Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра комп’ютерних наук

Пояснювальна записка

до магістерської кваліфікаційної роботи
на тему:

Інформаційна технологія бронювання місць в ресторанах: серверна частина

Виконав: студентка 1 курсу, групи 1КН-19м
Спеціальності 122 –
«Комп’ютерні науки»

Новіцький К.В.
(прізвище та ініціали)

Керівник ст. вик. Петришин С. І.
(прізвище та ініціали)

Рецензент д.т.н., проф.каф.ПЗ Ліщинська Л.Б.
(прізвище та ініціали)

Вінниця - 2020 року

АНОТАЦІЯ

Робота присвячена створенню інформаційної технології розробки серверної частини онлайн-сервісу для бронювання місць в ресторані. Метою магістерської роботи є покращення роботи додатку за рахунок зменшення часу для передачі даних від сервера до клієнта і навпаки.

Проаналізовано переваги та недоліки кожного з них, створений список вимог. Розроблена безспосередньо сама серверна частина та проведена інтергація з клієнтською частиною.

Ключові слова: база даних, Mongo db, серверна частина, Node.js.

ABSTRACT

The work is devoted to the creation of information technology for the development of the server part of the restaurant reservation service. The purpose of the master's thesis is to improve the application by reducing the time for data transfer from the server to the client and vice versa.

The advantages and disadvantages of each of them are analyzed, the list of requirements is created. The server part itself was developed and integration with the client part was performed.

Keywords: database, Mongo db, server part, Node.js.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Кількість нових ресторанів у всьому світі збільшується у геометричній прогресії. Кожен ресторан має якусь свою особливість, що виділяє його з поміж інших закладів такого ж типу. Сьогоднішній світ не стоїть на місці, а навпаки, активно розвивається в плані інформаційних технологій. Як вже згадувалось, що кількість ресторанів збільшується, а це означає, що збільшується і кількість паперової документації. Надмірність паперів призвела до того, що потрібно зберігати інформацію в електронному виді та уміло нею маніпулювати.

Розвиток інформаційних технологій дає нові можливості для автоматизованої роботи з даними.

Зростання інформаційних потоків призвело до появи ряду нових задач, одна з найважливіших полягає в швидкому отриманні потрібних даних з великого обсягу інформації. Для зберігання інформації використовуються бази даних, а для їх обробки системи керування базами даних.

Завдяки досягненням в області штучного інтелекту з'являються системи, що базуються на використанні знань. Систему, яка забезпечує створення, ведення і застосування баз знань, можна розглядати як інструментальну систему або прикладну систему з конкретною прикладною базою знань.

Але цією інформацією потрібно вміти правильно маніпулювати. І тут на допомогу приходять різного плану веб-сервіси та додатки, які дають змогу людині виконувати якісь дії на цими даними

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних

інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання дослідження. Метою даної роботи є покращення роботи додатку за рахунок зменшення часу для передачі даних від сервера до клієнта. Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати існуючі технології, методи і моделі розробки серверної частини онлайн-сервісів;

– сформулювати вимоги до програмного додатку для бронювання місць в ресторані та розробити ТЗ.

– спроектувати серверну частину програмного додатку для бронювання місць в ресторані;

– Провести тестування додатку та перевірити швидкість запитів до сервера.

Об’єкт дослідження – процес розробки серверної частини онлайн-сервісу для бронювання місць в ресторані.

Предмет дослідження – засоби розробки серверної частини онлайн-сервісу для бронювання місць в ресторані.

Методи дослідження. У роботі використані наступні методи наукових досліджень: методи моделювання та розробки інтерфейсу користувача, методи моделювання архітектури онлайн-сервісу.

Наукова новизна одержаних результатів полягає у тому, що:

Удосконалено інформаційну технологію бронювання столиків в ресторані за рахунок підвищення швидкості передачі даних між клієнтом та сервером. Результатом роботи буде онлайн-сервіс що дасть змогу бронювати столики, не відволікаючись на дзвінки до адміністрації ресторану, за допомогою різних девайсів.

Практичне значення одержаних результатів полягає у тому, що розроблено онлайн-сервіс для бронювання місць в ресторані, який допоможе бронювати столик в режимі онлайн з різних девайсів.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням різноманітних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ ОНЛАЙН-СЕРВІСУ

1.1 Аналіз проблеми розробки серверної частини онлайн-сервісу

Веб-додаток складається з клієнтської і серверної частин, тим самим реалізуючи технологію «клієнт-сервер».

Клієнтська частина реалізує користувальницький інтерфейс, формує запити до сервера і обробляє відповіді від нього.

Серверна частина отримує запит від клієнта, виконує обчислення, після цього формує веб-сторінку і відправляє її клієнту через мережу з використанням протоколу HTTP.

В сучасному світі існує цілі низка мов програмування, які дозволяють писати серверну частину. Кожна мова програмування є унікальною, а це означає, що кожна мова має як свої недоліки, так і переваги.

PHP (рекурсивний акронім словосполучення PHP: Hypertext Preprocessor) - це поширена мова програмування загального призначення з відкритим вихідним кодом. PHP спеціально сконструйований для веб-розробок та його код може впроваджуватися безпосередньо в HTML.

PHP відрізняється від JavaScript тим, що PHP-скрипти виконуються на сервері і генерують HTML, який надсилається клієнту. Якби у вас на сервері був розміщений скрипт, подібний вищенаведеного, клієнт отримав би тільки результат його виконання, але не зміг би з'ясувати, який саме код його виповнився. Ви навіть можете налаштувати свій сервер таким чином, щоб звичайні HTML-файли оброблялися процесором PHP, так що клієнти навіть не зможуть дізнатися, чи отримують вони звичайний HTML-файл або результат виконання скрипта.

З переваг PHP можна відмітити те, що розробка на цій мові є доволі швидкою, завдяки своїм фреймворкам (Symfony, CodeIgniter, Laravel, Joomla, WordPress). Недоліком є те, що у сучасному світі є багато різних засобів написання серверної частини для web-додатку, які працюють швидше, ніж PHP.

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то Node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

До переваг можна віднести можливість застосовувати одну мову на клієнті і сервері. Якщо програміст володіє JavaScript, йому буде легше вивчити "надбудову", ніж кардинально іншу технологію. Також створення робочого прототипу не відніме багато часу. Перший етап, коли програміст формує кістяк майбутнього продукту, проходить дуже швидко. Якщо архітектура добре продумана, то в подальшому труднощів з тим, щоб розширювати сайт на Node.js теж не з'явиться [1]. До мінусів можна віднести те, що дуже великий проект на Node.js написати можливо, але важко.

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду.

До плюсів даної мови програмування можна віднести те, що вона є доволі простою та універсальною, але в той же час повільною в порівнянні з іншими мовами, що і є її недоліком [2].

Node.js дуже гарно працює в парі з такою базою даних як MongoDB. До них є багато бібліотек, що полегшують та прискорюють розробку.

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

З переваг можно відмітити, що у MongoDB є вбудовані засоби із забезпечення шардінгу, комбінуючи який з реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин. Неділоком є те, що MongoDB не завжди підходить на роль заміни MySQL. Особливо, якщо ваш web-проект не передбачає виділення під MongoDB двох і більше окремих серверів. Також актуальною є проблема з безпекою [3].

1.2 Аналіз особливостей розробки серверної частини онлайн-сервісу

Веб браузери взаємодіють з веб-серверами за допомогою гіпертекстового транспортного протоколу (HTTP). Коли ви натискаєте на посилання на веб-сторінці, заповнюєте форму або запускаєте пошук, HTTP запит відправляється з вашого браузера на цільовий сервер.

Запит включає в себе URL, який визначає використаний ресурс, метод, який визначає необхідну дію (наприклад, отримати, видалити або опублікувати ресурс) і може включати додаткову інформацію, закодовану в параметрах URL

(пари поле-значення, вставлені як рядок запиту), як POST запит (дані, відправлені методом HTTP POST), або в кукі-файлах.

Веб сервери очікують повідомлень з клієнтськими запитами, обробляють їх по прибуттю і відповідають веб-браузеру за допомогою відповідного HTTP повідомлення. Відповідь містить рядок стану, який показує, чи був запит успішним, чи ні (наприклад, "HTTP / 1.1 200 OK" в разі успіху).

Тіло успішної відповіді на запит може містити запитувані дані (наприклад, нову HTML сторінку, або зображення, тощо), який може відображатися через веб-браузер.

Схема на рисунку 1.1 показує базову архітектуру веб-сервера для статичного сайту (статичний сайт - це той, який повертає один і той же жорстко закодоване вміст з сервера щоразу, коли запитується конкретний ресурс). Коли користувач хоче перейти на сторінку, браузер відправляє HTTP-запит «GET» із зазначенням його URL.

Сервер витягає запитаний документ зі своєї файлової системи і повертає HTTP-відповідь, що містить документ і успішний статус. Якщо файл не може бути витягнутий з яких-небудь причин, повертається статус помилки [4]. На рисунку 1.1 зображено схему статичного сайту.

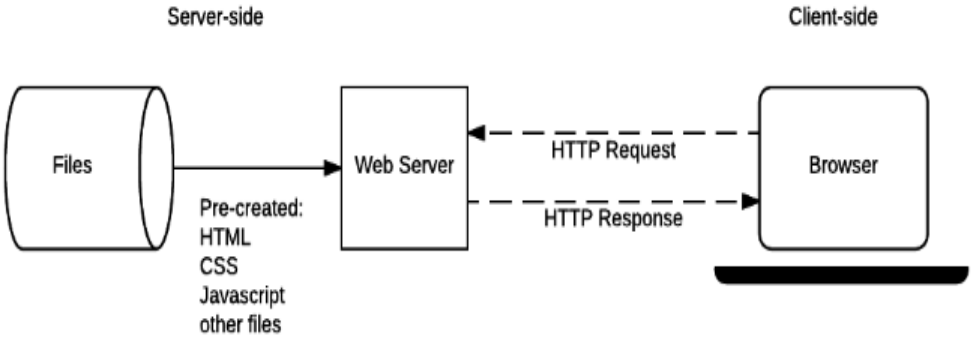

Рисунок 1.1 – Схема статичного сайту

Динамічний веб-сайт – це той, де частина вмісту відповіді генерується динамічно тільки при необхідності. На динамічному веб-сайті HTML-сторінки зазвичай створюються шляхом вставки даних з бази даних в наповнювачі в HTML-шаблони (це набагато більш ефективний спосіб зберігання великої кількості контенту, ніж використання статичних сайтів).

Динамічний сайт може повертати різні дані для URL-адреси на основі інформації, наданої користувачем або збереженими настройками, і може виконувати інші операції як частину повернення відповіді (наприклад, відправлення повідомлень).

Велика частина коду для підтримки динамічного веб-сайту повинна виконуватися на сервері. Створення цього коду відомо, як «програмування серверної частини» (або іноді «програмування бекенду»).

Схема на рисунку 1.2 показує просту архітектуру динамічного сайту. Як і на попередній схемі, браузери відправляють HTTP-запити на сервер, потім сервер обробляє запити і повертає відповідні HTTP-відповіді.

Запити статичних ресурсів обробляються так само, як і для статичних сайтів (статичні ресурси – це будь-які файли, які не змінюються, зазвичай це: CSS, JavaScript, зображення, попередньо створені PDF-файли та інше) [5]. На рисунку 1.2 зображено схему динамічного сайту.

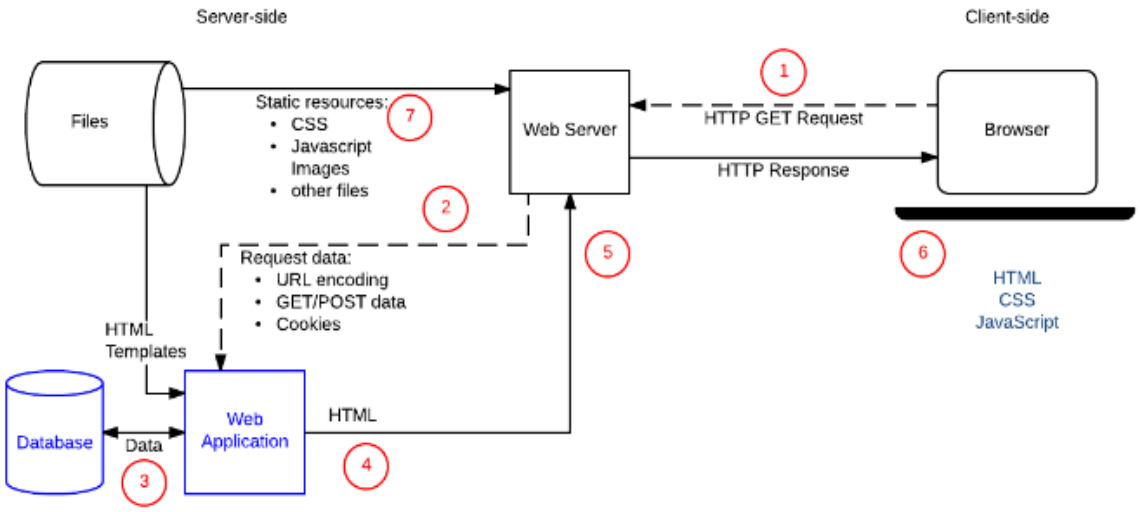

Рисунок 1.2 – Схема динамічного сайту

Запити динамічних даних відправляються (2) в код серверної частини. Для «динамічних запитів» сервер інтерпретує запит, читає необхідну інформацію з бази даних (3), комбінує витягнуті дані з шаблонами HTML і повертає відповідь, що містить згенерований HTML (5, 6).

Основні переваги створення серверної частини.

- Програмування серверної частини дуже корисно оскільки дозволяє ефективно доставляти інформацію, складену для індивідуальних користувачів і таким чином створювати набагато кращий досвід використання.

Компанії, такі як Amazon, використовують програмування серверної частини для побудови дослідницьких результатів для товарів, формування цільового пропозиції, заснованого на перевагах клієнта і попередні покупки, спрощення замовлень і т.д. Банки використовують програмування серверної частини, щоб зберігати облікову інформацію і дозволяти тільки авторизованим користувачам переглядати і здійснювати транзакції. Інші сервіси, такі як Facebook, Twitter, Instagram і Wikipedia використовують бекенд, щоб виділяти, поширювати і контролювати доступ до цікавого контенту.

Уявіть, скільки товарів є на Amazon і уявіть, скільки постів було написано на Facebook? Створення статичної сторінки для кожного товару або поста було б абсолютно неефективним.

- Програмування серверної частини дозволяє замість цього зберігати інформацію в базі даних і динамічно створювати, і повертати HTML і інші типи файлів (наприклад, PDF, зображення, і т.д.). Також є можливість просто повернути дані (JSON, XML, і т.д.) для відображення використовуючи підходящий фреймворк клієнтської частини (це зменшує завантаження процесора на сервері і кількість переданих даних.

Сервер не обмежений у відправці інформації з баз даних і може замість цього повертати результат інструментів програмного забезпечення або дані з сервісів комунікації. Контент навіть може бути цільовим щодо влаштування клієнта, який його отримує.

- Сервери можуть зберігати і використовувати інформацію про клієнтів щоб поставляти зручний і зроблений індивідуально для користувача досвід взаємодії. Наприклад, багато сайтів зберігають дані кредитних карт, щоб не потрібно було вводити їх повторно. Сайти на зразок Google Maps використовують домашню адресу і поточне місце розташування для надання інформації про маршрут і виділяючи місцеві установи в результатах пошуку.

Більш глибокий аналіз звичок користувача може бути використаний для передбачення їхніх інтересів і подальших налаштувань відповідей і повідомлень, наприклад, забезпечення списку раніше відвіданих популярних місць, які ви, можливо, захочете знайти на карті.

- Програмування серверної частини дозволяє сайтам обмежувати доступ авторизованим користувачам і надавати тільки ту інформацію, яку користувачеві дозволено бачити. Реальним прикладом є соціальні мережі.
- Програмування серверної частини дозволяє розробникам використовувати сесії - спочатку це механізм, що дозволяє сервера зберігати

інформацію про поточного користувача сайту і відправляти різні відповіді, засновані на цій інформації. Це дозволяє, наприклад, сайту знати, що користувач був попередньо авторизований і відображає посилання на їх адресу електронної пошти або історію замовлень, або можливо зберегти прогрес простої гри, так щоб користувач міг при наступному заході на сайт продовжити звідти, де він закінчив.

- Сервери можуть відправляти загальні або призначені для користувача повідомлення безпосередньо через сайт або по електронній пошті, смс, миттєві повідомлення, в реальному часі або інші засоби зв'язку.
- Веб-сайт може збирати багато даних про своїх користувачів: що вони шукають в пошуку, що вони купують, що вони рекомендують, як довго вони залишаються на кожній сторінці. Програмування серверної частини може бути використано, щоб удосконалити відповіді, засновані на аналізі цих даних [6].

1.3 Аналіз переваг та недоліків використання онлайн-сервісів для бронювання місць в ресторані

У США значна частина відвідувачів ресторанів знайомі з сервісами онлайн-бронювання - за даними статистики, більше 37% гостей хоча б пару раз в своєму житті ними користувалися [2].

Ринок подібних додатків в Штатах також вельми розвинений, показником чого, наприклад є угоди з купівлі подібних проектів більшими IT- компаніями. Наприклад, не так давно індійський сервіс пошуку ресторанів Zomato за $ 52 млн викупив NexTable, американську онлайн-платформу бронювання столиків в ресторанах. Це було зроблено для того, щоб скласти конкуренцію сервісам OpenTable від Priceline і SeatMe від Yelp.

В Україні і країнах СНД подібні інструменти поки не користуються такою серйозною популярністю. Для чого ж потрібно ресторанам співпрацювати з онлайн-сервісами бронювання столиків?

1. Залучення додаткової аудиторії

Найбільш очевидна мотивація власників ресторанів, що штовхає їх на співпрацю з системами онлайн-бронювання - це надія на те, що з їх допомогою закладу вдасться залучити нових відвідувачів. Кількість подібних сервісів досить велике навіть на СНГ ринку.

Подібна конкуренція змушує такі проекти уважніше ставитися до вимог ресторанів - наприклад, додавати можливості перегляду аналітичної інформації з бронювання. При цьому такі стартапи часто мають власний маркетинговий бюджет на залучення аудиторії, що дозволяє ресторанам розраховувати на те, що про них дізнаються користувачі, які раніше не чули про даний конкретний заклад.

2. Заповнення залу в непопулярні годинник

Другий популярний спосіб використання сервісів бронювання - це залучення аудиторії для заповнення закладу в так звані «тихі» годинн, коли велика частина столиків пустує. За допомогою невеликих знижок або компліментів від шеф-кухаря, які пропонуються користувачам таких сервісів, ресторан стимулює їх до здійснення бронювання.

В такому випадку, бізнес практично нічого не втрачає - якщо користувач потім не прийде, це не призведе до збитків, оскільки відвідувачів в цей час і так майже немає. А ось якщо він прийде і щось замовить, то заклад відразу отримає додатковий дохід.

3. Нові моделі монетизації

У тих же США конкуренція на ринку онлайн-бронювання сприяє розвитку нових моделей монетизації, які в майбутньому можуть перекочувати і на інші ринки. Приклад - так звані «квитки в ресторан», які продають сервіси типу Tock.

Якщо коротко, працює це так - все схоже на покупку квитка на концерт: користувач вибирає дату і час, ресторан, а потім оплачує все, що буде записане в замовлення. У підсумку потім клієнт просто приходить в заклад і отримує те, за що вже заплатив.

4. Зниження ризиків

Поширена проблема при бронюванні - клієнт, «забиває» столик на конкретний час, а потім просто не приходить, а ресторан зазнає збитків. Деякі онлайн-сервіси допомагають звести подібні проблеми до мінімуму - наприклад, при використанні сервісів на зразок Tock клієнт платить заздалегідь, а можливість потім «перепродати» бронювання обмежена, а це знижує ймовірність того, що гість в результаті не прийде.

5. Робота з власною аудиторією на сайті

Крім роботи зі сторонніми сервісами бронювання, деякі заклади йдуть і по шляху розміщення власних модулів на сторінках свого сайту. Плюси тут очевидні - не потрібно ні з ким ділитися даними про своїх клієнтів, умови бронювання також цілком визначаються керівництвом ресторану.

Мінуси також зрозумілі - залучати аудиторію на цю форму бронювання теж потрібно самостійно, і ніхто в цій справі допомагати не буде.

Незважаючи на те, що в певних випадках - особливо, якщо мова йде про популярному заклад - ресторанам може бути вигідне використання систем онлайн-бронювання, тут є і певні ризики. Наприклад, досить велике число таких проектів являють собою простих спекулянтів, які заздалегідь бронюють столики «на себе», а потім перепродають ці броні користувачам інтернету [3].

За такою схемою працює, наприклад, сервіс Killer Rezzy - він заздалегідь бронює місця в ресторанах на популярні дати, а потім перепродає бронь з націнкою. В такому випадку ресторани не отримують ніякої додаткової вигоди (хоча клієнтам це може бути зручно, оскільки дається шанс потрапити в потрібне

заклад в намічений святковий день). Більш того, менеджмент закладу може навіть не знати, що ціна броні була завищена.

1.4 Порівняння аналогів онлайн-сервісів для бронювання місць в ресторані

На сьогоднішній день вже існують сервіси, які реалізують дану проблему. Для прикладу візьмемо онлайн-серіс «Letsbar». «Letsbar» дає можливість клієнтам обрати місця, які они планують відвідати та забронювати місце в онлайн режимі. Даний сервіс працює лише зі сторони клієнта(користувача) та має не найкращий UI та UX. «Letsbar» дозволяє переглядати детальну інормацію про місця, ресторани, паби, також переглядати фото. Головною особливістю даного сервісу є можливість для користувачів бачити розташування столиків у залі.

Плюси:

– моделювання залу;

– можливість перегляду детальної інформації;

– пріоритетність завдань.

Мінуси:

– працює лише з клієнтами, тобто для власника ресторану немає можливості додати свій ресторан самотужки;

– немає реалізованого функціоналу для показу рекомендацій користувачу;

– no user friendly interface.

Приклад вікна системи " Letsbar " відображений на рисунку 1.3.

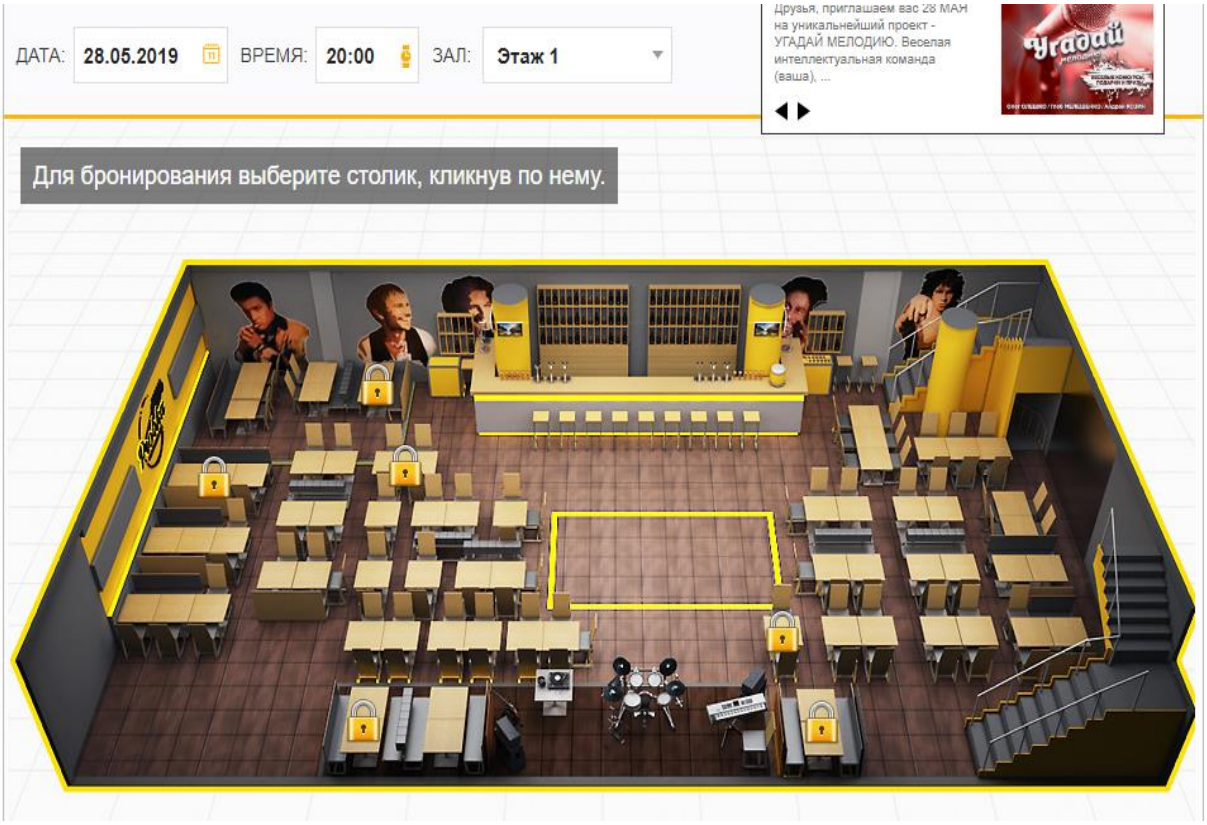

Рисунок 1.3 – Приклад вікна системи «Letsbar»

Ще одним аналогом для вирішення даної проблеми є сервіс «Murakami».

Даний сервіс розрахований лише під одну мережу ресторанів. Додаток включає в себе дуже детальну інформацію про мережу ресторанів «Murakami»: перегляд фото, відгуки, меню, опис, місцезнаходження і також дає змогу забронювати столик. Дизай виконаний краще, ніж у прикладі, який був розглянутий вище, але немає можливості подитись модель залу. Очевидно, що в даному сервісу немає можливості для власника додати самому свій ресторан, тому що це серіс тільки для однієї мережів ресторанів.

Одне з вікон сервісу «Murakami» відображено на рисунку 1.4.

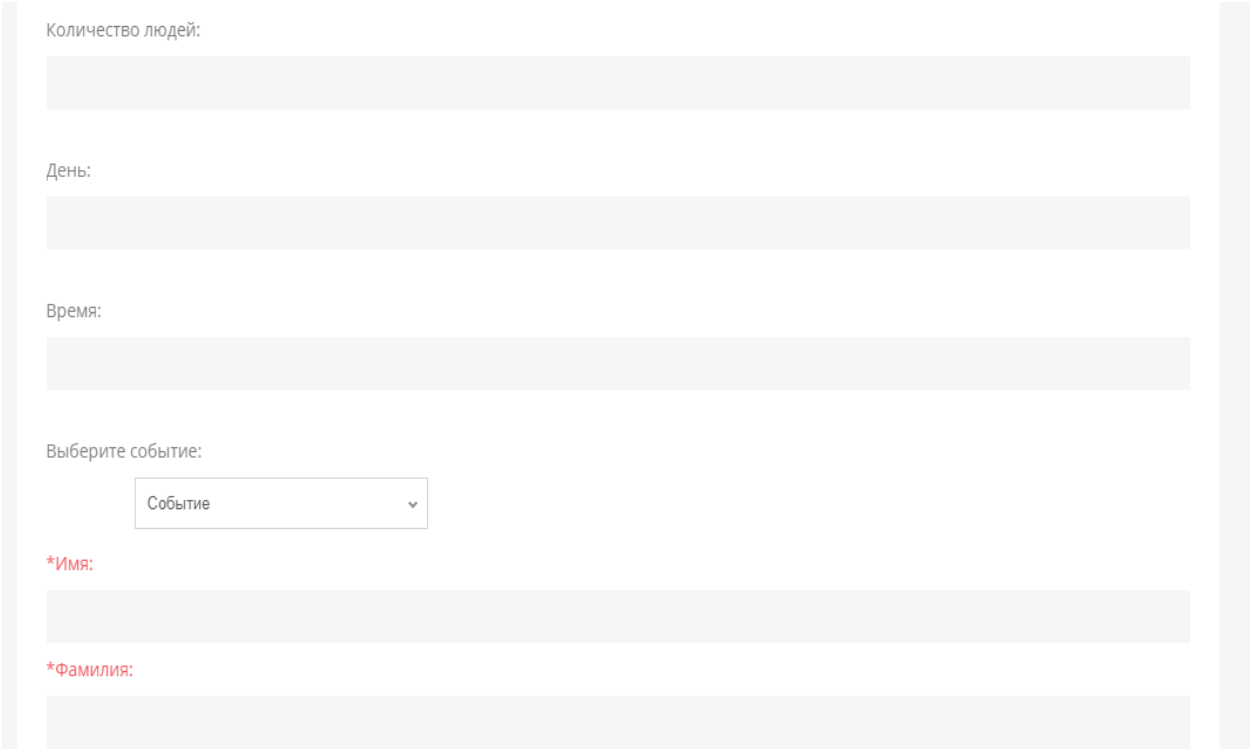

Рисунок 1.4. – Одне з вікон сервісу «Murakami»

Плюси:

– детальний опис;
– user friendly interface;
– облік часу;
– створення декількох проектів.

Мінуси:

– немає моделі залу;
– немає реалізованого функціоналу для показу рекомендацій користувачу;
– неможливо додати новий ресторан самотужки.

Таблиця 1.1 – Порівняльна характеристика аналогів

Назва	Сортування ресторанів за інтересами	Історія бронювань	Зручний інтерфейс	Можливість додати ресторан самотужки
Максимум	-	-	+	-
Murakami	-	-	+	+
Our Service	+	+	+	+

1.5 Висновок

У даному розділі було проведено основний аналіз проблем для створення серверної частини для онлайн-сервісу бронювання місць в ресторані.

Проаналізовано серверні мови програмування та надано перевагу для Node.js, тому що в парі з MongoDB отримуємо повну робочу машину для розробки, з усім наданим функціоналом.

Розглянуто основні переваги створення серверної частини, а саме:

- Зберігання та перевикористання даних
- Інкапсуляція даних
- Розділення клієнтської та серверної частин

Проведено порівняльну характеристику з аналогами, що дало змогу зрозуміти, що потрібно зробити, щоб покращити ефективність їхньої роботи.

2 РОЗРОБКА ТА ПРОЕКТУВАННЯ ОСНОВНИХ МОДУЛІВ СЕРВЕРНОЇ ЧАСТИНИ ОНЛАЙН-СЕРВІСУ ДЛЯ БРОНЮВАННЯ МІСЦЬ В РЕСТОРАНІ

2.1 Проектування серверної частини системи

Термін «сервер» може трохи збивати з пантелику тих, хто погано знайомий з галуззю, оскільки він може ставитися як до апаратного забезпечення (фізичні комп'ютери, на яких розміщені всі файли і програмне забезпечення, необхідне веб-сайтами), так і до програмного забезпечення (програмі, яка дозволяє користувачам отримувати доступ до цих файлів в Інтернеті).

Сьогодні ми поговоримо про програмну частину. Але спочатку кілька визначень. URL позначає Universal Resource Locator і складається з 3 частин: протоколу, сервера і запитуваного файлу. На рисунку 2.1 зображено графічне пояснення URL.

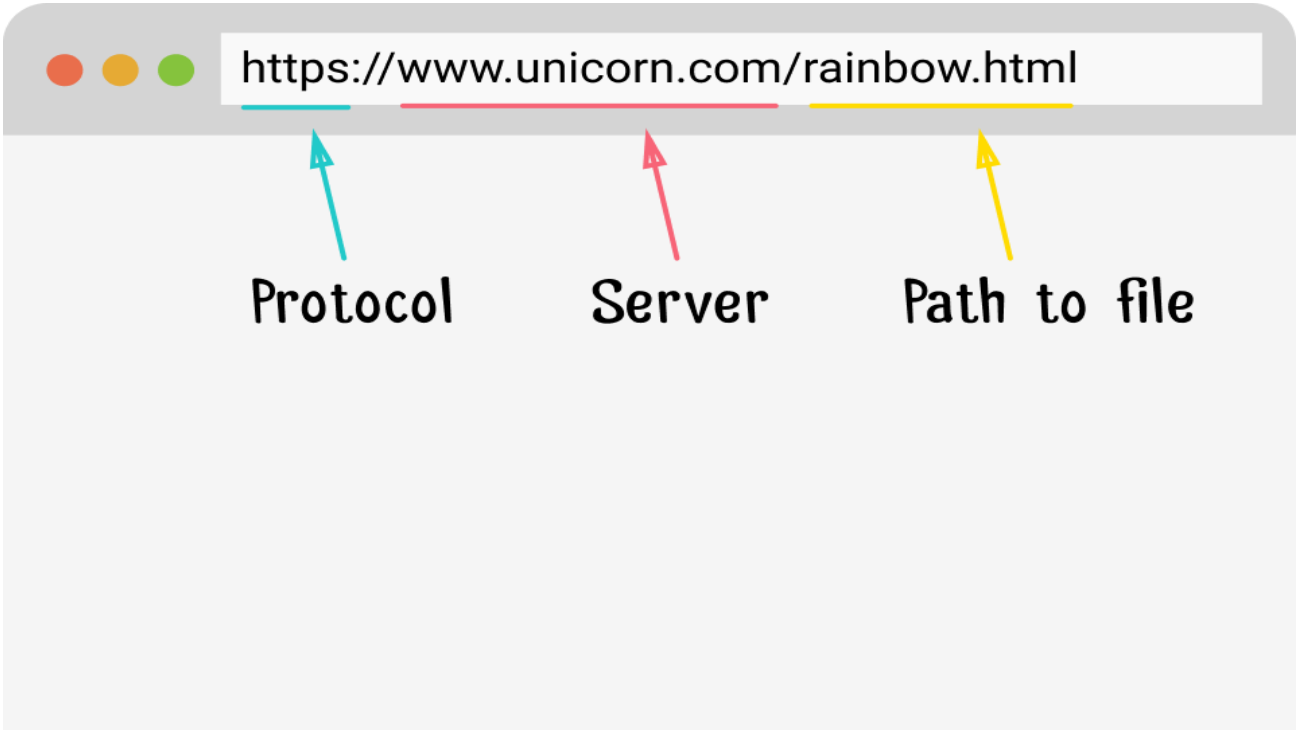

Рисунок 2.1 – Графічне пояснення URL

Протокол HTTP визначає кілька методів, які браузер може використовувати, щоб попросити сервер виконати купу різних дій, найбільш поширеними з яких є GET і POST. Коли користувач клацає посилання або вводить URL-адресу в адресний рядок, браузер відправляє серверу GET-запит на отримання ресурсу, визначеного в URL-адресу.

Сервер повинен знати, як обробляти цей HTTP-запит, щоб отримати правильний файл, а потім відправити його назад браузеру, який його запросив. Найбільш популярне програмне забезпечення веб-сервера, яке обробляє це Apache і NGINX. На рисунку 2.2 зображено схему роботи веб-серверу.

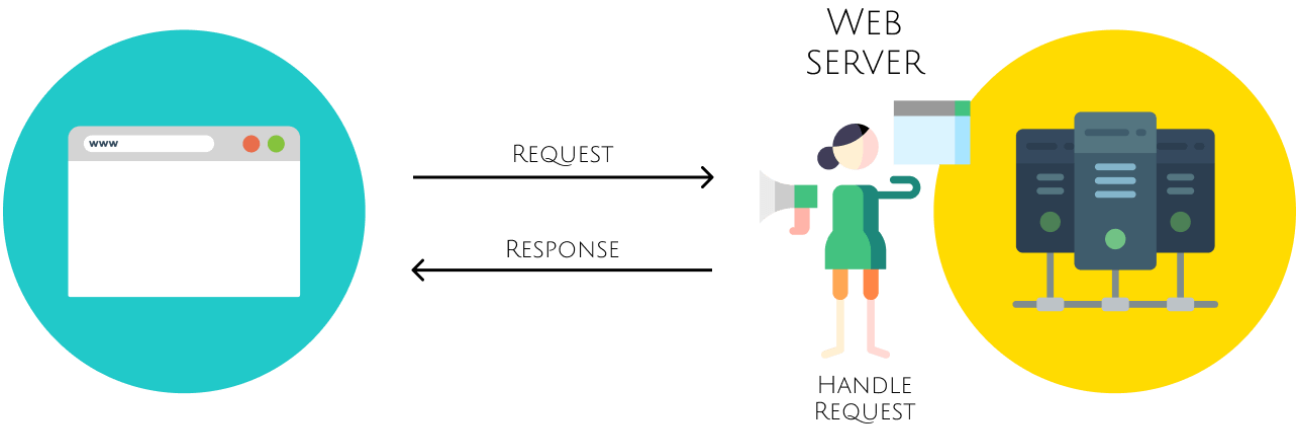

Рисунок 2.2 – Схема роботи веб-серверу

Обидва представляють собою повний пакет програм з відкритим вихідним кодом, які включають в себе такі функції, як схеми аутентифікації, перезапис URL-адрес, ведення журналу та проксінг, і це лише деякі з них. Apache і NGINX написані на C. Технічно, ви можете написати веб-сервер на будь-якій мові. Python, Go, Ruby, цей список можна тривати досить довго. Просто деякі мови краще роблять певні речі, ніж інші [7].

Створення HTTP-сервера за допомогою Node.js

Node.js - це середовище виконання Javascript, побудована на движку Chrome V8 Javascript. Він поставляється з модулем http, який надає набір функцій і класів для побудови HTTP-сервера.

Для цього простого HTTP-сервера ми також будемо використовувати file system, path і url, всі вони є нативними модулями Node.js. Почніть з імпорту необхідних модулів. На рисунку 2.3 зображено ініціалізацію базових модулів Node.js.

```
const http = require('http') // Чтобы использовать интерфейс HTTP в Node.js
const fs = require('fs') // Для взаимодействия с файловой системой
const path = require('path') // Для работы с путями файлов и папок
const url = require('url') // Для разрешения и парсинга URL-адресов
```

Рисунок 2.3 – Ініціалізація базових модулів Node.js

На рисунку 2.4 зображено створення серверу Node.js.

```
1 const server = http.createServer()
```

Рисунок 2.4 – Створення серверу Node.js

Об'єкт request є екземпляром IncomingMessage і дозволяє отримати доступ до різного роду інформації про запит, такий як відповідь стану, заголовки і дані.

Об'єкт response є екземпляром ServerResponse, який є потоком з можливістю запису, а також надає численні методи для відправки даних назад клієнту.

Тепер ми можемо створити HTTP-сервер за допомогою функції http.createServer (), яка буде повертати новий екземпляр http.Server [8].

2.2 Архітектура REST

Сервером в даному випадку ми вважаємо абстрактну машину в мережі, здатну отримати HTTP-запит, обробити його і повернути коректну відповідь. У контексті даної статті зовсім не важливі його фізична суть і внутрішня архітектура, будь то студентський ноутбук або величезний кластер з промислових серверів, розкиданих по всьому світу. Нам в тій же мірі зовсім неважливо, що у нього під капотом, хто зустрічає запит у дверей, Apache або Nginx, PHP, Python або Ruby виконує його обробку і формує відповідь, яке сховище даних використовується: Postgresql, MySQL або MongoDB . Головне, щоб сервер відповідав головному правилу - почути, зрозуміти і пробачити відповісти.

Клієнтом теж може бути все, що завгодно, що здатне сформувати і відправити HTTP-запит. До певного моменту в цій статті нам також не особливо будуть цікаві цілі, які ставить перед собою клієнт, відправляючи цей запит, як і те, що він буде робити з відповіддю. Клієнтом може бути JavaScript-сценарій, який працює в браузері, мобільний додаток.

Здебільшого ми будемо говорити про спосіб спілкування між вище перерахованими двома, такому способі, щоб вони один одного розуміли, і в жодного не залишалося питань.

Філософія REST

REST (Representational state transfer) спочатку був задуманий як простий і однозначний інтерфейс для управління даними, який передбачав лише кілька базових операцій з безпосереднім мережевим сховищем (сервером): вилучення даних (GET), збереження (POST), зміна (PUT / PATCH) і видалення (DELETE). Зрозуміло, цей перелік завжди супроводжувався такими опціями, як обробка помилок в запиті (чи коректно складений запит), розмежування доступу до даних (раптом цього вам знати не слід) і валідація входять даних (раптом ви написали

дурницю), в загальному, всіма можливими перевірками , які сервер виконує перед тим, як виконати бажання клієнта.

Крім цього REST має ряд архітектурних принципів:

- Незалежність сервера від клієнта - сервери і клієнти можуть бути миттєво замінені іншими незалежно один від одного, так як інтерфейс між ними не змінюється. Сервер не зберігає станів клієнта.

- Унікальність адрес ресурсів - кожна одиниця даних (будь-якого ступеня вкладеності) має свій власний унікальний URL, який, по суті, цілком є однозначним ідентифікатором ресурсу.

- Незалежність формату зберігання даних від формату їх передачі - сервер може підтримувати кілька різних форматів для передачі одних і тих же даних (JSON, XML і т.д.), але зберігає дані в своєму внутрішньому форматі, незалежно від підтримуваних.

- Присутність у відповіді всіх необхідних метаданих - крім самих даних сервер повинен повертати деталі обробки запиту, наприклад, повідомлення про помилки, різні властивості ресурсу, необхідні для подальшої роботи з ним, наприклад, загальне число записів в колекції для правильного відображення посторінкової навігації.

Класичний REST має на увазі роботу клієнта з сервером як з плоским сховищем даних, при цьому нічого не говориться про пов'язаності і взаємозалежності даних між собою. Все це за замовчуванням цілком лягає на плечі клієнтського додатку. Однак сучасні предметні області, для яких розробляються системи управління даними, будь то соціальні сервіси або системи інтернет-маркетингу, мають на увазі складну взаємозв'язок між сутностями, що зберігаються в базі даних. Підтримка цих зв'язків, тобто цілісності даних, знаходиться в зоні відповідальності серверної сторони, в той час, як клієнт є тільки інтерфейсом для доступу до цих даних. Так чого ж нам не вистачає в REST?

Щоб не міняти дані і зв'язку між ними вручну, ми просто викликаємо у ресурсу функцію і «скормлюємо» їй як аргумент необхідні дані. Ця операція не підходить під стандарти REST.

Найпростіший приклад - авторизація користувача. Ми викликаємо функцію login, передаємо їй як аргумент об'єкт, що містить облікові дані, і у відповідь отримуємо ключ доступу. Що твориться з даними на стороні сервера - нас не хвилює.

Ще варіант - створення і розрив зв'язків між даними. Наприклад, додавання користувача в групу. Викликаємо у сутності група функцію addUser, як параметр передаємо об'єкт користувач, отримуємо результат.

А ще бувають операції, які взагалі не пов'язані безпосередньо зі збереженням даних як таких, наприклад, розсилка повідомлень, підтвердження або відхилення будь-яких операцій (завершення звітного періоду etc) [9].

2.3 Обгрунтування вибору бази даних

Коли необхідно вибрати СУБД, головне питання зазвичай полягає у виборі реляційної (SQL) або нереляційних (NoSQL) структури. У обох варіантів є свої переваги, а також кілька ключових особливостей, які варто мати на увазі при виборі[10].

Реляційні бази даних використовують структуровану мову запитів (SQL) для визначення та обробки даних. З одного боку, це відкриває великі можливості для розвитку. SQL - одна з найбільш гнучких і популярних мов запитів, тому вибір SQL мінімізує деякі ризики і особливо корисний, коли вам потрібно працювати зі складними запитами. З іншого боку, SQL має деякі обмеження. Щоб створити запит цією мовою, потрібно визначити структуру даних. Подальші зміни структури даних можуть негативно вплинути на всю систему.

Нереляційні бази даних, в свою чергу, пропонують динамічну структуру даних, які можуть зберігатися кількома способами: орієнтовано по колонках, документо-орієнтовано, в вигляді графів або на основі пар «ключ-значення». Така гнучкість означає наступне:

- ви можете створювати документи, не ставлячи їх структуру заздалегідь;
- кожен документ може мати власну структурою;
- у кожної бази даних може бути власний синтаксис;
- ви можете додавати поля прямо під час роботи з даними.

У більшості випадків бази даних SQL є вертикально масштабованими. Тобто ви можете збільшити навантаження на один сервер, збільшивши потужність процесора, оперативну пам’ять або сховище. Бази даних NoSQL є горизонтально масштабованими. Тобто можна збільшити трафік, розподіливши трафік або додавши сервери до бази даних. Це як додавання підлоги до будинку або будівлі на вулиці. У другому випадку система буде настільки великою і потужною, що вибір баз даних NoSQL підходить для великих або постійно мінливих структур даних.

Структура реляційної бази даних подає дані у вигляді таблиць, але нереляційних документів, пар «ключ-значення», графіків або сховищ широких стовпців. Це робить бази даних SQL кращим вибором для програм, що включають транзакції з декількома записами, наприклад, облікові системи або застарілі системи, побудовані для реляційних структур.

СУБД для баз даних SQL включає MySQL, Oracle, PostgreSQL та Microsoft SQL Server. MongoDB, BigTable, Redis, RavenDB Cassandra, HBase, Neo4j та CouchDB підходять для роботи з NoSQL[11].

Розібравшись з ключовими структурними відмінностями SQL і NoSQL баз даних, варто уважно розглянути їх функціональні особливості на прикладі MySQL і MongoDB.

MySQL

Переваги MySQL:

- перевірено часом: MySQL - вкрай розвинена СУБД, що означає наявність великої спільноти навколо неї, безліч прикладів і високу надійність;
- сумісність: MySQL доступна на всіх основних платформах, включаючи Linux, Windows, Mac, BSD і Solaris. Також у неї є бібліотеки для мов на зразок Node.js, Ruby, C #, C ++, Java, Perl, Python і PHP;
- окупність: Це СУБД з відкритим вихідним кодом, що знаходиться у вільному доступі;
- базу даних MySQL можна розподіляти між кількома вузлами, таким чином зменшуючи навантаження і покращуючи масштабованість і доступність додатка;
- шардінг: В той час як шардінг неможливий на більшості SQL баз даних, MySQL є винятком.

MongoDB

Переваги MongoDB:

- динамічна схема: Як згадувалося вище, ця СУБД дозволяє гнучко працювати зі схемою даних без необхідності змінювати самі дані;
- масштабованість: MongoDB горизонтально масштабируема, що дозволяє легко зменшити навантаження на сервера при великих обсягах даних;
- зручність в управлінні: СУБД не потребує окремому адміністратора бази даних. Завдяки достатній зручності у використанні, їй легко можуть користуватися як розробники, так і системні адміністратори;
- швидкість: Висока продуктивність при виконанні простих запитів;

- гнучкість: В MongoDB можна без шкоди для існуючих даних, їх структури і продуктивності СУБД додавати поля або колонки.

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який з реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин.

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON-подібна схема даних)
- Досить гнучка мова для формування запитів
- Динамічні запити
- Повна підтримка індексів

- Профілювання запитів
- Швидкі оновлення «на місці»
- Ефективне зберігання бінарних даних великих обсягів, наприклад, фото та відео
- Журналювання операцій, що модифікують дані в БД
- Підтримка відмовостійкості і масштабованості: асинхронна реплікація, набір реплік і шардінг
- Може працювати відповідно до парадигми MapReduce.

Розглянемо, як буде виглядати стандартна реляційна база даних та база даних у MongoDB.

На рисунку 2.5 показано як виглядає реляційня БД: 2 об’єкти (developers і projects), між якими є зв’язок.

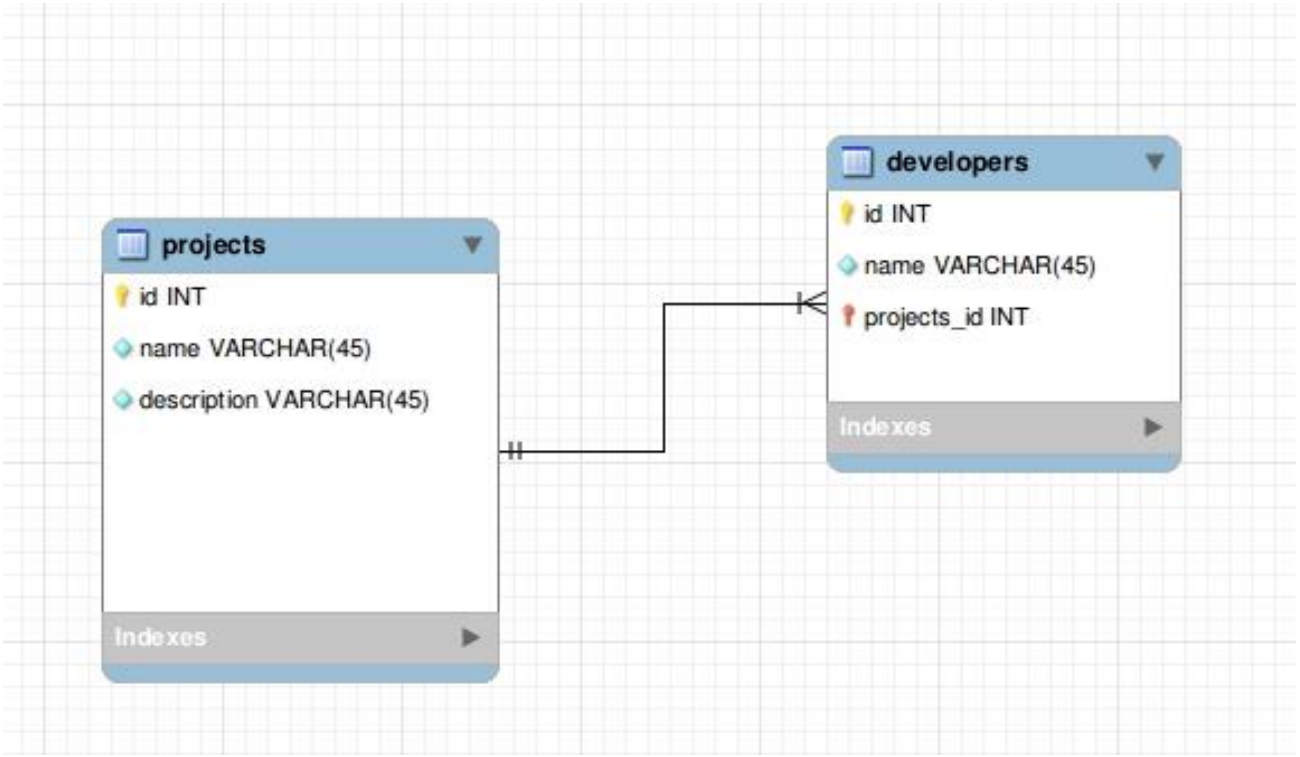

Рисунок 2.5 – Приклад реляційної бази даних

У MongoDB все відбувається по-іншому. Є один великий об’єкт з певною структурованою інформацією, назвемо це колекцією даних, і всередині цього

об’єкта є інші об’єкти які і зберігають потрібну інформацію. На рисунку 2.6 зображено приклад колекції даних у MongoDB.

```
{
   _id: PROJECT_ID
   name: NAME_OF_PROJECT,
   developers: [
      {
         name:'DEVELOPER_NAME',
      },
      {
         name: 'DEVELOPER_NAME'
      }
   ]
}
```

Рисунок 2.6 – Приклад колекцій даних у MongoDB

2.4 Розробка алгоритмів реєстрації та входу

Перш ніж приступити безпосередньо до створення онлайн-сервісу, необхідно розробити структуру основних модулів, щоб користувачі мали змогу легко користуватись онлайн-сервісом, а також, щоб вони у будь-який момент могли повернутися на домашню сторінку. Структура онлайн-сервісу зображена на рисунку 2.7.

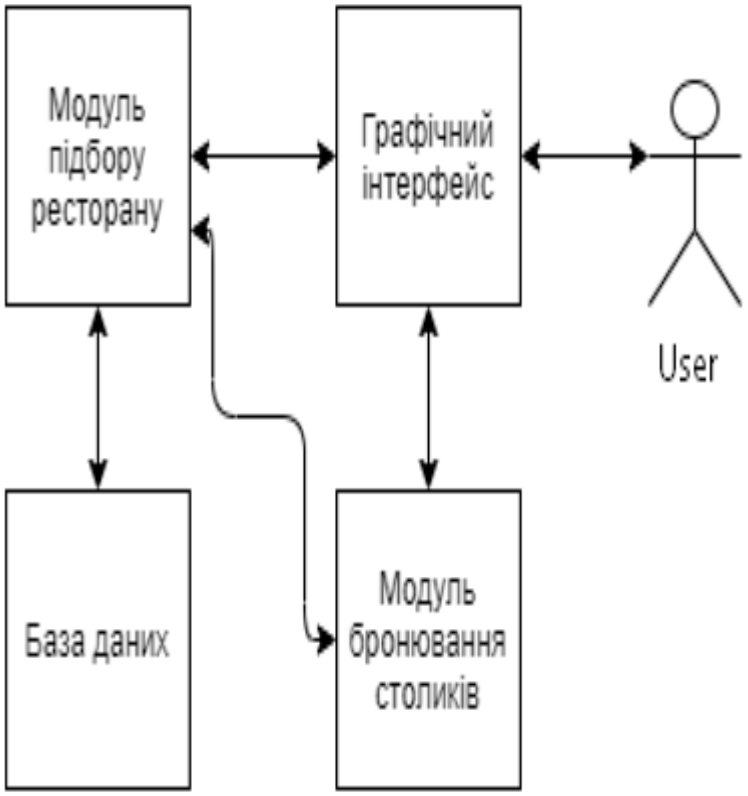

Рисунок 2.7 – Структурна схема онлайн-сервісу для бронювання столиків

Також необхідно зробити «логіку» сервера. Від клієнта до сервера будуть надходити дані і для реєстрації потрібно що б сервер записував дані користувача до БД, а при вході, перевіряв чи такий користувач наявний.

Під час реєстрації перевіряється чи email який поданий в формі не присутній. При його наявності користувачу буде висвітлена помилка «Користувач за таким email зареєстрований». Якщо ж email не зареєстрований, тоді в базі даних створюється новий користувач з даними які були записані в формі реєстрації.

Діаграма діяльності сервера при реєстрації зображена на рисунку 2.8

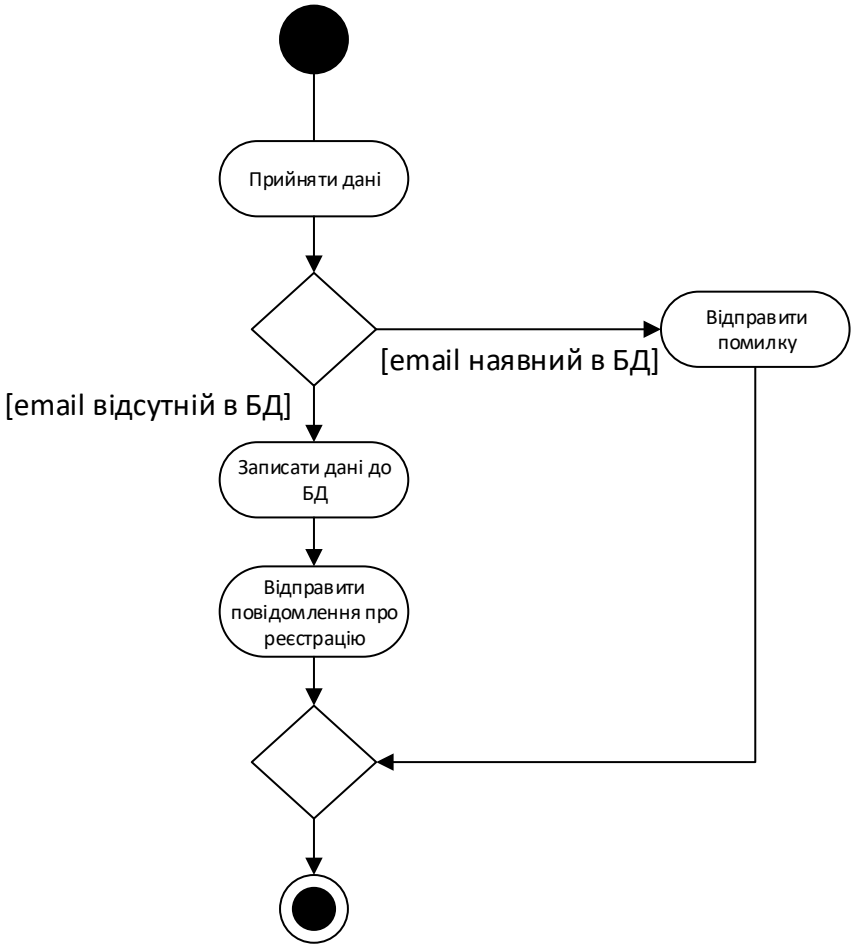

Рисунок 2.8 – Діаграма діяльності сервера при реєстрації

Зображений вище алгоритм можна записати у вигляді коду:

```
module.exports.register = async function (req, res) {
const candidate = await User.findOne({ email: req.body.email });
if (candidate) {
  res.status(409).json({
    message: "Користувач з таким email уже зареєстрований",
  });
} else {
  const salt = bcrypt.genSaltSync(10);
  const password = req.body.password;
```

```
const user = new User({
  firstName: req.body.firstName,
  lastName: req.body.lastName,
  email: req.body.email,
  password: bcrypt.hashSync(password, salt),
});
try {await user.save();
  res.status(201).json(user);}
        catch (e) {errorHandler(res, e);}}};
```

При вході, сервер також перевіряє email на наявність, але помилку видає якщо email не знайдено, далі йде перевірка паролю. Якщо ж email та пароль зівпали то сервер генерує процес входу.

Діаграма діяльності сервера при вході, рисунок 2.9

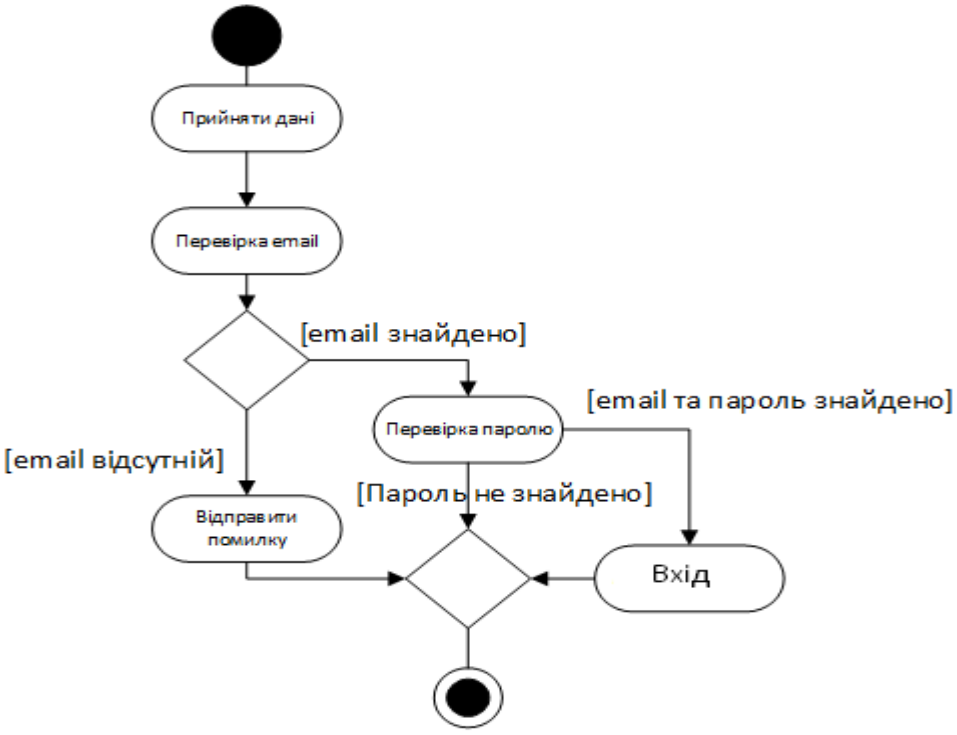

Рисунок 2.9 – Діаграма діяльності сервера при вході

Код модуля входу:

```
module.exports.login = async function (req, res) {
 const candidate = await User.findOne({ email: req.body.email });
 if (candidate) {const passwordResult = bcrypt.compareSync(req.body.password,
    candidate.password);
  if (passwordResult) {const token = jwt.sign({
     email: candidate.email,
     userId: candidate._id,},
   keys.jwt, {expiresIn: 60 * 60 * 60 *60 *60});
   res.status(200).json({token: `Bearer ${token}`, candidate });
  } else {
        res.status(401).json({message: "Пароль введений не вірно",
        });}
 } else {res.status(404).json({message: "Користувача з таким email не
знайдено",});}};
```

Для заборони доступу незалогованих користувачів потрібно зробити захист певних шляхів на стороні клієнту. У React.js це можна зробити через перевірку певних даних на стороні клієнта. Для простоти будемо зберігати поле «uid» тобто ідентифікатор користувача на стороні клієнта, в LocalStorage браузера. І при кожній спробі входу на сторінку користувача, будем перевіряти наявність цього поля. Якщо ж поле «uid» наявне, створимо також поле «logined» яке буде приймати в себе чи залоговарий користувач, чи ні. В разі відсутності логування, користувач буде перенаправлений на строрінку логування і буде надіслане повідомлення «Для доступу необхідно ввійти»

Код зображений нижче.

```
        beforeEnter: (to, from, next) => {
 if (!store.state.system.logined) { next( { path: '/login',  query: {  accessDenied: true } })  }
        else {  next()  }}.
```

2.5 Розробка основного алгоритму роботи онлайн-сервісу

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Нижче на рисунку 2.10 показано, як відбувається спілкування між клієнтом і сервером. Коли клієнт робить запит на сервер, щоб отримати якісь дані, то сервер звертається до бази даних і отримує потрібні йому дані(якщо не буде помлки) і потім повертає ці дані для клієнта [12].

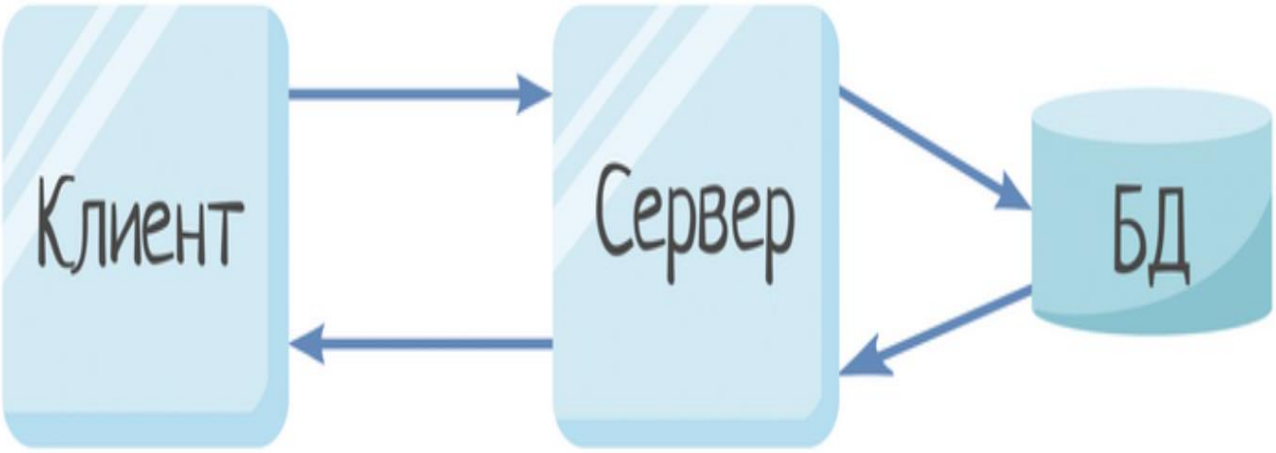

Рисунок 2.10 – Схема роботи клієнт-серверної архітектури

Алгоритм роботи серверної частини для інформаційної технології бронювання місць в ресторані працює наступним чином. Спочатку користувачу потрібно авторизуватись, щоб отримати доступ для користування онлайн-сервісом. Якщо авторизація закінчилась помилкою, юзер отрмає повідомлення про помилку. Після успішної авторизації, користувач, використовуючи клієнтську частину може шукати ресторан, який його цікавить та забронювати його. Після кожного успішного запиту на сервер інформація записується у відповідну колекцію даних у базі даних MongoDB. На рисунку 2.11 зображено алгоритм роботи серверної частини онлайн-сервісу.

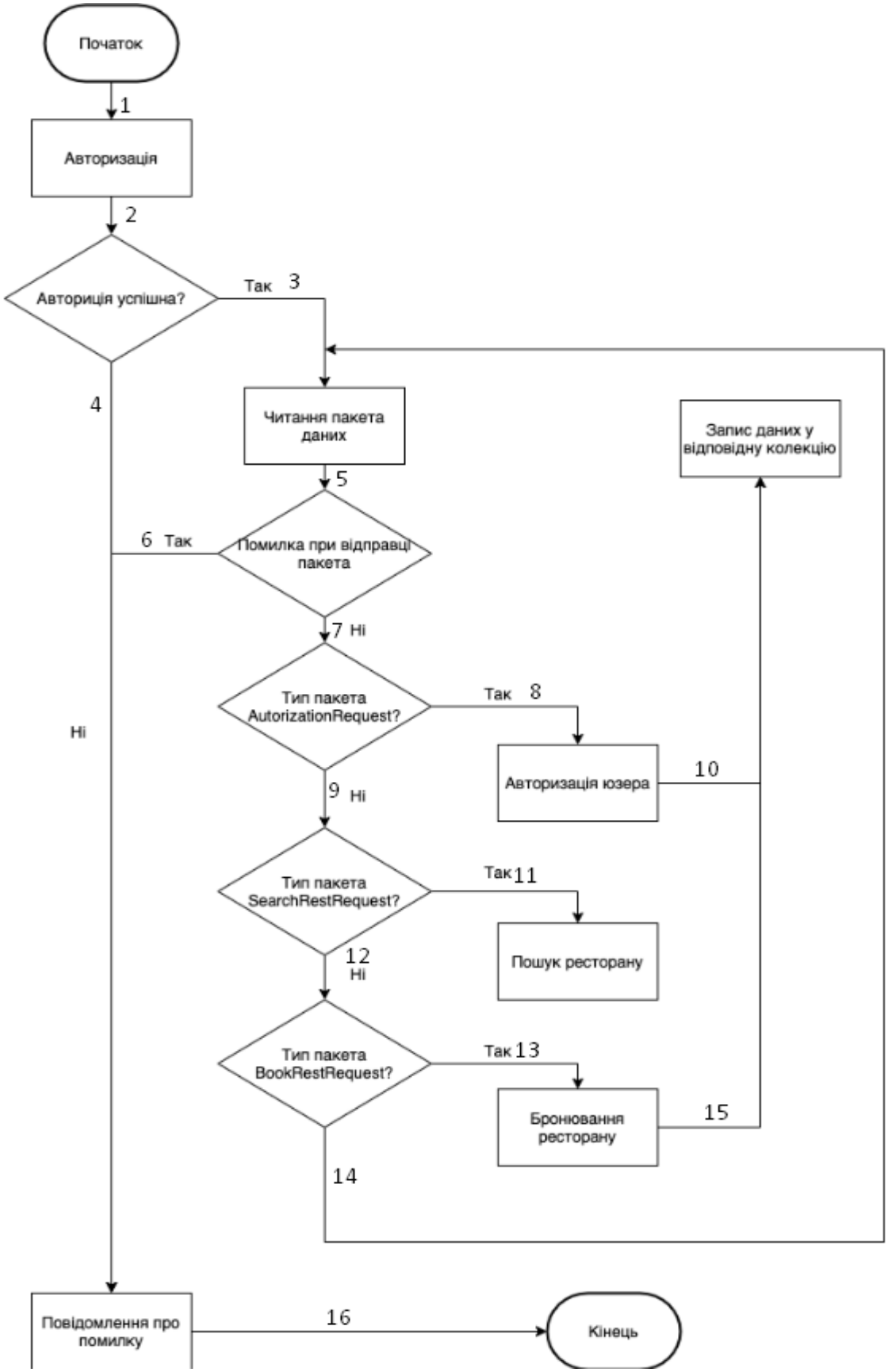

Рисунок 2.11 – Алгоритм роботи серверної частини

2.6 Висновок

В другому розділі магістерської кваліфікаційної роботи виконано:

– проаналізовано проектування серверної частини онлайн-сервісу;

– розглянуто переваги та неділоки баз даних та вибрано Mongo db для подальшої роботи;

– розроблено алгоритм реєстрації та авторизації користувачів

– розроблено основний алгоритм роботи серверної частини онлайн-сервісу

Як результат було обрано Mongo db середовищем для подальшого зберігання даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ОНЛАЙН СЕРВІСУ ДЛЯ БРОНЮВАННЯ МІСЦЬ В РЕСТОРІ

3.1 Підготовка до розробки серверної частини для онлайн сервісу бронювання місць в ресторані

Для того, щоб почати роботу з серверною частиною потрібно підготувати робоче середовище. Щоб розробити цей онлайн сервіс будуть використовуватись наступні технології:

1. NodeJS
2. Express
3. MondoDB
4. Mongoose

Встановлення NodeJS. Для того, щоб встановити NodoJS необхідно зайти на офіційни сайт NodeJS. Далі потрібно завантажити .exe файл. Після завантаження виконати встановлення. На рисунку 3.1 показано встановлення NodeJS[14].

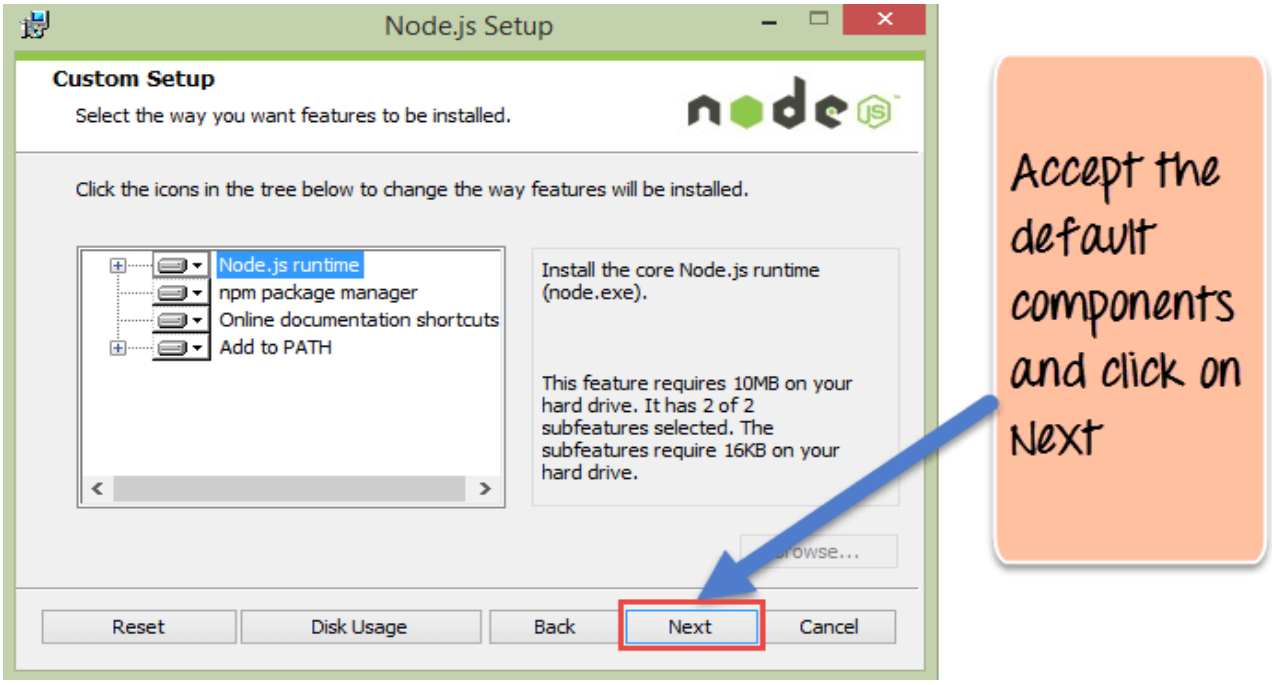

Рисунок 3.1 – Встановлення NodeJS

Встановлення Express. Після успішного встановлення бібліотеки, яка згадувалась вище перейдемо до встановлення Express. Express – це найпопулярніший веб-фреймворк для Node. Він є базовою бібліотекою для ряду інших популярних веб-фреймворків Node. Він надає наступні механізми:

- Написання обробників для запитів з різними HTTP-методами в різних URL-адреси (маршрутах).
- Інтеграцію з механізмами рендеринга «view», для генерації відповідей, вставляючи дані в шаблони.
- Установка загальних параметрів веб-додатки, такі як порт для підключення, і розташування шаблонів, які використовуються для відображення відповіді.
- «Проміжне ПО» для додаткової обробки запиту в будь-який момент в конвеєрі обробки запитів.

Щоб встановити даний фреймворк потрібно відкрити будь-який термінал та прописати наступну команду зображену на рисунку 3.2:

```
$ npm install express --save
```

Рисунок 3.2 – Встановлення Express

Встановлення MongoDB та Mongoose. Методика встановлення даних технологій аналогічна до методики встановлення Express[15].
Mongoose – це ODM (* Object Document Mapper - об'єктно-документний відображувач). Це означає, що Mongoose дозволяє вам визначати об'єкти зі строго-типізованою схемою, що відповідає документу MongoDB. Mongoose дає величезний набір функціональних можливостей для створення та роботи зі схемами. Наразі Mongoose має вісім SchemaTypes (* типи даних схеми), котрі може мати властивість, що зберігається до MongoDB. Ці типи наступні:

1. String

2. Number
3. Date
4. Buffer
5. Boolean
6. Mixed
7. ObjectId (* унікальний ідентифікатор об'єкта, первинний ключ, _id)
8. Array

Для кожного типу даних можна:

- зазначити значення за налаштуванням
- зазначити функцію користувача для перевірки даних
- зазначити, що поле необхідно заповнити
- зазначити get-функцію (геттер), яка дозволяє вам проводити маніпуляції над даними до їх повернення у вигляді об'єкта
- зазначити set-функцію (* сеттер), яка дозволяє вам проводити маніпуляції над даними перед їх зберіганням до бази даних
- визначити індекси для більш швидкого отримання даних.

Для встановлення цих інструментів необхідно зробити наступне зображене на рисунках 3.3 та 3.4 відповідно:

Рисунок 3.3 – Встановлення MongoDB

Рисунок 3.4 – Встановлення Mongoose

3.2 Особливості середовища розробки та обґрунтування вибору

Visual Studio Code - редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Позиціонується як «легкий» редактор коду для кроссплатформенной розробки веб-і хмарних додатків. Включає в себе відладчик, інструменти для роботи з Git, підсвічування синтаксису, IntelliSense і засоби для рефакторинга. Має широкі можливості для кастомізації: призначені для користувача теми, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, але готові збірки розповсюджуються під проприетарной ліцензією [13].

Visual Studio Code заснований на Electron - фреймворк, що дозволяє з використанням Node.js розробляти настільні додатки, які працюють на движку Blink. Незважаючи на те, що редактор заснований на Electron, він не використовує редактор Atom. Замість нього реалізується веб-редактор Monaco, розроблений для Visual Studio Online.

Visual Studio Code - це редактор вихідного коду. Він підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію по коду, підтримку Git та інші можливості. Багато можливості Visual Studio Code не доступні через графічний інтерфейс, найчастіше вони використовуються через палітру команд або JSON файли (наприклад, призначені для користувача настройки). Палітра команд представляє собою подобу командного рядка, яка викликається поєднанням клавіш.

Visual Studio також дозволяє замінювати кодову сторінку при збереженні документа, символи перекладу рядка і мову програмування поточного документа.

З 2018 року з'явилися розширення Python для Visual Studio Code з відкритим вихідним кодом. Воно надає розробникам широкі можливості для редагування, налагодження і тестування коду.

На березень 2019 року за допомогою вбудованого в продукт призначеного для користувача інтерфейсу можна завантажити і встановити кілька тисяч розширень тільки в категорії «programming languages» (мови програмування).

Приклад роботи програми Microsoft Visual Studio Code зображено на рисунку 3.5.

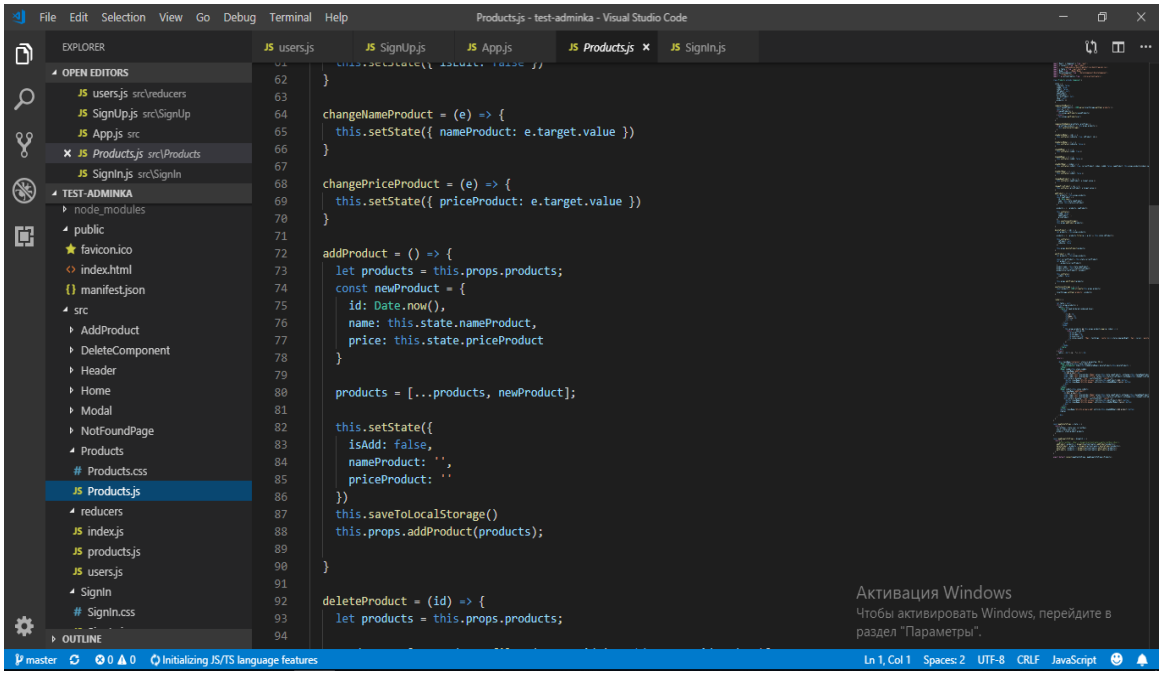

Рисунок 3.5 – Приклад роботи програми Microsoft Visual Studio Code

Brackets — текстовий редактор від компанії Adobe, призначений для редагування JavaScript, HTML і CSS. Сирцевий код Brackets написаний з використанням веб-технологій (JavaScript, HTML і CSS) і поширюється під ліцензією MIT. Редактор оформлений у вигляді відокремленого настільного застосунка, для установки якого підготовлені deb-, dmg- і msi- пакети для Linux, OS X і Windows.

Brackets підтримує режим Live-розробки, при якому редагований контент (JavaScript, HTML і CSS) у міру зміни відразу відображається в синхронізованому з редактором вікні браузера — розробник може змінювати вміст і відразу спостерігати до яких наслідків приводять дані зміни. Налагодження також може виконуватися синхронно із браузером, розробник може встановити точку зупину або відкотитися на крок назад при перегляді результатів. Є вбудована підтримка препроцесорів LESS і SCSS. В інтерфейсі застосовується система контекстно-залежних інструментів, що з'являються в міру необхідності в основному вікні розробки. Для розширення можливостей редактора розвивається система доповнень. Серед доступних плагінів рекомендуємо наступні:

– Emmet - розширення для швидкого введення шаблонних команд;

– Beautifer - додаток для оформлення коду;

– FTP-Sync - плагін для синхронізації з віддаленим сервером;

– ColorHints - надбудова для зручного підбору кольорів.

– Брекет підійде для комп'ютерів на базі різних версій операційної системи - Віндовс 7-10, Vista.

Приклад роботи програми Brackets зображено на рисунку 3.6.

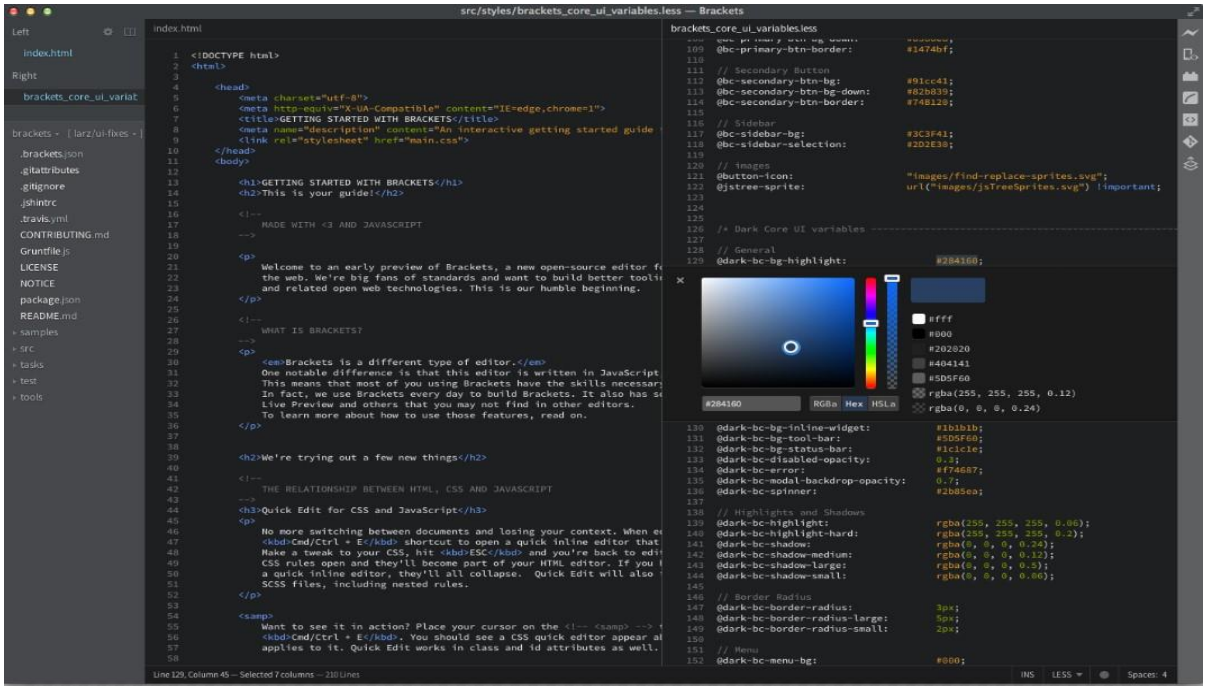

Рисунок 3.6 – Приклад роботи програми Brackets

Sublime Text — швидкий кросплатформенний текстовий редактор. Підтримує плагіни, розроблені за допомогою мови програмування Python.

Sublime Text не є вільним чи відкритим програмним забезпеченням, але деякі його плагіни розповсюджуються з вільною ліцензією, розробляються і підтримуються спільнотою розробників.

Редактор містить різні візуальні теми, з можливістю завантаження додаткових.

Користувачі бачать весь свій код в правій частині екрану у вигляді міні-карти, при кліці на яку можна здійснювати навігацію.

Є кілька режимів екрану. Один з них включає від 1 до 4 панелей, за допомогою яких можна показувати до чотирьох файлів одночасно. Повноцінний (free modes) режим показує тільки один файл без будь-яких додаткових навколо нього меню.

Виділення стовпців цілком або розстановка кілька покажчиків по тексту, що робить можливим миттєву правку. Покажчики поводяться, ніби кожен з них

- єдина в тексті. Команди типу: переміщення на знак, переміщення на рядок, вибірка тексту, переміщення на слово або його частини (CamelCase, розділений дефісом або підкресленням), перехід на початок або кінець рядка тощо, Впливає на всі покажчики незалежно і відразу, дозволяючи правити складноструктурований текст швидко, без використання макрокоманд або регулярних виразів.

Коли користувач набирає код, Sublime Text, в залежності від використовуваної мови, буде пропонувати різні варіанти для завершення запису. Редактор також автоматично завершує створені користувачем змінні.

Підсвічування синтаксису і висока контрастність

Темний фон Sublime Text призначений для збільшення контрастності тексту. Основні елементи синтаксису виділені різними кольорами, які краще поєднуються з темним тлом, ніж зі світлим.

Sublime Text дозволяє користувачеві збирати програми і запускати їх без необхідності перемикатися на командний рядок. Користувач також може налаштувати свою систему збирання та включити автоматичну збірку програми кожного разу при збереженні коду.

Приклад роботи програми Sublime Text 3 зображено на рисунку 3.7.

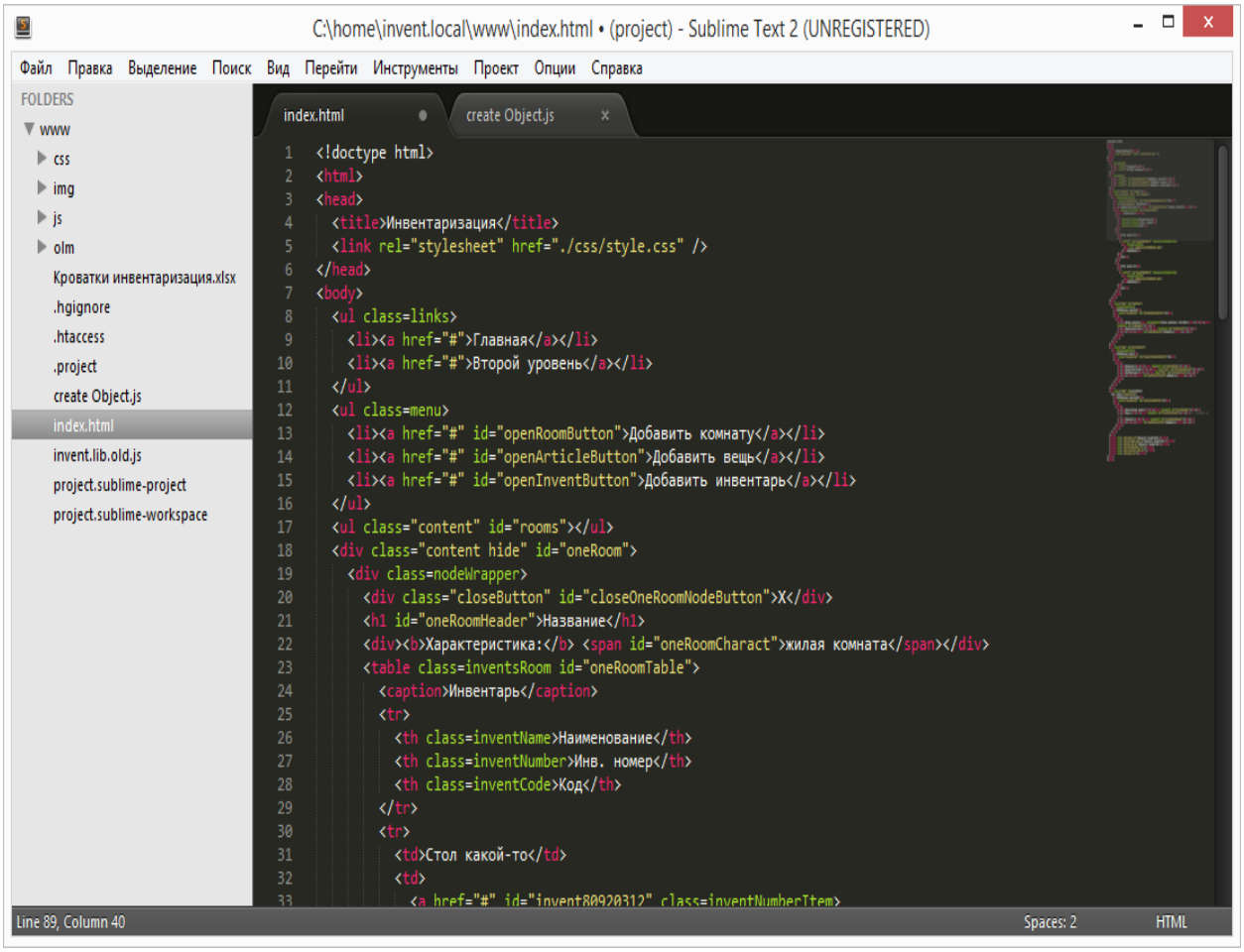

Рисунок 3.7 – Приклад роботи програми Sublime Text 3

Порівняння даних середовищ програмування наведено в Таблиці 3.1.

Таблиця 3.1 – Порівняння даних середовищ програмування

Функції	Середовище програмування		
	Brackets	Sublime Text 3	Microsoft Visual Studio
Режим відлагодження (0,9)	0.5	0.5	1
Кросплатформеність (0,6)	0	0	1

Функція авто заповнення (0,4)	0.5	0.5	1
Загальний коефіцієнт	0,65	0,5	1.9

3.3 Написання серверної частини для онлайн сервісу бронювання місць в ресторані

Ініціалізація серверу. Перш ніж почати працювати з сервером його потрібно ініціалізувати (створити). Для того, що створити і запустити сервер в NodeJS потрібно виконати наступні дії (фрагмент ініціалізації сервера наведений на рисунку 3.8).

```
// Загружаем HTTP модуль
const http = require("http");

const hostname = "127.0.0.1";
const port = 8000;

// Создаем HTTP-сервер
const server = http.createServer((req, res) => {

   // Устанавливаем HTTP-заголовок ответа с HTTP статусом и Content type
   res.writeHead(200, {'Content-Type': 'text/plain'});

   // Отсылаем тело ответа "Hello World"
   res.end('Hello World\n');
});

// Выводим лог как только сервер будет запущен
server.listen(port, hostname, () => {
   console.log(`Server running at http://${hostname}:${port}/`);
})
```

Рисунок 3.8 – Ініцалізація сервера в NodeJS

Далі перевіримо чи сервер працює і готовий до використання в подальшій розробці. Для цього запустимо сервер. Щоб запустити сервер потрібно перейти

через термінал в папку, в якій лежить даний файл і виконати команду *node hello.js.*

Створення бази даних. Після ініціалізації сервера створюємо базу даних, щоб ми могли тримати в ній дані про ресторани та користувачів. Фрагмент створення бази даних наведений на рисунку 3.9.

```
'use strict';

var nconf = require('nconf');
var databaseName = nconf.get('database');
var winston = require('winston');

if (!databaseName) {
	winston.error(new Error('Database type not set! Run ./nodebb setup'));
	process.exit();
}

const primaryDB = require('./' + databaseName);

primaryDB.parseIntFields = function (data, intFields, requestedFields) {
	intFields.forEach((field) => {
		if (!requestedFields.length || requestedFields.includes(field)) {
			data[field] = parseInt(data[field], 10) || 0;
		}
	});
};

primaryDB.initSessionStore = async function () {
	const sessionStoreConfig = nconf.get('session_store') || nconf.get('redis') || nconf.get(databaseName);
	let sessionStoreDB = primaryDB;

	if (nconf.get('session_store')) {
		sessionStoreDB = require('./' + sessionStoreConfig.name);
	} else if (nconf.get('redis')) {
		// if redis is specified, use it as session store over others
		sessionStoreDB = require('./redis');
	}

	primaryDB.sessionStore = await sessionStoreDB.createSessionStore(sessionStoreConfig);
};

module.exports = primaryDB;
```

Рисунок 3.9 – Створення бази даних MongoDB

Після базового налаштування середовища можемо приступити до самої розробки.

Створення контролеру для аутентифікації користувача. Перш за все потрібно визначитись, які поля будуть записані у користувача в базі даних. Користувач повинен мати наступні дані(ці дані беруться з форми реєстрації на клієнтській частину, які юзер вказує при реєстрації): імя, прізвище, e-mail, номер телефону.

Нижче, на рисунку 3.10, наведений фрагмент коду створення контролеру для аутентифікації.

```
authenticationController.localLogin = async function (req, username, password, next) {
	if (!username) {
		return next(new Error('[[error:invalid-username]]'));
	}

	if (!password || !utils.isPasswordValid(password)) {
		return next(new Error('[[error:invalid-password]]'));
	}

	if (password.length > 512) {
		return next(new Error('[[error:password-too-long]]'));
	}

	const userslug = slugify(username);
	const uid = await user.getUidByUserslug(userslug);
	try {
		const [userData, isAdminOrGlobalMod, banned, hasLoginPrivilege] = await Promise.all([
			user.getUserFields(uid, ['uid', 'passwordExpiry']),
			user.isAdminOrGlobalMod(uid),
			user.bans.isBanned(uid),
			privileges.global.can('local:login', uid),
		]);

		userData.isAdminOrGlobalMod = isAdminOrGlobalMod;

		if (parseInt(uid, 10) && !hasLoginPrivilege) {
			return next(new Error('[[error:local-login-disabled]]'));
		}

		if (banned) {
			const banMesage = await getBanInfo(uid);
			return next(new Error(banMesage));
		}
```

Рисунок 3.10 – Фрагмент контролеру аутентифікації користувача

3.4 Висновок

В третьому розділі магістерської кваліфікаційної роботи виконано:

– встановлення робочого середовища для розробки;

– створено алгоритми роботи окремих частин програми;

– обгрунтування та вибір середовища програмування;

– розглянуто розроблення серверної частини;

Як результат отримано базову функціонуючу серверну частину онлайн-сервісу для бронювання місць в ресторані.

4 ТЕСТУВАННЯ ОНЛАЙН СЕРВІСУ ДЛЯ БРОНЮВАННЯ МІСЦЬ В РЕСТОРАНІ

4.1 Аналіз методів і засобів тестування

Тестування – це процес експериментування з продуктом за допомогою тестів з метою виявлення в ньому помилок (неточностей, допущених розробниками ПЗ). Основна ідея тестування – запустити ПЗ і спостерігати за його роботою та її наслідками. Якщо збій в роботі програмного забезпечення відбувся – аналізується звіт з метою виявлення місцезнаходження помилки, яка його викликала.

Тестування може відбуватися різними способами, однак не варто забувати про сам процес і стратегії тестування. Від нього залежить послідовність ваших дій. На сьогоднішній день, фахівці з тестування веб-сайтів застосовують такі види як:

1. Функціональне тестування. Один з важливих і незамінних видів тестування. Найголовніше правило функціонального тестування є правильні розрахунки функцій. Наприклад, візьмемо інтернет-магазин, у якого є не тільки знижки на товар, але і безліч статусів при покупці, n кількість товарів. Всі ці варіанти слід враховувати. Адже якщо функціонал проекту не працює в певному браузері, то він не буде працювати ніде.

2. Тестування зручності користування (юзабіліті). Тестування зручності користування (юзабіліті) – це вид тестування, який надає сайту зручність і практичність у використанні.

3. Тестування продуктивності. Тестування продуктивності – в основному це тестування навантаження. Тестування навантаження сайту перевіряється в більшості випадків автоматом, тобто спеціальними програмами.

Це дає шанс перевірити, наскільки він буде працювати під певним навантаженням. Мета цього тестування, полягає в кількості віртуальних користувачів, які задають n кількість запитів, в один час (будь це секунди навіть). Тим самим результат дає зрозуміти, чи зміг наш проект витримати, наприклад, 100 користувачів, які одночасно купували товар або авторизувалися на сайті, відповідь показує, чи реально витримати сайту таке навантаження.

4. Тестування інтерфейсу користувача. UI testing – це тестування графічного інтерфейсу користувача, яке передбачає перевірку сайту на відповідність вимогам до графічного інтерфейсу.

5. Тестування безпеки. Це ключ до надійності веб-сайтів. Основні правила цього тестування – це перевірка на уразливість різних видів атак. Якщо це інтернет-магазин, то, швидше за все, слід перевіряти запити на Sql ін'єкцію (запити до бази даних). SQL-ін'єкції – це шкідливий код в запитах бази даних - найбільш небезпечний вид атак. Вона дає можливість впровадити довільний код, і атакувати комп'ютери користувачів, які переглядають заражені сторінки.

Важливими вимогами до сучасних сайтів є їх кросбраузерність та адаптивність. Під кросбраузерністю сайту розуміють коректне відображення елементів сторінки та їх робота в усіх браузерах.

Кросбраузерність розробленого сайту перевірялася в таких сучасних браузерах, як: Google Chrome, Firefox та Opera. При тестуванні не було виявлено жодних помилок, усі елементи сайту однаково відображаються в усіх цих браузерах.

4.2 Тестування роботи запитів серверної частини

Проведем тестування роботи запитів серверної частини. За допомогою графічного інтерфейсу Mongo Compas, розглянемо різні команди взаємодії с

Mongo db. Для початку розглянему команду добавлення нових юзерів. За допомогою команд додамо двох нових користувачів.

```
db.users.insertMany([
{
"_id": 1,
"name": "Kostya",
"email": "someemail@gmail.com",
"phone": "0677873432",
"intereses": ["Українська кухня", "Зал для курців"],
"password": "password"
},
{
"_id": 2,
"name": "Andrii",
"email": "someemail232@gmail.com",
"phone": "0677823432",
"intereses": ["Українська кухня", "Японська кухня"],
"password": "password1"
},
])
```

Результат зображено на рисунку 4.1.

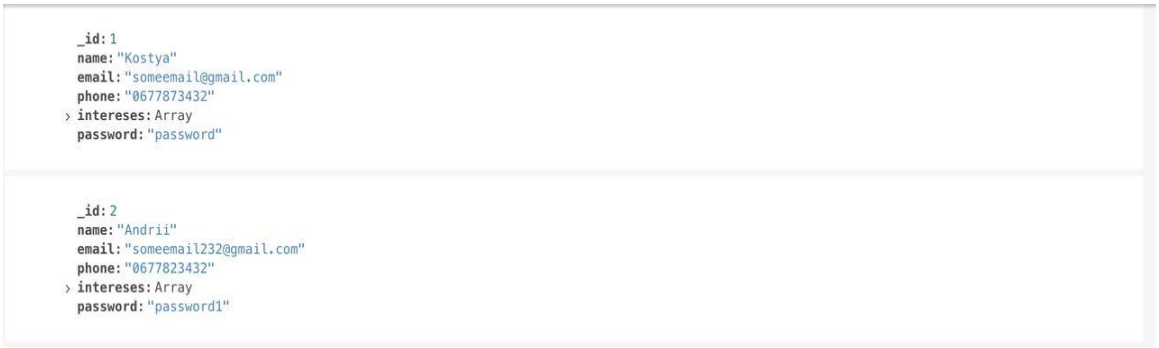

Рисунок 4.1 – Додавання користувачів до Mongo db

Тепер перевіримо правильність введення запитів для додавання інформації про ресторан до бази даних.

```
db.restaurants.insertMany([
{
"_id": 1,
"name": "Aura",
"adress": "площа Гагаріна, 2, Вінниця",
"from": "12:00",
"to": "02:00",
"features": ["кальян", "караоке", "європейська кухня"],
```

"keywords": ["кальян", "караоке", "європейська кухня", "центр", "Вінниця"],
"description" : "В центре города Винница зажглись огни нового фешенебельного заведения «Аура», который.. "

За допомогою даного коду створюємо ще декілька елементів та перевіряємо їх наявність в бд.

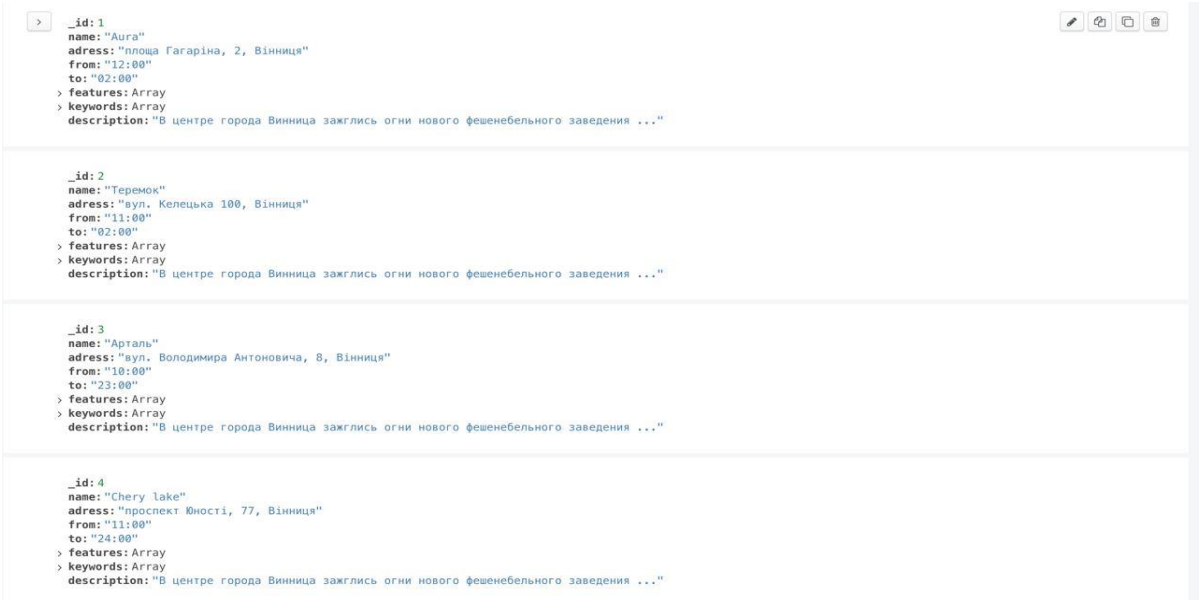

Рисунок 4.2 – Результат запитів на додавання ресторанів

Тепер після внесення даних про ресторани та користувачів перевірим виведення інформації за допомогою фільтрів. Виведемо ресторан “Aura” за допомогою фільтра name. Виведення за фільтром зображено на рисунку 4.3

```
db.restaurants.find({"name":"Aura"})
```

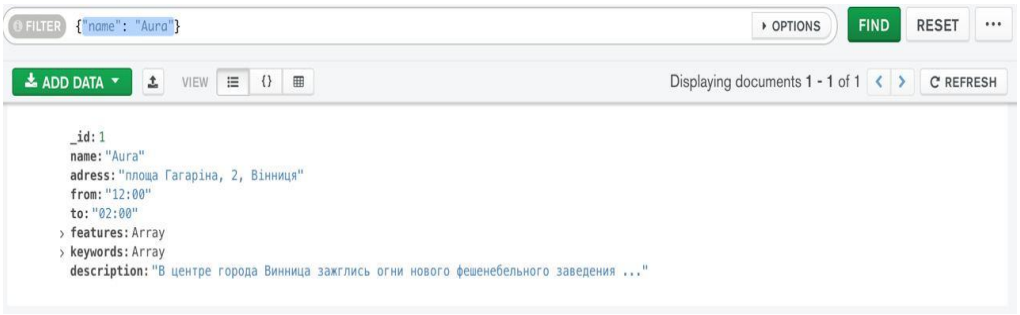

Рисунок 4.3 – Виведення ресторану за фільтром

Тепер перевіримо виведення інформації за допомогою декількох фільтрів. Виведомо в Mongo Compas Ресторани в яких є європейська кухня і початок роботи починається раніше ніж 11:00. Результат зображено на рисунку 4.4.

```
db.restaurants.find({"features": "європейська кухня", "from": {$lt: "11:00"}})
```

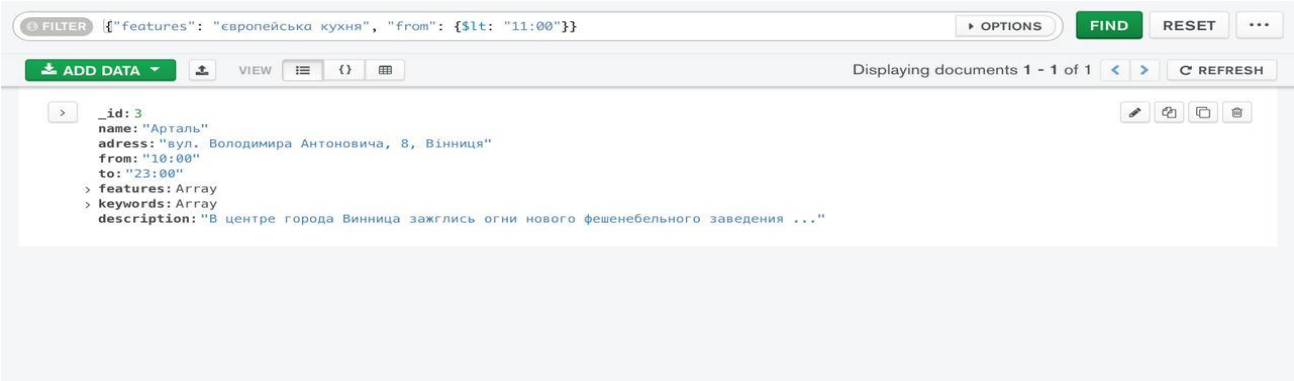

Рисунок 4.4 – Застосування декількох фільтрів

Отже, після заповнення Mongo db, проведемо тестування швидкості завантаження онлайн-сервісу. Швидкість завантаження з використанням іншого середовища зображено на рисунках 4.5 та 4.6 для комп'ютера та мобільного відповідно.

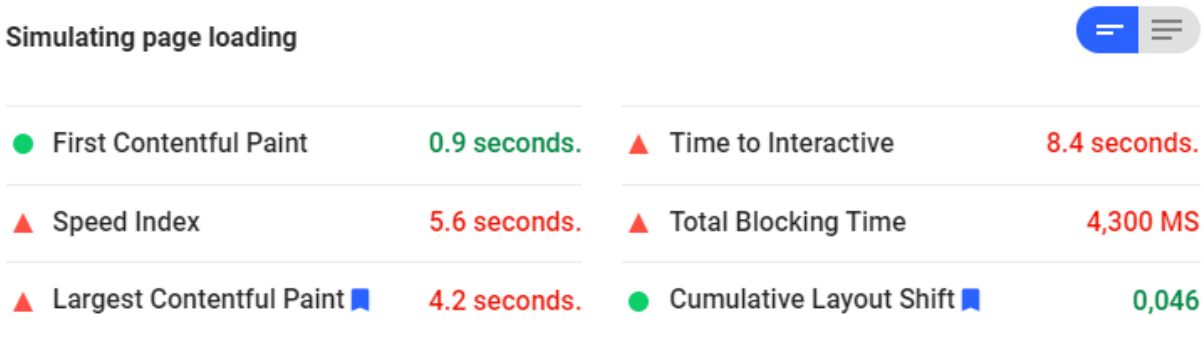

Рисунок 4.5 – Тестування швидкості завантаження онлайн-сервісу на комп'ютері з іншим середовищем

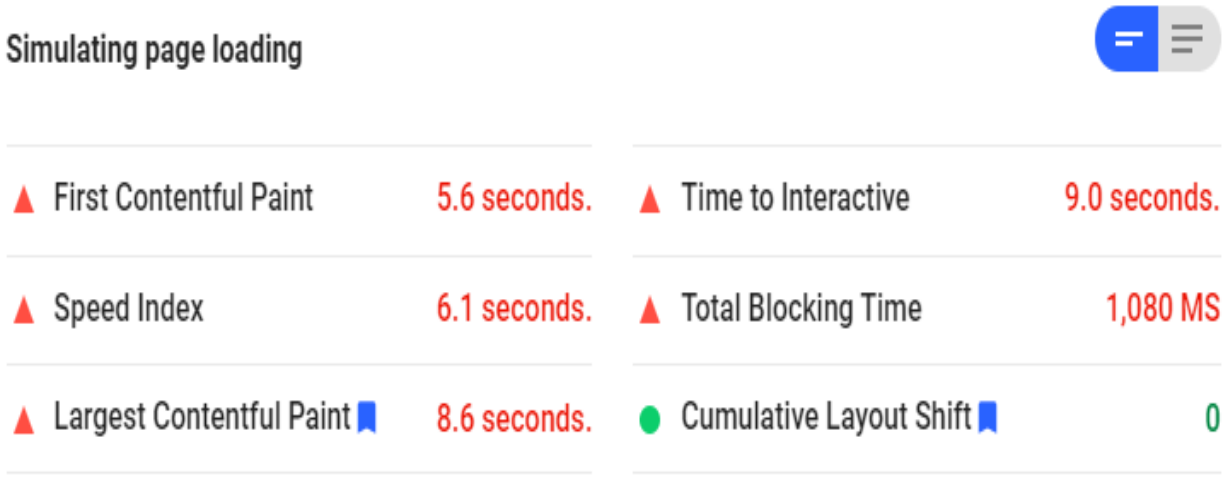

Рисунок 4.6 – Тестування швидкості завантаження онлайн-сервісу на телефоні з іншим середовищем

Тепер ж проведемо ті ж тестування але уже з використанням Mongo db, PWA та інших використаних технологій при розробці онлайн-сервісу. Швидкість завантаження онлайн-сервісу зображено на рисунках 4.7 та 4.8 для комп'ютера та мобільного відповідно.

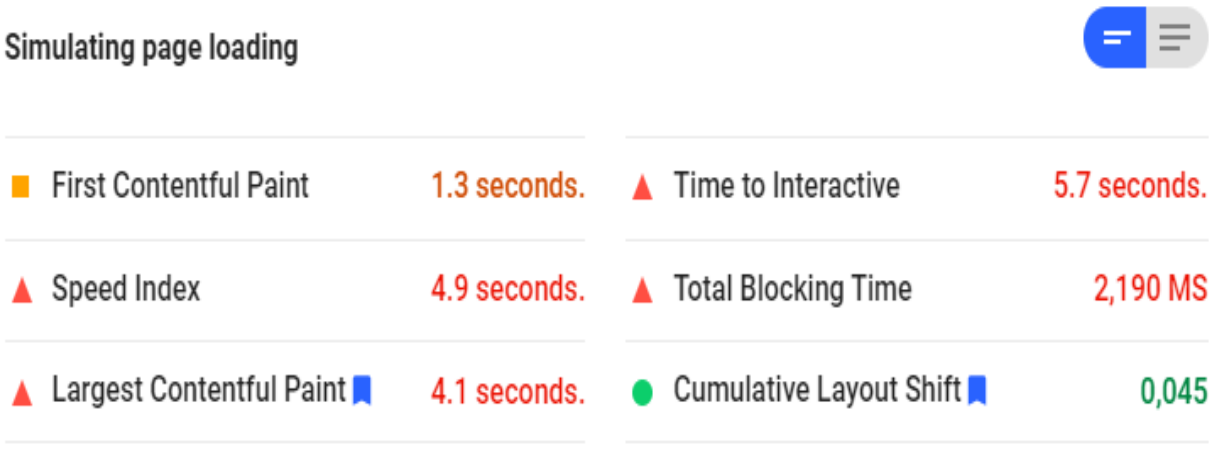

Рисунок 4.7 – Тестування швидкості завантаження онлайн-сервісу на телефоні

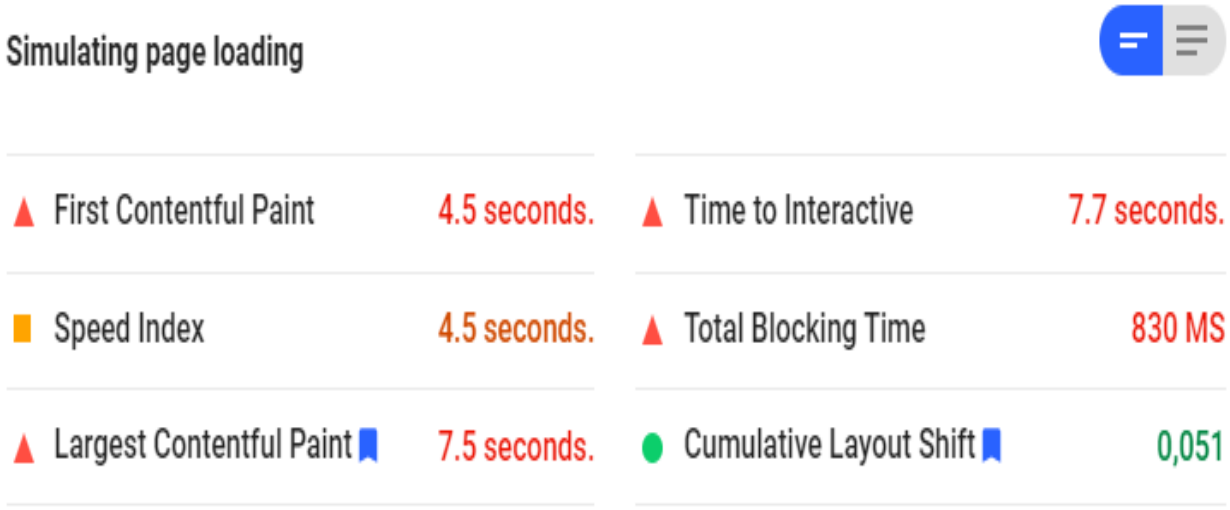

Рисунок 4.8 – Тестування швидкості завантаження
онлайн-сервісу на телефоні

First Contentful Paint (перше появлення контенту) – показник, який визначає інтервал часу між початком завантаження сторінки і появою першого зображення або блоку тексту.

Speed Index – (індекс швидкості завантаження показує), як швидко на сторінці з'являється контент.

Largest Contentful Paint (відображення великого контенту) – показник, який визначає час, необхідний на повне відображення великого тексту або зображення.

Time to Interactive (час завантаження для взаємодії) - це час, протягом якого сторінка стає повністю готова до взаємодії з користувачем.

Total Blocking Time – сума (в мілісекундах) всіх періодів від першого відтворення контенту до завантаження для взаємодії, коли швидкість виконання завдань перевищувала 50 мс.

Cumulative Layout Shift (процент зміщення макету) – це процентна величина, на яку зміщуються видимі елементи області перегляду при завантаженні.

Отже, після проведення тестування швидкості завантаження онлайн-сервісу було встановлено, що величина (Time to Interactive), що являє собою час завантаження онлайн-сервісу, при якому користувач може повністю взаємодіяти з ним була покращена на 2.7 секунд для завантаження на комп’ютері та 1.3 секунди для завантаження онлайн-сервісу на мобільному. Це означає, що мети роботи було досягнуто. Нажаль отримані дані мають певні похибки і можна побачити певні недоліки з погіршенням інших параметрів під час тестування, але основної мети було досягнуто.

4.3 Висновки

В четвертому розділі магістерської кваліфікаційної роботи виконано:

– проаналізовано методи тестування програми;

– вибрано метод тестування запитів та проведено його за допомогою Mongo Compas;

Як результат було обрано метод тестування серверної частини та протестовано основу серверної частини онлайн-сервісу.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Петришин С.І. та Яровий А.А.

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Петришин С.І.	2. Яровий А.А.
	Бали, виставлені експертами:	
1	3	4
2	4	4
3	3	3
4	4	3
5	3	4
6	4	3
7	3	3
8	4	4
9	3	4
10	4	4
11	4	4

12	4	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 43$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 4.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р = 21$ дні;

t - число днів роботи розробника, t = 90 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	5500	261.9	10	2619
Інженер-програміст	7000	333.33	90	29999,7
Всього:				32618,7

Розрахуємо додаткову заробітну плату:

$$З_{дод}= 0,1\cdot 32618,7=3261,87 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5-40% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп}= (З_о + З_р)\cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп}= (32618,7+3261,87)\cdot \frac{38.5}{100} = 13814 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А=\frac{Ц\cdot Н_а}{100}\cdot \frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
Персональний комп'ютер	15000	25	2	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n Н_і \cdot Ц_і \cdot К_і, \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Безпровідна мишка	шт.	350	1	350
Додатковий монітор	шт.	4000	1	4000
Флешка	шт.	200	1	150
Блокнот	шт.	130	1	90
Ручка	шт.	20	1	10
Всього з урахуванням транспортних витрат				5170

Витрати на силову електроенергію розраховуються за формулою:

$$В_е=В\cdot П\cdot Ф\cdot К_п ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1.5 грн/кВт);

Пк – установлена потужність комп’ютера (П=0,1кВт);

Пм – установлена потужність монітора (П=0,05кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф=600 год.);

$К_п$ – коефіцієнт використання потужності ($К_п< 1$, $К_п = 0,65$).

$$В_е= 1,5 \cdot 0,15 \cdot 600 \cdot 0,65 = 87.75 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (\mathbf{1}..\mathbf{3}) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = \mathbf{1.5} * (\mathbf{32618,7}+3261,87) = \mathbf{53820} \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

$$В = 32618,7+3261,87+\mathbf{13814} +625+5170+87.75+53820= 109397,32(грн.)$$

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\boldsymbol{\alpha}} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{\mathbf{109397,32}}{\mathbf{1}} = \mathbf{109397,32}$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{\mathbf{109397,32}}{\mathbf{0,85}} = \mathbf{128702,73} \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \qquad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки кількість ресторанів в онлайн-сервісі збільшиться на 75 процентів (що автоматично спричинить збільшення чистого прибутку підприємства на 50-75 процентів в залежності від вартості бронювання, далі будем використовувати середнє значення 62.5%), а кількість користувачів, які будуть користуватись онлайн-сервісом збільшиться: протягом першого року – на 250 користувачів, протягом другого року – на 300 користувачів, протягом третього року – 200 користувачів. Статистика ресторану до впровадження інформаційної технології складала 1000 активних клієнтів які в середньому приносили 350 гривень прибутку.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = (350 \cdot 0.625) \cdot 1000 + (340 \cdot 1.625) \cdot 250 = 356875 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = \Delta П_1 + (350 \cdot 1.625) \cdot 300 = 527500 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = \Delta П_2 + (340 \cdot 1.625) \cdot 200 = 641250 \text{ грн.}$$

Теперішню вартість інвестицій PV, що можуть бути вкладені в розроблену нами інтелектуальну систему, можна розрахувати за формулою:

$$PV = [(1…5) \times ЗВ] ,$$

де (1…5) – коефіцієнт, який враховує можливі додаткові витрати інвестора на можливе впровадження нашої розробки (оренда, підготовка персоналу, реклама тощо).

Для нашого випадку отримаємо:

$$PV = (1…5) \times ЗВ = 2.5 \times 128702{,}73 = 321756.8 \text{ (грн.)}.$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV),$$

(5.10)

де ПП – приведена вартість всіх можливих чистих прибутків від реалізації розробки, грн;

PV – теперішня вартість інвестицій, PV = 321756.8 грн.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_{1}^{m} \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{356875}{(1+0,1)^1} + \frac{527500}{(1+0,1)^2} + \frac{641250}{(1+0,1)^3} = 1242162.85 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$E_{абс} = 1242162.85 - 321756.8 = 920406.05 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$ за формулою:

$$E_в = \sqrt[T]{1 + \frac{E_{абс}}{PV}} - 1 \qquad (5.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

Т – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{920406.05}{321756.8}} - 1 = 0.57 \text{ або } 57\ \%$$

Далі, розраховану величина $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2020 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_в$ = 57% > $\tau_{мін}$ = 0,3 = 30%, то інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}$$

$$T_{ок} = \frac{1}{0.57} = 1{,}75 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

5.5 Висновок

В даному розділі було здійснено оцінювання комерційного потенціалу розробки інформаційної технології для розробки онлайн-сервісу для бронювання місць в ресторані.

Проведено технологічний аудит з залученням двох експертів. Аналіз експертних даних показав, що рівень комерційного потенціалу розробки вище середнього. Дослідження комерційного потенціалу розробки підтвердило, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною розробкою. Він має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальна вартість витрат на розробку і впровадження складає 128702,73 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 920406.05 грн свідчить про отримання прибутку інвестором від впровадження програмного продукту у діяльність підприємства.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 57%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 30%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності складає 1.75 року, що також свідчить про доцільність фінансування.

Усе це, узяте разом, забезпечує прийняття рішення про доцільність виготовлення нового продукту.

ВИСНОВКИ

Всі задачі, посавлені перед магістерською кваліфікаційною роботою виконано в повному об’ємі, а саме:

- обґрунтована доцільність створення онлайн-сервісу для бронювання столиків у ресторані;
- проаналізовані існуючі технології та методи створення серверної частини онлайн-сервісу;
- сформулювані вимоги до роботи технології та розроблене ТЗ;
- на основі існуючої моделі створення архітектури, розроблено архітектуру системи;
- розроблено алгоритми основних модулів системи та її структуру;
- проаналізовано основні технології для розробки серверної частини онлайн-сервісу та обрано найефективніший;
- проаналізовано основні технології тестування та проведено юзабіліті тестування системи;
- реалізовано та налаштовано роботу системи для бронювання столиків у ресторані;
- протестовано роботу системи
- виконано задачі економічного розділу.

Результати, одержані в процесі виконання магістерської кваліфікаційної роботи плануються до впровадження в розробки IT компанією «Delphi Software».

Метою роботи - розширення функціональних можливостей системи для бронювання місць в ресторані та покращення швидкості передачі даних між клієнтською частиною та серверною досягнуто шляхом розробки якісної серверної частини сервісу з використанням клієнтської частини, які найбільш краще взаємодіють між собою.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Node.js [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Node.js

2. PHP [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/PHP

3. Python [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

4. Введення в протокол HTTP [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

5. Розробка веб сайтів: динамічний веб сайт [Електронний ресурс] – Режим доступу: http://strong-planet.com.ua/dynamic_site.html

6. Поняття клієнт-серверної технології. Що таке технологія клієнт-сервер [Електронний ресурс] – Режим доступу: https://passportbdd.ru/uk/rabota-v-internete/ponyatie-klient-servernoi-tehnologii-chto-takoe-tehnologiya-klient-server/

7. Руководство для начинающик HTTP і REST [Електронний ресурс] – Режим доступу: https://code.tutsplus.com/ru/tutorials/a-beginners-guide-to-http-and-rest--net-16340

8. Руководство по веб-разработке серверной части с помощью Node.js [Електронний ресурс] – Режим доступу: https://webformyself.com/rukovodstvo-po-veb-razrabotke-s-node-js/

9. What is REST [Електронний ресурс] – Режим доступу: https://restfulapi.net/

10. SQL чи NoSQL – Режим доступу: https://habr.com/ru/company/ruvds/blog/324936/

11. SQL vs NoSQL –Режим доступу: https://tproger.ru/translations/sql-vs-nosql/

12. Руководство по MongoDB. Моделирование данных [Електронний ресурс] – Режим доступу: https://proselyte.net/tutorials/mongodb/data_modeling/

13. Клиент серверная архитектура в картинках [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/495698/

14. Node.js – Режим доступу: https://uk.wikipedia.org/wiki/Node.js

15. Програмування на стороні сервера – Режим доступу: https://en.wikipedia.org/wiki/Server-side_scripting