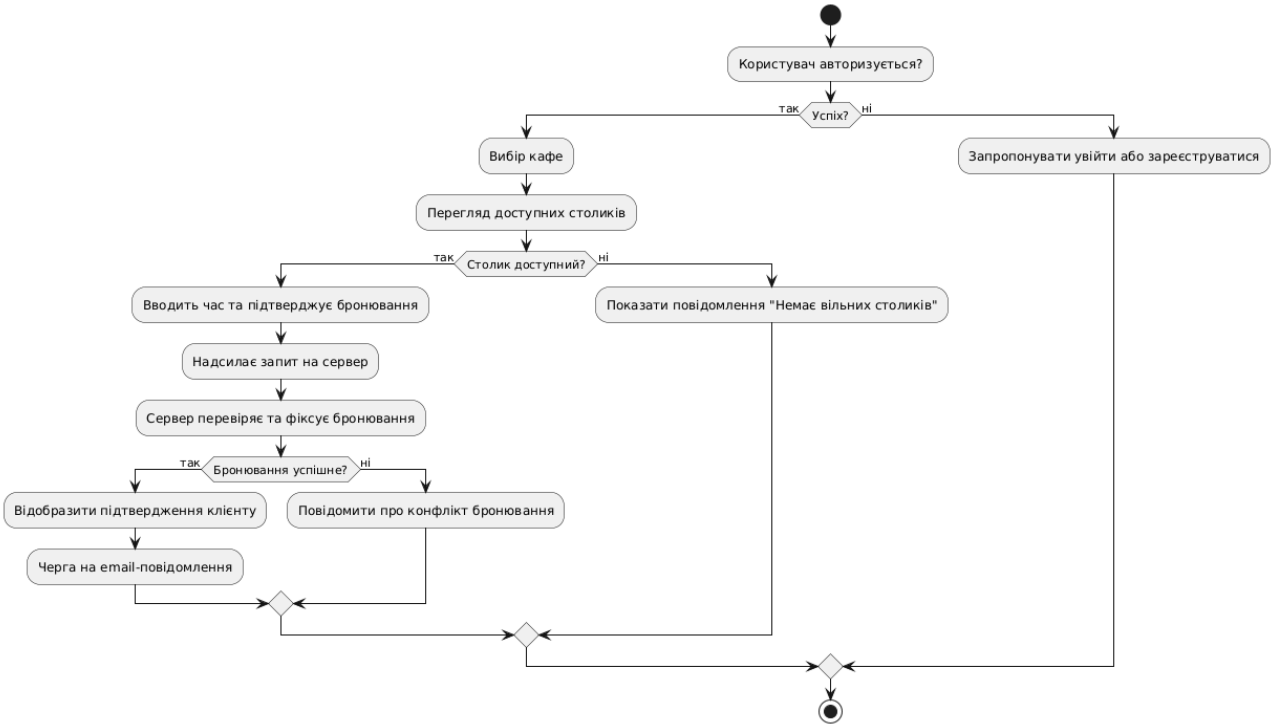

Рисунок 2.4 – Діаграма активності

У цій діаграмі відображено, що спочатку система перевіряє, чи авторизований користувач, і в разі негативного результату пропонує сформувати обліковий запис або увійти. Авторизований користувач обирає кафе та переглядає доступні столики; якщо жодного вільного немає, йому демонструється відповідне повідомлення. При наявності вільного столика

користувач вводить бажаний час, підтверджує бронювання та надсилає запит на сервер. Сервер проводить перевірку доступності й фіксує бронювання в базі даних; у разі успіху користувач отримує на екрані підтвердження замовлення, а система ставить у чергу email-сповіщення для підтвердження через пошту. Якщо ж бронювання неможливе через конфлікт часових слотів, клієнту виводиться повідомлення про помилку. Така послідовність гарантує прозорий користувацький досвід і контроль над цілісністю даних.

Діаграма компонентів показує логічні блоки клієнтської та серверної частин системи й їх взаємодію через REST-інтерфейси та базу даних (рис. 2.5).

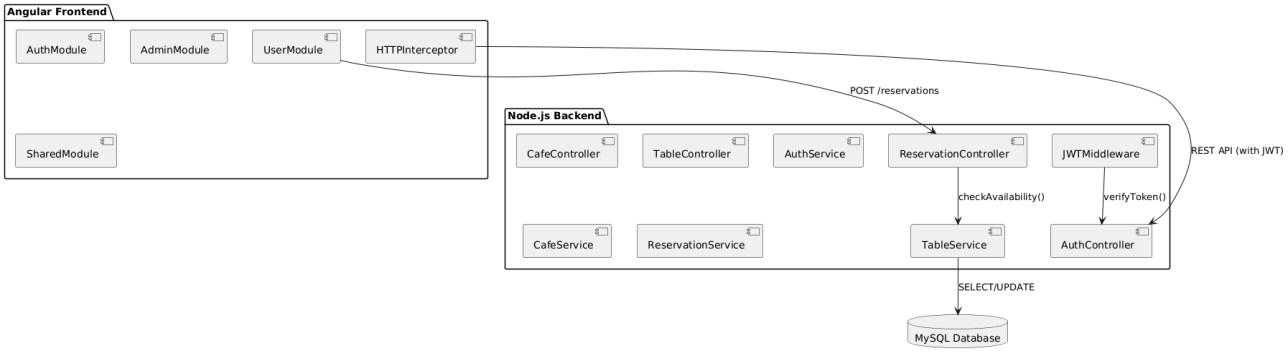

Рисунок 2.5 – Діаграма компонентів

У цій компонентній діаграмі клієнтська частина розбита на модулі Angular: AuthModule для реєстрації/входу, UserModule для перегляду кафе та бронювання, AdminModule для адміністрування, та SharedModule з загальними сервісами та компонентами. HTTP Interceptor відповідає за автоматичне додавання JWT-токена до кожного запиту.

Серверна частина згрупована за контролерами та відповідними сервісами: контролери (AuthController, CafeController, TableController, ReservationController) приймають HTTP-запити й викликають бізнес-логіку у сервісах (AuthService, CafeService тощо). JWT Middleware перевіряє достовірність токена перед передачею запиту контролерам.

Обмін між клієнтом і сервером відбувається через REST API, а сервер виконує SQL-запити до MySQL Database для читання та запису даних. Така

архітектура компонентів забезпечує чітке розділення відповідальностей, спрощує підтримку та розвиток окремих частин проєкту і дозволяє легко масштабувати систему за вертикаллю або горизонталлю.

Діаграма розгортання ілюструє фізичне розміщення компонентів системи та їх мережеві з’єднання (рис. 2.6).

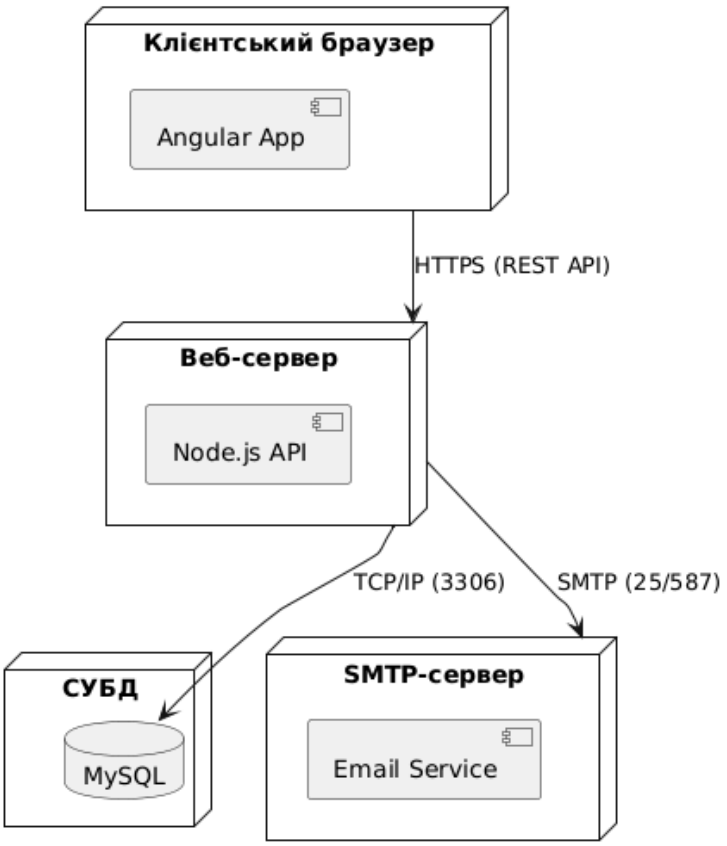

Рисунок 2.6 – Діаграма розгортання

У цій діаграмі:

– клієнтський браузер розгортає Angular-додаток, який взаємодіє з бекендом через захищений HTTPS-з’єднання (REST API);

– вебсервер розгортає Node.js-сервер (Express), який обробляє HTTP-запити, виконує бізнес-логіку та здійснює підключення до бази даних і поштового сервісу;

– СУБД MySQL приймає з’єднання від веб-сервера по TCP/IP на стандартному порту 3306 для читання та запису даних бронювань, користувачів і конфігурації кафе;

– SMTP-сервер використовується для асинхронного надсилання email-повідомлень (наприклад, підтверджень бронювання) через протокол SMTP на порту 25 або 587.

Ця модель показує, як компоненти розміщуються на окремих хостах або контейнерах, а також шляхи їх взаємодії у продакшн-середовищі.

У процесі розробки сервісу для бронювання столиків особливу увагу було приділено UX/UI-дизайну з огляду на специфіку університетського середовища. Інтерфейс створювався на основі корпоративного гайдлайна ЛНТУ, що забезпечило єдину візуальну стилістику та відповідність очікуванням користувачів із кампусу. Для кожного ключового екрану – від логіну й реєстрації до перегляду списку кафе та підтвердження бронювання – побудовано макети з чітким розташуванням елементів управління (кнопок, фільтрів, карток столиків), що зменшують кількість кроків до бажаної дії.

Окремо було змодельовано інформаційні потоки всередині системи: користувацький запит на перевірку вільних слотів спочатку проходить через інтерцептор, далі перетікає в контролер бекенду, а відтворена відповідь оновлює стан компонентів Angular без перезавантаження сторінки. Такий підхід гарантує, що користувачі завжди бачать актуальну інформацію, а затримки в обробці запитів мінімальні.

Технології інформаційного обслуговування й потреби користувачів були проаналізовані через серію фокус-груп із студентами та персоналом кафе. Це дозволило виокремити основні сценарії: швидке бронювання «на ходу», перегляд історії власних бронювань та оперативна взаємодія зі службою підтримки. На основі цих результатів визначено пріоритетність функцій та адаптовано навігацію таким чином, щоб користувачі, які вперше потрапили в додаток, могли інтуїтивно знайти потрібний розділ і завершити бронювання за кілька дотиків.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Angular був обраний як основний фронтенд-фреймворк завдяки своїй комплексності, стабільності й тісній інтеграції з TypeScript, що забезпечує строгі контракти типів та зменшує кількість помилок на етапі розробки. Підтримка офіційного CLI із «нативною» реалізацією таких функцій, як lazy loading модулів, автоматичне тестування та побудова, значно прискорює розробку складних SPA. Вбудована система залежностей (Dependency Injection) і декларативні форми спрощують реалізацію масштабованих бізнес-процесів, а RxJS дозволяє працювати з подіями та HTTP-запитами як з реактивними потоками, що критично для відображення в реальному часі даних про зайнятість столиків.

Згідно з опитуванням Stack Overflow, Angular посідає другу позицію за популярністю серед фронтенд-фреймворків із показником близько 20 % серед розробників (рис. 2.7) [4].

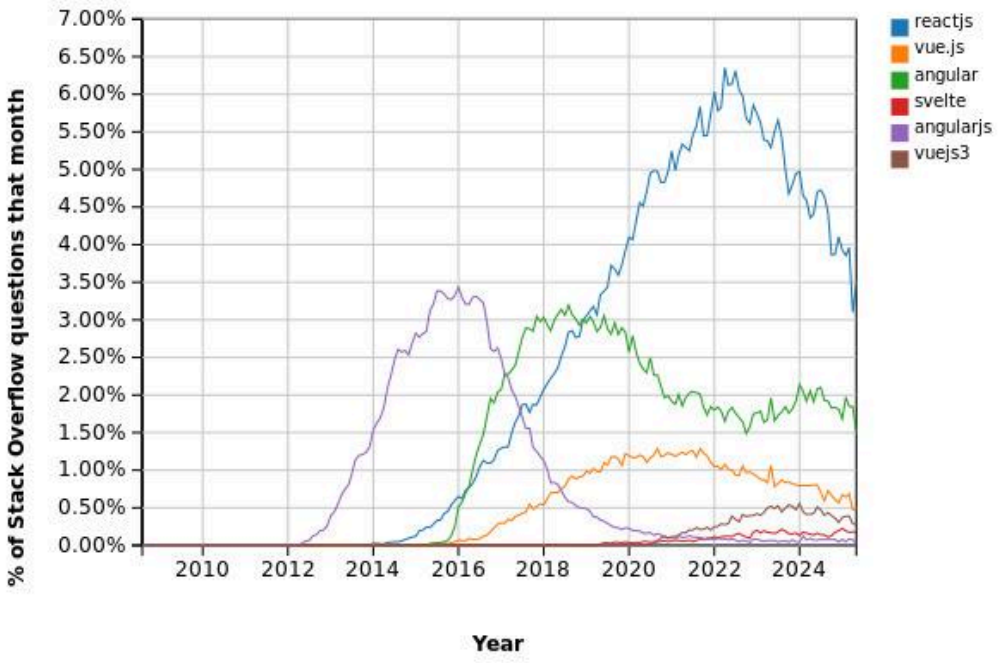

Рисунок 2.7 – Тренди запитань на Stack Overflow [4]

Angular було обрано як основний фронтенд-фреймворк не випадково: його розробляє й підтримує Google, що гарантує довгострокову стійкість та регулярні оновлення від команди, яка стоїть за ключовими вебплатформами компанії. Офіційний Angular-CLI й набір інструментів із коробки (зокрема Dependency Injection, декларативні форми, маршрутизація та підтримка Server-Side Rendering) значно спрощують створення масштабованих SPA з високою продуктивністю і дозволяють дотримуватися єдиного стандарту коду в команді будьякої величини [5].

Крім того, Angular позиціонується як корпоративне рішення: за даними дослідження DevsData, приблизно 58 % розробників у великих компаніях обирають Angular через його «всеводний» стек і консистентну архітектуру, що включає все необхідне для побудови складних додатків без потреби підключати велику кількість сторонніх бібліотек [6]. Наявність строгого набору правил, TypeScript із суворою типізацією та структура на основі компонентів дозволяють зменшити кількість помилок у рантаймі та полегшують підтримку коду, що особливо актуально для університетського сервісу з перспективою подальшого розвитку та інтеграції з іншими системами.

Node.js із фреймворком Express обрано для серверної частини завдяки їхній високій продуктивності в умовах великої кількості одночасних з’єднань і децентралізованій, подіє-орієнтованій моделі обробки запитів. Node.js широко застосовується в комерційних продуктах і має підтверджену репутацію: за даними W3Techs, на Node.js припадає 4,6 % усіх вебсайтів з відомим вебсервером (рис. 2.8), а серед топ-сайтів у категорії фреймворків Express посідає 7-ме місце в топ 10 000 сайтів [7], [8].

Express.js, як найпопулярніший вебфреймворк для Node.js, забезпечує мінімалістичний, але розширюваний шар маршрутизації та middleware, що дозволяє гнучко налаштовувати обробку REST-кінцевих точок, валідацію даних і автентифікацію через JWT. Стек Node.js + Express поєднує асинхронну модель вводу-виводу з простим API, що дозволяє досягати часу відповіді сервера менше за 50 мс у більшості сценаріїв. Крім того, згідно з опитуванням Stack

Overflow, Node.js входить до числа найпопулярніших технологій серед розробників (React і Node.js – лідери в категорії Web Frameworks) [9], що свідчить про широку підтримку спільноти та велику екосистему пакетів. Таким чином, Node.js + Express забезпечують масштабованість, швидкість та можливість легко інтегруватися з іншими компонентами (наприклад, WebSocket-сервісами для реального часу), що є критично важливим для сервісу бронювання з піковими навантаженнями в обідні години.

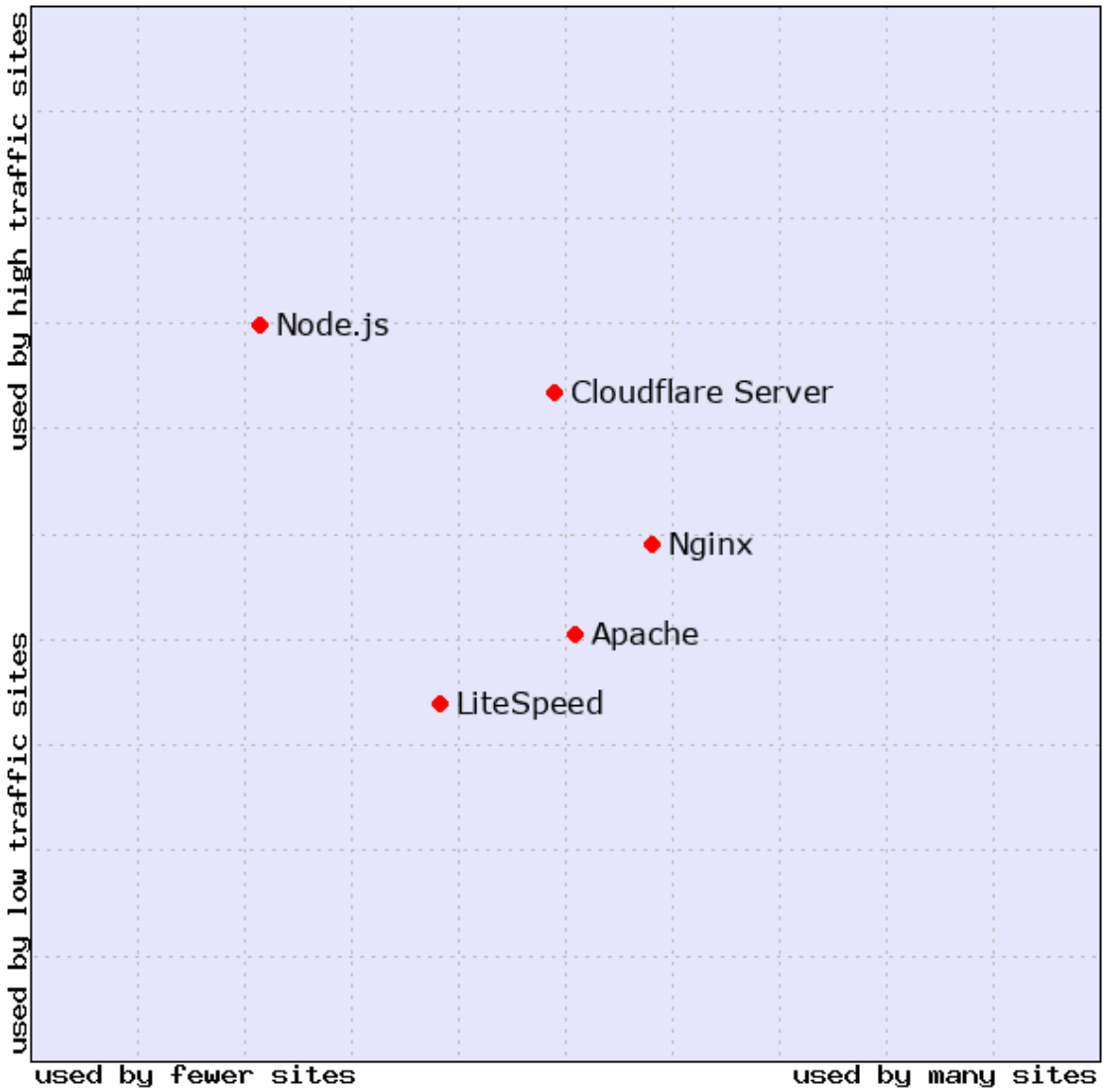

Рисунок 2.8 – Звіт про стан ринку вебсерверів [7]

MySQL з рушієм InnoDB обрано як систему керування базами даних завдяки її доведеній надійності, підтримці ACID-транзакцій і відмінній

продуктивності під навантаженням. InnoDB забезпечує механізми блокування на рівні рядків та відновлення після збоїв, що є критично важливими для коректного обслуговування одночасних бронювань та уникнення «гонок» за один і той же столик. За даними DB-Engines, MySQL стабільно посідає друге місце в рейтингу популярності реляційних СУБД у червні 2025 року, поступаючись лише Oracle, що підтверджує його широке розповсюдження в індустрії та величезну спільноту користувачів (рис. 2.9) [10].

Rank			DBMS	Database Model	Score		
Jun 2025	May 2025	Jun 2024			Jun 2025	May 2025	Jun 2024
1.	1.	1.	Oracle	Relational, Multi-model	1230.38	+3.82	-13.70
2.	2.	2.	MySQL	Relational, Multi-model	953.57	-11.41	-107.77
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	776.75	+1.86	-44.81
4.	4.	4.	PostgreSQL	Relational, Multi-model	680.65	+6.34	+44.41
5.	5.	5.	MongoDB	Document, Multi-model	402.85	+0.33	-18.23
6.	6.	↑ 8.	Snowflake	Relational	174.49	+2.48	+44.13
7.	7.	↓ 6.	Redis	Key-value, Multi-model	151.72	-0.47	-4.22
8.	8.	↑ 9.	IBM Db2	Relational, Multi-model	125.13	-1.27	-0.77
9.	9.	↓ 7.	Elasticsearch	Multi-model	121.28	-2.53	-11.55
10.	10.	10.	SQLite	Relational	117.03	-0.74	+5.63

Рисунок 2.9 – Рейтинг DB-движків [10]

Обрана комбінація Node.js/Express та MySQL/InnoDB дозволяє гарантувати цілісність даних, високу швидкість обробки транзакцій і простоту масштабування шляхом налаштування реплікації та шардування в майбутньому.

Для клієнтської частини застосовано Angular CLI, який автоматично налаштовує Webpack для збирання оптимізованих пакетів із мініфікацією, tree-shaking та lazy-loading модулів. Команди ng serve і ng build --prod дозволяють в один крок запустити локальний дев-сервіс із гарячою перезагрузкою або сформувати готовий до продакшену артефакт.

На сервері використовується ts-node-dev для швидкого перезапуску Node.js-програми при зміні TypeScript-файлів, що забезпечує зручний цикл «змінив-перезавантажив-перевірив» без потреби компілювати вручну.

Для контейнеризації та локального тестування всього стеку застосовано Docker Compose: одна конфігурація піднімає одночасно контейнери для клієнта, сервера та бази даних MySQL, гарантуючи ідентичність середовища в усіх членів команди та на CI-системі.

GitHub Actions налаштовано як CI/CD-пайплайн: при кожному пуші у гілку main автоматично запускаються лінтери (ESLint, Prettier), модульні тести (Jasmine/Karma для Angular, Jest для Node.js), збірка фронтенду та бэкенду, а успішні артефакти деплояться на тестовий сервер або зберігаються як релізні пакети. Такий набір інструментів і сценаріїв автоматизації забезпечує високу якість коду, стабільність збірок та швидкий зворотний зв’язок для розробників.

Розробка ведеться у контейнеризованому середовищі Docker, що гарантує однакові залежності та конфігурації на локальних машинах і в CI. Для кодування й налагодження використовується Visual Studio Code із розширеннями для TypeScript, ESLint і Docker, що прискорює інтеграцію з репозиторієм і дозволяє працювати з контейнерами безпосередньо з IDE.

У ролі DevOps-утиліт застосовано Docker Compose для одночасного підняття клієнтського, серверного та базового контейнерів, а також GitHub Actions як CI/CD-систему. Пайплайни в GitHub Actions виконують перевірку стилю коду, збірку артефактів, запуск тестів і розгортання на тестовому середовищі. Моніторинг і логування продуктивності сервера реалізовано через інтеграцію з Prometheus і Grafana, що дозволяє в режимі реального часу відстежувати метрики завантаження CPU, пам’яті й затримки REST-ендпоінтів. Для централізованого зберігання логів використовується ELK-стек (Elasticsearch, Logstash, Kibana), що полегшує пошук і аналіз помилок у продакшені.

Таким чином, обраний технологічний стек – Angular на фронтенді, Node.js з Express на бекенді і MySQL/InnoDB для зберігання даних – у поєднанні з TypeScript, RxJS, Docker, CI/CD-процесами та потужними DevOps-утилітами забезпечує швидкий і зрозумілий цикл розробки, легке масштабування та високу надійність сервісу. Angular CLI й модульна архітектура дозволяють

оперативно впроваджувати нові функції в інтерфейс, Node.js + Express гарантують асинхронну обробку запитів із мінімальними затримками, а MySQL/InnoDB дає змогу безпечно керувати транзакціями. Контейнеризація Docker і автоматизовані пайплайни GitHub Actions оптимізують тестування, збірку й розгортання, що робить процес інтеграції й доставки кожного оновлення максимально безшовним і передбачуваним. Такий підхід суттєво підвищує продуктивність команди розробників і спрощує подальше обслуговування та розвиток системи.

РОЗДІЛ 3
РОЗРОБКА СЕРВІСУ ДЛЯ БРОНЮВАННЯ СТОЛИКІВ В УНІВЕРСИТЕТСЬКОМУ КАФЕ

3.1 Практична реалізація об’єкта проектування

Спочатку встановлено Docker і Docker Compose (версія ≥ 1.29) згідно з офіційною документацією docs.docker.com [11] на локальному ПК. Далі у корені проєкту створено файл docker-compose.yml (ліст. 3.1), який описує три сервіси:

– mysql – офіційний образ mysql:8.0, змінні середовища для налаштування користувача, пароля і назви бази, а також монтування каталогу ./mysql/data для збереження даних між перезапусками;

– server – збірка власного образу з Dockerfile в папці server/, в якому встановлено Node.js (використовується образ node:16-alpine), встановлюються залежності через npm ci, компілюється TypeScript і запускається в режимі розробки за допомогою ts-node-dev;

– client – образ на базі node:16-alpine, де виконується npm ci у директорії client/ і потім стартує Angular CLI (ng serve --host 0.0.0.0), щоб забезпечити доступ із браузера хоста.

Лістинг 3.1 – Фрагмент docker-compose.yml

```
version: '3.8'
services:
  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: cafe_db
      MYSQL_USER: cafe_user
      MYSQL_PASSWORD: userpass
    volumes:
      - ./mysql/data:/var/lib/mysql
    ports:
      - "3306:3306"

  server:
    build: ./server
    volumes:
```

```
      - ./server:/app
      - /app/node_modules
    ports:
      - "3000:3000"
    depends_on:
      - mysql
    environment:
      DB_HOST: mysql
      DB_USER: cafe_user
      DB_PASS: userpass
      DB_NAME: cafe_db

  client:
    build: ./client
    volumes:
      - ./client:/app
      - /app/node_modules
    ports:
      - "4200:4200"
    command: ng serve --host 0.0.0.0 --port 4200
    depends_on:
      - server
```

кінець лістингу 3.1

У результаті цього налаштування отримано єдину команду для старту всіх частин проєкту – docker-compose up – і готовий до роботи інструментарій для розробки та налагодження.

Нижче наведено фрагмент виводу команди tree -L 2, яка показує ключові директорії в корені проєкту (ліст. 3.2).

Лістинг 3.2 – Структура каталогу на рівні 2

```
.
├── .github
│   └── workflows
│       └── ci.yml
├── client
│   ├── e2e
│   ├── src
│   ├── angular.json
│   └── package.json
├── server
│   ├── src
│   │   ├── controllers
│   │   ├── middlewares
│   │   ├── models
│   │   ├── repositories
```

```
│   │   ├── routes
│   │   └── services
│   ├── tsconfig.json
│   └── package.json
├── mysql
│   └── data
├── docker-compose.yml
├── Dockerfile.client
├── Dockerfile.server
└── README.md
```

кінець лістингу 3.2

Пояснення ключових директорій і файлів:

– .github/workflows/ci.yml – налаштування GitHub Actions для CI (лінтування, тестування, збірка);

– client/ – Angular-додаток:

а) src/ – вихідний код з модулями (app/auth, app/user, app/admin, app/shared);

б) e2e/ – інтеграційні тести cypress/protractor;

в) angular.json, package.json – конфігурація збірки та залежностей;

– server/ – Node.js API на TypeScript:

а) src/controllers/ – обробники REST-запитів;

б) src/routes/ – визначення маршрутів, що зв’язують URI з контролерами;

в) src/services/ – бізнес-логіка (перевірка доступності столиків, надсилання сповіщень);

г) src/repositories/ – прямі звернення до MySQL (через mysql2 або ORM);

д) src/models/ – опис структур даних/інтерфейсів TypeScript;

е) src/middlewares/ – JWT-автентифікація й обробка помилок;

є) tsconfig.json, package.json – конфігурація TypeScript та залежності;

– mysql/data/ – том Docker для збереження даних СУБД;

– docker-compose.yml – підйом трьох сервісів: клієнт, сервер, MySQL;

– Dockerfile.client та Dockerfile.server – окремі Dockerfile для фронтенду й бекенду;

– README.md – загальна документація з інструкціями щодо встановлення, запуску та тестування проєкту.

Така організація папок і конфігураційних файлів забезпечує розділення обов’язків між CI/CD, фронтендом, бекендом і БД, а також полегшує роботу в команді та подальше масштабування.

Реєстрацію та авторизацію користувачів реалізовано із використанням JWT, щоб захистити API та керувати доступом до ресурсів.

Спочатку клієнт надсилає POST-запит на /api/auth/register, передаючи email і пароль. У контролері пароль хешується через bcrypt, створюється запис у БД, і клієнт отримує підтвердження реєстрації. Далі при логіні (/api/auth/login) контролер порівнює переданий пароль із хешем у БД і, якщо дані коректні, генерує JWT із корисним навантаженням (userId, role) та терміном дії 8 годин (ліст. 3.3).

Лістинг 3.3 – AuthController

```
import { Request, Response } from 'express';
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt';
import { UserRepository } from '../repositories/userRepository';

export class AuthController {
  static async register(req: Request, res: Response) {
    const { email, password } = req.body;
    const hash = await bcrypt.hash(password, 10);
    const user = await UserRepository.create({ email,
passwordHash: hash, role: 'user' });
    return res.status(201).json({ id: user.id, email: user.email
});
  }

  static async login(req: Request, res: Response) {
    const { email, password } = req.body;
    const user = await UserRepository.findByEmail(email);
    if (!user || !(await bcrypt.compare(password,
user.passwordHash))) {
      return res.status(401).json({ message: 'Invalid credentials'
});
    }
```

```
    const token = jwt.sign(
      { userId: user.id, role: user.role },
      process.env.JWT_SECRET as string,
      { expiresIn: '8h' }
    );
    return res.json({ token });
  }
}
```

кінець лістингу 3.3

Після отримання токена клієнтська частина зберігає його в localStorage і автоматично додає до заголовків всіх наступних запитів за допомогою HTTP-Interceptor (ліст. 3.4).

Лістинг 3.4 – AuthService та HTTP Interceptor

```
@Injectable({ providedIn: 'root' })
export class AuthService {
  private api = 'http://localhost:3000/api/auth';
  constructor(private http: HttpClient) { }

  register(email: string, password: string) {
    return this.http.post(`${this.api}/register`, { email,
password });
  }

  login(email: string, password: string) {
    return this.http.post<{ token: string }>(`${this.api}/login`,
{ email, password })
      .pipe(tap(res => localStorage.setItem('jwt', res.token)));
  }

  logout() {
    localStorage.removeItem('jwt');
  }

  get token(): string | null {
    return localStorage.getItem('jwt');
  }
}

@Injectable()
export class JwtInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const token = localStorage.getItem('jwt');
    if (token) {
      req = req.clone({ setHeaders: { Authorization: `Bearer
${token}` } });
```

```
    }
    return next.handle(req);
  }
}
```

кінець лістингу 3.4

HTTP-Interceptor перевіряє наявність JWT у localStorage і додає заголовок Authorization: Bearer <token>, що дозволяє Express-middleware на сервері підтверджувати дійсність токена перед виконанням захищених роутів. У разі простроченого або некоректного токена сервер повертає 401, а клієнтка обробляє це переведенням користувача на сторінку входу. Такий підхід гарантує безпечний доступ до кінцевих точок API та чіткий контроль ролей (user vs. admin) на кожному запиті.

На наведеному скриншоті (рис. 3.1) представлено модальне вікно входу в систему «ЛНТУ Кафе». У центральній частині модалки містяться два поля для введення студентської електронної пошти і пароля.

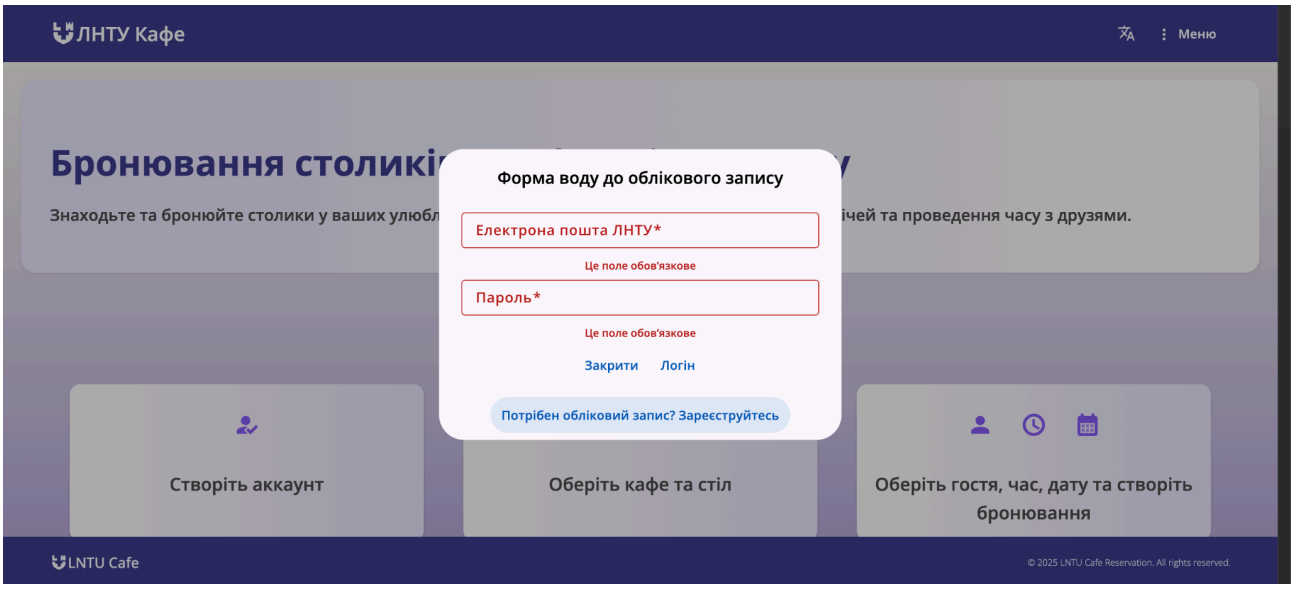

Рисунок 3.1 – Модальне вікно входу в систему «ЛНТУ Кафе»

Далі потрібно розглянути реалізацію базових CRUD-операцій для двох ключових сутностей системи – кафе та столиків. Адміністратор через інтерфейс Angular може створювати нові кав’ярні, редагувати їхні властивості (назва,

локація, години роботи), видаляти непотрібні записи та аналогічно керувати набором столиків у кожному кафе (номер столика, кількість місць). На сервері кожний запит обробляє відповідний контролер, а бізнес-логіка групується у сервісах, що гарантує чистоту коду та легке тестування (ліст. 3.5).

Лістинг 3.5 – CafeController

```
import { Request, Response } from 'express';
import { CafeService } from '../services/cafeService';

export class CafeController {
  static async list(req: Request, res: Response) {
    const cafes = await CafeService.getAll();
    res.json(cafes);
  }

  static async getOne(req: Request, res: Response) {
    const cafe = await CafeService.getById(+req.params.id);
    res.json(cafe);
  }

  static async create(req: Request, res: Response) {
    const newCafe = await CafeService.create(req.body);
    res.status(201).json(newCafe);
  }

  static async update(req: Request, res: Response) {
    const updated = await CafeService.update(+req.params.id,
req.body);
    res.json(updated);
  }

  static async delete(req: Request, res: Response) {
    await CafeService.delete(+req.params.id);
    res.status(204).send();
  }
}
```

кінець лістингу 3.5

Контролер CafeController приймає HTTP-запити та викликає методи сервісу CafeService, який взаємодіє з базою даних. Кожен метод відповідає одній із CRUD-операцій: list і getOne читають дані, create додає нове кафе, update змінює існуюче, а delete – видаляє запис (ліст. 3.6).

Лістинг 3.6 – AdminCafeComponent

```
@Component({ /* ... */ })
export class AdminCafeComponent implements OnInit {
  cafes: Cafe[] = [];
  selectedCafe: Cafe | null = null;

  constructor(private cafeService: CafeService, private dialog:
MatDialog) { }

  ngOnInit() {
    this.loadList();
  }

  loadList() {
    this.cafeService.getAll().subscribe(list => this.cafes =
list);
  }

  openEdit(cafe?: Cafe) {
    const dialogRef = this.dialog.open(CafeFormDialog, { data:
cafe });
    dialogRef.afterClosed().subscribe(result => {
      if (result) this.loadList();
    });
  }

  delete(id: number) {
    this.cafeService.delete(id).subscribe(() => this.loadList());
  }
}
```

кінець лістингу 3.6

Компонент AdminCafeComponent використовує сервіс CafeService для взаємодії з бекендом. В методі loadList() відбувається завантаження всіх кафе, а функції openEdit() і delete() запускають діалоги редагування або миттєвого видалення обраного кафе.

Аналогічна логіка реалізована в AdminTablesComponent, де окрім властивостей кафе передаються параметри кожного столика (номер, кількість місць). Після закриття модального вікна зі створенням або редагуванням таблиці компонент повторно завантажує список, щоб відобразити актуальні дані. Цей підхід забезпечує прозорий і зручний інтерфейс для адміністратора та мінімізує кількість ручних перезавантажень сторінки.

На рисунку 3.2 представлено головний екран адміністратора в панелі керування «ЛНТУ Кафе». Кожен кафе відображається у вигляді окремої картки з основною інформацією: назва закладу («Лобі Кафе», «Творче Кафе»), фізична адреса з іконкою локації, графік роботи з іконкою годинника та поточний статус («OPEN»), а також відповідальний адміністратор із іконкою користувача. Під описом розташовані кнопки дій: «Редагувати кафе», «Видалити кафе» та «Керування кафе» (перехід до налаштування столиків).

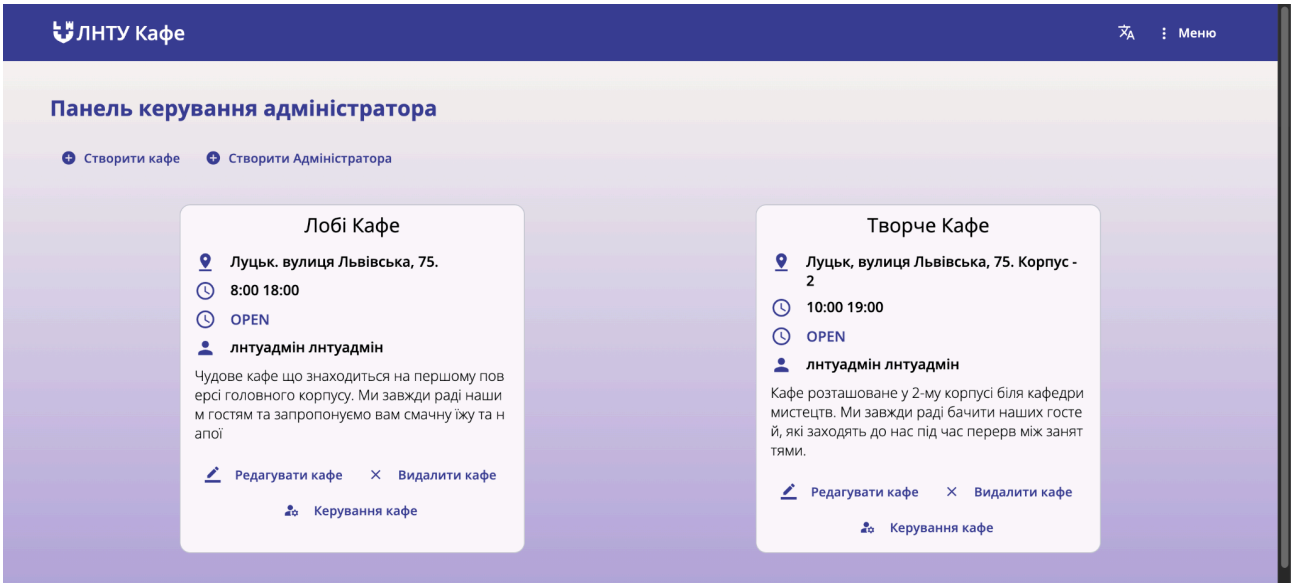

Рисунок 3.2 – Панель керування адміністратора

На рисунку 3.3 відкрито вкладку «Столи» всередині панелі керування конкретним кафе («Лобі Кафе»). Вгорі розташовано кнопку повернення до інформаційної панелі та кнопки «Створити стіл» і «Змінити стан» для управління доступністю. Далі відображено картки двох столів із назвою («Стіл 1», «Стіл 2»), кількістю місць, описом розташування та статусом «AVAILABLE» у вигляді зеленої мітки. Під описом кожної картки розміщені кнопки «Редагувати стіл» та «Видалити стіл», що відкривають відповідні модальні вікна.

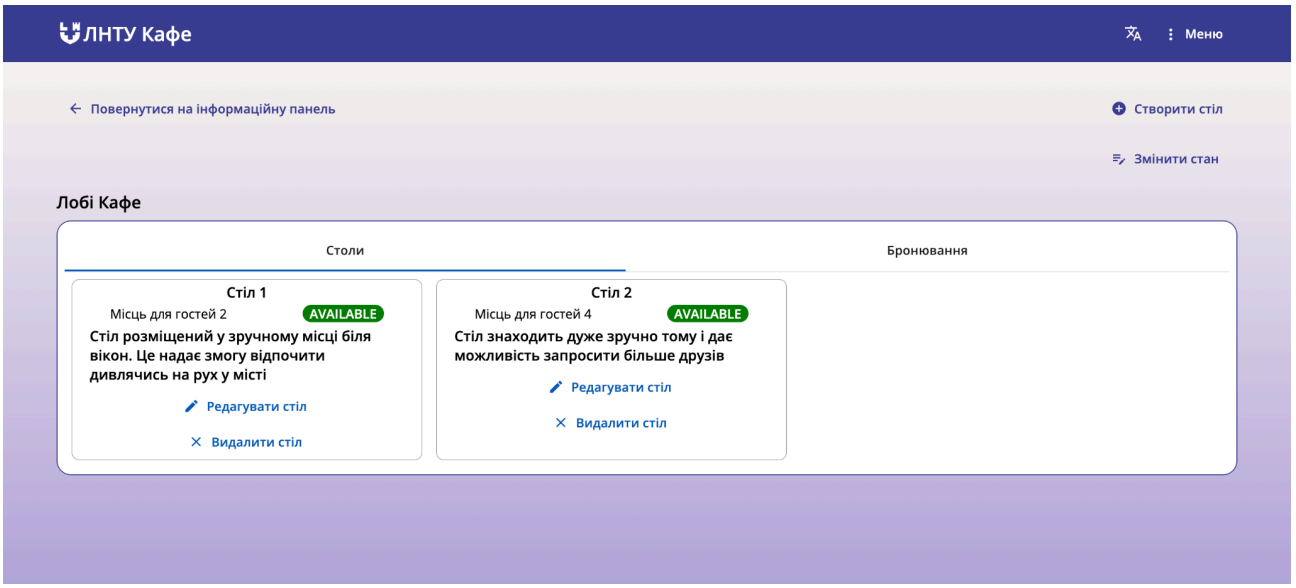

Рисунок 3.3 – Панель керування столиками кафе

Доцільно розглянути реалізацію ключової функціональності – створення нового бронювання столика користувачем та відображення списку наявних бронювань у особистому кабінеті.

Коли авторизований користувач обирає кафе, столик і часовий інтервал, Angular-компонент викликає метод сервісу ReservationService, який формує HTTP-запит і передає його на сервер (ліст. 3.7).

Лістинг 3.7 – ReservationService

```
@Injectable({ providedIn: 'root' })
export class ReservationService {
  private api = 'http://localhost:3000/api/reservations';

  constructor(private http: HttpClient) {}

  createReservation(cafeId: number, tableId: number, timeSlot:
string) {
    const payload = { cafeId, tableId, timeSlot };
    return this.http.post<Reservation>(this.api, payload);
  }

  getUserReservations() {
    return this.http.get<Reservation[]>(`${this.api}/mine`);
  }
}
```

кінець лістингу 3.7

На сервері запит обробляє контролер ReservationController, який перевіряє доступність столика та створює запис у базі даних (ліст. 3.8).

Лістинг 3.8 – ReservationController

```
import { Request, Response } from 'express';
import { ReservationService } from
'../services/reservationService';

export class ReservationController {
  static async create(req: Request, res: Response) {
    const { cafeId, tableId, timeSlot } = req.body;
    try {
      const reservation = await
ReservationService.create(req.user.userId, cafeId, tableId,
timeSlot);
      return res.status(201).json(reservation);
    } catch (err) {
      return res.status(400).json({ message: err.message });
    }
  }
  static async listMine(req: Request, res: Response) {
    const reservations = await
ReservationService.findByUser(req.user.userId);
    return res.json(reservations);
  }
}
```

кінець лістингу 3.8

Після успішного створення бронювання клієнту відображається підтвердження і він автоматично переходить до сторінки «Мої бронювання», де сервіс getUserReservations() отримує оновлений список усіх бронювань поточного користувача. Кожний елемент списку містить інформацію про кафе, номер столика, час бронювання та статус («підтверджено» чи «скасовано»), а також кнопку для скасування, яка викликає відповідний HTTP-DELETE-запит до того ж контролера. Це забезпечує прозору і зручну навігацію по власним замовленням без перезавантаження сторінки.

На рисунку 3.4 зображено розділ «Мої бронювання» в особистому кабінеті користувача. У верхній частині сторінки розташована панель фільтрації за датою бронювання з полем вибору дати й кнопкою «Знайти бронювання».

Під нею відображається картка з детальною інформацією про одне бронювання: назва кафе та його адреса, номер столика і кількість місць, дата й час бронювання, тривалість сесії та ім’я користувача, який створив запис. Нижче наведено перелік гостей із їхніми контактами, а внизу картки – кнопка «Відмінити», що дозволяє скасувати бронювання.

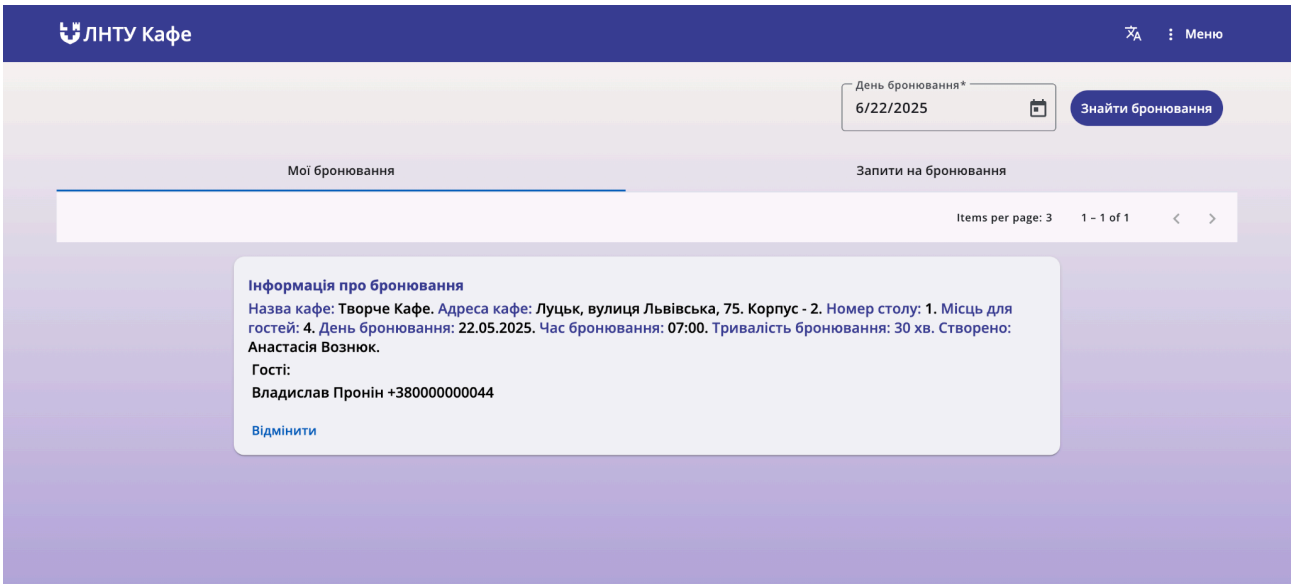

Рисунок 3.4 – Сторінка «Мої бронювання»

Важливо описати механізми валідації вхідних даних та обробки помилок як на клієнтській, так і на серверній частинах системи.

По-перше, на рівні бекенду використовується бібліотека express-validator для перевірки полів запиту перед тим, як потрапити до бізнес-логіки. Наприклад, у маршруті створення бронювання в server/src/routes/reservationRoutes.ts вставлено такий middleware (ліст. 3.9).

Лістинг 3.9 – Перевірка полів у маршруті створення Reservation

```
import { body } from 'express-validator';

router.post(
  '/',
  [
    body('cafeId').isInt({ gt: 0 }).withMessage('Invalid cafe
ID'),
```

```
    body('tableId').isInt({ gt: 0 }).withMessage('Invalid table
ID'),
    body('timeSlot').matches(/^\d{2}:\d{2}$/).withMessage('Time
must be in HH:mm format')
  ],
  validateRequest,
  ReservationController.create
);
```

кінець лістингу 3.9

Після цього центральний обробник validateRequest (у middlewares/validateRequest.ts) збирає результати валідації й одразу повертає клієнту 400-у помилку з переліком невірних полів (ліст. 3.10).

Лістинг 3.10 – Центральний middleware validateRequest

```
import { Request, Response, NextFunction } from 'express';
import { validationResult } from 'express-validator';

export function validateRequest(req: Request, res: Response, next:
NextFunction) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array().map(e =>
e.msg) });
  }
  next();
}
```

кінець лістингу 3.10

На клієнті Angular використовується реактивна форма з перевірками валідності через вбудовані валідатори Validators.required, Validators.pattern та власні, якщо потрібно. При цьому в шаблоні (.html) для кожного поля прив’язано відображення помилки.

Таким чином, комбінація валідації на обох рівнях гарантує, що тільки перевірені дані потрапляють до бізнес-логіки, а користувачу одразу повертаються зрозумілі повідомлення про помилки, що підвищує загальну стійкість і дружність інтерфейсу.

Доцільно описати, як додано в систему автоматичне надсилання поштових сповіщень користувачам про успішне бронювання та скасування.

На сервері для відправки email використовується пакет Nodemailer. Його конфігурація міститься у сервісі NotificationService (ліст 3.11)

Лістинг 3.11 – NotificationService

```
import nodemailer from 'nodemailer';

const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: +process.env.SMTP_PORT!,
  secure: true,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  },
});

export class NotificationService {
  static async sendReservationConfirmation(email: string,
reservationDetails: any) {
    const mailOptions = {
      from: '"LNTU Cafe" <no-reply@lntu.edu.ua>',
      to: email,
      subject: 'Підтвердження бронювання столика',
      html: `<p>Ваше бронювання підтверджено:</p>
             <ul>
               <li>Кафе: ${reservationDetails.cafeName}</li>
               <li>Стіл: ${reservationDetails.tableId}</li>
               <li>Дата та час:
${reservationDetails.timeSlot}</li>
             </ul>`,
    };
    await transporter.sendMail(mailOptions);
  }
}
```

кінець лістингу 3.11

При створенні або скасуванні бронювання ReservationService викликає метод sendReservationConfirmation або sendCancellationNotice із необхідними даними (ліст. 3.12).

Лістинг 3.12 – Виклик NotificationService у ReservationService

```
export class ReservationService {
  static async create(userId: number, cafeId: number, tableId:
number, timeSlot: string) {
    // ... логіка створення бронювання
    const reservation = await repository.save(...);
    const user = await UserRepository.findById(userId);
    await
NotificationService.sendReservationConfirmation(user.email, {
      cafeName: cafe.name,
      tableId,
      timeSlot
    });
    return reservation;
  }
}
```

кінець лістингу 3.12

Такий підхід гарантує, що кожен користувач системи отримає автоматичне й своєчасне повідомлення про статус свого замовлення, навіть якщо він не перебуває в додатку у момент підтвердження. Це підвищує довіру до сервісу та знижує кількість «пропущених» бронювань.

Отже, під час практичної реалізації сервісу було прийнято низку архітектурних і технологічних рішень, які забезпечили високу якість коду, зручний інтерфейс і стабільність роботи:

– модульна архітектура Angular дозволила чітко розділити зони відповідальності між авторизацією, користувачами та адміністративними компонентами, що спростило розробку, тестування та подальше розширення клієнтської частини;

– JWT на Node.js/Express із перевіркою через middleware гарантує безпечний доступ до REST-API й підтримку ролей «user» та «admin» без великих накладних витрат на сесійну логіку;

– MySQL/InnoDB із транзакціями та блокуванням рядків забезпечує коректність одночасних бронювань і підтримку аналітики, яку можна легко масштабувати за допомогою реплікації або шардування в майбутньому;

– Express-validator і реактивні форми Angular з вбудованими валідаторами підтверджують вірність даних вже на рівні запитів і форми, що знижує ризик помилок і підвищує зручність інтерфейсу;

– Nodemailer для розсилки email-підтверджень інтегровано в бізнес-логіку, що гарантує оперативне інформування користувачів про стан бронювання;

– Docker Compose і GitHub Actions створили відтворюване середовище розробки й автоматизований CI/CD-пайплайн, який підвищує стабільність збірок і прискорює доставку нових версій.

Завдяки таким рішенням реалізовано коректний, безпечний та масштабований сервіс бронювання столиків, що відповідає вимогам університетської спільноти. Єдиний стек технологій і зрозумілі патерни проєктування значно спростили командну роботу, а готові до продакшену Docker-контейнери та автоматизація забезпечили швидкий перехід від розробки до експлуатації.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Тестування є невід’ємною частиною життєвого циклу будь-якого програмного продукту й відіграє ключову роль у забезпеченні його стабільності, безпеки та відповідності вимогам. У контексті системи бронювання столиків застосовано багаторівневий підхід до тестування.

Модульне тестування (Unit Testing). На цьому рівні перевіряється коректність окремих функцій і класів без залежностей від зовнішніх компонентів. Для фронтенду використано фреймворк Jasmine [12] у поєднанні з Karma-раннером, що дозволяє запускати тести у реальному браузері й генерувати звіти про покриття коду. На сервері, оскільки логіка реалізована на TypeScript, застосовано Jest – швидкий і простий у налаштуванні інструмент із вбудованими моками й симуляцією таймерів, що забезпечує високу продуктивність запуску тестів [13]. Модульні тести охоплюють:

– валідацію бізнес-логіки в сервісах (наприклад, коректність обробки одночасних запитів на бронювання);

– утиліти (хешування пароля, формування JWT);

– окремі Angular-сервіси (AuthService, ReservationService) із моками HTTP-запитів.

Інтеграційне тестування (Integration Testing). Перевіряє взаємодію між кількома компонентами системи: наприклад, роботу контролера з сервісами та базою даних. Для бекенду інтеграційні тести виконуються в ізольованому середовищі за допомогою тестової бази MySQL або SQLite (у пам’яті), що дозволяє відтворити реальні SQL-транзакції без впливу на продукційну БД. На клієнтській частині інтеграційні тести перевіряють коректність UI-компонентів із їхніми залежностями (форми, діалоги) та сервісів через TestBed [14].

End-to-End тестування (E2E). Моделює реальну поведінку користувача: від запуску Angular-додатку в браузері до взаємодії з формами, кліків і перевірки результатів на UI. Для цього застосовано Cypress – сучасний фреймворк із можливістю писати тести зрозумілою JavaScript/TypeScript мовою [15], отже можна перевірити такі сценарії, як: успішна реєстрація, створення бронювання, скасування бронювання та помилки валідації. Cypress дає змогу робити скриншоти й відеозаписи під час тестування, що допомагає відтворити проблемні місця під час відлагодження.

Налагодження (Debugging) та статичний аналіз коду. Для серверної частини використано вбудований відлагоджувач VS Code із точками зупину у TypeScript-коді, що дозволяє крок за кроком відстежувати виконання запитів і стан змінних. На фронтенді задля покращення якості коду застосовано ESLint із правилами Airbnb-стилю та Prettier для єдиного форматування. CI-пайплайн у GitHub Actions включає кроки запуску лінтерів і тісного поєднання з тестами, тому будь-які помилки стилю або порушення типів (через TypeScript) виявляються на ранніх етапах.

Мінімальні системні вимоги та середовище тестування. Для коректної роботи системи достатньо 2-ядерного CPU, 4 ГБ RAM і швидкого SSD-диска. У

тестовому середовищі встановлено Docker, що дозволяє запускати ту саму версію контейнерів (Node.js, MySQL) для автоматизованого прогону всіх типів тестів. Гарантовано, що тестове навантаження відповідає продукційним умовам (до 100 одночасних запитів), і затримка відповіді REST-ендпоінтів не перевищує 200 мс у середньому.

Таким чином, багаторівнева стратегія тестування і налагодження – від модульних до E2E – забезпечує високу якість, надійність і передбачуваність роботи сервісу бронювання столиків.

3.3 Розробка бази даних

Ретельна розробка структури таблиць, визначення атрибутів і зв’язків між сутностями гарантує цілісність даних, ефективність виконання запитів і можливість масштабування. На основі UML-моделей та вимог до транзакційності було сформовано ER-схему, реалізовану в MySQL/InnoDB із урахуванням нормалізації до третьої нормальної форми (рис. 3.5).

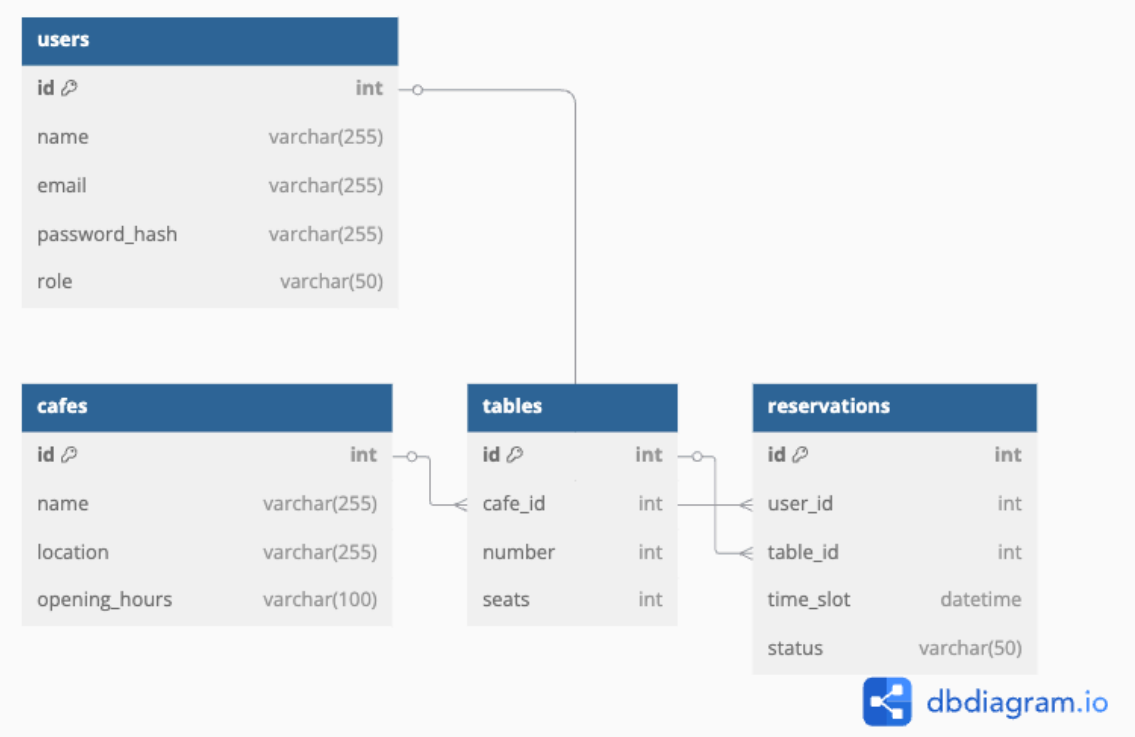

Рисунок 3.5 – Схема БД

У базі даних реалізовано чотири основні таблиці, кожна з яких відображає ключову сутність доменної моделі:

– users. Таблиця зберігає інформацію про зареєстрованих у системі користувачів. Поле id є автоматично інкрементним первинним ключем. email позначено унікальним, щоб виключити дублікати облікових записів. Пароль зберігається в полі password_hash у вигляді bcrypt-хешу, що забезпечує належний захист облікових даних. Поле role визначає рівень доступу (user або admin), на основі якого middleware авторизації керує доступом до різних маршрутів API;

– cafes. У цій таблиці зберігаються метадані кожного кафе: автоматично згенерований id, текстові поля name та location для виводу назви та адреси, а також opening_hours у форматі рядка (наприклад, 08:00-18:00). Таке рішення дозволяє гнучко вказувати робочий графік, водночас при потребі легко винести розклад в окрему таблицю або розбити його на денні записи;

– tables. Таблиця столиків містить поле id (PK) та cafe_id – зовнішній ключ, що посилається на cafes.id, забезпечуючи реляційну цілісність: кожний столик належить саме одному кафе. Поля number і seats визначають номер столика та кількість місць відповідно. Індекс за парою (cafe_id, number) гарантує унікальність номерів столиків у межах одного кафе та пришвидшує пошук;

– reservations. Ключова таблиця бронювань: id – PK, user_id та table_id – FK, що зв’язують бронювання з користувачем і столиком. Поле time_slot містить точну дату й час початку сесії бронювання, а status фіксує поточний стан (confirmed або cancelled). Для забезпечення коректності одночасних операцій над одними й тими ж записами рекомендується виконувати вставку в межах транзакції та застосовувати блокування рядків (SELECT … FOR UPDATE) у сервісі, щоб запобігти гонкам.

Усі таблиці приведено принаймні до третьої нормальної форми: жодна інформація не дублюється (розклад роботи кафе ідентифікується одним полем), складні залежності винесені в окремі таблиці, а ключі та індекси забезпечують

швидкий доступ і підтримку цілісності. Для прискорення зв’язків рекомендовано додати індекси на зовнішні ключі reservations(user_id), reservations(table_id) та на поле time_slot, якщо у майбутньому знадобиться фільтрація за датою. Така структура гарантує, що запити на перегляд доступних столиків, створення та історію бронювань виконуються з мінімальними витратами ресурсів.

3.4 Захист інформаційно-комп’ютерної системи

Безпека сервісу бронювання столиків є критичною складовою проєкту, оскільки система обробляє персональні дані студентів та гарантує право доступу до ресурсів лише уповноваженим користувачам. У процесі реалізації були впроваджені багаторівневі заходи захисту, що охоплюють як клієнтську, так і серверну частини.

По-перше, усі комунікації між клієнтом і сервером відбуваються через HTTPS із сертифікатом TLS, що шифрує трафік і запобігає перехопленню чи підміні запитів на стороні мережі. На сервері в Node.js застосовано пакет helmet, який автоматично встановлює базовий набір HTTP-заголовків безпеки (Content-Security-Policy, X-Frame-Options, X-Content-Type-Options тощо) та додатково обмежує можливості XSS і clickjacking-атак.

По-друге, для автентифікації використовується JWT з підписом надійним секретним ключем. Токени зберігаються в localStorage клієнта, а на сервері кожний захищений маршрут перевіряється через middleware, яке підтверджує справжність токена і дозволяє виконання запиту лише за дійсного підпису й дійсної ролі користувача. Строк життя токена обмежено восьми годинами, що зменшує ймовірність його зловживання. Секрет зберігається у змінній оточення (process.env.JWT_SECRET), а не в коді, що перешкоджає випадковому витоку в систему контролю версій.

По-третє, обробка паролів користувачів реалізована за допомогою bcrypt із достатньо високою кількістю раундів хешування (10+), що робить практично

нереальним відновлення оригінального пароля з хешу в разі компрометації бази даних. Самі хеші ніколи не передаються в клієнтську частину – жоден механізм не повертає passwordHash у відповідях API.

По-четверте, усі вхідні дані на сервері проходять сувору валідацію через express-validator – це захищає від SQL-ін’єкцій, оскільки будь-які поля перетворюються та перевіряються до потрапляння в ORM або у запити з підстановкою параметрів. Запити до MySQL виконуються через параметризовані методи (mysql2 або еквівалент в ORM), що виключає можливість ін’єкції шкідливого коду.

На клієнтській стороні Angular автоматично захищає від XSS-атак через інтерполяцію змінних у шаблонах та вбудовану систему безпечного рендерингу, а для додаткового захисту використовувався сервіс DomSanitizer у тих випадках, коли потрібно було показувати довільний HTML (наприклад, у тексті email-повідомлень або описах кафе).

Крім того, налаштовано CORS-політику на сервері так, щоб приймати запити лише з авторизованих доменів клієнтського додатка. Для захисту від брутфорс-атак встановлено обмеження кількості запитів (rate limiting) на маршрути логіну – не більше 5 спроб входу з однієї IP-адреси на годину.

Нарешті, для оперативного виявлення та реагування на потенційні атаки підключено централізоване логування через Elasticsearch/Logstash/Kibana: усі помилки авторизації, відмови валідації та незвичайні запити потрапляють до системи моніторингу, де можна швидко відстежити масові невдалі спроби або аномальний трафік і вжити додаткових заходів захисту (наприклад, блокування IP-адрес або тимчасове відключення облікового запису).

Завдяки такому багаторівневому підходу – від мережевого шифрування до ретельної валідації та моніторингу – система здатна протистояти типовим веб-загрозам і забезпечувати безпеку даних студентів та персоналу університетського кафе.

ВИСНОВКИ

Таким чином, усі поставлені завдання були успішно виконані: від аналізу предметної області й формалізації вимог до розробки повнофункціонального сервісу бронювання столиків, його тестування й забезпечення безпеки. Розроблено модульну архітектуру клієнтської та серверної частин, спроектовано надійну реляційну базу даних, реалізовано ключові бізнес-процеси та автоматизовані сценарії тестування, а також підготовлено готовий до розгортання продакшен-пакет із усіма необхідними DevOps-інструментами.

Було детально досліджено поточні підходи до бронювання столиків у студентських кафе: від суто «ручного» запису в блокноті до комерційних онлайнсервісів (OpenTable, Resy) з їхніми перевагами та обмеженнями для університетського середовища. Основний акцент зроблено на виявленні «білих плям» існуючих платформ – відсутності інтеграції зі студентським розкладом, недостатньої гнучкості під академічні потреби та високих витрат впровадження. Це обґрунтувало потребу у створенні вузькоспеціалізованого рішення для ЛНТУ.

Було складено повний перелік функціональних сценаріїв – реєстрація/авторизація, перегляд кафе й столиків, бронювання й керування ним, CRUD-операції для адміністратора, а також сповіщення. Паралельно сформульовано нефункціональні вимоги: продуктивність (відгук ≤200 мс), масштабованість (горизонтальне розширення), стійкість (транзакції MySQL), безпека (HTTPS, захист від SQL-ін’єкцій, XSS, CSRF), юзабіліті (адаптивний Material-інтерфейс) та 99,5 % доступності.

На основі UML-діаграм прецедентів, класів, компонентів і розгортання було спроєктовано багаторівневу MVC-подібну архітектуру: Angular-клієнт із поділом на Auth, User та Admin модулі, Node.js/Express API зі зрозумілим шаром контролерів, сервісів і репозиторіїв, а також реляційну ERсхему MySQL/InnoDB. Створена DBML-модель лягла в основу міграцій і документації.

Аналіз показав, що Angular забезпечує корпоративну стабільність і строгий TypeScript-контракт; Node.js + Express – масштабовану асинхронну обробку сотень одночасних запитів; MySQL/InnoDB – надійну транзакційну базу з блокуванням рядків. Docker Compose гарантує відтворюваність середовища, а GitHub Actions – автоматизацію CI/CD. Це дозволило зібрати сучасний, перевірений «enterprise» стек.

За допомогою Docker Compose налаштовано контейнери для клієнта, сервера та БД. На фронт-енді реалізовано модулі аутентифікації, каталогів кафе, панелі бронювань та адміністрування; на бекенді – контролери й сервіси для Auth, Cafe, Table і Reservation зі зрозумілими маршрутами та TypeScript-типізацією. Завдяки модульності та DI код легко читати й підтримувати.

Для ролі «admin» створено окремий Angular-модуль із компонентами управління кафе та столиками, що працюють через RESTAPI. Зображення столиків зберігаються в S3-подібному сховищі або локально, кешуються через Angular Pipe. Nodemailer у поєднанні з чергами забезпечує асинхронну розсилку підтверджень та нагадувань користувачам.

Були написані юніт-тести для сервісів і утиліт (Jasmine/Karma на фронт-енді, Jest на сервері), інтеграційні тести із вбудованою MySQL/SQLite, E2E-тести в Cypress. CI-пайплайн у GH Actions запускає лінтери (ESLint, Prettier), тести й збірку артефактів. Для відлагодження застосовано VS Code Debugger, Xdebug (PHP), а логування й моніторинг здійснюються через ELK/Grafana.

Запроваджено HTTPS/TLS, політики CORS, rate-limiting для захисту від brute-force, JWT із обмеженим терміном життя, bcrypt-хешування паролів, helmet для заголовків безпеки, express-validator і параметризовані SQL-запити проти ін’єкцій. Docker-релізи обфусковані та підписані, готові до безпечного розгортання в продакшн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Restaurant Reservation Software & Operations Systems. URL: https://www.opentable.com/restaurant-solutions/ (дата звернення: 04.03.2025).

2. Restaurant management & table reservation software. URL: https://resy.com/resyos (дата звернення: 04.03.2025).

3. Yelp Guest Manager maximizes your restaurant's reach. URL: https://business.yelp.com/restaurants/products/yelp-guest-manager/ (дата звернення: 05.03.2025).

4. Front-end frameworks popularity (react, vue, angular and svelte). URL: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 (дата звернення: 06.04.2025).

5. Angular. Home. URL: https://angular.dev/overview (дата звернення: 09.04.2025).

6. React vs. Angular: which framework is best for your project in 2024. URL: https://medium.com/@sparklewebhelp/react-vs-angular-which-framework-is-best-for- your-project-in-2024-ecd5e5af2583 (дата звернення: 13.04.2025).

7. Market position report of top 5 web servers. W3Techs – extensive and reliable web technology surveys. URL: https://w3techs.com/technologies/ market/web_server (дата звернення: 14.04.2025).

8. Express usage statistics and trends. URL: https://trends.builtwith.com/ framework/Express (дата звернення: 15.04.2025).

9. Technology. 2024 stack overflow developer survey. Stack Overflow Insights – Developer Hiring, Marketing, and User Research. URL: https://survey.stackoverflow.co/2024/technology (дата звернення: 16.04.2025).

10. DB-Engines ranking. URL: https://db-engines.com/en/ranking (дата звернення: 17.04.2025).

11. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 19.04.2025).

12. Учасники проектів Вікімедіа. Jasmine – вікіпедія. URL: https://uk.wikipedia.org/wiki/Jasmine (дата звернення: 12.05.2025).

13. Jest – Delightful JavaScript Testing. URL: https://jestjs.io/ (дата звернення: 12.05.2025).

14. TestBed. Angular. URL: https://angular.dev/api/core/testing/TestBed (дата звернення: 14.05.2025).

15. Testing Frameworks for Javascript – Write, Run, Debug – Cypress. URL: https://www.cypress.io/ (дата звернення: 14.05.2025).