**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ВОЛОДИМИРА ДАЛЯ**
**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ**
**КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ**

До захисту допускається
Завідувач кафедри ІТП
д.т.н., доц. Лифар В.О.
_____
«____»_____ 2023 р.

**ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА**

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

НА ТЕМУ:

# Цифрові технології в управлінсько-робочому середовищі ресторанного бізнесу

Освітній ступінь “бакалавр”
Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Керівник проекту: _____ (підпис) Іванов В. Г. (ініціали, прізвище)

Здобувач вищої освіти: _____ (підпис) А.В. Крайній (ініціали, прізвище)

Група: ІПЗ-19д

Київ 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Інформаційних технологій та програмування
Освітній ступінь бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)

**ЗАТВЕРДЖУЮ:**

Завідувач кафедри ІТП
_____ В.О. Лифар
«_____» _____2023 р.

**З А В Д А Н Н Я**
**НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) БАКАЛАВРА**

Крайній Антон Валерійович
(прізвище, ім'я, по батькові)

1. Тема роботи Цифрові технології в управлінсько- робочому середовищі ресторанного бізнесу,

керівник проекту (роботи) Іванов Віталій Геннадійович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "26" 04. 2023 р. № 240/15.15-ОД
2. Термін подання роботи здобувача вищої освіти 11.06.2023
3. Вихідні дані до роботи матеріали переддипломної практики, альтернативний додаток на місці роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області та постановка задачі
2) Вибір стеку технологій розробки
3) Опис процесу створення додатку
4) Охорона праці
5) Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
електронні плакати.

6. Консультанти розділів проекту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 24.03.2023 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів дипломного проекту (роботи) | Строк виконання етапів проекту ( роботи ) | Примітка |
|---|---|---|---|
| 1 | Огляд та аналіз вимог до роботи | 25.03.2023-01.04.2023 | |
| 2 | Дослідження та аналіз існуючих веб-додатків для ресторанів | 02.04.2023-16.04.2023 | |
| 3 | Постановка технічного завдання | 17.04.2023-22.04.2023 | |
| 4 | Розробка серверної архітектури з використанням Node.js | 23.04.2023-07.05.2023 | |
| 5 | Реалізація клієнтського інтерфейсу з використанням Vue.js | 08.05.2023-27.05.2023 | |
| 6 | Оформлення пояснювальної записки та презентації | 28.05.2023-10.06.2023 | |

**Здобувач вищої освіти** _____ **А.В. Крайній**
( підпис ) (ініціали, прізвище)

**Керівник** _____ **В.Г. Іванов**
( підпис ) (ініціали, прізвище)

# РЕФЕРАТ

Пояснювальна записка до дипломного проекту бакалавра: 86 с., 15 рис., 25 бібліографічних джерел, 2 додатки.

Об’єкт дослідження: процес створення веб-додатку для мережі ресторанів на сучасному стеку технологій.

Мета роботи: полегшити розробку та підтримку веб-додатків за допомогою сучасних зручних архітектурних рішень.

В проекті виконано:

– аналіз архітектурних рішень;

– аналіз стеків розробки та платформ розробки;

– розроблено додаток для мережі ресторанів з використанням обраного стеку технологій;

Отримано наступні результати: доведена доцільність впровадження пропонованого стеку технологій розробки; розроблено додаток за техничним завданням.

Практичне значення, галузь застосування роботи: мережи ресторанів, системи, де необхідно розподілити ролі між персоналом та звичайними користувачами; результати роботи можуть застосовуватися при розробці сторонніх проектів.

Ключові слова: Vue.js, Node.js, Express.js, WEB, ФРОНТЕНТ, БЕКЕНД, СЕРВЕР, БАЗИ ДАНИХ.

# ЗМІСТ

# ВСТУП

Інтернет та всесвітня павутина революціонізували спосіб ведення бізнесу, особливо у сфері послуг. У минулому бізнес був обмежений традиційними методами реклами, такими як друковані видання, радіо та телебачення. Однак, з появою Інтернету, бізнес може охопити набагато ширшу аудиторію за допомогою онлайн-реклами та маркетингу. Це полегшило підприємствам сфери послуг просування своїх послуг, залучення нових клієнтів і розвиток бізнесу. У наш час потужна присутність в Інтернеті має вирішальне значення для успіху будь-якого бізнесу, а інтернет спростив досягнення цієї мети як ніколи раніше.

Під потужною присутністю в інтернеті мається на увазі наявність сайта, аккаунти в соц. мережах, присутність на загальнодоступних мапах з можливістю зворотнього зв’язку для клієнтів.

Заклади харчування, такі як кафе та ресторани, почали створювати власні веб-сайти в середині 1990-х років, невдовзі після того, як Всесвітня павутина стала загальнодоступною. Спочатку ці веб-сайти були простими і статичними, надаючи основну інформацію, таку як місцезнаходження ресторану, години роботи та пункти меню. З розвитком Інтернету веб-сайти ресторанів стали більш досконалими, з'явилися системи онлайн-замовлення, резервування місць та відгуки клієнтів.

Сьогодні наявність веб-сайту є майже необхідністю для закладів харчування, оскільки він дозволяє клієнтам легко шукати та дізнаватися про ресторан, переглядати меню, а також здійснювати бронювання або замовлення онлайн. Багато ресторанів також використовують соціальні мережі для просування свого бізнесу та взаємодії з клієнтами. Інтернет став найважливішим інструментом для харчової промисловості, і важко уявити успішний ресторан чи кафе без присутності в Інтернеті.

З розвитком інтернету змінювалися і стратегії закладів харчування. Соціальні медіа-платформи, такі як Facebook і Twitter, стали популярними, і ресторани почали використовувати їх для взаємодії з клієнтами та формування лояльності до бренду. Також з'явилися сайти онлайн-відгуків, такі як Yelp і TripAdvisor, які надали

клієнтам платформу для обміну досвідом і думками про ресторани з глобальною аудиторією.

Розвиток мобільних технологій також мав значний вплив на харчову промисловість. Коли все більше людей почали користуватися смартфонами та планшетами, ресторани почали розробляти мобільні додатки, які полегшили клієнтам замовлення їжі, бронювання столиків та доступ до програм лояльності.

Сьогодні заклади харчування освоїли інтернет у багатьох відношеннях. Вони використовують складні алгоритми та аналітику даних, щоб націлювати клієнтів на персоналізовані акції та рекламу, а багато хто навіть впровадив штучний інтелект і чат-ботів для забезпечення підтримки клієнтів та покращення їхнього досвіду.

Загалом, освоєння харчовою промисловістю інтернету дозволило їм охопити більше клієнтів, збільшити продажі та побудувати міцніші стосунки зі своїми клієнтами завдяки персоналізованій взаємодії та досвіду.

В цій роботі буде розроблено WEB-застосунок для ресторану, в якому буде показаний сучасний приклад використання сайтів для ресторану.

Опис ідеї даної роботи є створення WEB-сайту, в якому будуть розподілені ролі: для клієнтів та для персоналу закладу і все це буде розроблено за допомогою сучасного стеку технологій.

Для реалізацій даного додатка було використано середу розробки WebStorm, мова розробки JavaScript з додатковими бібліотеками такими як Vue.js. Для запуску локального сервера використовується платформа NodeJS.

Для досягнення цієї мети в роботі сформульовані й вирішені наступні завдання:

1. Аналіз технологій WEB-розробки та розробки користувацького інтерфейсу;
2. Робота з базою даних;
3. Визначено набір інструментів, що використовується в розробці;
4. Зв’язка баз даних з інтерфейсом;
5. Тестування роботи;

Проаналізовано сайти інших харчових закладів, рівно як і технології розробки, за основу був обраний додаток ресторану, в якому я працюю та створено його альтернативу.

Основні переваги запропонованого рішення є в простоті й зрозумілості використання та ефективності розробки, а також можливості інтеграції з іншими сервісами.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Кратка історія створення та розвитку WEB

Всесвітня павутина (World Wide Web, WWW), широко відома як Інтернет, – це інформаційна система, що забезпечує доступ до документів та інших веб-ресурсів через Інтернет відповідно до певних правил, протоколу передачі гіпертексту (Hypertext Transfer Protocol, HTTP).

Варто зазначити, що технологія WEB не має сенсу без комп’ютерних мереж, як Інтернет, наприклад. Але це не означає, що обмін інформацією в мережі можливий тільки через всесвітню павутину, тому помилково вважати що Інтернет-це те що і WEB.

Історія Всесвітньої павутини (WWW або "павутини") почалася в 1989 році, коли британський вчений Тім Бернерс-Лі винайшов павутину під час роботи в ЦЕРН, Європейській організації з ядерних досліджень. Метою Бернерса-Лі було створити спосіб для вчених і дослідників легко обмінюватися інформацією та отримувати до неї доступ через різні комп'ютерні системи.

До появи Всесвітньої павутини (WWW) люди використовували інші засоби для обміну інформацією в Інтернеті. Одним з популярних способів був Usenet – розподілена дискусійна система, яка дозволяла користувачам розміщувати і читати повідомлення в різних групах новин. Usenet був організований ієрархічно, з групами новин, що охоплювали широкий спектр тем, від політики до технологій і хобі.

Іншим методом обміну інформацією була електронна пошта, яка передувала WWW на кілька десятиліть. Електронна пошта дозволяла користувачам надсилати повідомлення один одному, як індивідуально, так і в групах, і часто використовувалася як для особистого, так і для професійного спілкування.

Крім того, існували системи дошок оголошень (BBS), які були комп'ютерними системами, що дозволяли користувачам з'єднуватися та обмінюватися інформацією. BBS, як правило, управлялися окремими особами або невеликими групами і часто були зосереджені на конкретних темах або інтересах.

Загалом, хоча методи обміну інформацією в Інтернеті до появи WWW були більш обмеженими за обсягом і функціональністю, вони все ще дозволяли жваво обмінюватися ідеями та інформацією між користувачами.

Чого не вистачало цим технологіям, так це стандартизованого способу доступу до інформації та обміну нею у графічному та зручному для користувача вигляді. Саме тому Тім Бернерс-Лі створив WWW. WWW забезпечив графічний інтерфейс, який дозволив користувачам легко орієнтуватися і отримувати доступ до інформації на різних комп'ютерах і в різних мережах. Крім того, WWW використовував стандартизований протокол (HTTP) і мову розмітки (HTML), що дозволило легко обмінюватися інформацією та отримувати до неї доступ незалежно від типу використовуваного комп'ютера або мережі. Ці інновації значно спростили обмін інформацією та доступ до неї в глобальному масштабі, проклавши шлях до сучасного Інтернету, який ми знаємо сьогодні.

Перша веб-сторінка була створена в 1991 році, а до середини 1990-х років Інтернет став популярним інструментом для доступу та обміну інформацією. Розвиток веб-браузерів, таких як Mosaic і Netscape Navigator, полегшив людям перегляд і навігацію в Інтернеті.

На цьому етапі варто розділити історію на декілька гілок: розвиток протоколу та супутніх технологій, розвиток браузерів та індустрії в цілому (рис. 1.1)

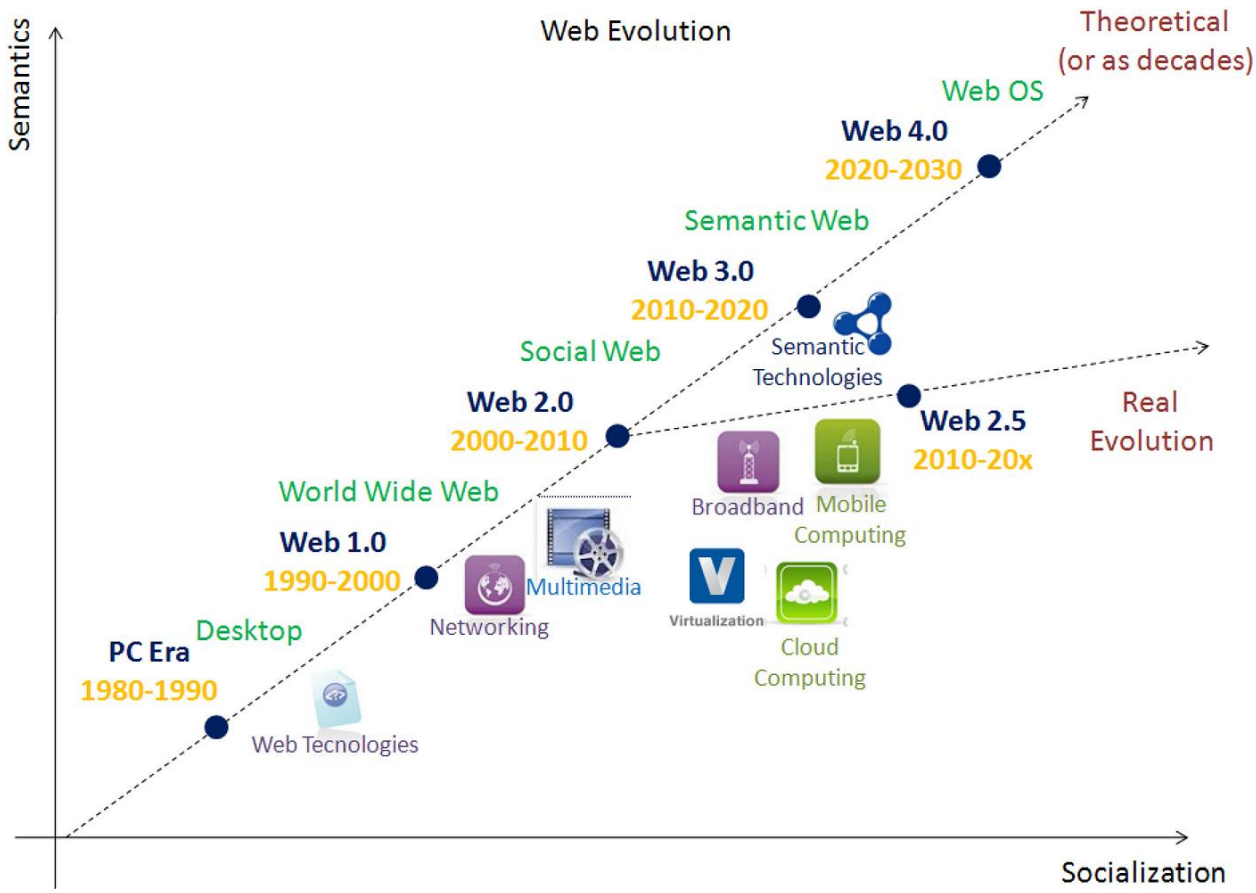



Рисунок. 1.1 – Еволюційний шлях WEB

Всесвітня павутина  зазнала значної еволюції з моменту свого створення на початку 1990-х років. Перша версія павутини, відома як Веб 1.0, характеризувалася статичними HTML-сторінками та обмеженою інтерактивністю і використовувалася здебільшого для публікації інформації, а не для інтерактивних додатків.

У середині 2000-х років з'явився Веб 2.0 – нова парадигма, яка зробила акцент на динамічному контенті та контенті, створеному користувачами, наприклад, на сайтах соціальних мереж, таких як Facebook, Twitter та Instagram. З поширенням смартфонів веб-дизайнери почали надавати перевагу мобільному дизайну, який пристосовується до менших екранів.

Власне, цим шляхом і пішов WEB, всупереч коцепції  Semantic Web.

Семантична павутина – це розширення Всесвітньої павутини, яке дозволяє машинам розуміти та інтерпретувати значення (або "семантику") інформації в Інтернеті. Метою семантичного павутиння є створення більш інтелектуальної,

ефективної та автоматизованої мережі, що дозволяє обмінюватися даними та повторно використовувати їх у різних додатках.

На традиційних веб-сторінках контент призначений, в першу чергу, для споживання людиною. Однак у середовищі семантичного ВЕБу дані структуруються за допомогою стандартизованих форматів, таких як RDF (Resource Description Framework) або OWL (Web Ontology Language), які дозволяють комп'ютерам обробляти їх і міркувати про них.

Семантичний веб надає багато переваг, таких як отримання більш точних результатів пошуку; автоматизація таких завдань, як планування зустрічей або бронювання квитків; полегшення пошуку знань з великих масивів даних за допомогою інтеграції даних; покращення процесів прийняття рішень завдяки швидшому доступу до релевантної інформації. Але зараз ця концепція тільки в планах. Можливо, буде прискорено втілення цієї концепції з можливістю моделей штучного інтелекту шукати інформацію в інтернеті.

### 1.1.1 Розвиток протоколу HTTP

Протокол передачі гіпертексту (HTTP) – це протокол прикладного рівня в моделі набору протоколів Інтернету для розподілених, спільних, гіпермедійних інформаційних систем. HTTP є основою передачі даних для Всесвітньої павутини, де гіпертекстові документи містять гіперпосилання на інші ресурси.

Але HTTP – це один з багатьох Інтернет-протоколів прикладного рівня, саме тому помилково вважати, що WEB і є Інтернетом.

HTTP є фундаментом технології WEB. Розробка HTTP була ініційована Тімом Бернерсом-Лі в ЦЕРНі в 1989 році і підсумована в простому документі, що описує поведінку клієнта і сервера, використовуючи першу версію HTTP, названу 0.9. Згодом ця версія була вдосконалена, і врешті-решт стала загальнодоступною версією 1.0. Але HTTP/1 був завершений і повністю задокументований тільки в 1996 р. Але на цьому розвиток протоколу не зупинявся, тому в 1997 році вийшло оновлення 1.1, яке досі використовується та не є застарілим, а потім його

специфікації оновлювалися у 1999, 2014 та 2022 роках, що говорить про актуальність протоколу.

Варто зазначити, що для більшого розуміння документації протоколу та його змін неможливо без огляду на Request for Comments (RFC).

RFC — документ із серії пронумерованих інформаційних документів Інтернету, що містить технічні специфікації та Стандарти, має широке застосування у всесвітній мережі. Назву «Request for Comments» ще можна перекласти як «заявка на обговорення» чи «тема для обговорення». Зараз публікацією документів RFC займається IETF під егідою відкритої організації Товариство Інтернету (ISOC). Тому історія зміни протоколу розписана по таким розділам RFC:

- RFC 1945 HTTP/1.0
- RFC 9110 HTTP Semantics
- RFC 9111 HTTP Caching
- RFC 9112 HTTP/1.1
- RFC 9113 HTTP/2
- RFC 7541 HTTP/2: HPACK Header Compression
- RFC 8164 HTTP/2: Opportunistic Security for HTTP/2
- RFC 8336 HTTP/2: The ORIGIN HTTP/2 Frame
- RFC 8441 HTTP/2: Bootstrapping WebSockets with HTTP/2
- RFC 9114 HTTP/3
- RFC 9204 HTTP/3: QPACK: Field Compression

Якщо пройтись по цьому списку, то можна кратко виділити такий шлях розвитку протоколу.

HTTP/0.9 Підтримував лише метод GET для отримання документів і не підтримував жодних заголовків.

В HTTP/1.0 вже додано підтримку методу POST для надсилання форм, а також було введено поняття заголовків. Заголовки використовуються для додавання додаткової інформації до повідомлень (метадані).

В HTTP/1.1 було додано ряд нових функцій, включаючи постійні з'єднання, кешування і конвеєрну обробку. Постійні з'єднання дозволяють надсилати декілька запитів через одне і те ж з'єднання. Кешування дозволяє веб-браузерам зберігати копії веб-сторінок на локальних жорстких дисках, щоб не завантажувати їх з сервера при кожному зверненні до них. Конвеєризація дозволяє надсилати на сервер кілька запитів одночасно.

HTTP/2 був випущений у 2015 році. Це велика ревізія протоколу HTTP, яка вводить ряд нових функцій, включаючи стиснення заголовків, мультиплексування і серверне підштовхування. Стиснення заголовків зменшує розмір заголовків за допомогою спеціального алгоритму стиснення. Мультиплексування дозволяє одночасно надсилати декілька запитів через одне з'єднання. Технологія Server push дозволяє серверу надсилати ресурси клієнту до того, як клієнт їх запитає.

HTTP/3 – це остання версія HTTP, яку стандартизували у 2022 році. HTTP/3 базується на протоколі QUIC, який є новим протоколом транспортного рівня, призначеним для підвищення продуктивності HTTP. QUIC використовує UDP замість TCP в якості базового транспортного протоколу, що дозволяє уникнути деяких обмежень TCP.

Протокол HTTP протягом багатьох років стикався з низкою проблем безпеки. Однією з найсерйозніших проблем була атака BEAST, яка була виявлена в 2011 році. Атака BEAST дозволяла зловмисникам викрадати конфіденційну інформацію, таку як файли cookie та паролі, з веб-браузерів через незашифроване HTTP-з'єднання.

Атака BEAST працювала, використовуючи вразливість в тому, як HTTP обробляє файли cookie. Файли cookie - це невеликі файли, які зберігаються на комп'ютері користувача веб-сервером. Файли cookie використовуються для відстеження користувачів на різних сторінках веб-сайту, а також для зберігання такої інформації, як облікові дані для входу в систему.

Ця вразливість була серйозною проблемою безпеки, і вона вплинула на всі основні веб-браузери. Однак проблема була вирішена завдяки впровадженню HTTPS. HTTPS – це безпечна версія HTTP, яка використовує шифрування для

захисту даних під час передачі. Це значно ускладнює зловмисникам викрадення даних за допомогою атак на кшталт BEAST.

Сьогодні більшість веб-сайтів використовують HTTPS, і атака BEAST більше не є серйозною загрозою. Однак важливо знати про ризики безпеки, пов'язані з HTTP, і завжди використовувати HTTPS, коли це можливо.

### 1.1.2 Розвиток браузерів

Браузер — програмне забезпечення для комп'ютера або іншого електронного пристрою, що дає можливість користувачеві взаємодіяти з текстом, малюнками або іншою інформацією на гіпертекстовій веб-сторінці.

Перший веб-браузер під назвою WorldWideWeb був створений у 1990 році сером Тімом Бернерсом-Лі. Потім він залучив Ніколу Пеллоу до написання браузера Line Mode Browser, який відображав веб-сторінки на німих терміналах (рис. 1.2).

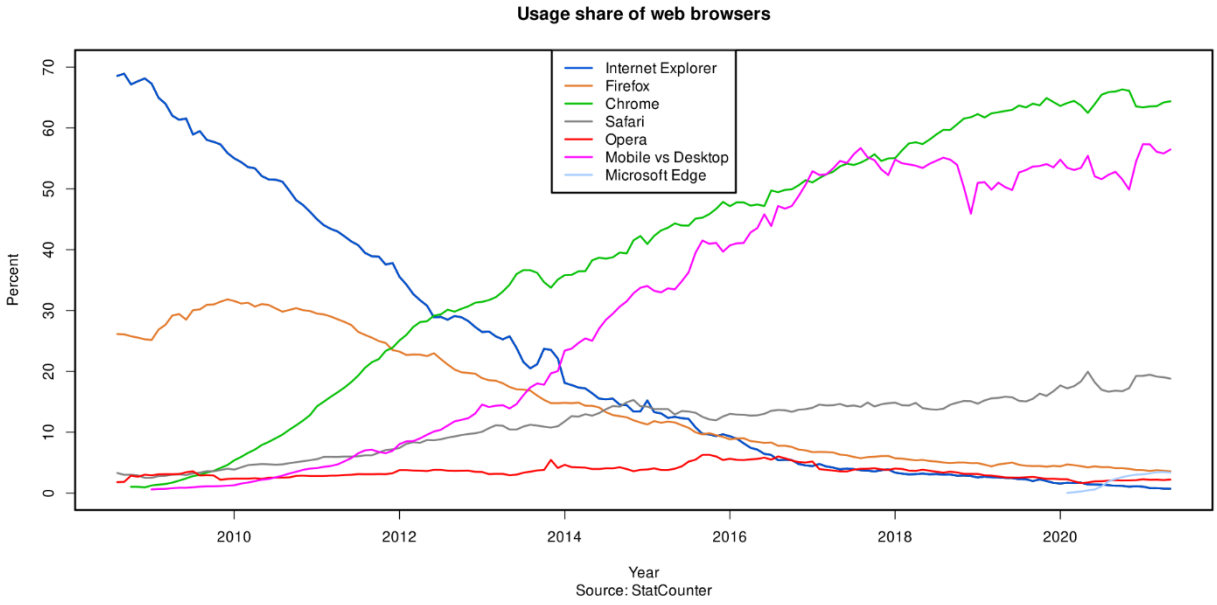



Рис. 1.2 – Історія використання сучасних браузерів

Веб-браузер Mosaic був випущений у квітні 1993 року і згодом став першим веб-браузером, який здобув широку популярність. Його інноваційний графічний інтерфейс користувача зробив Інтернет легким для навігації і, таким чином, набагато візуально привабливішим і доступнішим для ширшої аудиторії. Це, в свою

чергу, спричинило інтернет-бум 1990-х років, коли Мережа зростала дуже швидкими темпами. Mosaic став великим проривом, оскільки це був перший браузер, який підтримував зображення та інший мультимедійний контент (рис.1.3).

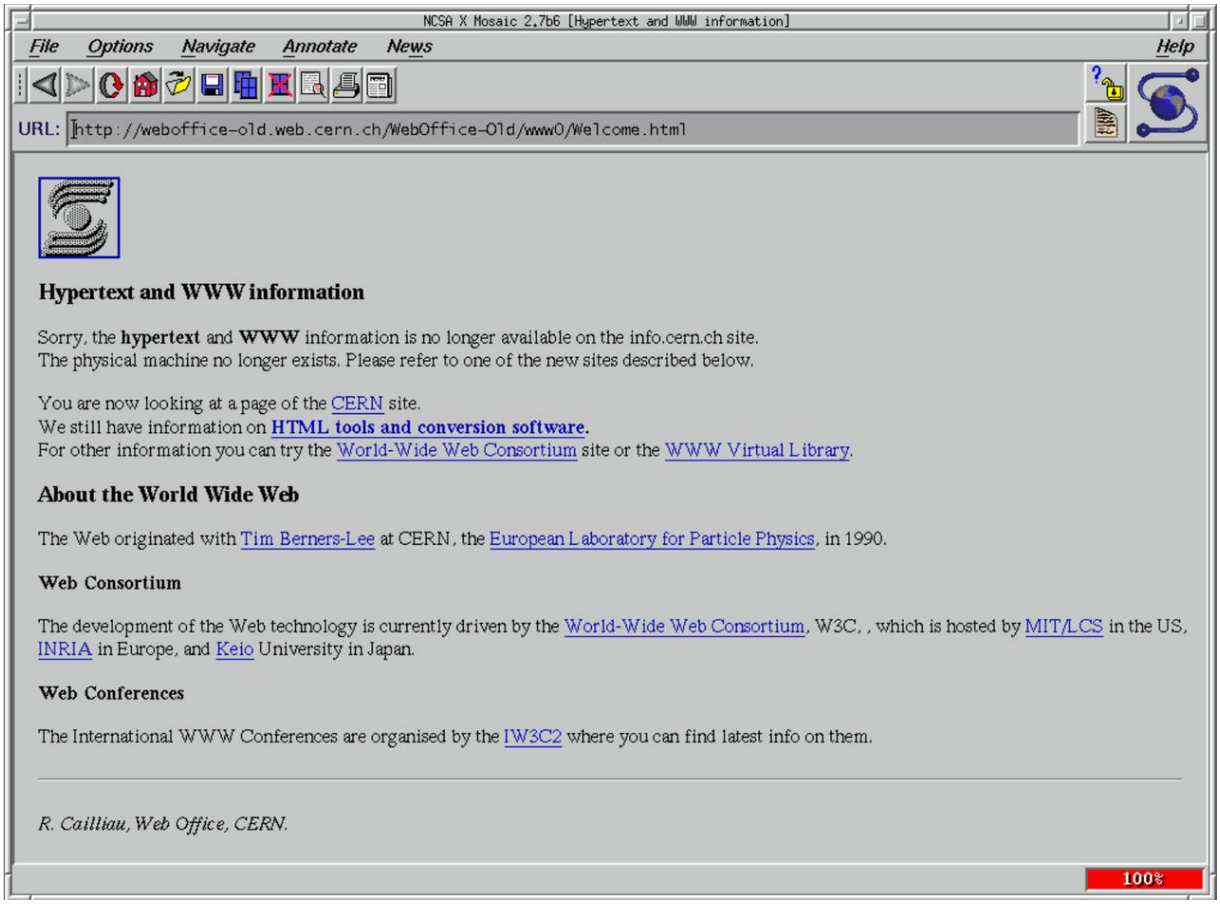



Рис. 1.3 – Інтерфейс браузера Mosaic версії 2.7b6

Марк Андрессен, лідер команди Mosaic, заснував власну компанію Netscape, яка в 1994 році випустила навігатор Netscape Navigator під впливом Mosaic. Navigator швидко став найпопулярнішим браузером. Netscape Navigator швидко став домінуючим веб-браузером. Netscape Navigator базувався на Mosaic, але до нього було додано низку нових функцій і вдосконалень. Наприклад, Netscape Navigator був першим браузером, який підтримував JavaScript – мову програмування, що дозволяє веб-сторінкам бути більш інтерактивними.

Однак вже 1995 році Microsoft випустила Internet Explorer 1.0. Спочатку Internet Explorer не був дуже популярним, але швидко завоював популярність

завдяки агресивній маркетинговій кампанії Microsoft. До 1998 року Internet Explorer випередив Netscape Navigator як найпопулярніший веб-браузер.

У 2004 році вийшов Mozilla Firefox. Firefox був безкоштовним браузером з відкритим вихідним кодом, який швидко завоював популярність завдяки своїй швидкості, безпеці та простоті використання. Firefox кинув виклик домінуванню Internet Explorer, і з тих пір ці два браузери ведуть запеклу боротьбу за частку ринку.

У 2008 році вийшов Google Chrome. Chrome був новим веб-браузером, який був розроблений, щоб бути швидким та ефективним. Chrome швидко завоював популярність, і зараз він є другим за популярністю веб-браузером у світі.

Сьогодні існує ціла низка різних веб-браузерів, кожен з яких має свої сильні та слабкі сторони. Найпопулярнішими веб-браузерами є Chrome, та Safari на девайсах компанії Apple. Динаміка популярності браузерів постійно змінюється, і на неї впливає низка факторів, серед яких випуск нових і вдосконалених браузерів, впровадження нових технологій і маркетингові зусилля виробників браузерів.

Випуск нових і вдосконалених браузерів є одним з найбільш значущих факторів, який може вплинути на популярність браузерів. Наприклад, випуск Chrome у 2008 році призвів до значного зниження популярності Internet Explorer. І справа була не тільки в маркетинговій стратегії браузерів, а і в впровадженні нових технологій, які покращували роботу браузерів. Наприклад, ключовою подією став вихід HTML5 та CSS3, що призвело до зниження популярності старих браузерів, які його не підтримують.

### 1.1.3 Розвиток індустрії в цілому

Технологічні інновації відіграли важливу роль в еволюції веб-сайтів. Розвиток нових технологій, таких як HTML, CSS та JavaScript, уможливив створення більш складних та інтерактивних веб-сайтів. Наприклад, використання HTML дозволяє веб-розробникам створювати веб-сторінки, які можна форматувати і стилізувати за допомогою CSS. JavaScript, з іншого боку, дозволяє веб-розробникам додавати інтерактивності веб-сторінкам.

Потреби бізнесу також відіграли певну роль в еволюції веб-сайтів. Компанії все більше усвідомлювали цінність наявності веб-сайту та інвестували в розробку більш досконалих веб-сайтів. Наприклад, багато компаній зараз використовують свої веб-сайти для продажу товарів чи послуг, надання підтримки клієнтам або найму нових працівників.

Очікування користувачів також відіграли певну роль в еволюції веб-сайтів. Користувачі почали очікувати від веб-сайтів більшого як з точки зору контенту, так і з точки зору функціональності. Наприклад, зараз користувачі очікують, що веб-сайти будуть простими у використанні, візуально привабливими та надаватимуть актуальну інформацію.

Була й певна криза, пов’язана зі стрімким ростом WEB, як, наприклад т.з. бульбашка доткомів.

Бульбашка доткомів – це період економічних спекуляцій та інвестиційного шаленства навколо компаній, пов'язаних з Інтернетом, наприкінці 1990-х років. У цей час ціни на акції компаній, пов'язаних з Інтернетом, стрімко зростали, досягнувши нестійкого рівня. Врешті-решт бульбашка луснула у 2000 році, і багато компаній, які були на висоті під час буму, збанкрутували.

Була низка обставин, які сприяли виникненню бульбашки доткомів. Одним з них було швидке зростання самого Інтернету. Наприкінці 1990-х років Інтернет був все ще відносно новою технологією, і його потенціал викликав великий ажіотаж. Цей ажіотаж призвів до великої кількості інвестицій у компанії, пов'язані з Інтернетом. Тому компанії просто робили собі вебсайти, частіше за усього доменні імена яких закінчувались на «.com». І в той період інвестори в першу чергу звертали на це увагу та придавали цьому завищене значення, навіть якщо компанія насправді, нічого не робила та вкладали свої кошти.

Але навіть коли ця бульбашки доткомів «луснула» це не призвело до падіння цієї індустрії.

З середини 90-х років, коли домашні комп'ютери стали доступнішими, а доступ до Інтернету швидко поширився серед домогосподарств у розвинених країнах, кількість веб-сайтів почала зростати з вражаючою швидкістю. За даними

Internet Live Stats станом на вересень 2021 року в інтернеті налічувалося понад 1,8 мільярда активних веб-сайтів (рис. 1.4).

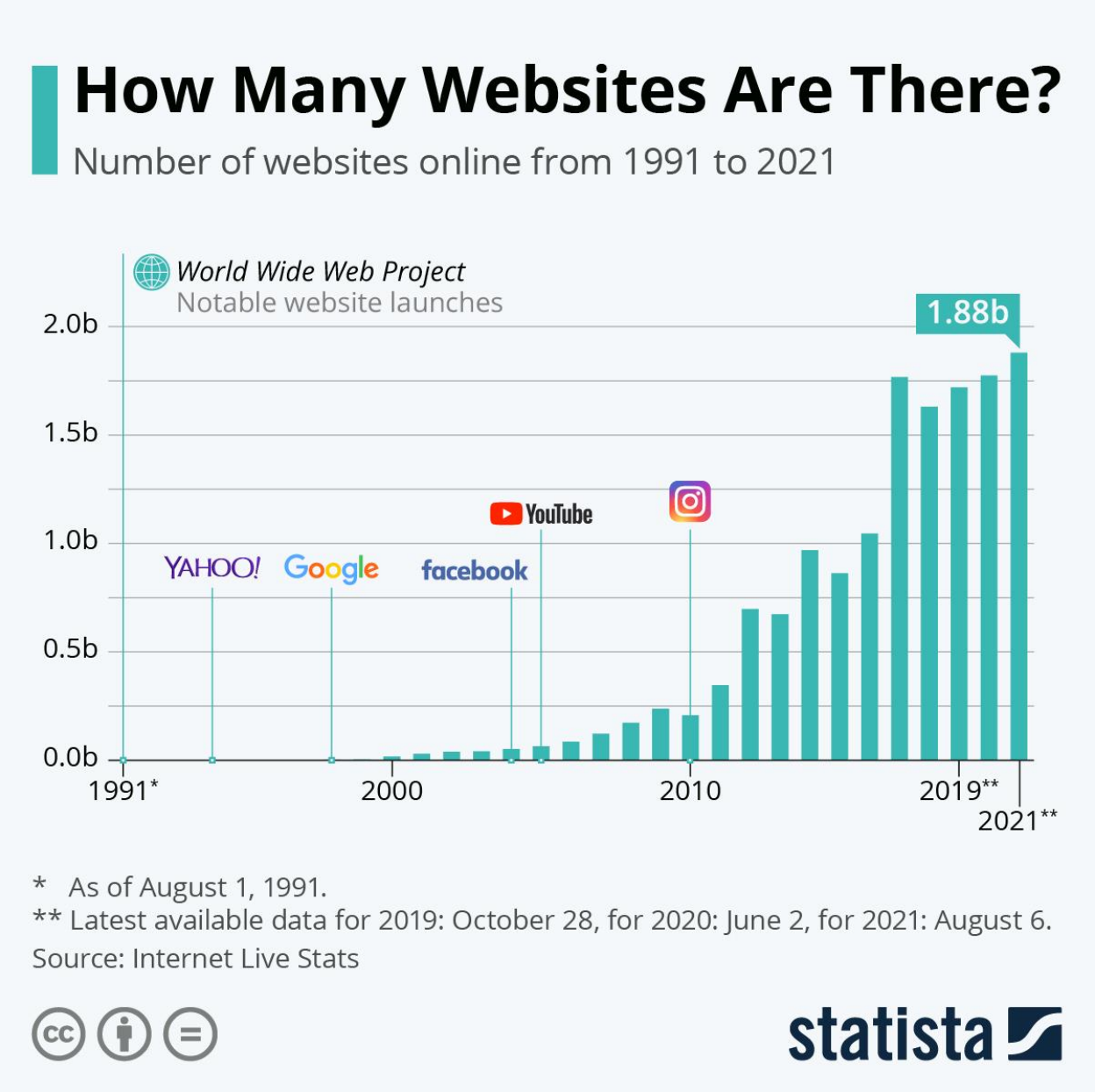



Рис. 1.4 – Зріст кількості активних вебсайтів

З цим вибуховим зростанням кількості веб-сайтів з'явилися нові можливості для бізнесу - сайти електронної комерції, такі як Amazon та eBay, вперше з'явилися наприкінці 90-х років, що дозволило споживачам купувати товари онлайн з будь-якої точки світу. Ця тенденція продовжилася на початку 2000-х років, коли такі компанії, як Google (заснована в 1998 році), почали пропонувати пошукові сервіси,

завдяки яким користувачам стало простіше, ніж будь-коли раніше, знаходити те, що вони шукали в Інтернеті.

Оскільки мобільні пристрої стали повсюдним явищем у цей період (смартфони набувають все більшого поширення з моменту випуску компанією Apple першої моделі iPhone у червні 2007 року), веб-розробники почали адаптувати свій дизайн так, щоб сторінки можна було переглядати на менших екранах; методи адаптивного дизайну тепер стали стандартною практикою.

Зовсім недавно ми стали свідками значних змін, таких як послуги хмарних обчислень, що пропонуються такими провайдерами, як Amazon Web Services або Microsoft Azure, які дозволяють компаніям будь-якого розміру отримати доступ до потужних обчислювальних ресурсів без необхідності вкладати значні кошти в апаратну інфраструктуру.

Загалом ці зміни відображають ширші тенденції до зростання цифровізації сучасного суспільства у всьому світі: згідно з дослідженням Statista, у січні 2021 року в усьому світі налічувалося 4,66 мільярда активних користувачів Інтернету – це більше половини населення планети. Отже, ринок веб-технологій експоненціально зростав з моменту свого зародження і продовжує розвиватися швидкими темпами, щороку з'являються нові інновації.

### 1.2. Технічне завдання на дипломну роботу

Необхідно розробити WEB-додаток для мережі ресторанів, на стеку технологій Node.js для забезпечення серверної архітектури та роботи з базою даних, а також Vue.js для впровадження користувацького інтерфейсу. Базу даних використовувати MongoBD. Має бути забезпечено API для можливості інтеграції з іншими сервісами для бізнесу. Користувач матиме можливість замолювати столик, обирати меню та дивитися рахунок. Додаток матиме такі сутності як: користувачі, меню, продукти, замовлення, статуси та рахунки.

Перед тим, як прийти в заклад, клієнт може зарезервувати столик заздалегідь та обрати меню. В меню може обрати блюда, час свого прибуття та час початку

приготування блюда. Замовлення передається співробітникам кухні, які мають підтвердити, що вони почали готувати блюдо. Тим самим статус замовлення змінюється. Після того, як замовлення приготовлено, ставиться новий статус, потім він змінюється, коли все меню доставлено. І після оплати ставиться останній статус.

# 2 ВИБІР СТЕКУ ТЕХНОЛОГІЙ РОЗРОБКИ

## 2.1 Вибір платформи розробки

Вибір технологічного стеку для розробки додатків є критично важливим рішенням для будь-якої організації. Стек технологій може мати значний вплив на вартість, час виходу на ринок та загальний успіх додатку.

Одразу варто зазначити, для яких пристроїв ця розробка: для комп’ютерів, ноутбуків, смартфонів та планшетів. За виключенням специфічних гаджетів та електронних пристроїв, на кшталт ігрової приставки, або окулярів віртуальної реальності, додаток має однаково працювати на більшості операційних систем, в першу чергу, на найпопулярніших.

Варіант з розробкою десктопного додатку можна відкинути, тому що є різні операційні системи та доведеться окремо розробляти додатки для смартфонів, що робить розробку значно дорожче та значно довше. Бо, в першу чергу, або доведеться шукати компроміс, тобто технологію розробки, яка дозволить працювати використовувати додаток одночасно на девайсах на iOS, Windows та більшості дистрибутививів Linux. Більш того, користувачу доведеться встановлювати додаток собі на пристрій, що далеко не завжди зручно і це дуже непопулярна практика.

Шляхом найменшого опору можна назвати розробку веб-додатку. Цей шлях виключає такі труднощі, які було перелічено вище, тому що всі браузери базово інтерпретують лише HTML, CSS та JavaScript.

HTML (Hypertext Markup Language) – це мова розмітки, яка використовується для структурування контенту в Інтернеті. Вона визначає такі елементи, як заголовки, абзаци, зображення, посилання тощо. Браузери читають HTML-код і використовують його для правильного відображення сторінок.

CSS (Cascading Style Sheets – каскадні таблиці стилів) – це мова таблиць стилів, яка використовується для опису представлення документа, написаного в HTML або XML. Вона відповідає за визначення того, як візуально виглядатиме вміст на сторінці - від кольорів і шрифтів до макета та інтервалів.

JavaScript – це об'єктно-орієнтована мова програмування, яка дозволяє розробникам створювати інтерактивні ефекти у веб-браузерах. Її можна використовувати для таких завдань, як перевірка форм, створення анімації або інтерактивних ігор на веб-сайтах.

Хоча ці три мови є важливими інструментами для створення веб-сайтів, які працюють у різних браузерах вони не виконуються безпосередньо в самих браузерах, а інтерпретуються механізмами браузерів щоразу, коли користувачі відвідують сайти, створені з їхньою допомогою.

Ці мови для розробки сайтів мають дуже низький «поріг входу», тобто навчитися розробляти з цим стеком технологій легше, ніж тими, що використовуються для розробки десктопних та мобільних додатків. Хоча економія коштів залежить від таких факторів, як складність функцій, кількість підтримуваних пристроїв/платформ тощо, ось ілюстративне порівняння на основі даних, зібраних Clutch.co:

Середня вартість створення простого бізнес-сайту коливається від $5000 до $10 000. Середня вартість розробки базового додатку для iOS/Android становить від $20 тис. до $60 тис. для кожної платформи.

Але бувають випадки, коли необхідно розробляти саме десктопний та мобільний додаток, це той випадок, коли треба забезпечити активну взаємодію з пристроєм, або операційною системою, але це не випадок цієї дипломної роботи, тому було обрано саме веб-додаток.

## 2.2 Вибір стеку веб-розробки

Не дивлячись на те, що браузери працюють з цим набором мов, це не означає, що розробляти вебсайти можна тільки за допомогою них. В першу чергу є ще й серверна архітектура та бази даних, які теж є невід’ємною частиною веб-сайтів. Ось приклад найпопулярніших стеків розробки:

- Стек LAMP: LAMP розшифровується як Linux, Apache, MySQL та PHP. Це популярний стек для розробки динамічних веб-сайтів і веб-додатків. Linux

- це операційна система, Apache - веб-сервер, MySQL - база даних, а PHP - мова програмування серверної логіки.

- Стек MEAN: MEAN розшифровується як MongoDB, Express, AngularJS та Node.js. Це популярний стек для розробки односторінкових додатків (SPA). MongoDB - це база даних документів, Express - веб-фреймворк, AngularJS - фреймворк JavaScript, а Node.js - середовище виконання JavaScript.
- Стек Ruby on Rails: Ruby on Rails - це веб-фреймворк, написаний на мові програмування Ruby. Це популярний вибір для розробки веб-додатків, оскільки він простий у вивченні та використанні, а також надає багато функцій "з коробки".
- Стек Django: Django - це веб-фреймворк, написаний на мові програмування Python. Це популярний вибір для розробки веб-додатків, оскільки він швидкий, масштабований і безпечний.
- ASP.NET - це популярний стек веб-розробки, який базується на .NET Framework. Це хороший вибір для розробки веб-додатків, які вимагають високого рівня продуктивності та безпеки.

Але необов’язково використовувати саме перелічені стеки розробки, бо можна технології фронтенд розробки, бекенд розробки та бази даних. Далі буде розглянуто детально кожний аспект розробки web-додатку.

### 2.3 Розгляд роботи з базами даних

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім самих даних, містить їх опис та може містити засоби для їх обробки.

У сучасних інформаційних системах для забезпечення роботи з базами даних використовують системи керування базами даних (СКБД). Система керування базами даних — це система, заснована на програмних та технічних засобах, яка забезпечує визначення, створення, маніпулювання, контроль, керування та використання баз даних. Застосунки для роботи з базою даних можуть бути частиною СКБД або автономними. Найпопулярнішими СКБД є MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, Interbase, Firebird та IBM DB2. СКБД дозволяють ефективно працювати з базами даних, обсяг яких робить неможливим їх ручне опрацювання.

За моделлю організації даних розрізняють такі бази даних:

- Ієрархічна. Ієрархічна база даних може бути представлена як дерево, що складається з об'єктів різних рівнів. Між об'єктами існують зв'язки типу «предок-нащадок». При цьому можлива ситуація, коли об'єкт не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок.
- Мережна. Така база даних подібна до ієрархічної, за винятком того, що кожен об'єкт може мати більше одного предка.
- Реляційна. Реляційна база даних зберігає дані у вигляді таблиць. Найвживаніші СКБД використовують реляційну модель даних.
- Об'єктно-орієнтована. У базі даних цього виду дані оформляють у вигляді моделей об'єктів.

За розміщенням даних виділяють такі види баз:

- Локальна, або централізована. Така база даних підтримується на одному комп'ютері.
- Розподілена. Частини такої бази даних розміщують на різних комп'ютерах мережі.

За технологією фізичного зберігання виділяють:

- БД у вторинній пам'яті (традиційні).
- БД в оперативній пам'яті (in-memory database).
- БД у третинній пам'яті (tertiary database).

Також є розділення баз даних на структуровані та неструктуровані:

- Структуровані БД використовують структури даних, тобто структурований опис типу фактів за допомогою схеми даних, більш відомої як модель даних. Модель даних описує об'єкти та взаємовідношення між ними. Існує декілька моделей (чи типів) баз даних, основні: ієрархічна, мережна та реляційна.
- До неструктурованих БД належать повнотекстові бази даних, які містять неструктуровані тексти статей чи книг у формі, що дозволяє здійснювати швидкий пошук (наприклад, як Вікіпедія).

В цій роботі буде використовуватись MongoDB.

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.

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.

При розробці автори виходили з необхідності спеціалізації баз даних, завдяки чому їм вдалося відійти від принципу «один розмір під усе». За рахунок мінімізації семантики для роботи з транзакціями з'являється можливість вирішення цілого ряду проблем, пов'язаних з нестачею продуктивності, причому горизонтальне

масштабування стає простішим. Використовувана модель документів зберігання даних (JSON/BSON) простіше кодується, простіше управляється (у тому числі за рахунок застосування так званого «безсхемного стилю», а внутрішнє угруповання релевантних даних забезпечує додатковий виграш в швидкодії. Нереляційний підхід досить зручний для створення баз даних, у яких горизонтальне масштабування означає розгортання на множині машин. Можливість забезпечувати найкращу продуктивність повинна існувати паралельно з підтримкою більшої функціональності, ніж це дозволяє використання пар «ключ-значення» (у чистому вигляді).

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;

СКБД управляє наборами JSON-подібних документів, що зберігаються в бінарному форматі BSON. Зберігання і пошук файлів в MongoDB відбувається завдяки викликам протоколу GridFS. Подібно до інших документо-орієнтованих СКБД (CouchDB тощо), MongoDB не є реляційною СКБД.

Враховуючи, що обуло обрано MongoDB, варто подивитись, які варіанти ще підходять для розробки веб-додатку з даним стеком.

PostgreSQL — це потужна реляційна система керування базами даних (СКБД) з відкритим вихідним кодом, яка відрізняється високою надійністю та широкими можливостями. Вона забезпечує відповідність стандарту ACID,

розширені можливості індексування та підтримку складних запитів, що робить її придатною для додатків з великими обсягами даних. Підтримка JSON в PostgreSQL дозволяє зберігати і запитувати дані в форматі JSON, забезпечуючи гнучкість для веб-додатків, створених за допомогою Vue.js і Node.js.

Ця СКБД є рівноцінною до MongoDB в цій роботі. Є багато вагомих причин використовувати в цій роботі PostgreSQL. Node.js відомий своєю масштабованістю та здатністю ефективно обробляти паралельні з'єднання. Він використовує модель вводу/виводу, керовану подіями, що не блокує, що робить його добре придатним для одночасної обробки декількох запитів. У поєднанні з PostgreSQL, потужною та масштабованою реляційною базою даних, вона дозволяє розширювати веб-додатки та ефективно обробляти великі набори даних.

Node.js та Vue.js розроблені для швидкої та ефективної роботи. Неблокуюча модель вводу/виводу Node.js дозволяє швидко обробляти запити, в той час як віртуальний DOM і реактивне зв'язування даних Vue.js оптимізують рендеринг і ефективність оновлення в браузері. У поєднанні з оптимізованим виконанням запитів PostgreSQL можна створити високопродуктивні веб-додатки.

Поєднуючи Node.js, Vue.js та PostgreSQL, ви отримуєте переваги цілісного, повностекового середовища розробки JavaScript, масштабованості, можливостей роботи в режимі реального часу, оптимізації продуктивності та потужності надійної системи управління реляційними базами даних. Цей стек технологій добре підходить для створення сучасних, адаптивних і багатофункціональних веб-додатків.

MySQL – ще одна популярна реляційна база даних з відкритим вихідним кодом, яка добре працює з Vue.js та Node.js. Вона широко розповсюджена, має хорошу продуктивність і велику спільноту. MySQL підходить для різноманітних веб-додатків, включаючи платформи електронної комерції, системи управління контентом та бізнес-додатки. Взагалі, ця СКБД притаманна для сайтів, написаних на PHP, але її теж можна без перешкод використовувати з іншими технологіями розробки сайтів. MySQL – це зріла і широко розповсюджена система управління реляційними базами даних (СКБД) з відкритим вихідним кодом. Вона надає надійну

підтримку для управління структурованими даними, забезпечення цілісності даних за допомогою обмежень та встановлення зв'язків між таблицями. Це робить її добре придатною для додатків, які вимагають складного моделювання даних і транзакцій.

Також MySQL забезпечує повну відповідність стандартам ACID (Atomicity, Consistency, Isolation, Durability), що гарантує цілісність і надійність даних. Властивості ACID гарантують, що транзакції з базою даних виконуються надійно і послідовно, що робить її придатною для додатків, які вимагають суворої узгодженості та надійності даних.

Ця СКБД теж не відстає в своєї продуктивності та масштабованості від інших. Вона може ефективно обробляти великі набори даних і складні запити, що робить її придатною для додатків з інтенсивним використанням даних. Вона підтримує різні методи індексування, оптимізації запитів і механізми кешування для покращення виконання запитів і загальної продуктивності.

Звісно MySQL має велику і активну спільноту розробників і користувачів, що виражається у великій документації, навчальних посібниках і підтримці. Широке розповсюдження MySQL гарантує, що можна знайти безліч ресурсів, бібліотек та фреймворків для інтеграції з веб-додатком Node.js + Vue.js. Крім того, спільнота активно сприяє вдосконаленню та підтримці MySQL, забезпечуючи її постійний розвиток та надійність. MySQL має хорошу сумісність з Node.js та Vue.js. Node.js має кілька бібліотек і модулів, які спрощують інтеграцію MySQL з бекендом.

Бібліотеки ORM (Object-Relational Mapping), такі як Sequelize і TypeORM, забезпечують рівень абстракції, який спрощує взаємодію з базами даних і дозволяє працювати з MySQL в об'єктно-орієнтованому режимі. Така сумісність та інтеграція полегшує розробку, підтримку та масштабування додатків з використанням MySQL у стеку Node.js + Vue.js.

Є різні механізми реплікації, які дозволяють створювати надлишкові копії бази даних для забезпечення високої доступності та відмовостійкості. Це гарантує, що додаток може продовжувати працювати навіть у випадку апаратних збоїв або інших перебоїв. Функції реплікації MySQL роблять її придатною для додатків, які потребують високої доступності та надмірності даних.

MySQL надає надійні засоби безпеки для захисту даних. Вона підтримує автентифікацію користувачів, контроль доступу на основі ролей (RBAC) і шифрування для захисту конфіденційної інформації. Функції безпеки MySQL допомагають вам відповідати галузевим стандартам і правилам захисту даних.

Загалом, MySQL пропонує надійний, продуктивний і багатофункціональний варіант для зберігання та управління даними у стеку веб-додатків Node.js + Vue.js. Можливості реляційної бази даних, відповідність ACID, масштабованість, потужна підтримка спільноти, сумісність, функції реплікації та безпека роблять її популярним вибором для широкого спектру веб-додатків.

SQLite –це легка файлова реляційна база даних, яка часто використовується під час розробки або для невеликих додатків. Вона не вимагає окремого серверного процесу, що робить її простою у налаштуванні та використанні. SQLite підходить для простих додатків або прототипів і може бути хорошим вибором для малих і середніх проектів, де масштабованість не є першочерговим завданням.

Бази даних SQLite не залежать від платформи і можуть бути легко переміщені між різними операційними системами без проблем сумісності. Ця гнучкість дозволяє розробникам безперешкодно працювати в різних середовищах розробки та розгортати додатки на різних платформах.

SQLite теж підтримує транзакції ACID також добре працює в сценаріях, де операції читання домінують над операціями запису. Його легка вага та оптимізований формат зберігання сприяють швидкому доступу для читання, що робить його придатним для додатків, які потребують швидкого отримання даних. Оскільки SQLite є безсерверною базою даних, вона усуває необхідність мережевого зв'язку та клієнт-серверної взаємодії. Це спрощує архітектуру і зменшує затримки, пов'язані з мережею, що робить її вигідною для локальних або однокористувацьких додатків.

Одним з помітних недоліків SQLite є його обмежена здатність змінювати схеми баз даних після їх створення. На відміну від інших реляційних баз даних, SQLite не надає вбудованих механізмів для безпосередньої зміни структур таблиць. Замість цього він вимагає перестворення всієї таблиці з потрібними змінами, що

може бути проблемою в ситуаціях, коли потрібні складні зміни схеми без втрати даних.

Важливо враховувати це обмеження при використанні SQLite, особливо якщо програма вимагає частих модифікацій схем таблиць або має більш складні потреби в управлінні даними. Якщо можливість динамічно змінювати таблиці є критичною вимогою для проекту, інші реляційні бази даних, такі як MySQL або PostgreSQL, можуть краще відповідати потребам.

Вибір найбільш підходящої бази даних для веб-розробки залежить від конкретних вимог проекту, потреб у масштабуванні, складності структури даних та уподобань розробника. Реляційні бази даних, такі як MySQL та PostgreSQL, чудово працюють зі структурованими даними та складними взаємозв'язками, тоді як NoSQL-підхід MongoDB вирізняється гнучкістю та масштабованістю для неструктурованих або швидкозмінних даних.

В цьому проекті було обрано MongoBD через можливість розміщувати бази даних на виділеному сервері, без необхідності створювати локальну базу даних.

### 2.4 Розгляд стеку технологій бекенд-розробки

Бекенд-розробка – це процес створення та підтримки серверних компонентів веб-сайту або веб-додатку. Сюди входить код, який виконується на сервері, база даних та інфраструктура, яка підтримує веб-сайт або веб-додаток. Розробка баз даних теж відноситься, але в цій роботі бази даних будуть описані окремо. Бекенд-розробка вирішує багато задач в розробці веб-додатків.

Таблиця 2.1 – Задачі, які вирішує бекенд-розробка

| Задача | Опис задачі |
|---|---|
| Програмування на стороні сервера | Розробка бекенду передбачає написання коду на таких мовах, як JavaScript (Node.js), Python, Ruby, PHP, Java або C#. Цей код виконується на сервері і обробляє вхідні запити, обробляє дані та генерує відповіді, які надсилаються назад на клієнтську частину (фронтенд) веб-сайту. |
| Створення фреймворків та бібіліотек та їх | Бекенд-розробники часто використовують фреймворки або бібліотеки, які надають набір готових інструментів, функцій та абстракцій для спрощення розробки та вирішення загальних завдань. Прикладами є Express.js для |

| | |
|---|---|
| викростання | Node.js, Django для Python, Ruby on Rails, Laravel для PHP та ASP.NET Core для C#. Ці фреймворки забезпечують структуру, маршрутизацію, інтеграцію з базами даних та інші функції для прискорення бекенд-розробки. |

Продовження табл. 2.1

| **Задача** | **Опис задачі** |
|---|---|
| Створення API та веб-сервисів | Внутрішня розробка включає в себе проектування та реалізацію інтерфейсів прикладного програмування (API) і веб-сервісів, які дозволяють різним системам або клієнтам (включаючи фронтенд-додатки) взаємодіяти з внутрішнім сервером. API визначають протоколи зв'язку та формати даних, що використовуються для обміну інформацією, уможливлюючи пошук даних, їх оновлення та інтеграцію із зовнішніми сервісами. |
| Управління базами даних | Бекенд-розробка часто включає роботу з базами даних для зберігання та пошуку даних. Бекенд-розробники проектують схеми баз даних, створюють таблиці, визначають зв'язки та пишуть запити для виконання CRUD-операцій (створення, читання, оновлення, видалення) над даними. Популярні бази даних, що використовуються в бекенд-розробці, включають MySQL, PostgreSQL, MongoDB і SQLite. |
| Забезпечення безпеки та автентифікації | Бекенд-розробники відіграють вирішальну роль у впровадженні заходів безпеки для захисту даних користувачів та запобігання несанкціонованому доступу. Це включає реалізацію механізмів автентифікації та авторизації, управління сеансами користувачів, шифрування конфіденційної інформації та захист від поширених вразливостей безпеки, таких як SQL-ін'єкції та міжсайтовий скриптинг (XSS). |
| Оптимізація продуктивності | Бекенд-розробники оптимізують код на стороні сервера та запити до бази даних, щоб забезпечити ефективну та масштабовану продуктивність. Вони аналізують вузькі місця в роботі, застосовують стратегії кешування, точно налаштовують індекси бази даних та використовують інші методи для підвищення загальної швидкості та чуйності веб-сайту. |
| Інтеграція із зовнішніми сервісами | Розробка бекенду часто передбачає інтеграцію із зовнішніми сервісами, такими як платіжні шлюзи, сторонні API, системи обміну повідомленнями, хмарні сервіси або платформи соціальних мереж. Бекенд-розробники займаються інтеграцією, обміном даними та комунікаційними протоколами, необхідними для взаємодії з цими сервісами. |
| Розгортання та обслуговування | Бекенд-розробники відповідають за розгортання серверного коду та управління серверною інфраструктурою. Це включає в себе конфігурацію веб-серверів, налаштування середовища, забезпечення масштабованості, моніторинг продуктивності та виконання завдань з обслуговування, таких як оновлення, резервне копіювання та усунення несправностей. |

В цій роботі будуть вирішені майже всі задачі з цього списку.

Перед тим як розглядати обрану в проекті технологію бекенд-розробки, варто взагалі ознайомитись з тим, які можуть бути варінати.

1) Node.js - це середовище виконання, яке дозволяє запускати JavaScript на стороні сервера, що робить його природним вибором для створення бекенду додатку Vue.js. Express.js - популярний фреймворк веб-додатків для Node.js,

який надає надійний набір функцій для створення API та обробки HTTP-запитів. Власне, цей варіант був обраний, в цьому розділі буде більш детально розписано його переваги.

2) Ruby on Rails - це фреймворк веб-додатків, який відповідає архітектурному шаблону Model-View-Controller (MVC). Він забезпечує підхід з превалюванням конвенцій над конфігурацією, що полегшує розробку API та обробку внутрішньої логіки для додатку Vue.js.
3) Django - це високорівневий веб-фреймворк на Python, відомий своєю простотою, масштабованістю та безпекою. Він відповідає шаблону MVC і надає повний набір інструментів для швидкої розробки та розгортання веб-додатків, включаючи API для передачі даних до фронтенду Vue.js.
4) Laravel - це PHP-фреймворк для створення веб-додатків, який підкреслює елегантний синтаксис, простоту та продуктивність розробників. Він надає широкий спектр функцій, включаючи маршрутизацію, абстракцію баз даних та підтримку RESTful API, що робить його придатним для створення надійних бекенд API для Vue.js додатку.
5) Firebase - це платформа від Google, яка надає низку бекенд-сервісів, включаючи базу даних у режимі реального часу, автентифікацію, хмарне сховище та хостинг. Вона пропонує легку інтеграцію з Vue.js, що дозволяє швидко створювати та розгортати повностекові додатки без управління серверною інфраструктурою.
6) ASP.NET Core - це кросплатформенний фреймворк для створення веб-додатків на C# або F#. Він надає потужну та модульну архітектуру, яка підтримує створення API, обробку серверної логіки та передачу даних до фронтенду Vue.js.

На відміну від баз даних, вибір з цього списку технологій є значно простішим. Тут є значущі переваги першого варіанту. В першу чергу, це синергія екосистеми JavaScript: І Vue.js, і Node.js написані на JavaScript, що дозволяє безперешкодно інтегрувати та обмінюватися кодом між фронтендом і бекендом. Ця синергія дозволяє розробникам працювати з єдиною мовою програмування і

використовувати свої навички JavaScript у всьому стеку, що підвищує продуктивність і зручність обслуговування коду.

Також Node.js з Express.js добре підходить для рендерингу на стороні сервера (SSR), що може покращити час початкового завантаження та пошукову оптимізацію (SEO) додатку Vue.js. SSR дозволяє серверу рендерити початковий HTML-контент і надсилати його клієнту, покращуючи сприйняття продуктивності та дозволяючи пошуковим системам ефективніше сканувати та індексувати вміст.

Express.js надає потужну систему маршрутизації, яка дозволяє визначати та керувати кінцевими точками API і обробляти різні HTTP-методи (GET, POST, PUT, DELETE тощо). Це полегшує розробку та організацію маршрутів API бекенда і забезпечує безперебійну комунікацію між фронтендом і бекендом.

Express.js пропонує широкий спектр модулів проміжного програмного забезпечення, які можна легко інтегрувати в конвеєр запитів/відповідей додатку. Функції проміжного програмного забезпечення можуть виконувати такі завдання, як автентифікація, перевірка вхідних даних, ведення журналів, стиснення і багато іншого. Така модульна архітектура спрощує реалізацію загальних функцій і підвищує можливість повторного використання коду.

Node.js та Express.js мають жваву та активну спільноту з великою кількістю документації, навчальних посібників та бібліотек. Ця підтримка спільноти надає доступ до безлічі ресурсів, досвіду та сторонніх пакетів, які можуть прискорити розробку, вирішити загальні проблеми та забезпечити надійність внутрішньої реалізації.

Node.js відомий своєю масштабованістю та здатністю ефективно обробляти велику кількість одночасних з'єднань. Його модель вводу/виводу, керована подіями, що не блокується, забезпечує високу пропускну здатність і швидкість реакції, що робить його придатним для додатків з високим трафіком і вимогами до зв'язку в реальному часі. Node.js з Express.js полегшує інтеграцію додатку Vue.js з іншими системами та сервісами. Будь то підключення до баз даних, взаємодія із зовнішніми API або реалізація функцій у режимі реального часу за допомогою веб-сокетів,

екосистема JavaScript надає численні бібліотеки та модулі, які спрощують завдання інтеграції.

Розгортання Node.js з додатком Express.js є відносно простим у порівнянні з деякими іншими бекенд-технологіями. Багато хостинг-провайдерів та хмарних платформ мають відмінну підтримку додатків Node.js, що полегшує розгортання, масштабування та управління проектом Vue.js у виробництві.

Загалом, вибір Node.js з Express.js в якості бекенду для проекту Vue.js пропонує безшовну інтеграцію JavaScript, ефективний рендеринг на стороні сервера, гнучку маршрутизацію, підтримку проміжного програмного забезпечення, велику спільноту, масштабованість та спрощену інтеграцію з іншими системами. Ці переваги роблять його популярним та ефективним вибором для створення повностекових Vue.js-додатків.

## 2.5 Технології frontend-розробки

Фронтенд веб-розробка – це процес створення призначених для користувача або клієнтських компонентів веб-сайту чи веб-додатку. Вона включає в себе проектування і створення інтерфейсу, з яким користувачі взаємодіють безпосередньо у своїх веб-браузерах. Фронтенд-розробка фокусується на візуальних та інтерактивних аспектах веб-сайту, забезпечуючи безперебійну та цікаву взаємодію з користувачем.

Для фронтент веб-розробки основні мови це HTML,CSS та JavaScript. Однак фронтенд розробка не обмежується лише використанням цих мов, а включає цілий комплекс підходів, аби сайт був зручним, швидким, цікавим та виконував свої задачі.

Фронтенд-розробники зосереджуються на створенні веб-сайтів, які є адаптивними, тобто можуть адаптуватися і забезпечувати оптимальний досвід перегляду на різних пристроях і розмірах екранів. Це передбачає використання медіа-запитів CSS, гнучких макетів і гнучких сіток, щоб веб-сайт добре виглядав і працював на настільних комп'ютерах, планшетах і мобільних пристроях.

Не менш важливо зробити веб-сайти доступними для всіх користувачів, включаючи людей з обмеженими можливостями. Це передбачає дотримання найкращих практик семантичної розмітки HTML, надання альтернативного тексту для зображень, увімкнення клавіатурної навігації та забезпечення належного колірного контрасту. Міркування доступності спрямовані на те, щоб зробити веб-сайти зручними та інклюзивними для всіх.

Не дивлячись на те, що різниця між браузерами, в цілому, має тенденцію до зменшення, розробники все-одно повинні переконатися, що веб-сайт працює правильно і виглядає однаково в різних веб-браузерах, таких як Chrome, Mozilla Firefox, Safari та Edge. Вони тестують і коригують код, щоб врахувати специфічні особливості браузерів, а також впроваджують прогресивні вдосконалення або поліфункції, щоб забезпечити узгоджену роботу для всіх користувачів.

Фронтенд веб-розробка зосереджена на створенні візуально привабливих, адаптивних та інтерактивних користувацьких інтерфейсів. Вона передбачає поєднання HTML, CSS та JavaScript для створення інтерфейсних компонентів веб-сайту та забезпечення їхньої безперебійної роботи з бекендом для створення цілісного веб-досвіду. Співпраця між фронтенд- і бекенд-розробниками має вирішальне значення для успішної розробки веб-додатків.

Для швидкої розробки користувацьких інтерфейсів доволі складно використовувати тільки JavaScript. Наразі, розвиток бібліотек та фреймворків настільки серйозний, що використання цих інструментів в розробці мають очевидні переваги.  Бібліотеки надають готові компоненти, функції та інструменти, які можуть значно пришвидшити час розробки. Замість того, щоб створювати однакові для всіх проектів необхідні конструкції, розробники можуть використовувати існуючу функціональність, надану бібліотеками, для більш ефективного виконання типових завдань. Це економить час і зусилля, дозволяючи розробникам зосередитися на створенні унікальних функцій і логіки, характерних для їхнього проекту.

По-друге, бібліотеки часто мають велику та активну спільноту розробників. Це означає, що вони отримують регулярні оновлення, виправлення помилок та

покращення, що гарантує розробникам доступ до найновіших функцій та вдосконалень. Спільнота також надає підтримку, документацію та ресурси, що полегшує вивчення та ефективне використання бібліотеки.

Крім того, бібліотеки сприяють повторному використанню коду та його підтримці. Використовуючи добре зарекомендували себе і широко прийняті бібліотеки, розробники можуть покладатися на перевірені і перевірені рішення. Це зменшує ймовірність появи помилок і підвищує загальну стабільність кодової бази. Крім того, коли над проектом працює кілька розробників, використання бібліотек сприяє узгодженості стилів і практик кодування, полегшуючи членам команди розуміння і спільну роботу над кодовою базою.

Бібліотеки також покращують досвід розробки, надаючи абстракції та високорівневі конструкції. Вони абстрагуються від складних деталей реалізації, дозволяючи розробникам зосередитися на бажаному результаті, а не зациклюватися на низькорівневих технічних деталях. Це підвищує продуктивність і робить процес розробки приємнішим і менш схильним до помилок.

Ще однією перевагою використання бібліотек є потенціал для оптимізації продуктивності. Багато бібліотек розроблено та оптимізовано для забезпечення ефективних та продуктивних рішень. Використовуючи ці оптимізовані рішення, розробники можуть підвищити загальну продуктивність своїх інтерфейсних додатків, не витрачаючи надмірного часу і зусиль на налаштування продуктивності. Ці переваги роблять бібліотеки цінним активом для фронтенд-розробників, дозволяючи їм створювати надійні, ефективні та багатофункціональні веб-додатки більш ефективно. Використання бібліотек та фреймворків дозволяє не просто бути конкурентноздатним розробником на ринку праці, а вже є майже обов’язковим досвідом для виходу на цей ринок.

Тому вибір для цієї роботи не в тому, використовувати бібліотеки, чи ні, а в тому, яку саме використовувати. Тут теж є різноманіття тут необхідно звузити коло найбільш підходящих для завдання бібліотек. На відміну від попереднього розділу, можна піти зворотнім шляхом, уявляючи, що спочатку було обрано бекенд-технології та базу даних.

Можна виділити 3 найкращі бібліотеки для рішення задачі цієї роботи:

1. Vue.js — популярний JavaScript-фреймворк для створення користувацьких інтерфейсів. Він пропонує архітектуру на основі компонентів, реактивність та інтуїтивно зрозумілий синтаксис.
2. React — це широко розповсюджена бібліотека JavaScript для створення користувацьких інтерфейсів. Вона використовує компонентний підхід і віртуальну DOM для ефективного рендерингу та управління компонентами інтерфейсу. React добре працює з Node.js + Express.js, оскільки його можна використовувати для споживання API та обробки даних з бекенду.
3. AngularJS — JavaScript-фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону Модель-вид-контролер (MVC), а також спрощення їх тестування та розробки.

Всі ці бібліотеки та фреймфорки без проблем працюють з Node.js + Express.js, оскільки дозволяє використовувати API та легко обробляти дані з бекенду. Vue.js має ряд характерних ознак, які його виділяють на фоні інших варіантів:

Vue.js має м'яку криву навчання, що робить його доступним як для початківців, так і для досвідчених розробників. Його синтаксис і концепції прості та інтуїтивно зрозумілі, що дозволяє розробникам швидко освоїти фреймворк і почати створювати додатки. Ця простота скорочує час розробки та дозволяє проводити швидші ітерації. Vue.js використовує компонентну архітектуру, яка добре узгоджується з модульною природою ресторанних веб-додатків. За допомогою Vue.js можна створювати багаторазові та автономні компоненти для різних частин програми, таких як меню, форми замовлень, системи бронювання та профілі користувачів. Така модульність сприяє багаторазовому використанню коду, його підтримці та масштабованості.

Vue.js використовує реактивне зв'язування даних, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних. Ця функція особливо корисна

для оновлень в режимі реального часу в ресторанних додатках, таких як динамічне оновлення статусів замовлень або наявність вільних місць в режимі реального часу. Vue.js також наголошує на декларативному підході, що дозволяє розробникам зосередитися на описі бажаного результату, а не на ручному маніпулюванні DOM.

Vue.js постачається з Vue Router, потужною бібліотекою маршрутизації, яка спрощує навігацію та дозволяє створювати кілька представлень і маршрутів у додатку. Це корисно для розділення розділів додатку для клієнтів та працівників ресторану, забезпечуючи чітке розділення функціональності та користувацького досвіду. Для додатків зі складним управлінням станами Vue.js пропонує Vuex, бібліотеку управління станами. Vuex забезпечує централізоване управління станами, дозволяючи різним компонентам отримувати доступ до спільних даних та оновлювати їх. Це корисно, коли йдеться про автентифікацію користувачів, управління замовленнями та спілкування між клієнтами і працівниками ресторану.

Vue.js має багату екосистему бібліотек компонентів інтерфейсу користувача, включаючи Vuetify, Quasar та Element UI. Ці бібліотеки надають готові, настроювані компоненти інтерфейсу, розроблені за принципами Material Design, що дозволяє створювати візуально привабливі та адаптивні інтерфейси для ресторанних додатків.

Хоча React та AngularJS також є популярними інструментами для розробки фронтенду, Vue.js виділяється своєю простотою, архітектурою на основі компонентів, реактивним дизайном та добре інтегрованим інструментарієм. Враховуючи існуючий стек бекенда MongoDB з Node.js + Express.js, Vue.js легко інтегрується, забезпечуючи безперебійну комунікацію між фронтендом і бекендом. Він пропонує міцну основу для створення багатофункціонального та зручного для користувача веб-додатку для ресторану.

### 2.6 Архитектура веб-додатків

Коли було обрано всі технології розробки, постає питання їх правильного поєднання. Адже на відміну від примітивних веб-додактів, де в одному файлі є і серверна логіка і фронтенд сайту, такий підхід є застарілим та має багато проблем.

Фронтенд відповідає за відображення користувацького інтерфейсу та обробку даних, введених користувачем (рис. 2.1).

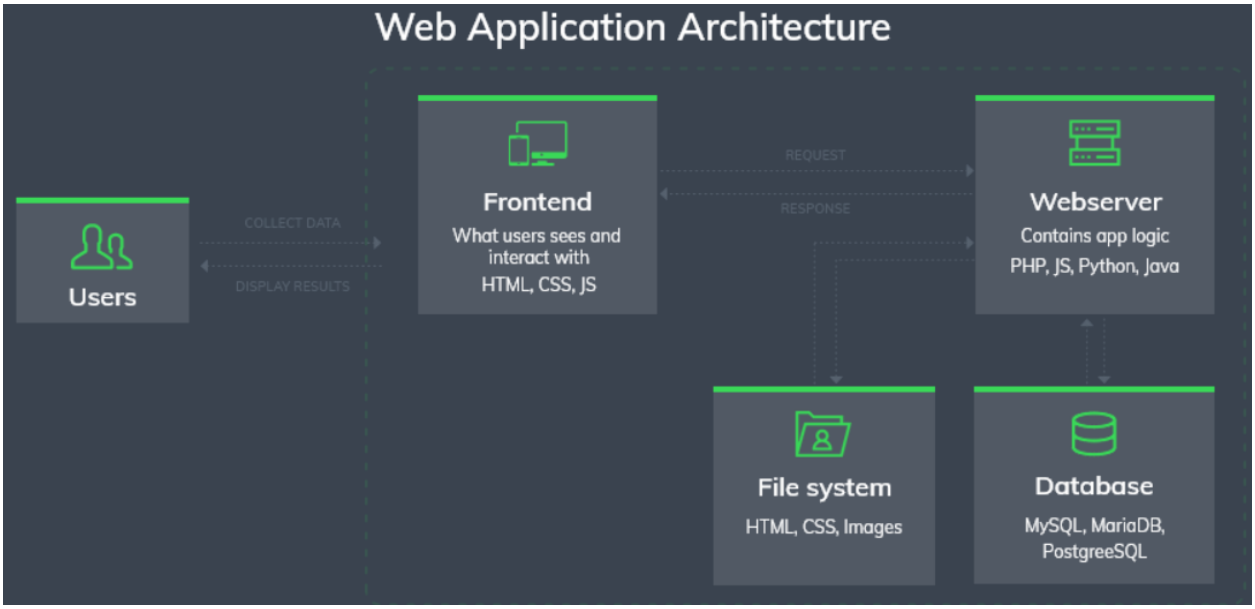



Рис.2.1. – Загальна архітектура сайту

Але якщо більш деталізовано розглядати архітектуру сучасних веб-додатків, то архітектура має схематичне відображення (рис. 2.2).

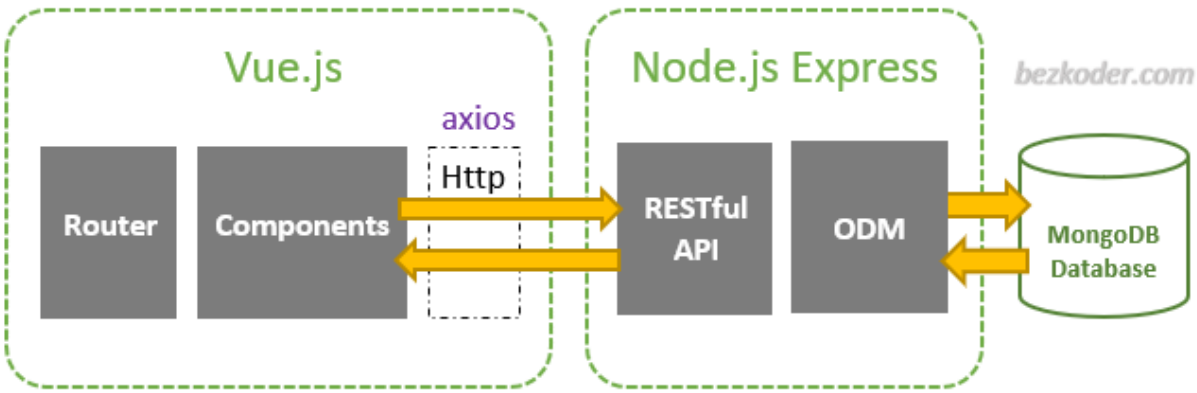



Рис. 2.2 – Архітектура сучасного веб-додатку

Компоненти Vue використовуються для створення багаторазових елементів інтерфейсу, таких як кнопки, форми та меню; ці компоненти можуть бути вкладені один в одного для формування складних інтерфейсів. В браузері ці компоненти

перетворюються на звичайний код HTML, CSS і JavaScript. Фронтенд робить HTTP-запити до бекенду для отримання даних, і той оновлює сторінку у відповідь на введення користувача.

Бекенд відповідає за зберігання даних, обробку даних, генерування даних і надання API для фронтенду. Він робить це за допомогою Node.js та Express.js. Бекенд робить запити до бази даних, щоб отримати дані, і робить HTTP-відповіді до фронтенду, щоб оновити сторінку.

База даних використовується для зберігання даних для додатку. Для цього використовується MongoDB. Бекенд отримує доступ до бази даних для зберігання та отримання даних.

Тобто в цьому випадку архітектура додатку не монолітна, є окремий фронтенд сервер, який працює на Vue.js, який робить http-запити до бекенд серверу.

# 3 ОПИС ПРОЦЕСУ СТВОРЕННЯ ДОДАТКУ

## 3.1 Проектування додатку

Перед тим як почати створювати сам додаток, треба спочатку зробити деякі підготовчі дії. Це передбачає розуміння призначення веб-застосунку, його цільової аудиторії та бажаної функціональності. Також потрібно буде визначити будь-які обмеження, такі як бюджет, часові рамки та технологічні уподобання.

На основі вимог створіть план проекту, який окреслює обсяг, цілі та результати. Сюди входить визначення функцій і функціональності, встановлення етапів і розподіл ресурсів. Ви також можете створити структуру розбиття робіт (WBS), щоб розбити проект на менші, керовані завдання.

Важливо розробити UI та UX веб-додатку, зосередившись на зручності, доступності та естетиці. Це передбачає створення каркасів, макетів і прототипів для візуалізації макета, навігації та загального вигляду додатку. Також потрібно подумати про адаптивний дизайн, щоб переконатися, що додаток добре працює на різних пристроях і розмірах екранів.

Задача проектного менеджера, після утвердження проекту та технічного завдання на розробку, розбити процесс розробки на підзадачі та розподілити між розробниками.

## 3.2 Бекенд додатку

Для того, аби сформувати бекенд, треба визначити сутності, з якими буде проходити робота. Ці сутності треба одразу створити в базі даних. Для додатку було створено наступні таблиці.

Таблиця 3.1 – Меню: таблиця з блюдами

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| name: {type: String, required: true} | Назва блюда, рядок, обов’язкове |

Продовження таблиці 3.1

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| cost: {type: Number, required: true} | Ціна блюда: число, обов’язкове |
| cookingTime: {type: Number, required: true} | Час приготування (в хвилинах): число, обов’язкове |
| ingredients: {type: Map,of: Number,required: true} | Перелік інгрідієнтів, ассоціативний массив, де ключ- це унікальний ідентифікатор продукта, а значення- кількість одиниць цього продукта, поле обов'язкове |

Таблиця 3.2 – Користувачі: таблиця з користувачами

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| name: {type: String,required: true} | Ім’я користувача, рядок, обов’язкове |
| email:{type: String,required: true,unique: true,} | Пошта користувача, рядок, обов’язкове, унікальне |
| password: {type: String,required: true,} | Пароль, рядок, обов'язкове |
| isClient: {type: Boolean,required: true, default: 1 } | Клієнт, бінарне, обов’язкове (0 якщо користувач не є клієнтом, 1, якщо є), за замовчуванням - 1 |
| staffRole: {<br>type: String,<br>enum: staffRoleEnum,<br>required: function() {<br>return !this.isClient;<br>}<br>} | Посада члена персонала, рядок, який обирається з переліку доступних, обов’язкове, якщо isClient == 0 |

Таблиця 3.3 – Продукт: таблиця з продуктами

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| fullName: {type: String,required: true} | Повна назва продукта (номенклатурна), строка, обов’язкове |
| menuName: {type: String,required: true} | Назва продукта для меню, строка, обов’язкове |
| unitOfMeasure:{type: String,enum: ['kg', 'piece'],required: true} | Одиниця виміру, строка з переліку, обов'язкове |
| costPerUnit: {type: Number,required: true,} | Вартість одиниці продукта, число, обов’язкове |
| deliveryDate: {type: Date,required: true} | Дата поставки, дата, обов’язкове |
| overdueDate: {type: Date,required: true} | Дата прострочки, дата, обов’язкове |

Таблиця 3.4 – Статус: таблиця зі статусами замовлення

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| name: {type: String,required: true} | Назва статусу, рядок, обов’язкове |
| sequence: {type: Number,required: true,} | Порадяковий номер статусу, число, обов’язкове |

Таблиця 3.5 – Замовлення: таблиця з замовленнями

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| foodId:{type: mongoose.Schema.Types.ObjectId,required: true} | Id блюда, посилання на запис в таблиці, обов'язкове |
| statusId:{type: mongoose.Schema.Types.ObjectId,required: true} | Id статуса, посилання на запис в таблиці, обов'язкове |
| tableId:{type: mongoose.Schema.Types.ObjectId, required: true} | Id столика, посилання на запис в таблиці, обов'язкове |
| userId{type: mongoose.Schema.Types.ObjectId,required: true} | Id користувача, посилання на запис в таблиці, обов'язкове |
| orderDate: {type: Date, default: Date.now, required: true } | Дата замовлення, дата, обов’язкове |

Таблиця 3.6 – Столи: таблиця з інформацію про столики

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| tableNumber: {type: Number,required: true} | Номер стола, число, обов'язкове |
| areaInCm:{type: Number,required: true,} | Id статуса, число, обов'язкове |

Таблиця 3.7 – Інформація про точку: таблиця з інформацію про точку ресторану

| Код схеми | Опис (назва, тип, параметри) |
|---|---|
| latitude: {type: Number,required: true} | Широта, число, обов'язкове |
| longitude: {type: Number,required: true} | Довгота, число, обов'язкове |
| address: {type: String,required: true} | Адреса, рядок, обов'язкове |
| currency: {type: String,required: true} | Валюта, рядок, обов'язкове |

Для грамотного оформлення бекенду краще використовувати розподілення архітектури. Для кожної сутності є котроллер, який дозволяє виконувати операції, які передбачені логікою програми. Частіше за усього це CRUD операції. А посилання, яке дозволює виконувати ці операції, знаходиться в роутері до кожної сутності.

Весь код буде представлено в додатках, в якості прикладу контролера можна представили, наприклад, userController:

//підключаємо модель користувача з файлу моделі, де представлена схема таблиці бази даних

```
const User = require('../models/user');
const bcrypt = require('bcrypt');

// Функції контроллера

// Отримати всіх користувачів
exports.getAllUsers = (req, res) => {
    User.find({}, { password: 0}) // Виключити поле пароля з відповіді
        .then(users => res.json(users))
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};

// Отримати користувача по ID
exports.getUserById = (req, res) => {
    const userId = req.params.id;
    User.findById(userId, { password: 0 }) // Виключити поле пароля з
відповіді
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found'
});
            }
            res.json(user);
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};
// Оновити дані користувача
exports.updateUser = (req, res) => {
    const userId = req.params.id;
```

```
    const { name, email, isClient, staffRole } = req.body;
    User.findByIdAndUpdate(userId, { name, email, isClient, staffRole
}, { new: true })
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found'
});
            }
            res.json(user);
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};
// Видалити користувача
exports.deleteUser = (req, res) => {
    const userId = req.params.id;
    User.findByIdAndDelete(userId)
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found'
});
            }
            res.json({ message: 'User deleted successfully' });
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};
Реєстрація користувача
exports.registerUser = (req, res) => {
    const { name, email, password } = req.body;
    // Перевірка, чи існує користуач з цим email
```

```
    User.findOne({ email })
        .then(existingUser => {
            if (existingUser) {
                return res.status(400).json({ error: 'Email already registered' });
            }
            // шифрування паролю
            bcrypt.hash(password, 10, (err, hashedPassword) => {
                if (err) {
                    console.error(err);
                    return res.status(500).json({ error: 'Server error' });
                }
                // створення нового користувача
                const newUser = new User({ name, email, password: hashedPassword });
                newUser.save()
                    .then(user => res.json(user))
                    .catch(error => {
                        console.error(error);
                        res.status(500).json({ error: 'Server error' });
                    });
            });
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};
// Логін користувача
exports.loginUser = (req, res) => {
    const { email, password } = req.body;

    // Знайти юзера за введеною поштою
```

```
    User.findOne({ email })
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found' });
            }
            // Порівняти введений пароль з тим що в базі
            bcrypt.compare(password, user.password, (err, result) => {
                if (err) {
                    console.error(err);
                    return res.status(500).json({ error: 'Server error' });
                }
                if (!result) {
                    return res.status(401).json({ error: 'Invalid password' });
                }
                res.json({ message: 'Login successful', user });
            });
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};
```

Функція getAllUsers отримує всіх користувачів з бази даних і повертає їх у форматі JSON.

Функція getUserById знаходить користувача за наданим ідентифікатором і повертає дані користувача у форматі JSON. Вона обробляє випадки, коли користувача не знайдено або виникла помилка сервера.

Функція updateUser оновлює існуючого користувача на основі наданого ідентифікатора. Вона витягує оновлені значення з тіла запиту і використовує findByIdAndUpdate, щоб знайти користувача за ID і оновити поля. Він повертає

оновленого користувача у форматі JSON і обробляє випадки, коли користувача не знайдено або виникла помилка сервера.

Функція deleteUser видаляє користувача на основі наданого ID. Вона використовує findByIdAndDelete, щоб знайти користувача за ідентифікатором і видалити його з бази даних. Вона повертає JSON-відповідь, що вказує на успішність видалення і обробляє випадки, коли користувача не знайдено або якщо виникла помилка сервера.

Функція registerUser використовує бібліотеку bcrypt для хешування пароля перед збереженням його в базі даних. Функція bcrypt.hash використовується для генерації хешованого пароля із значенням 10. Потім хешований пароль зберігається в об'єкті newUser і зберігається в базі даних.

Функція loginUser використовує функцію bcrypt.compare для порівняння наданого пароля зі збереженим хешованим паролем. Якщо паролі збігаються, вхід в систему буде успішним. В іншому випадку повертається повідомлення про помилку.

Для доступу до цих методів створено таких роутер:

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');
// Маршрути
//отримати всіх користувачів
router.get('/', userController.getAllUsers);
//отримати користувача по id
router.get('/:id', userController.getUserById);
//змінити користувача по його id
router.put('/:id', userController.updateUser);
//видалити користувача по його id
router.delete('/:id', userController.deleteUser);
// Створити нового користувача
router.post('/register', userController.registerUser);
// Логін користувача
router.post('/login', userController.loginUser);
```

```
module.exports = router;
```

Таблиця 3.8 – Відповіді на запити

| | |
|---|---|
| getAllUsers() | [<br>{<br>"_id": "603fdd2db9be814a0c8a88d7",<br>"name": "John Doe",<br>"email": "johndoe@example.com",<br>"isClient": true,<br>"staffRole":"admin",<br>"createdAt": "2023-03-02T10:20:30Z",<br>"updatedAt": "2023-03-02T11:25:40Z"<br>}<br>//And the other users<br>] |
| getUserById() | {<br>"_id": "603fdd2db9be814a0c8a88d7",<br>"name": "John Doe",<br>"email": "johndoe@example.com",<br>"isClient": true,<br>"staffRole":"admin",<br>"createdAt": "2023-03-02T10:20:30Z",<br>"updatedAt": "2023-03-02T11:25:40Z"<br>} |

### 3.3 Фронтенд додатку

Фронтенд в цьому проекті організований за допомогою головної сторінки: App.vue та компонентів, які є в папці components. Ці компоненти додаються на головну сторінку як «конструктор». Як показано на рисунку 2.2 фронтенд-сервер отримує дані від бекенду за допомогою бібліотеки axios, це бібліотека, написана на JavaScript, яка дозволяє робити асинхроні http запити, які не потребують перезагрузки сторінки. Приклад роботи такої системи по списку користувачів:

1. Отримуємо користувачів від бекенд серверу за допомогою запиту в axios.

```
export default {
  data() {
    return {
      users: [],
    };
```

```
  },
  mounted() {
    this.fetchUsers();
  },
  methods: {
    fetchUsers() {
      // робимо виклик по апі до нашого бекенд серверу
      axios.get('/api/users')
        .then(response => {
          this.users = response.data;
        })
        .catch(error => {
          console.error('Error fetching users:', error);
        });
    },
    // Rest of the methods
  },
};
</script>
```

2. Якщо запит був вдалим, без помилок, то отримаємо відповідь такого виду:

```
[
    {
        "_id": "1",
        "name": "John Doe",
        "email": "johndoe@example.com",
        "isClient": true,
        "staffRole": "admin",
        "createdAt": "2023-03-02T10:20:30Z",
        "updatedAt": "2023-03-02T11:25:40Z"
    },
    {
        "_id": "2",
        "name": "Jane Smith",
        "email": "janesmith@example.com",
```

```
            "isClient": false,
            "staffRole": "waiter",
            "createdAt": "2023-03-03T09:15:20Z",
            "updatedAt": "2023-03-03T10:30:45Z"
        },
        {
            "_id": "3",
            "name": "Robert Johnson",
            "email": "robertjohnson@example.com",
            "isClient": false,
            "staffRole": "-",
            "createdAt": "2023-03-04T13:45:10Z",
            "updatedAt": "2023-03-04T14:55:25Z"
        }
    ];
```


3. В компоненті, який зробив запит, ці данні використовуються для формування форми, яка редагується (рис. 3.1).

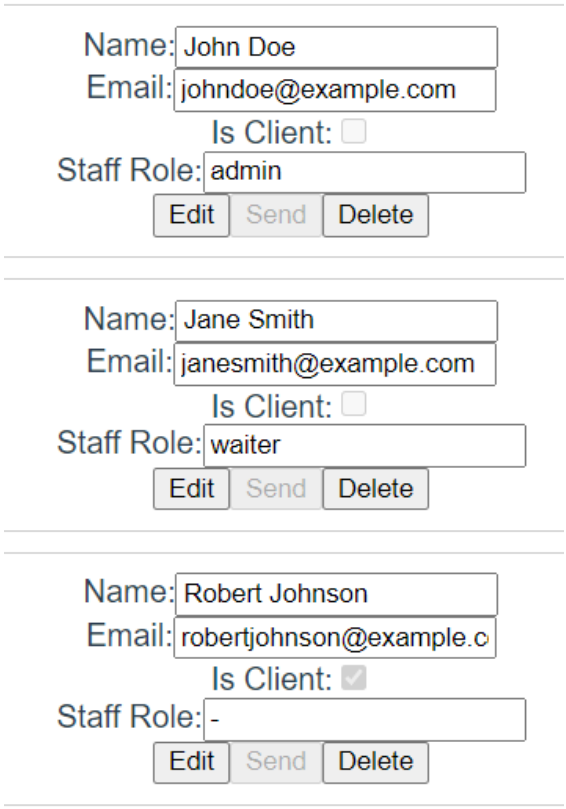



Рис. 3.1 – Форми користувачів

Якщо відредагувати поле користувача та натиснути edit, то відправиться http-запит

на backend сервер. В компоненті це можна зробити за допомогою такого кода:

```
sendUser(user) {
      axios.put(`/users/${user._id}`, user)
      .then(response => {
      console.log('User updated successfully:',
response.data);
      })
      .catch(error => {
      console.error('Error updating user:',
error.response.data);
});
```

Цей метод переходить по маршруту router.put('/:id', userController.updateUser); де, власне, вже бекенд викликає метод updateUser(), який оновлює ці дані.

Або, якщо було натиснуто на кнопку Delete, тоді, аналогічно, виглядає функція так:

```
deleteUser(userId) {
      axios.delete(`/users/${userId}`)
      .then(response => {
     console.log('User deleted successfully');
      Handle the response if needed
       })
      .catch(error => {
      console.error('Error deleting user:',
error.response.data);
      Handle the error if needed
      });
  }
```

Цей метод переходить по маршруту router.delete('/:id', userController.deleteUser); де, власне, вже бекенд викликає метод deleteUser(), який оновлює ці дані.

4. Оновлюються дані в базі. Відповідно цей компонент, як і інші, вбудовується в головну сторінку.

App.vue має будову, схожу на компонентну.

```
<template>
  <div id="app">
    <AdminMenu />
    <UserComponent />
  </div>
</template>
```

Як можна побачити, він має 2 компоненти: <AdminMenu /> та <UserComponent />, в тегу <template> формується структура, компонента.

Далі йде скрипт, в якому ці компоненти імпортуються:

```
<script>
import AdminMenu from './components/AdminMenu.vue'
import UserComponent from './components/UserComponent.vue';
export default {
  name: 'App',
  components: {
    AdminMenu,
    UserComponent
  }
}
</script>
```

Далі йде необов’язковий розділ компоненту зі стилям з тегом <style/>

# 4 ОХОРОНА ПРАЦІ

## 4.1 Загальні питання з охорони праці

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. В законі України «Про охорону праці» [27] визначається, що охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

При роботі з обчислювальною технікою змінюються фізичні і хімічні фактори навколишнього середовища: виникає статична електрика, електромагнітне випромінювання, змінюється температура і вологість, рівень вміст кисню і озону в повітрі. Повітря забруднюється шкідливими хімічними речовинами антропогенного походження за рахунок деструкції полімерних матеріалів, які використовуються для обробки приміщень та обладнання.

Неправильна організація робочого місця сприяє загальному і локальній напрузі м'язів шиї, тулуба, верхніх кінцівок, викривлення хребта і розвитку остеохондрозу. На всіх підприємствах, в установах, організаціях повинні створюватися безпечні і нешкідливі умови праці. Забезпечення цих умов покладається на власника або уповноважений ним орган (далі роботодавець). Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці.

Роботодавець повинен впроваджувати сучасні засоби техніки безпеки, які запобігають виробничому травматизмові, і забезпечувати санітарно-гігієнічні умови,

що запобігають виникненню професійних захворювань працівників.

Він не має права вимагати від працівника виконання роботи, поєднаної з явною небезпекою для життя, а також в умовах, що не відповідають законодавству про охорону праці. Працівник має право відмовитися від дорученої роботи, якщо створилася виробнича ситуація, небезпечна для його життя чи здоров'я або людей, які його оточують, і навколишнього середовища.

### 4.1.1 Правові та організаційні основи охорони праці

Основним організаційним напрямом у здійсненні управління в сфері охорони праці є усвідомлення пріоритету безпеки праці і підвищення соціальної відповідальності держави, і особистої відповідальності працівників.

Державна політика в галузі охорони праці визначається відповідно до Конституції України Верховною Радою України і спрямована на створення належних, безпечних і здорових умов праці, запобігання нещасним випадкам та професійним захворюванням. Відповідно до статті 3 Закону України «Про охорону праці» [16] (далі – Закону) законодавство про охорону праці складається з Закону, Кодексу законів про працю України [17], Закону України "Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності" [18] та прийнятих відповідно до них нормативно-правових актів, норм міжнародного договору (ратифіковані Конвенції і Рекомендації МОТ, директиви Європейської Ради).

На законодавчому рівні визначено такі пріоритетні напрямки з безпеки праці:

– кожен працівник несе безпосередню відповідальність за порушення зазначених Законом, нормами і правилами вимог;

– напрямки реалізації конституційного права громадян на їх життя і здоров’я в процесі трудової діяльності:

– пріоритет життя і здоров’я працівників по відношенню до результатів виробничої діяльності підприємства;

– повна відповідальність роботодавця за створення належних – безпечних і здорових умов праці;

– соціальний захист працівників, повне відшкодування збитків особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань;

– комплексне розв’язання завдань охорони праці;

– підвищення рівня промислової безпеки шляхом забезпечення суцільного технічного контролю за станом виробництв, технологій та продукції, а також сприяння підприємствам у створенні безпечних та нешкідливих умов праці;

– соціальний захист працівників, повне відшкодування збитків особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань;

– використання економічних методів управління охороною праці, участь держави у фінансуванні заходів щодо охорони праці;

– використання світового досвіду організації роботи щодо поліпшення умов і підвищення безпеки праці на основі міжнародної співпраці.

Користувачі персональних комп’ютерів, для яких ця робота є головною, підлягають медичним оглядам: попереднім — під час влаштування на роботу і періодичним — протягом професійної діяльності раз на два роки. Жінок з часу встановлення вагітності та в період годування дитини грудьми до роботи з ПК не допускають.

Наявні трудові відносини між працівниками і роботодавцями в Україні за темою дипломного проекту регулюються Кодексом законів про працю (КЗпП) України, відповідно до якого права працюючої людини на охорону праці охороняються всебічно та норми охорони праці неухильно інтегровані до правил внутрішнього розпорядку організації/підприємства.

### 4.1.2 Організаційно-технічні заходи з безпеки праці

В організації/підприємстві проводиться навчання і перевірка знань з питань охорони праці відповідно до вимог Типового положення про порядок проведення

навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнаглядохоронпраці України від 26.01.2005 N 15, зареєстрованого в Міністерстві юстиції України 15.02.2005 за N 231/10511 [19].

Обов’язковими вимогами враховане наступне:

– ознайомлення з правилами безпеки праці, одержання відповідних інструктажів засвідчується у журналі інструктажів.

– перед допуском до самостійної роботи кожен працівник має право на навчання з питань охорони праці і роботодавець зобов’язаний, і проводить таке навчання у вигляді двох інструктажів з питань охорони праці:

1) вступного, який проводять працівники служби охорони праці об’єкта господарювання з усіма працівниками, яких приймають на роботу незалежно від їхньої освіти та стажу роботи за програмою, в якій подають загальні питання охорони праці із врахуванням її особливостей на об’єкті господарювання;

2) первинного, який проводять керівники структурних підрозділів на місці праці з кожним працівником до початку їхньої роботи на цьому робочому місці.

Проходження працівником цих інструктажів з питань охорони праці підтверджується записами у відповідних журналах обліку інструктажів і скріплюється підписами осіб, які проводили інструктажі та осіб, які отримали інструктажі.

3) Повторний (не рідше одного разу в 6 місяців);

4) Позаплановий (при зміні правил охорони праці);

5) Поточний (проводять з працівниками перед виконанням робіт, на яких оформляється наряд-допуск) – обов’язкові організаційні заходи перед початком, під час і після завершення роботи повинні включати перевірку наявності, справності та заземлення електрообладнання, а під час виконання роботи вимогу «не залишати без нагляду обладнання, яке працює». Після закінчення роботи – вимагається прибирання робочого місця, відключення всіх електроприладів від електромережі.

## 4.2 Аналіз стану умов праці

Робота над створенням системи оптимізації запитів до бази даних проходитиме в приміщенні (70794 Filderstadt, Hintere Gasse 18). Для даної роботи достатньо однієї людини, для якої надано робоче місце зі стаціонарним комп'ютером.

### 4.2.1 Вимоги до приміщень

Геометричні розміри приміщення зазначені в табл. 4.1.

Таблиця 4.1 – Розміри приміщення.

| **Найменування** | **Значення** |
|---|---|
| Довжина, м | 6 |
| Ширина, м | 3 |
| Висота, м | 2,5 |
| Площа, $м^2$ | 18 |
| Об'єм, $м^3$ | 45 |

Згідно з вимогами Державних санітарних правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПІН 3.3.2.007-98) [20] розмір площі для одного робочого місця оператора персонального комп'ютера має бути не менше 6 кв. м,а об'єм — не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

Для забезпечення потрібного рівного освітленості кімната має вікно та систему загального рівномірного освітлення, що встановлена на стелі.

Для дотримання вимог пожежної безпеки встановлено порошковий вогнегасник та систему автоматичної пожежної сигналізації.

### 4.2.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця за ДСанПІН 3.3.2.007-98 [20] і

відповідними фактичними значеннями для робочого місця за ДСН 3.3.6.042-99 [21], констатуємо повну відповідність.

Таблиця 4.2 – Характеристики робочого місця

| Найменування параметра | Фактичне значення | Нормативне значення |
|---|---|---|
| Висота робочої поверхні, мм | 750 | 680 ÷800 |
| Висота простору для ніг, мм | 730 | не менше 600 |
| Ширина простору для ніг, мм | 660 | не менше 500 |
| Глибина простору для ніг, мм | 700 | не менше 650 |
| Висота поверхні сидіння, мм | 470 | 400 ÷500 |
| Ширина сидіння, мм | 400 | не менше 400 |
| Глибина сидіння, мм | 400 | не менше 400 |
| Висота поверхні спинки, мм | 600 | не менше 300 |
| Ширина | опорної | поверхні спинки, мм |
| Радіус | кривини | спинки |
| Відстань від очей до екрану дисплея, мм | 800 | 700 ÷800 |

Приміщення кабінету має об’єм 45 м$^3$, площу – 18 м$^2$.

Температура в приміщенні протягом року коливається у межах 18–24°С, відносна вологість – близько 50%. Система вентилювання приміщення — природна неорганізована, а опалення — централізоване.

Розміщення вікон забезпечує природне освітлення з коефіцієнтом природного освітлення не менше 1,5%, а загальне штучне освітлення, яке здійснюється за допомогою восьми люмінесцентних ламп, забезпечує рівень освітленості не менше 200 Лк. За ступенем пожежної безпеки приміщення належить до категорії В.

### 4.2.3 Навантаження та напруженість процесу праці

За фізичним навантаженням робота відноситься до категорії легкі роботи (Іа), її виконують сидячи з періодичним ходінням. Щодо характеру організування виконання дипломної роботи, то він підпадає під нав’язаний режим, оскільки певні

розділи роботи необхідно виконати у встановлені конкретні терміни.

За ступенем нервово-психічної напруги виконання роботи можна віднести до II – III ступеня і кваліфікувати як помірно напружений – напружений за умови успішного виконання поставлених завдань.

Під час виконання робіт використовують ПК та периферійні пристрої (лазерні та струменеві), що призводить до навантаження на окремі системи організму. Такі перекоси у напруженні різних систем організму, що трапляються під час роботи з ПК, зокрема, значна напруженість зорового аналізатора і довготривале малорухоме положення перед екраном, не тільки не зменшують загального напруження, а навпаки, призводять до його посилення і появи стресових реакцій.

Наявні психофізіологічні небезпечні та шкідливі фактори:

а) фізичного перевантаження:

- статичного;
- динамічного;

б) нервово-психічного перевантаження:

- розумового перенапруження;
- монотонності праці;
- перенапруження аналізаторів;
- емоційних перевантажень.

Роботу за дипломним проектом визнано, таку, що займає 50% часу робочого дня та за восьмигодинної робочої зміни рекомендовано встановити додаткові регламентовані перерви тривалістю 15 хв через кожну годину роботи.

### 4.3 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

### 4.3.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00-7.15-18 [24] «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», які встановлюють вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Переважно роботи за проектами виконують у кабінетах чи інших приміщеннях, де використовують різноманітне електрообладнання, зокрема персональні комп'ютери (ПК) та периферійні пристрої.

Основними робочими характеристиками персонального комп'ютера є наступні:

– робоча напруга U = +220В ± 5%;

– робочий струм I = 2А;

– споживана потужність P = 350 Вт.

Робоче місце має відповідати вимогам Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 [31].

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.3).

Таблиця 4.3 – Аналіз небезпечних і шкідливих виробничих факторів

| Небезпечні і шкідливі виробничі фактори | Джерела факторів (види робіт) | Кількісна оцінка | Нормативні документи |
|---|---|---|---|
| ***Фізичні*** | | | |
| - підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини | -//- | 4 | [33] |
| - недостатність природного світла | порушення умов праці (вимог до | 2 | [34] |

| | | | |
|---|---|---|---|
| | приміщень) | | |

Продовження таблиці 4.3

| Небезпечні і шкідливі виробничі фактори | Джерела факторів (види робіт) | Кількісна оцінка | Нормативні документи |
|---|---|---|---|
| - недостатнє освітлення робочої зони | порушення гігієнічних параметрів виробничого середовища | 3 | [34] |
| ***Психофізіологічні:*** | | | |
| - нервово-психічна перевантаження (розумове, перенапруження аналізаторів-зорових) | - пошук інформації для постановки теми;<br>- пошук та аналіз аналогів і літератури;<br>- пошук наявних технологій, мо делювання та аналіз алгоритмів;<br>- виконання роботи за темою ди-плома, тестування;<br>- оформлення роботи | 4 | [35]<br>[31] |
| - фізичні (статичне – сидіння) | порушення умов праці (організації місця праці- сидіння користувача, ) та організації робочого часу - безперервна робота) | 2 | [35]<br>[31] |

### 4.3.2 Пожежна безпека

Приміщення оснащено системою автоматичної пожежної сигналізації, має 1 вогнегасник ВП-5 із зарядом вогнегасної речовини 8-12 кг, відповідно до вимог чинного законодавства України. Проходи до засобів пожежогасіння вільні, не захаращуються та у разі потреби забезпечувати евакуацію всіх людей, які перебувають у приміщенні через один евакуаційний вихід з дверима на шляху евакуації, що відчинятися в напрямку виходу з будівлі від робочого місця. В приміщенні наявна затверджена «План-схема евакуації з кабінету (приміщення)».

Пожежна безпека при застосуванні ЕОМ забезпечується:

1) системою запобігання пожежі,

2) системою протипожежного захисту,

3) організаційно-технічними заходами.

Згідно ДСТУ Б В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» [25] таке приміщення, площею 18 м2, відноситься до категорії "В" (пожежонебезпечної) та для протипожежного захисту в ньому проектом передбачено устаткування автоматичною пожежною сигналізацією із застосуванням датчиків-сповіщувачів РІД-1 (сповіщувач димовий ізоляційний) в кількості 1 шт., і застосуванням первинних засобів пожежогасіння.

Горючими матеріалами в приміщенні, де розташовані ЕОМ, є:

1) поліамід – матеріал корпусу мікросхем, горюча речовина, температура самозаймання 420º С,

2) полівінілхлорид – ізоляційний матеріал, горюча речовина, температура запалювання 335º С, температура самозаймання 530º С,

3) склотекстоліт ДЦ – матеріал друкарських плат, важкогорючий матеріал, показник горючості 1.7А, не схильний до температурного самозаймання,

4) пластикат кабельний №.489 – матеріал ізоляції кабелів, горючий матеріал, показник горючості більше 2.1,

5) деревина – будівельний і обробний матеріал, з якого виготовлені меблі, горючий матеріал, показник горючості більше 2.1, температура запалювання 255º С, температура самозаймання 399º С.

Продуктами згорання, що виділяються на пожежі, є: окис вуглецю; сірчистий газ; окис азоту; синильна кислота; акромін; фосген; хлор і ін. При горінні пластмас, окрім звичних продуктів згорання, виділяються різні продукти термічного розкладання: хлорангідридні кислоти, формальдегіди, хлористий водень, фосген, синильна кислота, аміак, фенол, ацетон, стирол.

## 4.4 Гігієнічні вимоги до параметрів виробничого середовища

### 4.4.1 Освітлення

Збільшення освітленості сприяє поліпшенню працездатності навіть в тих випадках, коли процес праці практично не залежить від зорового сприйняття. При поганому освітленні людина швидко втомлюється, працює менш продуктивно, виникає потенційна небезпека помилкових дій і нещасних випадків.

Освітленість приміщення має велике значення при роботі на ПЕОМ. Вона багато в чому визначається колірною і мережевий обстановкою. Для зменшеного поглинання світла стеля і стіни вище панелей (1,5-1,7м.). Якщо вони не облицьовані звукопоглинальним матеріалом, фарбуються білою водоемульсійною фарбою (коефіцієнт відбиття повинен бути не менше 0,7). Для забарвлення стіни панелей рекомендується віддавати перевагу світлим фарбам.

Природне освітлення, коли робочі місця з ПЕОМ розташовуються в один ряд по довжині приміщення на відстані 0,8 - 1,0 м від стіни з віконними прорізами, і екрани знаходяться перпендикулярно цієї стіни. Основний потік природного світла при цій повинен бути зліва. Не допускається спрямування основного світлового потоку природного світла праворуч, ззаду і спереду працює на ПЕОМ. Оптимальна відстань очей до екрана відео монітора повинна становити 60-70 см, допустиме не менше 50 см. Розглядати інформацію ближче 50 см не рекомендується.

У приміщенні, де розташовані ЕОМ передбачається природне бічне освітлення, рівень якого відповідає ДБН В.2.5-28:2018 [23]. Джерелом природного освітлення є сонячне світло. Регулярно повинен проводитися контроль освітленості, який підтверджує, що рівень освітленості задовольняє [21] і для даного приміщення в світлий час доби достатньо природного освітлення.

Розрахунок освітлення.

Для будівель виробництв світловий коефіцієнт приймається в межах 1/6 - 1/10:

$$\sqrt{a^2 + b^2 * S_b} = \left({}^1\!/_8 \div {}^1\!/_{10}\right) * S_{n,} \quad (4.1)$$

де $S_b$ – площа віконних прорізів, м$^2$;

$S_n$ – площа підлоги, м$^2$

$$S_{п} = a * b = 6 * 3 = 18м^2$$

$$S_{вік} = 1/8 * 18 = 2,25м^2$$

Приймаємо 2 вікна площею S = 1,13м$^2$ кожне.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників *n* виробляється по формулі (4.2):

$$n = \frac{E * S * Z * K}{F * U * M} \quad (4.2)$$

де *E* – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м$^2$; S = 18 м$^2$;

*Z* – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ;

Z = 1,1для люмінесцентних ламп) приймаємо рівним 1,1;

*K* – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації

– 1,5;

*U* – коефіцієнт використання, залежний від типу світильника, показника індексу

приміщення і т.п. – 0,575

*M* – число люмінесцентних ламп в світильнику – 2;

*F* – світловий потік лампи – 5400лм (для ЛБ-80). Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300*18*1,1*1,5}{5400*0,575*2} = 1$$

Приймаємо освітлювальну установку, яка складається з одного світильника, який складаються з 2-х люмінесцентних ламп загальною потужністю 160 Вт, напругою – 220 В.

### 4.4.2 Вентилювання

Здійснюється провітрювання приміщення, в залежності від погодних умов, тривалість по- винна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

## 4.5 Розрахунок захисного заземлення (забезпечення електробезпеки будівлі).

Загальний опір захисного заземлення визначається за формулою:

$$R_{ззп} = \frac{R_з * R_n}{R_n * n * \eta_з + R_з * \eta_n}, \quad (4.3)$$

де $R_з$ – опір заземлення, якими когут бать труби, опори, кути і т.п., Ом;

$R_n$ – опір опори, яке з'єднує заземлювачі, Ом;

п – кількість заземлювачів;

$\eta_з$ – коефіцієнт екранування заземлювача; приймається в межах 0,2 ÷ 0,9; $\eta_з$ = 0,7

$\eta_n$ – коефіцієнт екранування сполучної стійки; приймається в межах 0,1 ÷ 0,7; $\eta_n$ = 0,5;

Опір заземлення визначається за формулою:

$$R_з = \frac{\rho}{2\pi * 1} * \left(1n\frac{2*1}{d} + \frac{1}{2}1n\frac{4*t+1}{4*t-1}\right), \quad (4.4)$$

де ρ – питомий опір ґрунту, залежить від типу грунту, Ом·м; для піску – 400 ÷

700 Ом·м; приймаємо ρ = 400 Ом·м;

l – довжина заземлювача, м; для труб – 2-3 м; l = 3 м;

d – діаметр заземлювача, м; для труб – 0,03-0,05 м; d = 0,05 м;

t – відстань від середини забитого в ґрунт заземлювача до рівня землі, м; t = 2 м.

$$R_з = \frac{400}{2 * 3{,}14 * 3}\left(1n\frac{2 * 3}{0{,}05} + \frac{1}{2}1n\frac{4 * 2 + 3}{4 * 2 - 3}\right) = 110, \text{Ом}$$

Опір смуги, що з'єднує заземлювачі, визначається за формулою:

$$R_n = \frac{\rho}{2\pi * L} * 1n\frac{2 * L^2}{b * t^1}, \quad (4.5)$$

де L – довжина смуги, що з'єднує заземлювачі (м) і приблизно дорівнює периметру будівлі: $П_{буд.}$ = 42·2 + 38·2 = 160 м; L = 160 м;

b – ширина смуги, м; b = 0,03 м;

$t_1$ – глибина заземлення від рівня землі, м; $t_1$ = 0,5 м.

$$R_n = \frac{400}{2 * 3{,}14 * 160} * 1n\frac{2 * 160^2}{0{,}03 * 0{,}5} = 5{,}00 \text{ Ом}$$

Кількість заземлювачів захисного заземлення визначається за формулою:

$$n = \frac{2 * R_з}{4 * \eta_з}, \quad (4.6)$$

де 4 – допустимий загальний опір, Ом;

2 – коефіцієнт сезонності.

Визначаємо загальний опір захисного заземлення:

$$R_{ззп} = \frac{110 * 5{,}99}{5{,}99 * 79 * 0{,}7 + 110 * 0{,}5} = 1{,}7 \text{ Ом}$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{ззп} < 4$ Ом.

3) При виникненню пожеж при роботі на ПЕОМ від таких можливими джерел запалювання як:

– іскри і дуги коротких замикань;

– перегрів провідників, резисторів та інших радіодеталей ПЕОМ, від тривалої перевантаження та наявність перехідного опору;

– іскри при розмиканні і розмиканні ланцюгів;

– розряди статичної електрики;

– необережному поводженню з вогнем, а також вибухи газоповітряних і паро-повітряних сумішей.

### 4.6 Висновки до четвертого розділу

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію.

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в дипломній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника.

Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Була наведені розміри приміщення та наведено значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника.

На підставі аналізу розроблені заходи з техніки безпеки, організації вимог освітлення робочого місця та рекомендації з пожежної профілактики.

# ВИСНОВКИ

У даній дипломній роботі було успішно розроблено web-додаток для мережі ресторанів з використанням технологій Node.js та Vue.js. Додаток має серверну архітектуру, базу даних MongoDB та забезпечує розділення на окремі розділи для клієнтів та персоналу. Було реалізовано API для інтеграції з іншими бізнес-сервісами.

Основними сутностями веб-додатку є користувачі, меню, продукти, замовлення, статуси та рахунки. Користувачам надається можливість замовлення столика, вибору блюд з меню та перегляду рахунку. Перед прибуттям в заклад, клієнти можуть зарезервувати столик та обрати блюда з меню. Замовлення передається на кухню, де співробітники підтверджують початок готування. Зміна статусів замовлення відбувається відповідно до його прогресу - від підтвердження готування до доставки та оплати.

Розроблений додаток надає зручний та ефективний інтерфейс для користувачів ресторану та співробітників. Він дозволяє здійснювати замовлення та контролювати процес його виконання. Додаток може бути використаний як базова платформа для розширення та додавання додаткового функціоналу відповідно до потреб ресторанної мережі.

У підсумку, розроблений web-додаток відповідає технічному завданню та забезпечує зручну та ефективну роботу мережі ресторанів. Використання сучасних технологій та бази даних MongoDB дозволяє забезпечити швидку та надійну роботу додатку. Дана робота відкриває перспективи для подальшого розвитку та удосконалення функціоналу веб-додатку для задоволення зростаючих потреб ресторанної індустрії.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Node.js [Електронний ресурс] - Офіційний веб-сайт Node.js, який надає інформацію про цю технологію для серверної архітектури та роботи з базою даних. Режим доступу до ресурсу: https://nodejs.org/

2. Vue.js [Електронний ресурс] - Офіційний веб-сайт Vue.js, де ви можете знайти документацію та приклади для впровадження користувацького інтерфейсу. Режим доступу до ресурсу: https://vuejs.org/

3. MongoDB [Електронний ресурс] - Офіційний веб-сайт MongoDB, який надає детальну інформацію про цю базу даних. Режим доступу до ресурсу: https://www.mongodb.com/

4. REST API [Електронний ресурс] - Ресурс, що пояснює концепцію та принципи проектування REST API, який ви зможете використати для реалізації інтеграції з іншими сервісами для бізнесу. Режим доступу до ресурсу: https://restfulapi.net/

5. Vue Router [Електронний ресурс] - Офіційна документація Vue Router, яка допоможе вам розділити веб-додаток на окремі розділи для клієнтів та персоналу. Режим доступу до ресурсу: https://router.vuejs.org/

6. Vue CLI [Електронний ресурс] - Офіційний веб-сайт Vue CLI, де ви можете знайти інформацію про цей інструмент для розробки веб-додатків з Vue.js. Режим доступу до ресурсу: https://cli.vuejs.org/

7. Express.js [Електронний ресурс] - Офіційний веб-сайт Express.js, який надає детальну інформацію про цей фреймворк для розробки серверних додатків на Node.js. Режим доступу до ресурсу: https://expressjs.com/

8. Mongoose [Електронний ресурс] - Офіційна документація Mongoose, яка допоможе вам працювати з MongoDB в додатку на Node.js. Режим доступу до ресурсу: https://mongoosejs.com/

9. MDN Web Docs [Електронний ресурс] - Веб-документація MDN надає широкий огляд WEB, HTTP, браузерів та веб-розробки загалом. Режим доступу до ресурсу: https://developer.mozilla.org/

10. W3Schools [Електронний ресурс] - Веб-сайт W3Schools містить багато матеріалів із WEB, HTTP, HTML, CSS та JavaScript, а також приклади коду та посібники. Режим доступу до ресурсу: https://www.w3schools.com/

11. HTTP - Hypertext Transfer Protocol [Електронний ресурс] - Розділ про HTTP на сайті MDN, де ви можете знайти детальну інформацію про цей протокол, який використовується в WEB. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP

12. Browser Rendering Optimization [Електронний ресурс] - Стаття на сайті Google Developers, яка розглядає оптимізацію візуалізації веб-сторінок в браузерах і надає корисні поради та практики. Режим доступу до ресурсу: https://developers.google.com/web/fundamentals/performance/rendering

13. Web Application Architecture [Електронний ресурс] - Стаття на сайті Towards Data Science, яка висвітлює різні архітектури веб-додатків та їх особливості. Режим доступу до ресурсу: https://towardsdatascience.com/web-application-architecture-101-a9d651544b18

14. RESTful API Design - Best Practices [Електронний ресурс] - Стаття на сайті Smashing Magazine, яка надає кращі практики для проектування RESTful API, що може бути корисним при реалізації API у вашому веб-додатку. Режим доступу до ресурсу: https://www.smashingmagazine.com/2020/05/best-practices-rest-api-design/

15. Microservices Architecture [Електронний ресурс] - Стаття на сайті Microsoft Docs, яка пояснює концепцію та переваги мікросервісної архітектури веб-додатків. Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/

16. Закон України "Про охорону праці". Вводиться в дію Постановою ВР № 2695-XII від 14.10.92, ВВР, 1992, № 49, ст.669. - Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/2694-12

17. Кодекс законів про працю України. Затверджується Законом № 322-VIII від 10.12.71 ВВР, 1971. Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/322-08

18. Закон України "Про загальнообов'язкове державне соціальне страхування

від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності". Наказ від 21 грудня 2000 року N2180-III. Режим доступу: www. URL: https://dnaop.com/html/2065/doc-zakon-ukrajini-pro-zagalynoobovjazkove-derzhavne-socialyne- strahuvannya-vid-neshhasnogo-vipadku-na-virobnictvi-ta-profesijnogo-z

19. Про затвердження Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці НПАОП 0.00-4.12-05. Наказ від 26.01.2005 №15. Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/z0231-05

20. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98. Затверджено Постановою Головного державного санітарного лікаря України 10 грудня 1998 р. N 7. Режим доступу: www. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98

21. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. Постанова N 42 від 01.12.99. Режим доступу: www. URL: https://zakon.rada.gov.ua/rada/show/va042282-99

22. Електробезпека в будівлях і спорудах. Вимоги до захисних заходів від ураження елек- тричним струмом. Наказ від 1 липня 2016 року N 204. Режим доступу: www. URL: http://epicentre.co.ua/dstu/doc28522.html

23. ДБН В.2.5-28:2018 «Природне і штучне освітлення». Режим доступу: www. URL: http://www.minregion.gov.ua/wp-content/uploads/2018/12/V2528-1.pdf

24. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час ро- боти з екранними пристроями». Зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за № 508/31960. Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/z0508-18

25. ДСТУ Б В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх уста- новок за вибухопожежною та пожежною небезпекою». Наказ від 15.06.2016 №158. Режим досту- пу: www. URL: https://zakon.rada.gov.ua/rada/show/v0158858-1

**Додаток А**

Презентація

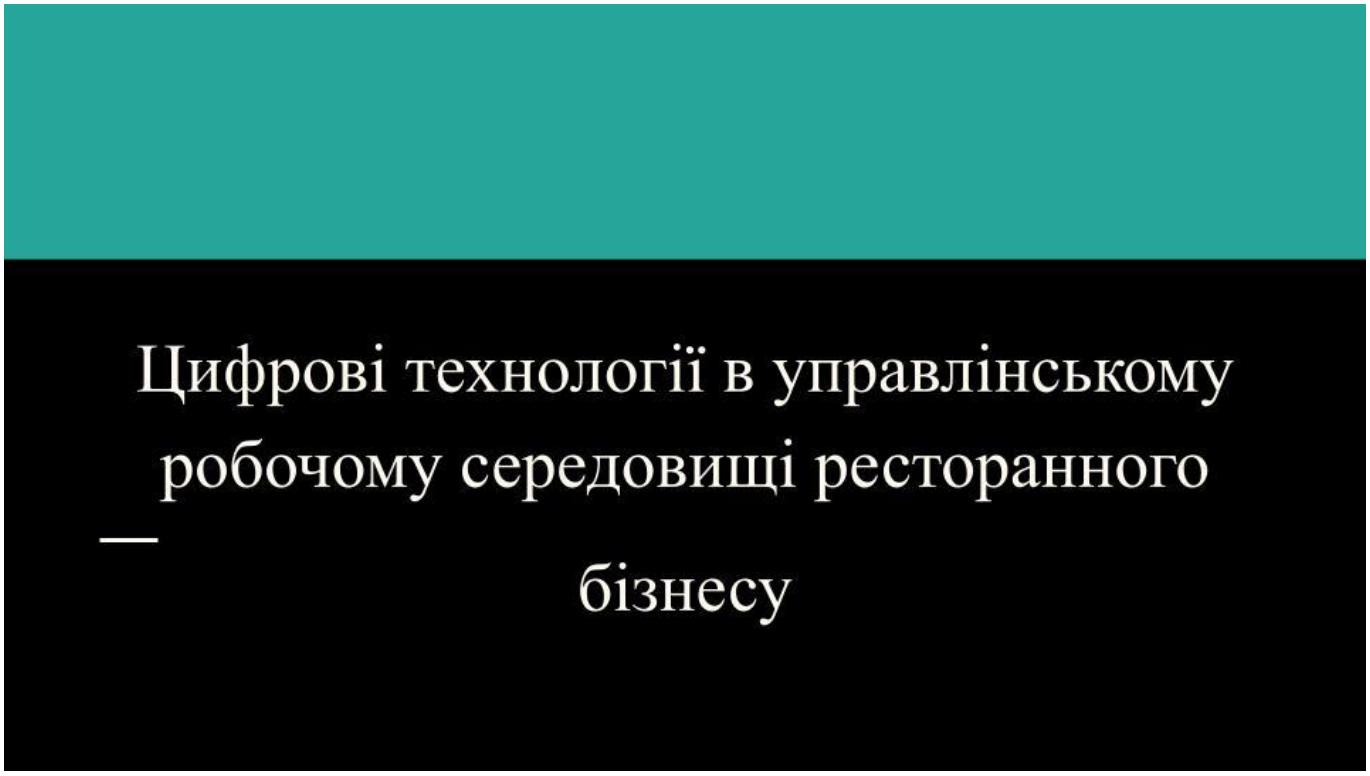



Рисунок А.1 – Слайд 1

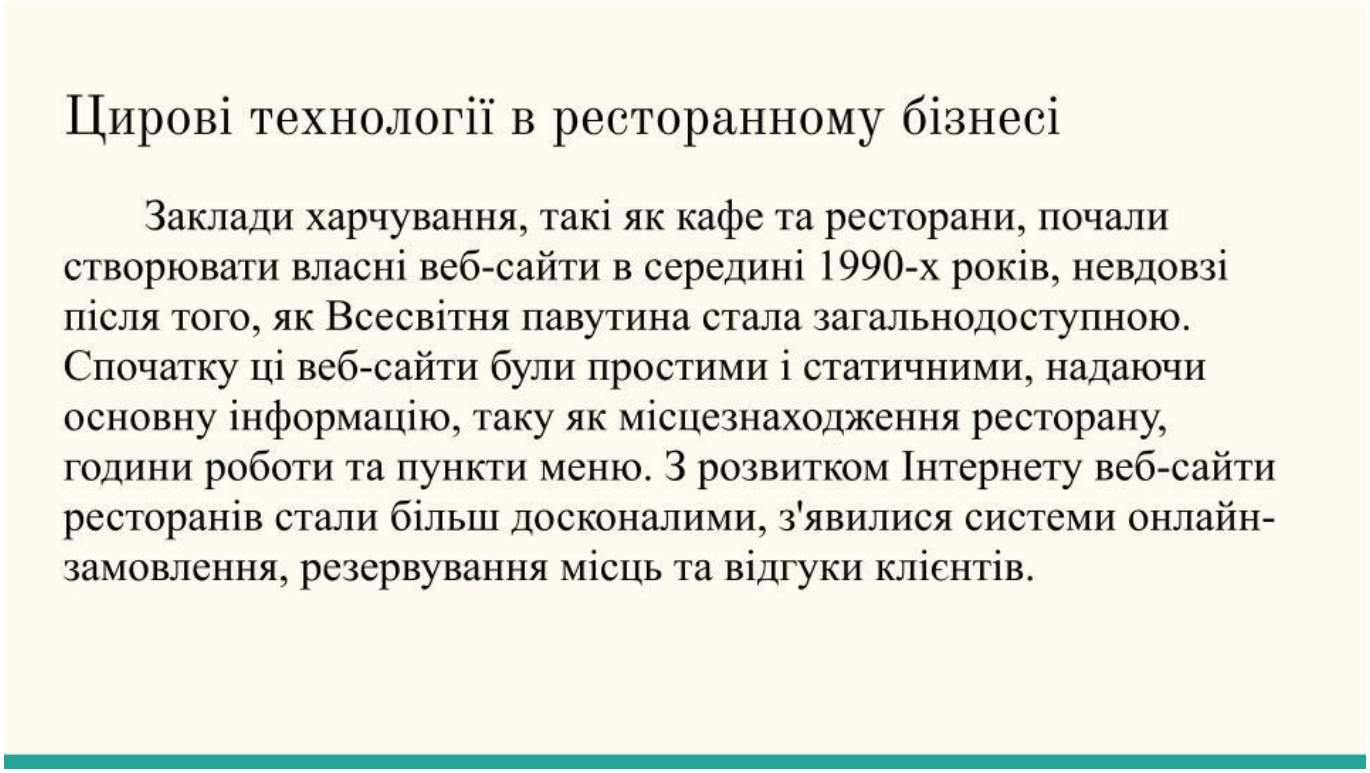


Рисунок А.2 – Слайд 2

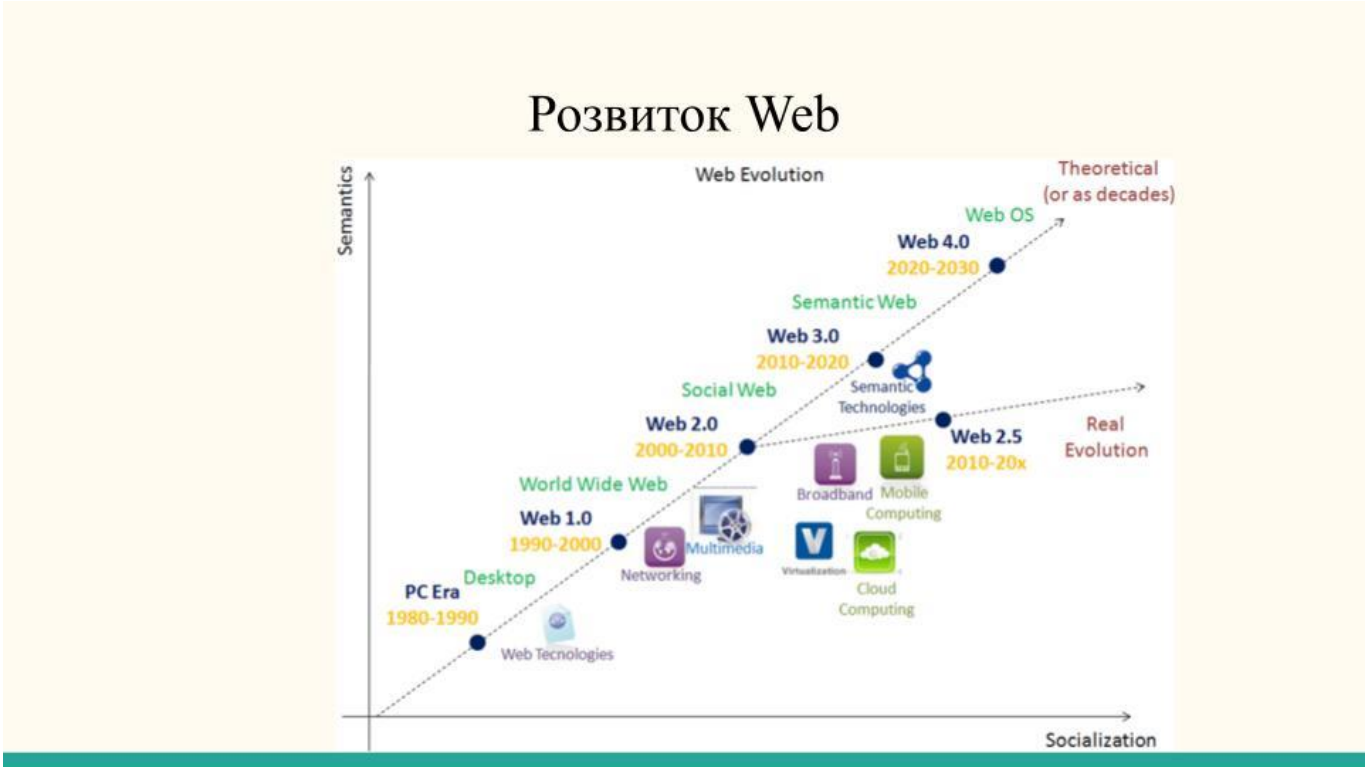



Рисунок А.3 – Слайд 3

Рисунок А.4 – Слайд 4

## Вибір платформи розробки

Середня вартість створення простого бізнес-сайту коливається від $5000 до $10 000. Середня вартість розробки базового додатку для iOS/Android становить від $20 тис. до $60 тис. для кожної платформи.

Але бувають випадки, коли необхідно розробляти саме десктопний та мобільний додаток, це той випадок, коли треба забезпечити активну взаємодію з пристроєм, або операційною системою, але це не випадок цієї дипломної роботи, тому було обрано саме веб-додаток.

Рисунок А.5 – Слайд 5

## Архітектура Web-додатку

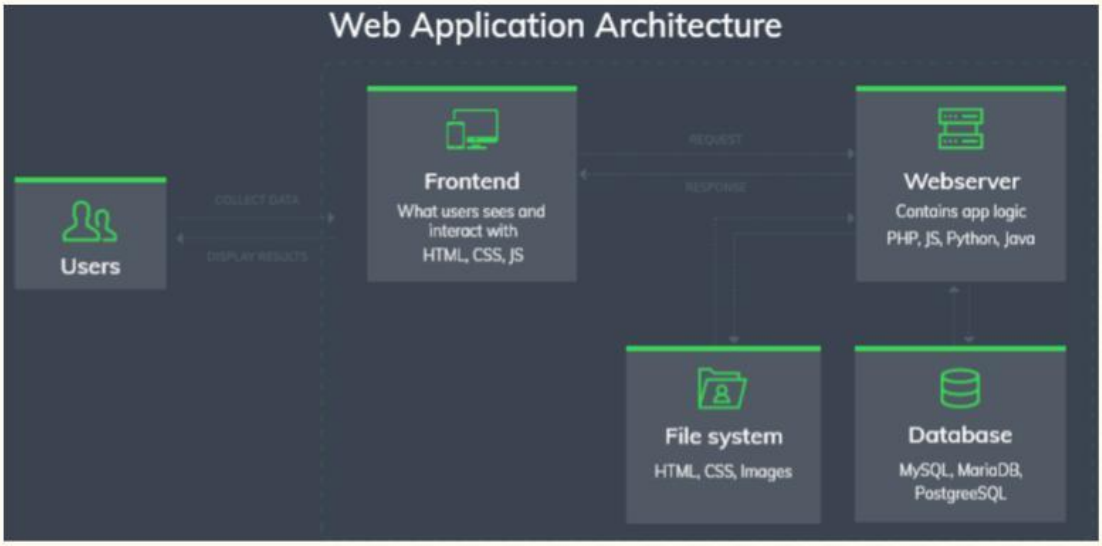



Рисунок А.6 – Слайд 6

Рисунок А.7 – Слайд 7

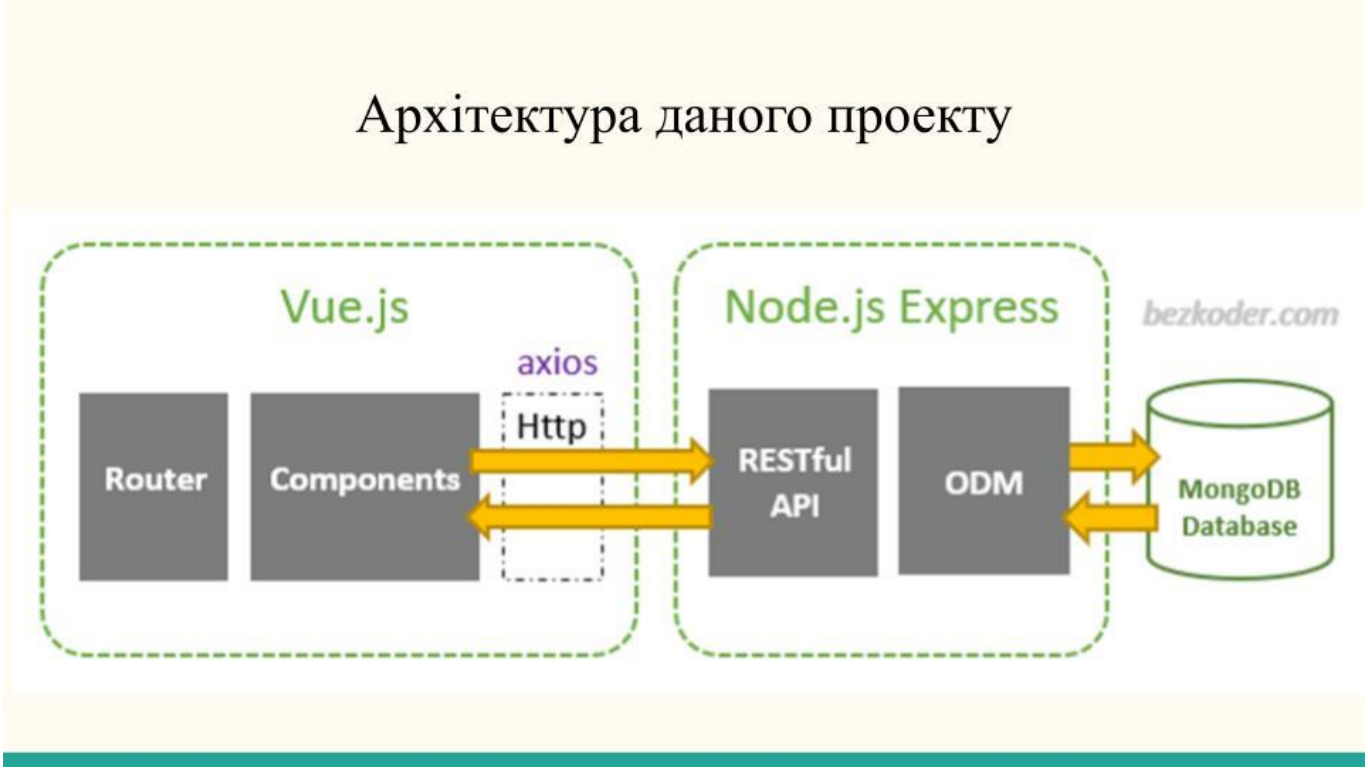



Рисунок А.8 – Слайд 8

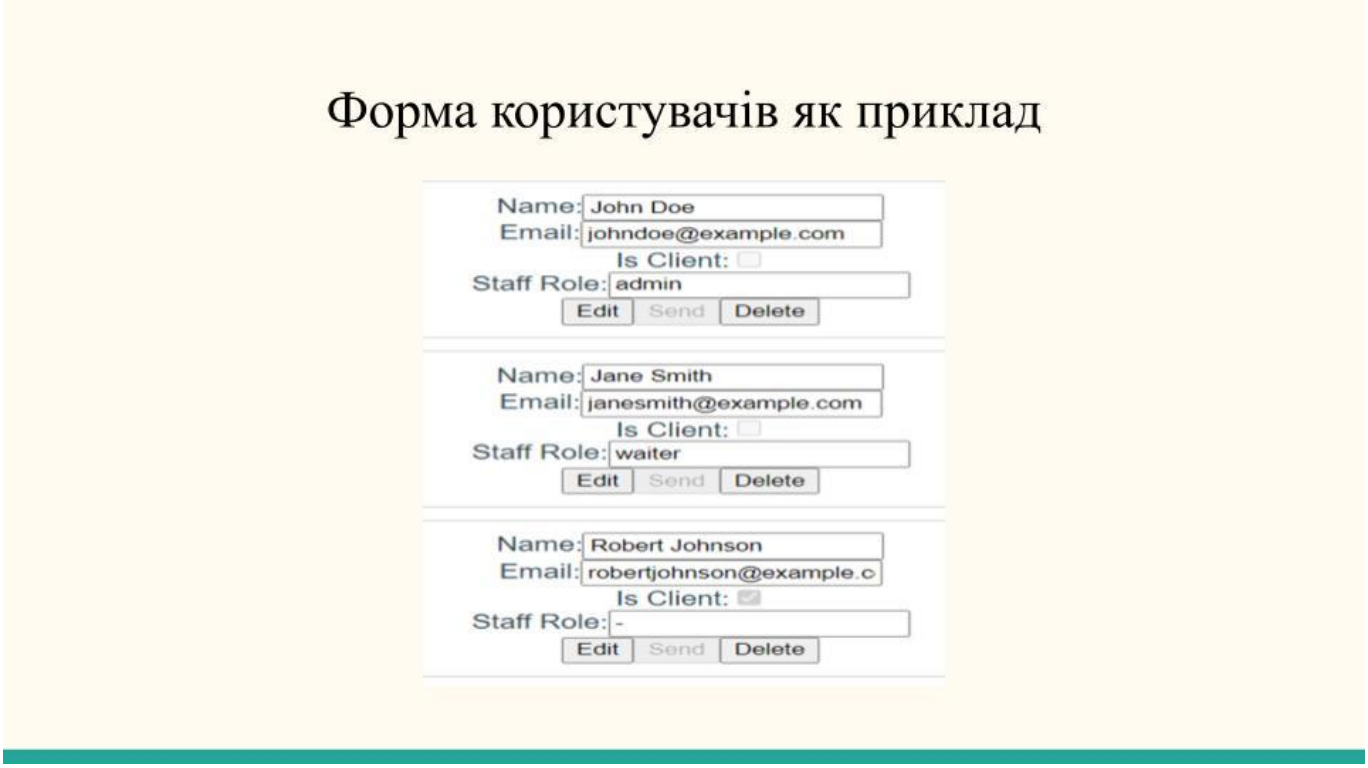
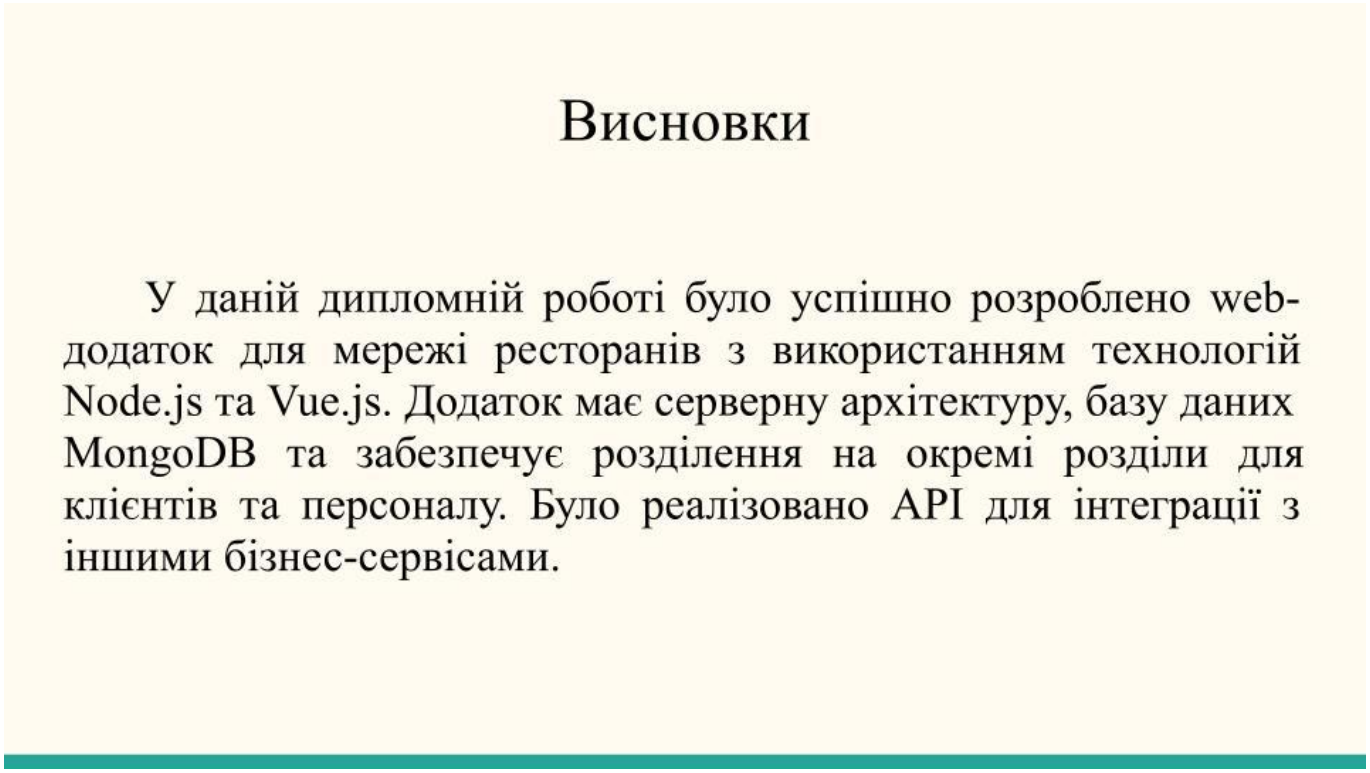


Рисунок А.9 – Слайд 9



# Додаток Б

## Програмний код проекта

## userController.js

```
const User = require('../models/user');
const bcrypt = require('bcrypt');
// Controller actions

// Get all users
exports.getAllUsers = (req, res) => {
    User.find({}, { password: 0}) // Excluding password and age fields
        .then(users => res.json(users))
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};

// Get user by ID
exports.getUserById = (req, res) => {
    const userId = req.params.id;
    User.findById(userId, { password: 0 }) // Excluding password field
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found' });
            }
            res.json(user);
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};

// Update a user
exports.updateUser = (req, res) => {
    const userId = req.params.id;
    const { name, email, isClient, staffRole } = req.body;
    User.findByIdAndUpdate(userId, { name, email, isClient, staffRole }, { new: true })
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found' });
            }
            res.json(user);
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};

// Delete a user
```

```
exports.deleteUser = (req, res) => {
    const userId = req.params.id;
    User.findByIdAndDelete(userId)
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found' });
            }
            res.json({ message: 'User deleted successfully' });
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });

};

exports.registerUser = (req, res) => {
    const { name, email, password } = req.body;

    // Check if the user with the provided email already exists
    User.findOne({ email })
        .then(existingUser => {
            if (existingUser) {
                return res.status(400).json({ error: 'Email already registered' });
            }

            // Hash the password
            bcrypt.hash(password, 10, (err, hashedPassword) => {
                if (err) {
                    console.error(err);
                    return res.status(500).json({ error: 'Server error' });
                }

                // Create a new user with hashed password
                const newUser = new User({ name, email, password: hashedPassword });
                newUser.save()
                    .then(user => res.json(user))
                    .catch(error => {
                        console.error(error);
                        res.status(500).json({ error: 'Server error' });
                    });
            });
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};

// User login
exports.loginUser = (req, res) => {
    const { email, password } = req.body;
```

```
    // Find the user by email
    User.findOne({ email })
        .then(user => {
            if (!user) {
                return res.status(404).json({ error: 'User not found' });
            }

            // Compare the provided password with the stored hashed password
            bcrypt.compare(password, user.password, (err, result) => {
                if (err) {
                    console.error(err);
                    return res.status(500).json({ error: 'Server error' });
                }

                if (!result) {
                    // Passwords do not match
                    return res.status(401).json({ error: 'Invalid password' });
                }

                // Login successful
                res.json({ message: 'Login successful', user });
            });
        })
        .catch(error => {
            console.error(error);
            res.status(500).json({ error: 'Server error' });
        });
};
```

## user.js

```
const mongoose = require('mongoose');
const staffRoleEnum = ['admin', 'chef', 'waiter']
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
    isClient: {
        type: Boolean,
        required: true,
        default: 1
    },
```

```
    staffRole: {
        type: String,
        enum: staffRoleEnum,
        required: function() {
            return !this.isClient;
        }
    }
    // Add other fields as needed
});
userSchema.pre('validate', function(next) {
    if (this.isClient && this.staffRole) {
        next(new Error('Staff role should not be set for a client.'));
    } else if (!this.isClient && !this.staffRole) {
        next(new Error('Staff role is required for non-client users.'));
    } else {
        next();
    }
});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

## users.js

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

// Routes
router.get('/', userController.getAllUsers);
router.get('/:id', userController.getUserById);
//router.post('/', userController.createUser);
router.put('/:id', userController.updateUser);
router.delete('/:id', userController.deleteUser);
// Create a new user
router.post('/register', userController.registerUser);
// User login
router.post('/login', userController.loginUser);

module.exports = router;
```

## UserComponent.vue

```
<template>
  <div>
    <div v-for="user in users" :key="user.id" class="user-card">
      <div>
        <label for="name">Name:</label>
        <input type="text" v-model="user.name" :readonly="!user.editable" />
      </div>
      <div>
        <label for="email">Email:</label>
        <input type="email" v-model="user.email" :readonly="!user.editable" />
```

```
      </div>
      <div>
        <label for="isClient">Is Client:</label>
        <input type="checkbox" v-model="user.isClient" :disabled="!user.editable" />
      </div>
      <div>
        <label for="staffRole">Staff Role:</label>
        <input type="text" v-model="user.staffRole" :readonly="!user.editable" />
      </div>
      <div>
        <button @click="toggleEdit(user)">
          {{ user.editable ? 'Cancel' : 'Edit' }}
        </button>
        <button @click="sendUser(user)" :disabled="!user.editable">Send</button>
        <button @click="deleteUser(user.id)">Delete</button>
      </div>
    </div>
  </div>
</template>

<script>
import axios from 'axios';

export default {
  data() {
    return {
      users: [],
    };
  },
  created() {
    this.fetchUsers();
  },
  methods: {
    fetchUsers() {
      axios.get('/users')
          .then(response => {
            this.users = response.data;
          })
          .catch(error => {
            console.error('Error fetching users:', error.response.data);
            // Handle the error if needed
          });
    },
    sendUser(user) {
      axios.put(`/users/${user.id}`, user)
          .then(response => {
            console.log('User updated successfully:', response.data);
            // Handle the response if needed
          })
          .catch(error => {
            console.error('Error updating user:', error.response.data);
            // Handle the error if needed
          });
```

```
    },
    deleteUser(userId) {
      axios.delete(`/users/${userId}`)
          .then(response => {
            console.log('User deleted successfully');
            // Handle the response if needed
          })
          .catch(error => {
            console.error('Error deleting user:', error.response.data);
            // Handle the error if needed
          });
    },
    toggleEdit(user) {
      user.editable = !user.editable;
    },
  },
};
</script>

<style>
.user-card {
  border: 1px solid #ccc;
  padding: 10px;
  margin-bottom: 10px;
}
</style>
```

App.vue

```
<template>
  <div id="app">
    <AdminMenu />
    <UserComponent />
  </div>
</template>
<script>
import AdminMenu from './components/AdminMenu.vue'
import UserComponent from './components/UserComponent.vue';
export default {
  name: 'App',
  components: {
    AdminMenu,
    UserComponent
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
```

```
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

## router.js

```
import Vue from 'vue';
import VueRouter from 'vue-router';

Vue.use(VueRouter);

const routes = [
    { path: '/admin/design', component: DesignComponent },
    { path: '/admin/users', component: UsersComponent },
    { path: '/admin/orders', component: OrdersComponent },
    { path: '/admin/menu', component: MenuComponent },
    { path: '/admin/point-information', component: PointInformationComponent },
    { path: '/admin/statuses', component: StatusesComponent },
    { path: '/admin/products', component: ProductsComponent }
];

const router = new VueRouter({
    routes
});

export default router;
```