ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ АВТОМАТИЗАЦІЇ РЕСТОРАННОГО БІЗНЕСУ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми «Інформаційні системи та технології»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Федір Богачкін
(підпис)

Виконав: здобувач(ка) вищої освіти групи ІСД-41

Федір БОГАЧКІН

Керівник: Іван ШАХМАТОВ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Бакалавр

Спеціальність 126 Інформаційні системи та технології

Освітньо-професійна програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри Інформаційних систем та технологій
_____ Каміла СТОРЧАК
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Богачкіна Федора Олексійовича

1. Тема кваліфікаційної роботи: «Інформаційна система для автоматизації ресторанного бізнесу»
керівник кваліфікаційної роботи викладач кафедри ІСТ Іван ШАХМАТОВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2025 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи організації процесу бронювання місць у закладах громадського харчування, опис методів структуризації інформації відповідно до логіки замовлення столика (вибір дати, часу, страв, місця в залі), технічна документація з описом бібліотек і фреймворків, використаних для реалізації web-застосунку з функціоналом інтерактивного вибору місця в ресторані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій сворення веб-застосунків.
2. Проектування застосунку для спрощення процесу оформлення замовлень та столиків у ресторані.
3.Програмна реалізація та опис функціонування застосунку для процесу оформлення замовлень та столиків у ресторані.
4.Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Титульний слайд.
2. Мета, об’єкт та предмет дослідження.Аналіз аналогів.
3. Технічні завдання
4. Функціональні вимоги.
5. Нефункціональні вимоги.
6. Аналіз аналогів.
7. Програмні засоби реалізації.
8. Діаграма послідовності.
9. Схема роботи Web-застосунку.
10. Інтерфейс розробленого застосунку.
11. Вигляд бази даних.
12. Висновки.
13. Апробація.
14. Дякую за увагу!

6. Дата видачі завдання «28» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02 - 07.03.2025	Виконано
2	Аналіз та дослідження існуючих аналогів	08.03 - 14.03.2025	Виконано
3	Огляд існуючих технологій резервування столиків	15.03 – 17.04.2025	Виконано
4	Проектування web-застосунку для процесу оформлення замовлень та місць	18.04 – 19.04.2025	Виконано
5	Програмна реалізація web-застосунку для оформлення замовлень	20.04 – 26.04.2025	Виконано
6	Тестування застосунку	27.04 – 28.04.2025	Виконано
7	Оформлення роботи: вступ, висновки, реферат	29.04 - 05.05.2025	Виконано
8	Розробка демонстраційних матеріалів	06.05 - 12.05.2025	Виконано

9	Попередній захист роботи	13.05 – 31.05.2025	Виконано

Здобувач(ка) вищої освіти _____ Федір Богачкін
(підпис)

Керівник
кваліфікаційної роботи _____ Іван ШАХМАТОВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., табл., 16 рис., 14 джерел.

Мета роботи — покращення процесу інтерактивного бронювання столиків у закладах громадського харчування шляхом створення web-застосунку з можливістю попереднього замовлення страв.

Об'єкт дослідження — процес автоматизації управління замовленнями та резервуванням місць у закладах ресторанного типу.

Предмет дослідження — веб-застосунок для бронювання столиків та оформлення замовлень.

Короткий зміст роботи: У даній роботі розглянуто процес розробки веб-додатку для системи з підтримки бронювання столиків та оформлення замовлень у ресторані. Основна мета — автоматизація обслуговування клієнтів та покращення внутрішньої організації роботи закладу. Описано аналіз предметної області, огляд існуючих рішень, обґрунтування вибору технологій та реалізацію проєкту з використанням Node.js (серверна частина), React (клієнтська частина) і MongoDB (база даних). Робота містить опис архітектури системи, її основних функціональних можливостей та переваг впровадження в ресторанний бізнес.

ЗМІСТ

ВСТУП

Впровадження інформаційних технологій у сферу обслуговування стає важливим чинником підвищення якості надання послуг, оптимізації бізнес-процесів та зростання конкурентоспроможності підприємств. Ресторанний бізнес не є винятком: автоматизація процесів бронювання, обробки замовлень і взаємодії з клієнтами дозволяє закладам громадського харчування ефективніше організовувати роботу, скорочувати витрати часу, зменшувати кількість помилок персоналу та підвищувати рівень задоволеності відвідувачів.

Актуальність теми дипломної роботи зумовлена потребою у створенні інтерактивних вебсистем, які забезпечують зручне онлайн-бронювання столиків, попереднє замовлення страв, сповіщення клієнтів і аналітику роботи закладу. З огляду на зростаючі вимоги споживачів до швидкості й зручності обслуговування, розвиток таких систем є важливим напрямом у галузі прикладного програмування та веброзробки.

Метою дипломної роботи є розробка web-застосунку, що реалізує функціональність системи з підтримки бронювання столиків та оформлення замовлень у ресторані з використанням сучасних інструментів розробки. Запропоноване рішення має забезпечити зручний інтерфейс для клієнтів і адміністрації, підтримку взаємодії з базою даних, модульність, масштабованість і відповідність вимогам до безпеки та адаптивності.

Об'єктом дослідження є процес автоматизації обслуговування в закладах ресторанного типу. Предметом дослідження — методи та засоби створення веборієнтованих інформаційних систем із функціональністю управління бронюваннями та замовленнями.

Розроблена система буде реалізована з використанням сучасного стеку вебтехнологій: React.js для клієнтської частини, Node.js з Express для серверної логіки,

MongoDB для зберігання даних. Такий підхід забезпечить ефективну роботу, швидкий доступ до інформації та зручну підтримку подальшого розширення функціоналу.

У процесі виконання дипломної роботи передбачається проведення аналізу предметної області, формалізація вимог, проєктування архітектури системи, розробка та тестування функціональних модулів, а також оцінка ефективності та перспектив подальшого впровадження розробленого рішення.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Технології та веб-додатки суттєво змінюють підхід до ведення ресторанного бізнесу, відкриваючи нові можливості як для закладів, так і для клієнтів. Їх вплив охоплює практично всі аспекти діяльності ресторану — від процесу обслуговування гостей до управління внутрішніми процесами, маркетингу та аналітики.

Перш за все, веб-додатки забезпечують зручність і доступність для клієнтів. Завдяки інтерактивним інтерфейсам користувачі можуть самостійно бронювати столики, переглядати меню, здійснювати ппереднє замовлення або оплачувати страви онлайн. Це зменшує навантаження на персонал, скорочує час обслуговування та створює позитивний користувацький досвід, що сприяє лояльності клієнтів.

Для ресторану веб-додаток стає інструментом автоматизації бізнес-процесів. Наприклад, система бронювання дозволяє уникнути конфліктів щодо вільних столиків, мінімізує людські помилки та забезпечує ефективніше управління розсадженням гостей. Крім того, система замовлень дає змогу кухні швидко отримувати точну інформацію про побажання клієнтів, що зменшує ймовірність помилок і підвищує якість обслуговування.

Інтеграція з базами даних та аналітичними модулями дозволяє адміністрації збирати статистику: які страви найчастіше замовляють, у які дні найбільше відвідувачів, який середній чек тощо. Це дає змогу приймати обґрунтовані управлінські рішення, оптимізувати меню, формувати акційні пропозиції та краще планувати навантаження персоналу.

Також веб-додатки є важливим каналом маркетингу та взаємодії з клієнтами. За допомогою push-сповіщень, електронних листів, інтеграції з соціальними мережами та системами лояльності ресторан може підтримувати зв’язок із постійними клієнтами, надсилати персоналізовані пропозиції та стимулювати повторні візити.

Не менш важливим є й вплив на репутацію: веб-додатки часто включають модулі з відгуками та рейтингами. Вчасне реагування на відгуки дозволяє зміцнювати довіру до закладу та демонструвати турботу про клієнтів.

1.1 Аналіз та характеристика web-застосунків

Веб-додаток — це програмне забезпечення, яке функціонує у веб-браузері та надає користувачам можливість взаємодіяти з різноманітними сервісами через Інтернет або локальну мережу. Основна особливість веб-додатків полягає в тому, що для їх використання не потрібно встановлювати окрему програму на пристрій — доступ здійснюється безпосередньо через браузер.

Типова архітектура веб-додатку передбачає наявність клієнтської частини (інтерфейсу), що створена за допомогою таких технологій, як HTML, CSS і JavaScript, та серверної частини, яка відповідає за обробку запитів, логіку роботи додатку і взаємодію з базою даних. Серверна частина може бути реалізована на різних платформах, зокрема на Node.js, Python, PHP або інших технологіях. Для збереження даних застосовуються бази даних, наприклад MongoDB або MySQL.

Веб-додатки є динамічними системами, які дозволяють користувачам виконувати дії: реєструватися, авторизуватися, створювати записи, переглядати інформацію, здійснювати замовлення тощо. Вони кросплатформенні, тобто можуть працювати на різних пристроях — комп’ютерах, планшетах або смартфонах — за умови наявності доступу до мережі.

Серед прикладів веб-додатків — сервіси електронної пошти (наприклад, Gmail), онлайн-редактори документів (Google Docs), системи управління завданнями (Trello, Notion), інтернет-магазини, а також системи бронювання столиків, готелів чи квитків.

Головними перевагами веб-додатків є їхня доступність, простота використання, відсутність потреби в інсталяції, автоматичне оновлення та легкість

підтримки. Водночас вони мають і певні недоліки, зокрема залежність від стабільного Інтернет-з’єднання та необхідність забезпечення належного рівня безпеки даних.

Отже, веб-додатки є ефективним засобом реалізації сучасних інтерактивних сервісів, що відповідають потребам як бізнесу, так і кінцевих користувачів, забезпечуючи широкий функціонал у зручному онлайн-форматі.

Робота веб-додатків базується на принципі клієнт-серверної взаємодії через мережу, найчастіше — через Інтернет. Це означає, що коли користувач відкриває веб-додаток у браузері, він взаємодіє з клієнтською частиною (інтерфейсом), яка надсилає запити до серверної частини — де й відбувається основна логіка обробки даних.

Коли користувач відкриває веб-додаток у браузері (наприклад, сторінку ресторану з можливістю бронювання столика), браузер надсилає запит до веб-сервера. У відповідь сервер передає клієнтську частину веб-додатку — HTML-сторінку з підключеними CSS- і JavaScript-файлами. Завдяки JavaScript-фреймворкам (наприклад, React), інтерфейс завантажується динамічно, реагує на дії користувача (наприклад, натискання кнопки "Забронювати") та взаємодіє з сервером без перезавантаження сторінки.

Коли користувач виконує певну дію — наприклад, вибирає дату і час бронювання — JavaScript на клієнтській стороні формує HTTP-запит (зазвичай через AJAX або Fetch API) і відправляє його до серверної частини веб-додатку. Сервер, реалізований на платформі Node.js, приймає цей запит, обробляє його (перевіряє наявність вільних столиків, перевіряє дані користувача), звертається до бази даних (наприклад, MongoDB), зчитує або змінює дані, а потім формує відповідь і надсилає її назад клієнту.

Клієнтська частина отримує відповідь від сервера (наприклад, повідомлення про успішне бронювання) та динамічно оновлює інтерфейс — відображає підтвердження або повідомлення про помилку. Усе це відбувається в режимі реального часу або близько до нього, без потреби оновлювати сторінку повністю.

Таким чином, веб-додаток забезпечує безперервну взаємодію між користувачем і сервером, де кожна дія обробляється на сервері, а результат миттєво відображається в браузері користувача. Усе це робить веб-додатки потужним інструментом для створення інтерфейсів, які дозволяють не лише переглядати інформацію, а й активно з нею працювати — наприклад, здійснювати замовлення, керувати бронюванням або спілкуватися онлайн.

1.2 Огляд та аналіз існуючих систем

У сучасних умовах цифрової трансформації галузі громадського харчування все більшої популярності набувають програмні системи, що автоматизують процеси обслуговування клієнтів, зокрема попереднє бронювання столиків та оформлення замовлень. Такі системи є важливим інструментом оптимізації бізнес-процесів, підвищення рівня обслуговування клієнтів, зниження навантаження на персонал та забезпечення конкурентоспроможності ресторанного закладу.

На ринку представлено значну кількість програмних рішень, які реалізують функціональність бронювання та прийому замовлень. Найбільш відомими серед них є OpenTable, Resy, Tock, та інші. Розглянемо ключові характеристики та особливості найбільш поширених рішень:

1.2.1 OpenTable

OpenTable — це одна з найвідоміших і найпоширеніших систем онлайн-бронювання столиків у ресторанах(Рис.1.1), заснована у 1998 році та на сьогодні належить компанії Booking Holdings. Сервіс охоплює понад 50 000 ресторанів у більш ніж 80 країнах світу та обслуговує мільйони користувачів, забезпечуючи ефективну взаємодію між клієнтами та закладами громадського харчування.

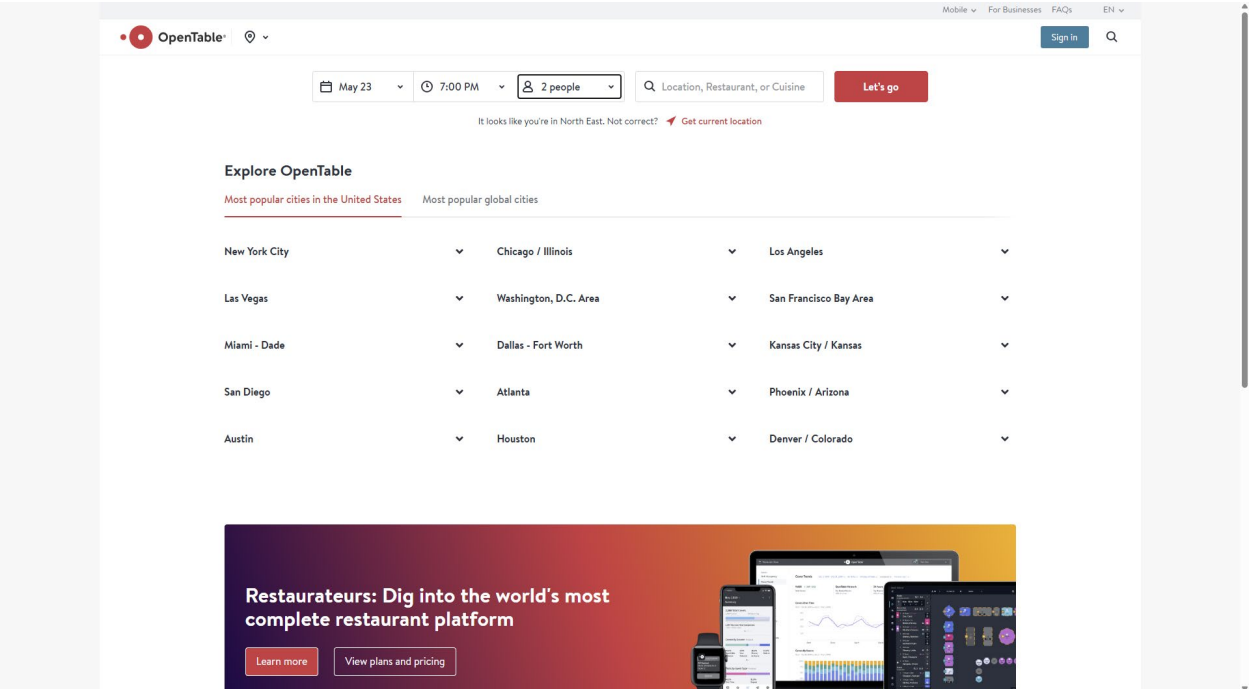


Рис.1.1 OpenTable

Основні функціональні можливості

Система OpenTable має широкий функціонал, який включає:

- Онлайн-бронювання столиків через веб-портал або мобільний додаток у режимі реального часу.
- Управління посадкою гостей (table management) – інтерактивна карта залу, що дозволяє контролювати заповненість залу, черги та оптимізувати розсадку клієнтів.
- Список очікування та SMS-сповіщення для інформування клієнтів про статус бронювання.
- Інтеграція з POS-системами для синхронізації бронювань та замовлень, збору аналітичних даних.
- Клієнтська база (CRM) – збереження історії відвідувань, уподобань, особливих запитів клієнтів.
- Автоматизована аналітика та звітність для оцінки завантаженості, популярності часу візитів, середнього чеку тощо.

- Промоційні інструменти – просування ресторану через платформу OpenTable, участь у кампаніях та спеціальних подіях.

Технічна реалізація

OpenTable реалізовано як хмарне рішення за моделлю SaaS (Software as a Service), що забезпечує високу доступність, безпеку та масштабованість. Користувачі взаємодіють із системою через веб-інтерфейс або мобільні додатки (iOS, Android). Власники ресторанів отримують окрему адміністративну панель для керування бронюваннями, зміни конфігурацій, перегляду статистики.

Система підтримує REST API, що дозволяє інтегрувати її з іншими зовнішніми сервісами (CRM, POS, маркетингові платформи). Для обробки даних використовується сучасна серверна інфраструктура з розподіленою архітектурою, що дозволяє забезпечити швидку обробку запитів та стабільність під навантаженням.

Переваги

- Високий рівень автоматизації процесів обслуговування.
- Зручність використання для клієнтів (бронювання 24/7, без дзвінків).
- Поглиблена аналітика для прийняття бізнес-рішень.
- Підвищення лояльності клієнтів за рахунок збереження їх уподобань.
- Широке охоплення ринку та репутаційна підтримка ресторану.

Недоліки

- Висока вартість обслуговування, що може бути критичною для малого бізнесу.
- Залежність від стороннього сервісу — у разі збою або зміни політики використання можуть виникнути ризики.
- Обмежена гнучкість у зміні логіки бізнес-процесів для унікальних потреб окремих закладів.

OpenTable є прикладом успішної реалізації багатофункціональної системи онлайн-бронювання, яка поєднує зручність для клієнтів та потужні інструменти для бізнесу. Вона виступає референтною моделлю для розробки власних систем бронювання, надаючи орієнтири щодо архітектури, функціоналу, UX-рішень та

технічної реалізації. Водночас, потреба в адаптації до локального ринку, бюджетні обмеження та бажання мати більший контроль над системою можуть стати підставою для розробки індивідуального рішення.

1.2.2 Resy

Resy — це сучасна платформа для онлайн-бронювання столиків у ресторанах(Рис 1.2), заснована у 2014 році та згодом придбана компанією American Express. Resy швидко здобула популярність завдяки інноваційному підходу до керування гостьовим потоком, гнучкій аналітиці та фокусі на висококласні ресторани. Система активно використовується в США, Канаді, Великобританії, Австралії та деяких країнах Європи.

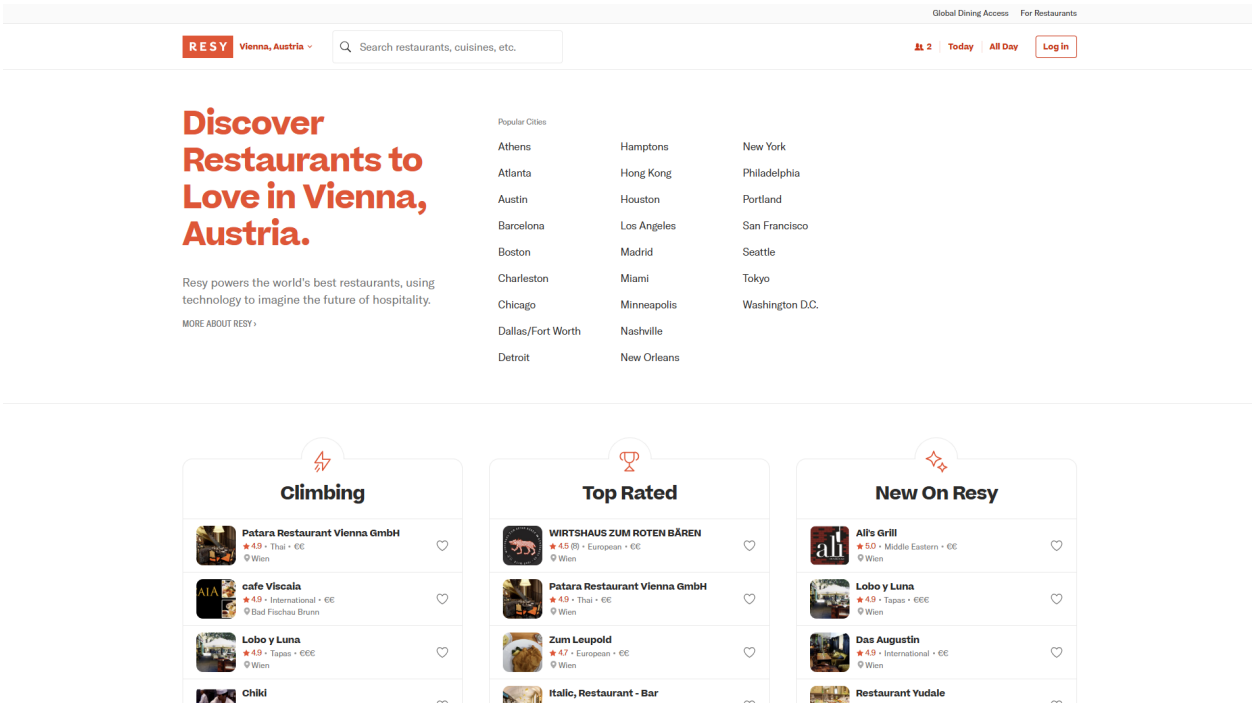


Рис 1.2 Resy

Основні функціональні можливості

Resy пропонує розвинений набір інструментів для автоматизації процесу бронювання та покращення взаємодії між закладом і відвідувачами:

- Онлайн-бронювання в реальному часі через сайт, мобільний додаток та соціальні мережі.
- Управління посадкою (table management) – гнучка сітка розміщення, з урахуванням часу перебування, типу бронювання та пріоритету гостей.
- Система контролю черги та списку очікування з можливістю сповіщень через SMS або мобільний додаток.
- Інструменти CRM – збереження історії відвідувань, звичок клієнтів, можливість налаштовувати персоналізовані сервіси.
- Інтеграція з POS-системами для отримання аналітики по замовленням та середньому чеку.
- Функціонал динамічного ціноутворення – можливість встановлення різних умов бронювання залежно від часу, дня тижня або завантаженості.
- Аналітичні інструменти – звіти про заповнюваність залу, ефективність персоналу, поведінкові моделі клієнтів.

Технічна реалізація

Resy є хмарною платформою, що працює за моделлю SaaS і доступна на всіх типах пристроїв (ПК, планшети, смартфони). Інтерфейс системи орієнтований на зручність як для адміністративного персоналу, так і для гостей. Використання REST API дозволяє інтегрувати Resy з іншими бізнес-інструментами ресторанів.

Система побудована з урахуванням високих вимог до продуктивності та безпеки. Для зберігання персональних даних клієнтів та платіжної інформації використовуються стандарти, сумісні з PCI DSS. Застосовуються технології аналітики великих даних для прогнозування попиту та оптимізації бронювання.

Переваги

- Глибока аналітика та інструменти персоналізації.

- Інноваційні підходи до керування трафіком гостей і часу посадки.
- Інтеграція з American Express забезпечує промоційні можливості для ресторанів.
- Зручний мобільний додаток для користувачів і менеджерів.
- Гнучкість у налаштуванні правил бронювання.

Недоліки

- Обмежена доступність за межами великих міст та розвинених країн.
- Орієнтація на середній і високий ціновий сегмент — може бути надмірною для невеликих або демократичних ресторанів.
- Платна модель з фіксованими та змінними витратами, яка може не відповідати бюджету малого бізнесу.

Resy — це приклад високотехнологічної платформи бронювання столиків, що поєднує інтуїтивний інтерфейс, потужну аналітику та елементи штучного інтелекту для підвищення ефективності ресторанного обслуговування. Її модель є орієнтиром для створення сучасних систем, які прагнуть не лише автоматизувати процес бронювання, а й покращити стратегічне управління клієнтським досвідом. У контексті розробки індивідуальної системи для ресторану досвід Resy може бути корисним при проектуванні модулів аналітики, управління чергою та персоналізованого сервісу.

1.2.3 Tock

Tock — це інноваційна платформа для управління бронюваннями та замовленнями у закладах громадського харчування(Рис. 1.3), що поєднує функціонал класичних систем онлайн-бронювання з розширеними можливостями комерціалізації послуг ресторану. Сервіс був заснований у 2014 році і згодом придбаний компанією Squarespace у 2021 році. Tock активно використовується не лише у ресторанах, а й у виноробнях, пекарнях, гастробарах і під час подій з фіксованим меню.

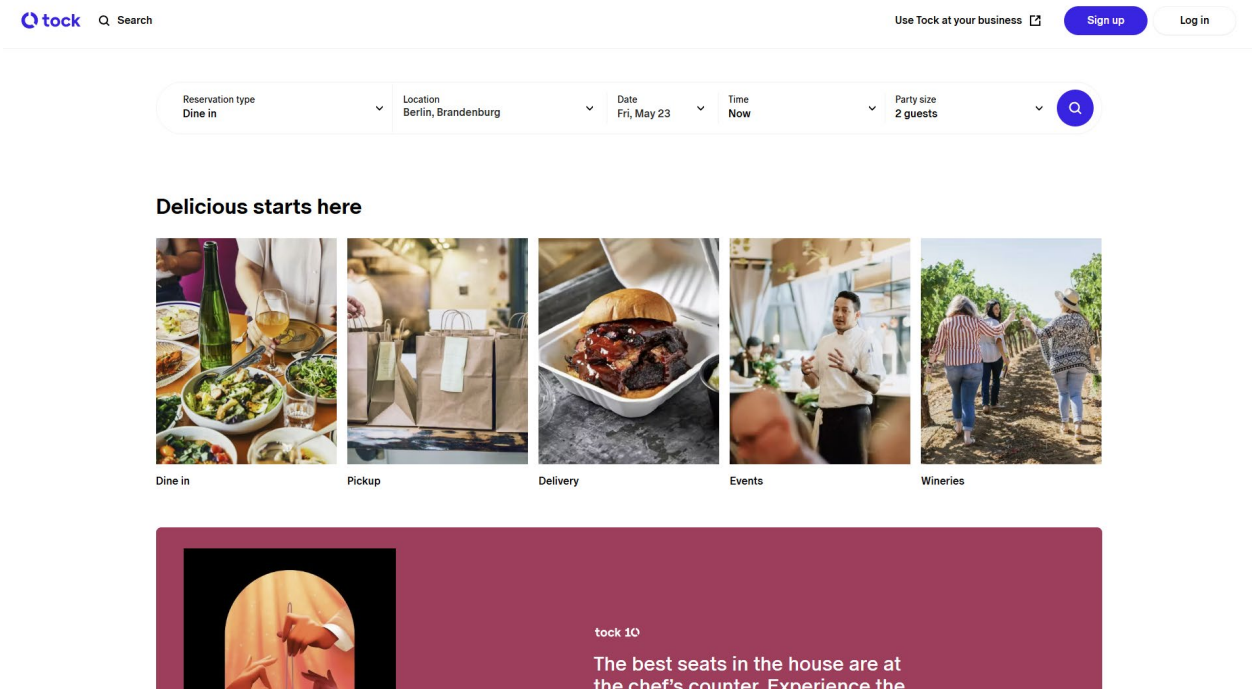


Рис. 1.3 Tock

Основні функціональні можливості

Система Tock розроблена як гнучкий інструмент для автоматизації всіх етапів обслуговування клієнтів — від бронювання до передоплати та організації подій:

- Онлайн-бронювання з можливістю вибору часу, кількості гостей, формату відвідування (стандартний стіл, дегустація, подія тощо).
- Попередня оплата або передоплата — користувачі можуть оплачувати частково або повністю під час бронювання, що зменшує кількість неявок.
- Бронювання подій з фіксованим меню або обмеженою кількістю місць.
- Замовлення їжі онлайн з функціоналом вибору часу отримання/доставки.
- Гнучка система керування залом – інтерактивне планування місць із урахуванням тривалості перебування гостей.
- Інструменти CRM – збереження історії візитів, автоматичні підтвердження, електронна пошта, персоналізовані повідомлення.
- Глибока аналітика щодо завантаженості, доходів, середнього чеку, поведінки клієнтів.

Технічна реалізація

Tock функціонує як веб-платформа з доступом через браузер та мобільні додатки. Система має сучасну архітектуру на основі хмарних технологій, забезпечує надійність, масштабованість і високу доступність сервісу. Для безпечної обробки платежів інтегровано платіжні шлюзи з підтримкою PCI DSS. Платформа підтримує REST API, що дозволяє її інтеграцію з POS-системами, CRM та бухгалтерськими програмами.

Особливістю Tock є її акцент на монетизації досвіду – можливість створювати комерційні пакети відвідувань (наприклад, вечеря + дегустація вина), що відкриває нові джерела доходу для закладів.

Переваги

- Високий рівень кастомізації типів бронювання та подій.
- Підтримка передплати або повної оплати наперед — зменшення втрат через неявки.
- Потужні інструменти планування та маркетингу.
- Розширені функції для проведення подій, майстер-класів, турів тощо.
- Інтеграція з Squarespace для створення сайтів ресторанів.

Недоліки

- Вартість використання може бути значною, особливо для невеликих закладів.
- Орієнтація на англомовний ринок — часткова адаптація для країн Східної Європи.
- Необхідність попереднього навчання персоналу для ефективного використання всього функціоналу.

Tock є яскравим прикладом сучасної системи бронювання, яка виходить за межі класичної моделі та інтегрує елементи e-commerce у ресторанний бізнес. Завдяки можливості продавати досвід, а не лише бронювання, Tock дозволяє закладам ефективніше управляти своїми ресурсами та отримувати стабільніший дохід. Ця

система може слугувати орієнтиром у проектуванні інноваційних модулів у власному рішенні, зокрема функцій попередньої оплати, організації подій та інтеграції з маркетинговими інструментами.

З урахуванням вищезазначених факторів та на основі проведеного аналізу було сформовано порівняльну таблицю(Таблю 1.1), яка відображає основні характеристики застосунків-аналогів у співставленні з розроблюваним web-застосунком. Таблиця узагальнює результати оцінки відповідно до поставлених вимог і задач програмного забезпечення.

Таблиця 1.1

Критерій	Tock	Resy	OpenTable	Restaurant Reservation
Цільова аудиторія	Преміум-ресторани, гастрономічні події	Ресторани середнього та преміум сегментів	Масовий ринок, усі категорії ресторанів	Невеликі заклади, власні потреби, локальні клієнти
Інтерактивна схема залу	-	-	(частково для окремих закладів)	+
Можливість попереднього замовлення страв	+	-	-	+

Продовження таблиці 1.1

Критерій	Tock	Resy	OpenTable	Restaurant Reservation
Доступність для користувача	Вимагає створення акаунта	Потрібен акаунт Resy або інтеграція з app	Працює без реєстрації (через гостьовий режим)	Простий інтерфейс, не вимагає акаунта
Панель адміністратора	+	+	+	+
Ціна використання	Платна підписка	Платна підписка	Платна підписка для ресторанів	Безкоштовна
Можливість кастомізації	- (обмежена)	- (лише брендинг)	-	+

1.3 Аналіз та вибір середовища розробки

Інтегроване середовище розробки (IDE, Integrated Development Environment) — це програмне забезпечення, що надає розробнику зручне, централізоване середовище для написання, тестування, налагодження та супроводу програмного коду. До складу типового IDE входить текстовий редактор з підсвічуванням синтаксису, компілятор або інтерпретатор, налагоджувач, а також засоби управління проєктом і інтеграція з системами контролю версій. Використання IDE є доцільним при створенні повноцінних програмних продуктів, оскільки воно забезпечує узгоджене середовище, у якому можна реалізовувати, перевіряти й вдосконалювати код.

Brackets — це легкий, зручний та орієнтований на веброзробку текстовий редактор з відкритим вихідним кодом, створений компанією Adobe Systems. Його головною особливістю є фокус на HTML, CSS та JavaScript, що робить його популярним серед фронтенд-розробників і початківців у вебпрограмуванні.

1.3.1 Brackets

Brackets спроектовано як редактор із сучасним інтерфейсом, який підтримує інноваційні функції, зокрема "Live Preview" — інтерактивний режим, за якого зміни в коді автоматично відображаються у браузері в реальному часі. Це значно спрощує процес верстки та стилізації сторінок.

Редактор має вбудовану підтримку Emmet, автодоповнення синтаксису, підсвічування коду, а також можливість розширення функціоналу через велику кількість плагінів. Brackets також дозволяє зручно працювати з кількома файлами, має простий менеджер тем та підтримку панелі швидкого доступу до CSS-стилів елементів.

Brackets працює на Windows, macOS і Linux, не потребує складної інсталяції, швидко запускається і не перевантажує систему, що робить його зручним вибором для легких проєктів або навчання.

Однак варто зазначити, що з 2021 року Adobe припинила офіційну підтримку Brackets. Незважаючи на це, редактор усе ще використовується завдяки відкритому коду та підтримці спільноти.

У контексті навчальних або невеликих фронтенд-проєктів, таких як базова верстка інтерфейсу вебзастосунку, Brackets може бути ефективним інструментом, особливо для користувачів, які тільки починають знайомство з веброзробкою.

Інтерфейс Brackets представлений на рисунку 1.4

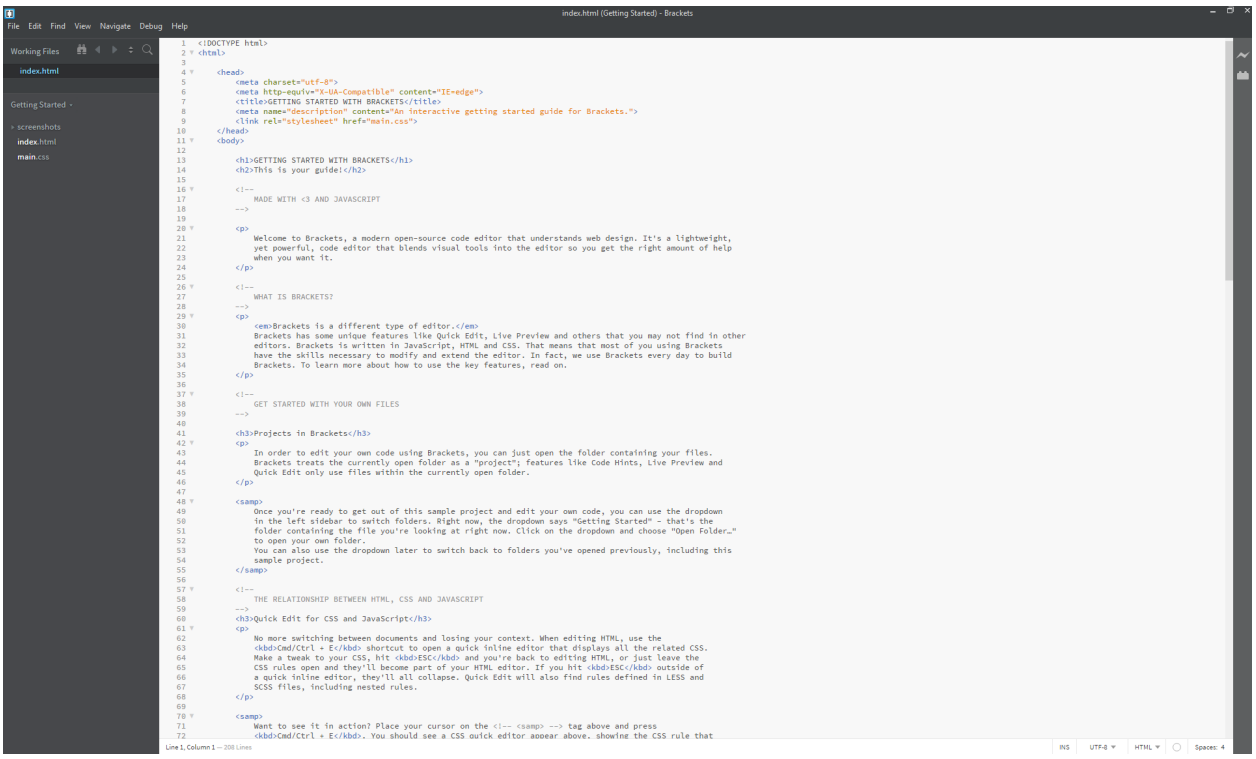


Рис. 1.4 Brackets

Переваги Brackets:

Орієнтація на веброзробку — Brackets спеціально створено для HTML, CSS та JavaScript, що робить його особливо зручним для фронтенду.

Функція Live Preview — дозволяє переглядати зміни в реальному часі у браузері без необхідності перезавантаження сторінки.

Інтуїтивно зрозумілий інтерфейс — простий, легкий для освоєння, особливо для початківців.

Підтримка розширень — редактор має власний менеджер плагінів, що дозволяє розширити функціональність.

Швидкодія та низькі системні вимоги — Brackets запускається швидко і не перевантажує ресурси комп’ютера.

Відкритий вихідний код — проєкт розповсюджується безкоштовно та підтримується спільнотою.

Недоліки Brackets:

Припинення офіційної підтримки — Adobe припинила розвиток редактора у 2021 році, що може обмежити актуальність у майбутньому.

Обмежена підтримка мов програмування — основний фокус на вебтехнологіях; для роботи з іншими мовами потрібні додаткові розширення.

Мала кількість оновлень і нових функцій — редактор поступово застаріває порівняно з Visual Studio Code або WebStorm.

Немає повноцінної інтеграції з терміналом, Git або віддаленим дебагом — ці функції реалізуються лише частково або через сторонні плагіни.

Не підходить для великих або багатокомпонентних проєктів — у таких випадках краще використовувати повноцінне IDE або потужніший редактор.

1.3.2 Atom

Atom — це текстовий редактор з відкритим вихідним кодом, розроблений компанією GitHub і вперше представлений у 2014 році. Його було створено як інструмент для програмістів, орієнтований на максимальну гнучкість, модульність та розширюваність.

Atom побудований на базі вебтехнологій (HTML, CSS, JavaScript) та працює з використанням фреймворку Electron, що дозволяє запускати його як десктопний застосунок на Windows, macOS та Linux.

Редактор підтримує велику кількість мов програмування і має вбудований менеджер пакетів, що дозволяє легко встановлювати плагіни, теми й інші розширення. Однією з важливих функцій Atom є інтеграція з Git та GitHub, а також підтримка спільної роботи над кодом через модуль Teletype. Завдяки сучасному інтерфейсу, підтримці автодоповнення, підсвічуванню синтаксису та можливості налаштовувати середовище під власні потреби, Atom став популярним інструментом серед веброзробників і прихильників відкритого програмного забезпечення.

Однак, починаючи з 2022 року, GitHub офіційно припинив підтримку Atom, що призвело до зменшення активності спільноти та поступового переходу користувачів на інші редактори, зокрема Visual Studio Code. Незважаючи на це, Atom залишається прикладом гнучкого редактора, який значною мірою вплинув на розвиток сучасних інструментів розробки

Інтерфейс Brackets представлений на рисунку 1.5

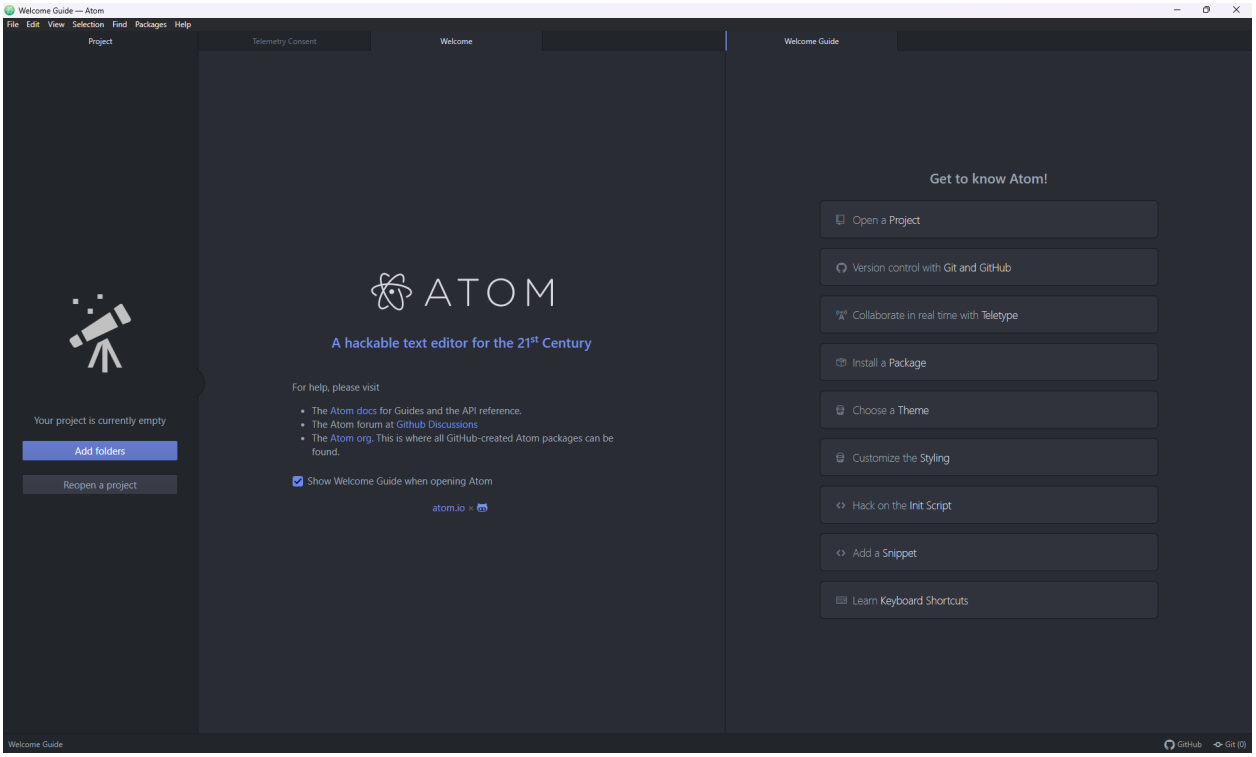


Рис 1.5 Atom

Переваги

Atom є потужним текстовим редактором з відкритим вихідним кодом, який відзначається високою гнучкістю та розширюваністю. Однією з головних його переваг є можливість повного налаштування — користувачі можуть змінювати інтерфейс, створювати власні плагіни, встановлювати тисячі доступних розширень. Завдяки інтеграції з Git та GitHub редактор зручно використовувати для проєктів із системою контролю версій. Atom підтримує сучасні вебтехнології, добре працює з HTML, CSS, JavaScript, а також із популярними фреймворками (React, Vue, Node.js). Простий та

зрозумілий інтерфейс редактора робить його доступним для новачків, а функція Teletype дозволяє співпрацювати над кодом у реальному часі.

Недоліки

Серед основних недоліків Atom — порівняно низька продуктивність, особливо при роботі з великими файлами або складними проєктами. Через побудову на Electron редактор споживає значну кількість системних ресурсів і запускається повільніше, ніж легші альтернативи, такі як Sublime Text чи Visual Studio Code. Додатково, у грудні 2022 року GitHub офіційно припинив розвиток та підтримку Atom, що означає відсутність оновлень, виправлень помилок та нових функцій. Це також впливає на зниження інтересу з боку розробницької спільноти та зменшення кількості актуальних плагінів.

Atom залишається прикладом гнучкого та функціонального редактора, який підходить для навчання, легких проєктів і тих, хто цінує кастомізацію. Водночас, через припинення підтримки та технічні обмеження, він втрачає актуальність для професійної розробки у порівнянні з більш сучасними і продуктивними інструментами.

1.3.3 VisualStudio Code

Visual Studio Code (VS Code) — це безкоштовний, кросплатформенний текстовий редактор, розроблений компанією Microsoft. Він орієнтований на розробку програмного забезпечення з використанням різноманітних мов програмування, включаючи JavaScript, TypeScript, Python, C++, PHP, HTML, CSS та інші. VS Code поєднує простоту текстового редактора з потужністю повноцінної IDE завдяки великій кількості доступних розширень. Редактор має вбудовані функції підсвічування синтаксису, автодоповнення, інтеграцію з Git, відлагодження коду, термінал, а також підтримує інтелектуальний аналіз коду та роботу з фреймворками.

Однією з ключових переваг VS Code є його розширюваність — через вбудований магазин розширень користувач може адаптувати редактор до будь-якої технології чи стеку (наприклад, React, Node.js, Django, .NET тощо). Завдяки використанню движка Electron, редактор працює на Windows, macOS і Linux. Незважаючи на простоту, VS Code пропонує інструменти, які задовольняють потреби як новачків, так і досвідчених розробників. Його швидка робота, гнучкість, підтримка спільноти та регулярні оновлення зробили Visual Studio Code одним із найпопулярніших середовищ для сучасної розробки програмного забезпечення.

Інтерфейс VS Code представлений на рисунку 1.6

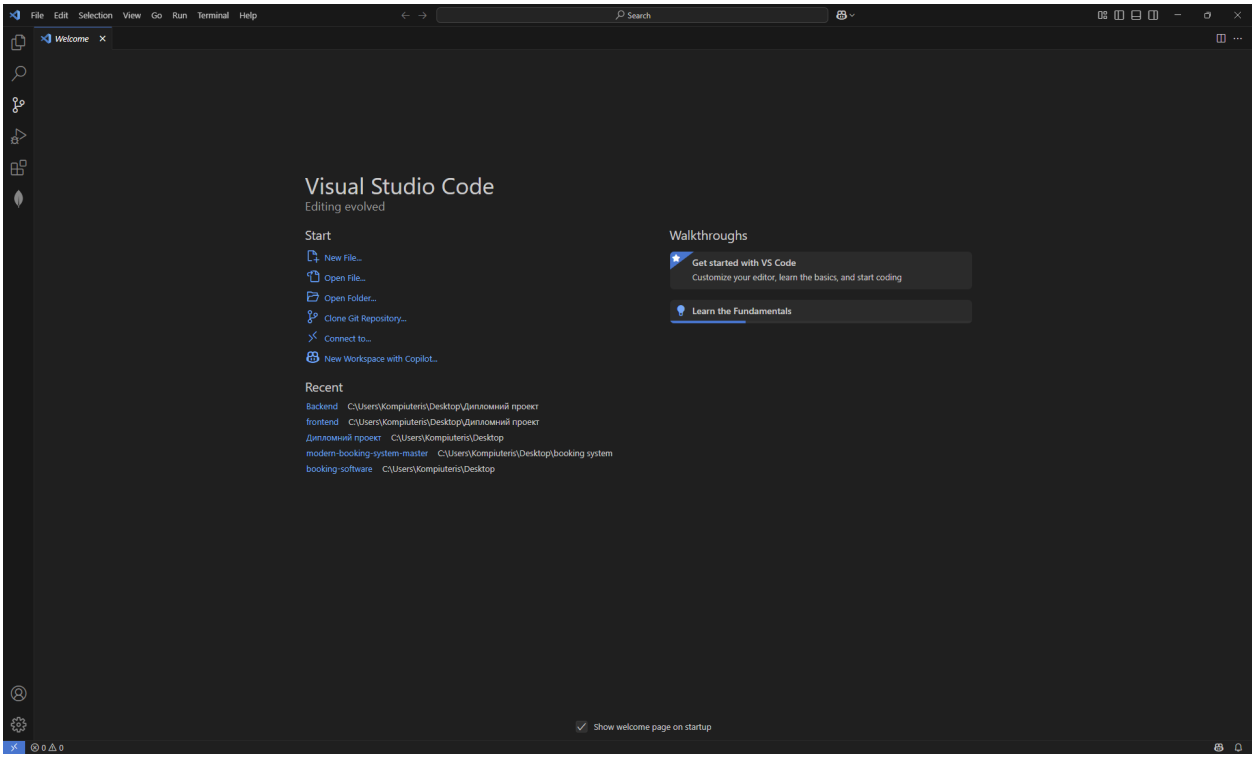


Рис. 1.6 VS Code

Переваги VS Code полягають у його багатофункціональності, гнучкості та підтримці великої кількості мов програмування. Редактор підтримує підсвічування синтаксису, автодоповнення, рефакторинг, інтеграцію з Git, вбудований термінал, відлагодження коду та розширення, які дозволяють адаптувати середовище під будь-який стек технологій — наприклад, для роботи з Node.js, React, MongoDB або REST

API. Значною перевагою є активна спільнота розробників та регулярні оновлення, які забезпечують стабільність, безпеку і підтримку нових функцій. Крім того, VS Code є кросплатформеним і працює однаково добре на Windows, Linux і macOS.

Недоліки VS Code пов'язані переважно з обмеженнями, що випливають з його природи як редактора, а не повноцінної IDE. Хоча за допомогою розширень VS Code можна адаптувати під будь-який проєкт, початково він не має вбудованих засобів для складних проєктів на рівні IntelliJ IDEA чи WebStorm. Іноді продуктивність редактора знижується при роботі з дуже великими файлами або великою кількістю одночасно активних розширень.

Порівняння средовищ розробки наведено в таблиці 1.2

Таблиця 1.2

Критерій	Brackets	Atom	Visual Studio Code
Розробник	Adobe	GitHub (належить Microsoft)	Microsoft
Платформи	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux
Швидкодія	Помірна	Повільна при великій кількості файлів	Висока
Інтерфейс	Мінімалістичний, орієнтований на веб	Високо налаштовуваний	Інтуїтивний і сучасний
Плагіни та розширення	Обмежена кількість	Велика кількість, але не завжди стабільні	Широкий вибір, офіційна підтримка

Продовження таблиці 1.2

Критерій	Brackets	Atom	Visual Studio Code
Підтримка Git	Обмежена через розширення	Через пакети	Вбудована підтримка
Оновлення та підтримка	Припинено у 2021 році	Припинено у 2022 році	Активно підтримується
Призначення	Веб-розробка (HTML, CSS, JS)	Загальне програмування	Універсальний редактор для будь-якого коду
Початок роботи	Простий для початківців	Простий	Простий і швидкий старт
Спільнота користувачів	Невелика, неактивна	Середня	Дуже велика і активна

Отже, VS Code є ідеальним вибором для реалізації веборієнтованих проєктів, таких як система бронювання, оскільки він надає все необхідне для розробки як клієнтської (React), так і серверної частини (Node.js + Express), підтримує MongoDB через відповідні плагіни, має чудову підтримку JavaScript та JSON і дозволяє швидко запускати локальні сервери, тестувати запити та керувати залежностями проєкту. Завдяки цьому VS Code забезпечує ефективне середовище для повного циклу розробки: від написання та тестування коду до його налагодження, запуску й супроводу.

2 ВИЗНАЧЕННЯ ВИМОГ ТА ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ТА АРХІТЕКТУРИ СИСТЕМИ

2.1 Визначення та моделювання вимог до web-застосунку

Метою розробки є створення web-застосунку для онлайн-бронювання столиків у ресторані з можливістю вибору часу, страв та місця за інтерактивною схемою залу. Система орієнтована на користувачів, які бажають здійснити бронювання заздалегідь, а також на адміністрацію закладу для перегляду поточної завантаженості.

ункціональні вимоги

а) Функціональні вимоги визначають, яку функціональність повинен забезпечити web-застосунок.

Реєстрація бронювання

- Користувач повинен мати можливість: Ввести своє ім’я;
- Обрати дату та час бронювання;
- Обрати стіл на інтерактивній схемі залу;
- Вибрати страви з меню та вказати їх кількість;
- Надіслати заявку на бронювання. Перевірка доступності столика
- Перед підтвердженням бронювання система перевіряє, чи не зайнятий стіл на вказану дату та час (з урахуванням періоду перебування клієнта).
- Візуальне відображення схеми залу
- Адміністративна частина (базова): Система повинна зберігати бронювання у базі даних для подальшого перегляду або аналізу.

Б) Нефункціональні вимоги

Нефункціональні вимоги визначають обмеження, які не стосуються конкретної поведінки системи, але впливають на її якість та ефективність.

- Продуктивність: веб-застосунок повинен обробляти запит бронювання протягом не більше ніж 2 секунд при середньому навантаженні.
- Масштабованість: архітектура має бути готовою до розширення — наприклад, для додавання реєстрації користувачів, адміністрування або підтримки кількох ресторанів.
- Зручність користування (Usability): Інтерфейс повинен бути інтуїтивно зрозумілим; Всі обов’язкові поля повинні мати підказки або позначення; Схема столів має бути зручною для мобільних пристроїв.
- Безпека: Дані користувача не передаються третім особам; Необхідна базова валідація введених даних для запобігання некоректному вводу.
- Надійність: У разі помилки сервера користувач повинен отримати відповідне повідомлення; Запити до серверу мають бути захищені від збоїв шляхом обробки винятків.
- Сумісність: Система повинна працювати у сучасних браузерах: Chrome, Firefox, Edge, Safari; Підтримка адаптивної верстки для планшетів і смартфонів.

Для моделювання функціональності веб-застосунку онлайн-бронювання столиків було побудовано діаграму варіантів використання(Рис. 2.1). Вона демонструє основні сценарії взаємодії користувачів із системою, відображаючи ключові дії, які можуть бути виконані як звичайними користувачами, так і адміністраторами.

На діаграмі визначено двох основних акторів: Користувач та Адміністратор.

Користувач має змогу здійснювати наступні дії:

- переглядати інтерактивну схему розміщення столиків у залі;
- обирати бажаний столик із доступних на момент бронювання;
- зазначати дату та час відвідування;
- формувати попереднє замовлення страв, вказуючи назву та кількість;
- надсилати запит на бронювання.

Перед підтвердженням, система автоматично здійснює перевірку доступності обраного столика на зазначений час, щоб уникнути конфлікту з уже існуючими

бронюваннями. Якщо столик є вільним, система приймає бронювання та зберігає відповідні дані.

Адміністратор має розширені можливості доступу до функцій системи, а саме:

- перегляд усіх створених бронювань;
- управління даними користувачів та їх бронюваннями (редагування, видалення, підтвердження тощо).

Використання діаграми варіантів дозволяє чітко ідентифікувати основні функціональні можливості системи та сформувати основу для подальшого моделювання логіки взаємодії компонентів на наступних етапах розробки.

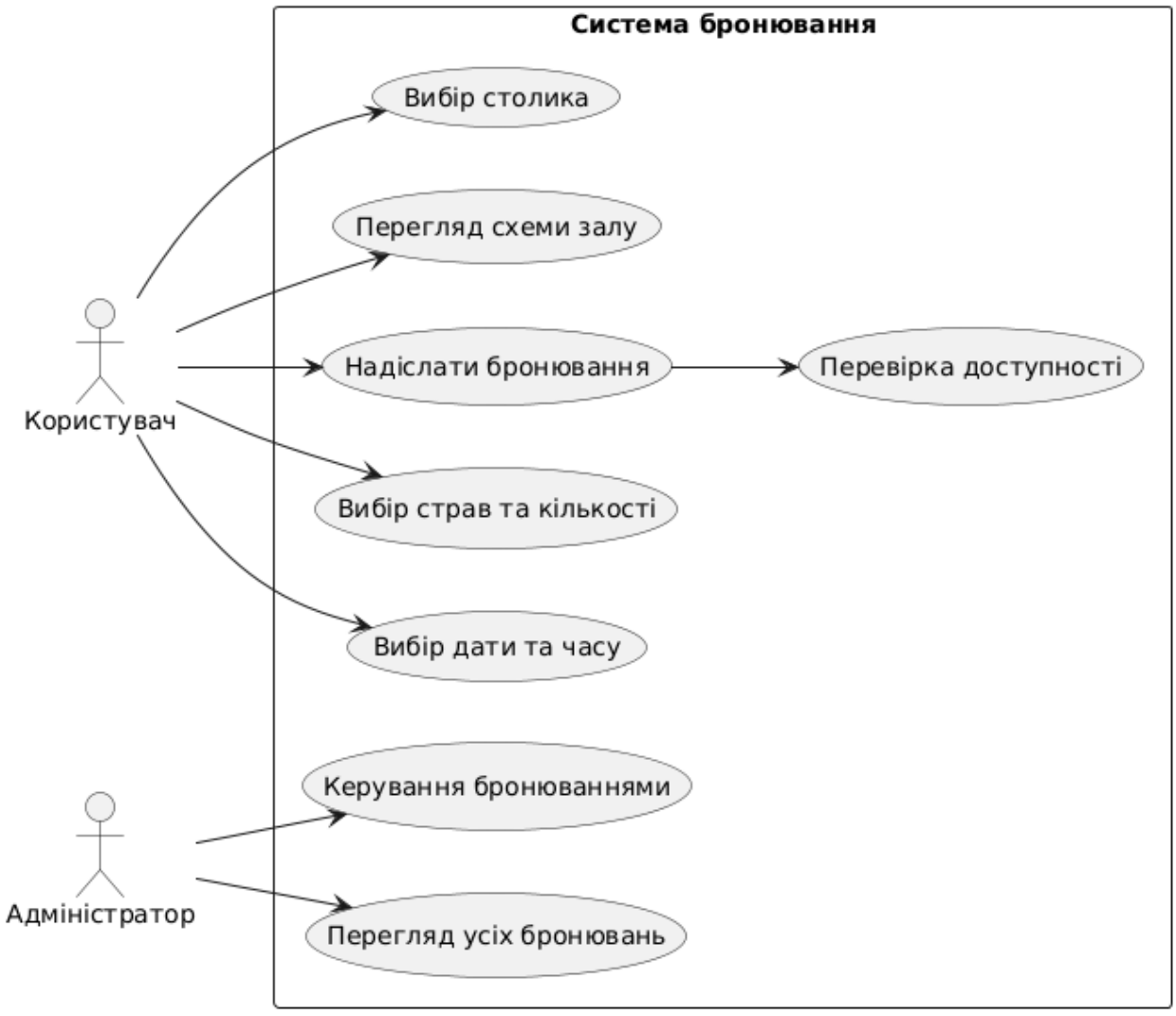


Рис. 2.1 Діаграма використання

Web-застосунок для бронювання столиків реалізований за принципом клієнт-серверної архітектури з чітким поділом на три основні рівні: клієнтський інтерфейс (frontend), серверна логіка (backend) та база даних (MongoDB). Нижче представлено детальний опис кожного з компонентів та їх взаємодії.

Користувач взаємодіє із застосунком через графічний інтерфейс, реалізований за допомогою бібліотеки React. Інтерфейс надає можливість вибрати стіл на інтерактивній схемі, вказати дату та час бронювання, а також замовити страви. Усі дії користувача супроводжуються відправленням відповідних HTTP-запитів на сервер.

Серверна частина реалізована з використанням фреймворку Express.js. Вона приймає вхідні запити (наприклад, POST /api/reservations), виконує валідацію та обробку даних, а також передає ці дані до моделі, яка відповідає за взаємодію з базою даних. Контролери у складі сервера координують логіку збереження та отримання інформації про бронювання.

Модель даних, побудована з використанням бібліотеки Mongoose, відповідає за структурування інформації, що зберігається у базі даних. Вона інкапсулює логіку створення, зчитування та оновлення документів у колекції reservations.

База даних MongoDB зберігає всі дані про здійснені бронювання, включаючи ім’я користувача, дату, час, номер столика, а також обрані страви. При надходженні нового запиту до сервера, система перевіряє наявність конфліктів (наприклад, чи вільний стіл на вказану дату та час), і лише за відсутності таких даних виконується збереження.

Узагальнено, web-застосунок функціонує за схемою: користувач → інтерфейс → сервер → база даних → сервер → відповідь інтерфейсу.

Ця архітектура забезпечує масштабованість, розширюваність та спрощує підтримку й тестування застосунку, оскільки дозволяє змінювати кожен компонент незалежно від інших.

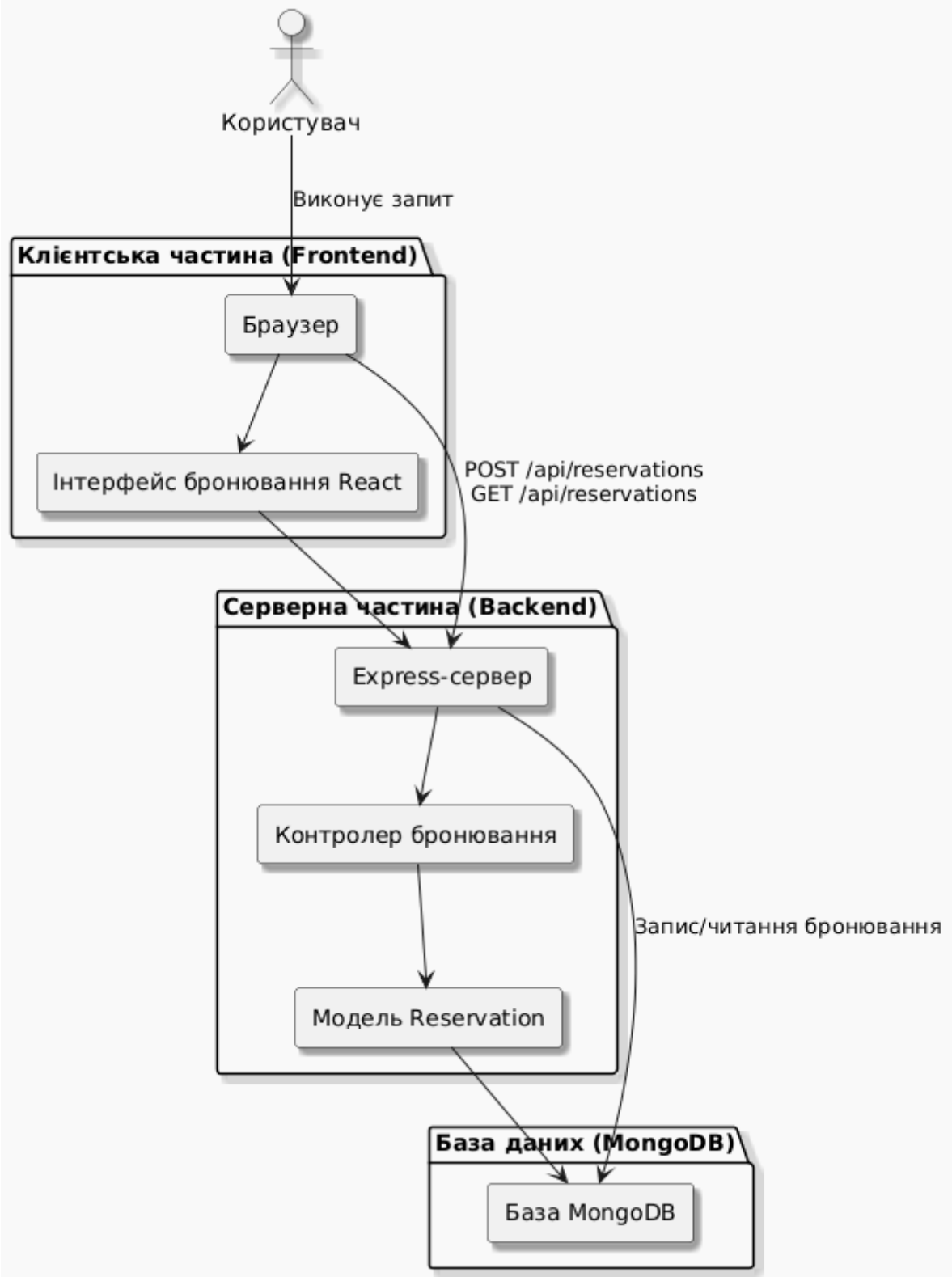


Рис 2.2 Схема роботи веб-додатку

2.2 Структура клієнтської частини web-застосунку

Клієнтська частина (frontend) web-застосунку є важливою складовою інформаційної системи з підтримки бронювання столиків та оформлення замовлень у ресторані. Вона відповідає за взаємодію кінцевого користувача з функціоналом системи, відображення інтерфейсу та обробку подій. Застосунок повинен бути інтуїтивно зрозумілим, адаптивним до різних пристроїв та забезпечувати швидку реакцію на дії користувача.

У процесі реалізації клієнтської частини обрано компонентно-орієнтований підхід, заснований на використанні бібліотеки React, що дозволяє створювати модульну, гнучку та повторно використовувану структуру інтерфейсу. Компоненти React формують цілісну ієрархію елементів, що відповідають за конкретні функції інтерфейсу, наприклад: сторінка реєстрації, форма бронювання, вікно перегляду меню, панель адміністратора тощо.

Дана частина представлена п’ятьма класами.

Клас Reservation моделює інформацію про бронювання столика користувачем. Він зберігає ключові атрибути, включаючи:

1. Клас Reservation

- id: унікальний ідентифікатор запису в базі даних MongoDB.
- name: ім’я користувача, який здійснює бронювання.
- table: номер або ідентифікатор столика, що бронюється.
- time: дата та час бронювання.
- dishes: список об’єктів типу Dish, які містять замовлені страви.

Також містить методи:

- createReservation(): створює новий запис бронювання.
- cancelReservation(): скасовує чинне бронювання.

2. Клас Dish

Клас Dish представляє замовлену страву у складі конкретного бронювання. Він виконує роль вкладеної сутності в межах Reservation:

- dish: назва страви (рядок).
- quantity: кількість одиниць обраної страви (ціле число).
- Цей клас спрощує моделювання кількох замовлень у межах однієї броні.

3. Клас Table

Клас Table моделює столик у закладі. Його атрибути дозволяють точно визначити розташування та доступність:

- tableId: унікальний ідентифікатор столика.
- positionX, positionY: координати столика на схемі залу.
- Метод isAvailable(time: DateTime): визначає, чи є столик вільним у заданий час.

Цей клас також може бути використаний для візуалізації в інтерфейсі користувача.

4. Клас UserInterface

Клас UserInterface описує дії, які може виконувати кінцевий користувач через графічний інтерфейс веб-застосунку:

- fillForm(): заповнення форми з іменем, часом та іншими даними.
- selectTable(): вибір столика зі схеми (наприклад, SVG).
- submitReservation(): надсилання бронювання до сервера.

Цей клас логічно моделює фронтенд і його взаємодію з користувачем.

5. Клас ReservationController

ReservationController — це серверний контролер (частина REST API), що обробляє HTTP-запити від клієнта:

- postReservation(): обробляє POST-запит для створення нового бронювання.
- getAvailability(): перевіряє, чи доступний обраний столик на вказану дату та час.

– Контролер взаємодіє з базою даних і логікою доступності столиків.

Ці класи разом утворюють логічну модель веб-застосунку, що дозволяє користувачам здійснювати бронювання столиків, замовляти страви та візуально обирати місця у залі закладу.

Клієнтська частина представлена у вигляді діаграми класів на рисунку 2.3

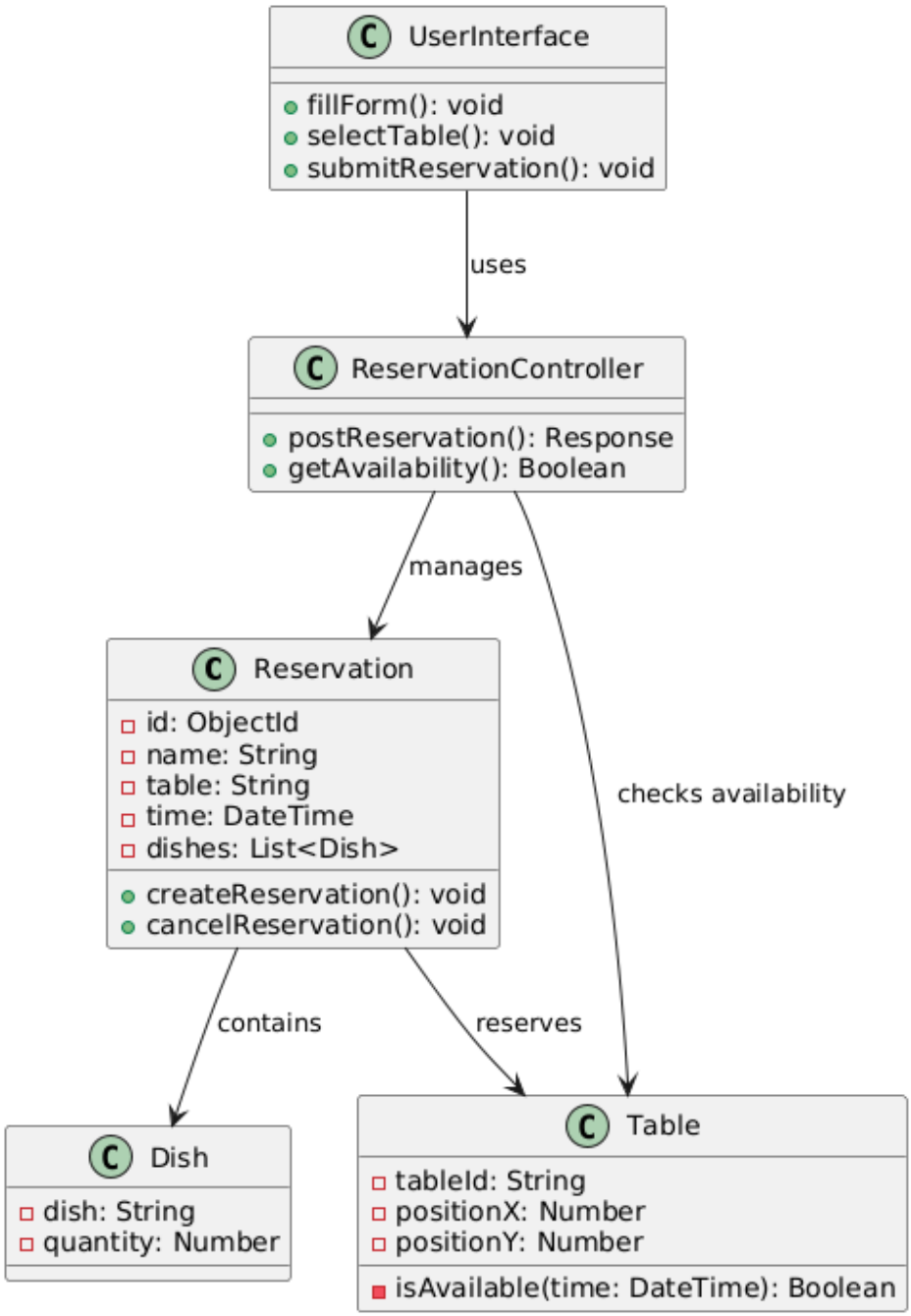


Рис. 2.3 Діаграма класів

2.3 Структура серверної частини web-застосунку

Серверна частина web-застосунку виконує ключову роль у забезпеченні взаємодії між клієнтським інтерфейсом користувача та базою даних. Вона реалізована з використанням технологій Node.js і Express.js, що дозволяє створювати продуктивні та масштабовані RESTful API. Основною функцією серверної частини є обробка запитів від клієнта, збереження даних про бронювання, перевірка доступності столиків на вказаний час, а також передача інформації назад до клієнта.

Головним файлом серверної частини є index.js. У ньому виконується підключення необхідних бібліотек, таких як Express, Mongoose (для взаємодії з MongoDB) та CORS (для забезпечення взаємодії між фронтендом і бекендом). Також у цьому файлі реалізовано підключення до бази даних MongoDB через змінні середовища, що зберігаються у .env файлі. Після успішного підключення сервер прослуховує порт і забезпечує обробку HTTP-запитів.

Маршрутизація (routing) реалізована в окремому файлі menuRoutes.js, який відповідає за обробку запитів на створення нових бронювань та перевірку доступності столиків. Зокрема, при створенні бронювання дані з клієнтської частини надходять до API-запиту POST /api/reservations, де вони зберігаються в базі даних. Для перевірки доступності столика використовується запит GET, який порівнює дату, час і номер столика з уже наявними бронюваннями.

Для збереження структури даних використовується Mongoose-схема, визначена у файлі Reservation.js. Вона включає основні поля: ім’я клієнта, номер столика, дату та час бронювання, а також масив обраних страв і кількість кожної з них. Завдяки цій структурі, серверна частина може ефективно опрацьовувати як загальну інформацію про замовлення, так і деталі замовлених страв.

Під час обробки запитів сервер виконує валідацію вхідних даних та обробку можливих помилок. Наприклад, у разі відсутності обов’язкових полів або при спробі зарезервувати вже зайнятий столик користувач отримає відповідне повідомлення.

Таким чином, серверна частина web-застосунку є логічно структурованою системою, що забезпечує надійне збереження даних, ефективну перевірку доступності ресурсів та безперебійну інтеграцію з клієнтською частиною. Обрана технологічна база дозволяє масштабувати застосунок у разі потреби, а також інтегрувати додаткові сервіси у майбутньому.

Reservation — основна колекція, яка зберігає бронювання.

DishOrder — вкладені об'єкти (масив), що описують обрані страви.

Відношення contains означає, що **Reservation** містить масив об'єктів **DishOrder**.

У MongoDB вся інформація зберігається як вкладені документи, тому ця діаграма(Рис 2.4) умовно подає ці зв'язки у вигляді окремих сутностей, як у реляційних БД.

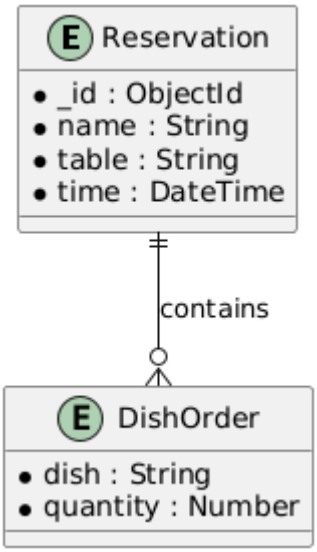


Рис. 2.4 Діаграма бази даних

3. РОЗРОБКА WEB-ДОДАТКУ

3.1 Опис інструментів реалізації

JavaScript — це динамічна, високорівнева мова програмування, яка найчастіше використовується для створення інтерактивних елементів на вебсторінках. Це одна з основних технологій сучасного веброзроблення поряд із HTML (структура сторінки) та CSS (оформлення).

JavaScript виконується безпосередньо у веббраузері користувача, що дозволяє змінювати вміст сторінки без її перезавантаження, реагувати на дії користувача (натискання кнопок, введення тексту тощо), обробляти події, створювати анімації, надсилати запити до сервера (через AJAX або Fetch API) та багато іншого.

На відміну від HTML і CSS, які лише описують, як виглядає і структурується сторінка, JavaScript дозволяє керувати поведінкою елементів сторінки в режимі реального часу. Сучасні вебзастосунки, такі як Gmail, Facebook або YouTube, значною мірою побудовані саме на JavaScript.

З розвитком екосистеми, JavaScript перестав бути лише «мовою для браузера» — завдяки платформі Node.js він також широко використовується на сервері. Крім того, з’явилися популярні фреймворки та бібліотеки, такі як React, Angular, Vue.js, що значно спрощують і пришвидшують розробку великих та динамічних інтерфейсів.

JavaScript підтримує об’єктно-орієнтований, функціональний і подієво-орієнтований стилі програмування. Його простий синтаксис і широка підтримка в браузерах зробили його однією з найпопулярніших мов програмування у світі, особливо в галузі веброзробки.

HTML (HyperText Markup Language) — це мова розмітки гіпертексту, яка використовується для створення структури та вмісту вебсторінок. HTML не є мовою

програмування — це мова, яка описує, що саме повинно відображатися у браузері: заголовки, абзаци, зображення, таблиці, посилання, форми та інші елементи.

HTML складається з тегів, які обгортають вміст і вказують браузеру, як його відображати. Наприклад, <h1> означає заголовок першого рівня, <p> — абзац, <img> — зображення, <a> — гіперпосилання. Всі елементи HTML разом формують DOM (Document Object Model) — ієрархічну структуру вебсторінки, з якою потім можуть взаємодіяти CSS (оформлення) та JavaScript (поведінка).

Одним із ключових інструментів для реалізації зовнішнього вигляду веб-застосунку є CSS (Cascading Style Sheets) — каскадні таблиці стилів. CSS використовується для візуального оформлення HTML-елементів, тобто визначає, як виглядатиме інтерфейс користувача. За допомогою CSS задаються кольори, шрифти, відступи, розміри, рамки, розташування елементів на сторінці та інші стилістичні властивості.

У даному проекті CSS відіграє важливу роль у формуванні зручного та інтуїтивно зрозумілого інтерфейсу для користувача. Він забезпечує адаптивність інтерфейсу до різних пристроїв, візуально виділяє активні або забронювані елементи, формує логічну структуру блоків та допомагає покращити користувацький досвід. Стилі були реалізовані як у вигляді зовнішніх CSS-файлів, так і безпосередньо у компонентах React.

Таким чином, CSS у проєкті є невід’ємною частиною фронтенд-розробки, забезпечуючи привабливий вигляд і комфортне використання веб-застосунку.

У сукупності з CSS та JavaScript, HTML формує фронтенд — ту частину вебзастосунку, яку бачить і з якою взаємодіє користувач. У дипломному проєкті HTML відповідає за розмітку таких елементів, як форма бронювання, меню, таблиця замовлень або сторінка входу до системи.

React.js (або просто React) — це JavaScript-бібліотека для створення інтерфейсів користувача (UI), розроблена компанією Meta (Facebook). Вона дозволяє створювати

швидкі, динамічні та багаторазово використовувані компоненти інтерфейсу для вебзастосунків.

Головна ідея React полягає у компонентному підході — замість цілісних сторінок розробник створює окремі модулі (компоненти), які відповідають за певну частину інтерфейсу: кнопка, форма, блок меню, список тощо. Ці компоненти можуть мати свій стан (state), приймати властивості (props) і змінювати свій вигляд або поведінку залежно від подій користувача.

React використовує віртуальний DOM (Virtual DOM) — внутрішнє представлення сторінки в пам’яті, що дозволяє швидко виявляти й оновлювати тільки ті елементи, які реально змінились. Це робить React дуже швидким і ефективним навіть для великих вебзастосунків.

Ще однією особливістю React є синтаксис JSX — розширення JavaScript, що дозволяє писати HTML-подібний код прямо в JavaScript-функціях, що значно покращує читабельність та структуру коду.

React активно використовується в сучасній фронтенд-розробці, зокрема для побудови односторінкових застосунків (SPA), а також у поєднанні з бекендом на Node.js. У дипломному проєкті, присвяченому системі бронювання столиків у ресторані, React дозволяє створити зручний, адаптивний та інтерактивний інтерфейс для користувачів і адміністраторів, забезпечуючи швидкий відгук та комфортну взаємодію з системою.

Node.js — це програмна платформа, яка дозволяє запускати JavaScript-код на стороні сервера, тобто поза веббраузером. Вона створена на основі рушія V8, який також використовується в браузері Google Chrome для виконання JavaScript.

Головна перевага Node.js полягає у тому, що вона забезпечує асинхронне, неблокувальне виконання коду, що дозволяє ефективно обробляти велику кількість одночасних запитів — наприклад, у чатах, системах бронювання, реальному часі тощо.

Express.js — це легкий та гнучкий фреймворк для Node.js, який використовується для створення серверної частини (backend) вебзастосунків і REST API. Він забезпечує зручні інструменти для обробки HTTP-запитів, маршрутизації, підключення до баз даних, управління сесіями, обробки помилок та багато іншого.

Фактично, Express.js спрощує процес створення вебсерверів на Node.js, абстрагуючи складні низькорівневі операції та надаючи компактний синтаксис і чітку архітектуру. Завдяки цьому розробники можуть швидко створювати надійні, масштабовані вебсервіси.

MongoDB — це нереляційна база даних (NoSQL), яка зберігає дані у вигляді документів у форматі BSON (розширений JSON). На відміну від класичних SQL-баз, де дані зберігаються у таблицях зі строгими схемами, MongoDB дозволяє зберігати документи з довільною структурою, що робить її дуже гнучкою та зручною для швидкої розробки.

MongoDB чудово підходить для вебзастосунків, де дані часто змінюються, мають вкладену структуру або різні формати. База даних легко масштабується, швидко працює з великими обсягами інформації, підтримує шардінг (розподілення даних між серверами), реплікацію (резервне дублювання) та високий рівень доступності.

У дипломному проєкті MongoDB використовується для зберігання інформації про користувачів, бронювання, замовлення, меню, столики тощо.

Mongoose — це бібліотека для Node.js, яка забезпечує об’єктно-документне моделювання (ODM) для MongoDB. Вона надає розробникам інтерфейс для роботи з MongoDB через схеми моделей, подібно до того, як ORM (Object-Relational Mapping) працює з SQL-базами.

У контексті дипломного проєкту Mongoose спрощує інтеграцію між сервером на Node.js (через Express) та базою MongoDB, забезпечуючи зручну роботу з базовими даними — бронюваннями, замовленнями, профілями клієнтів тощо.

3.2 Опис реалізації backend

Backend — це серверна частина веб-застосунку, яка працює “за лаштунками” і не відображається напряму користувачам. Його головне завдання — обробка запитів, взаємодія з базою даних, виконання бізнес-логіки та забезпечення цілісності й безпеки даних.

У проєкті, присвяченому системі бронювання столиків у закладі громадського харчування, бекенд відіграє ключову роль. Він приймає запити від користувача, наприклад, на створення, зміну чи видалення бронювання. Уся інформація обробляється через сервер і зберігається в базі даних MongoDB.

Для розробки серверної частини було використано платформу Node.js та фреймворк Express, які дозволили створити структуру REST API — інтерфейс, через який frontend (клієнтська частина) може обмінюватися даними з сервером. Це забезпечує зручну та гнучку взаємодію між частинами застосунку.Основні компоненти backend:

1. Файл index.js – головний серверний файл(Рис 3.1), що:

– Ініціалізує Express-додаток.

– Встановлює middleware (cors, express.json()).

– Встановлює з'єднання з MongoDB за допомогою mongoose.connect(), зчитуючи рядок підключення з .env.

– Підключає маршрути з файлу routes/menuRoutes.js.

– Реалізує базовий маршрут для перевірки доступності (GET /).

– Запускає сервер на визначеному порту.

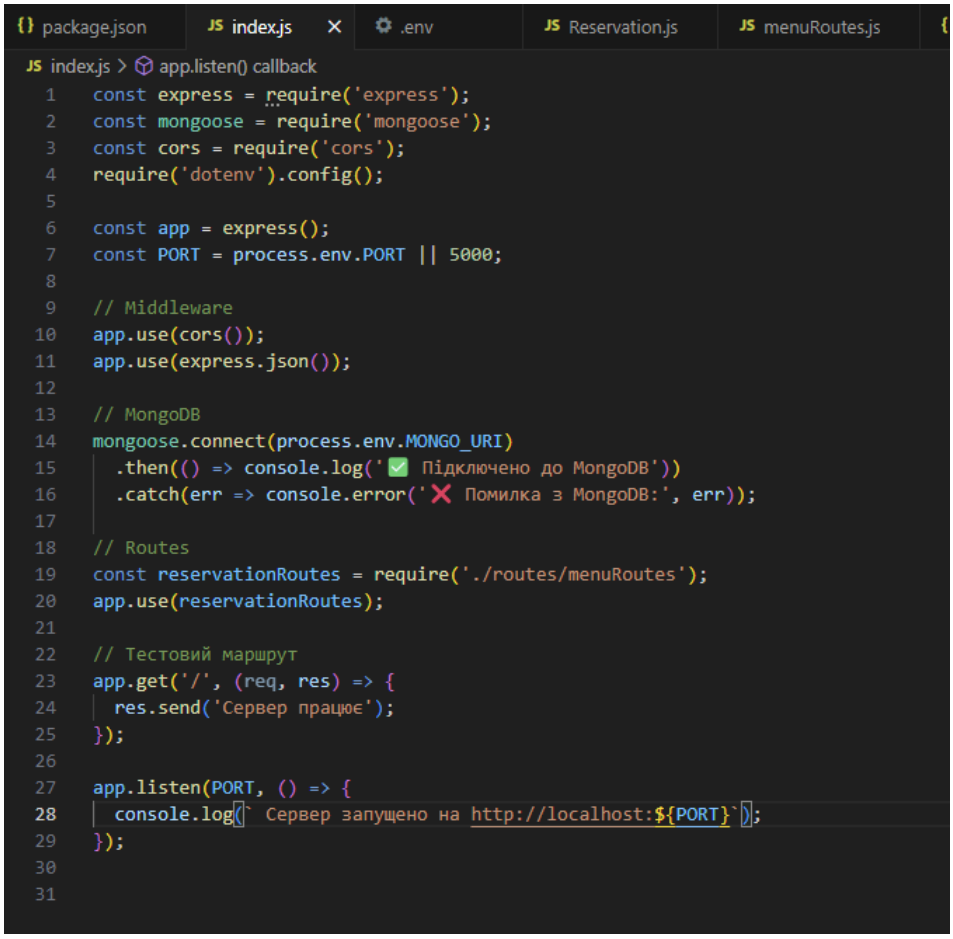


Рис. 3.1

2. Маршрутизація:

Файл routes/menuRoutes.js(Рис. 3.2) містить обробник POST-запиту для створення нового бронювання (endpoint: /api/reservations). Усі отримані з клієнта дані перевіряються та зберігаються до бази.

routes > menuRoutes.js > router.post('/api/reservations') callback > existing

```
const express = require('express');
const router = express.Router();
const Reservation = require('../models/Reservation');

router.post('/api/reservations', async (req, res) => {
  try {
    const { table, time } = req.body;

    const existing = await Reservation.findOne({
      table,
      time: { $lte: time }
    });

    if (existing) {
      return res.status(409).json({ message: 'Цей столик вже зарезервовано на обраний час або пізніше' });
    }

    const reservation = new Reservation(req.body);
    await reservation.save();

    res.status(201).json({ message: 'Успішно збережено!' });
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
});

router.get('/api/reservations', async (req, res) => {
  try {
    const reservations = await Reservation.find();
    res.json(reservations);
  } catch (err) {
    res.status(500).json({ error: 'Помилка при отриманні даних' });
  }
});

module.exports = router;
```

Рис. 3.2

3. Модель Reservation:

– Знаходиться у файлі models/Reservation.js.(Рис. 3.3)

– Визначає схему з такими основними полями:

– name – ім’я користувача.

– table – номер або ідентифікатор столика.

– time – дата і час бронювання.

– dishes – масив об’єктів із назвою страви та кількістю (поля: dish, quantity).

Рис. 3.3

4. Файл .env:

– Містить конфіденційні змінні середовища, зокрема рядок підключення до бази даних MongoDB (не розголошується у відкритому коді для безпеки).

5. Залежності (у package.json):

– "express" – для створення серверу.

– "mongoose" – для взаємодії з MongoDB.

– "cors" – для підтримки кросдоменної взаємодії.

– "dotenv" – для роботи з конфігураційними змінними середовища.

Коли користувач на фронтенді заповнює форму бронювання, зокрема вибирає стіл, час, ім’я та страви, дані надсилаються до бекенду через POST-запит. Сервер отримує ці дані, перевіряє їх відповідність схемі, зберігає у MongoDB і повертає клієнту повідомлення про успіх або помилку.

У процесі реалізації серверної частини веб-застосунку було використано низку бібліотек, що забезпечили зручне створення REST API, роботу з базою даних, обробку запитів і підвищення безпеки застосунку:

– Express: основний фреймворк для створення серверної частини, маршрутів, обробки запитів.

– Mongoose: ORM-бібліотека для взаємодії з базою даних MongoDB, дозволяє працювати з моделями.

– Cors: дозволяє налаштувати політику доступу між клієнтом і сервером (дозвіл CORS).

– Body-parser: забезпечує розбір тіла HTTP-запитів, особливо у форматі JSON.

– Dotenv: дозволяє зберігати конфіденційні дані (наприклад, URL бази даних) у файлі .env.

– Nodemon: інструмент для автоматичного перезапуску сервера під час розробки при зміні коду.

– MongoDB: бібліотека/драйвер для прямої роботи з базою даних MongoDB (опційно, якщо не через Mongoose).

3.3 Опис реалізації frontend

Frontend — це клієнтська частина веб-застосунку, яка безпосередньо взаємодіє з користувачем. Це все, що бачить і з чим взаємодіє відвідувач сайту: інтерфейс, кнопки, форми, меню, таблиці, повідомлення тощо.

У проєкті системи бронювання столиків frontend відіграє ключову роль у забезпеченні зручного, інтуїтивно зрозумілого та привабливого користувацького досвіду. Саме через цю частину користувач вводить дані (наприклад, ім’я, час бронювання, номер столика або замовлені страви), надсилає запити до сервера і отримує зворотний зв’язок (наприклад, повідомлення про успішне бронювання).

Фронтенд частина web-застосунку реалізована з використанням бібліотеки React.js, що забезпечує створення компонентно-орієнтованого, динамічного і масштабованого інтерфейсу користувача(Рис. 3.4). Додатково використано стандартні технології HTML5, CSS3 для стилізації та формування адаптивного інтерфейсу.

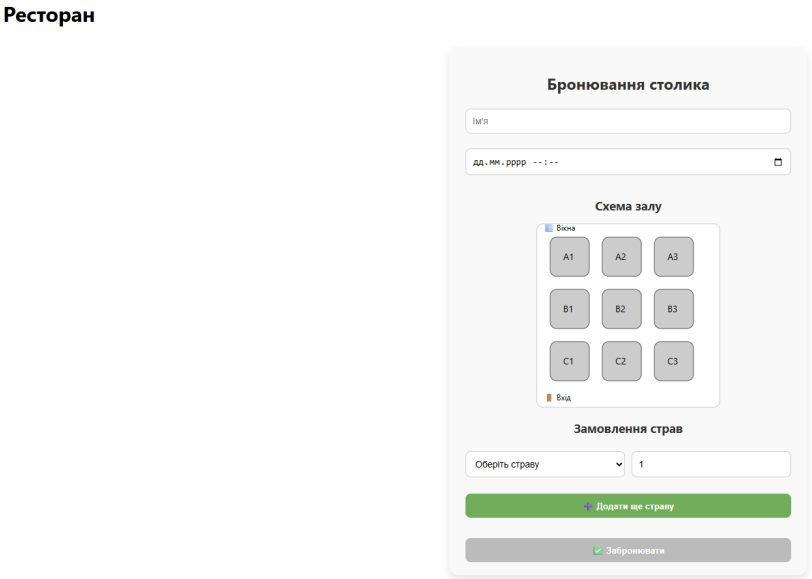


Рис. 3.4 Інтерфейс

Основні компоненти

Ключовим компонентом є ReservationForm, який відповідає за формування заявки на бронювання, взаємодію зі схемою залу та відправку інформації на сервер(Рис. 3.5).

– Функціональність компонента охоплює:

– Введення даних користувача: поле для введення імені.

– Вибір дати та часу бронювання: через input type="datetime-local".

– Інтерактивна схема залу: реалізована у вигляді SVG-сітки 3×3 столиків, де кожен елемент можна вибрати натисканням.

– Відображення зайнятих столиків: при виборі дати й часу система отримує з API список зарезервованих столиків та динамічно блокує їх вибір (візуально позначаючи сірим кольором).

– Формування замовлення страв: реалізовано у вигляді списку, де можна обрати страву та кількість; список можна динамічно розширювати кнопкою «Додати ще страву».

– Відправлення форми: всі введені дані надсилаються POST-запитом до серверної частини застосунку.

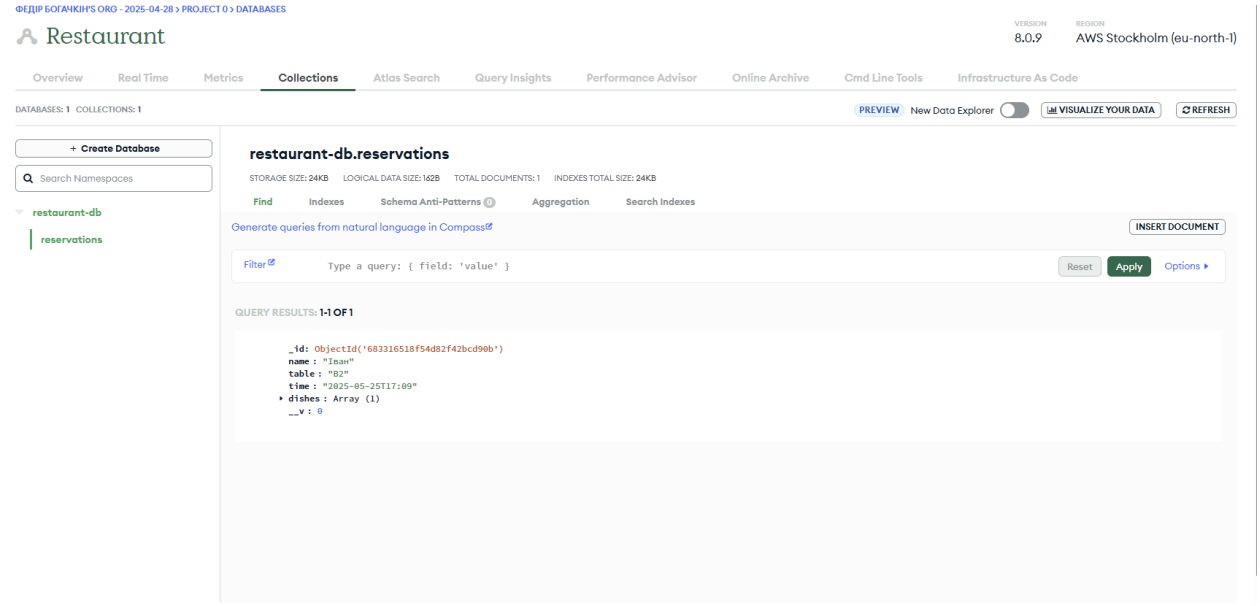


Рис. 3.5 Вигляд замовлення в MongoDB

Стилізація

Для оформлення інтерфейсу використовується окремий CSS-файл (ReservationForm.css), що забезпечує:

– сучасний і мінімалістичний дизайн;

– інтуїтивно зрозумілу структуру сторінки;

– візуальне виділення активних, зайнятих та вибраних столиків;

– підписи до схеми залу (наприклад, «Вхід», «Вікна») для просторової орієнтації користувача.

UX-дизайн

Було враховано принципи **доступності та зручності**:

- кнопки мають зрозуміле призначення;
- обов’язкові поля мають валідацію;
- реакція на дії користувача — миттєва (наприклад, зміна кольору при виборі столика);
- використано емодзі й текстові підказки для покращення досвіду.

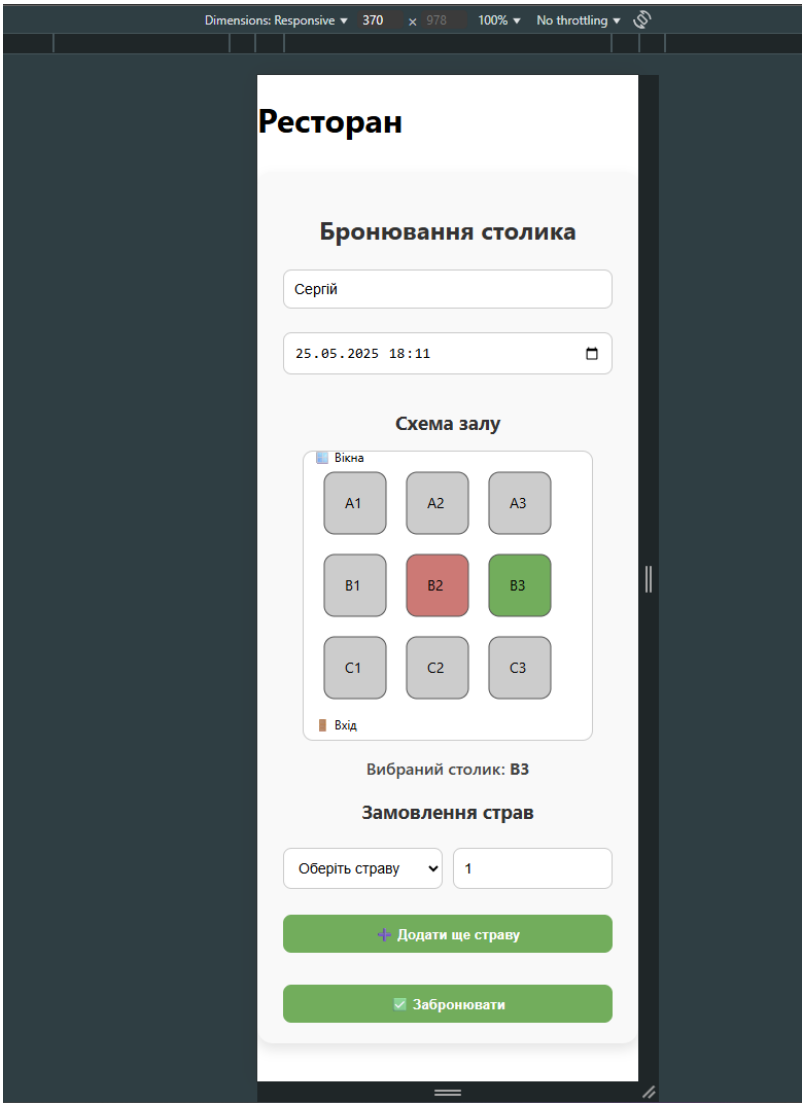


Рис 3.6 Інтерфейс на мобільному пристрої

Для створення клієнтської частини веб-застосунку використано фреймворк React та допоміжні бібліотеки, які забезпечують зручну розробку, гнучкий інтерфейс та інтерактивність:

– React: основна бібліотека для побудови інтерфейсу користувача за компонентним підходом.

– React DOM: бібліотека для рендерингу компонентів у DOM (використовується автоматично в React).

– Axios / Fetch API: інструменти для надсилання HTTP-запитів до backend-сервера.

– React Icons / SVG: використання векторної графіки та іконок у інтерфейсі.

– CSS / SCSS: каскадні таблиці стилів або препроцесори для оформлення інтерфейсу.

– React Hook Form / Formik: бібліотеки для обробки форм, валідації, зручного керування станом форм.

Інструменти розробки:

– Create React App — шаблон для швидкого створення проєкту з готовими налаштуваннями.

– ESLint + Prettier — для дотримання єдиного стилю коду (якщо налаштовано).

– DevTools (React Developer Tools) — розширення браузера для налагодження стану компонентів.

4. ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Актуальність тестування

Перед запуском web-застосунку в реальне середовище важливо впевнитися в його стабільності, правильній роботі всіх функцій та відповідності сформульованим вимогам. Це досягається шляхом проведення тестування, яке є невід'ємною частиною життєвого циклу розробки програмного забезпечення. Тестування — це процес оцінювання якості програмного продукту з метою виявлення дефектів, забезпечення відповідності вимогам, зменшення ризиків і підвищення надійності системи.

Тестування починається ще на ранніх етапах — під час формування вимог, коли перевіряється повнота, точність, несуперечливість та однозначність описаного функціоналу. Надалі створення прототипів дозволяє перевірити зручність інтерфейсу користувача та виявити недоліки у взаємодії ще до реалізації основного функціоналу.

У процесі розробки застосовується низка видів тестування, кожен із яких виконує свою функцію. Модульне тестування забезпечує перевірку окремих частин коду (функцій, компонентів) на правильність виконання. Інтеграційне тестування перевіряє взаємодію між модулями та дозволяє виявити помилки, що виникають при їх об'єднанні. Після цього виконується системне тестування, яке перевіряє роботу програми в цілому, в умовах, наближених до реального середовища. На цьому етапі використовуються техніки тестування граничних значень, введення випадкових даних, методи вгадування помилок, а також регресійне тестування, яке дозволяє переконатися, що зміни в коді не спричинили нових збоїв у вже реалізованому функціоналі.

Заключним етапом є приймальне тестування, під час якого перевіряється відповідність системи технічним вимогам, специфікації та очікуванням замовника. Це

тестування може виконуватись як розробниками, так і незалежними тестувальниками або представниками цільової аудиторії.

Особливу увагу в рамках дипломної роботи приділено тестуванню web-застосунків, що включає перевірку функціональності, адаптивності інтерфейсу для різних пристроїв, продуктивності, безпеки та зручності користування. Функціональне тестування забезпечує коректне виконання основних сценаріїв взаємодії користувача із системою, перевіряючи виконання кожної функції відповідно до вимог. Тестування безпеки дозволяє виявити потенційні уразливості, пов’язані з доступом до даних, автентифікацією та збереженням конфіденційної інформації. Перевірка продуктивності включає оцінку часу завантаження сторінок і швидкості реакції сервера, що є критично важливим для забезпечення якісного користувацького досвіду.

Таким чином, тестування є ключовим етапом у розробці web-застосунку, оскільки воно не лише дозволяє виявити та усунути потенційні проблеми ще до релізу, але й гарантує якість, безпеку, надійність та задоволення потреб кінцевого користувача. Виконання повного циклу тестування забезпечує високий рівень довіри до програмного продукту, зменшує ризики фінансових втрат і сприяє успішному впровадженню системи в реальне середовище.

4.2 Обґрунтування вибору підходів та технік тестування

Для забезпечення надійності та коректної роботи розробленого web-застосунку в процесі розробки було прийнято рішення застосовувати низку перевірених методів і технік тестування, які дозволяють виявити помилки на різних рівнях функціональності та забезпечити відповідність системи вимогам користувача.

У рамках даної дипломної роботи обрано функціональний підхід до тестування, з орієнтацією на очікувану поведінку системи згідно з технічними та бізнес-вимогами. Основним методом тестування, що буде використано на фазі активної перевірки системи, є метод чорного ящика. Цей підхід не вимагає знань про внутрішню

реалізацію програмного коду, а зосереджується на вхідних та вихідних даних, що дає змогу моделювати взаємодію кінцевого користувача з інтерфейсом застосунку. Метод чорного ящика є найбільш доцільним для тестування інтерфейсів, форм введення, бізнес-логіки та відповідей від сервера.

Для фази практичного тестування буде застосовано мануальне (ручне) тестування, що дозволить гнучко реагувати на зміни, оперативно фіксувати помилки та оцінювати поведінку інтерфейсу в реальному середовищі. Мануальне тестування є доцільним для первинної перевірки логіки дій користувача, функціонування адаптивного інтерфейсу, валідації форм, обробки повідомлень про помилки, навігації та інших аспектів, що вимагають людського спостереження та суб'єктивної оцінки зручності використання.

У процесі тестування планується реалізація як позитивного, так і негативного тестування.

Позитивне тестування спрямоване на перевірку очікуваної поведінки системи при введенні коректних даних і дотриманні правильних сценаріїв використання. Наприклад, тестування реєстрації користувача з валідною електронною поштою та паролем, або підтвердження бронювання за доступною датою.

Негативне тестування, навпаки, перевіряє реакцію системи на некоректні або неочікувані дії, такі як введення неправильного формату даних, залишення обов’язкових полів порожніми, спроба повторного бронювання зайнятого столика тощо. Мета негативного тестування — переконатися, що система правильно обробляє помилки, не допускає збоїв, надає користувачеві зрозумілий зворотний зв’язок і забезпечує стійкість до можливих маніпуляцій або помилкових дій.

Також при виконанні тестування буде використано техніки класів еквівалентності та граничних значень, які дозволяють охопити значну частину вхідних даних з мінімальною кількістю тестів. Це сприятиме виявленню дефектів у критичних точках логіки введення та перевірки користувацьких даних.

Вибір зазначених підходів зумовлений специфікою web-застосунку, який має орієнтацію на кінцевого користувача, роботу через браузер та інтерактивну взаємодію з сервером. Використання методу чорного ящика в комбінації з мануальним тестуванням дозволить ефективно перевірити роботу інтерфейсу, логіку бронювання, валідацію даних, навігацію та обробку помилок у реальних сценаріях.

Таким чином, обрані техніки тестування є обґрунтованими та забезпечують повне покриття основних аспектів функціонування системи, що дозволяє гарантувати її якість, зручність та стійкість до помилкових дій користувачів.

4.3 Результати тестування

Етап тестування розпочався з верифікації вимог до системи. Було проведено перевірку на повноту, логічну узгодженість і коректність сформульованих функціональних та нефункціональних вимог. Усі описані функції мають чітке призначення, однозначну інтерпретацію та деталізовані умови реалізації. Жодна вимога не суперечить іншій, а кожна категорія функціоналу відповідає своєму контексту використання. Особливу увагу приділено обов’язковості полів у формах, правильності валідації введених даних та послідовності користувацьких дій.

Для перевірки роботи web-застосунку було створено набір тест-кейсів на основі попередньо складеного чек-листа, який охоплює ключові функції системи. У процесі тестування були реалізовані як позитивні, так і негативні сценарії з метою виявлення можливих помилок при коректному та некоректному введенні даних. Кожен тест-кейс включав покроковий опис дій, очікуваний результат та порівняння з фактичною поведінкою системи, що дозволило зафіксувати результати перевірки та оцінити стабільність і надійність розробленого програмного продукту.

Тест кейси з перевірками представлені у таблиці 4.1.

Таблиця 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
1	Відкрити web-застосунок вперше	1. Перейти за посиланням на сторінку додатку.	Відкривається вікно браузера. Користувач бачить форму для створення бронювання.	Відкривається вікно браузера. Користувач бачить форму для створення бронювання.
2	Створити бронювання без заповнення поля «Ім’я»	1. Відкрити додаток 2. Запонти поле «Дата» 3. Вибрати столик на схемі 4. Вибрати страви для замовлення 5.Натиснути на кнопку «Забронювати»	Бронювання не створиться. З’явиться повідомлення «Заповніть це поле»	Бронювання не створиться. З’явиться повідомлення «Заповніть це поле»
3	Створити бронювання без заповнення поля «Дата та час»	1. Відкрити додаток 2. Запонти поле «Ім’я» 3. Вибрати столик на схемі 4.Вибрати страви для замовлення 5.Натиснути на кнопку «Забронювати»	Бронювання не створиться. З’явиться повідомлення «Заповніть це поле»	Бронювання не створиться. З’явиться повідомлення «Заповніть це поле»

Продовження таблиці 4.1

ID	**Опис**	**Кроки**	**Очікуваний результат**	**Фактичний результат**
4	Створити бронювання без вибору місця на схемі	1. Відкрити додаток 2. Запонти поле «Ім’я» 3. Запонти поле «Дата» 4.Вибрати страви для замовлення 5.Натиснути на кнопку «Забронювати»	Бронювання не створиться.	Бронювання не створиться.
5	Створити бронювання без вибору страв	1. Відкрити додаток 2. Запонти поле «Ім’я» 3. Запонти поле «Дата» 4. Вибрати столик на схемі 5. Натиснути на кнопку «Забронювати»	Бронювання не створиться. З’явиться повідомлення «Виберіть елемент зі списку»	Бронювання не створиться. З’явиться повідомлення «Виберіть елемент зі списку»
6	Створити бронювання заповнивши усі поля	1. Відкрити додаток 2. Запонти поле «Ім’я» 3. Запонти поле «Дата» 4. Вибрати столик на схемі 5. Вибрати страви для замовлення 6. Натиснути на кнопку «Забронювати»	Бронювання ствоиться. Сторінка створить повідомлення «Бронювання збережено !».	Бронювання ствоиться. Сторінка створить повідомлення «Бронювання збережено !».

Продовження таблиці 4.1

ID	**Опис**	**Кроки**	**Очікуваний результат**	**Фактичний результат**
7	Створити бронювання заповнивши усі поля та вибравши один і тей ж столик дві години після іншого резервування.	1. Відкрити додаток 2. Запонти поле «Ім’я» 3. Запонти поле «Дата» 4. Вибрати той самий столик на схемі 5. Вибрати страви для замовлення 6. Натиснути на кнопку «Забронювати»	Бронювання не створиться. Клітинка зайнятого столику буде червоного кольру.	Бронювання не створиться. Клітинка зайнятого столику буде червоного кольру.

У результаті проведеного тестування було підтверджено відповідність роботи web-застосунку попередньо визначеним функціональним вимогам. На основі тест-кейсів було перевірено як позитивні сценарії взаємодії користувача з системою, так і типові помилки введення, які можуть призводити до збоїв або некоректного функціонування.

Усі перевірки дали очікувані результати, що свідчить про стабільність, надійність і готовність системи до практичного використання. Особливу увагу було приділено перевірці валідації форм — жодне бронювання не створюється без обов’язкових полів, неправильні дії коректно блокуються повідомленнями, а інтерфейс дає зворотний зв’язок користувачу.

Таким чином, web-застосунок демонструє коректну обробку користувацьких дій, ефективне управління даними та забезпечує зручність взаємодії завдяки продуманому інтерфейсу.

ВИСНОВОК

У дипломній роботі було розглянуто теоретичні засади, проведено аналіз предметної області, визначено вимоги до інформаційної системи та здійснено розробку вебзастосунку на основі сучасних технологій.

У процесі дослідження було проаналізовано сучасний стан автоматизації процесів у закладах ресторанного типу, вивчено програмні рішення конкурентного ринку, сформульовано функціональні та нефункціональні вимоги до програмного продукту. В результаті проєктування запропоновано архітектурну модель системи, яка включає клієнтську частину (React.js), серверну частину (Node.js з Express), базу даних MongoDB та інтерфейс взаємодії між компонентами.

На основі моделювання було реалізовано прототип системи, що забезпечує реєстрацію користувачів, пошук і бронювання столиків, оформлення замовлень, адміністрування залу та обробку запитів у реальному часі. Реалізовано перевірку доступності ресурсів, систему сповіщень, адаптивний інтерфейс, а також базові засоби захисту даних.

Результати розробки підтверджують доцільність використання обраного технологічного стеку для створення швидкої, масштабованої та зручної в експлуатації системи. Розроблений застосунок може бути використаний як основа для впровадження в реальні заклади ресторанного типу або масштабований відповідно до потреб конкретного підприємства.

Таким чином, мету дипломної роботи досягнуто, а поставлені завдання виконано. Робота має практичну цінність і може бути використана в подальших дослідженнях або впроваджена у виробниче середовище.

Диплом пройшов апробацію на наступних конференціях :

Богачкін Ф.О. Шахматов І.О. Система з підтримки бронювання столиків та оформлення замовлень у ресторані // Формування сучасної науки: методика та практика: матеріали VII Всеукр. студент. наук. конф. (Київ, 23 трав. 2025 р.). – Київ: UKRLOGOS Group, 2025. – С. 482–483.

Перелік посилань

1. Carter R. Leveraging hybrid cloud solutions for data management. *Cloud technology review*. 2023. Vol. 9. P. 45–60.
2. Chowdhury R., Amir Hossain Fahad M., Hasan -Al –Shabbir. Evolution and future trends in web development: a comprehensive review. *Pathfinder of research*. 2022. Vol. 3. P. 31–39. URL: https://doi.org/10.69937/pf.por.3.1.35.
3. Cuba C. Building a ReactJS App with GraphQL Middleware and NodeJS-MongoDB Backend. *Medium*. URL: https://carloscuba014.medium.com/building-a-reactjs-app-with-graphql-middleware-and-nodejs-mongodb-backend-b19dbce9ddcb (date of access: 15.05.2025).
4. Fauzan R., Ice Krisnahati I., Dinda Nurwibawa B. A systematic literature review on progressive web application practice and challenges. *The journal of technology and science*. 2022. P. 44–55. URL: https://doi.org/10.12962/j20882033.v33i1.13904.
5. Fortunato D., Bernardino J. Progressive web apps: an alternative to the native mobile apps. *2018 13th iberian conference on information systems and technologies (CISTI)*. 2018. P. 1–6.
6. Gambhir A., Raj J. Analysis of cache in service worker and performance scoring of progressive web application. *2018 international conference on advances in computing and communication engineering (ICACCE)*. 2018. P. 294–299.
7. Garousi V. Web application testing: a systematic literature review. *Journal of systems and software*. 2014. Vol. 91. P. 174–201. URL: https://doi.org/10.1016/j.jss.2014.01.010 (date of access: 15.05.2025).
8. Harness Team. Introduction to building a CRUD API with node.js and express. *Harness.io*. URL: https://www.harness.io/blog/introduction-to-building-a-crud-api-with-node-js-and-express (date of access: 15.05.2025).
9. Jak S., Kolbe L., Hongli Li H. Meta-analytic structural equation modeling made easy: A tutorial and web application for one-stage MASEM. *Research synthesis methods*. 2021. Vol. 12, no. 5. P. 590–606. URL: https://doi.org/10.1002/jrsm.1498 (date of access: 09.05.2025).
10. Jonathan R. Development of front-end web applications utilizing single page application framework and react.js library. *Nternational journal software engineering and computer science*. 2023. Vol. 3, no. 3. P. 529–536. URL: https://doi.org/10.35870/ijsecs.v3i3.1943.

11. Ricks N. Building restful apis with node.js and express. *Medium*. URL: https://medium.com/weekly-webtips/building-restful-apis-with-node-js-and-express-a9f648219f5b (date of access: 18.05.2025).
12. Shah A. Build modern web apps with node.js & react – A step-by-step guide. *Techify*. URL: https://techifysolutions.com/blog/building-modern-web-apps/?utm_source (date of access: 16.05.2025).
13. Team P. Full Stack Development with React, Node.js, and MongoDB. *Programmers.io*. URL: https://programmers.io/blog/fullstack-development-react-nodejs-mongodb/?utm_source (date of access: 13.05.2025).
14. Wanyoike M. Build a Node.js CRUD App Using React and FeathersJS â SitePoint. *SitePoint*. URL: https://www.sitepoint.com/crud-app-node-react-feathersjs/?utm_source (date of access: 15.05.2025).

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

«Інформаційна система для автоматизації ресторанного бізнесу»

Виконав студент 4 курсу
групи ІСД-41
Богачкін Федір Олексійович
Керівник роботи: Викладач кафедри ІСТ
Шахматов Іван Олександрович

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета дослідження: покращення процесу інтерактивного бронювання столиків у закладах громадського харчування шляхом створення web-застосунку з можливістю попереднього замовлення страв.

Об’єкт дослідження: процес автоматизації управління замовленнями та резервуванням місць у закладах ресторанного типу.

Предмет дослідження: веб-застосунок для бронювання столиків та оформлення замовлень.

ТЕХНІЧНІ ЗАВДАННЯ

1. Проаналізувати сферу онлайн-бронювання столиків у ресторанах.
2. Порівняти популярні сервіси.
3. Обрати технології для реалізації.
4. Розробити web-застосунок з функціями бронювання та замовлення.
5. Протестувати застосунок у браузерах на функціональність та зручність.

ФУНКЦІОНАЛЬНІ ВИМОГИ

Користувач повинен мати можливість:

1. Ввести своє ім'я;
2. Обрати дату та час бронювання;
3. Обрати стіл на інтерактивній схемі залу;
4. Вибрати страви з меню та вказати їх кількість;
5. Надіслати заявку на бронювання. Перевірка доступності столика;
6. Перед підтвердженням бронювання система перевіряє, чи не зайнятий стіл на вказану дату та час;
7. Візуальне відображення схеми залу.

НЕФУНКЦІОНАЛЬНІ ВИМОГИ

1. Продуктивність: веб-застосунок повинен обробляти запит бронювання протягом не більше ніж 2 секунд при середньому навантаженні.
2. Масштабованість: архітектура має бути готовою до розширення — наприклад, для додавання реєстрації користувачів, адміністрування або підтримки кількох ресторанів.
3. Зручність користування: інтерфейс повинен бути інтуїтивно зрозумілим;
4. Безпека: дані користувача не передаються третім особам;

5

АНАЛОГИ

tock

RESY

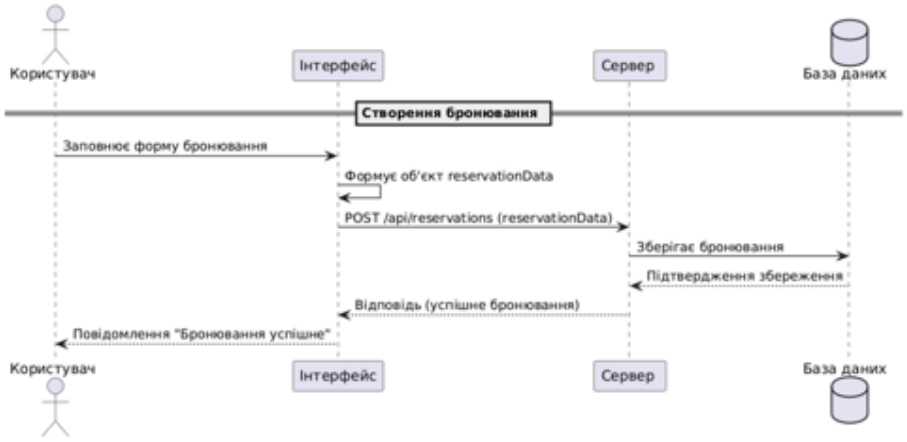
Критерій	Tock	Resy	Restarurant Reservation
Інтерактивна схема залу	-	-	+
Можливість замовлення страв	+	-	+
Ціна використання	Платна підписка	Платна підписка	Бескоштовна
Доступність для користувачів	Вимагає створення акаунта	Потрібен акаунт Resy або інтеграція з app	Простий інтерфейс, не вимагає акаунта

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

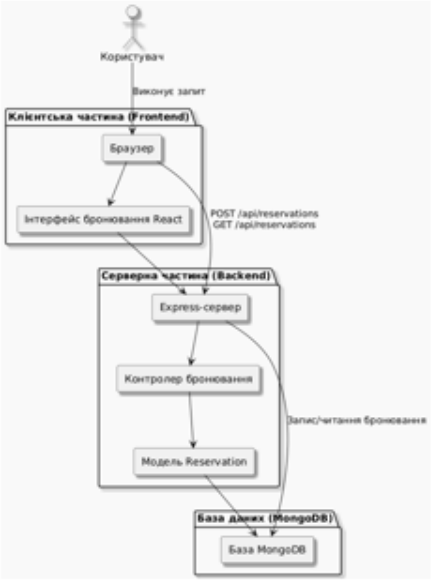
7

ДІАГРАМА ПОСЛІДОВНОСТІ СИСТЕМИ



8

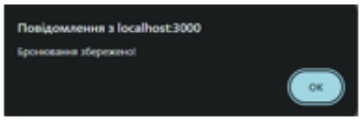
СХЕМА РОБОТИ WEB-ЗАСТОСУНКУ



9

ІНТЕРФЕЙС РОЗРОБЛЕНОГО ЗАСТОСУНКУ

Вигляд сторінки в браузері



Повідомлення про успішне бронювання

Вигляд сторінки на телефоні

Вигляд сторінки з незаповненим полем

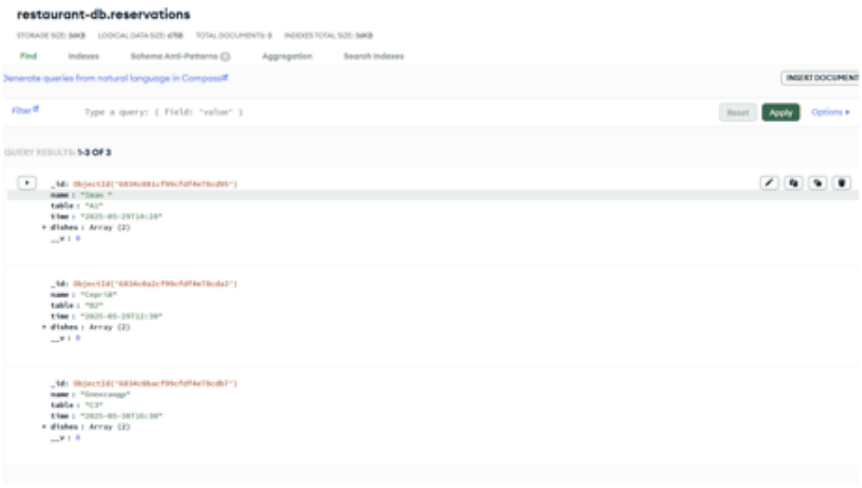
10

ВИСНОВКИ

1. Було проаналізовано предметну область онлайн-бронювання у сфері ресторанного бізнесу, що дозволило чітко сформулювати вимоги до функціоналу веб-застосунку.
2. Проведено порівняльний аналіз існуючих сервісів (Tock, Resy тощо), що допомогло визначити ключові переваги та недоліки конкурентів.
3. На основі аналізу обрано оптимальні технології розробки: React для frontend, Node.js + Express для backend, MongoDB для зберігання даних.
4. Розроблено та протестовано працездатний веб-додаток у браузерному середовищі, що відповідає функціональним та нефункціональним вимогам.

12

ВИГЛЯД ДАНИХ В MONGODB



11

АПРОБАЦІЯ

Богачкін Ф. О. Система з підтримки бронювання столиків та оформлення замовлень у ресторані // Формування сучасної науки: методика та практика: матеріали VII Всеукр. студент. наук. конф. (Київ, 23 трав. 2025 р.). – Київ: UKRLOGOS Group, 2025. – С. 482–483.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б

Frontend

Reservation js

```
import React, { useState, useEffect } from 'react';
import './ReservationForm.css';


const ReservationForm = () => {
  const [name, setName] = useState('');
  const [table, setTable] = useState('');
  const [time, setTime] = useState('');
  const [orders, setOrders] = useState([{ dish: '', quantity: 1 }]);
  const [busyTables, setBusyTables] = useState([]);


  const tablePositions = [
    { id: 'A1', x: 20, y: 20 },
    { id: 'A2', x: 100, y: 20 },
    { id: 'A3', x: 180, y: 20 },
    { id: 'B1', x: 20, y: 100 },
    { id: 'B2', x: 100, y: 100 },
    { id: 'B3', x: 180, y: 100 },
    { id: 'C1', x: 20, y: 180 },
    { id: 'C2', x: 100, y: 180 },
    { id: 'C3', x: 180, y: 180 },
  ];

  useEffect(() => {
    if (time) {

fetch(`http://localhost:5000/api/reservations?date=${encodeURIComponent(time)}`)
        .then(res => res.json())
        .then(data => {
          const taken = data.map(r => r.table);
          setBusyTables(taken);
        });
    }
  }, [time]);


  const handleOrderChange = (index, field, value) => {
    const updatedOrders = [...orders];
    updatedOrders[index][field] = value;
    setOrders(updatedOrders);
  };


  const addDish = () => {
    setOrders([...orders, { dish: '', quantity: 1 }]);
  };


  const handleTableClick = (id) => {
    if (!busyTables.includes(id)) {
      setTable(id);
```

```
  }
 };

 const handleSubmit = async (e) => {
  e.preventDefault();

  const data = {
   name,
   table,
   time,
   dishes: orders,
  };

  const response = await
fetch('http://localhost:5000/api/reservations', {
   method: 'POST',
   headers: { 'Content-Type': 'application/json' },
   body: JSON.stringify(data),
  });

  const result = await response.json();
  console.log(result);
  alert('Бронювання збережено!');
  setName('');
  setTable('');
  setTime('');
  setOrders([{ dish: '', quantity: 1 }]);
 };

 return (
  <form className="reservation-form"
onSubmit={handleSubmit}>
   <h2>Бронювання столика</h2>

   <input
    type="text"
    placeholder="Ім'я"
    value={name}
    onChange={(e) => setName(e.target.value)}
    required
   />

   <input
    type="datetime-local"
    value={time}
    onChange={(e) => setTime(e.target.value)}
    required
   />

   <h3>Схема залу</h3>
   <svg width="280" height="280" style={{
border: '1px solid #ccc', marginBottom: '10px' }}>
    <text x="10" y="10" fontSize="12">
Вікна</text>
```

```
      <text x="10" y="270" fontSize="12">🚪 Вхід</text>
      {tablePositions.map(({ id, x, y }) => {
        const isBusy = busyTables.includes(id);
        const isSelected = table === id;
        return (
          <g key={id} onClick={() => handleTableClick(id)} style={{ cursor: isBusy ? 'not-allowed' : 'pointer' }}>
            <rect
              x={x}
              y={y}
              width="60"
              height="60"
              fill={isBusy ? '#e57373' : isSelected ? '#4caf50' : '#ccc'}
              stroke="#555"
              rx="10"
            />
            <text x={x + 20} y={y + 35} fontSize="14" fill="#000">{id}</text>
          </g>
        );
      })}
    </svg>

    {table && <p>Вибраний столик: <strong>{table}</strong></p>}
```

```
    <h3>Замовлення страв</h3>
    {orders.map((order, index) => (
      <div key={index} className="dish-row">
        <select
          value={order.dish}
          onChange={(e) => handleOrderChange(index, 'dish', e.target.value)}
          required
        >
          <option value="">Оберіть страву</option>
          <option value="Піца">Піца</option>
          <option value="Салат">Салат</option>
          <option value="Суп">Суп</option>
        </select>

        <input
          type="number"
          min="1"
          value={order.quantity}
          onChange={(e) => handleOrderChange(index, 'quantity', e.target.value)}
          required
        />
      </div>
    ))}
```

```
    <button type="button"
onClick={addDish}>➕ Додати ще
страву</button>
    <br />
    <button type="submit" disabled={!table}>✅
Забронювати</button>
  </form>
 );
};

export default ReservationForm;
```

Reservation css

```
.reservation-form {
 max-width: 500px;
 margin: 30px auto;
 padding: 20px 25px;
 background-color: #f9f9f9;
 border-radius: 15px;
 box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
 font-family: 'Segoe UI', sans-serif;
}

.reservation-form h2,
.reservation-form h3 {
 text-align: center;
 margin-bottom: 15px;
 color: #333;
}

.reservation-form input,
.reservation-form select {
 width: 100%;
 padding: 10px;
 margin: 8px 0 15px;
 border: 1px solid #ccc;
 border-radius: 8px;
 font-size: 14px;
 box-sizing: border-box;
}

.reservation-form svg {
 display: block;
 margin: 0 auto 15px;
 background-color: #fff;
 border-radius: 10px;
}

.reservation-form .dish-row {
 display: flex;
 gap: 10px;
 margin-bottom: 10px;
}

.reservation-form .dish-row select,
```

```
.reservation-form .dish-row input {
  flex: 1;
}

.reservation-form button {
  display: block;
  width: 100%;
  background-color: #4caf50;
  color: white;
  font-weight: bold;
  padding: 10px;
  margin-top: 10px;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background 0.3s ease;
}

.reservation-form button:hover:not(:disabled) {
  background-color: #45a049;
}

.reservation-form button:disabled {
  background-color: #bbb;
  cursor: not-allowed;
}

.reservation-form p {
  text-align: center;
  font-weight: 500;
  margin-bottom: 10px;
  color: #444;
}
```

Backend

Indedex.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

// MongoDB
mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('✅ Підключено до MongoDB'))
  .catch(err => console.error('❌ Помилка з MongoDB:', err));
```

```
// Routes
const reservationRoutes = require('./routes/menuRoutes');
app.use(reservationRoutes);

// Тестовий маршрут
app.get('/', (req, res) => {
  res.send('Сервер працює');
});

app.listen(PORT, () => {
  console.log(` Сервер запущено на http://localhost:${PORT}`);
});
const express = require('express');
const router = express.Router();
const Reservation = require('../models/Reservation');
router.post('/api/reservations', async (req, res) => {
  try {
    const { table, time } = req.body;

    const existing = await Reservation.findOne({
      table,
      time: { $lte: time }
    });
    if (existing) {
      return res.status(409).json({ message: 'Цей столик вже зарезервовано на обраний час або пізніше' });
    }

    const reservation = new Reservation(req.body);
    await reservation.save();

    res.status(201).json({ message: 'Успішно збережено!' });
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
});
router.get('/api/reservations', async (req, res) => {
  try {
    const reservations = await Reservation.find();
    res.json(reservations);
  } catch (err) {
    res.status(500).json({ error: 'Помилка при отриманні даних' });
  } });
module.exports = router;
```