

БАКАЛАВРСЬКА РОБОТА
БР.ІІ-02.00.00.000 ІЗ

Група ІІ-22-1

Прудиус Наталія

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Прудиус Наталія Ігорівна

(прізвище, ім'я, по батькові)

УДК 004.42
(індекс)

БАКАЛАВРСЬКА РОБОТА

**«Розробка клієнт-орієнтованого веб-застосунку для автоматизації роботи
ресторану»**

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Прудиус Н.І.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Дмитрик Тарас Богданович, асистент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

м. Івано-Франківськ - 2026 р.

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІІЗ

доц. В.В. Бандура

“ ” 2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Прудиус Наталії Ігорівні

(прізвище, ім'я, по-батькові)

1. **Тема проекту (роботи)** «Розробка клієнт-орієнтованого веб-застосунку для автоматизації роботи ресторану»
керівник проекту (роботи) асист. Дмитрик Тарас Богданович
затверджені наказом закладу вищої освіти від “21” квітня 2026 р. № 179/7
2. **Строк подання студентом проекту (роботи)** 10 червня 2026 р.
3. Вихідні дані до проекту (роботи) програмний код веб-застосунку Crescendo, локальне Docker-середовище, конфігурація Nginx, Node.js backend, REST JSON API для столів і бронювань, JSON-файл бази даних у Docker volume, статичні дані меню, модель карти залу, seed-дані бронювань і технічний звіт актуальної Docker-версії сайту.
4. **Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)**
 1. Аналіз предметної області автоматизації ресторанного сервісу та онлайн-бронювання
 2. Формування функціональних і нефункціональних вимог до веб-застосунку Crescendo
 3. Проектування архітектури, моделі даних, REST API та сценаріїв бронювання
 4. Реалізація клієнтської частини, карти залу, меню, бронювання та адміністративної панелі
 5. Тестування, оцінка результатів і опис Docker-розгортання через Nginx і Node.js backend
5. **Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)**
 1. UML-діаграма випадків використання Crescendo (рис. 2.1, ст. 31)
 2. Архітектура веб-застосунку Crescendo (рис. 3.1, ст. 39)
 3. Activity-схема бізнес-процесу онлайн-бронювання (рис. 3.2, ст. 41)
 4. Узагальнена модель даних бронювань, столів і гостей (рис. 3.3, ст. 43)
 5. Схема Docker-розгортання React/Nginx і Node.js backend (рис. 5.1, ст. 67)

6. Консультанти розділів проекту (роботи)

<i>Розділ</i>	<i>Консультант</i>	<i>Підпис, дата</i>	
		<i>Завдання видав</i>	<i>Завдання прийняв</i>

7. Дата видачі завдання _____

Керівник

(підпис)

Дмитрик Т. Б.
(розшифровка підпису)

Завдання прийняв до виконання

(підпис)

Прудіус Н. І.
(розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

<i>№ з/п</i>	<i>Назва етапів дипломного проекту (роботи)</i>	<i>Строк виконання етапів проекту</i>	<i>Примітка</i>
1	Опрацювання теоретичних матеріалів з теми роботи	27.02.2026	виконано
2	Вибір інструментів, технологій та методів розробки	05.03.2026	виконано
3	Налаштування середовища та конфігурація бази даних	23.03.2026	виконано
4	Розробка компонентів та функціональних модулів системи	13.04.2026	виконано
5	Тестування системи та усунення виявлених помилок	23.04.2026	виконано
6	Підготовка та оформлення бакалаврської роботи	29.05.2026	виконано

Студент – дипломник

(підпис) (розшифровка підпису)

Керівник роботи

(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 89 сторінок, 31 рисунок, список використаних джерел із 32 найменувань, 2 додатки.

Метою роботи є розроблення веб-застосунку Crescendo для автоматизації ресторанного сервісу та онлайн-бронювання.

Об'єкт дослідження — процес організації ресторанного обслуговування та приймання бронювань засобами веб-технологій.

Предмет дослідження — методи, моделі та засоби проектування клієнт-серверного веб-застосунку для автоматизації бронювання та управління ресторанним сервісом.

Результати дослідження: створено веб-застосунок Crescendo з використанням React, Material UI Node.js, Docker та Nginx. Реалізовано перегляд меню, онлайн-бронювання столиків, керування резервуваннями та адміністративну панель.

У першому розділі проведено аналіз предметної області.

У другому розділі сформульовано завдання проектування, функціональні та нефункціональні вимоги до системи, межі застосунку.

У третьому розділі описано проектування системи, архітектуру застосунку, структуру REST API та логіку бронювання столиків .

У четвертому розділі описано реалізацію веб-застосунку.

У п'ятому розділі описано розгортання системи та перспективи її розвитку.

Висновок: розроблений веб-застосунок Crescendo автоматизує основні процеси ресторанного сервісу та онлайн-бронювання і може бути використаний як прототип цифрової системи управління рестораном.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, РЕСТОРАН, ОНЛАЙН-БРОНЮВАННЯ, REACT, MATERIAL UI, REACT QUERY, AXIOS, NODE.JS, REST API, DOCKER, NGINX, АДМІНІСТРАТИВНА ПАНЕЛЬ, КАРТА ЗАЛУ, LIQPAY.

ABSTRACT

The bachelor's thesis contains 89 pages, 31 figures, a list of references with 33 sources, and 2 appendices.

The aim of the thesis is to develop the Crescendo web application for restaurant service automation and online table reservation.

The object of the research is the process of organizing restaurant service and handling reservations using web technologies.

The subject of the research is the methods, models, and tools for designing a client-server web application for reservation automation and restaurant service management.

Research results: a functional Crescendo web application was created using React, Material UI, Node.js, Docker, and Nginx. The system provides menu browsing, online table reservation, reservation management, and an administrative panel.

The first chapter analyzes the subject area of restaurant automation.

The second chapter formulates the design objectives, functional and non-functional requirements of the system, and the application boundaries.

The third chapter describes the system design, application architecture, REST API structure, and table reservation logic.

The fourth chapter presents the implementation of the web application.

The fifth chapter describes system deployment and prospects for further development.

Conclusion: the developed Crescendo web application automates the main processes of restaurant service and online reservation and can be used as a prototype of a digital restaurant management system.

KEYWORDS: WEB APPLICATION, RESTAURANT, ONLINE RESERVATION, REACT, MATERIAL UI, REACT QUERY, AXIOS, NODE.JS, REST API, DOCKER, NGINX, ADMIN PANEL, HALL MAP, LIQPAY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	9
ВСТУП.....	10
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЧНИХ ТЕНДЕНЦІЙ	13
1.1 Характеристика предметної області автоматизації ресторанного сервісу..	13
1.2 Основні поняття та терміни онлайн-бронювання і ресторанних веб-систем	16
1.3 Аналіз існуючих рішень для ресторанного бронювання та керування залом	20
1.4 Висновки по розділу	23
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЄКТУВАННЯ	25
2.1 Збір та аналіз вимог зацікавлених сторін	25
2.2 Функціональні вимоги до веб-застосунку Crescendo	28
2.3 Нефункціональні вимоги та критерії приймання	34
2.4 Висновки по розділу	37
РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ, МОДЕЛІ ДАНИХ ТА API	39
3.1 Обґрунтування архітектурної моделі системи	39
3.2 Моделювання бізнес-процесів і користувацьких сценаріїв.....	44
3.3 Проєктування моделі даних та REST API	49
3.4 Висновки по розділу	55
РОЗДІЛ 4. РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ CRESCENDO.....	57
4.1 Реалізація клієнтської частини	5
4.2 Реалізація бронювання, карти залу та адміністративної підсистеми	61
4.3 Організація програмного коду, алгоритми та структури даних.....	68
4.4 Висновки по розділу	72

РОЗДІЛ 5. СИСТЕМНЕ ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ.....	74
5.1 Стратегія та методика тестування системи	74
5.2 Результати функціонального, інтеграційного та frontend/backend-тестування	77
5.3 Розгортання через Docker, Nginx і Node.js backend.....	80
5.4 Висновки по розділу	82
ВИСНОВКИ	85
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	87
ДОДАТКИ	
БІБЛІОГРАФІЧНА ДОВІДКА	

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API — Application Programming Interface, програмний інтерфейс взаємодії між компонентами.

CRA — Create React App, інструмент створення та збірки React-застосунків.

CRUD — Create, Read, Update, Delete, базові операції з даними.

SPA — Single Page Application, односторінковий веб-застосунок.

REST — Representational State Transfer, архітектурний стиль взаємодії через HTTP.

UI — User Interface, користувацький інтерфейс.

UX — User Experience, користувацький досвід.

JSON — JavaScript Object Notation, формат обміну структурованими даними.

MUI — Material UI, бібліотека компонентів інтерфейсу для React.

CI/CD — Continuous Integration / Continuous Delivery, автоматизована інтеграція та доставка змін.

MVP — Minimum Viable Product, мінімально життєздатний продукт.

Nginx — веб-сервер і reverse-proxy, використаний для роздавання production-збірки та прокидання /api-запитів до backend.

Node.js — середовище виконання JavaScript на сервері, використане для реалізації локального backend.

CORS — Cross-Origin Resource Sharing, механізм керування дозволами HTTP-запитів між клієнтом і сервером.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

ВСТУП

Ресторанний бізнес належить до сфер, у яких якість сервісу залежить не тільки від кухні, а й від швидкості організації взаємодії з гостем. Гість очікує, що зможе переглянути меню, обрати зручний час, зрозуміти доступність місць і підтвердити бронювання без телефонного дзвінка. Для персоналу ресторану важливими є інші аспекти: контроль заповненості залу, підготовка столів, облік постійних гостей, знання алергій, побажань і статусу депозиту. Тому веб-застосунок для автоматизації ресторану повинен поєднувати публічний інтерфейс і внутрішній операційний інструмент.

Актуальність теми роботи зумовлена тим, що багато ресторанів використовують фрагментовані канали комунікації: телефон, соціальні мережі, месенджери, таблиці та ручні нотатки адміністратора. Такий підхід призводить до дублювання даних, ризику помилок у часі бронювання, невчасного скасування, втрати інформації про гостя або неправильного розподілу столів. Розроблений застосунок Crescendo демонструє, як ці задачі можуть бути об'єднані в єдиному веб-інтерфейсі.

Особливістю ресторанного бронювання є необхідність одночасно враховувати часовий слот, кількість гостей, місткість столика, поверх, зону залу, статус зайнятості та політику оплати. На відміну від звичайної форми зворотного зв'язку, бронювання змінює операційний стан ресторану. Саме тому система повинна не просто збирати контактні дані, а виконувати перевірку доступності, блокувати вже зайняті столи, відображати результат у списку бронювань і передавати ці дані адміністративній панелі.

Об'єктом дослідження є процес автоматизації приймання бронювань і підготовки ресторанного залу до обслуговування гостей.

Предметом дослідження є програмні засоби, архітектурні рішення та інтерфейсні моделі, що забезпечують роботу ресторанного веб-застосунку з меню, картою залу, бронюванням, адміністративним кабінетом і REST-взаємодією.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

Метою бакалаврської роботи є розроблення веб-застосунку Crescendo для автоматизації роботи ресторану з інтегрованою системою онлайн-бронювання. Для досягнення мети потрібно проаналізувати предметну область, сформулювати вимоги, спроектувати структуру застосунку, реалізувати клієнтську частину, описати роботу з даними, розгорнути застосунок у Docker-середовищі та оцінити результати.

Для досягнення поставленої мети у роботі вирішуються такі завдання: проаналізувати існуючі підходи до онлайн-бронювання ресторанів; визначити ролі гостей, адміністратора та зовнішніх сервісів; описати функціональні й нефункціональні вимоги; побудувати UML-діаграму випадків використання; спроектувати архітектуру React SPA; реалізувати форму бронювання, карту залу, меню, перегляд замовлень і admin dashboard; перевірити працездатність Docker-розгортання.

У процесі виконання роботи використано методи системного аналізу, моделювання бізнес-процесів, структурного проектування, компонентного підходу до розробки інтерфейсу, функціонального тестування та аналізу ризиків. Такий набір методів дає змогу розглянути систему не лише як набір сторінок, а як послідовний цифровий сценарій: від наміру гостя забронювати столик до підготовки ресторанного залу персоналом. Методологічну основу оформлення й структурування роботи становлять методичні рекомендації кафедри та стандарти оформлення звітів і бібліографічних посилань [1], [2], [3]. Підхід до формування вимог і оцінювання якості узгоджено з моделями інженерії вимог та якості програмних систем [4], [5], а загальні принципи інженерії програмного забезпечення враховано за базовими працями з проектування і життєвого циклу ПЗ [6], [7].

Практична новизна роботи полягає у поєднанні інтерактивної карти залу, гнучкої форми бронювання, сценарію першого візиту з депозитом, guest-card механіки для постійних гостей і адміністративної панелі з календарем, таймлайном сервісу, профілями гостей та інформацією для кухні. Це формує основу для MVP, який може бути розширений серверною частиною, реальною

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

платіжною інтеграцією і системою повідомлень.

Крім того, запропонований веб-застосунок дозволяє централізувати управління всіма каналами комунікації, зменшити ризики дублювання даних і забезпечити своєчасне реагування персоналу на зміни у бронюваннях. Інтеграція адміністративної панелі з календарем і картою залу надає можливість оптимізувати розподіл столів, уникати конфліктів бронювань та забезпечити більш ефективну роботу ресторану під час пікових періодів.

Також реалізація MVP включає гнучку структуру для подальшого розширення функціоналу: підключення реальних платіжних систем, автоматизація повідомлень для гостей, аналітика відвідуваності та формування звітів для менеджменту. Це забезпечує потенціал для масштабування системи, підвищення якості обслуговування та впровадження додаткових сервісів, що підвищують конкурентоспроможність ресторанного закладу.

Бакалаврська робота містить 5 розділів, 89 сторінок, 31 рисунок, список використаних джерел із 32 найменувань, 2 додатки.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЧНИХ ТЕНДЕНЦІЙ

1.1. Характеристика предметної області автоматизації ресторанного сервісу

Автоматизація ресторанного сервісу охоплює сукупність процесів, пов'язаних із взаємодією закладу з гостем, організацією роботи залу, керуванням столами, меню, попередніми замовленнями, оплатою депозиту, обліком постійних клієнтів та підготовкою персоналу до обслуговування. У сучасному ресторані бронювання вже не є лише телефонним записом у журналі адміністратора. Воно стає частиною загальної інформаційної системи, де кожна дія гостя або працівника має бути зафіксована, оброблена і відображена в інтерфейсі. Саме тому автоматизація ресторану повинна поєднувати зручний публічний інтерфейс для клієнта та внутрішній робочий простір для персоналу.

У межах цієї бакалаврської роботи основна увага приділена процесам, які безпосередньо впливають на онлайн-бронювання: вибору дати й часу, визначенню кількості гостей, пошуку доступного столика, фіксації контактних даних, обробці статусу бронювання та передаванню інформації адміністратору, офіціантам і кухні. Такий підхід дозволяє розглядати ресторан не лише як місце надання послуг, а як систему з обмеженими ресурсами. Основним ресурсом у цьому випадку є посадкові місця, розподілені між столами, зонами та поверхами. Якщо ці ресурси не відображаються у зрозумілій цифровій формі, працівники можуть втрачати час на ручну перевірку доступності, а гості — стикатися з неточностями або необхідністю додатково уточнювати деталі бронювання телефоном.

Для ресторану бронювання є різновидом ресурсного планування. Кожен стіл має певну місткість, розташування у залі, прив'язку до поверху, зони та поточний статус. Наприклад, один стіл може бути розміщений біля вікна, інший — у приватній зоні, третій — на другому поверсі або біля бару. Ці

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

характеристики мають значення як для гостя, так і для адміністратора. Гість може бажати обрати комфортне місце відповідно до формату візиту, а адміністратор повинен розуміти, які столи вже зайняті, які ще доступні, які потребують підготовки і які можуть бути використані для груп різного розміру.

Якщо адміністратор не бачить структури залу в одному місці, виникає ризик накладення бронювань, неефективного використання столів, помилок у розподілі гостей та перевантаження окремих зон ресторану. Наприклад, невелику групу можна випадково посадити за великий стіл, що зменшить кількість доступних місць для інших бронювань. Або навпаки — для великої компанії може бути обраний стіл недостатньої місткості. Тому автоматизована система повинна не просто приймати заявку, а враховувати взаємозв'язок між часом, кількістю гостей, місткістю столу та його розташуванням.

У системі Crescendo карта залу є центральним елементом предметної області. Вона не лише візуалізує простір ресторану, а й виконує роль інструмента вибору доступного столика. Через карту користувач бачить структуру залу, поверхи, зони, столи, їхню місткість і статус. Такий підхід наближує онлайн-бронювання до реального досвіду відвідування ресторану, оскільки гість отримує більше контролю над вибором місця. Для працівників карта також має практичне значення, адже дозволяє швидко оцінити завантаження залу та підготувати потрібні столи до приходу гостей.

Предметна область ресторанного сервісу характеризується високою залежністю від часу. Ресторан працює не лише з датами, а й із часовими слотами, наприклад 12:00–14:00, 14:00–16:00 або 18:00–20:00. Для кожного слота доступність столів може відрізнятися. Один і той самий стіл може бути вільним у першій половині дня, але заброньованим увечері. Через це система бронювання повинна будувати логіку вибору послідовно: спочатку дата, потім час, кількість гостей і лише після цього — доступні столи. Така послідовність зменшує ймовірність помилки, тому що користувач не обирає стіл без урахування конкретного часу та кількості гостей.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

Важливим завданням інтерфейсу є зменшення когнітивного навантаження на користувача. Якщо форма бронювання містить забагато полів без логічної послідовності, користувач може помилитися або не завершити бронювання. Тому в розробленому застосунку сценарій бронювання побудований як поступовий процес. Користувач спочатку вводить контактні дані, потім обирає тип візиту, дату, час, кількість гостей і тільки після цього переходить до вибору столика. Недоступні або замалі столи мають бути заблоковані, щоб користувач не міг обрати неправильний варіант. Це робить систему не пасивною формою, а активним помічником у процесі бронювання.

Окремим аспектом предметної області є взаємодія з постійними гостями. Для ресторану важливо знати не лише факт бронювання, а й контекст попередньої взаємодії з клієнтом. До такого контексту належать алергії, смакові вподобання, частота відвідування, бажана зона залу, історія попередніх візитів, наявність гостьової картки та особливі нотатки персоналу. У прототипі ці дані представлені через Guest CRM, що дає адміністратору можливість швидко отримати інформацію про гостя і підготувати персоналізований сервіс. Наприклад, якщо гість має алергію або надає перевагу певному типу столика, ця інформація може бути врахована ще до його приходу.

Наявність Guest CRM розширює функціональність системи бронювання. Вона перетворює застосунок із простої форми резервування на інструмент роботи з клієнтським досвідом. Для ресторану це важливо, оскільки якість сервісу часто визначається не лише швидкістю обслуговування, а й здатністю персоналу враховувати індивідуальні потреби гостя. Якщо адміністратор має доступ до нотаток і вподобань, заклад може створити відчуття персональної уваги. Це особливо актуально для ресторанів преміального або авторського формату, де очікування клієнтів щодо сервісу є вищими.

Проблема депозиту також є типовою для ресторанної практики. Депозит може використовуватися для підтвердження серйозності наміру гостя, особливо під час першого візиту, бронювання великої групи, приватної вечері або популярного часового слота. У застосунку закладено бізнес-правило, за

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

яким новий гість має пройти сценарій депозитної оплати, а постійний гість із guest card може бронювати без депозиту. Це демонструє гнучкість правил системи: різні категорії користувачів можуть мати різні умови бронювання.

Окрім публічного сценарію для гостя, предметна область включає внутрішні операційні процеси ресторану. Після створення бронювання інформація повинна бути доступна персоналу: адміністратор бачить запис у календарі, офіціанти можуть переглянути час приходу гостей і номер столу, кухня може отримати попередні замовлення або сервісні нотатки. Тобто бронювання є початковою точкою ширшого процесу підготовки ресторану до обслуговування. Саме тому в Crescendo передбачено адміністративну панель, календар, timeline сервісу, нотатки про гостей і блок підготовки столів та кухні.

Таким чином, предметна область ресторанної автоматизації вимагає поєднання кількох рівнів: публічного інтерфейсу для гостя, логіки перевірки доступності столів, збереження даних про бронювання, операційного інтерфейсу для персоналу та механізмів персоналізації сервісу. Розроблений веб-застосунок орієнтований саме на цей перетин. Він показує ресторан гостю, допомагає оформити бронювання, фіксує потрібні дані та одночасно готує інформацію для працівників ресторану. Такий підхід відповідає практичним потребам закладу і демонструє, як сучасні веб-технології можуть бути використані для автоматизації реального сервісного процесу.

1.2. Основні поняття та терміни онлайн-бронювання і ресторанных веб-систем

Онлайн-бронювання — це сценарій, у якому користувач самостійно створює запит на резервування ресторанного ресурсу через веб-інтерфейс. У межах розробленої системи таким ресурсом є конкретний стіл, прив'язаний до поверху, зони, місткості й статусу доступності. На відміну від простого повідомлення через контактну форму, бронювання змінює стан системи. Після успішного створення запис має бути збережений, відображений у списку

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

бронювань, врахований у карті залу та доступний в адміністративній панелі. Тому онлайн-бронювання слід розглядати як повноцінну операцію з даними, а не лише як спосіб зв'язку з рестораном.

У ресторанній веб-системі важливим поняттям є стан столу. Стіл може бути доступним, заброньованим або недоступним для вибору через невідповідність кількості гостей. Крім цього, стіл має статичні характеристики: номер, місткість, форму, координати на карті, поверх і зону. Поєднання цих характеристик дозволяє системі правильно визначати, які столи можна показати користувачу після вибору дати, часу та кількості гостей. Якщо, наприклад, користувач вказує шість гостей, система не повинна дозволяти вибір столу на дві особи. Така логіка зменшує кількість помилок і робить процес бронювання більш контрольованим.

Карта залу — це візуальна модель простору ресторану. Вона містить столи різної форми, допоміжні зони, кухню, бар, вихід, вікна, проходи та інші елементи простору. У Crescendo карта реалізована як інтерактивний компонент, у якому координати столів визначаються даними з TableLayoutDB. Завдяки цьому візуальний шар відокремлений від логіки бронювання. Такий підхід має практичне значення: якщо потрібно змінити розташування столів, додати нову зону або уточнити координати, це можна зробити через структуру даних, не переписуючи всю логіку бронювання.

Інтерактивна карта залу відрізняється від звичайного списку столів тим, що надає користувачу просторове уявлення про ресторан. Гість бачить не лише номер столу, а й його розташування. Це важливо для ресторанів, де атмосфера, зона посадки або приватність мають значення. Для адміністратора така карта виконує функцію швидкого операційного огляду: можна оцінити, які столи зайняті, які залишаються вільними, скільки місць доступно на певному поверсі та які зони найбільше завантажені. Отже, карта залу є одночасно елементом користувацького досвіду і службовим інструментом.

REST API у роботі розглядається як спосіб отримання і зміни стану столів та бронювань. У розробленій системі API працює поверх локального

					БР.ІП - 02.00.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

backend на Node.js 20, який реалізовано як native HTTP server. Через нього клієнтська частина отримує список столів, переглядає доступні або заброньовані записи, створює нові бронювання та скасовує існуючі. Основними HTTP-методами є GET для отримання даних, POST для створення запису та PATCH для часткового оновлення статусу, reservation-даних або скасування бронювання.

Клієнтська частина використовує Axios для виконання HTTP-запитів. Axios дає змогу централізовано налаштувати базову адресу API, обробляти відповіді сервера, передавати дані у форматі JSON і реагувати на помилки. У контексті ресторанної системи це важливо, оскільки користувач не повинен бачити технічні деталі помилки. Якщо запит не виконався, інтерфейс має повідомити про проблему у зрозумілій формі. Також Axios спрощує роботу з різними endpoint-ами, наприклад отриманням списку столів, фільтрацією за статусом або оновленням конкретного запису бронювання.

Для керування серверним станом використовується React Query. У веб-застосунках є різниця між локальним станом інтерфейсу і серверними даними. Локальний стан може відповідати за відкриття вкладки, введення значення у полі або вибрану кнопку. Серверний стан — це дані, які надходять з backend: столи, бронювання, статуси, записи гостей. React Query дозволяє кешувати ці дані, повторно отримувати їх у потрібний момент і виконувати інвалідацію кешу після створення або скасування бронювання. Завдяки цьому після зміни даних інтерфейс оновлюється передбачувано, без ручного керування великою кількістю станів.

Формат JSON у роботі використовується як основний формат обміну даними між frontend і backend. JSON зручний для передавання структурованих об'єктів: столів, бронювань, замовлень, контактних даних і платіжної інформації. Наприклад, запис бронювання може містити ім'я гостя, номер телефону, дату, час, кількість гостей, номер столу, тип гостя, статус оплати та попередні замовлення. Така структура легко обробляється JavaScript-компонентами і може бути збережена у backend-частині. Для опису HTTP-

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

взаємодії, формату JSON, HTML-структури та браузерних механізмів збереження стану використано відкриті веб-стандарти й довідкові матеріали MDN, WHATWG та IETF [10], [11], [12], [13], [14].

SPA-підхід означає, що застосунок завантажується як одна HTML-сторінка, а переходи між розділами виконуються на клієнтській стороні. У Crescendo користувач може переходити між Home, Menu, Booking, Orders, Table Overview та Admin без повного перезавантаження сторінки. Для цього використовується React Router, який відповідає за маршрутизацію всередині застосунку. Для користувача це створює відчуття швидкої навігації, а для розробника дає можливість розділити систему на окремі сторінки та компоненти. Наприклад, сторінка бронювання, меню, список замовлень і адміністративна панель мають власну логіку, але працюють у межах одного застосунку.

Компонентний підхід React є важливим для підтримки структури системи. Кожен функціональний блок може бути реалізований як окремий компонент: форма бронювання, картка столу, сторінка меню, картка замовлення, календар адміністратора, список гостей або панель нотаток. Це спрощує розробку і подальше оновлення проєкту. Якщо потрібно змінити логіку відображення столу, не потрібно змінювати весь застосунок. Достатньо змінити відповідний компонент. Такий підхід відповідає сучасній практиці розробки веб-застосунків і добре підходить для дипломного проєкту, де потрібно показати не лише результат, а й структуровану архітектуру.

Адміністративна панель у роботі трактується як робочий простір персоналу ресторану. Вона відрізняється від публічної частини тим, що концентрує службову інформацію: календар бронювань, timeline сервісу, нотатки про гостей, підготовку столів і попередні замовлення для кухні. Публічна частина орієнтована на гостя і має бути простою, візуально привабливою та зрозумілою. Адміністративна частина, навпаки, повинна показувати більше даних, оскільки її завдання — допомогти працівникам приймати рішення, готувати зал і контролювати поточні бронювання.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

Отже, основні поняття ресторанної веб-системи охоплюють онлайн-бронювання, карту залу, стан столу, REST API, JSON-обмін даними, SPA-архітектуру, компонентний підхід, клієнтську маршрутизацію, серверний стан і адміністративний інтерфейс. Усі ці поняття в розробленому застосунку взаємопов'язані: користувач обирає параметри бронювання, система перевіряє доступність столу, frontend звертається до API, backend оновлює дані, а адміністративна панель відображає результат для персоналу. Саме ця взаємодія формує технічну і функціональну основу проєкту.

1.3. Аналіз існуючих рішень для ресторанного бронювання та керування залом

Існуючі рішення для ресторанного бронювання можна умовно поділити на три групи: універсальні сервіси резервування, вбудовані модулі в POS-системах і власні веб-рішення ресторану. Кожна з цих груп має свої переваги й обмеження. Універсальні сервіси резервування дають ресторану швидкий доступ до готової платформи, де вже реалізовано базові сценарії пошуку закладу, вибору часу та оформлення бронювання. Проте такі сервіси часто обмежують гнучкість бренду, дизайн інтерфейсу, логіку вибору столика та бізнес-правила. Ресторан стає частиною зовнішньої платформи, а не повністю контролює досвід користувача.

Вбудовані модулі в POS-системах мають іншу перевагу. Вони можуть бути пов'язані з оплатою, кухнею, замовленнями, касовими операціями й обліком. Для великих закладів або мереж це зручно, оскільки різні операційні процеси об'єднані в одному середовищі. Однак такі рішення можуть бути складними, дорогими або надмірними для невеликого чи авторського ресторану. Крім того, POS-системи часто орієнтовані передусім на внутрішню роботу закладу, а не на створення індивідуального публічного інтерфейсу для гостя.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

Третій підхід — власний веб-застосунок ресторану. Він дає найбільшу свободу в дизайні, структурі сторінок, сценаріях бронювання, візуальному представленні залу та реалізації специфічних бізнес-правил. Ресторан може самостійно визначати, які дані збирати, як показувати меню, чи дозволяти вибір конкретного столика, як працювати з депозитом, як відображати постійних гостей і які службові інструменти надати персоналу. Недоліком такого підходу є потреба в розробці, підтримці та подальшому технічному розвитку. Однак для дипломного проєкту саме власний веб-застосунок є найбільш доцільним, оскільки дозволяє продемонструвати повний цикл проєктування і реалізації системи.

Для ресторану преміального або авторського формату важливо, щоб бронювання було частиною цілісного користувацького досвіду. Гість повинен бачити атмосферу закладу, ознайомитися з меню, зрозуміти логіку вибору столика, умови депозиту та переваги гостьової картки в одному стилістичному середовищі. Якщо бронювання виведене на зовнішню платформу, ресторан частково втрачає контроль над цим досвідом. У Crescendo головна сторінка, меню, форма бронювання, карта залу та адміністративні модулі виконані як частини одного інтерфейсного середовища. Це дозволяє поєднати маркетингову, сервісну й операційну функції в одному веб-застосунку.

Багато сервісів бронювання приховують реальну структуру залу й дозволяють гостю обрати лише кількість осіб, дату і час. Такий підхід простий і швидкий, але він не враховує бажання гостя сісти біля вікна, у приватній зоні, на балконі або в іншій конкретній частині ресторану. Для частини закладів це не є проблемою, але для ресторанів, де простір має значення, такий підхід знижує якість взаємодії. У розробленому застосунку карта залу є прозорою: користувач бачить поверх, зону, місткість і статус столика. Це робить процес бронювання більш інформативним і наближеним до реального вибору місця.

Ще одним критерієм аналізу є робота з різними категоріями гостей. Універсальні сервіси бронювання здебільшого обробляють гостя як стандартного користувача: ім'я, контакт, дата, час і кількість осіб. Однак

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

ресторанна практика показує, що постійні клієнти мають інший статус. Для них можуть діяти спеціальні умови, наприклад бронювання без депозиту, врахування улюбленого столика, особливих уподобань або алергій. У Crescendo ця логіка представлена через guest card і Guest CRM. Постійний гість може бути розпізнаний системою, а персонал отримує додатковий контекст для підготовки сервісу.

Для адміністратора важливо не лише прийняти бронювання, а й підготувати сервіс. Тому окремим критерієм аналізу є наявність календаря, гостьових нотаток, алергій, уподобань, timeline обслуговування та підготовки кухні. У багатьох простих веб-формах після надсилання заявки адміністратор отримує лише повідомлення або електронний лист. Це не створює повноцінного робочого простору. У Crescendo ці можливості зібрано в admin dashboard, що робить систему не тільки формою для гостя, а й операційним інструментом. Працівник може бачити поточні бронювання, очікувану кількість гостей, часові проміжки, підготовку столів і попередні замовлення.

Значення має також зв'язок між бронюванням і кухнею. Якщо гість або адміністратор вносить попередні замовлення, ця інформація повинна бути доступна не лише в загальному списку, а й у контексті конкретного часу, столу та гостя. Це дозволяє кухні краще підготуватися до сервісу, а офіціантам — розуміти очікування гостей. У Crescendo така логіка представлена через блок Floor & Kitchen, де відображаються столи до підготовки та preorders. Це показує, що система охоплює не тільки момент створення бронювання, а й подальшу роботу ресторану перед приходом гостей.

При аналізі існуючих рішень важливо враховувати і складність впровадження. Повноцінні POS-системи або великі платформи можуть мати багато функцій, але для навчального проєкту їх повне дублювання є недоцільним. Завдання бакалаврської роботи полягає не в тому, щоб створити промислову систему для всіх процесів ресторану, а в тому, щоб реалізувати ключові сценарії автоматизації бронювання і керування залом. Тому в Crescendo акцент зроблено на тих модулях, які найкраще демонструють логіку

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

предметної області: публічна сторінка, меню, форма бронювання, карта столів, список замовлень, адміністративна панель, CRM гостей і операційні вкладки для сервісу.

Результати аналізу показують, що для навчального проєкту доцільно реалізувати власний веб-застосунок, який не намагається дублювати повну POS-систему, але покриває ключові сценарії бронювання. Обраний підхід дозволяє показати предметну область через конкретні функції: вибір столика, перевірку доступності, обробку бронювання, відображення замовлень, роботу адміністратора, guest card, нотатки про гостей і підготовку сервісу. Такий обсяг відповідає меті бакалаврської роботи та демонструє практичне застосування сучасного фронтенд-стеку в реальному бізнес-контексті ресторанної автоматизації.

1.4. Висновки по розділу

У першому розділі було проаналізовано предметну область автоматизації ресторанного сервісу, визначено основні поняття онлайн-бронювання та розглянуто підходи до створення ресторанных веб-систем. Встановлено, що бронювання в ресторані є не лише дією з боку гостя, а частиною ширшого процесу керування ресурсами закладу. До таких ресурсів належать столи, часові слоти, зони залу, поверхи, інформація про гостей, попередні замовлення та службові нотатки для персоналу.

У процесі аналізу з'ясовано, що ефективна система бронювання повинна поєднувати публічний інтерфейс для гостя та внутрішній інструмент для адміністратора. Гість має отримати можливість швидко переглянути меню, обрати дату, час, кількість гостей і конкретний столик. Адміністратор, зі свого боку, повинен бачити календар бронювань, завантаженість залу, timeline сервісу, інформацію про постійних гостей, алергії, уподобання та підготовку столів. Саме таке поєднання дозволяє розглядати веб-застосунок не як просту форму заявки, а як елемент цифрової організації ресторанного сервісу.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

Окрему увагу було приділено інтерактивній карті залу. Вона є важливим елементом системи, оскільки дозволяє враховувати не тільки факт наявності вільного місця, а й конкретні характеристики столу: місткість, поверх, зону, форму та поточний статус. Це зменшує ризик помилок під час бронювання, допомагає краще використовувати простір ресторану та робить процес вибору місця більш зрозумілим для користувача.

У розділі також визначено основні технічні поняття, які використовуються в роботі: онлайн-бронювання, стан столу, карта залу, REST API, JSON-обмін даними, SPA-архітектура, компонентний підхід, серверний стан і адміністративна панель. Було показано, що ці поняття взаємопов'язані між собою: користувач обирає параметри бронювання, frontend звертається до API, backend оновлює дані, а службова частина застосунку відображає актуальний стан для персоналу ресторану.

Аналіз існуючих рішень показав, що універсальні сервіси резервування та POS-модулі мають свої переваги, однак не завжди забезпечують потрібну гнучкість для авторського або преміального ресторану. Універсальні платформи часто обмежують можливості брендування та вибору конкретного столика, а POS-системи можуть бути надмірними для навчального проєкту або невеликого закладу. Тому для бакалаврської роботи доцільним є створення власного веб-застосунку, який покриває ключові сценарії бронювання, роботи з картою залу, guest card, адміністративною панеллю та підготовкою сервісу.

Отже, проведений аналіз підтвердив актуальність розробки веб-застосунку Crescendo. Обраний підхід дозволяє поєднати сучасний користувацький інтерфейс, логіку бронювання, роботу з даними через REST API, локальний backend на Node.js, інтерактивну карту столів і службові інструменти для персоналу. Це створює основу для подальшого проєктування системи, формування функціональних і нефункціональних вимог, а також розробки архітектури програмного рішення.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЄКТУВАННЯ

2.1. Збір та аналіз вимог зацікавлених сторін

Вимоги до веб-застосунку Crescendo формуються на перетині очікувань кількох груп зацікавлених сторін: гостя ресторану, адміністратора, персоналу залу, кухні, власника бізнесу та розробника. Кожна з цих груп має власні потреби, але всі вони пов'язані з одним процесом — якісною організацією бронювання та подальшого ресторанного сервісу. Саме тому під час аналізу вимог важливо враховувати не лише зовнішній інтерфейс для гостя, а й внутрішні операційні процеси, які відбуваються після створення бронювання.

Гість ресторану очікує, що веб-застосунок буде простим, зрозумілим і швидким. Для нього важливо без телефонного дзвінка переглянути меню, ознайомитися з атмосферою закладу, вибрати дату, час, кількість гостей і конкретний столик. Також користувач повинен розуміти, які столи доступні, які вже заброньовані, а які не підходять за місткістю. Якщо система не пояснює ці стани, гість може помилково обрати невідповідний варіант або не завершити бронювання. Тому однією з головних вимог є послідовний сценарій взаємодії, у якому користувач рухається від простих даних до вибору столу.

Для гостя критично важливо, щоб сценарій не вимагав зайвих дій. У Crescendo форма бронювання побудована як логічна послідовність: контактні дані, тип гостя, дата, час, кількість людей і вибір столика. Після вибору дати, часу та кількості гостей карта залу показує, які столи підходять за місткістю, а які вже зайняті або недоступні. Це зменшує кількість помилкових бронювань і робить процес більш керованим. Особливо важливо, що користувач не просто надсилає заявку, а взаємодіє з конкретним ресурсом ресторану — столом.

Адміністратор ресторану має інші вимоги. Йому недостатньо бачити лише факт бронювання. Для повноцінної роботи потрібен контекст: хто приходить, коли, скільки гостей очікується, який стіл заброньовано, чи є

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		25

гостьова картка, чи потрібен депозит, чи є алергії, уподобання або попередні замовлення. Саме тому в Crescendo адміністративна панель містить вкладки Guests, Calendar, Timeline, Guest notes та Floor & kitchen. Кожна з цих вкладок відповідає окремому операційному завданню: знати гостей, планувати тижневу завантаженість, бачити сьогоднішній сервіс, враховувати алергії та готувати зал.

Працівники залу потребують швидкого доступу до інформації про столи, час приходу гостей і підготовку місць. Для них важливо бачити, які столи потрібно підготувати, які гості є постійними, а також які особливі нотатки варто врахувати під час обслуговування. Кухня, своєю чергою, зацікавлена в інформації про попередні замовлення, якщо вони прив'язані до бронювання. Таким чином, бронювання є не лише дією користувача, а й початковою точкою для внутрішньої роботи ресторану.

Власник бізнесу очікує, що система зменшить обсяг ручної роботи, мінімізує помилки, покращить контроль за завантаженням залу та підвищить якість сервісу. Для нього важливими є не тільки окремі функції, а й загальна ефективність: чи дозволяє система швидше приймати бронювання, чи зменшує ризик накладення столів, чи допомагає працювати з постійними гостями, чи створює основу для майбутньої аналітики. У цьому контексті Crescendo розглядається як MVP ресторанної інформаційної системи, яку можна надалі розширювати.

Вимоги розробника пов'язані з підтримуваністю коду, зрозумілою архітектурою та можливістю подальшого розвитку. Проєкт розділено на сторінки, компоненти, API-модуль, DB-модулі з демонстраційними даними та конфігураційні файли. Така структура спрощує пошук відповідальності: маршрути знаходяться в системі маршрутизації, бронювання реалізується через відповідні компоненти форми та карти столів, карта залу спирається на дані розміщення столів, а адміністративні функції зібрані в AdminPage. Окремо виділено взаємодію з REST API, де frontend через Axios звертається до backend, а React Query відповідає за кешування, оновлення серверних даних.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.1

Зацікавлені сторони та їхні основні вимоги

Зацікавлена сторона	Основні потреби	Вимоги до системи
Гість ресторану	Швидке бронювання, зрозуміла карта залу, перегляд меню	Послідовна форма, доступні стани столів, вибір дати/часу/столу
Постійний гість	Бронювання з урахуванням гостьової картки	Поле guest card, можливість бронювання без депозиту
Адміністратор	Контроль бронювань і завантаження залу	Calendar, Timeline, Orders, перегляд деталей бронювання
Офіціанти	Підготовка до приходу гостей	Guest notes, інформація про алергії, уподобання, стіл і час
Кухня	Попередні замовлення та підготовка сервісу	Floor & kitchen, список preorders
Власник бізнесу	Зменшення ручної роботи, контроль процесів	Єдина система бронювання, CRM гостей, операційні дані
Розробник	Підтримуваність і розширюваність	Компонентна структура, REST API, окремий backend, Docker-розгортання

Після аналізу зацікавлених сторін можна зробити висновок, що система повинна підтримувати два різні режими роботи: публічний і службовий. Публічний режим орієнтований на гостя і містить головну сторінку, меню, форму бронювання та карту залу. Службовий режим орієнтований на адміністратора й персонал ресторану та містить календар, список бронювань, профілі гостей, нотатки й підготовку кухні. Саме тому у вимогах окремо виділено гостьові сценарії, операційні сценарії адміністратора, інтеграційні вимоги до API та вимоги до подальшого розгортання (рис. 2.1).

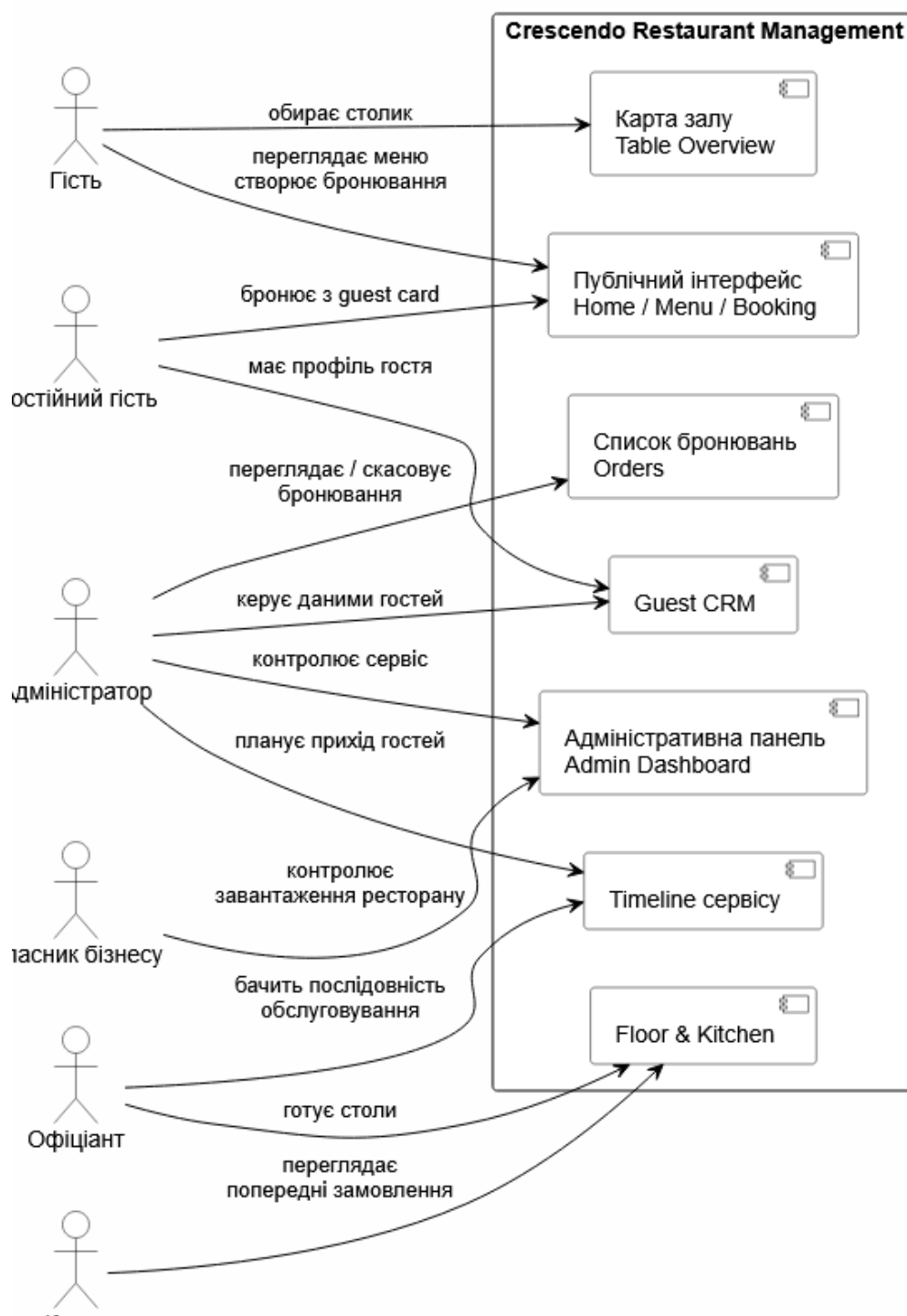


Рисунок 2.1 - Схема взаємодії зацікавлених сторін із веб-застосунком Crescendo

2.2. Функціональні вимоги до веб-застосунку Crescendo

Функціональні вимоги описують, що саме має виконувати система. Для веб-застосунку Crescendo вони згруповані за основними модулями: головна

сторінка, меню, бронювання, карта залу, перегляд активних бронювань, деталі бронювання, адміністративна панель, авторизація адміністратора, робота з REST API та контейнеризоване розгортання. Такий поділ дозволяє описати систему не як набір окремих сторінок, а як взаємопов'язаний набір сценаріїв.

Головна сторінка виконує роль першої точки входу до системи. Вона повинна представити ресторан, показати основні сценарії та надати швидкий перехід до бронювання, меню й плану столів. Для користувача ця сторінка має бути не лише інформаційною, а й навігаційною. Вона повинна одразу пояснювати, що система дозволяє не просто переглянути ресторан, а виконати конкретну дію — забронювати столик.

Модуль меню повинен відображати повний перелік ресторанних позицій, поділених на секції: Chef Tasting, Raw Bar, Seasonal Starters, Handmade Pasta, Fire & Grill, Seafood, Plant Forward, Dessert Room, Pairings & Bar. Користувач може переглянути всі страви або звузити список за конкретною секцією. Для кожної позиції виводяться назва, опис, ціна, бейдж і зображення. Це важливо, оскільки меню формує очікування гостя ще до бронювання і підсилює загальний користувацький досвід.

Модуль бронювання є центральною функцією веб-застосунку. Він повинен збирати ім'я, email, телефон, дату, час, кількість гостей і тип гостя. Для постійного гостя додатково вводиться номер guest card, а для нового гостя застосовується політика депозиту через LiqPay-конфігурацію. Якщо LiqPay-ключі не налаштовано, депозитний сценарій не повинен імітувати успішну оплату. Така поведінка є коректною для прототипу, оскільки вона не створює хибного враження про реальну платіжну інтеграцію.

Карта залу повинна підтримувати два поверхи, таблицю розміщення столів, їхню місткість, форму, зону і статус. Після вибору дати, часу та кількості гостей система має підсвічувати доступні столи, блокувати зайняті та позначати столи, що не підходять за місткістю. Обраний столик зберігається у формі як частина моделі бронювання, після чого frontend формує reservation-об'єкт і

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		29

передає його через API. Це забезпечує зв'язок між UI-рівнем і серверною логікою збереження стану.

Модуль Orders повинен відображати активні бронювання, отримані з API. Користувач або адміністратор має бачити список заброньованих столів, ім'я гостя, час, дату та можливість перейти до деталей конкретного запису. Деталі бронювання повинні показувати повнішу інформацію про reservation: контактні дані, номер столу, кількість гостей, тип гостя, оплату та попередні замовлення. Також система повинна дозволяти скасування бронювання. Скасування виконується через оновлення статусу столика на Available і очищення reservation-даних.

Адміністративна панель повинна показувати службову картину ресторану. До неї входять статистика сьогоднішніх бронювань, очікувана кількість гостей, кількість постійних гостей, стан депозитів, календар на тиждень, timeline сервісу, профілі гостей, алергії, уподобання, підготовка столів і попередні замовлення для кухні. Такий набір функцій дозволяє використовувати адміністративну панель як цифрове робоче місце працівника ресторану, а не лише як сторінку перегляду даних.

Окремою функціональною вимогою є авторизація адміністратора. У прототипі вона реалізується на демонстраційному рівні через localStorage. Такий механізм дозволяє показати різницю між публічними сторінками і службовою частиною, але не є повноцінною production-системою безпеки.

Робота з REST API є інтеграційною функціональною вимогою. Frontend повинен отримувати дані про столи, бронювання та статуси через HTTP-запити. У системі використовується локальний backend на Node.js 20 з JSON DB, що дозволяє зберігати стан столів і бронювань. Axios виконує запити до API, а React Query відповідає за кешування, повторне отримання та інвалідацію даних після створення або скасування бронювання.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

Основні функціональні вимоги

Код	Вимога	Критерій приймання
FR-01	Перегляд головної сторінки	Користувач бачить навігацію, опис сервісу та кнопки переходу до бронювання, меню й плану столів
FR-02	Перегляд меню за секціями	Користувач бачить повний список страв і може фільтрувати його за розділами
FR-03	Створення бронювання	Форма приймає валідні дані та передає reservation-об'єкт в API
FR-04	Вибір типу гостя	Для нового гостя активується депозитний сценарій, для постійного - поле guest card
FR-05	Інтерактивна карта залу	Недоступні, заброньовані або замалі столи не можна обрати
FR-06	Перегляд активних бронювань	Сторінка Orders показує список заброньованих столів
FR-07	Перегляд деталей бронювання	Користувач може відкрити окремий запис і переглянути його дані
FR-08	Скасування бронювання	Після cancel action статус столу оновлюється, а список бронювань перезавантажується
FR-09	Адміністративний календар	Адмін бачить завантаженість за днями та основні показники сервісу
FR-10	Guest CRM	Адмін бачить профілі гостей, guest card, алергії, уподобання й історію
FR-11	Timeline сервісу	Адмін бачить послідовність приходу гостей за часом
FR-12	Floor & kitchen	Персонал бачить столи до підготовки та попередні замовлення
FR-13	Демо-авторизація	Після входу адміністратор отримує доступ до службових сторінок
FR-14	Робота з REST API	Дані отримуються та оновлюються через GET/POST/PATCH-запити
FR-15	Оновлення UI після змін	Після створення або скасування бронювання React Query оновлює кеш

Функціональні вимоги відображають повний цикл роботи системи: від першого входу гостя на сайт до підготовки ресторану до обслуговування. UML-діаграма на рисунку 2.2 показує, які користувачі взаємодіють із системою і які дії вони можуть виконувати. Це дає наочне уявлення про межі та зв'язки функцій веб-застосунку Crescendo.

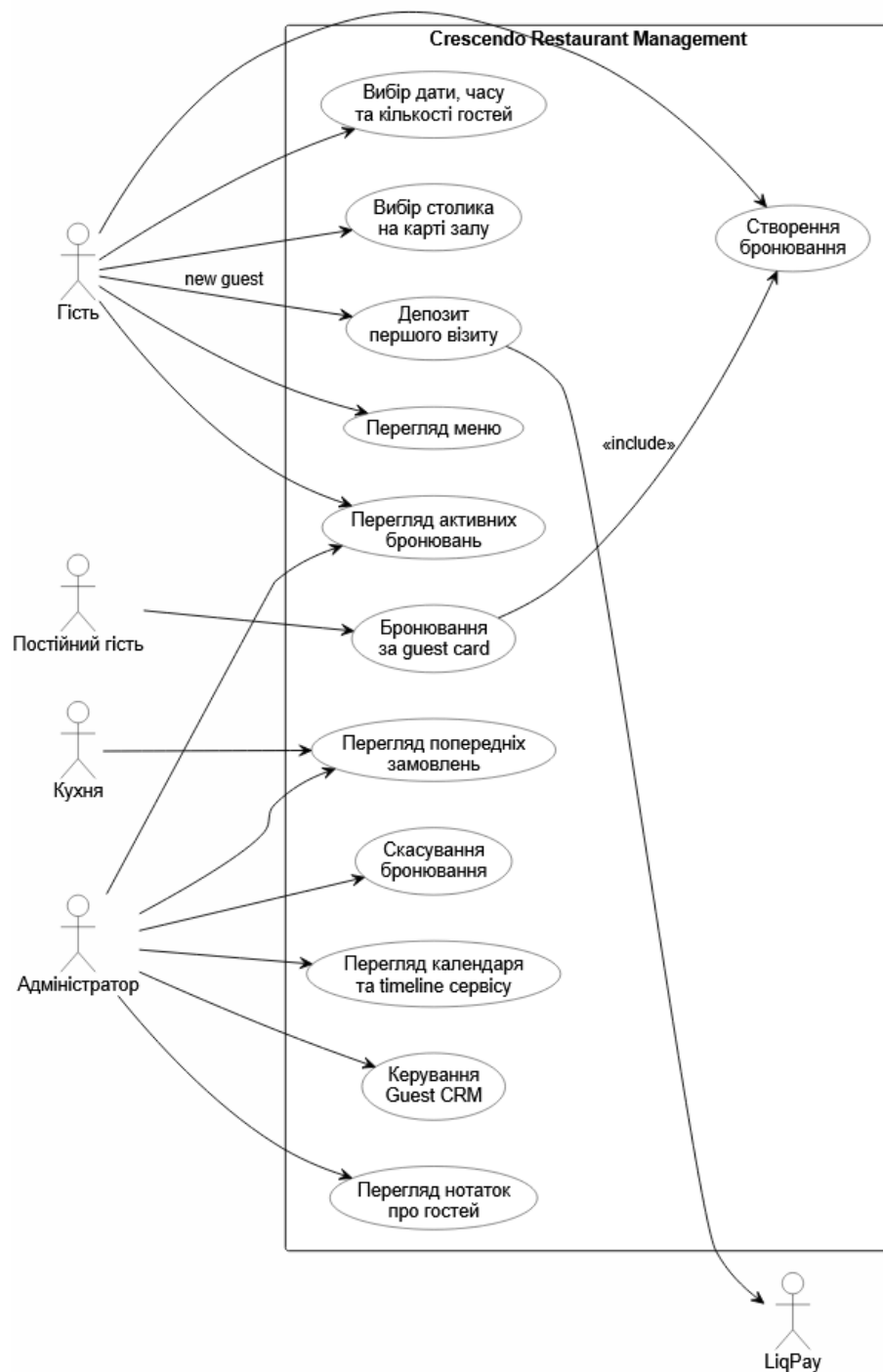


Рисунок 2.2 - UML-діаграма випадків використання веб-застосунку Crescendo

Функціональні вимоги охоплюють повний цикл роботи системи — від першого входу гостя на сайт до підготовки ресторану до обслуговування. UML-діаграма на рисунку 2.2 показує взаємодію користувачів із системою та основні сценарії дій для гостей, постійних гостей і адміністратора.

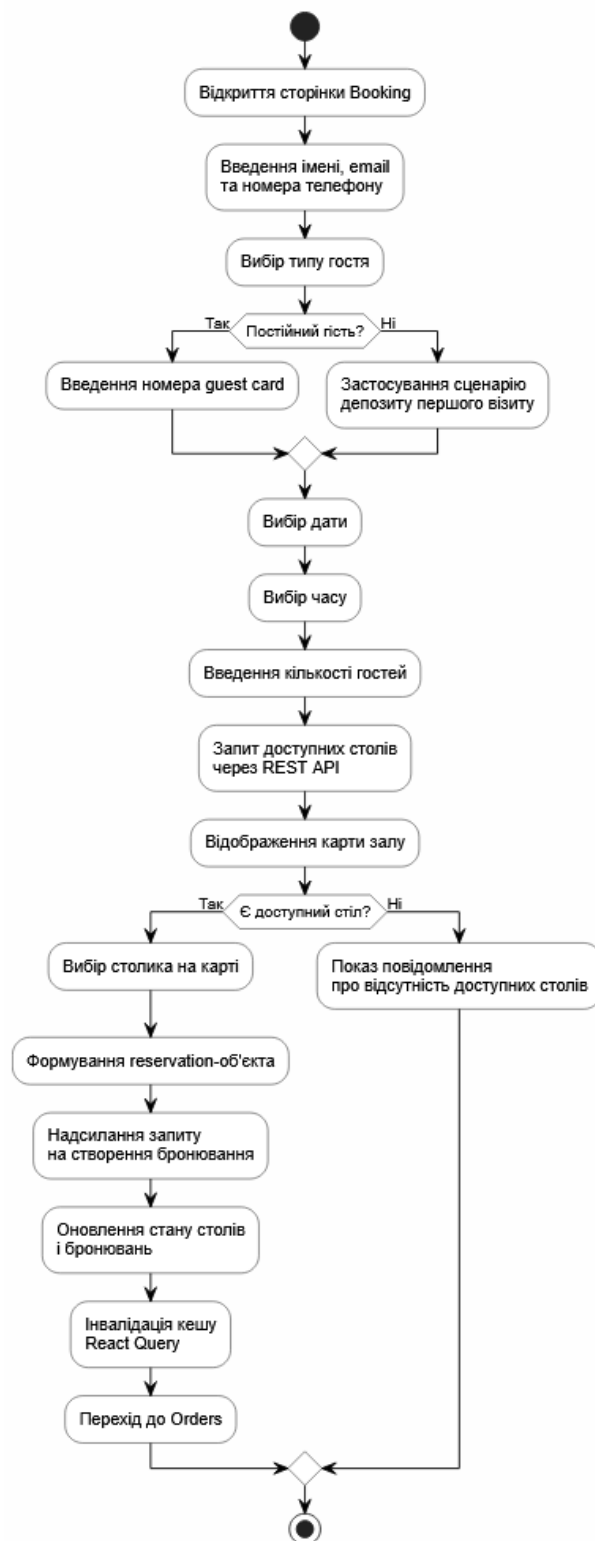


Рисунок 2.3 - Блок-схема сценарію створення бронювання

Блок-схема на рисунку 2.3 відображає послідовність кроків під час створення бронювання: введення даних користувача, вибір типу гостя, дати та часу, перевірка доступності столів, формування reservation-об'єкта та передача

даних через API.

2.3. Нефункціональні вимоги та критерії приймання

Нефункціональні вимоги визначають якісні характеристики системи. Якщо функціональні вимоги відповідають на питання, що саме має робити застосунок, то нефункціональні описують, як саме він має це робити. Для веб-застосунку ресторану важливими є зручність інтерфейсу, швидкість навігації, зрозумілість станів, адаптивність, підтримуваність коду, надійність взаємодії з API, базова безпека, контейнеризація та можливість подальшого розвитку.

Зручність використання є однією з головних вимог, оскільки застосунок орієнтований на гостей ресторану. Користувач повинен зрозуміти сценарій бронювання без додаткових пояснень. Для цього форма має бути послідовною, поля — зрозумілими, а стани столів — візуально відмінними. Якщо стіл зайнятий або не підходить за кількістю гостей, він не повинен бути доступним для вибору. Такий підхід зменшує ризик помилок і підвищує довіру до системи.

Інтерфейс повинен бути адаптивним. Оскільки гості можуть заходити на сайт як із комп'ютера, так і з телефону, сторінки мають коректно відображатися на різних розмірах екрана. У проєкті використовуються гнучкі контейнери, компоненти Material UI та CSS-медіа-запити. Це дозволяє адаптувати публічні сторінки, меню, форму бронювання й карту залу до різних пристроїв. Для production-рівня доцільно додатково протестувати всі критичні сценарії на смартфонах, планшетах і desktop-екранах.

Швидкість і передбачуваність роботи інтерфейсу забезпечуються SPA-підходом і клієнтською маршрутизацією. Користувач може переходити між основними сторінками без повного перезавантаження документа. Це особливо важливо для сценарію бронювання, де швидкість реакції інтерфейсу впливає на завершення дії. Якщо сторінки відкриваються повільно або після кожного кроку відбувається повне перезавантаження, користувач може залишити сайт.

Надійність системи пов'язана з коректною обробкою помилок API.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

Якщо GET, POST або PATCH-запит завершується помилкою, користувач повинен отримати зрозуміле повідомлення, а не технічний збій. У проєкті для таких випадків можуть використовуватися toast-повідомлення. Для адміністративної панелі важливо також передбачити fallback-поведінку, щоб під час демонстрації можна було показати базові екрани навіть у разі недоступності частини даних.

Підтримуваність забезпечується поділом коду на сторінки, компоненти, API-функції та окремі модулі даних. Наприклад, карта залу має власну структуру даних, меню винесене в окремий модуль, API-запити зосереджені в окремому файлі, а адміністративні сценарії згруповані в межах відповідної сторінки. Це дозволяє швидко знайти потрібну частину коду і змінити її без ризику зламати інші модулі.

Окремою нефункціональною вимогою є збереження стану даних. У прототипі операційний стан столів і бронювань зберігається через локальний Node.js backend та JSON DB. Для навчального MVP це дозволяє показати реальну зміну стану після створення або скасування бронювання. Водночас така модель має обмеження: для production-версії краще використовувати повноцінну базу даних із транзакціями, ролями доступу та надійнішим збереженням даних.

Безпека у прототипі має демонстраційний характер. Адміністративний доступ реалізовано через localStorage, що показує принцип розмежування публічної та службової частини. Однак для реальної системи такий підхід є недостатнім. У подальшій версії необхідно реалізувати серверну авторизацію, ролі користувачів, захищені API-endpoint-и та безпечне збереження токенів або сесій. Також приватні ключі платіжних сервісів не повинні зберігатися в React-застосунку, оскільки frontend-код доступний користувачу.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

Нефункціональні вимоги та критерії приймання

Характеристика	Вимога	Критерій приймання
Зручність	Послідовна форма та зрозумілі стани столів	Користувач проходить бронювання без сторонніх пояснень
Адаптивність	Коректне відображення на різних екранах	Основні сторінки доступні на desktop і mobile
Швидкість навігації	Переходи між сторінками без повного перезавантаження	SPA-маршрути відкриваються через React Router
Надійність	Обробка помилок API	Помилка показується toast-повідомленням або fallback-даними
Підтримуваність	Поділ коду на сторінки, компоненти, API та DB-модулі	Функціональність легко локалізувати у файловій структурі
Цілісність даних	Збереження стану столів і бронювань	Після створення або скасування бронювання стан оновлюється
Безпека	Не зберігати приватні LiqPay-ключі у React	Private key має бути винесений у server-side зону майбутньої інтеграції
Розгортання	Docker/Nginx production-збірка	HTTP-запит до локального frontend повертає сторінку застосунку
Масштабованість	Можливість заміни JSON DB на повноцінну СУБД	Модель даних дозволяє перейти до бази даних у майбутньому

Загалом можна зробити висновок, що нефункціональні вимоги спрямовані не лише на комфорт користувача, а й на якість технічної реалізації. Для дипломного проєкту важливо показати, що система не просто має потрібні кнопки й сторінки, а побудована так, щоб її можна було підтримувати, демонструвати, тестувати й розширювати. Особливо важливими є вимоги до обробки помилок, оновлення стану після API-запитів і чіткого розмежування публічної та адміністративної частини.

Критерієм приймання є можливість виконати основні сценарії роботи системи: відкрити головну сторінку, перейти до меню, відкрити форму бронювання, вибрати дату, час і кількість гостей, побачити карту залу, обрати доступний столик, створити бронювання, перейти до списку Orders, відкрити деталі та скасувати бронювання. Для адміністратора критерієм є доступ до календаря, профілів гостей, нотаток, timeline сервісу та підготовки залу після авторизації (рис. 2.4).

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

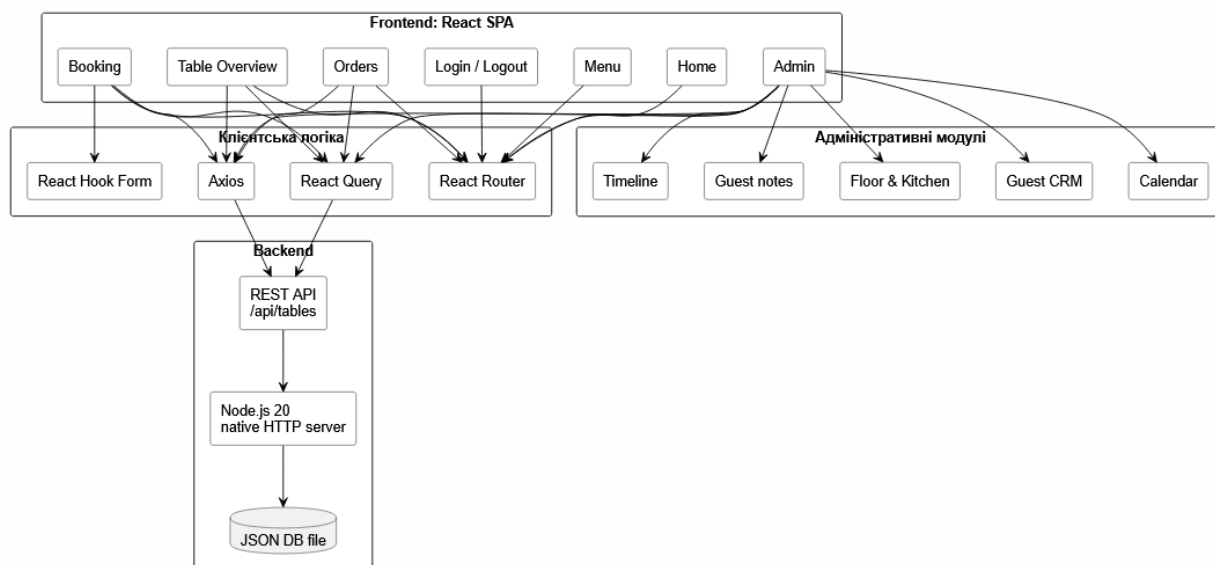


Рисунок 2.4 - Логічна структура модулів веб-застосунку Crescendo

2.4. Висновки по розділу

У розділі 2 проведено комплексний аналіз вимог до веб-застосунку Crescendo, що охоплює потреби всіх зацікавлених сторін: гостей ресторану, адміністратора, персоналу залу, кухні, власника та розробника. Кожна група має специфічні очікування, проте всі вони об'єднані спільною метою — забезпечити ефективну організацію бронювання та високий рівень ресторанного сервісу. Система повинна підтримувати послідовний сценарій бронювання для гостей, надавати швидкий та наочний доступ до меню, карти залу та інформації про доступність столів, а також гарантувати, що адміністративна панель і внутрішні модулі дозволяють персоналу ефективно планувати роботу та контролювати всі процеси. При цьому архітектура застосунку передбачає можливість подальшого розвитку та інтеграції з REST API, що забезпечує масштабованість та легке додавання нових функцій у майбутньому.

Функціональні вимоги відображають весь цикл взаємодії з системою — від першого входу гостя на сайт до підготовки ресторану до обслуговування. Вони охоплюють перегляд головної сторінки, меню, створення та управління

бронюваннями, інтерактивну карту залу, адміністративну панель та механізми авторизації. Діаграми UML і блок-схеми, наведені в розділі, ілюструють взаємозв'язки користувачів із системою та послідовність ключових сценаріїв, що дозволяє наочно показати, як відбувається обмін даними між frontend, backend та адміністративними модулями. Такий підхід гарантує, що навіть новий користувач або розробник зможе швидко зрозуміти логіку системи та принципи взаємодії компонентів без необхідності вивчати весь код окремо.

Нефункціональні вимоги забезпечують якість та передбачуваність роботи системи, створюючи умови для комфортного користування та надійної експлуатації. Вони включають зручність інтерфейсу, адаптивність до різних пристроїв, швидку навігацію, коректну обробку помилок API, збереження стану даних, безпеку адміністративного доступу та можливість контейнеризованого розгортання через Docker/Nginx. Такі характеристики гарантують, що Crescendo не лише відповідає очікуванням користувачів і персоналу, а й створює міцну основу для демонстрації, тестування та подальшого розвитку застосунку, забезпечуючи баланс між функціональністю, зручністю та технічною підтримуваністю системи.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		38

РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ, МОДЕЛІ ДАНИХ ТА API

3.1. Обґрунтування архітектурної моделі системи

Проектування архітектури веб-застосунку Crescendo виконано з урахуванням того, що система має обслуговувати два різні, але пов'язані між собою режими роботи: публічну взаємодію гостя з рестораном і службову роботу адміністратора та персоналу. Публічний режим охоплює перегляд головної сторінки, меню, вибір дати, часу, кількості гостей і конкретного столика. Службовий режим зосереджений на календарі бронювань, профілях гостей, timeline сервісу, підготовці столів і попередніх замовленнях для кухні. Тому архітектура не може бути лише набором статичних сторінок: вона повинна підтримувати зміну стану, синхронізацію даних і контроль доступності ресурсів ресторану.

Базовою архітектурною моделлю обрано клієнт-серверний підхід. Клієнтська частина реалізована як React SPA, тобто односторінковий веб-застосунок, який після початкового завантаження виконує навігацію між сторінками без повного перезавантаження документа. Серверна частина реалізована на Node.js 20 і надає REST API для роботи зі столами та бронюваннями. Така модель відповідає характеру предметної області: інтерфейс має бути швидким і зручним, але стан бронювань повинен зберігатися на серверному рівні, а не лише в браузері користувача.

На клієнтському рівні кореневий компонент App підключає QueryClientProvider, ToastContainer та AuthContext. QueryClientProvider потрібен для роботи React Query і синхронізації server-state, ToastContainer використовується для зрозумілих повідомлень про успішні або помилкові дії, а AuthContext відповідає за демонстраційний стан адміністративного входу. Такий набір провайдерів дозволяє не дублювати службову логіку в кожній сторінці, а винести її в загальний контекст застосунку.

Маршрутизацію винесено в PageRoutes. Це рішення спрощує підтримку

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

системи, оскільки всі сторінки зібрані в одному логічному місці: Home, Menu, Booking, Orders, OrderDetail, Table Overview, Admin, Login і Logout. Частина сторінок завантажується через React.lazy, що демонструє підхід до code splitting. Для дипломного MVP це не є критичною оптимізацією, але вона показує правильний напрям розвитку, коли великі частини інтерфейсу не завантажуються до того моменту, коли вони реально потрібні користувачу.

Архітектура Crescendo спеціально розділяє presentation layer, client state, server state, API layer і persistence layer (табл. 3.1). Presentation layer відповідає за сторінки, компоненти, форму бронювання та карту залу. Client state містить поточні дії користувача: вибраний поверх, введені значення форми, активну вкладку або обраний столик. Server state охоплює дані, які надходять із backend: столи, бронювання, статуси та службові записи. API layer приймає HTTP-запити, валідовує дані й повертає відповіді у форматі JSON. Persistence layer зберігає стан у JSON DB-файлі Docker volume.

Для ресторанного бронювання такий поділ є принциповим. Якщо весь стан зберігати тільки на frontend, користувач бачитиме красивий інтерфейс, але дані не матимуть операційної цінності для ресторану. Якщо, навпаки, перенести всю логіку в backend без гнучкого клієнтського інтерфейсу, процес вибору столика стане повільним і незручним. Обрана архітектура поєднує обидва підходи: frontend відповідає за швидку взаємодію, а backend - за цілісність і збереження стану.

Nginx у системі виконує дві функції. Перша функція - роздавання production-збірки React SPA як набору статичних файлів. Друга функція - reverse проху для запитів, що починаються з /api/. Завдяки цьому браузер звертається до одного origin, а Nginx уже всередині Docker Compose передає API-запити до backend-сервісу. Це зменшує складність налаштування клієнтської частини й наближає локальне розгортання до production-підходу [26], [27].

JSON DB використано як демонстраційне сховище даних. У ньому зберігаються фізичні столи, майбутні бронювання, лічильники nextId і

					БР.ІП - 02.00.00.000 ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

nextBookingId. Перевагою такого рішення для MVP є простота, прозорість і можливість швидко перевірити зміну стану після бронювання або скасування. Недоліком є відсутність транзакцій і складної конкурентної обробки, тому в production-версії JSON DB має бути замінена на повноцінну СУБД, наприклад PostgreSQL або MySQL.

Архітектурна модель також враховує майбутню платіжну інтеграцію. У поточній версії LiqPay представлено як конфігураційний шар для сценарію first visit deposit. Це означає, що система вже має місце в логіці бронювання, де перевіряється необхідність депозиту, але реальний підпис платежу не виконується на frontend. Такий підхід є правильним з погляду безпеки: приватний ключ платіжної системи не повинен потрапляти в React-код, оскільки він доступний користувачу після збірки [30].

Для розмежування публічної і службової частини використовується демонстраційна авторизація через AuthContext і localStorage. Вона не є production-механізмом, але дозволяє показати принцип: гість працює з відкритими сторінками, а адміністратор отримує доступ до службових екранів після входу. У майбутній версії цей рівень має бути замінений на серверну автентифікацію, ролі доступу, захищені cookie або JWT і перевірку прав на рівні API [29].

Обрана архітектура відповідає вимогам до підтримуваності. Кожен рівень має власну відповідальність: React-компоненти формують UI, API-модуль із Axios виконує HTTP-запити, React Query керує кешем і повторним отриманням даних, backend валідовує payload і записує зміни, Nginx відповідає за доставку frontend та маршрутизацію /api (рис. 3.1). Завдяки цьому помилка або зміна в одному модулі не повинна вимагати переписування всієї системи.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		41

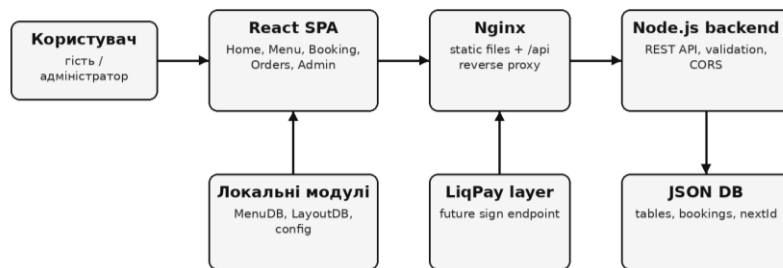


Рисунок 3.1 - Архітектурна модель веб-застосунку Crescendo

Таблиця 3.1

Архітектурні рівні веб-застосунку Crescendo

Архітектурний рівень	Складові	Відповідальність
Presentation layer	React-сторінки, компоненти, карта залу, форми	Відображення інтерфейсу та сценаріїв користувача
Client state	React Hook Form, локальні useState, AuthContext	Тимчасові значення форми, вкладки, вибір поверху, demo-auth
Server state	React Query, Axios, cache keys tables/bookings	Отримання, кешування й оновлення API-даних
API layer	Node.js native HTTP server, /api/tables, /ready	Обробка GET/POST/PATCH, CORS, валідація payload
Persistence layer	JSON DB у Docker volume backend-data	Збереження стану столів, бронювань і лічильників
Deployment layer	Docker Compose, Nginx, backend container	Відтворюване локальне розгортання та /api reverse proxy

На практиці архітектура Crescendo працює як ланцюжок узгоджених дій (рис. 3.2). Користувач відкриває сторінку Booking, frontend через React Query отримує список столів із /api/tables, після вибору параметрів бронювання інтерфейс визначає доступність столів, а після submit надсилає PATCH або POST-запит. Backend приймає дані, нормалізує їх, записує у JSON DB і повертає оновлений стан. Після цього React Query інвалідує кеш, і користувач бачить актуальну інформацію без ручного оновлення сторінки.

Важливо, що архітектура не прив'язана лише до одного сценарію бронювання. Ті самі дані tables і bookings використовуються в Booking, Orders, Table Overview та AdminPage (табл. 3.2). Це забезпечує єдине джерело істини

для різних частин системи. Якщо стіл змінює статус у результаті бронювання, цей стан повинен бути однаково відображений на карті залу, у списку активних бронювань і в адміністративній панелі.

Такий підхід дозволяє уникнути дублювання логіки. Наприклад, список столів не зберігається окремо в кожній сторінці. Він отримується з API і обробляється відповідними компонентами. Якщо в майбутньому зміниться модель столу або з'явиться нове поле, наприклад мінімальний депозит чи зона обслуговування, зміни можна внести в модель даних і API, а не підтримувати кілька незалежних копій.

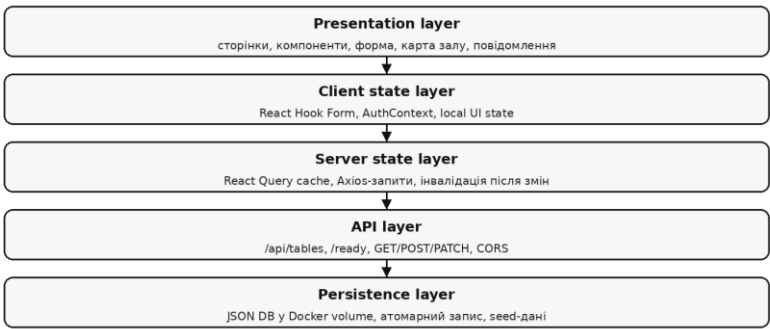


Рисунок 3.2 - Розподіл архітектурних рівнів і відповідальності

Таблиця 3.2

Розподіл відповідальності між основними компонентами

Компонент	Основна роль у системі	Чому винесено окремо
React SPA	Публічні та службові сторінки, інтерактивна карта, форма бронювання	Потрібна швидка клієнтська взаємодія без перезавантаження сторінок
React Query	Кешування та синхронізація server-state	Дані tables/bookings використовуються кількома сторінками
Axios API layer	Єдина точка HTTP-запитів до /api	Зменшує дублювання endpoint-ів у компонентах
Node.js backend	REST API, валідація, зміна стану JSON DB	Стан бронювань не повинен зберігатися лише на frontend
Nginx	Статична роздача SPA та проху /api	Один origin для frontend і API, простіше Docker-розгортання
Docker volume	Збереження restik-db.json між restart контейнерів	Дані демонстраційного backend не втрачаються після перезапуску

3.2. Моделювання бізнес-процесів і користувацьких сценаріїв

Моделювання бізнес-процесів у Crescendo починається не з технічних endpoint-ів, а з реальної логіки роботи ресторану (рис. 3.3). Гість хоче забронювати столик у певний день і час, ресторан має обмежену кількість місць, адміністратор повинен бачити майбутнє навантаження, а персонал - підготувати зал і кухню. Тому основний процес системи розглядається як послідовність дій, де кожен крок змінює або уточнює операційний стан закладу.

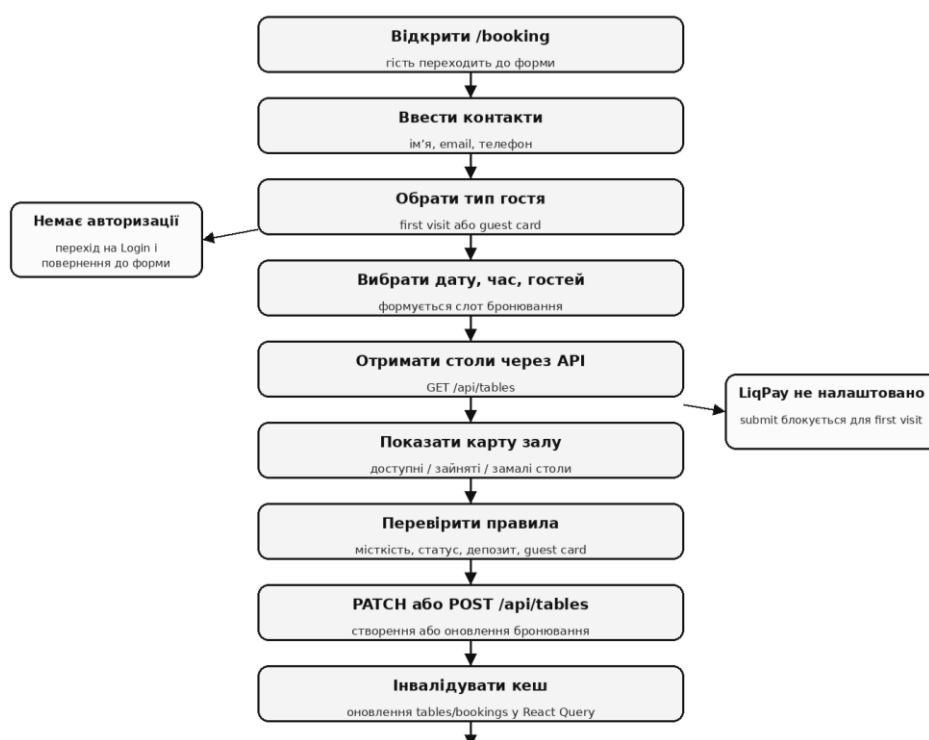


Рисунок 3.3 - Activity-схема бізнес-процесу онлайн-бронювання

Головним бізнес-процесом є онлайн-бронювання столика. Він починається з переходу гостя на сторінку Booking і введення контактних даних. Далі користувач обирає тип гостя: first visit або guest card. Цей вибір не є декоративним, оскільки він впливає на подальші правила. Для нового гостя система застосовує депозитний сценарій, а для постійного гостя очікує номер гостьової картки та дозволяє бронювання без депозиту.

Після вибору типу гостя користувач вводить дату, час і кількість гостей.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		44

Саме ці параметри формують слот бронювання. Без них карта залу не може коректно показати доступність столів, тому що один і той самий стіл може бути вільним в один час і заброньованим в інший. Крім того, кількість гостей впливає на фільтрацію за місткістю: стіл на дві особи не повинен бути доступним для компанії з шести людей.

Коли сформовано часовий контекст, frontend отримує список столів через API і поєднує операційний стан із layout-даними. На цьому етапі карта залу виконує не лише візуальну, а й логічну роль. Вона показує, які столи доступні, які вже заброньовані, які не підходять за місткістю, а який стіл зараз вибрано користувачем. Таке моделювання наближує систему до реальної роботи хостес, яка одночасно дивиться на зал, час і кількість гостей.

Процес має кілька контрольних точок, основні етапи якого наведені в таблиці 3.3. Перша контрольна точка - валідність контактних даних. Друга - вибір типу гостя. Третя - наявність дати, часу і кількості гостей. Четверта - можливість вибрати стіл відповідної місткості. П'ята - перевірка депозитної логіки. Якщо хоча б одна з цих умов не виконана, система не повинна створювати бронювання, бо це призведе до некоректного операційного запису.

Для нового гостя контроль депозиту має окреме значення. У прототипі LiqPay не виконує реальний платіж, але конфігураційна перевірка демонструє місце, де повинен бути платіжний сценарій. Якщо LiqPay-ключі або sign endpoint не налаштовано, система не імітує успішну оплату. Це важливо з погляду коректності: MVP не повинен створювати враження, що депозит справді списаний або зарезервований, якщо такого server-side механізму ще немає.

Після успішної перевірки frontend формує reservation-об'єкт. У ньому зберігаються ім'я, email, телефон, дата, час, кількість гостей, тип гостя, номер гостьової картки, номер столу і payment-дані. Далі система виконує PATCH /api/tables/:id для оновлення фізичного столу або POST /api/tables для table-like fallback-сценарію. У звичайному сценарії оновлюється статус конкретного столика, а reservation записується в його дані.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

Завершальним кроком є інвалідація кешу React Query. Після створення або скасування бронювання не можна залишати старі дані в інтерфейсі, бо користувач або адміністратор побачить неактуальний стан. Тому після мутації система оновлює tables/bookings і перенаправляє користувача до Orders або показує оновлену карту залу. Це забезпечує відчуття цілісності: дія користувача одразу відображається в інших частинах застосунку.

Таблиця 3.3

Етапи бізнес-процесу бронювання та контрольні умови

Етап процесу	Дані або дія	Контрольна умова	Результат
Вхід на сторінку	Перехід до /booking	Сторінка доступна через React Router	Відображення форми бронювання
Контакти гостя	Ім'я, email, телефон	Поля не порожні, формат email коректний	Можна перейти до параметрів бронювання
Тип гостя	First visit або guest card	Для guest card потрібен номер картки	Визначається депозитна політика
Слот	Дата, час, кількість гостей	Дата не минула, guests відповідає межах форми	Можна фільтрувати столи
Карта залу	Список tables + layout	Стіл доступний і має достатню місткість	Користувач обирає конкретний table_no
Створення запису	Reservation model	Payload містить дозволені поля	PATCH або POST до backend
Оновлення стану	React Query invalidate	Запит завершився без помилки	Orders і карта показують актуальні дані

Другим важливим процесом є перегляд і скасування бронювання. Після створення запису користувач або адміністратор може перейти на сторінку Orders і відкрити деталі конкретного бронювання. У деталях відображається не тільки номер столика, а й контекст запису: дата, час, гість, кількість людей, тип гостя, payment-статус і можливі попередні замовлення. Цей сценарій потрібен для контролю того, що бронювання справді збережено, а не залишилося лише у стані форми.

Скасування бронювання змінює стан системи у зворотному напрямку. Якщо користувач або адміністратор натискає cancel action, frontend надсилає

PATCH-запит до /api/tables/:id. Backend очищає reservation, змінює status на Available і повертає оновлений об'єкт. Після цього React Query перезавантажує відповідні дані. У результаті стіл знову стає доступним для нового бронювання, а список активних бронювань не показує скасований запис чинним (табл. 3.4).

Таблиця 3.4

Основні бізнес-правила сценарію бронювання

Бізнес-правило	Опис правила	Реалізація в системі
Вибір слота перед столом	Стіл має сенс лише в контексті дати, часу і кількості гостей	Форма очищає вибір столика після зміни параметрів
Місткість столу	Кількість гостей не повинна перевищувати seating_capacity	Недостатні столи отримують стан too-small
Зайнятий стіл	Booked-стіл не можна повторно вибрати для того самого слота	UI блокує вибір, backend зберігає статус
Перший візит	Новий гість має пройти депозитний сценарій	Перевіряється LiqPay-конфігурація
Guest card	Постійний гість бронює без депозиту, але вводить номер картки	Форма вимагає memberCard для відповідного режиму
Скасування	Після скасування стіл повертається у доступний стан	PATCH очищає reservation і ставить Available

Окремим сценарієм є робота адміністратора (рис. 3.4). Адміністратор не створює кожне бронювання вручну, але він повинен бачити наслідки дій гостей. Після входу в AdminPage він отримує доступ до показників поточного дня, календаря, timeline сервісу, Guest CRM, нотаток і блоку Floor & kitchen. Така структура відповідає реальному робочому дню ресторану: спочатку оцінюється загальне навантаження, потім конкретні часові слоти, далі особливості гостей і підготовка залу.

Calendar у службовій частині потрібен для огляду завантаження за днями. Timeline показує послідовність приходу гостей у межах дня. Guest CRM зберігає інформацію про постійних гостей, алергії, уподобання та історію. Floor & kitchen переводить бронювання в операційне завдання: які столи потрібно підготувати, скільки людей очікується, чи є попередні замовлення, чи потрібно

попередити кухню про особливі вимоги.

Для власника бізнесу ці сценарії мають управлінське значення. Система не лише приймає заявки, а й формує основу для контролю завантаження ресторану. Надалі ці дані можна використати для аналітики: популярні години, середня кількість гостей, частка постійних клієнтів, кількість скасувань і потреба в депозитах. У поточному MVP аналітика ще не винесена в окремий модуль, але модель процесів уже збирає потрібний операційний контекст.

Моделювання бізнес-процесів також допомагає визначити межі відповідальності системи. Crescendo не замінює повну POS-систему, не керує касовими операціями і не виконує production-платежі. Його межа - бронювання, карта залу, службова підготовка ресторану та демонстраційний backend. Така межа є коректною для бакалаврського проєкту, оскільки дозволяє реалізувати завершений MVP без надмірного розширення теми.

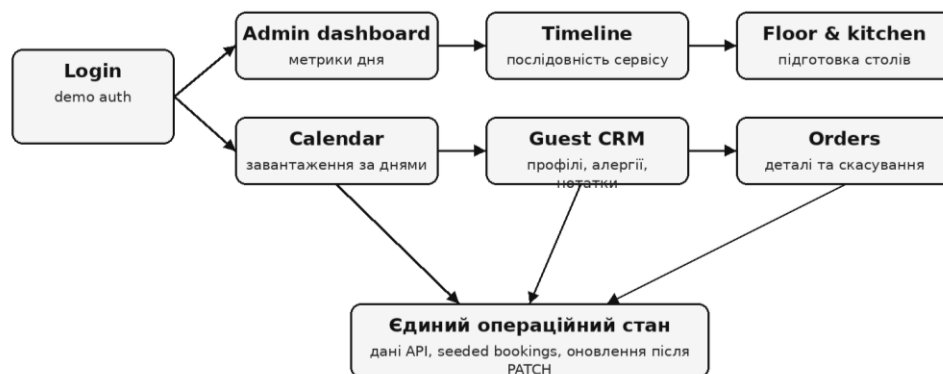


Рисунок 3.4 - Операційний сценарій адміністратора ресторану

Підсумовуючи, моделювання процесів показує, що Crescendo працює як єдиний ланцюг: гість створює бронювання, frontend перевіряє інтерфейсні умови, backend змінює операційний стан, а адміністратор отримує службову інформацію для підготовки сервісу. Саме це відрізняє систему від простої контактної форми: кожна дія користувача має конкретний наслідок для карти залу, списку бронювань і роботи персоналу.

3.3. Проектування моделі даних та REST API

Модель даних Crescendo побудована навколо двох основних понять: фізичного столу ресторану і бронювання (рис. 3.5). Фізичний стіл існує незалежно від конкретної дати чи гостя: він має номер, місткість, зону, поверх, форму і координати на карті. Бронювання, навпаки, є подією, прив'язаною до гостя, дати, часу, кількості людей і конкретного столу. Розділення цих понять дає змогу не плутати постійний план ресторану з тимчасовим операційним станом.

У JSON DB кореневий об'єкт містить nextId, nextBookingId, масив tables і масив bookings. Масив tables описує 24 фізичні столи ресторану. Кожен елемент має id, table_no, seating_capacity, status, reservation та orders. Масив bookings використовується для демонстраційних майбутніх записів і може бути розширений у майбутній production-версії. Така структура є простою, але достатньою для демонстрації API, карти залу та адміністративної панелі.

Сутність Table виконує роль операційного представлення столу. Її поле status показує поточний стан: Available або Booked. Поле reservation містить дані бронювання, якщо стіл зайнятий. Поле orders дозволяє зберігати попередні замовлення або службову інформацію, яка може знадобитися кухні. Такий підхід не є ідеально нормалізованою реляційною моделлю, але він зручний для MVP і добре показує зв'язок між фізичним ресурсом і бронюванням.

Окремо існують layout-дані з TableLayoutDB. Вони містять floor, x, y, shape і zone. Ці дані не потрібно змінювати щоразу після бронювання, тому вони винесені в статичний frontend-модуль. Завдяки цьому візуальне розміщення столів не залежить від оперативного статусу. Компонент карти поєднує дані API з layout-даними на клієнті: backend говорить, чи стіл вільний, а layout описує, де він розташований.

Reservation є вкладеною моделлю, яка зберігає дані конкретного бронювання (табл. 3.5). Вона містить guestName, email, phone, date, time, guests, table, guestType, memberCard і payment. Такої структури достатньо, щоб

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		49

відобразити бронювання у списку Orders, деталях бронювання, timeline і службових блоках адміністратора. У майбутньому її можна винести в окрему таблицю reservations у реляційній базі даних.

Payment у поточній версії не є повноцінною транзакційною сутністю. Він зберігає provider, status, amount і currency, тобто дані, потрібні для демонстрації депозитного правила. Для реального production-сценарію payment має містити також transactionId, createdAt, updatedAt, callback status, raw provider response і зв'язок із server-side підписом LiqPay. У бакалаврській роботі достатньо зафіксувати місце цієї інтеграції та обмеження MVP.

GuestProfile або guest CRM-дані потрібні для службового сценарію. Постійний гість у ресторанній системі не зводиться лише до номера телефону. Для персоналізованого сервісу важливі алергії, уподобання, бажана зона, історія відвідувань і нотатки адміністратора. У поточній версії ці дані можуть бути представлені демонстраційними масивами, але логічно вони належать до окремої сутності guests у майбутній базі даних.

Меню ресторану в поточному MVP зберігається як статичний MenuDB. Воно містить секції, описи та масиви items. Кожна позиція має назву, опис, ціну, зображення і бейдж. З технічного погляду меню не бере участі безпосередньо в бронюванні, але воно важливе для користувацького сценарію: гість переглядає меню перед прийняттям рішення про бронювання. У майбутній версії MenuDB можна замінити CRUD-модулем backend.

Для REST API обрано ресурсний підхід (табл 3.6). Основним ресурсом є /api/tables, оскільки саме через столи поточна система створює, показує і скасовує бронювання. Метод GET використовується для читання, POST - для створення нового table-like запису або fallback-бронювання, PATCH - для часткового оновлення статусу та reservation. Метод OPTIONS потрібен для CORS preflight-запитів. Додатковий endpoint /ready використовується для healthcheck backend.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		50

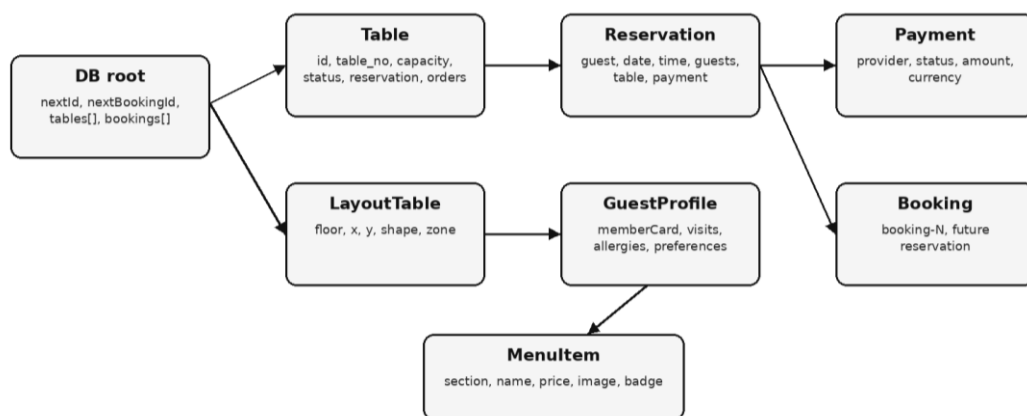


Рисунок 3.5 - Узагальнена модель даних столів, бронювань і гостей

Таблиця 3.5

Логічна структура моделі даних Crescendo

Сутність	Основні поля	Призначення
DB root	nextId, nextBookingId, tables[], bookings[]	Кореневий об'єкт JSON DB та лічильники записів
Table	id, table_no, seating_capacity, status, reservation, orders	Фізичний стіл і його поточний операційний стан
LayoutTable	floor, x, y, shape, zone	Візуальне розміщення столу на карті залу
Reservation	guestName, email, phone, date, time, guests, guestType	Дані конкретного бронювання
Payment	provider, status, amount, currency	Демонстраційний стан депозитної логіки
GuestProfile	memberCard, visits, allergies, preferences, notes	Службовий контекст постійного гостя
MenuItem	section, name, description, price, image, badge	Позиція меню, яку бачить гість

Під час проєктування API важливо було зберегти простоту endpoint-ів. Компоненти frontend не повинні знати, як саме backend читає файл, виконує нормалізацію чи атомарний запис. Вони повинні працювати з передбачуваними HTTP-операціями (рис. 3.6). Тому в src/API/Table.js винесено helper-функції TableData, TableFilterById, TableFilterByGuest, TableFilterByStatus, BookingUpdate і TableCreate. Це зменшує залежність UI-компонентів від деталей реалізації backend.

TableData виконує базове отримання списку столів. Ця функція використовується на сторінці бронювання, у Table Overview, Orders і частково в AdminPage. TableFilterByStatus потрібна для сценаріїв, де треба показати лише заброньовані або лише доступні столи. BookingUpdate виконує PATCH-запит і застосовується під час створення, оновлення або скасування бронювання. TableCreate залишено як додатковий механізм, якщо потрібно створити запис через POST.

Backend виконує нормалізацію даних перед записом. Це означає, що API не повинен безконтрольно зберігати будь-який payload, який прийшов із браузера. Для MVP достатньо приймати лише дозволені поля, перевіряти базові типи та формувати узгоджену структуру table/reservation. У production-рішенні таку перевірку варто доповнити schema validation, наприклад через Zod, Joi або аналогічний механізм.

Атомарний запис JSON DB реалізується через тимчасовий файл restik-db.json.tmp з подальшим rename. Це важливе технічне рішення для локального MVP. Якщо процес запису обірветься, основний файл має менший ризик бути пошкодженим. Звичайно, це не замінює транзакції бази даних, але є кращим варіантом, ніж прямий перезапис файлу без проміжного кроку.

REST API повертає JSON-відповіді. Такий формат природний для JavaScript-стеку і відповідає сучасній практиці обміну даними між frontend і backend [14]. Для HTTP-семантики використано стандартні методи: GET не змінює стан, POST створює запис, PATCH частково оновлює запис, OPTIONS обслуговує CORS-перевірку. Такий поділ робить API зрозумілим і придатним для подальшого документування [13].

Для endpoint-a GET /api/tables можливе використання query-параметра status. Наприклад, status=Available дозволяє отримати доступні столи, а status=Booked - заброньовані. У поточному MVP частина фільтрації додатково виконується на frontend, особливо перевірка місткості та відповідності вибраному слоту.

Особливо важливим є сценарій PATCH /api/tables/:id. Він

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

використовується для зміни статусу конкретного столу. Коли створюється бронювання, status змінюється на Booked і додається reservation. Коли бронювання скасовується, reservation очищається, а status повертається на Available. Це простий, але наочний механізм зміни операційного стану ресторану.

Таблиця 3.6

Основні REST endpoint-и backend Crescendo

Endpoint	Метод	Призначення	Приклад результату
/ready	GET	Перевірка готовності backend	{"status":"ok"}
/api/tables	GET	Отримання всіх фізичних столів і їхніх станів	Масив tables
/api/tables?status=Available	GET	Отримання доступних столів	Масив Available-столів
/api/tables?status=Booked	GET	Отримання заброньованих столів	Масив Booked-столів
/api/tables/:id	GET	Перегляд конкретного столу або запису	Об'єкт table
/api/tables	POST	Створення table-like запису або fallback-бронювання	Створений об'єкт
/api/tables/:id	PATCH	Оновлення reservation, status або скасування	Оновлений об'єкт
/api/tables	OPTIONS	CORS preflight для браузера	Дозволені методи і заголовки

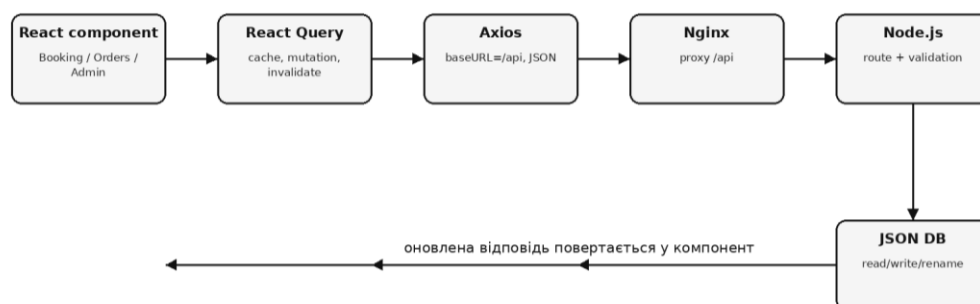


Рисунок 3.6 - Життєвий цикл REST API-запиту в Crescendo

Під час проєктування API окремо враховано обробку помилок (таб. 3.7). Користувач не повинен бачити внутрішні технічні деталі backend, однак

інтерфейс має пояснити, чому дія не виконана. Наприклад, якщо не обрано стіл, форма блокує submit ще на frontend. Якщо backend недоступний, React-компонент показує повідомлення через toast або fallback-стан. Якщо payload некоректний, API має повернути зрозумілий код помилки.

Коди помилок у REST API мають бути пов'язані зі змістом проблеми. 400 доречний для некоректного payload, 404 - для відсутнього столу або запису, 409 - для конфлікту бронювання, 500 - для непередбаченої помилки сервера. У поточній версії повна конфліктна логіка ще не реалізована, але під час опису API її варто зафіксувати як вимогу до подальшого розвитку.

Модель даних також повинна враховувати часовий аспект. Дата і час не є простими декоративними полями: саме вони визначають доступність столика. У production-системі для одного столу має існувати кілька бронювань у різні часові слоти, а конфлікти потрібно перевіряти за перетином часових інтервалів. У поточному MVP статус Booked спрощує цей процес, але структура booking-даних уже показує напрям розвитку.

Таблиця 3.7

Приклади валідації та обробки помилок

Ситуація	Де перевіряється	Очікувана реакція системи
Не введено обов'язкове поле форми	Frontend / React Hook Form	Поле підсвічується, submit не виконується
Не вибрано дату, час або кількість гостей	Frontend / TableCard	Карта не дозволяє завершити вибір столу
Стіл замалий для кількості гостей	Frontend getTableStatus	Стіл має стан too-small і не обирається
Стіл уже заброньований	Frontend + backend status	Стіл заблокований, PATCH не має створювати дубль
LiqPay не налаштовано для first visit	Frontend config check	Бронювання блокується без імітації оплати
API недоступний	React Query / Axios	Показується повідомлення про помилку або fallback-дані
Немає запису за id	Backend route	Повернення 404 або відповідного error object

Отже, модель даних і REST API у Crescendo спроектовано так, щоб забезпечити головний цикл системи: отримати столи, показати їх у карті залу, створити бронювання, оновити стан і відобразити результат у публічній та службовій частинах. Поточна JSON-модель є достатньою для локального MVP, а структура endpoint-ів залишає простір для переходу до повноцінної бази даних, окремих ресурсів guests/reservations/payments і більш суворої серверної перевірки конфліктів.

3.4 Висновки по розділу

У третьому розділі виконано проектування архітектури, бізнес-процесів, моделі даних, REST API та технологічного стеку веб-застосунку Crescendo. Система спроектована як клієнт-серверний React SPA з локальним Node.js backend, Nginx reverse proxy та JSON DB у Docker volume. Така модель дозволяє поєднати швидкий публічний інтерфейс для гостя і службову частину для адміністратора.

Архітектурна модель розділяє відповідальність між frontend, server-state шаром, API, backend і сховищем даних. Frontend відповідає за користувацькі сценарії, карта залу та форма працюють із локальним станом, React Query синхронізує дані з backend, а Node.js API зберігає операційний стан столів і бронювань. Це дає змогу використовувати один набір даних у Booking, Orders, Table Overview та AdminPage.

Під час моделювання бізнес-процесів визначено основний сценарій онлайн-бронювання: введення контактів, вибір типу гостя, дати, часу, кількості гостей, отримання доступних столів, перевірка бізнес-правил, створення reservation і оновлення кешу після API-запиту. Окремо описано сценарії скасування бронювання та роботи адміністратора з календарем, timeline, Guest CRM і Floor & kitchen.

Модель даних побудована навколо сутностей Table, Reservation, Booking,

					БР.ІІІ - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		55

Payment, GuestProfile, LayoutTable і MenuItem. Поточна JSON DB є достатньою для локального MVP, але має обмеження щодо транзакцій, конкурентності та історії змін. Тому для production-версії передбачено розвиток у напрямі повноцінної бази даних із окремими таблицями guests, reservations, payments, users і roles.

REST API спроектовано на основі ресурсу /api/tables. Через нього frontend отримує столи, фільтрує їх за статусом, відкриває деталі, створює або оновлює бронювання та скасовує записи. Такий API є простим для поточного прототипу і водночас залишає можливість розширення на окремі ресурси гостей, платежів, меню та серверної авторизації.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ CRESCENDO

4.1. Реалізація клієнтської частини

Клієнтська частина веб-застосунку Crescendo реалізована як односторінковий React-застосунок, у якому публічні сторінки, форма бронювання, інтерактивна карта залу та службові екрани адміністратора працюють у межах спільного інтерфейсного середовища. Такий підхід дає змогу не розривати користувацький сценарій між різними сайтами або зовнішніми сервісами: гість може перейти від головної сторінки до меню, бронювання, плану столів і перегляду активних резервувань без повного перезавантаження сторінки.

Основою клієнтської частини є компонент App, який підключає загальні провайдери застосунку. QueryClientProvider забезпечує роботу TanStack React Query та синхронізацію даних з backend, ToastContainer відповідає за повідомлення про успішні й помилкові дії, а AuthContext зберігає демонстраційний стан входу адміністратора. Завдяки цьому службова логіка не дублюється в кожному окремому компоненті, а доступна всім сторінкам через спільний контекст.

Компонування сторінок виконується через Layout. Він об'єднує NavBar, PageRoutes і Footer, тобто формує стабільний каркас застосунку. У межах цього каркаса змінюється тільки основний вміст сторінки, а навігація та службові елементи залишаються однаковими. Це важливо для зручності користувача, оскільки він завжди бачить однакову логіку переходів: Home, Menu, Reserve, Floor plan, Admin, Book та Reservations.

Маршрутизацію винесено до PageRoutes. Для основних сторінок застосовано React Router DOM v6, а частину сторінок підключено через React.lazy і Suspense. Таке рішення дозволяє розділяти застосунок на логічні модулі та завантажувати окремі екрани лише тоді, коли вони потрібні користувачу. Для MVP це також корисно з погляду структури коду: сторінки

					БР.ІІІ - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

HomePage, MenuPage, BookingPage, Orders, OrderDetail, TableOverview, AdminPage і Login мають окремі межі відповідальності (табл. 4.1).

Головна сторінка виконує роль публічної вітрини ресторану (рис. 4.1). Вона складається з блоків Home, About, RestaurantSystem, GuestExperience, BookingBanner і Menu. Через ці блоки користувач бачить бренд Crescendo, опис ресторанного сервісу, цінність онлайн-бронювання, коротке пояснення роботи guest card і депозитної політики. Важливо, що перший екран не обмежується інформаційним текстом, а одразу веде до практичних дій — перегляду меню або бронювання столика.

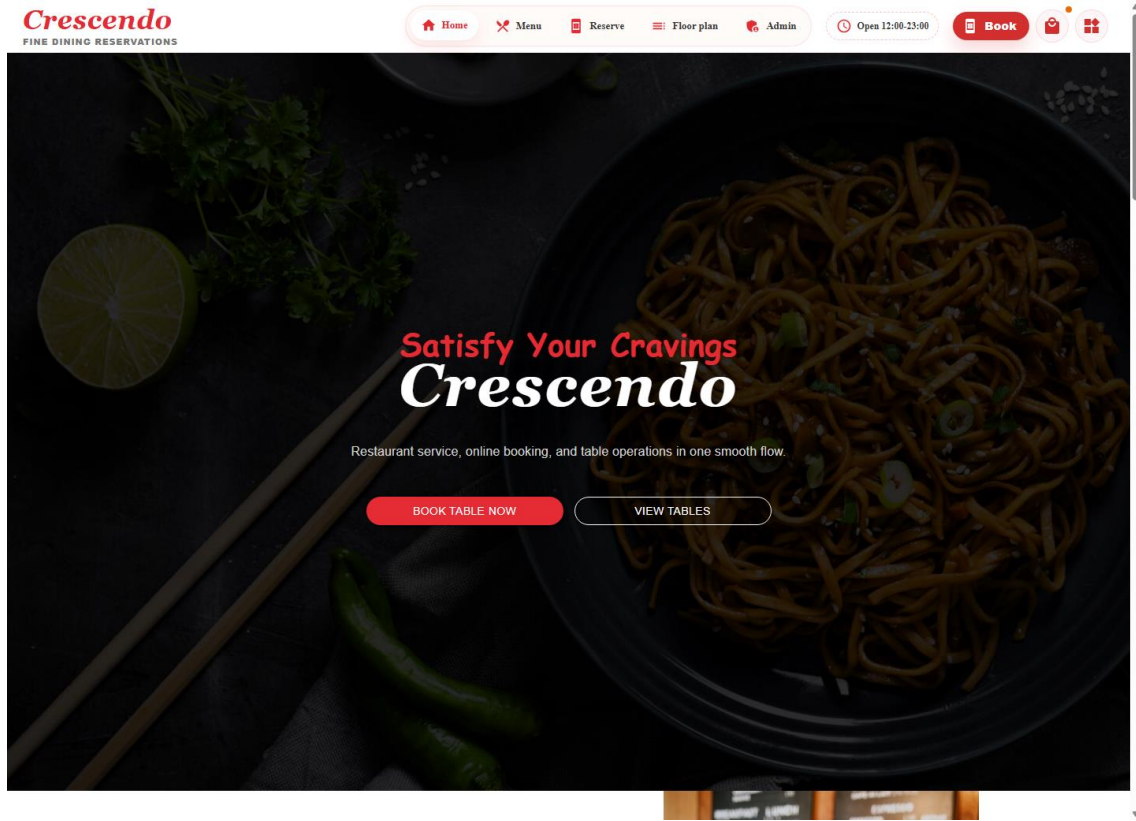


Рисунок 4.1 - Головна сторінка веб-застосунку Crescendo

Навігаційна панель NavBar реалізує основні переходи між функціональними частинами системи. У ній виводяться бренд, посилання на сторінки, службова інформація про години роботи, телефон, кнопка Book і піктограма Reservations. Активний маршрут визначається через location.pathname, тому користувач бачить, у якому розділі перебуває. Це

зменшує ризик втрати контексту під час перемикання між публічними і службовими сторінками.

Сторінка MenuPage працює зі статичним модулем MenuDB. У ньому описано дев'ять секцій меню та тридцять шість позицій страв. Кожна позиція містить назву, опис, ціну, бейдж і зображення. Для користувача це створює повноцінне уявлення про ресторан до моменту бронювання, а для розробки дає просту й контрольовану модель даних, яку в майбутньому можна замінити на backend CRUD-модуль.

Візуальна стилістика клієнтської частини побудована на поєднанні ресторанної тематики, контрастних акцентів і зрозумілої композиції сторінок. CSS-файли компонентів відповідають за конкретні блоки інтерфейсу, а глобальні правила в index.css задають базові шрифти, заголовки, відступи та адаптивну поведінку. Такий поділ спрощує підтримку стилів: зміни в одному компоненті не потребують переписування всієї системи.

Окрему роль відіграє адаптивність. Оскільки гість може відкривати сайт як з комп'ютера, так і зі смартфона, основні сторінки повинні зберігати читабельність і логічну послідовність на різних екранах. У реалізації використано гнучкі контейнери, адаптивні сітки, компонентний підхід Material UI та CSS-правила для перебудови блоків. Це дає змогу зберегти основний сценарій: перегляд меню, вибір дати, часу, кількості гостей і столика.

Клієнтська частина не є лише статичною презентацією ресторану. Вона взаємодіє з backend через /api/tables, отримує актуальний стан столів, передає дані бронювання та оновлює інтерфейс після зміни стану. Завдяки цьому React-компоненти показують не тільки візуальний макет, а й реальний операційний контекст системи.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

Основні компоненти клієнтської частини Crescendo

Компонент або сторінка	Роль у застосунку	Практичне значення
App.js	Підключення QueryClientProvider, ToastContainer і AuthContext	Формує спільний рівень роботи з API, повідомленнями та demo-auth
Layout	Об'єднання NavBar, PageRoutes і Footer	Забезпечує єдину структуру для всіх публічних і службових екранів
NavBar	Головна навігація, активні маршрути, кнопки Book і Reservations	Дає користувачу швидкий доступ до ключових сценаріїв
HomePage	Публічна вітрина ресторану, СТА, опис сервісу	Пояснює цінність системи і переводить гостя до бронювання
MenuPage	Відображення MenuDB, секцій і карток страв	Дає змогу переглянути ресторанне меню до створення бронювання
BookingPage	Контейнер для форми бронювання і карти залу	Запускає головний користувацький сценарій системи
TableOverview	Окремий перегляд плану столів і зон ресторану	Корисний для гостя та працівника, який оцінює завантаження залу
Orders / OrderDetail	Список активних бронювань і деталі запису	Підтверджує, що бронювання збережене й може бути скасоване
AdminPage	Службова панель для ресторанного персоналу	Показує календар, гостей, timeline, notes і Floor & kitchen

Розподіл клієнтської частини на окремі компоненти дозволяє підтримувати принцип локальної відповідальності. Кожен модуль відповідає за власну частину сценарію: MenuPage працює з меню, TableCard — з бронюванням, Orders — з переглядом бронювань, а AdminPage — зі службовими екранами. Це важливо для дипломного проєкту, оскільки демонструє не лише кінцевий інтерфейс, а й зрозумілу організацію коду.

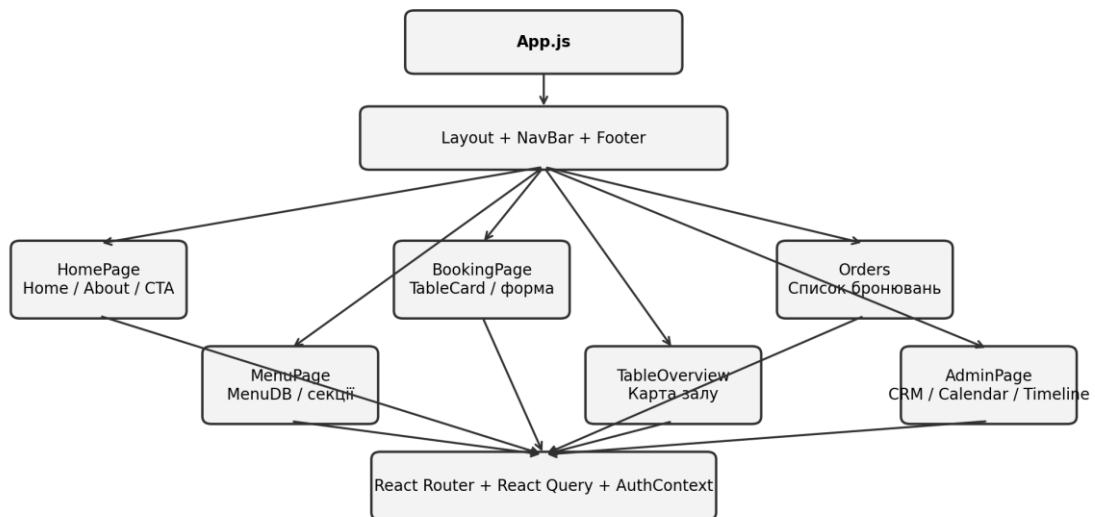


Рисунок 4.2 - Структура реалізації клієнтських модулів Crescendo

На рисунку 4.2 показано логіку взаємодії клієнтських модулів. App.js формує верхній рівень застосунку, Layout утримує спільну оболонку, а PageRoutes розподіляє користувача між функціональними сторінками. Таке розділення спрощує подальше розширення: у майбутньому можна додати окрему сторінку аналітики, налаштування ресторану або повноцінний кабінет адміністратора без зміни базової структури застосунку.

4.2. Реалізація бронювання, карти залу та адміністративної підсистеми

Центральним елементом реалізації бронювання є компонент TableCard. Він поєднує кілька функцій, які в простіших системах часто рознесені по різних екранах: введення контактів, вибір типу гостя, вибір дати й часу, введення кількості гостей, відображення карти залу, перевірку місткості, перевірку статусу столу та надсилання reservation-даних до backend. Завдяки такому об'єднанню користувач проходить сценарій бронювання послідовно і не переходить між зайвими сторінками.

Форма бронювання побудована на логічній послідовності кроків (рис.

4.3). Спочатку користувач вводить ім'я, email і телефон. Далі він обирає режим First visit або Guest card. Після цього задаються дата, час і кількість гостей. Тільки після вибору цих параметрів карта залу може коректно оцінити доступність столів, тому що стіл має значення лише в конкретному часовому слоті та для конкретної кількості людей.

Для нового гостя активується депозитна логіка першого візиту. У поточній версії LiqPay не виконує production-платіж, але конфігураційний шар уже передбачений. Якщо ключі або sign endpoint не налаштовано, система не імітує успішну оплату. Це важливе обмеження MVP: користувач не повинен отримувати помилкове враження, що депозит справді списаний або зарезервований.

Для постійного гостя використовується сценарій Guest card. У цьому режимі форма вимагає номер гостьової картки, а депозит позначається як waived або member_card. Такий підхід реалізує бізнес-правило ресторану: постійний клієнт має спрощені умови бронювання, але система все одно повинна мати ідентифікатор гостя для службової частини та майбутнього CRM-зв'язку.

Інтерактивна карта залу побудована на статичній структурі RESTAURANT_TABLES. Для кожного столика вказано table_no, seating_capacity, floor, x, y, shape і zone. Координати задаються у відсотках, тому карта може масштабуватися разом з контейнером. Це практично зручно: візуальне розміщення столів не залежить від фіксованої ширини екрана, а дані можна редагувати без переписування всієї логіки компонента.

Операційний стан столів приходить із backend через TableData. Після отримання відповіді /api/tables функція mergeTablesWithLayout поєднує фізичний опис столів із поточними status, reservation і orders. Якщо backend тимчасово не повертає конкретний запис, компонент може показати layout-версію столу зі статусом Available. Це дає змогу зберегти карту залу навіть у демонстраційних або частково заповнених даних.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		62

Доступність столу визначається функцією `getTableStatus`. Вона враховує три ключові умови: чи стіл уже заброньований, чи його місткість достатня для вказаної кількості гостей, і чи користувач обрав усі потрібні параметри слота. На основі цієї перевірки UI показує стани `available`, `booked`, `too-small`, `preview` або `selected`. Така логіка блокує неправильний вибір ще на рівні інтерфейсу [33].

Після підтвердження форми формується `reservation`-об'єкт. Він містить дані гостя, дату, час, кількість гостей, номер столу, тип гостя, номер гостьової картки, `payment`-стан і службові поля, потрібні для `Orders` та `AdminPage`. Якщо оновлюється наявний фізичний стіл, `frontend` надсилає `PATCH /api/tables/:id`. Якщо необхідний `fallback`-сценарій, використовується `POST /api/tables` [35].

Після успішної мутації `React Query` виконує інвалідацію кешу `tables/bookings`. Це означає, що інтерфейс не покладається на застарілі локальні дані, а повторно отримує актуальний стан із `backend`. Завдяки цьому стіл одразу відображається як заброньований у карті залу, у списку `Orders` та на службових екранах адміністратора [34].

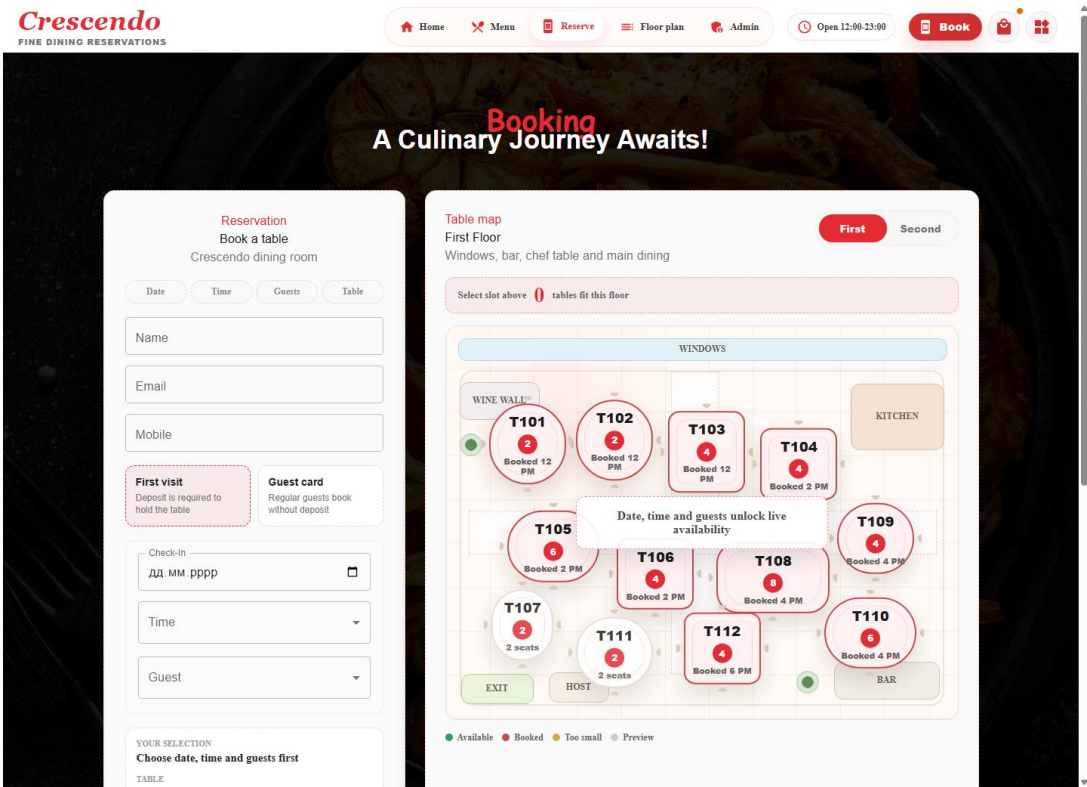


Рисунок 4.3 - Форма бронювання та інтерактивна карта залу

Стани столика на карті залу та реакція інтерфейсу

Стан	Умова появи	Дія користувача	Реакція системи
Available	Стіл вільний, слот обраний, місткість достатня	Може натиснути на стіл	Стіл стає selected і записується у форму
Booked	Статус столу Booked або є активне reservation	Не може обрати	UI блокує вибір і показує зайнятий стан
Too-small	Кількість гостей перевищує seating_capacity	Не може обрати	Стіл позначається як невідповідний за місткістю
Preview	Користувач ще не задав дату, час або кількість гостей	Може переглядати карту	Submit не завершується без повного слота
Selected	Стіл відповідає умовам і вибраний користувачем	Може підтвердити бронювання	У reservation додається table_no і table id
Cancelled	Після cancel action бронювання очищене	Може бути обраний повторно	PATCH повертає status Available і очищає reservation

Таблиця 4.2 показує, що карта залу не є декоративним елементом. Вона виконує функцію контролю якості бронювання. Користувач бачить причину, чому певний столик не можна обрати, а система не дозволяє створити запис із порушенням базових бізнес-правил ресторану.

Алгоритм на рисунку 4.4 відображає послідовність створення бронювання. Спочатку перевіряються контактні дані, потім тип гостя і депозитна політика, після цього формується часовий слот і отримуються столи через API. Лише після успішної перевірки доступності система надсилає PATCH або POST-запит і оновлює кеш React Query.

Особливості реалізації основних бізнес-режимів бронювання наведено в табл. 4.3.

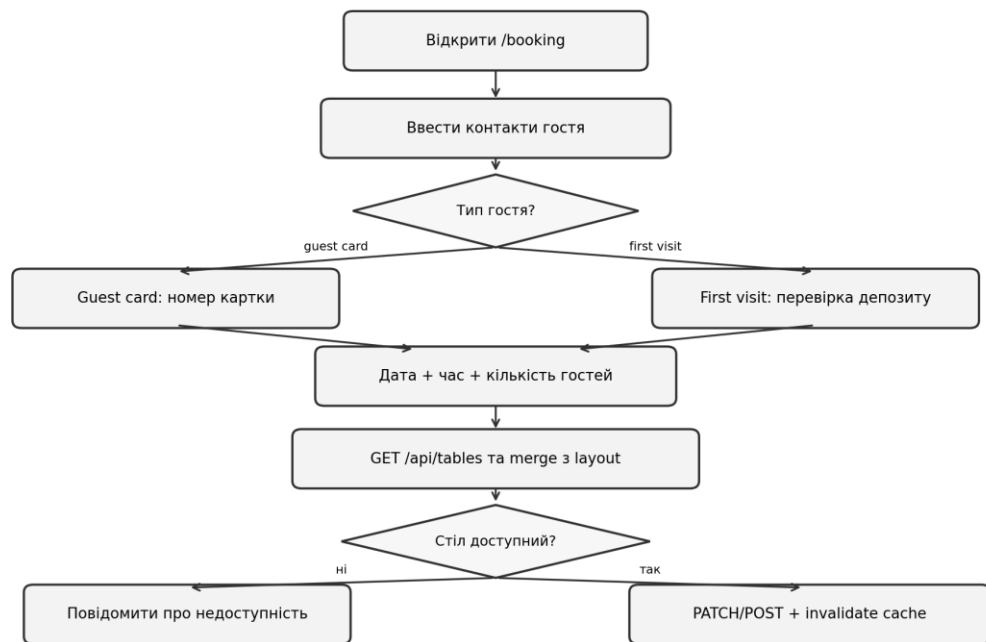


Рисунок 4.4 - Алгоритм створення бронювання у компоненті TableCard

Таблиця 4.3

Реалізація бізнес-режимів бронювання

Режим	Дані, які вводить користувач	Правило системи	Результат
First visit	Ім'я, email, телефон, дата, час, guests	Потрібна перевірка LiqPay-конфігурації	Без готового payment sign endpoint бронювання не імітує оплату
Guest card	Контакти, дата, час, guests, memberCard	Номер картки є обов'язковим	Депозит позначається як waived/member_card
Зміна параметрів слота	Дата, час або кількість гостей змінені	Попередній вибір столу очищається	Карта перераховує доступність столів
Створення бронювання	Reservation payload і selected table	PATCH /api/tables/:id або POST fallback	Стан столу змінюється на Booked
Скасування	Дія cancel у деталях або Orders	Reservation очищується	Стіл повертається в Available

Адміністративна підсистема реалізована в AdminPage. Вона доступна

після демонстраційного входу через AuthContext і localStorage. У межах бакалаврського MVP такий механізм потрібен не для production-безпеки, а для демонстрації розмежування публічної та службової частини. Гість бачить меню, бронювання і план столів, а адміністратор отримує доступ до календаря, CRM-гостей, таймлайну сервісу, нотаток і підготовки залу.

AdminPage працює з seeded booking-даними backend. У демонстраційному наборі використано 24 фізичні столи, 46 booking-записів на 11 дат, зокрема 16 записів на 03.06.2026. Ці дані дають змогу показати не порожню службову панель, а реальний сценарій ресторанного дня: кількість гостей, депозити, завантаженість за днями, час приходу гостей і підготовку столів.

Вкладка Calendar показує загальну картину завантаження за тиждень. Timeline групує бронювання за часом прибуття, щоб адміністратор бачив послідовність сервісу. Guest CRM відображає інформацію про постійних гостей, алергії, уподобання й історію. Guest notes перетворює ці дані на короткі службові підказки, а Floor & kitchen з'єднує зал і кухню через столи до підготовки та попередні замовлення (рис. 4.5).

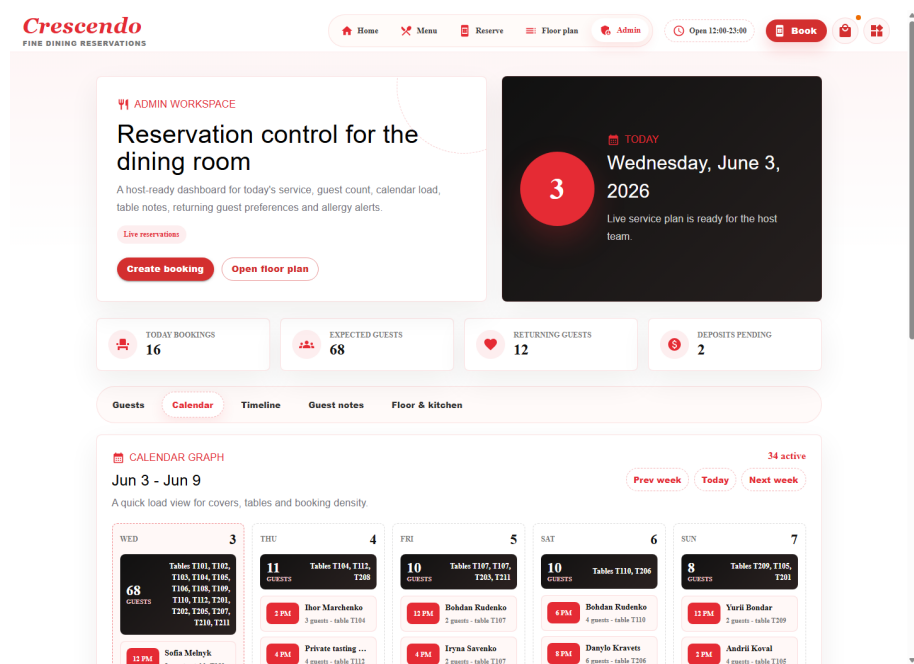


Рисунок 4.5 - Адміністративна панель з календарем і Guest CRM

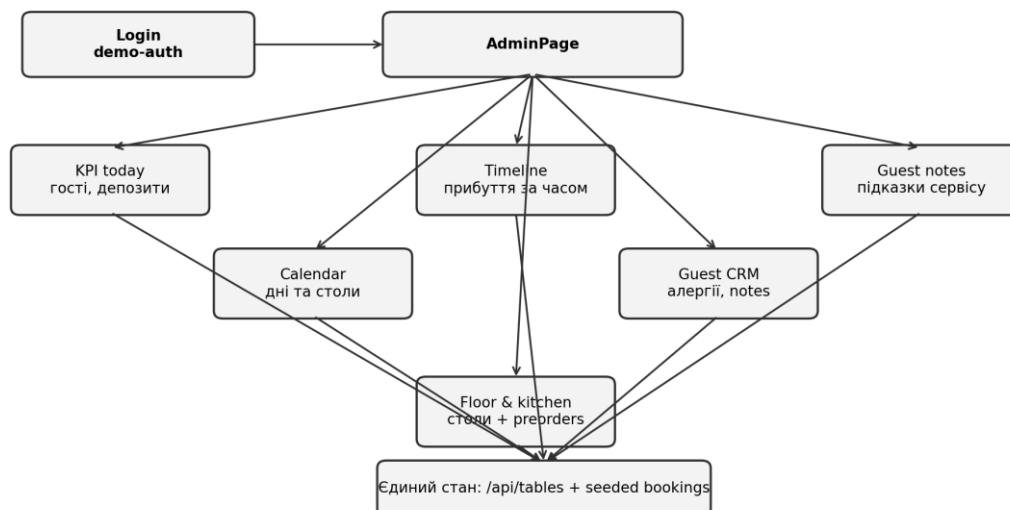


Рисунок 4.6 - Взаємодія адміністративних вкладок зі спільним операційним станом

Рисунок 4.6 демонструє, що адміністративні вкладки не є ізольованими сторінками. Вони працюють із тим самим операційним станом, який змінюється під час бронювання або скасування. Це важливо для цілісності системи: якщо стіл заброньовано у публічній формі, адміністратор має побачити його в календарі, timeline і списку підготовки без ручного перенесення даних.

4.3. Організація програмного коду, алгоритми та структури даних

Організація програмного коду побудована за функціональним принципом. Сторінки маршрутизації розміщено в `src/pages`, повторно використовувані компоненти — у `src/components`, функції взаємодії з backend — у `src/API`, локальні набори даних — у `src/DB`, конфігураційні параметри LiqPay — у `src/config`, а контекст і сторінки авторизації — у `src/Validate`. Серверна частина винесена в окремий каталог `backend`.

Такий поділ має практичне значення для підтримки дипломного проєкту. Якщо потрібно змінити логіку меню, розробник працює з `MenuDB` і `MenuPage`. Якщо потрібно змінити бронювання, основний фокус переноситься на `TableCard`, `API helpers` і `backend /api/tables`. Якщо потрібно допрацювати

службову частину, зміни вносяться в AdminPage і пов'язані компоненти. Це зменшує кількість перехресних залежностей.

Файл src/API/api.js створює базовий Axios instance. У Docker-версії baseUrl задається як /api або через REACT_APP_API_BASE_URL. Завдяки цьому frontend не повинен знати внутрішню адресу backend-контейнера. Браузер звертається до Nginx, а Nginx прокидає /api-запити до Node.js backend. Такий підхід спрощує локальне розгортання і робить структуру ближчою до production-сценарію.

Файл src/API/Table.js містить helper-функції для роботи зі столами і бронюваннями. TableData отримує повний список столів, TableFilterByStatus допомагає фільтрувати записи, BookingUpdate виконує PATCH-запит, а TableCreate використовується для POST-сценарію (табл. 4.5). Винесення цих функцій в окремий модуль запобігає дублюванню endpoint-ів у UI-компонентах.

Дані меню зберігаються у структурі MenuDB. Вона містить секції меню, описи секцій і масиви items. У MenuPage вкладені items можуть перетворюватися у плоский список allItems, коли користувач обирає режим All Dishes. Такий підхід дає змогу одночасно підтримувати фільтрацію за секціями і загальний перегляд усіх позицій.

Дані карти залу описані у RESTAURANT_TABLES. Це статична модель простору ресторану, яка зберігає не статус, а геометрію та характеристики столів: номер, поверх, координати, форму, зону і місткість. Операційний стан — Available або Booked — надходить із backend. Саме поєднання цих двох джерел дає інтерактивну карту, яка відображає і простір, і стан бронювань.

Backend реалізовано у файлі backend/server.js. Він працює на Node.js 20 і використовує native http, fs та url modules без Express. Сервер обробляє REST API routes, CORS, healthcheck /ready, нормалізацію JSON DB, базову валідацію payload і атомарний запис даних. Атомарний запис виконується через тимчасовий файл із подальшим rename, що зменшує ризик пошкодження JSON-файлу під час перезапису.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл backend/seed-demo.js створює демонстраційні дані. Seed потрібен для того, щоб після запуску Docker користувач бачив реалістичний стан системи: фізичні столи, бронювання на різні дати, дані для календаря і службових екранів. Без такого seed-набору AdminPage виглядала б порожньою, а тестування сценаріїв потребувало б ручного створення великої кількості записів.

Docker-оточення складається з web-сервісу та backend-сервісу. Web-контейнер збирає React-застосунок і віддає production-збірку через Nginx, а backend-контейнер запускає Node.js API. JSON-файл бази даних зберігається у Docker volume backend-data, тому дані демонстраційного backend не втрачаються після перезапуску контейнера.

Таблиця 4.4

Організація програмного коду Crescendo

Файл/модуль	Призначення	Ключові рішення
src/App.js	Підключення провайдерів застосунку	QueryClientProvider, ToastContainer, AuthContext
src/Routes/PageRoutes.jsx	Клієнтська маршрутизація	React.lazy, Suspense, сторінки Home/Menu/Booking/Admin
src/API/api.js	Базовий HTTP-клієнт	Axios baseURL = /api або REACT_APP_API_BASE_URL
src/API/Table.js	API helpers для столів і бронювань	TableData, TableFilterByStatus, BookingUpdate, TableCreate
src/components/TableCard /TableCard.jsx	Основна логіка бронювання	Форма, карта столів, submit, guest card/deposit
src/DB/MenuDB.js	Статичне меню ресторану	9 секцій, 36 позицій, опис, ціна, бейджі, фото
src/DB/TableLayoutDB.js	Статична модель плану залу	floor, x, y, shape, zone, seating_capacity
backend/server.js	Node.js backend	REST API, CORS, JSON DB, validation, atomic write
backend/seed-demo.js	Демонстраційні дані	24 столи, 46 booking-записів на 11 дат
docker-compose.yml / nginx.conf	Локальне розгортання	web/backend сервіси, volume, healthcheck, /api proxy

У таблиці 4.4 узагальнено карту файлів, яка частково переносить у розділ реалізації технічні відомості з додатків. Це зроблено для того, щоб текст розділу був самодостатнім: читач бачить не лише опис функцій, а й конкретні файли, у яких ці функції реалізовано.

Таблиця 4.5

Основні алгоритми та helper-функції реалізації

Алгоритм або функція	Вхідні дані	Результат	Де використовується
mergeTablesWithLayout	tables з API, RESTAURANT_TABLES	Об'єднаний список столів із геометрією та статусом	TableCard, TableOverview
getTableStatus	table, selectedSlot, guests, selectedTable	available / booked / too-small / selected	Карта залу в TableCard
TableData	HTTP GET /api/tables	Масив фізичних столів	Booking, Orders, Admin
BookingUpdate	id столу, reservation payload	Оновлений table object	Створення і скасування бронювання
React Query invalidate	query keys tables/bookings	Оновлений кеш після мутації	Після POST/PATCH
normalizeDb	JSON DB object	Узгоджена структура nextId, tables, bookings	backend/server.js
atomic write	оновлена JSON DB	Безпечніший запис через tmp + rename	backend/server.js
seed-demo	початковий сценарій демо	24 столи і booking-дані	локальний Docker запуск

Алгоритм вибору столика можна описати як послідовну фільтрацію. Спочатку користувач задає дату, час і кількість гостей. Потім система отримує столи через API та накладає на них layout-дані. Далі для кожного столика

визначається статус: чи він зайнятий, чи достатня місткість, чи доступний він для конкретного слота. Після успішного вибору формується reservation payload і надсилається до backend.

Для підтримки цілісності даних важливо, що UI не змінює стан столу самостійно як остаточне джерело істини. Він надсилає запит до backend, очікує відповідь і тільки після цього оновлює кеш. Така модель зменшує ризик розсинхронізації між формою бронювання, списком Orders, TableOverview і AdminPage.

Структура JSON DB у поточному MVP є спрощеною, але достатньою для демонстрації. У ній є кореневий об'єкт із nextId, nextBookingId, tables і bookings. Сутність Table містить table_no, seating_capacity, status, reservation і orders. Reservation зберігає дані гостя, дату, час, кількість гостей, тип гостя й payment-стан. Для production-версії цю структуру доцільно перенести в реляційну базу даних із таблицями guests, reservations, tables, payments, users і roles.

4.4. Висновки по розділу

У четвертому розділі було розглянуто практичну реалізацію веб-застосунку Crescendo, призначеного для автоматизації роботи ресторану та організації онлайн-бронювання столиків. Реалізована клієнтська частина побудована як односторінковий React-застосунок, у якому поєднано публічні сторінки ресторану, меню, форму бронювання, інтерактивну карту залу, перегляд резервувань і службову адміністративну панель. Така структура дозволила сформувати єдиний користувацький сценарій без зайвих переходів між різними сервісами. Особливу увагу приділено компонентній організації інтерфейсу, адаптивності сторінок, зручній навігації та поділу відповідальності між окремими модулями застосунку.

У процесі реалізації основний функціональний акцент було зроблено на бронюванні столиків. Система дає змогу користувачу ввести контактні дані,

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

обрати тип гостя, дату, час, кількість осіб і конкретний столик на карті залу. Інтерактивна карта виконує не лише візуальну, а й логічну функцію: вона перевіряє статус столу, його місткість і можливість вибору в межах заданого часового слота. Реалізація станів available, booked, too-small, preview, selected і cancelled забезпечує зрозумілу реакцію інтерфейсу на дії користувача та зменшує ймовірність помилкового бронювання. Окремо реалізовано бізнес-режими First visit і Guest card, що дозволяє враховувати депозитну політику ресторану та спрощені умови для постійних клієнтів.

Також у розділі було описано адміністративну підсистему, яка демонструє службову сторону роботи ресторану. Адміністратор отримує доступ до календаря бронювань, таймлайну прибуття гостей, CRM-інформації, нотаток і підготовки залу. Важливо, що адміністративні вкладки працюють із тим самим операційним станом, що й публічна форма бронювання. Завдяки цьому зміни, внесені користувачем під час створення або скасування бронювання, можуть відображатися в службових екранах без ручного дублювання даних. Це підвищує цілісність системи та робить її придатною для демонстрації реального ресторанного процесу.

Окрему увагу приділено організації програмного коду та взаємодії frontend із backend. API-запити винесено в окремі helper-модулі, що спрощує підтримку коду та зменшує дублювання логіки в компонентах. Серверна частина реалізована на Node.js із використанням JSON DB, базової валідації даних, REST API routes, CORS, healthcheck і атомарного запису файлу. Для локального запуску передбачено Docker-оточення з окремими web- і backend-сервісами, Nginx-проксі та volume для збереження демонстраційних даних. Це дозволяє запускати застосунок у контрольованому середовищі та перевіряти його роботу як єдину систему.

Отже, у межах четвертого розділу було підтверджено, що веб-застосунок Crescendo реалізує ключові функції, необхідні для демонстраційної системи онлайн-бронювання ресторану: перегляд інформації про заклад, меню, вибір параметрів бронювання, роботу з картою залу, створення й скасування

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

резервувань, а також службовий контроль через адміністративну панель. Поточна реалізація відповідає рівню MVP і створює основу для подальшого розвитку системи. У майбутньому її можна розширити повноцінною авторизацією, production-інтеграцією LiqPay, переходом від JSON DB до реляційної бази даних, ролями користувачів, аналітикою завантаженості ресторану та більш гнучким керуванням меню, гостями й бронюваннями.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

РОЗДІЛ 5. СИСТЕМНЕ ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

5.1. Стратегія та методика тестування системи

Системне тестування веб-застосунку Crescendo розглядається як завершальний етап перевірки, на якому оцінюється не окремий компонент, а весь сценарій роботи ресторанної системи. Для такого проєкту недостатньо встановити, що сторінки відкриваються або що форма візуально відображається. Потрібно перевірити, чи може користувач пройти повний шлях від відкриття головної сторінки до створення бронювання, а адміністратор — побачити це бронювання у службовій частині застосунку.

Стратегія тестування побудована за принципом поступового розширення зони перевірки (табл. 5.1). Спочатку перевіряються базові умови запуску середовища: доступність Docker-контейнерів, відповідь backend на /ready, робота Nginx і відкриття SPA-маршрутів. Після цього оцінюються окремі модулі: меню, форма бронювання, карта залу, сторінка Orders, деталі бронювання, авторизація адміністратора та вкладки AdminPage.

Наступний рівень перевірки стосується взаємодії frontend і backend. У межах цього рівня важливо підтвердити, що TableData отримує список фізичних столів, TableFilterByStatus повертає записи за статусом, BookingUpdate змінює стан бронювання через PATCH, а TableCreate може створити додатковий table-like запис через POST. Усі ці дії мають завершуватися оновленням кешу React Query, інакше користувач бачитиме застарілий стан карти або списку бронювань.

Окремо перевіряється логіка доступності столів. Система повинна блокувати стіл, якщо він уже має статус Booked, якщо його місткість менша за кількість гостей або якщо користувач не вказав дату, час і кількість осіб. Така перевірка є важливою, тому що в ресторанній системі помилка вибору столика має не лише інтерфейсний, а й операційний наслідок: адміністратор може

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		74

отримати некоректну інформацію про підготовку залу.

Методика тестування також включає перевірку службового сценарію. Адміністративна панель має показувати календар, таймлайн сервісу, CRM гостей, нотатки та блок Floor & Kitchen на основі демонстраційних booking-даних. У Docker-версії backend містить 46 booking-записів на 11 дат, зокрема 16 записів на 03.06.2026. Це дає змогу перевірити не порожній інтерфейс, а реальне заповнення календаря й операційних вкладок.

Для оцінки якості тестування використано критерії приймання, сформовані в розділі 2. До них належать доступність основних сторінок, коректна робота форми, відображення станів карти залу, створення й скасування бронювання, робота REST API, збереження стану в JSON DB, коректне розгортання через Docker/Nginx і розмежування публічної та адміністративної зон.

Тестування проводиться не як одноразова дія, а як послідовність перевірок після кожної важливої зміни (рис. 5.1). Наприклад, після зміни логіки карти потрібно повторно перевірити бронювання, Orders і AdminPage, оскільки всі ці частини використовують спільні дані tables/bookings. Такий підхід зменшує ризик регресії, коли виправлення одного модуля випадково порушує роботу іншого.



Рисунок 5.1 - Логіка системного тестування веб-застосунку Crescendo

Матриця методики тестування системи

Об'єкт перевірки	Зміст перевірки	Критерій приймання
Docker-середовище	Запуск web і backend контейнерів, доступність портів 8080 і 8001, перевірка /ready.	Nginx повертає сторінку SPA, backend повертає status ok.
Клієнтська маршрутизація	Відкриття Home, Menu, Booking, Orders, Table Overview, Admin, Login і Logout.	Маршрути відкриваються без помилки 404 завдяки SPA fallback.
Форма бронювання	Перевірка обов'язкових полів, типу гостя, дати, часу, кількості гостей і submit-логіки.	Некоректні дані блокуються, коректні — формують reservation model.
Карта залу	Перевірка двох поверхів, зон, місткості, статусів available/booked/too-small/preview.	Користувач може обрати лише доступний і відповідний за місткістю стіл.
REST API	GET /api/tables, GET з query status, GET by id, POST, PATCH, OPTIONS.	Запити повертають JSON і змінюють стан відповідно до сценарію.
Адміністративна панель	Calendar, Timeline, Guest CRM, Guest notes, Floor & Kitchen.	Адміністратор бачить службові дані й може планувати підготовку сервісу.
Надійність даних	Оновлення JSON DB і повторне отримання даних через React Query.	Після створення або скасування бронювання UI показує актуальний стан.

Отже, стратегія тестування охоплює не лише формальну перевірку сторінок, а повний цикл поведінки системи. Це важливо для Crescendo, оскільки застосунок поєднує публічний інтерфейс, карту залу, REST API, локальне сховище даних і службову панель. Якщо хоча б один із цих рівнів працює некоректно, користувач або адміністратор отримує неповну картину ресторанного сервісу.

Особливу увагу приділено зв'язку між тестуванням і майбутнім впровадженням. Поточна версія є MVP, тому тестування має показати не лише те, що прототип працює, а й те, які місця потребують посилення перед production-експлуатацією. До таких місць належать серверна авторизація, транзакційна база даних, перевірка конфліктів часових слотів, реальний платіжний підпис і резервне копіювання.

5.2. Результати функціонального, інтеграційного та frontend/backend-тестування

Функціональна перевірка підтвердила, що у веб-застосунку реалізовано основні сторінки та сценарії, передбачені вимогами (табл. 5.2). Користувач може перейти на головну сторінку, переглянути меню, відкрити форму бронювання, вибрати параметри візиту, взаємодіяти з картою залу, створити бронювання, переглянути активні записи та відкрити деталі конкретного бронювання.

Таблиця 5.2

Результати функціональної перевірки

Перевірка	Очікуваний результат	Фактичний результат
Відкриття Docker-застосунку	HTTP 200 від Nginx на порту 8080, завантаження SPA.	Виконано: web-контейнер віддає сторінку застосунку.
Backend healthcheck	GET /ready повертає status ok.	Виконано: backend підтверджує готовність до API-запитів.
Головна сторінка	Видимі навігація, бренд, СТА до бронювання та плану столів.	Виконано: користувач отримує швидкий доступ до ключових сценаріїв.
Меню ресторану	Показ секцій, карток страв, фільтрація за категоріями.	Виконано: MenuDB коректно перетворюється у список позицій.
Форма бронювання	Валідація полів, вибір типу гостя, дати, часу, кількості гостей.	Виконано: submit блокується без обов'язкових даних.
Карта залу	Відображення двох поверхів і статусів столів.	Виконано: доступні, заброньовані та замалі столи мають різні стани.
Orders	Показ активних бронювань і перехід до деталей.	Виконано: список використовує дані backend.
AdminPage	Доступ після demo-login, показ календаря, CRM і операційних вкладок.	Виконано: службовий сценарій доступний для демонстрації.

Перевірка головної сторінки показала, що вона виконує роль вхідної точки до застосунку. Вона не обмежується презентацією ресторану, а містить навігацію до основних дій: перегляду меню, бронювання та плану столів. Це відповідає логіці ресторанного сервісу, де користувач зазвичай приходить на сайт не лише для ознайомлення, а й для виконання конкретної дії.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		77

Модуль меню коректно працює з даними MenuDB. Страви відображаються за секціями, а режим All Dishes формує загальний список усіх позицій. Перевірка показала, що структура меню придатна для розширення: у майбутньому статичний модуль можна замінити backend CRUD без зміни основної логіки виведення карток.

Найважливішим результатом функціональної перевірки є працездатність модуля бронювання. Форма реагує на помилки валідації, не дозволяє submit без вибраного столика, очищає попередній вибір після зміни дати, часу або кількості гостей і розрізняє сценарії First visit та Guest card. Завдяки цьому користувач не може випадково створити запис без основних параметрів.

Перевірка карти залу підтвердила наявність 24 фізичних столів на двох поверхах. Для першого поверху передбачено зони Window, Main hall, Family, Bar, Chef table, Garden view, Entrance і Quiet corner, для другого — Balcony, Lounge, Private, Banquet і Gallery. Це важливо, оскільки карта залу має відображати не абстрактний список столів, а реальну логіку простору ресторану.

Інтеграційна перевірка frontend/backend показала, що клієнтська частина працює не тільки зі статичними даними (табл. 5.3). Через Axios і React Query застосунок звертається до /api/tables, отримує список столів, фільтрує їх за статусом, оновлює записи через PATCH і перезавантажує дані після зміни стану. У результаті Booking, Orders, Table Overview і AdminPage використовують узгоджене джерело даних.

Backend-перевірка підтвердила доступність /ready, роботу GET /api/tables, фільтрацію за status=Available або status=Booked, отримання конкретного запису за id, створення table-like запису через POST та оновлення бронювання через PATCH. Для MVP цього достатньо, щоб показати повний цикл зміни стану ресторанного ресурсу.

Службовий сценарій адміністратора також був перевірений із seeded booking-даними. Адміністративна панель отримує заповнений календар,

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		78

таймлайн сервісу, профілі гостей, нотатки та підготовку залу. Це дозволяє продемонструвати, що система не завершується на моменті створення бронювання, а підтримує роботу персоналу перед приходом гостей.

Таблиця 5.3

Перевірка інтеграції frontend, REST API та backend

Інтеграційний сценарій	Що перевірено	Підсумок
React Query + TableData	Отримання tables через GET /api/tables і кешування результату.	Дані доступні для Booking, Orders, Table Overview і AdminPage.
Фільтрація за статусом	Запити status=Available і status=Booked.	API повертає підмножини столів за операційним станом.
Створення бронювання	Формування reservation model і PATCH конкретного столу.	Статус столу змінюється, reservation зберігається.
Скасування бронювання	PATCH очищає reservation і повертає status Available.	Стіл знову доступний, список активних бронювань оновлюється.
React Query invalidate	Оновлення кешу після POST/PATCH.	Карта і список Orders не залишаються в старому стані.
Nginx reverse proxy	Передача /api-запитів до backend у compose network.	Frontend і API працюють з одного origin.

Результати інтеграційного тестування свідчать, що головний потік даних працює послідовно. React-компонент ініціює запит, Axios передає його через відносний шлях /api, Nginx прокидує запит до backend, Node.js обробляє route, читає або змінює JSON DB, а відповідь повертається у компонент. Після успішної мутації React Query інвалідує кеш і повторно отримує актуальні дані.

Особливо важливо, що перевірка не обмежувалася позитивними сценаріями. Для форми бронювання перевірялися ситуації без вибраного столика, без дати або часу, із замалою місткістю столу, з already booked status і з неналаштованою LiqPay-конфігурацією для first visit. У таких випадках система не повинна створювати хибне бронювання.

У межах frontend/backend-перевірки було оцінено fallback-поведінку. Якщо частина даних не отримана з backend, карта залу все одно зберігає

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		79

статичний layout, але не повинна створювати помилкове відчуття успішного бронювання. Це важливо для демонстрації: інтерфейс може залишатися зрозумілим, але операційні дії мають залежати від реального стану API.

Таким чином, результати тестування підтверджують, що Crescendo реалізовано як цілісний клієнт-серверний MVP. Система працює не лише як набір сторінок, а як пов'язаний процес: дані вводяться на frontend, перевіряються в інтерфейсі, передаються до API, зберігаються backend і відображаються в інших модулях після оновлення кешу.

Водночас під час тестування виявлено межі поточного прототипу. Система ще не має production-рівня автентифікації, транзакційної перевірки конфліктів часових слотів, реального платіжного підпису та повноцінної бази даних. Ці обмеження не заперечують працездатності MVP, але визначають пріоритети подальшого розвитку.

5.3. Розгортання через Docker, Nginx і Node.js backend

Розгортання Crescendo реалізовано через Docker Compose, що дозволяє запускати frontend, backend і сховище даних як узгоджений набір сервісів. Такий підхід важливий для дипломного проєкту, оскільки він демонструє не лише локальний запуск React-застосунку в режимі розробки, а наблизений до production сценарій із Nginx, Node.js backend і Docker volume.

Web-частина використовує multi-stage Dockerfile. На першій стадії Node-образ встановлює залежності й збирає production-версію React SPA. На другій стадії Nginx-образ отримує готові статичні файли та віддає їх з каталогу /usr/share/nginx/html. Це зменшує розмір фінального web-контейнера і відокремлює процес збірки від процесу роздавання готового застосунку.

Backend винесено в окремий контейнер на Node.js 20-alpine. Зпускає backend/server.js, обслуговує REST API, відповідає на /ready, виконує CORS-обробку, читає та оновлює JSON DB. Усередині docker compose network backend слухає порт 8000, а назовні буде доступний через host-порт 8001 для прямої

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		80

перевірки або налагодження.

Nginx виконує дві функції. По-перше, він віддає production-збірку SPA. По-друге, він прокидає /api-запити до backend:8000. Це дозволяє frontend використовувати відносний шлях /api без окремого публічного backend URL. Крім того, try_files забезпечує SPA fallback: маршрути React Router відкриваються навіть після прямого переходу на /booking, /admin або /orders.

Окремим елементом розгортання є Docker volume backend-data (рис. 5.2). У ньому зберігається JSON-файл restik-db.json, що містить стан столів, бронювань і демонстраційні записи. Завдяки volume дані не зникають після restart контейнерів, що важливо для демонстраційної версії. Без volume кожен перезапуск міг би повертати систему до початкового стану.

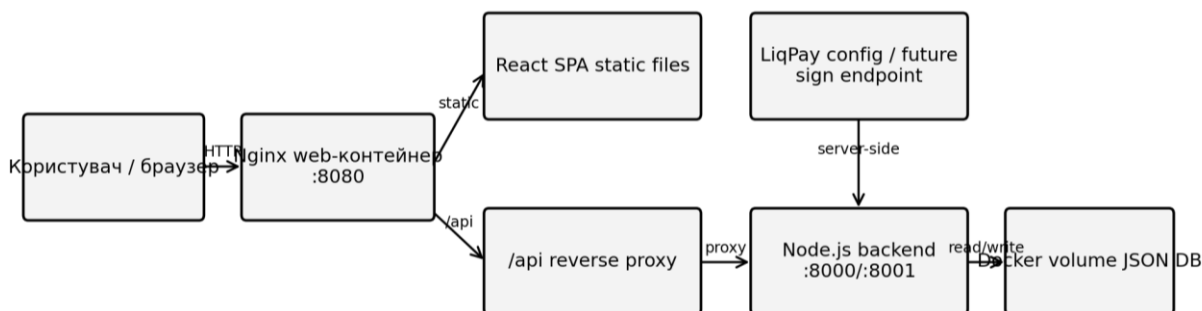


Рисунок 5.2 - Схема Docker-розгортання React/Nginx і Node.js backend

Поточний Docker-стан включає web-контейнер restik2-nginx на http://127.0.0.1:8080 і backend-контейнер restik2-backend на http://127.0.0.1:8001 (табл. 5.4). Для перевірки готовності backend використовується GET /ready. Такий контроль дозволяє швидко відокремити проблеми frontend від проблем backend або контейнерної мережі.

Компоненти Docker-розгортання Crescendo

Компонент	Роль у розгортанні	Перевірка працездатності
web / Nginx	Віддає production-збірку React SPA та виконує reverse proxy для /api.	Відкриття http://127.0.0.1:8080 , прямі переходи на SPA-маршрути.
React build	Статичні HTML, JS і CSS файли, сформовані під час multi-stage build.	Головна сторінка і маршрути відкриваються без dev-server.
backend / Node.js	Обробляє REST API, healthcheck, JSON DB, CORS і валідацію payload.	GET /ready, GET /api/tables, POST/PATCH-запити.
Nginx proxy	Прокидає /api/ до backend:8000 у внутрішній мережі Docker Compose.	Frontend звертається до /api без окремого backend origin.
backend-data volume	Зберігає restik-db.json між restart контейнерів.	Після restart дані бронювань не втрачаються.
seed-demo.js	Формує демонстраційні столи й booking-дані для AdminPage.	Система має 24 столи й 46 booking-записів на 11 дат.

Контейнеризоване розгортання підвищує відтворюваність проєкту. Якщо запускати frontend і backend вручну, результат залежить від локальної версії Node.js, налаштувань портів, шляхів до файлів і середовища користувача. Docker фіксує ці залежності в описі контейнерів і дає змогу швидше повторити запуск на іншому комп'ютері.

Важливим практичним результатом є те, що застосунок запускається як повний клієнт-серверний сценарій. Користувач відкриває сайт через Nginx, frontend завантажує SPA, запити до /api проходять через reverse proxy, backend змінює JSON DB, а оновлений стан повертається в інтерфейс. Таким чином, розгортання не є формальністю, а напряду пов'язане з перевіркою реальної роботи бронювання.

Для production-експлуатації поточну Docker-схему потрібно доповнити змінними середовища, секретами, HTTPS, серверним журналюванням, автоматичними backups, monitoring і CI/CD. Проте для бакалаврського MVP поточна структура вже демонструє правильне розділення ролей між web, backend і persistent volume.

5.4 Висновки по розділу

У п'ятому розділі було проведено системне тестування, розгортання та оцінку результатів роботи веб-застосунку Crescendo. Перевірка показала, що розроблена система функціонує як цілісний клієнт-серверний застосунок, який поєднує публічний інтерфейс користувача, інтерактивну карту залу, форму бронювання, REST API, локальне збереження даних і службову адміністративну панель.

У межах тестування було підтверджено працездатність основних користувацьких сценаріїв. Користувач може відкрити головну сторінку, переглянути меню ресторану, перейти до бронювання, вибрати дату, час, кількість гостей і доступний стіл, а також створити бронювання. Система коректно реагує на помилки введення, блокує недоступні або замалі за місткістю столи та не дозволяє створити запис без обов'язкових параметрів.

Окремо було перевірено роботу адміністративної частини застосунку. Адміністратор має доступ до службових модулів, зокрема календаря, таймлайну сервісу, CRM гостей, нотаток і блоку підготовки залу. Це підтверджує, що Crescendo підтримує не лише етап створення бронювання, а й подальшу організацію ресторанного обслуговування.

Інтеграційне тестування засвідчило, що frontend і backend взаємодіють узгоджено. Компоненти React отримують дані через Axios і React Query, Nginx прокидує API-запити до Node.js backend, backend читає та змінює JSON DB, а після оновлення дані повторно відображаються в інтерфейсі. Завдяки цьому карта залу, список бронювань, сторінка Orders і адміністративна панель використовують спільне джерело актуальних даних.

Розгортання через Docker, Nginx і Node.js backend підтвердило відтворюваність запуску системи. Frontend працює як production-збірка React SPA, Nginx забезпечує роздавання статичних файлів і reverse проху для API, backend обробляє REST-запити, а Docker volume зберігає дані між

					БР.ІІІ - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		83

перезапусками контейнерів. Такий підхід наближує MVP до реального сценарію експлуатації та спрощує демонстрацію роботи проєкту.

У результаті тестування було встановлено, що веб-застосунок Crescendo відповідає основним функціональним вимогам, сформульованим у попередніх розділах. Система забезпечує перегляд меню, вибір параметрів бронювання, роботу з картою залу, створення та скасування бронювань, відображення активних записів і доступ до адміністративних інструментів.

Водночас було визначено обмеження поточної версії. Розроблений застосунок є MVP, тому для подальшого впровадження в реальну ресторанну практику необхідно посилити серверну авторизацію, додати транзакційну базу даних, реалізувати повну перевірку конфліктів часових слотів, підключити реальний платіжний підпис, налаштувати HTTPS, резервне копіювання, журналювання, моніторинг і CI/CD.

Отже, результати п'ятого розділу підтверджують, що поставлену мету досягнуто: створено працездатний веб-застосунок для автоматизації ресторанного сервісу та онлайн-бронювання. Crescendo демонструє повний цикл роботи системи — від взаємодії користувача з інтерфейсом до збереження даних на backend і відображення результатів в адміністративній частині. Це дає підстави вважати розробку завершеною на рівні бакалаврського MVP і придатною для подальшого розвитку.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		84

ВИСНОВКИ

У бакалаврській роботі успішно вирішено актуальне науково-практичне завдання — розроблення комплексного веб-застосунку Crescendo, призначеного для автоматизації роботи ресторану та організації інтегрованої системи онлайн-бронювання столиків. Створене рішення гармонійно поєднує публічну вітрину для гостей закладу, багатофункціональну службову частину для персоналу та локальний Node.js backend для надійного керування даними.

У ході аналізу предметної області було визначено ключові проблеми сучасного ресторанного сервісу, зокрема фрагментованість каналів комунікації, високий ризик дублювання бронювань та відсутність єдиного простору для контролю посадки. Встановлено, що використання сторонніх агрегаторів або громіздких POS-систем не завжди задовольняє потреби авторських та преміальних ресторанів, де важливий індивідуальний підхід. Обґрунтовано потребу у розробці власного веб-застосунку, який би дозволяв враховувати місткість столів, часові слоти, депозитну політику та специфічні дані постійних гостей у межах єдиного фірмового інтерфейсу.

На етапі аналізу вимог було чітко розмежовано потреби різних зацікавлених сторін: гостей, адміністраторів, персоналу залу та власників бізнесу. Сформовано комплекс функціональних і нефункціональних вимог. До ключових функцій віднесено: перегляд інтерактивного меню, вибір дати, часу та кількості гостей, вибір доступного столика на візуальній карті залу, а також створення й скасування бронювань. Для персоналу передбачено адміністративний календар, систему управління взаємовідносинами з клієнтами (Guest CRM), таймлайн сервісу та модуль підготовки залу і кухні.

Під час проєктування розроблено клієнт-серверну архітектуру, що базується на підході Single Page Application (SPA) для клієнтської частини та локальному сервері на базі Node.js. Архітектурна модель чітко розділяє рівні представлення (React-компоненти), клієнтського і серверного стану (React Query), API (Axios) та персистенції (JSON DB). Спроєктowana модель даних

					БР.ІІІ - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		85

охоплює сутності столів, бронювань, платежів, профілів гостей та меню. Для забезпечення взаємодії між frontend і backend розроблено REST API-шар. Логіку роботи системи було візуалізовано за допомогою UML-діаграми випадків використання, Activity-схеми процесу бронювання, узагальненої моделі даних та схеми Docker-розгортання.

Практична реалізація системи виконана із застосуванням сучасного технологічного стеку: React 18, React Router, Material UI, React Query, React Hook Form, Axios та React Toastify для клієнтської частини, а також Node.js 20, Docker і Nginx для серверної та інфраструктурної частин. Клієнтська частина включає повноцінне цифрове меню та дворівневу інтерактивну карту залу, яка динамічно реагує на введені параметри (дата, час, кількість осіб) і блокує недоступні або замалі столи. Реалізовано два бізнес-режими бронювання: «First visit» (із конфігурацією депозиту через LiqPay) та «Guest card» (для постійних клієнтів). Службова підсистема (Admin dashboard) забезпечує персонал демонстраційними профілями гостей, історією візитів, нотатками щодо алергій та вподобань, а також зручним інструментом для контролю завантаженості ресторану.

Системне тестування, яке охоплювало функціональні, інтеграційні та інфраструктурні перевірки, підтвердило повну працездатність веб-застосунку. Перевірка розгортання довела, що production-збірка коректно віддається у Docker-контейнері через веб-сервер Nginx (на локальному порту 8080), який також виконує роль reverse проху для передачі API-запитів до backend-сервісу (порт 8001). Backend успішно обробляє HTTP-запити (зокрема healthcheck /ready), здійснює базову валідацію та атомарний запис до файлу JSON DB, який надійно зберігається у Docker volume.

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		86

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Піх В. Я., Піх М. М., Бандура В. В. Методичні рекомендації та вимоги щодо виконання та оформлення бакалаврських робіт для студентів спеціальності 121 «Інженерія програмного забезпечення». Івано-Франківськ : ІФНТУНГ, 2024. 67 с.

2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016.

3. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016.

4. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

5. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering.

6. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

7. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 8th ed. New York : McGraw-Hill Education, 2015. 976 p.

8. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

10. MDN Web Docs. HTTP. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP> (дата звернення: 01.06.2026).

11. MDN Web Docs. Web Storage API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата звернення: 01.06.2026).

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

12. WHATWG. HTML Living Standard. URL: <https://html.spec.whatwg.org/> (дата звернення: 01.06.2026).

13. Fielding R., Nottingham M., Reschke J. RFC 9110: HTTP Semantics. IETF, 2022. URL: <https://www.rfc-editor.org/rfc/rfc9110> (дата звернення: 01.06.2026).

14. Bray T. RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. IETF, 2017. URL: <https://www.rfc-editor.org/rfc/rfc8259> (дата звернення: 01.06.2026).

15. React Documentation. React Reference Overview. URL: <https://react.dev/reference/react> (дата звернення: 01.06.2026).

16. React Documentation. React DOM APIs. URL: <https://react.dev/reference/react-dom> (дата звернення: 01.06.2026).

17. React Router Documentation. React Router Home and API Reference. URL: <https://reactrouter.com/> (дата звернення: 01.06.2026).

18. Create React App Documentation. Getting Started. URL: <https://create-react-app.dev/docs/getting-started/> (дата звернення: 01.06.2026).

19. Material UI Documentation. Material UI Overview. URL: <https://mui.com/material-ui/getting-started/> (дата звернення: 01.06.2026).

20. TanStack Query Documentation. Overview for React. URL: <https://tanstack.com/query/latest/docs/framework/react/overview> (дата звернення: 01.06.2026).

21. React Hook Form Documentation. Get Started. URL: <https://react-hook-form.com/get-started> (дата звернення: 01.06.2026).

22. Axios Documentation. Getting Started. URL: <https://axios-http.com/docs/intro> (дата звернення: 01.06.2026).

23. React Toastify Documentation. Introduction. URL: <https://fkhadra.github.io/react-toastify/introduction/> (дата звернення: 01.06.2026).

24. Docker Documentation. Multi-stage builds. URL: <https://docs.docker.com/build/building/multi-stage/> (дата звернення: 01.06.2026).

					БР.ІІІ - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		88

25. Docker Documentation. Docker Compose overview. URL: <https://docs.docker.com/compose/> (дата звернення: 01.06.2026).

26. Nginx Documentation. Serving Static Content. URL: <https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/> (дата звернення: 01.06.2026).

27. Nginx Documentation. Beginner's Guide. URL: https://nginx.org/en/docs/beginners_guide.html (дата звернення: 01.06.2026).

28. OWASP Foundation. Application Security Verification Standard 4.0.3. URL: <https://owasp.org/www-project-application-security-verification-standard/> (дата звернення: 01.06.2026).

29. OWASP Foundation. API Security Top 10 — 2023. URL: <https://owasp.org/API-Security/editions/2023/en/0x00-header/> (дата звернення: 01.06.2026).

30. LiqPay. API Documentation. URL: <https://www.liqpay.ua/documentation/> (дата звернення: 01.06.2026).

31. PlantUML Documentation. PlantUML language reference. URL: <https://plantuml.com/> (дата звернення: 01.06.2026).

32. Testing Library Documentation. React Testing Library. URL: <https://testing-library.com/docs/react-testing-library/intro/> (дата звернення: 01.06.2026).

33. Nielsen Norman Group. User Interface Design Guidelines. URL: <https://www.nngroup.com/> (дата звернення: 01.06.2026).

34. React Documentation. React Developer Documentation. URL: <https://react.dev/> (дата звернення: 01.06.2026).

35. REST API Tutorial. REST Architectural Constraints and Principles. URL: <https://restfulapi.net/> (дата звернення: 01.06.2026).

					БР.ІП - 02.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		89

ДОДАТКИ

ДОДАТОК А

Фрагменти структури проєкту Crescendo

Таблиця А.1

Основні файли реалізації

Файл	Призначення
src/components/Header/NavBar.jsx	Головна навігація, посилання на функціональні сторінки, меню користувача
src/components/TableCard/TableCard.jsx	Форма бронювання, карта столів, guest card/deposit логіка, submit до API
src/API/api.js	Axios instance для /api або REACT_APP_API_BASE_URL
src/API/Table.js	Client API helpers для GET/POST/PATCH /api/tables
src/pages/AdminPage.jsx	Admin dashboard: calendar, CRM, timeline, notes, floor/kitchen operations
src/DB/TableLayoutDB.js	Статичний план залу: 24 столи, 2 поверхи, координати та зони
src/DB/MenuDB.js	Статичне меню: 9 секцій, 36 позицій
backend/server.js	Node.js backend, JSON DB, REST API, validation, CORS, atomic write
backend/seed-demo.js	Seed-дані для столів і бронювань
docker-compose.yml	web/backend сервіси, порти, volume, healthcheck
nginx.conf	SPA fallback і reverse proxy /api/ на backend

ДОДАТОК Б

Ілюстрації веб-застосунку Crescendo

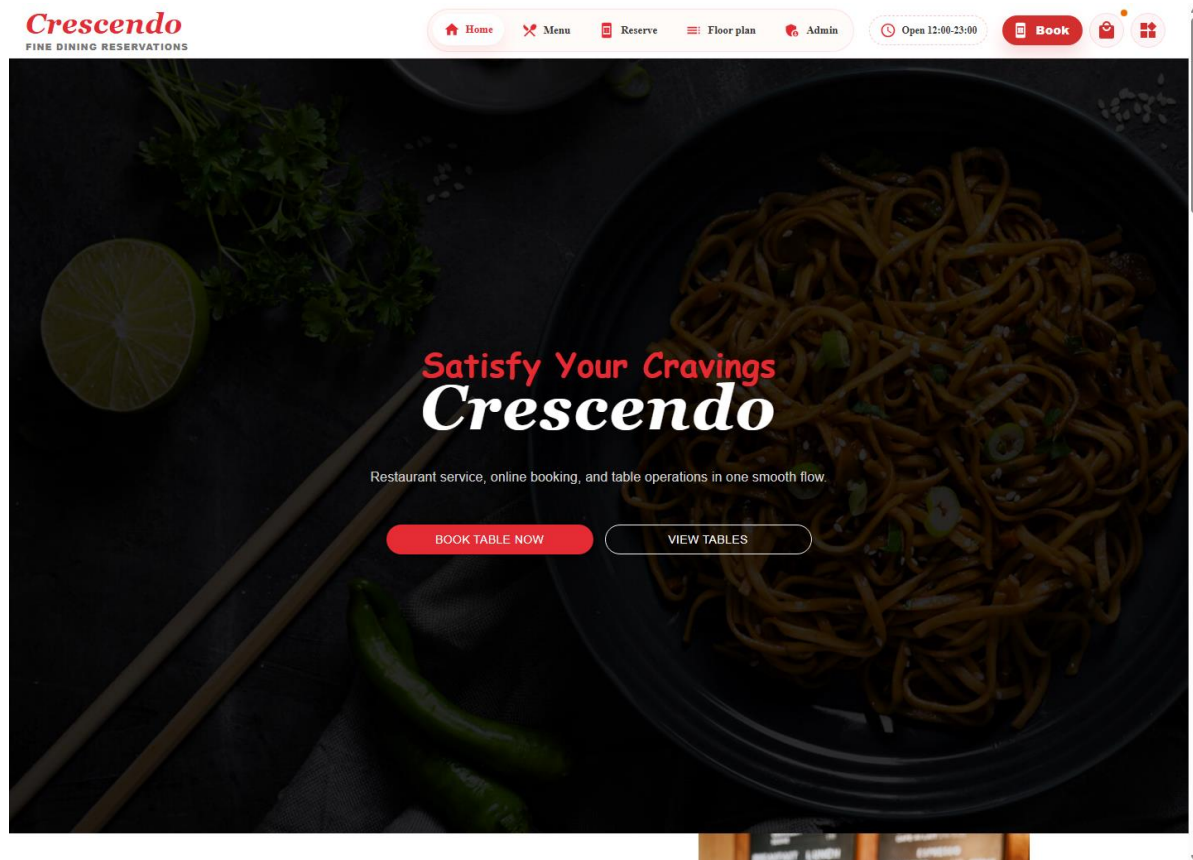


Рисунок Б.1 - Головна сторінка: публічна вітрина та навігація

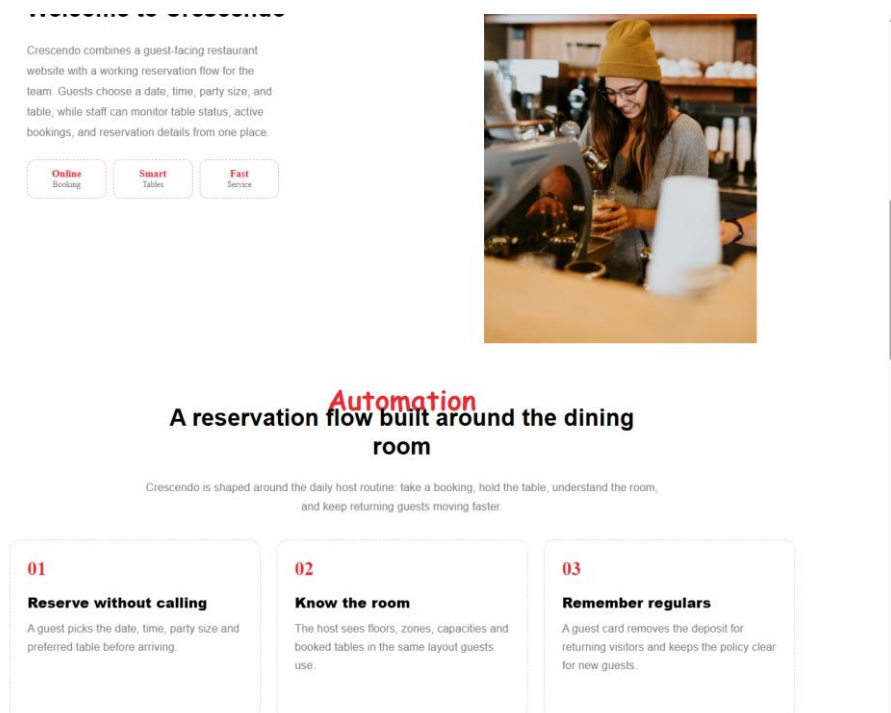


Рисунок Б.2 - Головна сторінка: логіка сервісу.

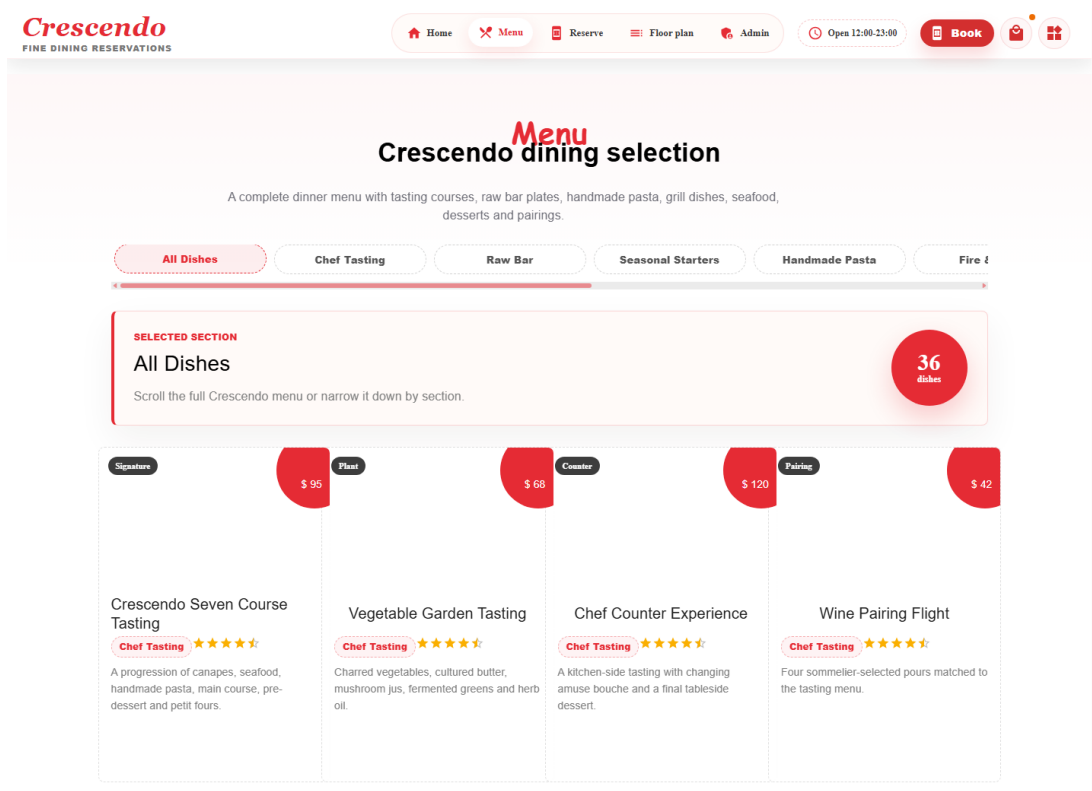


Рисунок Б.3 - Меню ресторану.

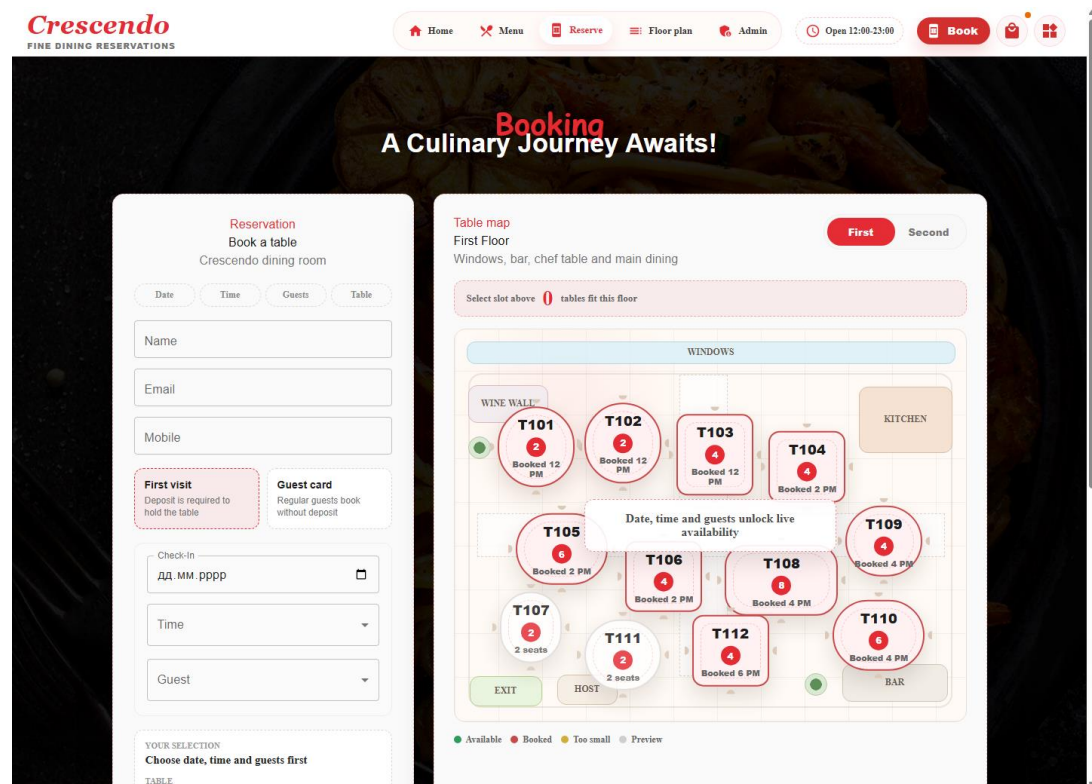


Рисунок Б.4 - Бронювання: перший візит.

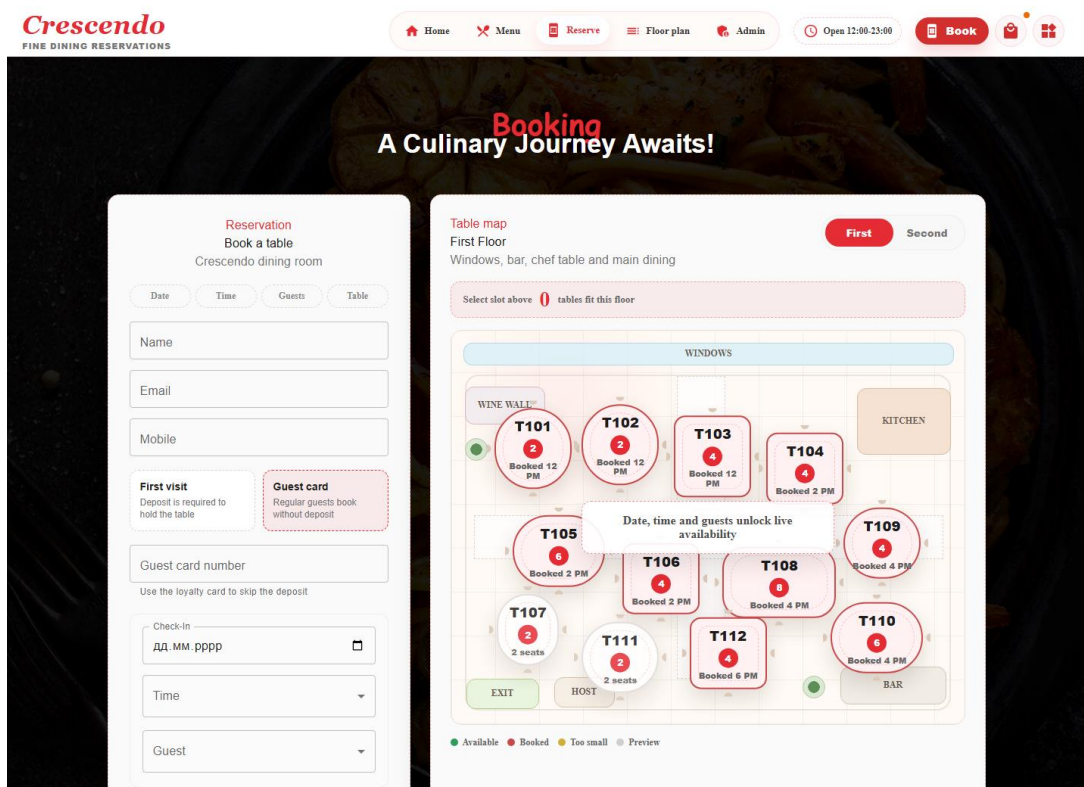


Рисунок Б.5 - Бронювання: гостьова картка.

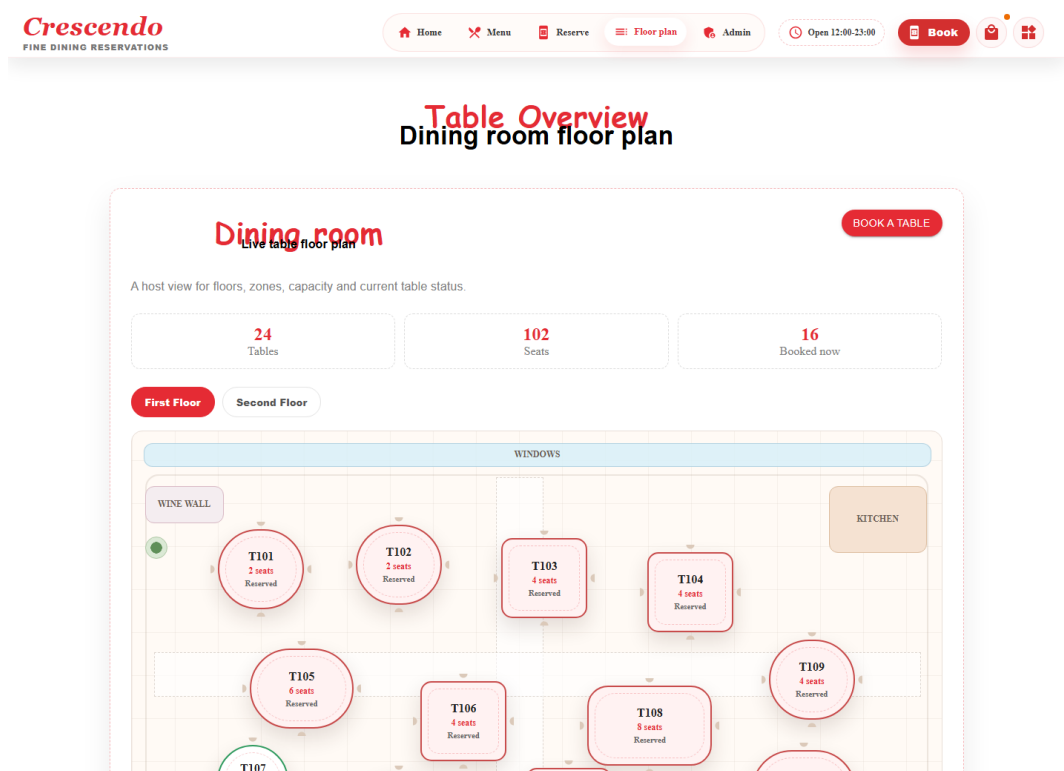


Рисунок Б.6 - План столів: перший поверх.

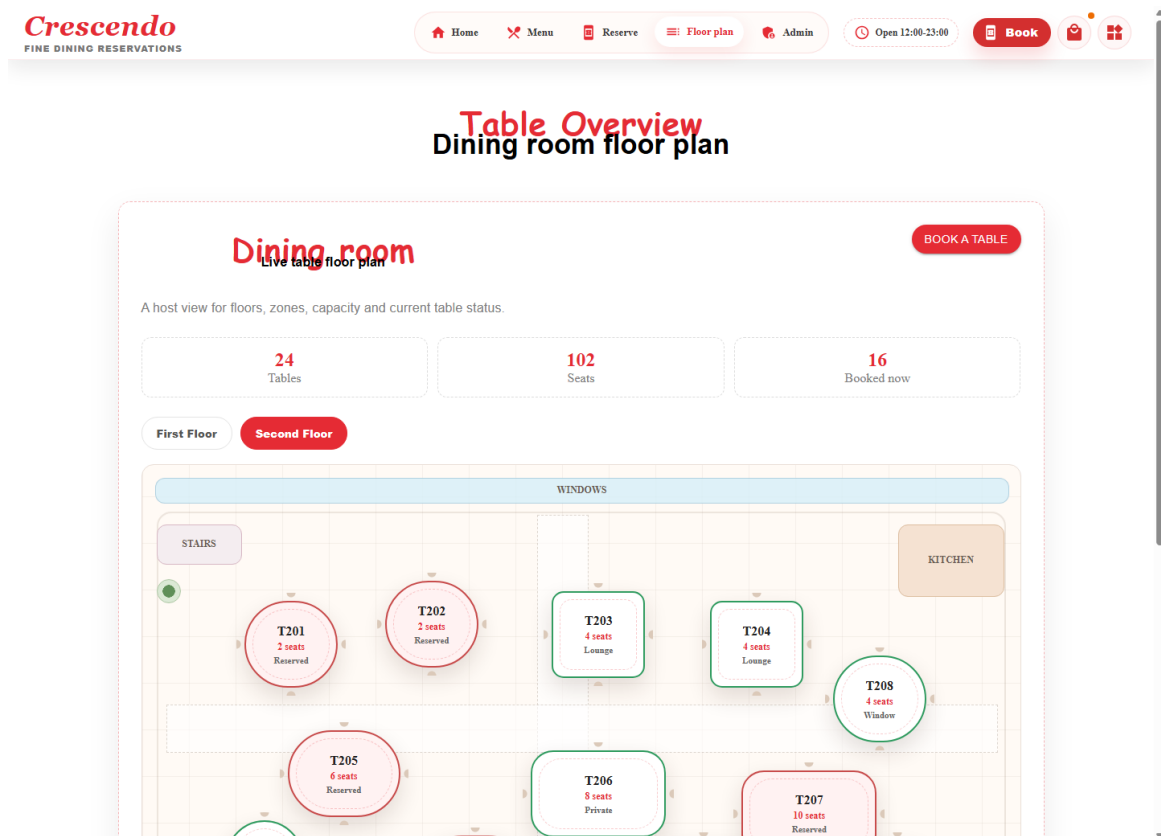


Рисунок Б.7 - План столів: другий поверх.

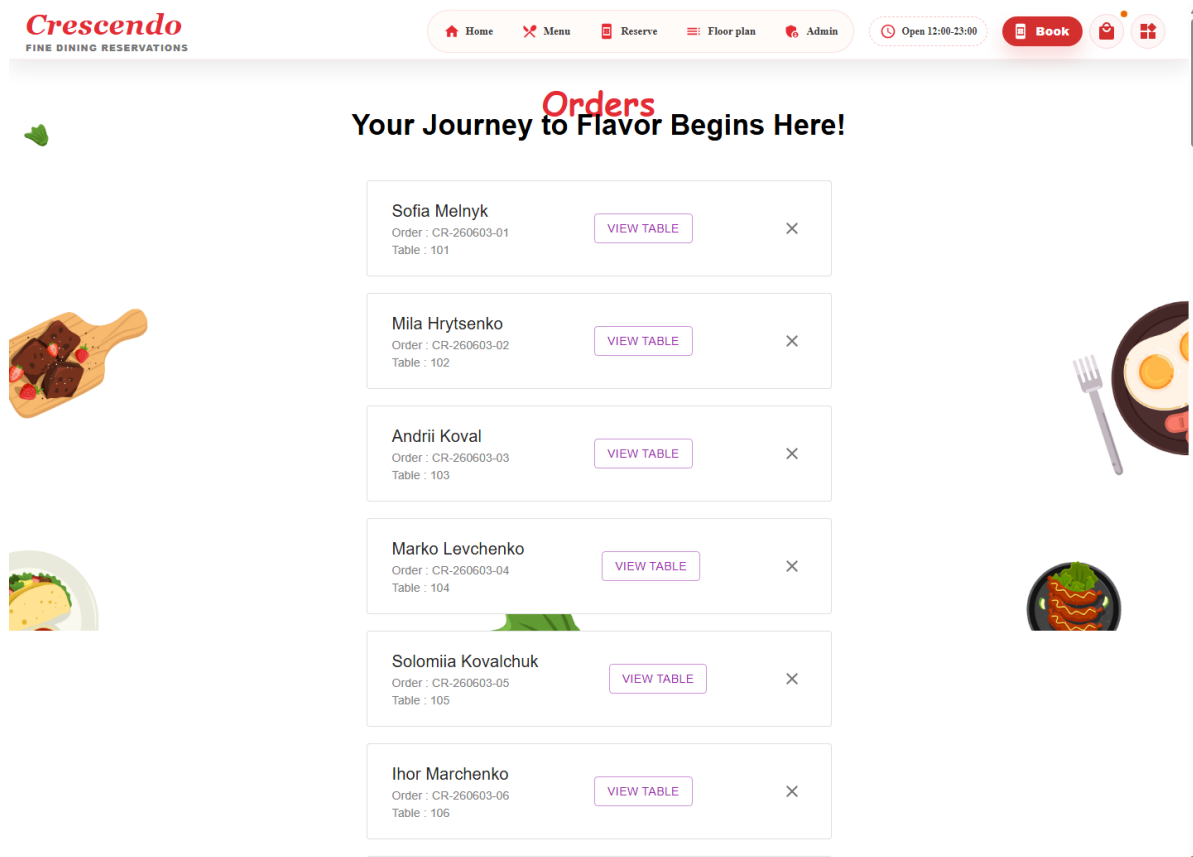


Рисунок Б.8 - Список бронювань.

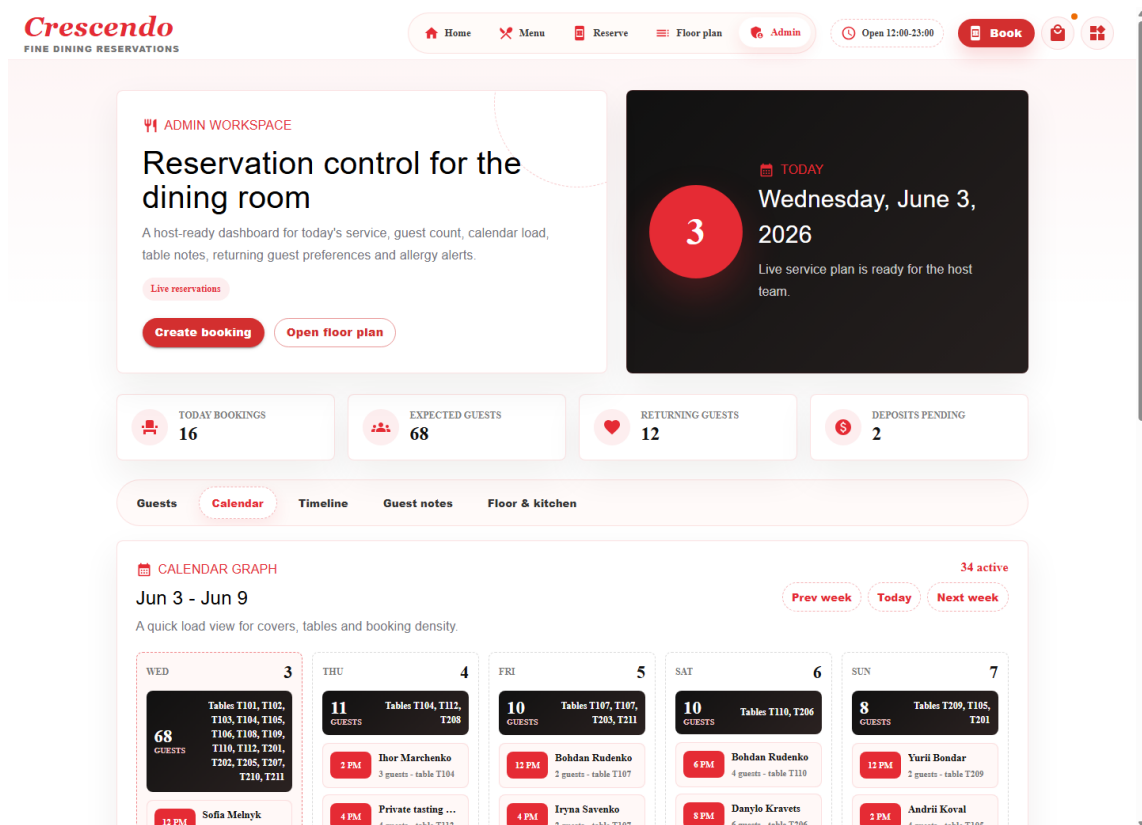


Рисунок Б.9 - Адмінка: календар завантаження.

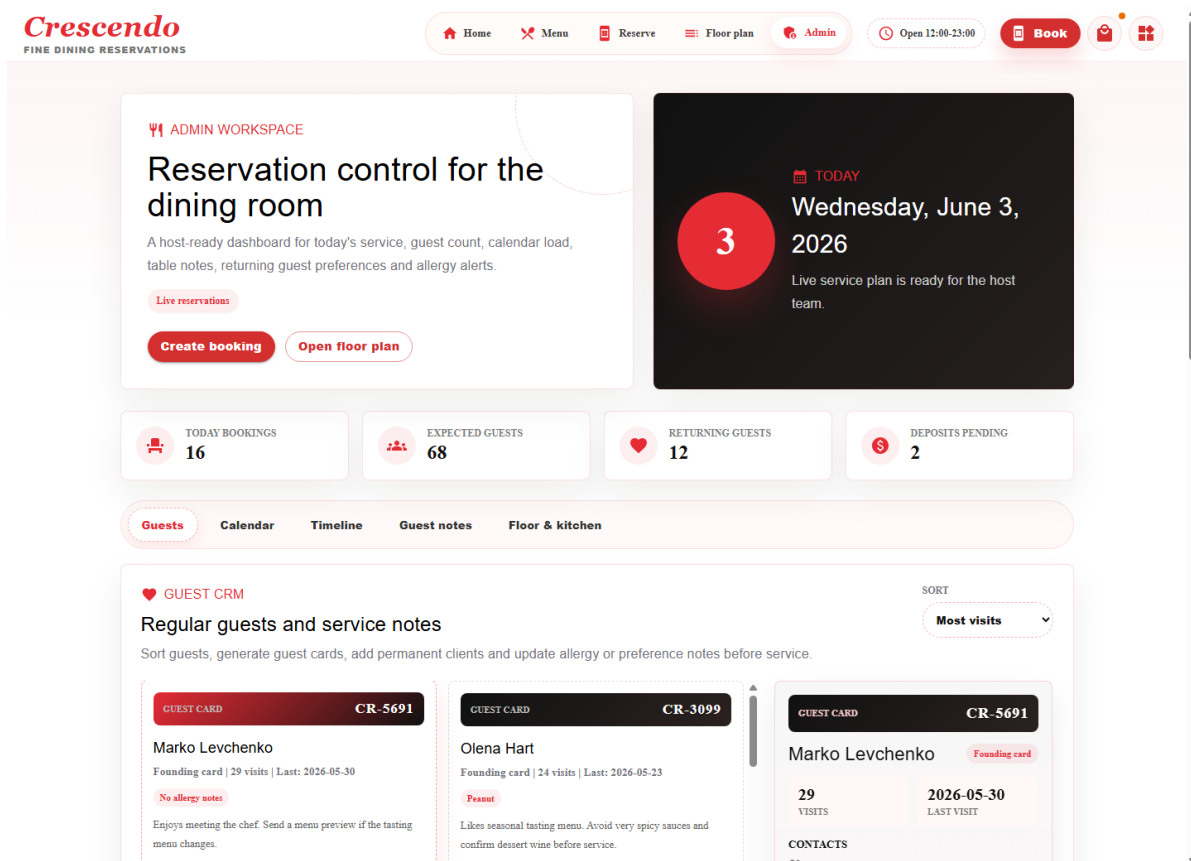


Рисунок Б.11 - Адмінка: CRM гостей і гостьові картки.

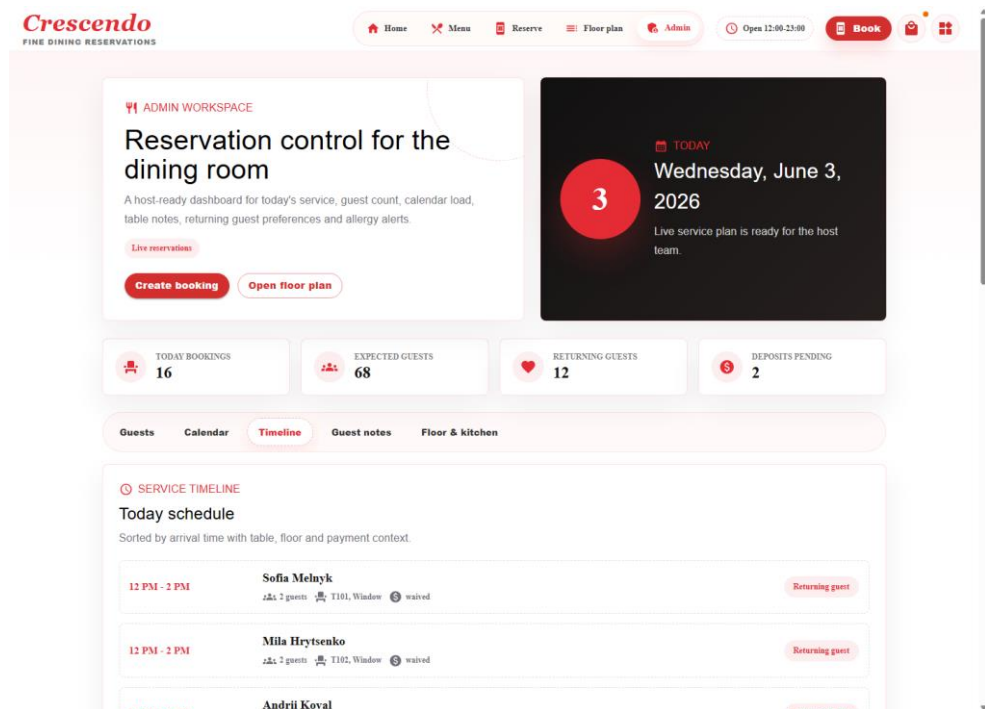


Рисунок Б.12 - Адмінка: таймлайн сервісу.

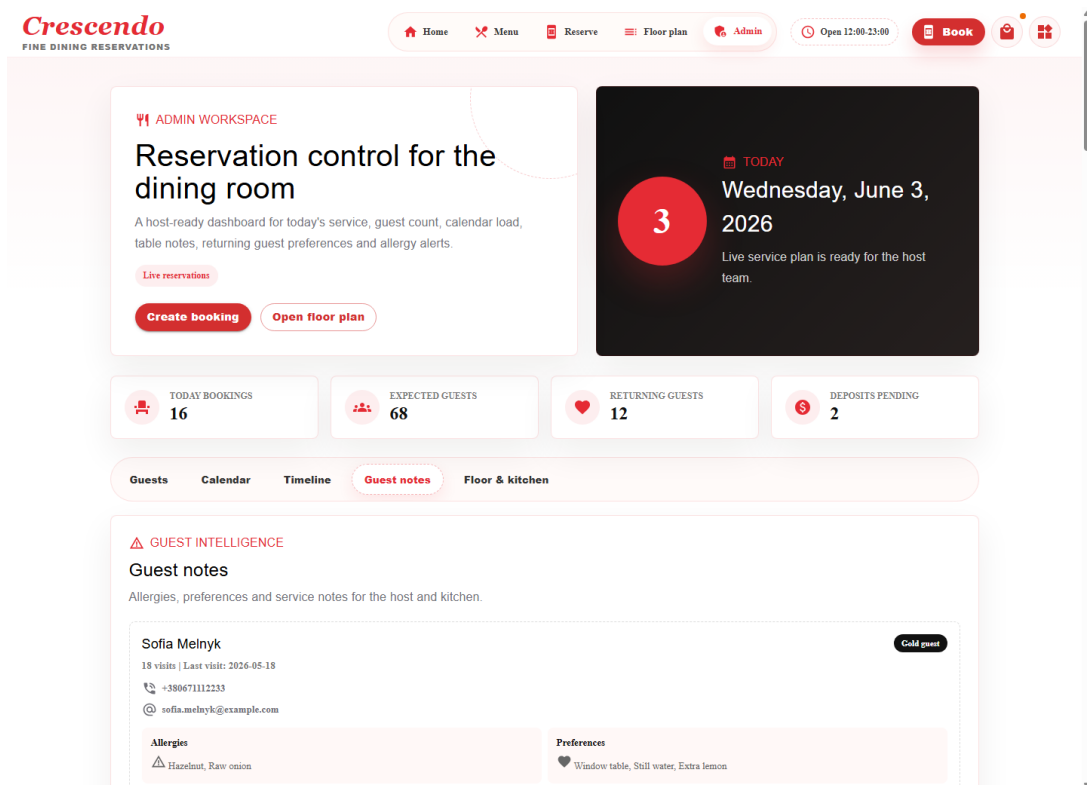


Рисунок Б.13 - Адмінка: нотатки для сервісу.

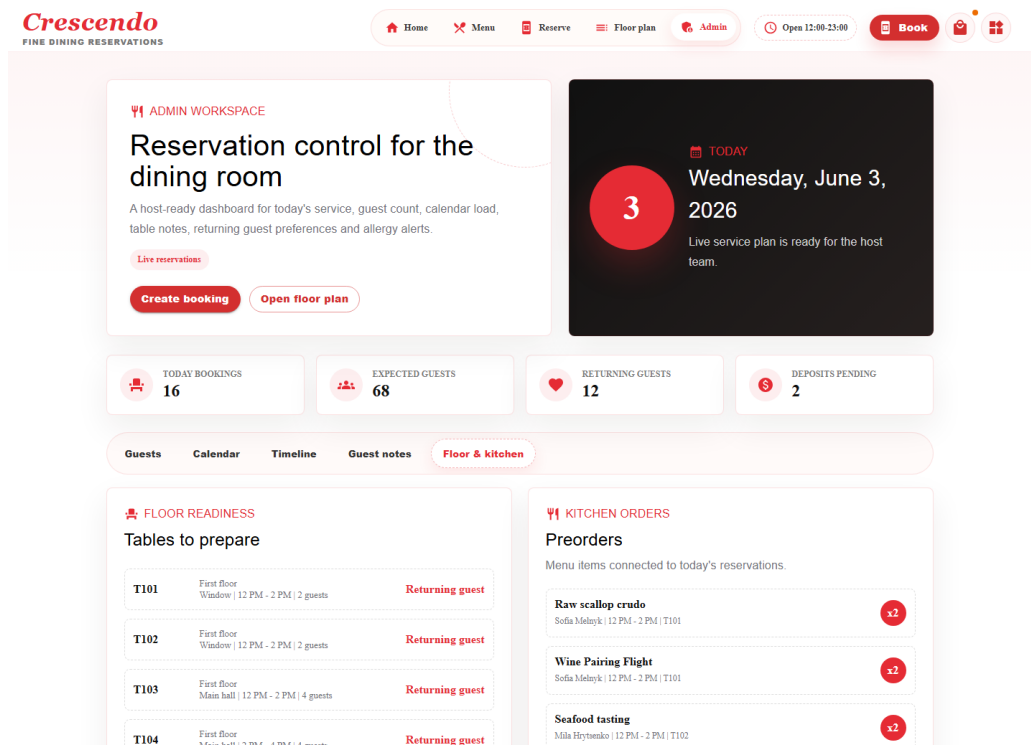


Рисунок Б.14 - Адмінка: підготовка залу та кухні.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: «Розробка клієнт-орієнтованого веб-застосунку для автоматизації роботи ресторану».

Обсяг пояснювальної записки: 100 аркушів

Дата завершення роботи: 30 травня 2026 р.

Здобувач _____ Прудиус Н.І.