МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН СЕРВІСУ РЕЗЕРВУВАННЯ СТОЛІВ У РЕСТОРАНІ»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.О. Ставецький
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ставецькому Максиму Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн сервісу резервування столів у ресторані

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Основні теоретичні відомості.
2. Проєктування та постановка задачі.
3. Розробка вебдодатку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.01.2025	
2.	Збір вихідних даних.	24.01.2025	
3.	Обробка методичних та теоретичних	14.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	04.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2025	

Студент _____ М.О. Ставецький
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн сервісу резервування столів у ресторані»: 67 с., 22 рис., 2 табл., 10 джерел, 1 додаток.

АДАПТИВНИЙ ІНТЕРФЕЙС, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ГОСТЬОВИЙ РЕЖИМ, ІНТЕРАКТИВНА КАРТА, КОРИСТУВАЧ, РЕЗЕРВУВАННЯ.

Об’єкт дослідження – процес автоматизації резервування столів у закладах громадського харчування за допомогою вебтехнологій.

Мета роботи: розробити зручний, сучасний вебзастосунок для резервування столиків у ресторані, який забезпечує повноцінну взаємодію як для зареєстрованих користувачів, так і для гостей, з використанням інтерактивної карти та збереженням даних у хмарну базу.

Метод дослідження – структурно-функціональний підхід до побудови архітектури вебзастосунку, метод компонентного моделювання, метод аналізу і синтезу програмних рішень, інженерно-практичний метод реалізації засобами сучасних фреймворків.

У ході виконання роботи було здійснено аналіз сучасних засобів розробки, обґрунтовано вибір інструментів, спроєктовано архітектуру застосунку, реалізовано систему резервування з інтерактивною картою розміщення столиків, автентифікацією, збереженням інформації у Firebase Firestore та адаптивним інтерфейсом. Застосунок підтримує гостьовий режим, перевірку конфліктів часу, перевірку форм та відображення поточного стану залу в реальному часі. Демонстрація роботи підтверджує відповідність поставленим вимогам і зручність взаємодії для кінцевого користувача.

SUMMARY

Bachelor’s qualifying paper “Development of an Online Restaurant Booking Service”: 67 pages, 22 figures, 2 tables, 10 references, 1 supplement.

ADAPTIVE INTERFACE, DATABASE, WEB APPLICATION, GUEST MODE, INTERACTIVE MAP, USER, RESERVATION.

The object of the study is the process of automating table reservation in public catering establishments using web technologies.

The aim of the study is to develop a convenient and modern single-page web application for table reservation in a restaurant, providing full interaction for both registered users and guests, using an interactive hall map and cloud-based data storage.

The methods of research are structural-functional approach to the architecture of a web application, component modeling, analysis and synthesis of software solutions, engineering-practical implementation using modern frameworks.

In the course of the study, modern development tools were analyzed and selected, the application architecture was designed, and a functional system for table reservation was implemented. The system includes an interactive map of the restaurant’s layout, authentication and guest modes, data storage in Firebase Firestore, and a responsive interface based on Tailwind CSS. The application supports reservation conflict checks, form validation, real-time updates of table availability, and user profile access for viewing reservation history. A demonstration of the system confirms its compliance with the requirements and its usability for end users.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз засобів і технологій вебзастосунків 10
1.1 Мова програмування JavaScript 10
1.2 Огляд бібліотеки React 13
1.3 Порівняння React, Vue та Angular 15
1.4 Екосистема React 18
1.4.1 Способи створення React-додатків 18
1.4.2 Вбудовані бібліотеки React 21
1.4.3 Бібліотека Tailwind CSS 23
1.4.4 База даних Firebase 26
1.4.5 Допоміжні бібліотеки React 28
1.5 Висновки до розділу 30
2 Проєктування архітектури та функціональних компонентів системи 31
2.1 Постановка задачі 31
2.2 Загальна архітектура застосунку 32
2.3 Основні сторінки та компоненти застосунку 34
2.4 Реалізація процесу резервування 36
2.5 Обробка даних користувача 39
2.6 Інтерактивна карта ресторану 41
2.7 Робота з Firebase 43
2.8 Забезпечення надійності та валідація 45
2.9 Висновки до розділу 47
3 Реалізація та демонстрація роботи вебзастосунку 48
3.1 Підготовка середовища розробки 48

3.2 Головний функціонал додатку 50
3.3 Реалізація інтерфейсу 54
3.4 Демонстрація роботи системи 57
3.5 Висновки до розділу 61
Висновки 63
Перелік посилань 65
Додаток А Програмна реалізація компоненту Tablemap 66

ВСТУП

У сучасному цифровому суспільстві все більшої актуальності набуває автоматизація процесів обслуговування у сфері громадського харчування. Замість телефонних дзвінків або особистих візитів клієнти надають перевагу зручним вебзастосункам. Це зумовлює потребу в розробці сервісів резервування, які поєднують зручність, швидкодію та доступність.

Розробка онлайн сервісу резервування столів у ресторані дозволяє покращити взаємодію між закладом і клієнтами, а також оптимізувати внутрішні процеси. Застосунок має бути інтуїтивно зрозумілим, підтримувати гостьовий режим, відображати доступні столи й забезпечувати збереження інформації в хмарній базі.

Мета дослідження – розробити сучасний вебзастосунок для резервування столів у ресторані з інтерактивною картою, підтримкою зареєстрованих користувачів і гостей.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

- проаналізувати сучасні технології розробки вебзастосунків та вибрати оптимальний стек;
- дослідити особливості побудови односторінкових інтерфейсів з використанням React;
- спроєктувати архітектуру системи резервування;
- розробити компоненти інтерфейсу з інтерактивною картою залу;
- реалізувати сценарій резервування для зареєстрованих користувачів і гостей;
- інтегрувати базу даних Firebase для зберігання інформації про користувачів, столи та резервування;
- протестувати функціональність системи та забезпечити її публічне розгортання.

Об’єкт дослідження – процес автоматизації резервування столів у закладах громадського харчування за допомогою вебтехнологій.

Предмет дослідження – особливості розробки клієнтської частини вебзастосунку для онлайн-резервування з використанням React, інтерактивної карти, Firebase і адаптивного інтерфейсу.

Об’єктом дослідження є загальний механізм організації електронного резервування, що передбачає збирання, обробку й збереження даних у режимі реального часу. Предметом дослідження виступає технічна реалізація інтерфейсу, логіки взаємодії та збереження даних у межах обраного технологічного стеку. Саме ці аспекти дозволяють розкрити ефективність впровадженого рішення.

Методи дослідження – структурно-функціональний підхід, компонентне моделювання, інженерна реалізація за допомогою сучасних фреймворків.

Для досягнення результату було обрано JavaScript як основну мову розробки, а бібліотеку React – як засіб реалізації адаптивного інтерфейсу. Для зберігання даних було використано NoSQL базу даних Firebase Firestore, яка забезпечує гнучку структуру та синхронізацію в реальному часі. Система підтримує гостьовий режим, перевірку конфліктів резервування, інтерактивну карту залу та особистий кабінет користувача.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та додатків. Загальний обсяг роботи становить 67 сторінок. Додаток викладено на 2 сторінках. У переліку посилань – 10 джерел.

У першому розділі проаналізовано сучасні технології веброзробки, обґрунтовано вибір React, Firebase та Tailwind CSS. У другому розділі представлено архітектуру застосунку, ключові сторінки та компоненти, їхню логіку та функціональність. У третьому розділі описано реалізацію процесу резервування: від вибору параметрів до збереження в базі даних. У додатку наведено фрагменти програмної реалізації та приклади інтерфейсу.

1 АНАЛІЗ ЗАСОБІВ І ТЕХНОЛОГІЙ ВЕБЗАСТОСУНКІВ

1.1 Мова програмування JavaScript

JavaScript – одна з найбільш розповсюджених і важливих мов програмування в сучасному вебпросторі. Мова була створена в 1995 році компанією Netscape для розширення функціональних можливостей браузера Netscape Navigator. Первинною метою було забезпечення динамічної взаємодії з користувачем без необхідності постійного звернення до сервера. Згодом JavaScript стала стандартом у веброзробці, отримавши офіційне визначення через специфікацію ECMAScript, яка сьогодні постійно оновлюється (ES6, ES7, ESNext тощо) [2].

JavaScript – динамічно типізована, інтерпретована мова, що поєднує можливості кількох парадигм програмування:

- прототипна об’єктна модель дозволяє створювати об’єкти на основі інших об’єктів, без класів;
- функціональна парадигма реалізована через підтримку замикань, анонімних функцій, каррінгу, функцій як об’єктів першого класу;
- імперативний стиль дозволяє описувати послідовність дій через змінні, умовні оператори, цикли та інші класичні конструкції управління потоком.

JavaScript використовується як на стороні клієнта (всередині браузера), так і на стороні сервера. Завдяки середовищу Node.js, мова отримала можливість виконання на сервері, що зробило її універсальним інструментом для створення повноцінних застосунків. Однією з ключових переваг є те, що клієнтська і серверна частина можуть писатися однією мовою.

Основні сфери використання:

- інтерактивність вебсторінок: динамічна зміна DOM-структури, обробка подій, валідація форм, анімації;

– серверна логіка: побудова REST API, WebSocket серверів, обробка запитів у реальному часі з Node.js;
– мобільні застосунки: за допомогою React Native або Cordova;
– десктопні застосунки: Electron дозволяє створювати кросплатформені настільні програми;
– ігрова розробка: з використанням HTML5 Canvas, WebGL і бібліотек на кшталт Three.js або Phaser.

Характерні риси JavaScript:

– відсутність строгого обмеження типів змінних (динамічна типізація);
– можливість створення функцій у будь-якому місці коду;
– відкладене виконання (event loop, проміси, async/await);
– повна інтеграція з HTML та CSS через DOM API;
– підтримка модулів (ES Modules), що дозволяє організовувати код за допомогою імпорту та експорту.

До основних переваг JavaScript відносять:

– простота входження – завдяки схожому з C синтаксису та широкій документації;
– широка екосистема – наявність великої кількості бібліотек (Lodash, Axios, Moment.js) та інструментів;
– кросплатформеність – один і той самий код може працювати в браузері, на сервері або навіть у мобільному застосунку;
– швидкий цикл розробки та оновлення – завдяки популярності мови спільнота активно впроваджує інновації.

Не дивлячись на переваги у JavaScript також є і недоліки:

– відсутність суворої типізації, що збільшує ризик помилок на етапі виконання;
– семантична неоднозначність деяких конструкцій (наприклад, == і ===, автоматичне приведення типів);
– нестабільність підходів – постійна поява нових стандартів і фреймворків ускладнює підтримку великих проєктів;

– низька передбачуваність коду без додаткових інструментів перевірки (linting, типізація через TypeScript тощо).

Для подолання вищезазначених проблем у рамках цієї кваліфікаційної роботи використовується React – сучасна бібліотека для побудови інтерфейсів, яка була створена компанією Facebook. React дозволяє впровадити чітку структуру в проєкт і працювати з компонентною архітектурою, що сприяє покращенню масштабованості, тестованості та підтримки коду.

У таблиці 1.1 наведено, які проблеми JavaScript вирішує React.

Таблиця 1.1 – Проблеми та їх вирішення за допомогою React

Проблема в JavaScript	**Рішення, яке надає React**
Відсутність типів	Можливість використовувати TypeScript
Важко підтримувати великий проєкт	Компонентна модель з розподілом логіки
Складність роботи з DOM	Віртуальний DOM підвищує продуктивність
Відсутність шаблонізації	JSX поєднує HTML та JS у зручному форматі
Ризик повторного коду	Повторне використання компонентів

Сучасна розробка майже не обходиться без використання JavaScript. Завдяки своїй універсальності мова стала основою Front-end розробки, а з появою Node.js – набула широкого застосування і в Back-end середовищі. Активна еволюція екосистеми, постійне впровадження нових стандартів, розвиток численних бібліотек і фреймворків, а також широка підтримка з боку браузерів зміцнили статус JavaScript як провідної мови в екосистемі веброзробки. Але не дивлячись на це, JavaScript, не є ідеальною мовою програмування. У зв’язку з цим у межах даної кваліфікаційної роботи використовується фреймворк React, який не лише дозволяє реалізувати інтерфейс користувача з високою продуктивністю, а й нівелює деякі типові недоліки JavaScript. Завдяки компонентній архітектурі, підтримці керування станом, ефективному віртуальному DOM і можливості інтеграції з

додатковими інструментами (такими як Redux, React Router, Firebase), React виступає оптимальним рішенням для побудови масштабованих, зручних та підтримуваних інтерфейсів у вебзастосунках.

1.2 Огляд бібліотеки React

React – це популярна JavaScript-бібліотека, призначена для створення динамічних, швидких і масштабованих інтерфейсів користувача. Ця бібліотека активно розвивається та підтримується компанією Meta (раніше Facebook), а також численною спільнотою незалежних розробників та компаній у всьому світі. React забезпечує декларативний підхід до розробки, дозволяючи створювати адаптивні та гнучкі вебзастосунки, що швидко реагують на зміни даних без необхідності повного перезавантаження сторінки [7].

Основна концепція бібліотеки полягає у представленні інтерфейсу у вигляді ієрархії компонентів, кожен з яких інкапсулює власну логіку, стан та спосіб візуалізації. Компоненти можуть бути як функціональними, так і класовими, проте у сучасній розробці перевага надається функціональним компонентам із використанням хуків useState, useEffect та інші, що дозволяє зменшити обсяг коду й зробити структуру застосунку прозорішою.

React виконує роль View у шаблоні архітектури Model View Controller (MVC). Це означає, що бібліотека фокусується виключно на відображенні вмісту, залишаючи питання бізнес-логіки та зберігання стану стороннім інструментам або архітектурним підходам, таким як Flux чи Redux. React може бути інтегрований з іншими фреймворками або бібліотеками, що дозволяє гнучко налаштовувати архітектуру застосунку відповідно до потреб проєкту [1].

Бібліотека React була створена інженером Meta Джорданом Уолке. Ідеї для реалізації були частково запозичені з XHP – HTML-компоненту для PHP. Спочатку React використовувався у стрічці новин Facebook (2011 рік), а згодом

і в Instagram (2012). У 2013 році проєкт був відкритий як open-source і швидко здобув популярність серед Front-end розробників завдяки зручному підходу до створення компонентів і ефективності при оновленні DOM.

Однією з найінноваційніших функцій React є використання віртуального DOM. Віртуальний DOM (VDOM) – це концепція програмування, в якій ідеальне чи віртуальне представлення інтерфейсу користувача зберігається в пам'яті і синхронізується зі «справжнім DOM за допомогою бібліотеки, такої як ReactDOM. Цей процес називається узгодженням. Власне такий підхід і робить API React декларативним: вказується React-у, у якому стані повинен знаходитись інтерфейс користувача, а React, у свою чергу, вже домагається того, щоб DOM відповідав цьому стану. Це абстрагує маніпуляції з атрибутами, обробку подій та оновлення DOM вручну, які в іншому випадку, довелося б використовувати при розробці додатків.

Оскільки «віртуальний DOM» – це швидше патерн, ніж конкретна технологія, цим терміном іноді позначають різні поняття. У світі React термін «віртуальний DOM» зазвичай асоціюється з React елементами, оскільки ці елемегьи є об'єктами, що представляють інтерфейс користувача. Однак, React також використовує внутрішні об'єкти, так звані «волокна» (fibers) для зберігання додаткової інформації про дерево компонентів. Також можуть вважатися частиною реалізації «віртуальний DOM» в React.

Замість безпосереднього маніпулювання DOM браузера, React створює в пам'яті копію структури сторінки, яку періодично порівнює з новою версією інтерфейсу. Результатом такого порівняння є так званий diff, який React застосовує до реального DOM, мінімізуючи кількість операцій оновлення. Це дозволяє досягати високої продуктивності навіть у складних динамічних інтерфейсах.

Для опису компонентів часто використовується JSX (JavaScript XML) – синтаксичне розширення JavaScript, яке дає змогу комбінувати логіку та розмітку в межах одного компоненту. JSX виглядає як HTML-код, вбудований у JavaScript-функцію. Такий підхід робить структуру коду більш наочною та

спрощує розробку. JSX проходить трансформацію у звичайні виклики функцій React, які створюють елементи компонентного дерева [4].

Ключовим принципом взаємодії у React є односпрямований потік даних: “properties flow down; actions flow up”. Компонент отримує вхідні дані через властивості (props), які не змінюються безпосередньо, а зміни здійснюються за допомогою функцій зворотного виклику, які впливають на стан у батьківських компонентах або глобальному сховищі. Ця ідея стала основою архітектури Flux, яка пізніше трансформувалася у спрощену й потужну реалізацію – Redux, що також використовується у багатьох React-застосунках.

React також має розвинену екосистему, яка включає:

– React Router – для організації маршрутизації в SPA;
– Redux або Context API – для глобального керування станом;
– React Hook Form, Formik – для роботи з формами;
– Next.js – фреймворк для серверного рендерингу на основі React;
– React Native – для розробки нативних мобільних застосунків з використанням React-компонентів.

Завдяки зручності, високій продуктивності та активній підтримці спільноти, React став де-факто стандартом у створенні сучасних Front-end застосунків. Саме тому в межах цього проєкту обрана бібліотека React – як оптимальне рішення для реалізації інтерфейсу системи резервування з динамічною побудовою сторінок, гнучкою архітектурою та масштабованістю.

1.3 Порівняння React, Vue та Angular

Після детального розгляду бібліотеки React доцільно проаналізувати інші поширені фреймворки JavaScript, які конкурують з React у сучасній веброзробці. Найбільш вагомими альтернативами є Vue.js та Angular. Кожен з цих інструментів має власну філософію, архітектуру та специфіку використання, тому порівняння є важливим кроком для обґрунтованого вибору технологій у реалізації вебзастосунків.

Vue.js – це легкий та гнучкий JavaScript-фреймворк, що реалізує модель MVVM (Model-View-View-Model). Архітектура Vue передбачає двостороннє реактивне зв’язування даних, що дозволяє автоматично оновлювати інтерфейс користувача при зміні стану моделі. Завдяки цьому механізму Vue добре підходить для створення інтерактивних вебінтерфейсів [9].

До основних переваг Vue.js належать:

– низький поріг входження: зрозуміла документація, простий синтаксис та логіка дозволяють швидко засвоїти фреймворк навіть новачкам;
– реактивність: вбудована система реактивності дає змогу ефективно відслідковувати зміни в даних і миттєво оновлювати DOM;
– продуктивність: невеликий розмір ядра (≈20 КБ у стиснутому вигляді) забезпечує високу швидкість завантаження та рендерингу;
– можливість поступового впровадження: Vue можна інтегрувати до вже існуючого проєкту без необхідності повної перебудови архітектури.

Попри це, Vue.js має і обмеження:

– менша популярність серед великих компаній, порівняно з React або Angular, що впливає на кількість готових рішень і ресурсів;
– обмежена кількість навчальних матеріалів та підтримки, особливо у порівнянні з більш «дорослими» інструментами;
– може бути менш придатним для масштабних проєктів, які потребують жорстко структурованої архітектури та комплексного контролю над усіма аспектами системи.

Загалом, Vue – це чудовий вибір для невеликих та середніх проєктів, де важливі швидкість розробки, простота підтримки й низька складність вхідного порогу.

Angular – це комплексний фреймворк з відкритим кодом, створений і підтримуваний компанією Google, розроблений на базі TypeScript, що забезпечує жорстку типізацію, автодоповнення та виявлення помилок на етапі компіляції. Angular реалізує підхід повної інтеграції: усі необхідні

інструменти (модулі, шаблони, форми, маршрутизація, аутентифікація тощо) вже входять до складу фреймворку [5].

Основні переваги Angular:

– компонентна архітектура, яка забезпечує повторне використання коду та модульність;
– повноцінна інфраструктура для створення великих SPA – усе «з коробки», без потреби підключення сторонніх бібліотек;
– надійна підтримка та активна спільнота, завдяки чому Angular постійно вдосконалюється;
– сувора типізація через TypeScript, що сприяє підвищенню якості коду та зменшенню кількості помилок.

Проте Angular має і низку недоліків, які слід враховувати:

– складність вивчення: порівняно високий поріг входження через велику кількість концепцій, інструментів та синтаксичних конструкцій;
– більша кількість коду: створення навіть простого компонента потребує більше шаблонного коду, ніж у React чи Vue;
– менш зручна розробка мобільних застосунків: у порівнянні з React Native чи Flutter, Angular не надає повноцінних інструментів для нативної мобільної розробки;
– великий розмір фінального пакета, що може впливати на продуктивність у повільних мережах.

Angular орієнтований на великі корпоративні проєкти, де важливі безпека, масштабованість, структурованість та довгострокова підтримка. Цей фреймворк ідеально підходить для команд, які працюють над складними застосунками з великою кількістю бізнес-логіки.

Vue.js та Angular, як і React, пропонують свої підходи до побудови сучасних вебзастосунків. Vue – легкий, простий і гнучкий, ідеальний для швидкої розробки невеликих проєктів. Angular – потужний, строгий і комплексний, створений для масштабних рішень у великих командах. Вибір

між цими фреймворками залежить від специфіки проєкту, досвіду розробників і функціональних вимог до системи.

Порівнюючи React з іншими технологіями, можна сказати, що React є досить універсальним для використання, як в малих так і в великих проєктах, також React поєднує в собі гнучкість і масштабованість, забезпечуючи оптимальний баланс між простотою та потужністю. Саме тому для розробки Онлайн сервісу для резервування столів в ресторані, було обрано React.

1.4 Екосистема React

React є не лише бібліотекою для побудови інтерфейсів, а й основою для цілої екосистеми, яка включає низку внутрішніх та супутніх бібліотек, що забезпечують повноцінний цикл розробки сучасних вебзастосунків. Завдяки таким розширенням React перетворюється на гнучку платформу, яку можна адаптувати до різноманітних задач – від SPA до складних інтерактивних систем.

1.4.1 Способи створення React-додатків

Початковий етап створення вебзастосунку на React передбачає вибір інструменту для ініціалізації проєкту та налаштування середовища розробки. У сучасній практиці існує кілька поширених способів створення React-додатків: Create React App, Vite, Next.js, Ручна конфігурація (Webpack + Babel), кожен із яких має свої особливості, переваги та недоліки. У цьому підпункті розглянуто основні підходи, які використовуються в розробці.

Create React App (CRA) – офіційний шаблонізатор від команди React, який дозволяє швидко створити повністю готове до розробки середовище без потреби вручну конфігурувати Webpack, Babel, ESLint, PostCSS та інші

інструменти. Створення проєкту з CRA відбувається за допомогою команди: “npx create-react-app@latest table-reserv” [10].

Основні переваги Create React App:

– автоматичне налаштування збірки, трансляції JSX, обробки CSS;
– швидкий старт проєкту для новачків і середніх за обсягом проєктів;
– вбудована підтримка перевірки коду (linting), запуску локального сервера, збірки для продакшену;
– активна підтримка спільноти та велика кількість навчальних ресурсів.

Недоліки:

– обмежена гнучкість конфігурації без використання eject;
– відносно повільна збірка великих проєктів у порівнянні з новішими інструментами.

Vite – сучасний інструмент збирання застосунків, що забезпечує блискавичний старт та оновлення під час розробки. Vite використовує esbuild для попередньої обробки, а також підтримує React-проєкти через відповідний шаблон.

Переваги:

– майже миттєвий запуск навіть великих проєктів;
– дуже швидке оновлення при зміні коду (hot module replacement);
– проста конфігурація, сумісність із багатьма плагінами.

Недоліки:

– менш зрозумілий початківцям, ніж CRA;
– у деяких випадках вимагає ручного налаштування сумісності з окремими бібліотеками.

Next.js – фреймворк, побудований на базі React, який додає підтримку серверного рендерингу (SSR), статичної генерації (SSG), а також автоматичну маршрутизацію та SEO-оптимізацію.

Переваги:

– підтримка SSR і SSG «із коробки»;
– автоматична оптимізація продуктивності;

– вбудована маршрутизація на основі структури директорій.

Недоліки:

– вищий рівень складності;

– не потрібен для SPA-застосунків без SEO-вимог;

– структура проєкту жорстко регламентована.

Ручна конфігурація (**Webpack + Babel**). Цей підхід полягає у створенні проєкту «з нуля», з повною ручною конфігурацією Webpack, Babel, ESLint, PostCSS та інших інструментів.

Переваги:

– повний контроль над конфігурацією;

– можливість оптимізації під вузькі задачі.

Недоліки:

– велика кількість рутинної роботи;

– потреба в глибоких знаннях Webpack і суміжних інструментів;

– низька швидкість старту розробки.

В таблиці 1.2 наведено порівняння способів ініціалізації React.

Таблиця 1.2 – Порівняння способів ініціалізації

Підхід	**Продуктивність у розробці**	**Гнучкість**	**SEO/SSR**	**Рекомендовано для**
Create React App	Середня	Низька	Ні	Новачки, навчальні проєкти
Vite	Висока	Висока	Ні	Сучасні SPA, невеликі PWA
Next.js	Висока	Середня	Так	SEO-орієнтовані сайти
Webpack + Babel	Залежить від налаштувань	Максимальна	Можливо	Унікальні/вузько спеціалізовані проєкти

У рамках розробки онлайн-сервісу для бронювання столів було обрано Create React App як оптимальне рішення для швидкого старту, зручного середовища розробки та простої підтримки. Такий підхід дозволив зосередитись на бізнес-логіці та інтерфейсі користувача, не витрачаючи додаткових ресурсів на складну конфігурацію середовища. Разом із тим, аналіз альтернативних рішень дозволяє зробити висновок, що вибір інструменту має базуватись на типі проєкту, його масштабі та вимогах до SEO, продуктивності та розгортання.

1.4.2 Вбудовані бібліотеки React

React як бібліотека для побудови інтерфейсів не існує у вакуумі, бо має тісну екосистему вбудованих і супутніх інструментів, які забезпечують повний цикл створення, тестування, маршрутизації та рендерингу застосунків. У межах цього проєкту використано ключові бібліотеки та інструменти з офіційного React-оточення: React DOM, React Router, React Hooks, JSX (JavaScript XML), React Scripts.

Бібліотека react-dom використовується для інтеграції React-компонентів з деревом DOM браузера. Ця бібліотека відповідає за рендеринг компонентів, оновлення інтерфейсу після зміни стану, а також за очищення елементів при демонтажі.

У проєкті використовується для монтування головного компоненту в HTML-елемент з ідентифікатором root (ReactDOM.createRoot(document.getElementById('root'))).

Бібліотека react-router-dom реалізує клієнтську маршрутизацію в односторінкових застосунках (SPA), дозволяє створювати навігаційні маршрути, вкладені шляхи, обробку параметрів URL і реалізацію багатокрокових процесів.

Основна функціональність:

– навігація за допомогою <Link> і useNavigate();

– побудова маршрутизованої структури сторінок через компоненти <BrowserRouter>, <Routes>, <Route>;
– отримання параметрів маршруту за допомогою useParams() та поточного стану за допомогою useLocation().

У проєкті використовується для:

– переходу між сторінками: вхід, реєстрація, бронювання, підтвердження, особистий кабінет;
– організації покрокового бронювання та передачі параметрів між етапами.

Хуки – це функції, які дозволяють використовувати можливості стану, ефектів, контексту та інших особливостей React у функціональних компонентах, без необхідності використання класів.

Найважливіші хуки, використані в проєкті:

– useState – створення і зміна стану компонентів (вибір дати, столу, часу, користувача);
– useEffect – виконання дій при монтуванні компонента або зміні залежностей (отримання даних із Firebase);
– useNavigate – програмна навігація між сторінками;
– useRef – доступ до DOM-елементів або збереження посилань на змінні;
– useSearchParams – читання параметрів URL (наприклад, в режимі гостьового бронювання).

Переваги хуків:

– зменшення обсягу шаблонного коду;
– покращення читабельності;
– можливість повторного використання логіки через користувацькі хуки.

JSX – це синтаксичне розширення JavaScript, яке дозволяє використовувати HTML-подібну розмітку в середині JavaScript-коду. JSX компілюється у виклики функції React.createElement() і дозволяє створювати дерево компонентів у декларативному вигляді.

У проєкті JSX застосовується для:

– побудови структури сторінок;
– використання умовного рендерингу ({умова && <Компонент />});
– відображення таблиць, картки столів, форм введення.

Переваги JSX:

– інтуїтивність для верстальників та Front-end розробників;
– зручне поєднання логіки та розмітки;
– можливість вкладати компоненти один в одного.

react-scripts – це службовий пакет, який використовується в Create React App для запуску, збірки та тестування проєкту. Цей службовий пакет автоматизує конфігурацію таких інструментів, як Babel, Webpack, ESLint, PostCSS тощо.

Команди, що використовуються в проєкті:

– “npm start” – запуск локального сервера;
– “npm run build” – створення оптимізованої продакшен-версії;
– “npm run eject” – розкриття внутрішніх налаштувань (не застосовувалося у цьому проєкті) [6].

Вбудовані бібліотеки React та ключові механізми екосистеми (React DOM, маршрутизація, хуки, JSX, react-scripts) створюють надійне середовище для побудови односторінкових вебзастосунків. У межах цього проєкту вбудовані бібліотеки дозволили реалізувати логічну структуровану архітектуру інтерфейсу, забезпечити динамічну зміну стану без перезавантаження сторінки, та інтегрувати необхідну маршрутизацію для процесу бронювання.

1.4.3 Бібліотека Tailwind CSS

Tailwind CSS – це сучасний CSS-фреймворк, що використовує утилітарний підхід до стилізації інтерфейсів. На відміну від традиційних CSS-

фреймворків, де застосовуються готові компоненти або блоки стилів (як-от у Bootstrap), Tailwind пропонує набір маленьких класів, які відповідають за окремі властивості (відступи, кольори, шрифти, розміри тощо). Це дозволяє без написання кастомного CSS створювати гнучкі та адаптивні компоненти безпосередньо в HTML або JSX [8].

У межах реалізації онлайн-сервісу бронювання столів Tailwind CSS було обрано завдяки наступним перевагам:

– швидкість розробки: стилі можна писати без переходу до окремого CSS-файлу;
– повний контроль над дизайном: кожен елемент можна налаштувати вручну, до пікселя;
– мінімізація перевикористання/перевизначення стилів: завдяки атомарним класам уникнуто конфліктів CSS;
– гнучкість адаптивності: Tailwind дозволяє додавати стилі для різних розмірів екрана в одній конструкції (md:, lg:, xl:);
– хороша інтеграція з React: класи можна легко вбудовувати в JSX-структуру компонентів.

Недоліки Tailwind CSS:

– перевантаження HTML/JSX-структури класами, що може знижувати читабельність без форматування або сторонніх інструментів (наприклад, Prettier);
– не підходить для проєктів, де потрібна єдина стилістика «із коробки» (наприклад, корпоративні UI-гайди);
– не надає готових компонентів – усе будується вручну або через бібліотеки типу Headless UI, Tailwind UI [3].

Під час вибору CSS фреймворка, було також розглянуто Bootstrap. Bootstrap – це рішення для швидкого старту, розробки прототипів, лендінгів та інформаційних сайтів. Завдяки простоті, стабільності й великій кількості компонентів, Bootstrap залишається актуальним вибором для розробників усіх рівнів.

Основні можливості Bootstrap:

– сітка на основі Flexbox (.container, .row, .col- класів);
– готові компоненти (кнопки, модальні вікна, алерти, навігаційні панелі тощо);
– адаптивна типографіка та кнопки;
– інтегровані JavaScript-плагіни (наприклад, каруселі, dropdown-меню);
– темізація через змінні SASS.

Переваги:

– велика кількість готових стилів та компонентів дозволяє швидко створити прототип;
– побудована мобільна перша сітка гарантує підтримку різних пристроїв;
– одна з найкращих документацій серед CSS-фреймворків;
– велика спільнота;
– безліч шаблонів, прикладів сайтів та тем оформлення;
– стилі протестовані на більшості популярних браузерів.

Недоліки:

– велику кількість сайтів на Bootstrap легко впізнати – інтерфейси схожі;
– кастомізація вимагає перевизначення численних класів, що ускладнює підтримку;
– навіть базова збірка містить стилі, які можуть бути не використані у проєкті;
– складніше реалізовувати унікальні компоненти без написання додаткового CSS.

У підсумку, обидва фреймворки – і Tailwind CSS, і Bootstrap – мають свої переваги та сфери застосування. Bootstrap краще підходить для швидкої розробки сайтів із типовим інтерфейсом, коли важлива уніфікованість і мінімізація кастомних стилів. Натомість Tailwind CSS забезпечує гнучкість, повний контроль над дизайном та чудову інтеграцію з React, що особливо

актуально для сучасних SPA-застосунків із кастомною структурою компонентів.

Саме завдяки утилітарному підходу, адаптивності, легкій масштабованості та можливості швидкої стилізації Tailwind CSS було обрано для реалізації інтерфейсу системи онлайн-бронювання столів. Цей вибір дозволив уникнути зайвої перевизначеності стилів, зосередитися на логіці застосунку та досягти індивідуального зовнішнього вигляду без втрати продуктивності розробки.

1.4.4 База даних Firebase

Для будь-якого сучасного вебзастосунку, що оперує змінними користувацькими даними, важливим елементом є наявність надійної, масштабованої та зручної у використанні бази даних. У контексті системи онлайн-бронювання столів, база даних є критичним компонентом, оскільки зберігає інформацію про користувачів, доступні столи, активні та завершені бронювання, а також забезпечує безпечну авторизацію і доступ до даних. Саме тому у цьому проєкті було обрано Firebase – хмарну платформу від Google, яка поєднує простоту реалізації, широку функціональність та хорошу інтеграцію з React.

Firebase – це платформа для розробників, яка надає інструменти для створення, розгортання та підтримки веб та мобільних застосунків. Firebase була створена компанією Firebase, Inc., а пізніше придбана Google. Firebase належить до категорії Backend-as-a-Service (BaaS) – це означає, що більшість бекенд-функцій доступна у вигляді готових хмарних сервісів, без потреби створення власного сервера.

У цьому проєкті використовується кілька ключових модулів Firebase.

Firebase Authentication:

– забезпечує механізм автентифікації користувачів за допомогою електронної пошти та пароля;

– підтримує обробку реєстрації, входу, виходу, відновлення паролю;
– дозволяє реалізувати як реєстрацію зареєстрованих користувачів, так і підтримку гостьового режиму бронювання.

Cloud Firestore:

– гнучка, масштабована база даних NoSQL;
– зберігає дані у форматі колекцій та документів (структура подібна до JSON);
– підтримує реальну синхронізацію та офлайн-режим.

У проєкті Firestore використовується для зберігання:

– інформації про користувачів (ім’я, email, телефон);
– бронювань (дата, час, стіл, тривалість);
– статусу столів (вільний / заброньований).

Firebase Hosting:

– дає змогу швидко розгорнути готовий застосунок у продакшен-режимі;
– підтримує HTTPS, глобальну CDN-мережу та автооптимізацію продуктивності.

Безпека, правила доступу:

– Firebase Firestore дозволяє задавати гнучкі правила доступу до даних на рівні колекцій або окремих документів;
– обмеження доступу до резервувань лише для поточного користувача;
– дозвіл на створення бронювання в гостьовому режимі лише з перевіреними полями (ім’я, телефон);
– можливість адміністрування резервувань лише для авторизованих менеджерів (передбачено для майбутнього розвитку системи).

Варто звернути увагу на сильні та слабкі сторони Firebase. Переваги використання Firebase:

– відсутність необхідності у власному сервері значно зменшує складність розгортання та обслуговування застосунку;
– вбудована система безпеки дозволяє отримати гнучкі правила доступу та авторизації;

– підходить як для MVP, так і для великих проєктів з великою кількістю користувачів;
– оновлення даних без перезавантаження сторінки;
– інтеграція з інструментами Google, аналітика, хмарні функції, повідомлення.

Обмеження Firebase:

– ліміт безкоштовного тарифу (кількість операцій, зберігання, трафік) для великих проєктів може знадобитися платний план;
– обмежена структура даних – не підтримує складні SQL-запити або джойни;
– vendor lock-in – важко перенести дані на іншу платформу без переписування логіки.

Firebase забезпечує надійне, швидке й масштабоване серверне середовище для зберігання користувацьких даних, резервувань та автентифікації. Його переваги – інтеграція з React, простота налаштування та відсутність потреби у власному сервері – роблять його ідеальним вибором для MVP і середніх вебзастосунків.

У межах цього проєкту використання Firebase дозволило сконцентруватися на логіці інтерфейсу й взаємодії з користувачем, з мінімальними витратами часу на інфраструктуру.

1.4.5 Допоміжні бібліотеки React

Окрім основних фреймворків і платформ, таких як React, Tailwind CSS і Firebase, у межах цього проєкту було використано низку допоміжних бібліотек, які значно спрощують реалізацію типових задач, покращують продуктивність розробки, забезпечують кращу інтеграцію з інтерфейсом та додають корисний функціонал. У цьому підпункті розглянуто найбільш суттєві сторонні бібліотеки, які доповнили функціональність застосунку.

Date-fns – це сучасна бібліотека для роботи з датами, яка забезпечує прості у використанні та ефективні функції для форматування, обчислення інтервалів і маніпуляції з датами.

У проєкті використовується для:

– форматування вибраної дати бронювання;
– перевірки чи обрана дата не в минулому;
– обчислення тривалості та конфліктів резервувань.

PostCSS – це інструмент для трансформації CSS з використанням JavaScript-плагінів. У поєднанні з autoprefixer автоматично додає вендорні префікси до CSS-властивостей для забезпечення кросбраузерної сумісності.

У проєкті використовується:

– у складі конфігурації Tailwind CSS;
– для оптимізації фінального CSS-коду;
– без необхідності ручного додавання префіксів.

Бібліотека web-vitals – це інструмент від Google, який дозволяє вимірювати ключові показники продуктивності сайту (LCP, FID, CLS тощо). У проєкті використовується для внутрішнього моніторингу продуктивності під час розробки.

Переваги:

– легка інтеграція в React через CRA;
– можливість подальшого використання для аналітики;
– у проєкті застосовується стандартна конфігурація eslint через Create React App (eslintConfig: react-app). Це забезпечує:
– контроль за якістю коду;
– попередження про потенційні помилки;
– дотримання загальноприйнятих стилів.

Пакети @babel/preset-env і @babel/preset-react дозволяють компілювати JSX та сучасні конструкції JavaScript у сумісний код, який підтримується всіма браузерами. Ці пакети входять до складу залежностей проєкту та автоматично використовуються при збірці.

Використання допоміжних бібліотек є важливою частиною реалізації проєкту, забезпечуючи оптимізацію стилів, обробку дат, перевірку якості коду та моніторинг продуктивності. Хоч ці інструменти і не є ключовими, без них реалізація повноцінного, стабільного і зручного у підтримці застосунку була б значно складнішою. Завдяки такому набору засобів вдалося зосередитись на бізнес-логіці й досвіді користувача, мінімізувавши рутинні технічні труднощі.

1.5 Висновки до розділу

У цьому розділі було проведено аналіз ключових технологій, використаних для розробки системи онлайн-бронювання столів у ресторані. Розгляд мови програмування JavaScript та бібліотеки React дозволив обґрунтувати вибір інструментів для створення динамічного одно-сторінкового вебзастосунку. Окрему увагу було приділено порівнянню альтернативних інструментів, таких як Vue.js, Angular, Bootstrap, Next.js та Vite. Це дозволило оцінити переваги й обмеження кожного рішення та переконатися в доцільності вибору саме тих технологій, які було використано в проєкті.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ СИСТЕМИ

2.1 Постановка задачі

У сучасних умовах цифрової трансформації закладів громадського харчування зростає потреба в автоматизованих рішеннях для взаємодії з клієнтами. Однією з актуальних задач є створення системи, яка дозволяє зручно й швидко здійснювати резервування столів у ресторані в онлайн-режимі, без необхідності телефонного дзвінка або особистого візиту. Такий підхід не лише покращує користувацький досвід, а й оптимізує внутрішні процеси закладу.

У межах цієї кваліфікаційної роботи було поставлено задачу розробити вебзастосунок для онлайн-резервування столів, який мав реалізовувати наступні функції:

– вибір дати, часу та тривалості візиту;
– перегляд доступності столів на інтерактивній карті;
– підтвердження резервування з обліковим записом або в гостьовому режимі;
– збереження історії резервувань у профілі користувача.

Також було передбачено можливість подальшого розширення функціональності для адміністрування резервувань, підтримки декількох ролей користувачів (клієнт, менеджер, адміністратор), адаптивності інтерфейсу та розміщення системи в публічному доступі.

Основні функціональні вимоги:

– реалізація автентифікації;
– можливість вибору дати, часу та тривалості з перевіркою коректності;
– відображення статусу столів на карті закладу (вільні / зарезервовані);
– здійснення резервування з фіксацією в базі даних;

– перегляд історії резервувань користувача;
– відображення сповіщень про результат дій.

Нефункціональні вимоги:

– зручний та інтуїтивно зрозумілий інтерфейс користувача;
– висока швидкодія та стабільність роботи інтерфейсу;
– надійне зберігання даних користувачів та резервувань;
– можливість масштабування системи у майбутньому.

У результаті виконання роботи було розроблено односторінковий вебзастосунок, який реалізовував функціональність резервування з використанням сучасного технологічного стеку – React, Firebase, Tailwind CSS та допоміжних бібліотек.

2.2 Загальна архітектура застосунку

Під час розробки вебзастосунку для онлайн-резервування столів було обрано односторінкову архітектуру (SPA) з клієнтською логікою, реалізованою на основі бібліотеки React. Вся основна функціональність, пов’язана з вибором параметрів резервування, інтерактивною картою залу, автентифікацією та взаємодією з базою даних, виконувалась на стороні клієнта. Серверну частину було замінено на базу даних Firebase, завдяки цьому, вдалося зосередитися на розробці функціоналу додатку, зберігши структуру як, в самостійно написаному back-end.

Застосунок було побудовано у вигляді ієрархії компонентів, кожен із яких відповідав за окрему функціональну частину системи. Основні компоненти:

– BookingPage – головна сторінка процесу резервування проєкт, що містила покроковий сценарій;
– TableMap – інтерактивна карта залу з візуалізацією розміщення столів;

– BookingConfirmation – модуль підтвердження вибраного резервування;
– AuthPage – сторінка входу та реєстрації користувача;
– EditProfilePage – сторінка з особистою інформацію користувача та можливістю змінити її;
– Header, Footer, Modal, BookingButton – допоміжні компоненти інтерфейсу.

Усі компоненти взаємодіяли між собою через:

– властивості (props) – для передачі параметрів і функцій;
– локальний стан (useState) – для зберігання обраних значень;
– глобальний стан через Firebase – для відображення актуального статусу столів;
– React Router – для маршрутизації між сторінками та кроками резервування.

Стан системи змінювався динамічно у відповідь на дії користувача, а зміни вносилися до Firestore у режимі реального часу або після підтвердження дії. Архітектура передбачала використання Firebase для зберігання даних та автентифікації користувачів. В застосунок було інтегровано Firestore для роботи з даними користувачів, резервуваннями та статусами столів, а також Firebase Authentication для реалізації реєстрації, входу та гостьового режиму. Комунікація здійснювалась безпосередньо через офіційний SDK Firebase.

Після завершення розробки застосунок було розгорнуто на платформі Vercel, яка забезпечує автоматизоване розгортання односторінкових застосунків (SPA) на основі React. Завдяки інтеграції з GitHub та автоматичній обробці збірки (npm run build), вдалося швидко і зручно підготувати застосунок для публічного доступу через інтернет.

Архітектура застосунку була реалізована відповідно до сучасних принципів компонентного підходу та клієнтсько-орієнтованої розробки. Рішення на базі React та Firebase забезпечило просту, гнучку й ефективну реалізацію функціональності резервування без необхідності створення

складної серверної інфраструктури. Розгортання застосунку на Vercel забезпечило зручний публічний доступ до системи онлайн-резервування та автоматичне оновлення після змін в репозиторії та зручний публічний доступ до системи онлайн-резервування. Такий підхід дозволив сфокусуватись на інтерфейсі, зручності для користувача та адаптивній взаємодії з системою.

2.3 Основні сторінки та компоненти застосунку

У процесі розробки вебзастосунку для онлайн-резервування столів було створено низку функціональних сторінок і компонентів, які забезпечували логічну структуру інтерфейсу, інтерактивність та зручність користування. Завдяки компонентному підходу, що властивий бібліотеці React, інтерфейс було розбито на окремі блоки, кожен з яких відповідав за свою частину функціональності та міг повторно використовуватись.

Головна сторінка містить:

– логотип та/або назву ресторану;
– кнопку переходу до сторінки резервування;
– кнопку переходу до Google maps, з точним місцезнаходженням закладу;
– відображення графіку роботи ресторану та опису закладу.

Інтерфейс головної сторінки було реалізовано відповідно до спрощеного макета, який наведено рисунку 2.1.

Сторінку резервування, було реалізовано як покрокову форму резервування:

– вибір дати, часу та тривалості;
– вибір столу на інтерактивній мапі;
– перевірка та підтвердження резервування.

Було забезпечено перевірку валідності введених даних (дата, час, тривалість). Створено підтримку, як зареєстрованих користувачів, так і гостей.

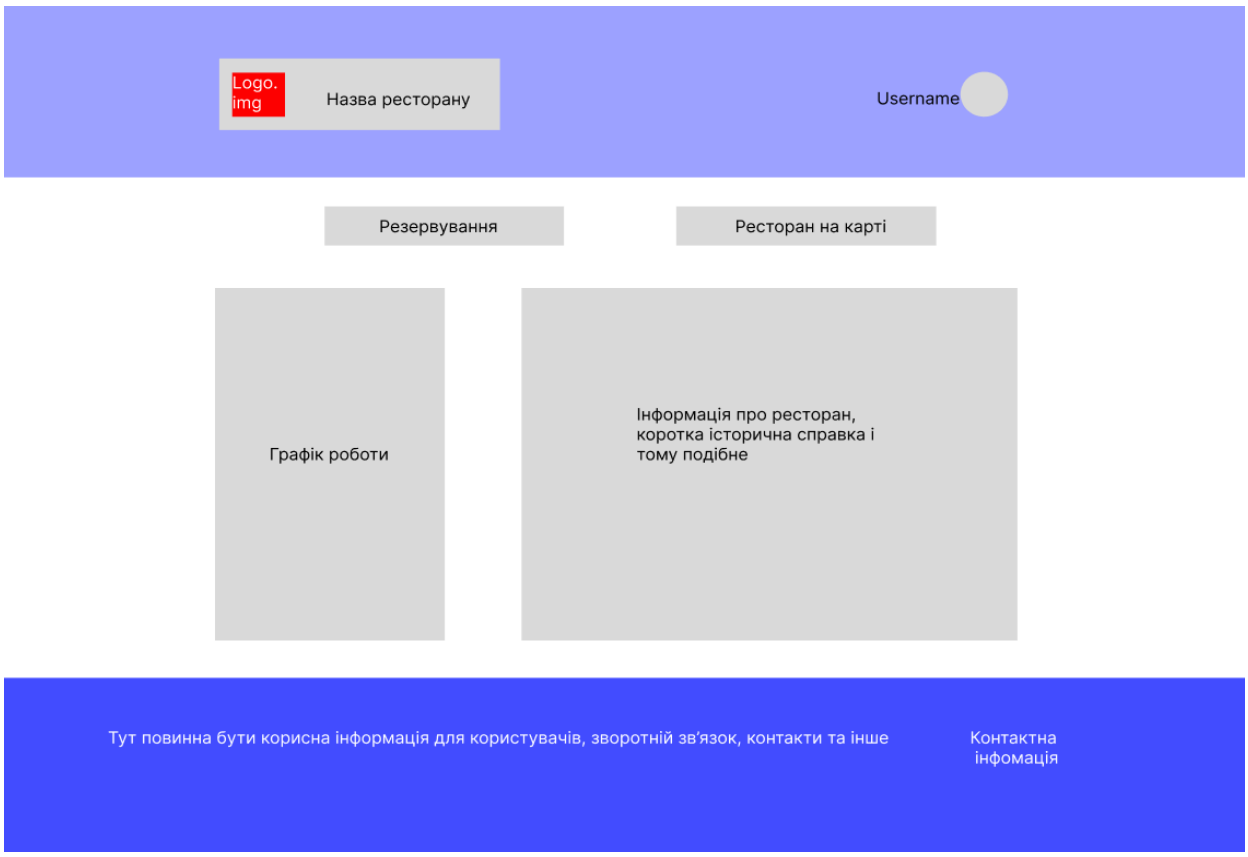


Рисунок 2.1 – Макет головної сторінки

Сторінка автентифікації, включає реєстрацію нового користувача та вхід для наявного. Було створено перевірку на правильність введеного email, паролю, імені, прізвища та телефону. Користувач має можливість перемикатися між режимами входу та реєстрації.

Профіль користувача було вирішено розділити на дві сторінки. Сторінка перегляду/зміни особистих даних та сторінка історії резервувань. На сторінці історії резервувань, зображуються майбутні та завершені резервування, також є можливість скасувати майбутнє резервування. На сторінці перегляду/зміни особистих даних було забезпечено зміну та перегляд персональних даних (email, ім'я, телефон).

До основних компонентів інтерфейсу відносяться: TableMap, BookingConfirmation, EditProfileForm.

Header, Footer, BaseModal, BookingButton, ShowOnMapButton, AboutRestaurant, WorkingHours відповідають за відображення інформації, навігації між сторінками та повідомлення. Для маршрутизації між сторінками

було використано react-router-dom. Навігація між кроками реалізовувалась через useState та useNavigate. Обробка даних здійснювалась у відповідних компонентах, а результат взаємодії з Firebase (додання документу, авторизація) – через асинхронні функції.

Усі сторінки та компоненти було створено з урахуванням логіки резервування, простоти навігації та повторного використання. Компонентна структура застосунку забезпечила зручну організацію коду, розділення відповідальностей і легкість у підтримці. Завдяки чіткому розділенню сторінок за функціональністю, користувач міг швидко знайти потрібну інформацію, здійснити резервування та переглянути свої дані без зайвих дій.

2.4 Реалізація процесу резервування

Процес резервування був основною функціональністю розробленого вебзастосунку. Його реалізація вимагала побудови покрокового сценарію взаємодії користувача з інтерфейсом, валідації введених даних, вибору доступного столика та фіксації резервування у базі даних Firebase.

На першому кроці користувач має можливість обрати дату, час початку та тривалість резервування. Для цього використовуються елементи введення input типу date, time та числовий спінер для тривалості.

Щоб забезпечити безпечний процес резервування для користувача та системи, було реалізовано валідацію введених даних:

– заборонено вибирати дати в минулому;
– час резервування обмежувався робочим графіком ресторану;
– тривалість резервування не могла була не меншої мінімальні дві години, які встановлювалися за замовчуванням, але цей час міг бути збільшеним за потреби користувача.

Після вибору дати, часу та тривалості резервування, користувач переходив до другого кроку вибору столу на інтерактивній карті ресторану (TableMap):

- столи відображаються із зазначенням їхнього типу на 2, 4 та 6 осіб;
- столи, що вже були зарезервовані на вибраний час, автоматично позначаються як недоступні;
- у користувача була можливість натиснути на вільний стіл для його вибору, а також змінити свій вибір у будь-який час.

Логіка доступності столу базувалась на перевірці існуючих резервувань у Firestore у вибраний проміжок часу та дату.

Після вибору столу, користувач переходив до третього кроку, на якому перевіряється коректність вибраної дати, часу, тривалості резервування та столу, якщо користувача не влаштовував будь-який з вибраних ним параметрів, користувач міг змінити їх у будь-який момент.

У випадку, коли користувача все влаштовує, залишається підтвердити резервування. Після підтвердження користувача переправляє на сторінку з історією резервувань, але тільки в тому випадку, якщо користувач увійшов до системи.

У випадку коли користувач не увійшов до системи, система розпізнавала його, як гостя і після підтвердження не переправляла на сторінку історії резервувань, а позувало модальне вікно з інформацією про успішне резервування.

На рисунку 2.2 зображено діаграму використання системи клієнтом, користувачем, який увійшов до системи. У цьому випадку користувач має всю можливо повноту дій зв’язану з керуванням процесу резервування.

У випадку, коли користувач не виконав вхід до системи, система розпізнає користувача як Гостя. В цьому випадку у користувача не буде таких можливостей: переглядати історію своїх резервувань, здійснювати скасування в своєму профілі, змінювати особисті дані, також користувача знадобиться вводити ім’я, прізвище та номер телефону на етапі підтвердження резервування.

На рисунку 2.3 зображено діаграму варіантів використання системи користувачем, який не зареєструвався та не увійшов до системи.

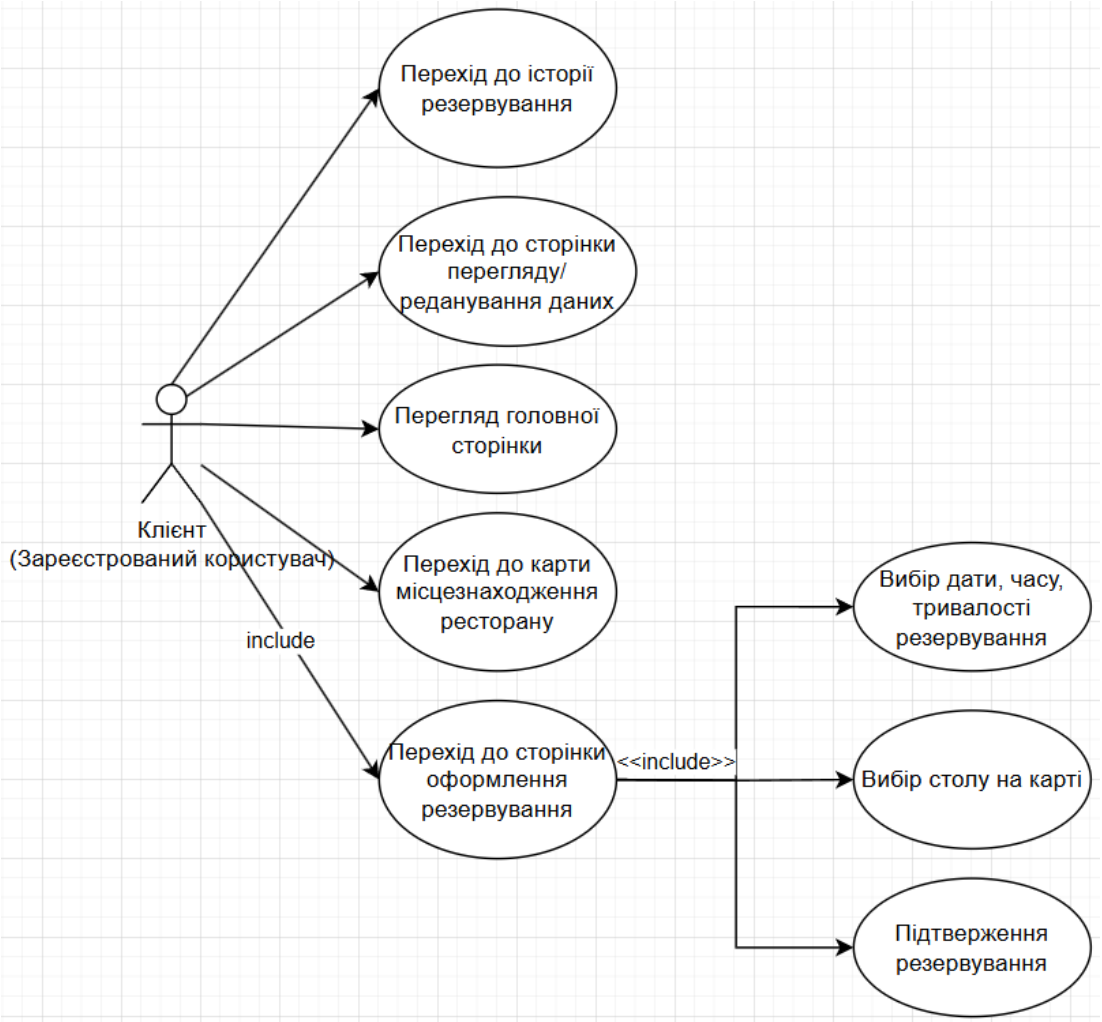


Рисунок 2.2 – Діаграма варіантів використання системи клієнтом

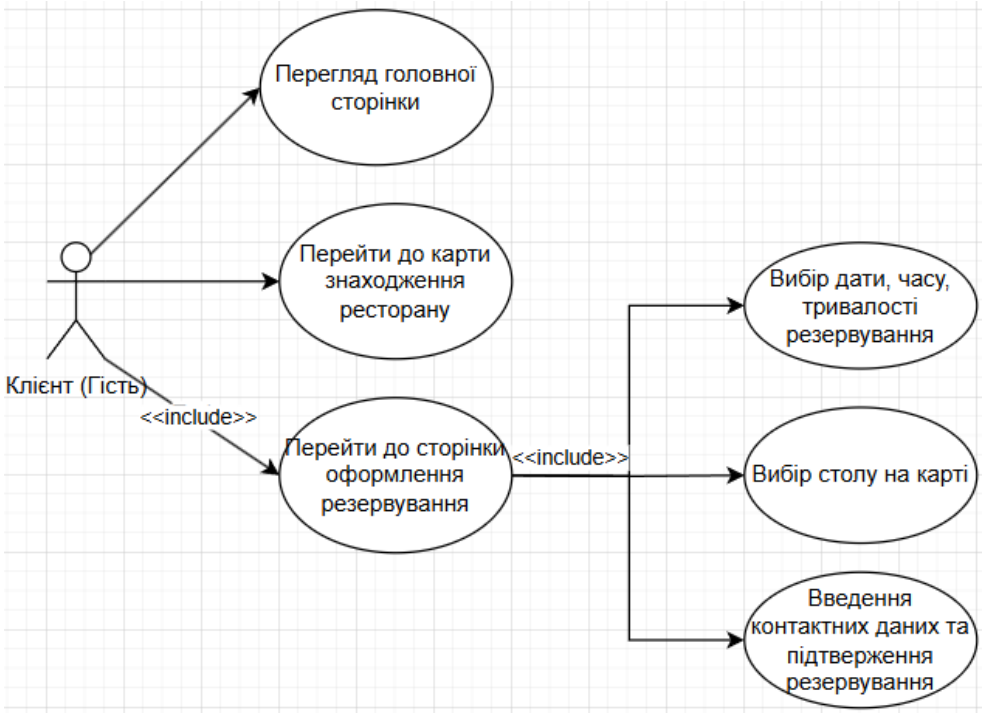


Рисунок 2.3 – Діаграма варіантів використання системи гостем

Після підтвердження бронювання, користувач побачить модальне вікно підтвердження для гостей, з інформацією про деталі резервування після його успішного створення. Резервування зберігається у Firestore у відповідну колекцію. Статус обраного столика оновлювався для запобігання повторному вибору. Перед збереженням резервування застосунок здійснює перевірки:

– чи не перетинається нове резервування з існуючими за часом;
– чи залишається вибраний стіл доступним на момент підтвердження.

У випадку конфлікту користувачу виводиться повідомлення про необхідність вибору іншого столу або зміни часу.

Для забезпечення коректної роботи були передбачені:

– відображення повідомлень про помилки при невдалому збереженні резервування;
– валідація всіх полів перед переходом до наступного кроку;
– обробка випадків втрати з’єднання з Firebase.

Процес резервування було реалізовано у вигляді покрокової форми з перевіркою всіх необхідних даних і інтерактивною взаємодією з картою залу. Такий підхід забезпечив логічну послідовність дій користувача, зменшив кількість помилок і зробив процес резервування максимально простим та зручним.

Завдяки інтеграції з Firestore резервування зберігалися надійно та могли бути використані для подальшого управління відвідуваннями закладу.

2.5 Обробка даних користувача

Однією з важливих складових функціональності розробленого вебзастосунку стала обробка даних користувача. У межах системи онлайн-резервування було реалізовано механізми автентифікації, реєстрації, роботи в гостьовому режимі, зберігання профільної інформації та управління резервуваннями користувачів.

У застосунку було реалізовано стандартний механізм автентифікації через Firebase Authentication, що передбачав:

– реєстрацію нового користувача за допомогою електронної пошти та пароля;
– вхід у систему з перевіркою правильності облікових даних;
– можливість виходу зі свого акаунта.

Під час реєстрації додатково збирались особисті дані користувача: ім’я, прізвище, номер телефону. Усі дані зберігалися у Firestore у відповідному документі профілю користувача.

Для забезпечення зручності використання застосунку було реалізовано можливість резервування столу без попередньої реєстрації. У гостьовому режимі користувач:

– вводив мінімально необхідні дані (ім’я, прізвище, номер телефону) на етапі підтвердження резервування;
– міг створити одноразове резервування без створення облікового запису;
– отримував повідомлення з деталями резервування після підтвердження;
– гостьове резервування також зберігалося у Firestore, позначене як гостьове у відповідному полі.

Після реєстрації, користувач має можливість:

– переглядати особисті дані;
– змінювати особисті дані;
– передивлятися історію резервувань;
– скасовувати майбутні резервування.

Організація структури даних у Firebase. У Firestore було організовано окремі колекції:

– users – зберігання профільної інформації користувачів;
– reservations – зберігання даних про всі резервування, з прив’язкою до користувача або гостьового бронювання;
– tables – опис столів, їхнього типу та статусу.

Між колекціями встановлюються логічні зв’язки через унікальні ідентифікатори користувачів (UID) або відповідні параметри резервувань.

Для безпечної роботи з даними користувачів було налаштовано:

– обмеження доступу до документів у Firestore через правила безпеки;
– можливість читання та запису лише своїх особистих даних і резервувань;
– валідацію обов’язкових полів при реєстрації, вході та резервуванні.

Кожен запит до бази даних проходив через перевірку авторизації або обмеження для гостьових користувачів.

Обробка даних користувачів була реалізована таким чином, щоб забезпечити простоту реєстрації та взаємодії із системою, зберігаючи при цьому безпеку, конфіденційність і коректність збереження даних. Механізм автентифікації, профільна інформація та можливість роботи в гостьовому режимі дозволили розширити доступність сервісу та підвищити комфорт користувачів під час резервування столів.

2.6 Інтерактивна карта ресторану

Однією з ключових особливостей розробленого вебзастосунку стала інтерактивна карта залу ресторану, яка дозволяла користувачам здійснювати вибір столика для резервування у візуальному режимі. Реалізація інтерактивної карти сприяла підвищенню зручності взаємодії з системою та наближала процес вибору до реального розташування об’єктів у ресторані.

Розробка макета карти. Перед початком реалізації було створено макет розташування столів та елементів інтер’єру ресторану. Макет містив:

– столи різних типів (на 2, 4, 6 осіб);
– позначення входу до ресторану;
– розташування вікон, каміна, санвузла;
– логічне розміщення проходів між столами.

На основі макета було визначено координати для кожного столика у відносних одиницях (top, left у відсотках). На рисунку 2.4 представлено візуальний макет розміщення столів, який було покладено в основу компонента інтерактивної карти ресторану.

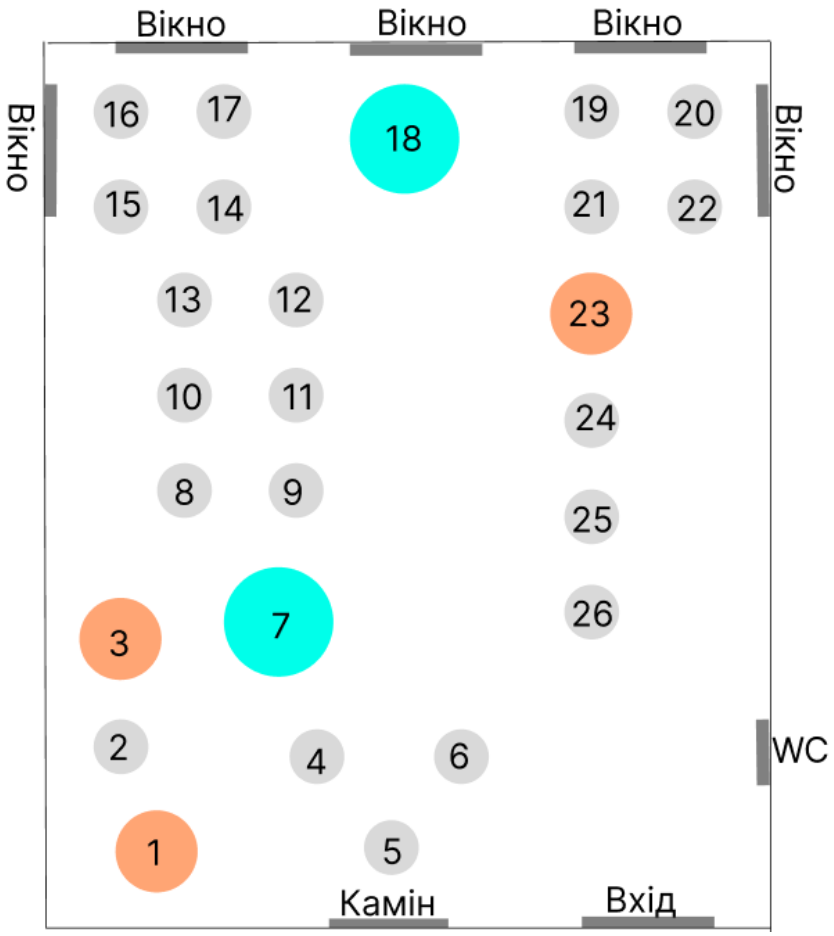


Рисунок 2.4 – Макет карти ресторану

TableMap відповідає за відображення інтерактивної карти столів. Візуалізує їх розташування, відображає статус столу, вільний або зарезервований, також відображає тип столу при наведені. Дозволяє користувачу обрати вільний стіл шляхом кліку. Реагує на обрану дату та час, змінюючи доступність столів.

Для кожного столу у коді зберігається:

– тип столу;
– координати розташування;
– статус на поточний момент, який залежить від вибраної дати і часу).

Доступність столів на мапі визначається за даними з бази Firestore. Під час завантаження карти, застосунок отримує інформацію про активні резервування на обрану дату та час. Столи, які вже були зарезервовані, відображаються сірим кольором і є недоступними для вибору. Вільні столи залишалися активними та можуть бути обрані для резервування. Вибраний користувачем стіл підсвічується окремим стилем.

Інтерактивна карта ресторану стала важливою складовою застосунку, яка забезпечила наочність, інтуїтивність та простоту процесу вибору столика. Реалізація візуальної карти із динамічною зміною статусу столів на основі даних Firestore дозволила підвищити користувацький досвід та зробити процес резервування максимально близьким до реального вибору місця в ресторані.

2.7 Робота з Firebase

У процесі розробки вебзастосунку для онлайн резервування столів було використано платформу Firebase як основне хмарне рішення для зберігання даних та автентифікації користувачів. Завдяки своїй простоті інтеграції, масштабованості та широкому функціоналу, Firebase дозволив реалізувати бекенд-сервіси без створення окремого серверного застосунку.

Сервіс Firebase Authentication було використано для реалізації функціональності реєстрації нового користувача з електронною поштою і паролем та входу до облікового запису для авторизованих користувачів.

Firebase Authentication забезпечив:

– збереження безпечного токена сеансу;
– отримання унікального UID для кожного користувача, що використовувався для зв’язку з іншими колекціями у базі даних;
– можливість автоматичного відстеження стану автентифікації (onAuthStateChanged), що використовувалося в React для оновлення інтерфейсу.

Для зберігання даних у Firebase використовується Cloud Firestore – гнучка NoSQL-базу даних. У Firestore було створено кілька основних колекцій:

– users – зберігає профільну інформацію користувачів (ім’я, email, телефон);
– reservations – містить дані про всі резервування (дата, час, тривалість, статус, UID або гостьова інформація);
– tables – містить характеристики столів (номер, тип, позиція на карті).

Firestore дозволяє:

– здійснювати CRUD-операції з документами (створення, читання, оновлення, видалення);
– формувати запити з умовами (where, orderBy) для перевірки доступності столів;
– реалізувати реальну синхронізацію даних;
– працювати з вкладеними колекціями.

Реалізація логіки взаємодії. Інтеграція Firebase з React відбувалася через SDK firebase. У застосунку було створено окремий модуль (firebase.js), у якому:

– ініціалізувався застосунок за допомогою ключів проєкту;
– експортувалися об’єкти auth та db (для автентифікації та доступу до бази даних);
– використовувались функції getDocs, addDoc, updateDoc, onSnapshot тощо.

Компоненти зверталися до Firestore або Authentication через ці об’єкти для реалізації потрібної логіки:

– додавання нового резервування;
– перевірка наявності конфлікту часу;
– отримання історії резервувань;
– збереження даних користувача під час реєстрації.

Для захисту даних у Firestore було налаштовано правила безпеки, що передбачали:

– доступ до колекції reservations лише для автентифікованого користувача з відповідним UID або для гостьового внесення з обмеженим обсягом даних;
– обмеження на зміну даних користувачів іншими користувачами;
– перевірку авторизації при кожному запиті до бази даних.

Використання Firebase дозволило реалізувати усі необхідні функції зберігання даних, автентифікації та захисту без створення власного сервера. Завдяки зручному SDK, інтеграція з React відбувалася швидко та стабільно, а гнучка структура Firestore забезпечила ефективне управління даними про користувачів, столики та резервування. Такий підхід значно скоротив час розробки та спростив подальшу підтримку системи.

2.8 Забезпечення надійності та валідація

Одним із пріоритетів під час реалізації вебзастосунку було забезпечення надійної роботи інтерфейсу, коректної обробки даних та захисту від некоректного введення або помилкових дій користувача. Усі ключові етапи взаємодії з системою були доповнені валідаційними перевірками та механізмами обробки помилок, що дозволило покращити стабільність застосунку та користувацький досвід.

Валідацію було реалізовано на етапах реєстрації, входу, створення резервування та підтвердження дій.

Реєстрація:

– перевірка правильності email-адреси;
– мінімальна довжина пароля;
– обов’язкове заповнення імені, прізвища та телефону;
– відображення повідомлень у разі некоректного введення.

Резервування:

– заборона вибору дати в минулому;
– вимога обрати хоча б один доступний стіл;

– перевірка відповідності часу робочим годинам закладу (будні: 12:00–23:00, вихідні: 11:00–23:00);
– перевірка, щоб тривалість резервування була не менше 30 хвилин;
– валідація гостьових даних (ім’я, телефон) перед підтвердженням.

Перед збереженням нового резервування здійснювалась перевірка на конфлікти з існуючими бронюваннями:

– запити до Firestore фільтрували резервування за вибраною датою та столиком;
– за допомогою бібліотеки date–fns обчислювалось, чи перетинається інтервал часу;
– якщо конфлікт виявлявся, користувачеві виводилось повідомлення з рекомендацією змінити час або обрати інший стіл.

У застосунку було реалізовано систему сповіщень:

– про успішне резервування (в тому числі через модальне вікно для гостей);
– про помилки введення або спробу доступу до недоступного функціоналу;
– про помилки з’єднання з Firebase або проблеми авторизації.

Це дозволило уникнути ситуацій, коли користувач не розумів, чи його дія була виконана успішно.

Було реалізовано механізми обробки винятків (try/catch) у запитах до Firebase:

– під час реєстрації або входу (наприклад, неправильний пароль або вже існуючий email);
– під час збереження або читання резервувань;
– під час взаємодії з таблицею доступних столів.

Усі помилки логувались у консоль для зручності тестування та були відображені користувачу у формі дружніх повідомлень.

Завдяки онлайн-запитам до Firestore, при виборі часу та столика враховувались актуальні дані в реальному часі. Була реалізована повторна

перевірка доступності столика на момент підтвердження резервування. Це запобігало ситуаціям, коли кілька користувачів могли одночасно обрати один і той самий стіл.

Сукупність реалізованих механізмів валідації, обробки виняткових ситуацій і користувацьких сповіщень дозволила забезпечити високу стабільність і передбачуваність поведінки застосунку. Завдяки перевірці даних на всіх ключових етапах взаємодії з системою, а також контролю за конфліктами резервування, було досягнуто надійного функціонування вебзастосунку навіть при паралельній роботі кількох користувачів.

2.9 Висновки до розділу

У другому розділі було детально розглянуто процес проєктування та реалізації вебзастосунку для онлайн-резервування столів у ресторані. Було сформульовано основні вимоги до системи, визначено її архітектуру та описано ключові етапи реалізації функціональності з використанням бібліотеки React та хмарного середовища Firebase.

3 РЕАЛІЗАЦІЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ ВЕБЗАСТОСУНКУ

3.1 Підготовка середовища розробки

Для реалізації вебзастосунку онлайн-резервування столиків було обрано технології, що дозволяли швидко створити односторінковий застосунок (SPA) з сучасною архітектурою і масштабованою структурою.

Проєкт було створено за допомогою офіційного інструменту Create React App, який забезпечив базову конфігурацію для роботи з React, включаючи підтримку JSX, Babel, Webpack та ESLint. Створення проєкту було здійснено за допомогою команди: “npx create-react-app table-reserv” .

Для реалізації маршрутизації між сторінками було встановлено React Router Dom за допомогою команди “npm i react-router-dom”.

Для автентифікації користувачів та зберігання даних про резервування було встановлено Firebase за допомогою команди “npm install firebase”. Для використання Firebase, необхідно створити файл firebase.js та заповнити його конфігураційними даними проєкту створеними раніше на сайті Firebase. На рисунку 3.1 зображено код за допомогою якого підключається Firebase.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyC4cyMkB00mqlzbJg_7O_Wrj2qo9DTE_aU",
  authDomain: "table-reserv-ab094.firebaseapp.com",
  projectId: "table-reserv-ab094",
  storageBucket: "table-reserv-ab094.firebasestorage.app",
  messagingSenderId: "632081945088",
  appId: "1:632081945088:web:cf07a61bab3c447eea3ee5"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Рисунок 3.1 – Приклад коду для підключення Firebase

Для налаштування інтерфейсу користувача використовувався CSS фреймворк Tailwind CSS та допоміжні інструменти postcss і autoprefixer. Для їх встановлення необхідно використати таку команду “npm install -D tailwindcss postcss autoprefixer”. Після встановлення, необхідно ініціалізувати Tailwind за допомогою команди: “npx tailwindcss init -p”. Після введення команди буде створено файл конфігурації Tailwind – tailwind.config.js та файл конфігурації PostCSS – postcss.config.js.

Для правильної роботи Tailwind з React, необхідно налаштувати tailwind.config.js вказавши шлях до всіх компонентів проєкту. На рисунку 3.2 наведено налаштований postcss.config.js.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

Рисунок 3.2 – Налаштований postcss.config.js

Останнім кроком буде підключення Tailwind до CSS, для цього буде достатньо вписати в index.css основні директиви: @tailwind base, @tailwind components, @tailwind utilities. Після цього Tailwind буде налаштованим і готовим для використання в проєкті.

Ще одна допоможна бібліотека date-fns – необхідна для роботи з датами та форматуванням часу. Ця бібліотека швидша та зручніша за інші аналоги і вбудовану Date. Встановлюється за допомогою команди: “npm install date-fns”.

Підсумок до пункту. На етапі підготовки середовища розробки було забезпечено створення повноцінної основи для подальшої реалізації застосунку: налаштовано систему збірки, маршрутизації, стилізації та зберігання даних, що дозволило зосередитись на розробці функціональності відповідно до поставлених вимог.

3.2 Головний функціонал додатку

Першим етапом розробки застосунку після налаштування середовища була створена функціональна частина – зокрема, логіка обробки даних, зберігання інформації про користувачів і резервування.

У Cloud Firestore було створено три основні колекції:

– users – для зберігання інформації про зареєстрованих користувачів (ім’я, прізвище, email, телефон);
– reservations – для зберігання інформації про всі створені резервування (дата, час, тривалість, стіл, статус, гостьова або користувацька форма);
– tables – для опису столів у залі (тип, позиція, статус).

Кожна колекція містить документи з унікальним ідентифікатором. Для зручності взаємозв’язків між об’єктами зберігалається UID користувача або гостьові дані, ідентифікатор столу та часові мітки створення.

Взаємодія з Firestore відбувалась за допомогою зчитування та запису даних. На рисунку 3.3 продемонстровано процес додавання нового документа до колекції reservations у хмарній базі даних Firestore.

```
await addDoc(collection(db, "reservations"), {
  userId,
  tableId,
  date,
  time,
  duration,
  status: "active",
  createdAt: new Date(),
  guestInfo,
});
```

Рисунок 3.3 – Додавання нового запису резервування до колекції Firestore

Метод addDoc() створює новий запис з такими ключовими полями, як:

– userId – ідентифікатор зареєстрованого користувача або null у разі гостьового бронювання;
– tableId – номер столика, що резервується;

– date, time – дата та час початку резервування;
– duration – тривалість у годинах;
– status – поточний статус бронювання;
– createdAt – мітка часу створення запису;
– guestInfo – об’єкт із контактною інформацією гостя (для незареєстрованих користувачів).

Було реалізовано вхід і реєстрацію через Firebase Authentication. На рисунку 3.4 зображено приклад взаємодії з Firebase Authentication при спробі користувача зареєструватися.

```
if (isRegistering) {
  const userCredential = await createUserWithEmailAndPassword(
    auth,
    email,
    password
  );
  const user = userCredential.user;

  await setDoc(doc(db, "users", user.uid), {
    name,
    surname,
    email,
    phone,
  });
  navigate(redirectTo);
```

Рисунок 3.4 – Реалізація запису реєстрації користувача у Firestore

Детальний огляд фрагменту коду на рисунку наведеного вище:

– спочатку створюється обліковий запис через Firebase (createUserWithEmailAndPassword);
– далі отримується UID нового користувача;
– після цього у базу Firestore записується документ із додатковими даними користувача (ім’я, прізвище, телефон, email) у колекцію users;
– у разі успіху, користувча перенаправляють на вибрану сторінку після успішної реєстрації, в цьому випадку під змінною redirectTo знаходиться шлях на головну сторінку.

На рисунку 3.5 зображено фрагмент коду реалізації входу користувача до системи до допомого електронної адреси та паролю.

```
else {
const userCredential = await signInWithEmailAndPassword(
  auth,
  email,
  password
);

const user = userCredential.user;
const userRef = doc(db, "users", user.uid);
const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {
  throw new Error("Користувача не знайдено в базі.");
}

navigate(redirectTo);
```

Рисунок 3.5 – Реалізація входу користувача

Для кращого розуміння процесу входу користувача до системи, треба детально розглянути що відбувається на цьому фрагменті коду:

– спочатку виконується вхід користувача за email і паролем;
– далі отримуємо UID користувача;
– після чого відбувається перевірка чи існує документ з даними користувача в базі users;
– якщо такого документа немає – користувач отримує повідомлення, а вхід не виконується;
– якщо такий документ є – користувач входить до системи та переправляється до головної сторінки.

Головним фунціоналом додатку є процес резервування столів у ресторані. Процес створення нового резервування реалізовано у компоненті BookingConfirmation. Після того, як користувач обирав дату, час, тривалість і стіл, відбувалось формування об’єкта резервування та його збереження в колекцію reservations. Для зареєстрованих користувачів у документ додається поле userId, а для гостей – блок з контактною інформацією (guestInfo).

На рисунку 3.6 наведено фрагмент коду резервування столу.

```
await addDoc(collection(db, "reservations"), {
  userId,
  tableId,
  date,
  time,
  duration,
  status: "active",
  createdAt: new Date(),
  guestInfo,
});
```

Рисунок 3.6 – Фрагмент коду запису резервування столу до Firestore

Це асинхронна операція, яка додає новий документ до колекції reservations у Firestore. Кожне резервування містить усі необхідні дані, щоб ідентифікувати користувача, стіл, дату, час і стан дії. Цей спосіб можна назвати універсальним, бо одна функція обробляє як резервування зареєстрованого користувача, так і гостяю. Якщо резервування здійснює гість – userId буде null, а guestInfo міститиме ім’я, прізвище та номер телефону. Якщо користувач авторизований – передається userId, а guestInfo може не включатись.

Також у Firestore було налаштовано правила безпеки:

– зареєстрований користувач міг читати та змінювати лише свої резервування;
– гостьове резервування дозволялось лише з перевіркою обмежених полів;
– неавторизовані запити до інших колекцій блокувались.

Організація роботи з даними забезпечила стабільне зберігання, перевірку та обробку резервувань, а також персоналізовану взаємодію користувача із системою. Завдяки використанню Firebase було досягнуто високої швидкодії, безпеки та гнучкості доступу до даних, що стало основою для подальшої реалізації інтерфейсу та логіки взаємодії користувача з системою.

3.3 Реалізація інтерфейсу

У межах реалізації застосунку для онлайн–резервування столів було розроблено інтерфейс користувача, що відповідав принципам зручності, доступності та простоти.

Інтерфейс було створено з використанням бібліотеки React із застосуванням компонентного підходу.

Для стилізації використовувалась бібліотека Tailwind CSS, що дозволила забезпечити адаптивність і сучасний зовнішній вигляд без написання додаткових CSS-файлів.

Інтерфейс було поділено на логічно ізольовані компоненти:

- Header – верхня панель з логотипом та кнопками навігації;
- MainContent – головна сторінка з інформацією про ресторан, графіком роботи, історією закладу та кнопками переходу до резервування;
- BookingPage – сторінка, в якій реалізовувано покроковий процес резервування;
- TableMap – інтерактивна карта ресторану, на якій відображається доступні та зарезервовані столики з урахуванням їх типу та розташування;
- BookingConfirmation – компонент для підтвердження резервування, і введення контактних даних у гостьовому режимі;
- Footer – нижня панель з контактною інформацією та навігацією;
- AuthPage – сторінка входу та реєстрації користувача;
- BookingsPage – сторінка на якій зберігаються резервування користувача;
- EditProfilePage – сторінка на якій відображається особисті дані користувача та можливість їх змінити;
- Modal-компоненти – GuestSuccessModal, ModalWarning, BaseModal для відображення сповіщень і діалогових вікон.

Адаптивну верстку було реалізовано за допомогою утилітарних класів Tailwind CSS. Інтерфейс коректно відображається на різних розмірах екранів. Для гнучкого компонування елементів застосовувались класи:

– flex, grid, gap, justify-between, items-center;
– медіа-запити (sm:, md:, lg:) – для мобільної та десктопної версій;
– max-w, min-h, w-full, h-auto – для адаптації розмірів блоків.

Реактивна поведінка була забезпечена завдяки використанню хуків React:

– useState – для зберігання стану вибраної дати, часу, тривалості, столика тощо;
– useEffect – для завантаження даних з Firebase;
– useNavigate – для переходу між кроками;
– useSearchParams – для передачі стану в URL (наприклад, гостьовий режим).

Інтерфейс реагує на дії користувача в режимі реального часу: вибрані дати, доступність столів, підтвердження резервування – усе оновлювалося миттєво без перезавантаження сторінки.

Особливу увагу варто звернути на реалізацію інтерактивної карти ресторану, так як реалізація цього фунціоналу найважливіша в проєкті. Для вибору столу в залі ресторану було реалізовано компонент TableMap, що відображає карту закладу з візуальним представленням усіх столів. Усі столи мають унікальні ідентифікатори (T1–T26), а також атрибути типу (на 2, 4 або 6 осіб), координати позиціонування (top, left) та статус (available, reserved). На рисунку 3.7 зображено приклад того, як інформація про столи зберігається у Firebase (колекція tables) та локальній структурі в компоненті.

Кожен стіл відображається як елемент з позиціонуванням через Tailwind CSS (absolute, top, left) і маж колір, що відповідає його типу:

– бірюзовий – на 6 осіб;
– помаранчевий – на 4 особи;
– жовтий – на 2 особи;
– сірий – недоступний для резервування.

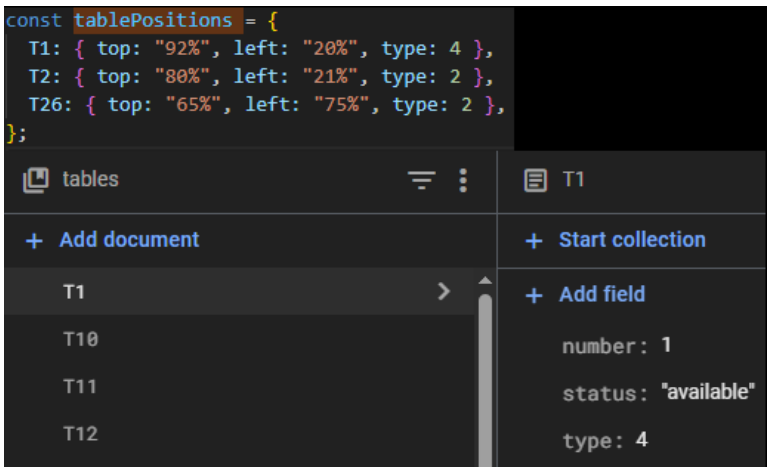


Рисунок 3.7 – Приклад збереження інформації про столи

Під час завантаження карти застосунок перевіряє статуси столів на обрану дату й час. Столи, що вже були зарезервовані, неможливо вибрати, бо їх колір змінювався на сірий та відбувалося блокування pointer-events, тобто користувач ніяк не міг вибрати недоступний стіл.

На рисунку 3.8 представлено фрагмент JSX-коду, який відповідає за відображення столів на інтерактивній карті ресторану.

```
{Object.entries(tablePositions).map(([id, pos]) => {
  const table = tables[id];
  const isSelected = selectedTableId === id;
  const size = getSize(table);

  return (
    <button
      key={id}
      onClick={() => table?.status !== "reserved" && onTableSelect(id)}
      title={`Стіл №${id.replace("T", "")}, ${table?.type} місць${
        table?.status === "reserved" ? " (заброньовано)" : ""
      }`}
      className={`absolute rounded-full text-sm font-bold flex items-center justify-center border
        ${getColor(table)}
        ${isSelected ? "ring-4 ring-green-500" : ""}
        ${
          table?.status !== "reserved"
            ? "hover:ring hover:ring-blue-400"
            : ""
        }
        transition-transform duration-200 ease-in-out
      `}
      style={{
        top: pos.top,
        left: pos.left,
        width: `${size}px`,
        height: `${size}px`,
        transform: "translate(-50%, -50%)",
        cursor: table?.status === "reserved" ? "not-allowed" : "pointer",
      }}
    >
      {id.replace("T", "")}
    </button>
```

Рисунок 3.8 – Фрагмент коду, що відповідає за відображення столів на карті

Кожен стіл позиціонується за допомогою абсолютних координат top і left, має колір, який залежить від типу столика, та статус – вільний або зарезервований. Заброньовані столи неактивні для вибору (pointer-events: none). У разі вибору столика – до нього застосовується візуальне підсвічування (ring-4 ring-green-500), що полегшує навігацію для користувача.

Ще одною особливістю інтерактивної карти ресторану, є відображення елементів інтер'єру. На рисунку 3.9 зображено фрагмент коду, який відповідає за відображення елементів інтер`єру, додаткова інформація допомагає користувачу краще зрозуміти де буде знаходитися його стіл.

```
<div className="relative w-full aspect-[3/4] bg-white border rounded-md shadow-lg">
  {furnitureElements.map((el) => (
    <div
      key={el.id}
      className="absolute bg-gray-100 text-gray-800 text-xs font-medium border rounded-full px-2 py-0.5 shadow-sm"
      style={{
        top: el.top,
        left: el.left,
        transform: "translate(-50%, -50%)",
        whiteSpace: "nowrap",
      }}
    >
      {getFurnitureLabel(el.label)}
    </div>
```

Рисунок 3.9 – Фрагмент коду, що відповідає за відображення елементів інтер`єру

Інтерфейс користувача було реалізовано відповідно до сучасних принципів розробки SPA-застосунків. Компонентна архітектура React, у поєднанні з Tailwind CSS, дозволило створити адаптивний, зручний і візуально привабливий інтерфейс, який забезпечував комфортну взаємодію користувача з системою резервування на всіх етапах. Особливу увагу було приділено інтерактивній карті залу, яка дозволяла інтуїтивно обирати доступний стіл, орієнтуючись на реальне розташування об'єктів у ресторані.

3.4 Демонстрація роботи системи

Для кращого розуміння послідовності виконання дій користувачем для досягнення успішного резервування, було створено діаграму активності.

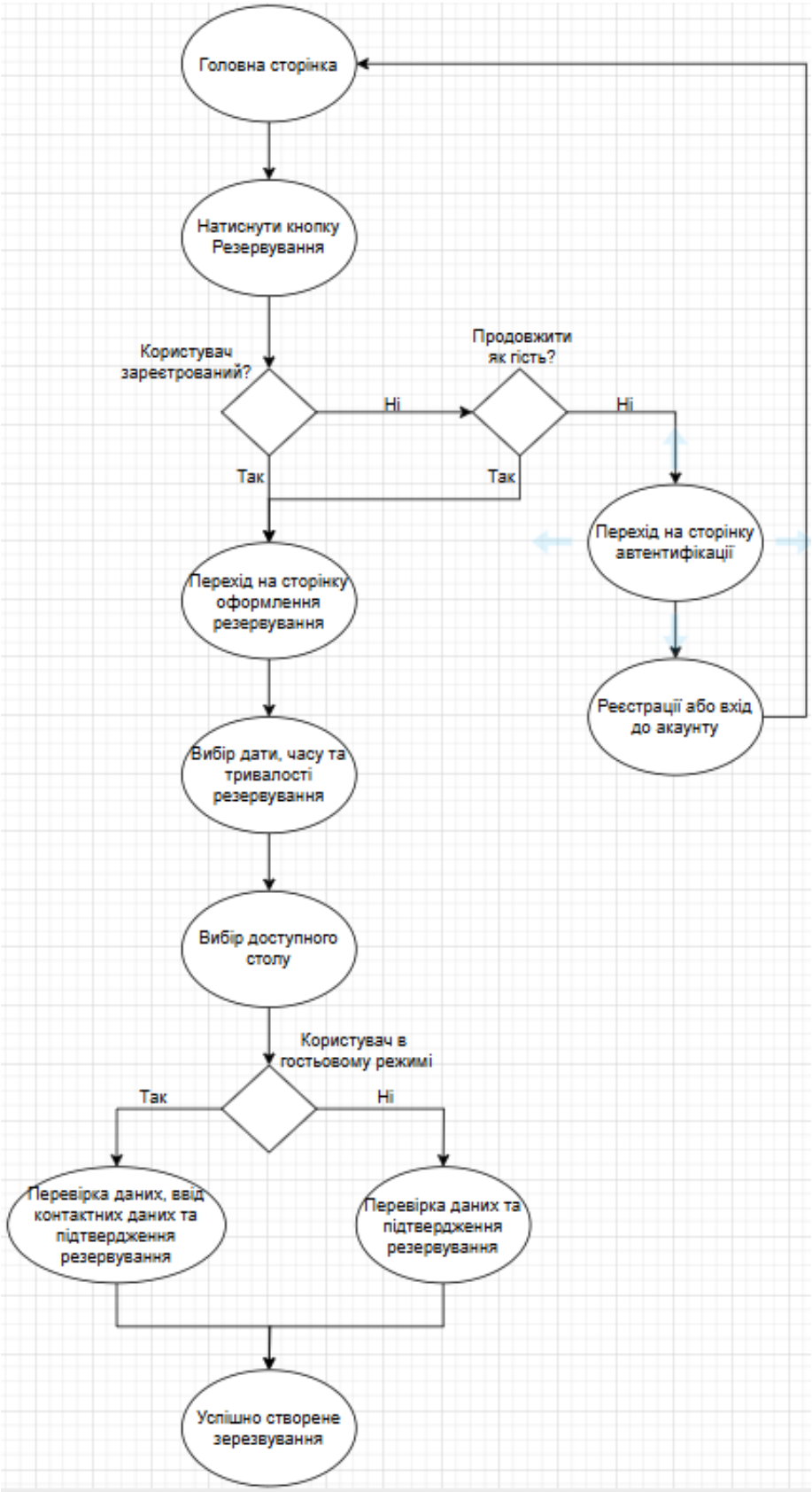


Рисунок 3.10 – Діаграма активності

На рисунку 3.11 зображено головну сторінку сайту, на цій сторінці є назва ресторану, клікабельна іконка у вигляді фігури чоловічка, кнопка

«Резервування», яка відповідає за навігацію та при натисканні переправляє користувача на сторінку оформлення резервування, інша кнопка «Показати на карті» відповідає за перенаправлення користувача на сторінку сервісу Goggle maps, де знаходить ресторан. Також на сторінці зображено два блоки з інформацією, а саме Графік роботи та блок з інформацією про ресторан.

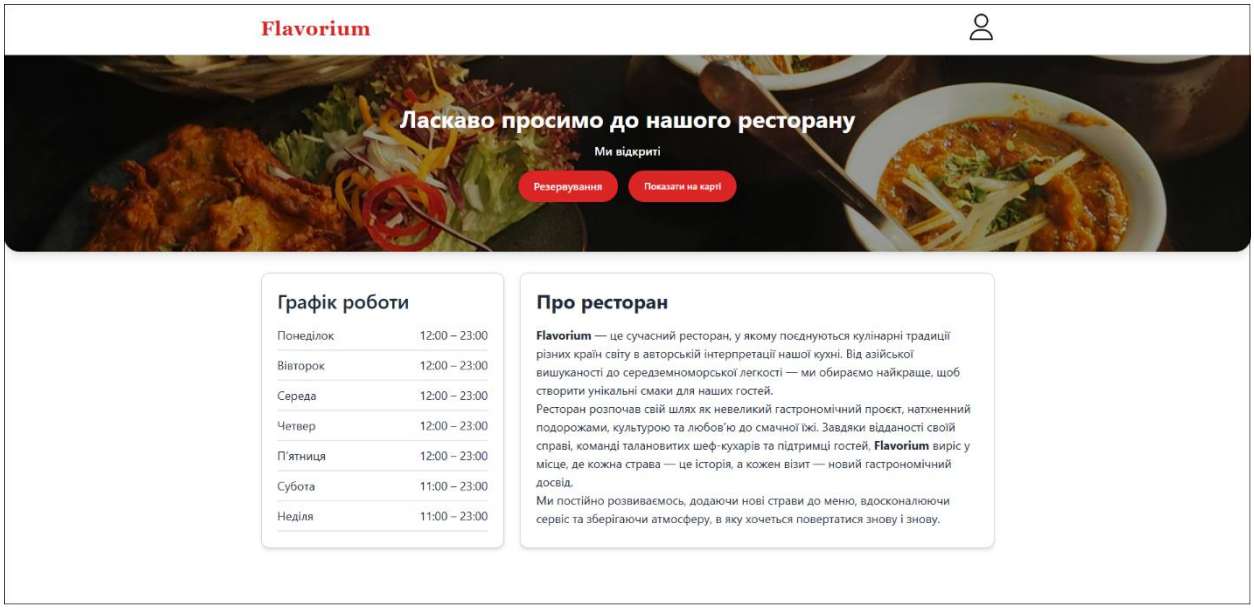


Рисунок 3.11 – Головна сторінка сайту

Щоб перейти до сторінки резервування, користувачу потрібно натиснути кнопку «Резервування», у випадку, якщо користувач не зареєстрований і не увійшов до свого аккаунту, буде показано модально вікно, яке попереджає користувача то те, що не було виконано вхід до системи. На рисунку 3.12 зображено модальне вікно з попередженням.

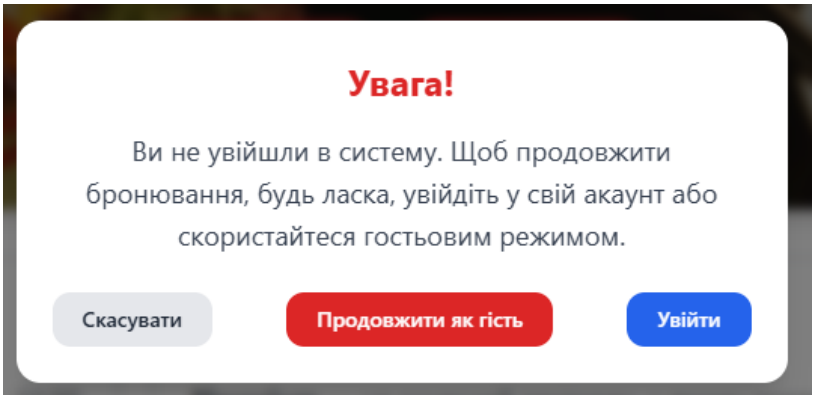


Рисунок 3.12 – Модальне вікно з попередженням

Розглянемо випадок, коли користувач вирішив увійти до системи. Після натискання синьої кнопки «Увійти», користувача перенаправить на сторінку реєстрації (рис. 3.13), тут користувачу потрібно заповнити всі поля, після цього натиснути «Зареєструватися», якщо введені дані коректні, користувача буде зареєстровано, після цього користувач зможе увійти до свого аккаунта та почати оформлювати резервування.

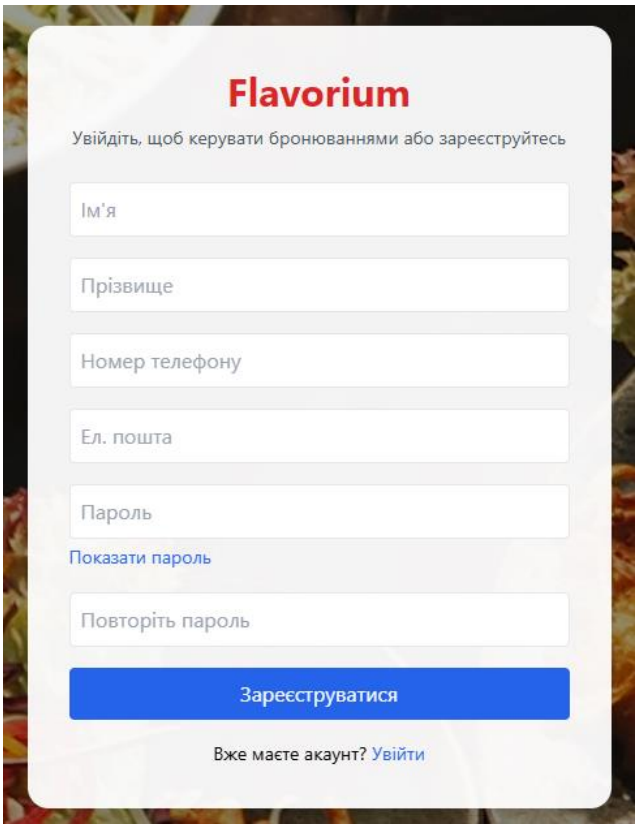


Рисунок 3.13 – Форма реєстрації нового користувача

Після входу до свого акаунту і переходу на сторінку резервування, користувач може приступати до оформлення резервування. На першому кроці користувачу необхідно вибрати дату, час та тривалість резервування (за замовчанням 2 години), після цього стане доступною інтерактивна мапа ресторану, для вибору столу, буде достатньо натиснути на будь-який кружечок, після цього з’явиться блок з інформацією про резервування, користувач може змінити кожен з параметрів на будь-якому кроці. Після перевірки всіх параметрів, користувачу залишається натиснути кнопку «Підтвердити».

На рисунку 3.14 зображено сторінку резервувань, на цій сторінці, користувач може перегляди свої актуальні та минулі резервування, а також скасовувати свої резервування.

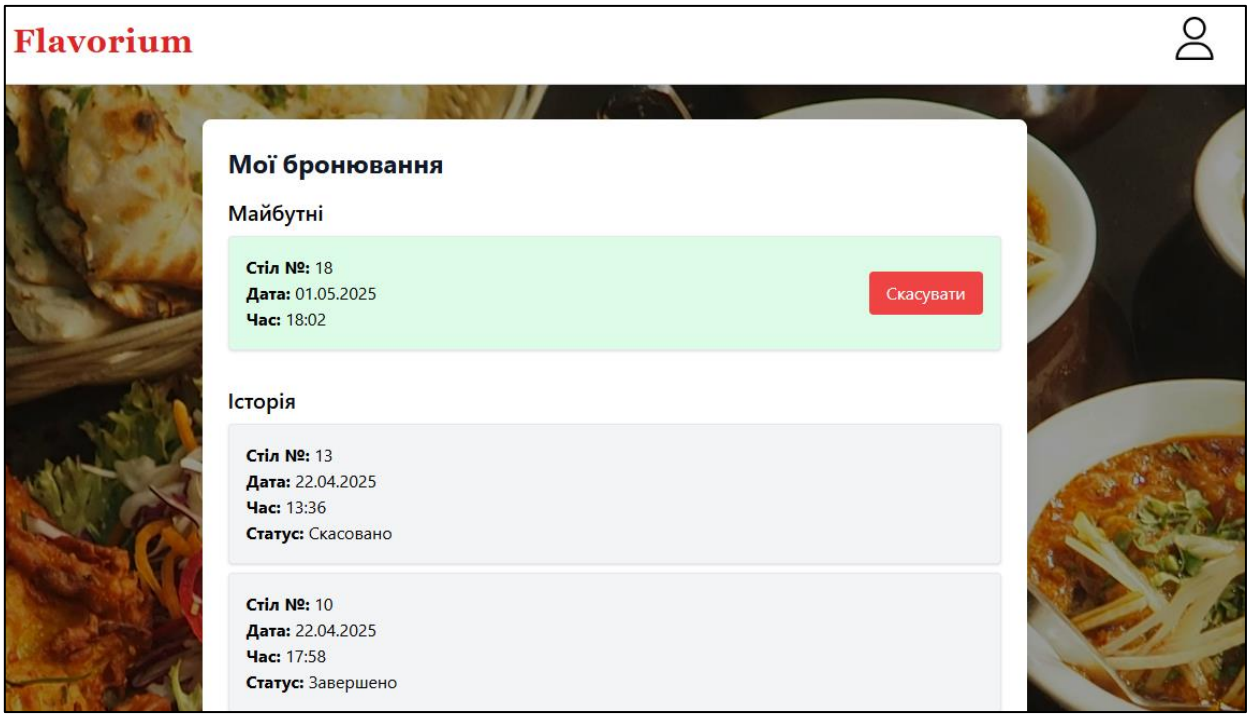


Рисунок 3.14 – Сторінка історії резервувань

Якщо дата та час були вибрані правильно, тоді користувача перенаправить до сторінки з історією резервувань.

3.5 Висновки до розділу

У третьому розділі було розглянуто реалізацію інтерфейсу користувача вебзастосунку для онлайн-резервування столів у ресторані. Було реалізовано покроковий процес резервування, який охоплює вибір дати, часу та тривалості, вибір столу на інтерактивній карті, а також підтвердження резервування з урахуванням статусу користувача (зареєстрований або гість). Інтерфейс забезпечує адаптивність, зручну навігацію між етапами, валідацію введених даних та обробку можливих помилок.

Система враховує конфлікти резервувань у базі даних, дозволяє відображати доступність столів у реальному часі та підтримує гостьовий режим із обов’язковим введенням контактної інформації. Розділення логіки за компонентами сприяє підтримуваності та масштабованості застосунку. Реалізація інтерфейсу відповідає сучасним вимогам до UX/UI, забезпечує інтуїтивність та простоту використання для кінцевого користувача, а також створює основу для подальшого розвитку системи.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто процес розробки вебзастосунку для резервування столів у ресторані з використанням сучасних вебтехнологій. У ході виконання проєкту були досягнуті такі результати:

- проведено аналіз сучасних засобів веброзробки та обґрунтовано вибір технологій React, Firebase та Tailwind CSS;
- сформульовано вимоги до функціональності системи з урахуванням потреб різних категорій користувачів (зареєстровані, гостьові);
- спроєктовано архітектуру вебзастосунку з урахуванням розмежування ролей, інтерактивної карти розміщення столів та перевірки коректності даних;
- реалізовано клієнтський інтерфейс із використанням компонентної архітектури React і стилізовано за допомогою Tailwind CSS;
- впроваджено інтеграцію з Firebase Firestore для зберігання інформації про користувачів, резервування та статус столів;
- здійснено валідацію введених даних, обробку конфліктів резервувань та перевірку робочих годин ресторану;
- протестовано основну функціональність системи, а також здійснено публічне розгортання проєкту для доступності через Інтернет.

У першому розділі було проаналізовано сучасні технології веброзробки, зокрема React, Vue та Angular, наведено їх порівняння та обґрунтовано вибір React як основної бібліотеки для реалізації SPA.

У другому розділі описано архітектуру застосунку, реалізовані компоненти, логіку взаємодії з користувачем, а також інтерфейсні рішення для різних етапів резервування. Окрему увагу приділено реалізації інтерактивної карти та адаптивному дизайну.

У третьому розділі наведено програмну реалізацію функціоналу резервування столів: вибір дати та часу, відображення статусу столів,

підтвердження резервування з перевірками, збереження інформації у Firestore та підтримкою гостьового режиму.

Загалом, поставлену мету роботи було досягнуто повністю. Розроблений вебзастосунок є прикладом ефективного поєднання гнучкої архітектури, адаптивного дизайну та інтеграції з хмарними технологіями. Отримані результати можуть бути використані в реальних умовах ресторанного бізнесу або слугувати основою для подальшого розвитку (мобільна версія, система оплат, аналітика тощо).

ПЕРЕЛІК ПОСИЛАНЬ

1. Awesome JavaScript Libraries and Frameworks You Should Know About. URL: https://code.tutsplus.com/essential-javascript-libraries-and-frameworks-you-should-know-about--cms-29540a (дата звернення: 15.01.2025).

2. JavaScript – MDN Web Docs – Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 02.02.2025).

3. User Interface Layout: 5 Essential Tips from Appmaster.io. URL: https://appmaster.io/blog/user-interface-layout-5-essential-tips-from-appmasterio (дата звернення: 22.02.2025).

4. Writing Markup with JSX. URL: https://react.dev/learn/writing-markup-with-jsx (дата звернення: 23.02.2025).

5. Офіційний сайт з документацією Angular. URL: https://angular.io/docs?category=community (дата звернення: 06.03.2025).

6. Офіційний сайт з документацією Node Js. URL: https://nodejs.org/en/about (дата звернення: 23.03.2025).

7. Офіційний сайт з документацією React Js. URL: https://react.dev/blog/2023/03/16/introducing-react-dev (дата звернення: 25.03.2025).

8. Офіційний сайт з документацією Tailwind. URL: https://tailwindcss.com/docs/installation/using-vite (дата звернення: 13.04.2025).

9. Офіційний сайт з документацією Vue Js. URL: https://vuejs.org/guide/introduction.html (дата звернення: 22.03.2025).

10. Офіційний сайт інсталятора Create React App. URL: https://create-react-app.dev/docs/getting-started/ (дата звернення: 25.03.2025).

ДОДАТОК А

Програмна реалізація компоненту Tablemap

```
import { useEffect, useState } from "react";
import { collection, getDocs, query, where } from "firebase/firestore";
import { db } from "../../firebase";
```

Рисунок А.1 – Імпорти та конфігурація TableMap

```
const tablePositions = {
  T1: { top: "92%", left: "20%", type: 4 },
  T2: { top: "80%", left: "21%", type: 2 },
  T3: { top: "68%", left: "15%", type: 4 },
  T4: { top: "80%", left: "40%", type: 2 },
  T5: { top: "90%", left: "50%", type: 2 },
  T6: { top: "80%", left: "60%", type: 2 },
  T7: { top: "67%", left: "44%", type: 6 },
  T8: { top: "54%", left: "23%", type: 2 },
  T9: { top: "54%", left: "40%", type: 2 },
  T10: { top: "43%", left: "23%", type: 4 },
  T11: { top: "43%", left: "40%", type: 2 },
  T12: { top: "32%", left: "40%", type: 2 },
  T13: { top: "32%", left: "23%", type: 2 },
  T14: { top: "21%", left: "30%", type: 2 },
  T15: { top: "21%", left: "13%", type: 2 },
  T16: { top: "10%", left: "13%", type: 2 },
  T17: { top: "10%", left: "30%", type: 2 },
  T18: { top: "12%", left: "55%", type: 6 },
  T19: { top: "10%", left: "75%", type: 2 },
  T20: { top: "10%", left: "90%", type: 2 },
  T21: { top: "21%", left: "75%", type: 2 },
  T22: { top: "21%", left: "90%", type: 2 },
  T23: { top: "32%", left: "75%", type: 4 },
  T24: { top: "43%", left: "75%", type: 2 },
  T25: { top: "54%", left: "75%", type: 2 },
  T26: { top: "65%", left: "75%", type: 2 },
};
```

Рисунок А.2 – Конфігурація розташування столів

```
const TableMap = ({ selectedDate, selectedTime, duration, onTableSelect }) => {
  const [tables, setTables] = useState([]);

  useEffect(() => {
    const fetchTables = async () => {
      const q = query(collection(db, "tables"));
      const snapshot = await getDocs(q);
      const data = snapshot.docs.map((doc) => ({
        id: doc.id,
        ...doc.data(),
      }));
      setTables(data);
    };
    fetchTables();
  }, []);
```

Рисунок А.3 – Логіка отримання даних

```
return (
    <div className="relative w-full aspect-[3/4] bg-white border rounded-md shadow-lg">
      {furnitureElements.map((el) => (
        <div
          key={el.id}
          className="absolute bg-gray-100 text-gray-800 text-xs font-medium border rounded-full px-2 py-0.5 shadow-sm"
          style={{
            top: el.top,
            left: el.left,
            transform: "translate(-50%, -50%)",
            whiteSpace: "nowrap",
          }}
        >
          {getFurnitureLabel(el.label)}
        </div>
      ))}

      {Object.entries(tablePositions).map(([id, pos]) => {
        const table = tables[id];
        const isSelected = selectedTableId === id;
        const size = getSize(table);

        return (
          <button
            key={id}
            onClick={() => table?.status !== "reserved" && onTableSelect(id)}
            title={`Стіл №${id.replace("T", "")}, ${table?.type} місць${
              table?.status === "reserved" ? " (заброньовано)" : ""
            }`}
            className={`absolute rounded-full text-sm font-bold flex items-center justify-center border
              ${getColor(table)}
              ${isSelected ? "ring-4 ring-green-500" : ""}
              ${
                table?.status !== "reserved"
                  ? "hover:ring hover:ring-blue-400"
                  : ""
              }
              transition-transform duration-200 ease-in-out
            `}
            style={{
              top: pos.top,
              left: pos.left,
              width: `${size}px`,
              height: `${size}px`,
              transform: "translate(-50%, -50%)",
              cursor: table?.status === "reserved" ? "not-allowed" : "pointer",
            }}
          >
            {id.replace("T", "")}
          </button>
        );
      })}
    </div>
```

Рисунок А.4 – Відображення карти столів