

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ**
З ПОШУКУ РЕЦЕПТІВ

Виконав: студент 4 курсу, групи **КНТ-520**
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

КОТЕЛЕВСЬКИЙ Д.С.

(ПРИЗВИЩЕ та ініціали)

Керівник **СКРУПСЬКИЙ С.Ю.**

(ПРИЗВИЩЕ та ініціали)

Рецензент **КОЗІНА Г.Л.**

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
“ 14 ” травня 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

КОТЕЛЕВСЬКОГО Данила Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Розробка комп'ютерної системи з пошуку рецептів
- керівник проєкту (роботи) к. т. н., доцент, СКРУПСЬКИЙ Степан Юрійович
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від “01” квітня 2024 року № 105
- Строк подання студентом проєкту (роботи) 24 травня 2024 року
- Вихідні дані до проєкту (роботи) наявні методи та алгоритми з пошуку рецептів та побудови вебсайтів
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
 - Визначення технологій розробки
 - Моделювання системи;
 - Розробка системи;
 - Випробування комп'ютерної системи
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
 - Діаграма використання;
 - Діаграма послідовності обробки запиту на автентифікацію;
 - Інтерфейс вебсайту

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	СКРУПСЬКИЙ С. Ю., к.т.н., доцент		
нормоконтроль	ПОЛЬСЬКА О. В., ст. викл.		

7. Дата видачі завдання 25.02.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів пошуку об'єктів, перегляд отриманих результатів, виділення недоліків та переваг	17.03.2024 р.	
2	Розробка структури системи	20.03.2024 р.	
3	Розробка алгоритму роботи системи	23.03.2024 р.	
4	Реалізація системи уточненого розпізнавання	27.03.2024 р.	
5	Дослідження системи уточненого розпізнавання	01.04.2024 р.	
6	Оформлення отриманих результатів у ПЗ	15.04.2024 р.	
7	Оформлення графічного матеріалу	02.05.2024 р.	
8	Оформлення допоміжного матеріалу	20.05.2024 р.	

Студент (ка)


(підпис)

Данило КОТЕЛЕВСЬКИЙ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)

Степан СКРУПСЬКИЙ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 78 стор., 16 рис., 6 табл., 1 додаток, 32 джерела.

ВЕБСАЙТ, ПОШУК РЕЦЕПТІВ, КЛІЄНТ, СЕРВЕР, NEXT.JS, POSTGRESQL, ORM PRISMA

Розроблений вебсайт з пошуку рецептів містить серверну і клієнтську частини.

Клієнтська частина відображає користувацький інтерфейс, завдяки якому користувачі можуть взаємодіяти з системою. Реагуючи на дії користувачів, клієнтська частина відправляє запити на сервер та відображає отримані результати. Серверна частина приймає запити від клієнта, обробляє їх, здійснюючи запити до бази даних або стороннього API, та надсилає відповіді клієнту.

Архітектура клієнт-сервер, яка використовується в даному проєкті, широко застосовується в мережевих технологіях, що забезпечує доступ до різноманітних мережевих сервісів.

ЗМІСТ

Вступ.....	7
1 Визначення технологій розробки	8
1.1 Вибір архітектури системи	8
1.1.1 Монолітна архітектура	8
1.1.2 Мікросервісна архітектура.....	9
1.1.3 Архітектура клієнт-сервер	10
1.1.4 Serverless архітектура	11
1.2 Вибір технологій для фронтенду та бекенду	12
1.2.1 React	13
1.2.2 Angular	14
1.2.3 Vue.js	15
1.2.4 Django.....	16
1.2.5 Laravel	16
1.2.6 Next.js.....	17
1.2.7 Nuxt.js.....	17
1.3 Вибір бази даних	18
1.3.1 Oracle SQL Developer	20
1.3.2 MySQL Workbench	21
1.3.3 Microsoft SQL Server	22
1.3.4 PostgreSQL	22
1.4 Висновки з аналізу технологій	23
2 Моделювання системи	24
2.1 Загальні вимоги до системи.....	24
2.2 Вимоги до користувача	25
2.3 Функціональні вимоги системи.....	26
2.4 Діаграма використання	27
2.5 Діаграма послідовностей	28

2.6 Моделювання бази даних	30
3 Розробка системи	32
3.1 Архітектура середовища розробки.....	32
3.2 Налаштування середовища розробки.....	34
3.2.1 Налаштування препроцесору Sass	34
3.2.2 Глобальні змінні стилів	35
3.2.3 Налаштування пакету для роботи з SVG форматом.....	37
3.3 Імплементация бази даних.....	38
3.4 Розробка користувацького інтерфесу.....	41
3.4.1 Налаштування загального макету сайту	41
3.4.2 Створення загальних компонентів Header та Footer.....	42
3.4.3 Створення компонентів UI.....	44
3.4.4 Створення компонентів сторінок	50
3.5 Імплементация реєстрації та автентифікації.....	54
3.6 Динамічне формування контенту.....	58
4 Випробування комп'ютерної системи	60
4.1 Випробування користувацького інтерфейсу	60
4.2 Функціональне випробування	65
Висновки.....	68
Перелік джерел посилання	69
Додаток А Основні коди програмних модулів	72

ВСТУП

У сучасному світі, де інтернет-технології глибоко проникають у всі сфери нашого життя, з'являється безліч можливостей для інновацій та поліпшення наших щоденних дій. Особливо це стосується таких повсякденних завдань, як приготування їжі. З наростанням глобалізації та розширенням кулінарних горизонтів, виникла потреба у розвитку комп'ютеризованих систем, що допомагають знаходити, організовувати та навіть адаптувати рецепти з усього світу.

Системи пошуку рецептів не лише спрощують процес вибору страв на кожен день, але й відіграють важливу роль у збереженні та популяризації національних кулінарних традицій, адаптуючи їх до сучасних технологічних реалій. У цьому контексті, особлива увага приділяється створенню інтуїтивно зрозумілих, зручних і ефективних систем, які б відповідали вимогам широкого кола користувачів.

Дана дипломна робота зосереджена на створенні комп'ютерної системи для пошуку рецептів, метою якої є поліпшення процесу пошуку та надання користувачам доступу до широкого асортименту кулінарних варіантів. Проект орієнтований на розробку інтуїтивно зрозумілого інтерфейсу, що сприяє легкій навігації по вебсайту та інтеграції з надійними джерелами даних, забезпечуючи актуальність та повноту інформації, доступної користувачам.

Проект включає аналіз існуючих рішень у цій області, визначення основних вимог до системи, розробку архітектури та інтерфейсу, а також випробування розробленого прототипу. Робота спрямована на створення ефективного інструменту, який би враховував сучасні тенденції у веб-розробці та інтерактивному дизайні, надаючи користувачам комфортний та продуктивний досвід використання.

1 ВИЗНАЧЕННЯ ТЕХНОЛОГІЙ РОЗРОБКИ

1.1 Вибір архітектури системи

Архітектура програмного забезпечення визначає структуру рішень, яка задовольняє технічні та операційні вимоги проекту. Цей методологічний підхід сприяє оптимізації ключових аспектів системи, включаючи безпеку, продуктивність та управління, що значно впливає на якість обслуговування, ефективність та загальний успіх програмного продукту. Застосування цих стратегічних рішень допомагає створити стабільну та надійну систему, готову до масштабування та адаптації до мінливих вимог ринку [1].

Правильний вибір архітектури відіграє критичну роль у розробці програмних систем, оскільки визначає, як компоненти взаємодіятимуть, і як система в цілому відповідатиме технічним та операційним вимогам. Вибір відповідної архітектури може забезпечити систему, яка є легкою у масштабуванні, підтримці та оновленні, а також може ефективно вирішувати комплексні бізнес-виклики, стоячи перед компаніями сьогодення [2].

Архітектури веб-додатків можуть бути різноманітними, кожна з яких володіє своїми перевагами та недоліками в залежності від вимог проекту. Ось декілька поширених типів архітектур програмних продуктів:

- монолітна;
- мікросервісна;
- клієнт-серверна;
- serverless архітектура.

1.1.1 Монолітна архітектура

Монолітна архітектура програмного забезпечення характеризується її здатністю об'єднувати всі компоненти програми (клієнтський інтерфейс, бізнес-логіку, обробку даних та серверну логіку) у єдину систему, яка розгортається як один цілісний блок. Така структура дозволяє компонентам ефективно взаємодіяти в межах одного процесу, що може забезпечувати високу продуктивність та

спрощення управління програмою.

Монолітна архітектура програмного забезпечення володіє кількома ключовими перевагами, які роблять її привабливою на ранніх етапах розробки додатків. Зокрема, вона забезпечує простоту розробки та розгортання, оскільки всі компоненти додатку знаходяться в одному кодовому репозиторії, що полегшує випробування та виявлення помилок. Це також може бути більш економічно ефективним на початкових стадіях, оскільки зменшує потребу в значних інвестиціях у інфраструктуру та ресурси [3].

Виходячи із попередньо зазначеного, у монолітній архітектурі зазвичай використовується одна мова програмування. Основна причина цього полягає у тому, що всі компоненти комп'ютерної системи – від бізнес-логіки до серверної частини тісно інтегровані і працюють у рамках одного процесу.

Однак, зі зростанням та розвитком додатку, монолітна архітектура може стати обтяжливою. Оскільки всі компоненти тісно інтегровані, масштабування специфічних функцій може вимагати масштабування всієї системи, що часто не є оптимальним з точки зору ресурсів та ефективності. Інтеграція нових технологій може потребувати значних зусиль та витрат, адже це може вимагати переписування значної частини коду. Крім того, будь-яка помилка в одному компоненті може призвести до збоїв у всій системі, підвищуючи ризик простою та зменшуючи загальну надійність додатку.

Монолітна архітектура буде оптимальним рішенням сценаріях, коли важливі швидкість розробки та простота управління проектом. Цей підхід найкраще підходить для проектів, де вся функціональність додатку може бути розроблена, тестована та впроваджена спільно без необхідності великої масштабності або складної міжсервісної взаємодії.

1.1.2 Мікросервісна архітектура

Мікросервісна архітектура, яка з'явилася як відповідь на обмеження монолітних архітектур, пропонує ряд переваг, які роблять її ідеальною для сучасних динамічних бізнес-середовищ, що вимагають швидкої адаптації та інновацій. Однією з основних переваг є підвищена гнучкість та масштабованість,

завдяки незалежності мікросервісів, які можуть бути розгорнуті та масштабовані окремо один від одного. Це дозволяє більш ефективно використовувати ресурси та швидше реагувати на зміни у навантаженні [4].

Крім того, мікросервіси сприяють швидкішому виведенню продуктів на ринок, оскільки розробка, випробування та впровадження кожного сервісу відбуваються незалежно. Це зменшує час на деплоймент та ризики, пов'язані з оновленнями, оскільки можливі помилки в одному сервісі не вплинуть на роботу всієї системи.

Однак, мікросервісна архітектура не позбавлена недоліків. Вона може вносити додаткову складність у розробку та управління, оскільки кожен мікросервіс може використовувати свій власний стек технологій та мати власну базу даних. Це ускладнює управління даними та вимагає високого рівня взаємодії між сервісами. Також збільшується навантаження на мережу, оскільки сервіси постійно комунікують один з одним через мережу.

Оперування в середовищі мікросервісів вимагає сильних навичок у сфері DevOps, оскільки належне розгортання, моніторинг та масштабування мікросервісів є ключовими для забезпечення стабільності та високої доступності сервісів [4].

Таким чином, вибір між монолітною та мікросервісною архітектурою повинен базуватися на конкретних потребах проекту, його масштабах, складності, а також ресурсах, які можуть бути виділені на підтримку відповідної інфраструктури.

1.1.3 Архітектура клієнт-сервер

Клієнт-серверна архітектура є досить поширеною моделлю в інформаційних системах, що дозволяє ефективно розділити функції між сервером, який обробляє дані та запити, та клієнтами, які надсилають ці запити та отримують відповіді. Такий підхід забезпечує централізоване управління даними та високий рівень безпеки, оскільки вся важлива інформація та обробка запитів зосереджені в одному місці [5].

Однією з головних переваг клієнт-серверної архітектури є її

масштабованість. Сервери можуть обслуговувати велику кількість клієнтів, а систему можна розширювати додаванням нових серверів або оновленням існуючих без зміни клієнтських додатків. Це також забезпечує відносно просте управління оскільки оновлення та зміни конфігурації виконуються централізовано на сервері.

Однак, не обходиться без недоліків. Залежність від централізованого сервера може стати проблемою: якщо сервер вийде з ладу, це може призупинити роботу всієї мережі. Також, велика кількість запитів від клієнтів може створювати вузькі місця у продуктивності та збільшувати час відгуку. Існують також питання безпеки, пов'язані з трансляцією даних через мережу, що вимагає застосування додаткових заходів захисту для забезпечення конфіденційності та цілісності інформації.

Враховуючи ці аспекти, клієнт-серверна архітектура залишається важливою та ефективною для багатьох застосувань, де потрібне надійне централізоване управління ресурсами та даними, особливо в великих організаціях та системах з великою кількістю користувачів.

1.1.4 Serverless архітектура

Serverless архітектура дозволяє розробникам зосередитися на кодуванні та інноваціях, оскільки вся управлінська робота по обслуговуванню серверів виконується провайдером. Це сприяє швидшому впровадженню змін і знижує витрати, оскільки плата здійснюється тільки за фактично використані ресурси [6].

Завдяки автоматичному масштабуванню, застосунки можуть легко адаптуватися до зростання або падіння навантаження, що особливо важливо при нестабільному трафіку [6].

Проте, цей підхід може ускладнити випробування та відлагодження, оскільки розробники не мають доступу до низькорівневих серверних процесів. Безпека також може бути під загрозою через залежність від зовнішніх провайдерів, які управляють усією інфраструктурою.

Важливо зазначити, що цей підхід не підходить для додатків, які вимагають тривалих процесів обробки. "Холодні старты" функцій можуть призвести до затримок при обробці запитів, особливо якщо додаток не використовується

постійно.

Ці особливості роблять цю технологію привабливою для певних типів додатків та сценаріїв використання, зокрема для обробки подій в реальному часі та мікросервісів, але вимагають ретельного аналізу потреб проекту перед її впровадженням.

1.2 Вибір технологій для фронтенду та бекенду

На сьогоднішній день у світі веб-розробки існує безліч технологій, які дозволяють створювати як окремі фронтенд або бекенд компоненти, так і комплексні рішення, що об'єднують обидві частини.

Фронтенд технології фокусуються на розробці інтерфейсу, який користувачі бачать і з яким взаємодіють, використовуючи HTML, CSS та JavaScript.

Бекенд технології забезпечують серверну логіку і обробку даних, використовуючи найрізноманітніші мови програмування такі як Python, Php чи Node.js.

Слід зазначити, що на сьогоднішній день у сфері програмування майже не використовуються прості мови програмування в їх первісному вигляді для створення великих програмних продуктів. Замість цього, перевагу надають фреймворкам та бібліотекам, які значно спрощують та оптимізують процес розробки.

Однак, існують також платформи та фреймворки, що дозволяють розробникам ефективно працювати над фронтендом та бекендом одночасно, що може значно прискорити розробку та впровадження веб-додатків.

До фреймворків та бібліотек для розробки клієнтської частини відносять: React, Angular, Vue.js. Для написання серверної частини: Django, Laravel, Express.js. Також існують рішення для роботи над фронтендом та бекендом одночасно: Next.js та Nuxt.js.

1.2.1 React

React – це бібліотека JavaScript, розроблена Facebook для створення інтерактивних користувацьких інтерфейсів. Дана бібліотека дозволяє будувати динамічні веб-додатки, які можуть оновлювати інформацію на сторінці без потреби її перезавантаження.

React працює, створюючи віртуальний DOM у пам'яті, а не безпосередньо маніпулюючи DOM браузера. Він виконує необхідні маніпуляції всередині цього віртуального представлення, перш ніж застосувати зміни до реального DOM браузера. Бібліотека має підвищену продуктивність, змінюючи лише те, що потребує модифікації [7].

React дозволяє побудову інтерфейсів на основі компонентів, кожен з яких є самодостатнім і має власний стан та параметри, що сприяє модульності і повторному використанню коду. Використання JSX полегшує написання HTML-подібного коду прямо у JavaScript, роблячи компоненти більш читабельними.

У традиційних JavaScript-додатках зміни даних вимагають ручного оновлення DOM (Document object model), що може призвести до перезавантаження всієї сторінки. Відмінно від цього, React реалізує підхід, який дозволяє створювати односторінкові додатки (SPA), де перший запит завантажує один HTML-документ, а подальші оновлення контенту відбуваються через JavaScript без перезавантаження сторінки [8].

React впроваджує клієнтську маршрутизацію, де запити на нові сторінки обробляються без перезавантаження, оновлюючи лише потрібні частини сторінки. Це підвищує продуктивність та поліпшує користувацький досвід, забезпечуючи динамічну взаємодію.

Основою React є віртуальний DOM, копія реального DOM, яка дозволяє бібліотеці швидко реагувати на зміни. Віртуальний DOM оновлюється при кожній зміні даних, після чого React аналізує різницю з реальним DOM та оновлює лише змінені елементи, мінімізуючи витрати на рендеринг та сприяючи високій продуктивності інтерфейсу. Цей процес забезпечує швидке відображення змін, що вирізняє React серед інших технологій веб-розробки.

Важливою особливістю React є однонаправлений потік даних. Змінні стану та параметри компонентів рухаються лише в одному напрямку: від батьківського до дочірнього компоненту. Параметри (props) в дочірньому компоненті не можуть повертати дані до батьківського компонента, але він може мати зв'язок з батьківськими компонентами, щоб змінювати стан відповідно до наданих вхідних даних.

Дана бібліотека ідеально підходить для розробки односторінкових додатків (SPA), забезпечуючи плавні оновлення контенту без перезавантаження сторінки.

1.2.2 Angular

Angular – це потужний фреймворк та платформа для розробки, створена Google, базується на TypeScript. Він використовується для створення односторінкових веб-додатків і включає компонентну архітектуру, що дозволяє будувати масштабовані веб-додатки. Angular має вбудовані бібліотеки для маршрутизації, управління формами, клієнт-серверного спілкування та багато іншого, а також набір інструментів для розробки, випробування та оновлення коду [9].

Даний фреймворк надає можливість розбивати додатки на перевикористовувані компоненти завдяки компонентній архітектурі. Two-Way Data Binding в Angular автоматично синхронізує дані між моделлю та виглядом, що зменшує необхідність вручну керувати DOM.

На відміну від React, Angular використовує лише TypeScript. Це забезпечує підвищену безпеку типів, що сприяє написанню більш чистого і стабільного коду. Модульність і система впровадження залежностей Angular дозволяють ефективно організувати код та легше впроваджувати зміни.

Порівнюючи Angular з іншими популярними фреймворками для веб-розробки, важливо зазначити, що кожен з них має свої унікальні особливості та сильні сторони. На відміну від VueJS та ReactJS, які в першу чергу є бібліотеками, Angular - це повноцінний фреймворк для створення веб-додатків. Це означає, що він надає більш повний набір інструментів для створення складних додатків прямо з коробки [10].

У порівнянні з Vue, який може похвалитися простішим синтаксисом і швидшою кривою навчання, Angular пропонує розширений набір інструментів і більшу екосистему, що може бути корисним в управлінні великомасштабними проектами. Однак це відбувається за рахунок складності; початківцям може бути складніше освоїти Angular, ніж Vue.

ReactJS працює як інтерфейсна бібліотека JavaScript з відкритим вихідним кодом. У той час як React популярний завдяки функції віртуального DOM, яка підвищує продуктивність додатків, Angular використовує звичайний DOM разом з TypeScript для кращої читабельності коду та зручності супроводу.

Angular ідеально підходить для створення масштабованих веб-додатків, особливо в корпоративному середовищі, де необхідна висока гнучкість та здатність швидко адаптуватися до змінних вимог. Його робота на основі компонентів робить його відмінним вибором для складних інтерфейсів.

1.2.3 Vue.js

Vue.js – це прогресивний JavaScript фреймворк, що зосереджений на створенні користувацьких інтерфейсів. Розроблений Еваном Ю, Vue вирізняється своєю інкрементальною приймальністю, що дозволяє розробникам легко інтегрувати його в існуючі проекти або використовувати для складних додатків з нуля. Vue має гнучку архітектуру, що спрощує створення інтерактивних інтерфейсів за допомогою простого і зрозумілого синтаксису [11].

Vue.js надає важливі можливості, які роблять його зручним для широкого спектру веб-додатків:

- компонентна архітектура, яка дозволяє розробникам будувати великі застосунки, розділені на самодостатні та перевикористовувані компоненти;
- двонаправлений зв'язок даних, що полегшує синхронізацію між моделлю та виглядом;
- віртуальний DOM, який оптимізує оновлення інтерфейсу та забезпечує швидку відповідь на взаємодії користувача.

Vue широко використовується для створення односторінкових застосунків (SPA) та інтерфейсів реального часу, де важлива швидкість реагування та зручність

користувача. Завдяки своїй легкості та інтегрованості, Vue також є популярним вибором для стартапів і середніх проектів, які потребують швидкої розробки та високої продуктивності.

1.2.4 Django

Django – це високорівневий веб-фреймворк на Python, який сприяє швидкому розвитку та чистому, прагматичному дизайну. Він був розроблений для того, щоб допомогти розробникам переводити концепції в готові застосунки якомога швидше і при цьому вимагає меншої кількості коду. Django слідує принципу "Batteries included", надаючи багато загальних функціональностей "з коробки", таких як адміністрування контенту, аутентифікація, карта сайту та багато іншого, що дозволяє розробникам зосередитись на написанні додатку, а не на переосмисленні колеса [12].

Django включає добре організовану архітектуру на основі моделі-шаблону-виду (MVT), що сприяє чіткому розділенню логіки обробки даних, інтерфейсу користувача та логіки взаємодії, роблячи код більш чистим і легшим для підтримки.

Завдяки своїй гнучкості, Django широко використовується для створення складних веб-сайтів, що вимагають чіткої структуризації та масштабування, зокрема в новинних порталах, соціальних мережах та інших великих проектах.

1.2.5 Laravel

Laravel – це міжплатформовий PHP-фреймворк для створення веб-додатків, що має багато переваг у використанні, оскільки дозволяє розробнику скористатися можливостями великої бібліотеки попередньо запрограмованих функцій (таких як автентифікація, маршрутизація та створення шаблонів HTML). Доступ до цієї бібліотеки спрощує швидке створення надійних веб-додатків, зводячи до мінімуму кількість необхідного коду [13].

Laravel пропонує високофункціональне середовище розробки, а також інтуїтивно зрозумілі та виразні інтерфейси командного рядка. Крім того, Laravel використовує об'єктно-реляційне відображення (ORM) для спрощення доступу до даних і маніпулювання ними.

Програми Laravel дуже масштабовані та мають прості в обслуговуванні кодові бази. Розробники також можуть безперешкодно додавати функціональні можливості до своїх програм завдяки модульній системі упаковки Laravel і надійному управлінню залежностями.

1.2.6 Next.js

Next.js це гнучкий фреймворк React, який надає будівельні блоки для створення швидких повностекових веб-додатків. Він визначає інструменти та конфігурацію, необхідні для React, а також надає додаткову структуру, функції та оптимізацію для додатку.

Однією з ключових можливостей фреймворку є підтримка статичної генерації сторінок (Static Site Generation - SSG) та серверного рендерингу (Server-Side Rendering - SSR). Перша дозволяє попередньо генерувати сторінки під час збирання додатку, забезпечуючи високу швидкість завантаження та оптимізацію для пошукових систем. Серверний рендеринг же покращує відгук на динамічний контент, оптимізуючи обробку SEO для додатків, що змінюються у реальному часі [14].

Також даний фреймворк має менш значні інструменти та рішення:

- визначена API маршрутизація та роутинг;
- автоматичне розділення коду по сторінках додатката компонентах;
- вбудована підтримка CSS-IN-JS фреймворку та інших рішень для імплементації CSS стилів сторінки;
- оптимізація зображень за допомогою компоненту Image;
- вбудована підтримка інтернаціоналізованої (i18n) маршрутизації.

Next.js ідеально підходить для розробки великомасштабних проектів та веб-додатків, які потребують швидкого рендерингу та оптимальної продуктивності. Фреймворк часто використовується для створення стартапів, корпоративних сайтів і електронної комерції, де важлива швидкість завантаження та SEO-оптимізація.

1.2.7 Nuxt.js

Nuxt.js – це фреймворк, побудований на Vue.js, який спрощує розробку сучасних веб-додатків. Він підтримує серверне рендерування (SSR) та генерацію

статичних сайтів (SSG), аналогічно до Next.js, що покращує SEO та продуктивність [15].

Однією з особливостей Nuxt є функція автоматичної маршрутизації. Це усуває необхідність ручного налаштування, оскільки Nuxt автоматично генерує маршрути на основі вашої файлової структури. Це означає, що кожен файл .vue у вашому каталозі сторінок стає маршрутом у вашому додатку.

Основні можливості Nuxt включають розбиття коду, деревоподібну структуру та оптимізовані стратегії холодного старту, які значно підвищують продуктивність додатків. Він також постачається з попередньо налаштованими Webpack та Babel, що спрощує використання однофайлових компонентів та сучасних версій JavaScript.

Даний фреймворк також використовується для створення швидких та масштабованих застосунків, зокрема для односторінкових додатків, які потребують кращого SEO та швидшого завантаження сторінок. Nuxt.js має потужну екосистему та бібліотеки, що робить його вибором для багатьох розробників, особливо тих, хто вже використовує Vue.js.

1.3 Вибір бази даних

Вибір відповідної бази даних є критичним аспектом при розробці будь-якої інформаційної системи, оскільки тип бази даних значно впливає на функціональність, продуктивність, та легкість супроводу системи.

Обираючи базу даних важливо розуміти поняття бази даних та систему управління базами даних (СУБД).

База даних (БД) – це організована структура, яка призначена для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів. Бази даних призначені для зберігання великих обсягів інформації та її організації таким чином, щоб можна було ефективно виконувати пошук, вибірку, оновлення та

управління даними [18].

СУБД – це саме програмне забезпечення, яке забезпечує взаємодію з користувачами, іншими програмами, а також з самою базою даних. Це дозволяє керувати створенням, збереженням та обробкою даних.

Бази даних зазвичай розділяють на реляційні та нереляційні (NoSQL) бази даних.

Реляційна модель використовує набір таблиць для представлення як даних, так і зв'язків між ними. Кожна таблиця має кілька стовпців, і кожен стовець має унікальне ім'я. Таблиці також відомі як відношення. Реляційна модель є прикладом моделі на основі записів. Моделі на основі записів називаються так тому, що база даних структурована у вигляді записів фіксованого формату декількох типів. Кожна таблиця містить записи певного типу. Кожен тип запису визначає фіксовану кількість полів або атрибутів. Стовпці таблиці відповідають атрибутам типу запису. Реляційна модель даних є найпоширенішою моделлю даних, і переважна більшість сучасних систем баз даних заснована на реляційній моделі [16].

Відмінність реляційних баз даних від інших типів, таких як NoSQL або об'єктно-орієнтовані бази даних, полягає у їх строгій структурованості та суворому дотриманні схем даних, що забезпечує точність та послідовність даних через нормалізацію. Реляційні бази даних підтримують складні запити (SQL запити), транзакції та операції злиття, що робить їх особливо корисними для додатків, де важлива точність та надійність обробки даних.

Реляційні бази даних варто використовувати там, де потрібен чіткий порядок та точність у зберіганні інформації. Такі бази ідеально підходять для фінансових операцій, обліку, управління активами компаній та інших галузей, де важливо точно вести облік великої кількості даних та їх взаємозв'язків. Це також відмінний варіант для систем, яким потрібно виконувати складні запити до бази даних та створювати детальні звіти.

NoSQL – це тип системи управління базами даних (СУБД), яка призначена для обробки та зберігання великих обсягів неструктурованих і напівструктурованих даних. На відміну від традиційних реляційних баз даних, які

використовують таблиці із заздалегідь визначеними схемами для зберігання даних, бази даних NoSQL використовують гнучкі моделі даних, які можуть адаптуватися до змін у структурі даних і здатні до горизонтального масштабування для обробки зростаючих обсягів даних [17].

Спочатку термін NoSQL стосувався нереляційних баз даних, але з тих пір він став означати "не тільки SQL", оскільки бази даних NoSQL розширилися, включивши в себе широкий спектр різних архітектур баз даних і моделей даних.

Бази даних NoSQL часто використовуються в додатках, де є великий обсяг даних, які потрібно обробляти та аналізувати в режимі реального часу, наприклад, в аналітиці соціальних мереж, електронній комерції та іграх. Вони також можуть бути використані для інших додатків, таких як системи управління контентом, документообігу та управління взаємовідносинами з клієнтами.

Однак, бази даних NoSQL можуть не підходити для всіх додатків, оскільки вони можуть не забезпечувати такий же рівень узгодженості даних і гарантії транзакцій, як традиційні реляційні бази даних. При виборі системи управління базами даних важливо ретельно оцінити конкретні потреби програми.

Переходячи до вибору програмного забезпечення для керування базою даних, слід зазначити, що на сьогодні найбільш поширеними є наступні СУБД [19]:

- Oracle SQL Developer;
- MySQL Workbench;
- Microsoft SQL Server;
- PostgreSQL.

1.3.1 Oracle SQL Developer

Oracle SQL Developer є безкоштовним інтегрованим середовищем розробки, яке спрощує розробку та управління базами даних Oracle, як у традиційних, так і в хмарних розгортаннях. Це графічний інструмент який дозволяє переглядати об'єкти баз даних, виконувати SQL-запити та скрипти, редагувати та налагоджувати код PL/SQL, маніпулювати та експортувати дані, а також переглядати та створювати звіти [20].

Основні відмінності та переваги Oracle SQL Developer включають:

- повна підтримка PL/SQL;
- адміністрування баз даних;
- моделювання даних;
- міграція з сторонніх СУБД на Oracle;
- міграція з Oracle On-Premises на Oracle Cloud.

Написана мовою Java Oracle SQL Developer вирізняється своєю кроссплатформною, оскільки це програмне забезпечення може бути встановлене та використане на різних операційних системах, включаючи Windows, Linux та macOS.

1.3.2 MySQL Workbench

MySQL Workbench – це уніфікований візуальний інструмент для архітекторів, розробників та адміністраторів баз даних. Програмне забезпечення забезпечує моделювання даних, розробку SQL, а також комплексні інструменти адміністрування для конфігурації сервера, адміністрування користувачів, резервного копіювання та багато іншого. MySQL Workbench доступний для Windows, Linux та Mac OS X [21].

Програма надає візуальні інструменти для створення, виконання та оптимізації SQL запитів. Редактор SQL впроваджує кольорове підсвічування синтаксису, автозавершення, повторне використання фрагментів SQL та історію виконання SQL. Панель з'єднань з базами даних дозволяє розробникам легко керувати стандартними з'єднаннями з базами даних, включаючи MySQL Fabric. Браузер об'єктів надає миттєвий доступ до схеми та об'єктів бази даних.

Для адміністрування середовищ MySQL, візуальна консоль MySQL Workbench спрощує управління та забезпечує кращу видимість баз даних. Завдяки візуальним інструментам, адміністратори можуть налаштовувати сервери, виконувати резервне копіювання та відновлення, перевіряти дані аудиту, а також переглядати стан бази даних.

Крім того, MySQL Workbench пропонує розширені інструменти для підвищення продуктивності MySQL-додатків. DBA мають можливість швидко аналізувати ключові показники продуктивності через панель Performance

Dashboard. Звіти про продуктивність допомагають ідентифікувати критичні точки в системі, а Visual Explain Plan дозволяє розробникам ефективно оптимізувати SQL-запити.

1.3.3 Microsoft SQL Server

Microsoft SQL Server – це система управління реляційними базами даних (СУБД), яка підтримує широкий спектр додатків для обробки транзакцій, бізнес-аналітики та аналізу даних у корпоративних IT-середовищах. Як і інші СУБД, Microsoft SQL Server використовує стандартизовану мову SQL для керування базами даних і обробки запитів, а також власну мову запитів Microsoft, Transact-SQL (T-SQL), для взаємодії з сервером [22].

SQL Server забезпечує реляційну модель зберігання даних, яка гарантує цілісність посилань та інші обмеження для збереження точності даних, що допомагає забезпечити надійну обробку транзакцій. Основний компонент SQL Server, механізм баз даних, контролює зберігання, доступ, обробку та безпеку даних. Він включає реляційний механізм для обробки команд і запитів та механізм зберігання для управління файлами бази даних і транзакціями.

Клієнти підключаються до механізму бази даних через клієнтські бібліотеки, що можуть використовувати графічний інтерфейс або інтерфейс командного рядка, забезпечуючи гнучкість у виборі інструментів для роботи з базою даних. Нижчий рівень, SQL Server Operating System, виконує функції управління пам'яттю, вводом/виводом та блокуванням даних. Завдяки цьому SQL Server забезпечує високу продуктивність і стабільність для корпоративних додатків.

Розширені функції безпеки включають три технології Always Encrypted, яка дозволяє користувачеві оновлювати зашифровані дані без необхідності розшифровувати їх; захист на рівні рядків, що дозволяє контролювати доступ до даних на рівні рядків у таблицях баз даних; і динамічне маскуванню даних, яке автоматично приховує елементи конфіденційних даних від користувачів без повних привілеїв доступу.

1.3.4 PostgreSQL

PostgreSQL – це відкрита система управління базами даних (СУБД) з

відкритим вихідним кодом. Базована на мові SQL, PostgreSQL належить до об'єктно-реляційного типу СУБД, тобто поєднує в собі переваги реляційних баз даних і об'єктно-орієнтованого програмування [23].

PostgreSQL підтримує широкий спектр функцій, які роблять її потужною і гнучкою СУБД:

- PostgreSQL забезпечує транзакції з ACID-властивостями, що гарантує цілісність даних;
- підтримує автоматично оновлювані подання, що забезпечує можливість користувачам переглядати дані в базі даних у режимі реального часу;
- підтримує матеріалізовані подання, які зберігають копію даних подання в базі даних;
- підтримує тригери, які дають змогу виконувати дії під час зміни даних у таблиці;
- підтримує зовнішні ключі для створення зв'язку між таблицями;
- підтримує збережені процедури для виконання складних обчислень на стороні сервера.

База PostgreSQL успішно застосовується в різних галузях. У веб-розробці вона служить основою для динамічних сайтів і додатків. В аналізі даних, вона забезпечує високу продуктивність при обробці величезних обсягів інформації. Великомасштабні додатки, такі як соціальні мережі, спираються на масштабованість і надійність цієї системи.

1.4 Висновки з аналізу технологій

При виборі архітектури для розробки вебсайту пошуку рецептів визначено основні вимоги: централізоване управління даними, масштабованість та забезпечення ефективності реалізації. Даним вимогам відповідає клієнт-серверна архітектура. Вона забезпечить більш простий процес розробки, чітко розділивши

клієнтську та серверну логіку, а також надасть можливість легко масштабувати функціональність сайту.

Через наявність досвіду використання бібліотеки React, серед технологій розробки фронтенду перевага надається саме тим, що базуються на даній JavaScript бібліотеці. Для більш комфортного процесу розробки вибір зупинився на використанні готових фреймворків які містять в собі готові інструменти для вирішення переважної більшості проблем. Це дозволить зменшити час на написання вебсайту та акцентувати увагу на унікальних аспектах та функціональності проекту, замість того, щоб вирішувати загальні технічні проблеми. Таким чином обрано фреймворк Next.js.

У процесі аналізу систем керування базами даних виявлено, що вони всі пропонують приблизно однакову функціональність та високий рівень якості. Таким чином, за критерієм зручності у використанні, обрано Oracle SQL Developer.

2 МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Загальні вимоги до системи

Основною метою розробки вебсайту з пошуку рецептів є створення платформи, яка забезпечує зручний та швидкий доступ до великої кількості рецептів. Для досягнення цієї мети система повинна відповідати наступним загальним вимогам:

- зручність використання: інтерфейс користувача повинен бути інтуїтивно зрозумілим і простим у навігації;
- адаптивність: всі елементи інтерфейсу мають коректно відображатись на усіх поширених на сьогодні пристроях;
- продуктивність: система повинна швидко відповідати на запити користувачів, включаючи пошук рецептів та завантаження сторінок;

- безпека: захист персональних даних користувачів є пріоритетом; система повинна використовувати сучасні методи шифрування та автентифікації для захисту інформації;
- масштабованість: система повинна бути здатною обробляти збільшення кількості користувачів та обсягу даних без втрати продуктивності;
- надійність: система повинна мати високий рівень доступності та мінімізувати можливість збоїв.

Розроблюваний вебсайт буде називатися MenuMastermind. Це ім'я повинно відображатись на всіх сторінках сайту, а також бути легко впізнаваним для користувачів.

2.2 Вимоги до користувача

Потенційний користувач розроблюваної комп'ютерної системи має відповідати мінімальним вимогам: користувачі повинні мати сучасний веб-браузер (Chrome, Firefox, Safari, Edge), стабільне підключення до інтернету та пристрій з екраном. Рекомендовані та мінімальні вимоги до користувача комп'ютерної системи наведено в таблицях 2.1 та 2.2.

Таблиця 2.1 – Мінімальні вимоги до користувача

Компонент	Мінімальні вимоги
Процесор	Intel Core i3 (2010), AMD Athlon II (2009)
Оперативна пам'ять	4 ГБ
Операційна система	Windows 7, macOS 10.12, Linux
Веб-браузер	Chrome, Firefox, Safari, Edge
Підключення до Інтернет	Стабільне підключення, швидкість 2 Мбіт/с

Крім основних вимог, користувачам також рекомендується мати оновлене програмне забезпечення та браузери для забезпечення сумісності та безпеки під час

використання вебсайту. Оновлені версії браузерів забезпечують кращу продуктивність, безпеку та підтримку сучасних вебтехнологій, що є важливим для коректної роботи вебсайту.

Таблиця 2.2 – Рекомендовані вимоги до користувача

Компонент	Рекомендовані вимоги
Процесор	Intel Core i5 (2015), AMD Ryzen 3 (2017)
Оперативна пам'ять	8 ГБ або більше
Операційна система	Windows 10, macOS 10.15, Linux
Веб-браузер	Chrome, Firefox, Safari, Edge
Підключення до Інтернет	Стабільне підключення, швидкість 10 Мбіт/с

Для користувачів мобільних пристроїв рекомендовано використовувати сучасні смартфони та планшети з останніми версіями операційних систем iOS або Android. Це забезпечить плавну роботу та доступ до всіх функцій вебсайту.

2.3 Функціональні вимоги системи

Для забезпечення виконання загальних вимог, система повинна мати такі функціональні можливості:

- пошук рецептів: користувачі повинні мати можливість шукати рецепти за різними критеріями (назва, інгредієнти, категорії тощо);
- детальний перегляд рецептів: кожен рецепт повинен мати окрему сторінку з описом, списком інгредієнтів, покроковими інструкціями, а також фото;
- реєстрація та автентифікація: користувачі повинні мати можливість реєструватися та входити до системи для доступу до додаткових функцій, таких як збереження улюблених рецептів;

- додавання рецептів до списку “улюблених”: авторизовані користувачі повинні мати можливість додавати рецепти до списку “улюблених” для швидкого доступу до них у майбутньому;

- коментування та оцінювання: користувачі повинні мати можливість залишати відгуки та оцінки до рецептів.

2.4 Діаграма використання

Діаграма використання (Use Case Diagram) демонструє, як користувач буде взаємодіяти з веб-застосунком та відображає базову функціональність системи. У таблиці 2.3 наведено перелік елементів, використаних при побудові діаграми [24].

Таблиця 2.3 – Елементи діаграми варіантів використання

Назва елемента	Візуальне відображення та призначення
Актор	Показані на діаграмі у вигляді паличок з іменами під ними, представляють елементи, які будуть безпосередньо взаємодіяти з системою.
Сценарій користування	Овальні фігури, з їх назвою в центрі, представляють безпосередню функціональність в системі.
Включення	Пунктирні лінії, позначені <<include>>. Означають, що варіант використання без стрілки викликає функціональність варіанту використання зі стрілкою.
Розширення	Пунктирні лінії, позначені <<extend>>. Означають, що варіант використання без стрілки бере всю функціональність варіанту використання зі стрілкою і додає додаткову функціональність.
Границя системи	Великий прямокутник, вміщує все, за реалізацію чого відповідає система

При побудові діаграми було визначено основні дії користувача:

- реєстрація;

- автентифікація;
- пошук рецептів;
- детальний перегляд рецепту;
- додавання рецепту до списку “улюблених”;
- написання відгуку до рецепту.

Побудована діаграма зображена на рисунку 2.1.

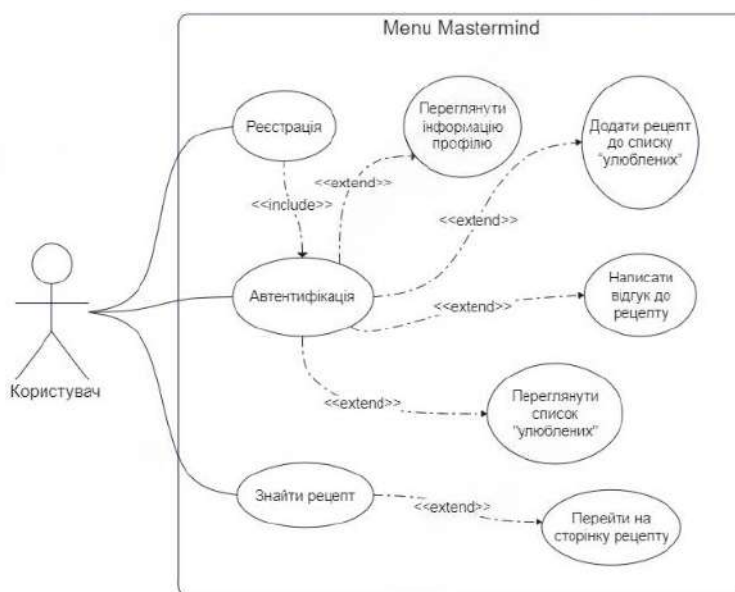


Рисунок 2.1 – Діаграма “Use Case” розроблюваної системи

У якості актору діаграми зображено користувача, якому доступні всі функціональні можливості системи. Так як розроблювана система передбачає множинні варіанти розвитку подій, то задля уникнення перевантаження діаграми на ній відображена тільки базова функціональність.

2.5 Діаграма послідовностей

Діаграма послідовностей (Sequence Diagram) показує порядок взаємодії між об'єктами системи в процесі виконання певних функцій.

До створеної вище діаграми використань створено діаграму послідовності у випадку автентифікації користувача на сайті. Дану діаграму зображено на рисунку 2.2 [25].

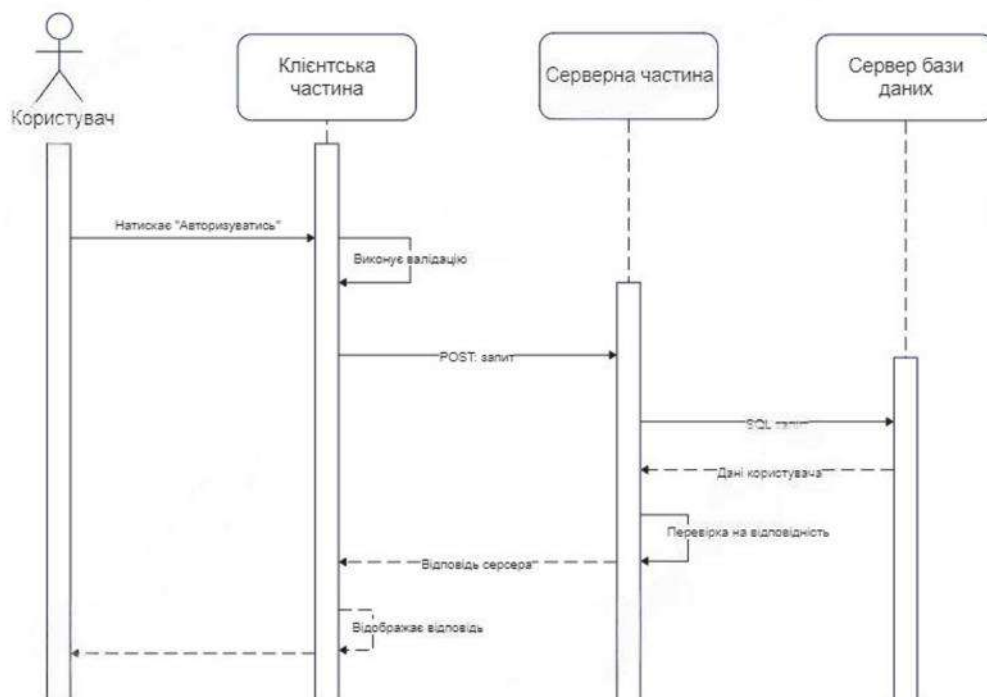


Рисунок 2.2 – Діаграма послідовності

Процес автентифікації включає в себе наступні етапи:

- натискання на кнопку "Авторизуватись" користувачем;
- валідація введених даних з боку клієнтської частини застосунку;
- виконання POST запиту на сервер задля перевірки введених даних;
- надсилання SQL запиту до бази даних;
- отримання даних користувача та перевірка еквівалентності введених та отриманих даних;
- надсилання до клієнтської частини застосунку результату про успішне виконання POST запиту;
- збереження даних користувацької сесії у форматі "cookie";
- перенаправлення користувача на головну сторінку.

2.6 Моделювання бази даних

Для пошуку та отримання інформації про рецепти буде використовуватись стороннє API. Тому у базі даних будуть зберігатися лише дані користувачів, їх улюблені рецепти та відгуки. Основними сутностями бази даних є користувачі, список улюбленого та відгуки.

ER-модель бази даних включає наступні сутності:

- користувачі (“Users”);
- список улюбленого (“Favorites”);
- відгуки (“Reviews”).

Структура кожної сутності зображена в таблицях 2.4-2.6.

Таблиця 2.4 – Склад таблиці “Users”

Атрибут	Тип даних	Допустимі значення	Значення за умовчанням	Обов'язковість	Примітки
Код користувача	Лічильник	Натуральне Число	-	Так	Первинний ключ
Ім'я користувача	Текстовий	Латиниця	-	Так	Унікальне значення
Пароль	Текстовий	Кирилиця, латиниця	-	Так	-
Пошта	Текстовий	Латиниця	-	Так	Унікальне значення
Аватар користувача	Текстовий	Латиниця	-	Ні	-

Таблиця 2.5 – Склад таблиці “Favorites”

Атрибут	Тип даних	Допустимі значення	Значення за умовчанням	Обов'язковість	Примітки
Код елементу	Лічильник	Натуральне число	-	Так	Первинний ключ
Код користувача	Лічильник	Натуральне число	-	Так	Зовнішній ключ

Продовження таблиці 2.5

Код рецепту	Текстовий	Латиниця	-	Так	-
-------------	-----------	----------	---	-----	---

Таблиця 2.6 – Склад таблиці “Reviews”

Атрибут	Тип даних	Допустимі значення	Значення за умовчанням	Обов'язковість	Примітки
Код відгуку	Лічильник	Натуральне число	-	Так	Первинний ключ
Код користувача	Лічильник	Натуральне число	-	Так	Зовнішній ключ
Дата	Дата та час	-	Поточна дата	Так	-
Текст	Текстовий	Кирилиця, латиниця	-	Так	-
Оцінка	Ціле число	Від 1 до 5	-	Так	-
Код рецепту	Текстовий	Латиниця	-	Так	-

Маючи визначені сутності побудовано реляційну модель бази даних, зображену на рисунку 2.3.

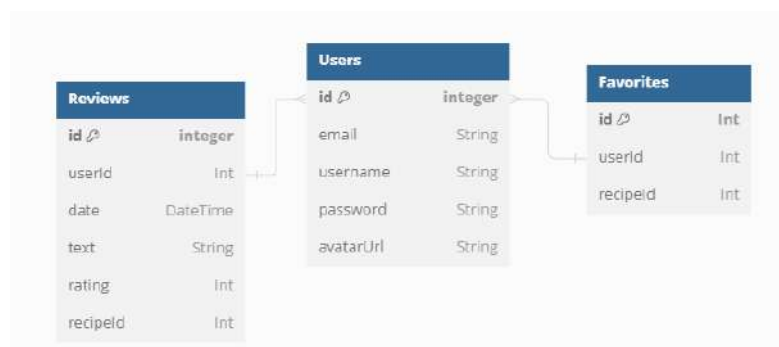


Рисунок 2.3 – Реляційна модель бази даних

Структура бази даних розроблена таким чином, щоб ефективно підтримувати функціональність користувачів щодо керування улюбленими рецептами та відгуками, забезпечуючи при цьому простоту інтеграції з стороннім API для пошуку рецептів.

3 РОЗРОБКА СИСТЕМИ

У цьому розділі розглядається процес розробки вебсайту з пошуку рецептів. Основна увага приділяється архітектурі середовища розробки, налаштуванню інструментів, імплементації бази даних, розробці користувацького інтерфейсу, а також динамічному формуванню контенту. В якості основного фреймворка для розробки було обрано Next.js, що забезпечує зручну інтеграцію з React та підтримку сучасних веб-технологій.

3.1 Архітектура середовища розробки

Задля ініціалізації проекту на Next.js виконано команду “`npx create-next-app@latest menu-mastermind`”. Дана команда автоматично створює новий проект з базовими налаштуваннями. Після цього можливо перейти до директорії створеного проекту.

Створений проект містить доволі просту файлову структуру. У корні проекту містяться файли файли конфігурації та папка “`src`” із власне файлами для розробки.

До файлів конфігурації належать:

- `.gitignore`: файл, що вказує на файли і директорії, які мають бути ігноровані Git;
- `next.config.js`: файл конфігурації Next.js;
- `package.json`: файл з інформацією про проект та його залежності;
- `README.md`: файл з описом проекту;
- `package-lock.json`: файл з інформацією про точні версії встановлених пакетів.

Директорія `src`:

- `src/app`: містить основні файли маршрутизації та компоненти сторінок, такі як `page.js` та `layout.js`;
- `src/components`: містить всі компоненти, які будуть використовуватись у додатку;
- `src/styles`: зберігає глобальні та модульні CSS файли для стилізації додатку;
- `src/pages`: використовується для застарілого підходу маршрутизації;
- `src/public`: містить статичні файли, такі як зображення;
- `src/utils`: містить допоміжні утиліти та функції.

Важливо розуміти, що Next.js підтримує два основні підходи до маршрутизації: App Router та Page Router.

Page Router використовується у директорії `src/pages`, де кожен файл представляє собою окрему сторінку. Цей підхід є традиційним для Next.js, але поступово замінюється на App Router, що забезпечує більш гнучку організацію коду.

App Router використовує файлову структуру для автоматичної маршрутизації. Кожен файл у директорії `src/app` представляє собою окремий маршрут. Наприклад, файл `page.js` є основною сторінкою для кожного маршруту, а файл `layout.js` визначає загальний макет для певного маршруту.

При розробці комп'ютерної системи для пошуку рецептів буде використовуватись саме App Router.

Архітектура проекту на Next.js спрямована на забезпечення модульності, зручності підтримки та розширюваності, наслідуючи ці ідеї з бібліотеки React. Основні особливості файлової архітектури включають компоненти, стилі, статичні файли та утиліти.

Компоненти зберігаються в папці `“src/components”`, що дозволяє легко повторно використовувати їх у різних частинах додатку та забезпечує кращу організацію коду.

Стилі організовані в папці `“src/styles”`, де глобальні стилі зберігаються у файлі `globals.css`, а інші стилі можуть бути організовані у модульні CSS або SCSS файли.

Всі статичні файли, такі як зображення, шрифти або іконки, зберігаються у директорії “src/public”, що дозволяє легко керувати ресурсами та забезпечує швидкий доступ до них.

Допоміжні функції та утиліти зберігаються у папці src/utils, що включає функції для обробки даних, роботи з API або інші допоміжні функції, які можуть бути використані в різних частинах додатку.

Також даний фреймворк дозволяє додавати власні директорії для організації більш зручного процесу розробки. Прикладами таких директорій є: lib, store, server-actions та інші.

3.2 Налаштування середовища розробки

Для забезпечення ефективної розробки вебсайту з пошуку рецептів, були виконані додаткові налаштування середовища розробки. Ці налаштування покращують організацію коду, зручність стилізації компонентів та інтеграцію з зовнішніми ресурсами.

3.2.1 Налаштування препроцесору Sass

Однією з перших змін було налаштування використання препроцесору Sass в синтаксисі Scss для більш зручного написання стилів компонентів. Використання Scss дозволяє писати стилі у більш структурований та зручний спосіб, що включає використання змінних, вкладення, міксинів та інших корисних функцій. Це значно покращує читабельність і підтримку стилів у великому проекті.

Для встановлення пакету підтримки Sass достатньо виконати команду “npm install --save-dev sass” [26].

Так як Next.js має вбудовану підтримку інтеграції з Sass після встановлення пакунка достатньо змінити розширення файлів стилів на .scss або .sass. Для використання Sass або Scss на рівні компонентів достатньо імпортувати CSS-модулі із зазначенням розширення .module.scss або .module.sass.

3.2.2 Глобальні змінні стилів

Щоб централізовано керувати кольорами, розмірами та іншими важливими параметрами, був створений файл `variables.scss` у директорії “src/app”.

Слід зазначити, що ініціалізація змінних на сьогодні реалізована як в чистому `css` так і як окреме рішення в `Scss`.

`CSS` змінні, також відомі як “кастомні властивості”, визначаються з префіксом “--” і використовуються за допомогою функції `var()`. Вони обмежені лише зміною значень, але не можуть включати інші особливості `SCSS`, такі як вкладення або міксини `CSS` змінні дозволяють створювати глобальні та локальні змінні. Глобальні змінні визначаються у псевдокласі `:root` і доступні у всьому документі, тоді як локальні змінні доступні лише у межах обраного селектора [27].

`Scss` змінні є частиною препроцесора `Sass`, який розширює можливості стандартного `Css`. `Scss` змінні визначаються з префіксом `$` і можуть зберігати будь-які `Css` властивості, значення кольорів, відступів тощо. Також підтримується вкладення, міксини та функції, що робить їх більш гнучкими для використання у складних проектах [28].

Варто зазначити, що змінні в `Scss` мають один суттєвий недолік: вони не дозволяють змінювати значення змінних за певних умов, наприклад, при медіа-запитах. У випадку визначення глобальних змінних цей недолік є критичним, тому обрано використання звичайних `Css` змінних.

У файлі `variables.scss` визначені основні змінні, які використовуються для стилізації додатку. Код даного файлу наведено у лістингу 3.1.

Лістинг 3.1 – Код файлу `variables.scss`

```
#body {
  --accent-color-primary: rgba(244, 241, 232, 1);
  --accent-color-secondary: rgba(249, 115, 22, 1);
  --text-color-primary: rgba(51, 51, 51, 1);
  --text-color-secondary: rgba(28, 39, 76, 1);

  --border-radius: 8px;

  --container-width: 1280px;
  --container-padding-horizontal: 20px;
}

@media (max-width: 768px) {
```

```
#body {
  --container-padding-horizontal: 10px;
}
```

Змінні визначені всередині селектора `#body`, що дозволяє їх використання в межах створених елементів вебсайту. Змінні `Css` можуть бути використані у стилях через функцію `var()`.

Усі загальні стилі можна поділити на значення основних кольорів та стилі блоків. Змінна `“container-width”` визначає максимальну ширину контейнера з контентом вебсайту, при зменшенні ширини екрану дана ширина буде відповідно зменшуватись, що забезпечить адаптивність розмітки. Змінна `“container-padding-horizontal”` забезпечить відступи від країв екрану на пристроях з шириною екрану менше 1280 пікселів.

Для оптимізації зайнятого простору на пристроях з шириною екрану менше ніж у середньостатистичному планшеті виконано медіа-запит із параметром `“max-width: 768px”` в середині якого зменшено значення змінної `“container-padding-horizontal”`.

`Next.js` за замовчування підтримує використання `Css-модулів`. Вони гарантують, що стилі, визначені в одному модулі, будуть унікальними і ніяк не вплинуть на стилі інших модулів. Але, одночасно з цим даний інструмент робить неможливим використання глобальних змінних. Щоб вирішити дану проблему відкореговано файл налаштувань проекту `next.config.js`. Змінений код файлу конфігурації наведено у лістингу 3.2.

Лістинг 3.2 – Налаштування файлу конфігурації для використання `variables.scss`

```
module.exports = {
  sassOptions: {
    includePaths: [path.join(__dirname, 'styles')],
    prependData: `@import "@app/variables.scss";`,
  },
};
```

Код, наведений у лістингу 3.2, імпортує файл `variables.scss` до усіх файлів

стилів, що забезпечує доступ до визначених загальних змінних.

3.2.4 Налаштування пакету для роботи з SVG форматом

Вбудований тег Image у Next.js не надає можливості стилізувати svg зображення так, як це можна зробити за допомогою компонентів. Зокрема, неможливо змінювати атрибути векторних зображень безпосередньо, що значно обмежує можливості динамічного стилізування SVG графіки.

Імпорт svg файлів як React компонентів забезпечує більшу гнучкість у використанні векторної графіки. Це дозволяє змінювати атрибути svg, такі як розміри, кольори та інші стилі.

Тому для комфортного використання svg графіки у проект був встановлений пакет svgr, що дозволяє імпортувати svg файли як React компоненти. Було внесено зміни до файлу next.config.js, щоб забезпечити правильну обробку зображень векторного формату [29]. Зміст конфігурації пакету взято з фіційного сайту react-svgr та зображено в лістингу 3.3.

Лістинг 3.3 – Налаштування svgr пакету в next.config.js

```
module.exports = nextConfig;
module.exports = {
  sassOptions: {
    includePaths: [path.join(__dirname, 'styles')],
    prependData: `@import "@/app/variables.scss";`,
  },
  webpack(config) {
    const fileLoaderRule = config.module.rules.find(rule =>
rule.test?.test?.('.svg'));

    config.module.rules.push(
      {
        ...fileLoaderRule,
        test: /\.svg$/i,
        resourceQuery: /url/, // *.svg?url
      },
      {
        test: /\.svg$/i,
        issuer: fileLoaderRule.issuer,
        resourceQuery: { not: [...fileLoaderRule.resourceQuery.not, /url/] }, //
exclude if *.svg?url
        use: ['@svgr/webpack'],
      }
    );

    fileLoaderRule.exclude = /\.svg$/i;

    return config;
  },
};
```

3.3 Імплементація бази даних

Процес налаштування ORM Prisma для роботи з базою даних у проєкті на Next.js включає кілька важливих етапів.

Першим кроком є встановлення пакету Prisma у проєкт. Для цього використовується команда “`npm install prisma --save-dev`”. Ця команда додає Prisma як залежність для розробки, що дозволяє використовувати його інструменти для налаштування та управління базою даних [30].

Після встановлення пакету необхідно згенерувати базову конфігурацію Prisma. Це робиться за допомогою команди “`prisma init`”.

Ця команда створює файл `schema.prisma` у директорії `prisma`, а також файл `.env`, який буде використовуватись для зберігання конфіденційних налаштувань, таких як рядки підключення до бази даних. Файл `schema.prisma` містить основні налаштування для роботи з базою даних, включаючи визначення моделей даних.

Для зберігання даних було вирішено використовувати віддалену базу даних, створену у сервісі Supabase. Supabase є популярним сервісом, який забезпечує зручний інтерфейс для управління базами даних, а також підтримує інтеграцію з Prisma [31].

Для створення бази даних зареєструвано на сайті Supabase та створено новий проєкт. Після створення проєкту отримано рядок URI для підключення до бази даних, який додано у змінну файлу `.env` з назвою “`DATABASE_URL`”.

Перед підключенням до віддаленої бази даних необхідно відкорегувати файл `schema.prisma`. Додано моделі таблиць: `Users`, `Favorites`, `Reviews`. Відкорегований код файлу схеми зображено у лістингу 3.4.

Лістинг 3.4 – Код файлу `schema.prisma`

```
generator client {  
  provider = "prisma-client-js"  
}  
  
datasource db {  
  provider = "postgresql"}
```

```

    url      = env("DATABASE_URL")
  }

model Users {
  id        Int          @id @default(autoincrement())
  email     String       @unique
  username  String       @unique
  password  String
  avatarUrl String?
  favorites Favorites[]
  reviews  Reviews[]
}

model Favorites {
  id        Int          @id @default(autoincrement())
  userId    Int
  recipeId  Int

  user      Users        @relation(fields: [userId], references: [id])

  @@index([userId])
}

model Reviews {
  id        Int          @id @default(autoincrement())
  userId    Int
  date      DateTime     @default(now())
  text      String
  rating    Int
  recipeId  Int

  user      Users        @relation(fields: [userId], references: [id])

  @@index([userId])
}

```

Блок “generator” визначає, який клієнт буде згенерований для взаємодії з базою даних, в даному випадку він вказує, що буде згенерований Prisma Client для JavaScript.

Блок datasource визначає джерело даних для підключення до бази даних. Provider вказує на використання PostgreSQL як бази даних, а url використовує змінну оточення DATABASE_URL для використання рядка підключення до бази даних Supabase.

Опис моделей представляє таблиці із вказанням полів та зв’язків між таблицями. Наприклад: модель Users представляє таблицю користувачів. Вона включає кілька полів, таких як ідентифікатор користувача (id), електронна пошта (email), ім'я користувача (username), пароль (password) та URL аватара користувача (avatarUrl). Поле id є первинним ключем з автоматичним інкрементом, а поля email та username мають унікальні значення. Крім того, модель Users має зв'язки з

моделями Favorites і Reviews, що дозволяє зберігати інформацію про улюблені рецепти та відгуки користувачів.

Рядки “@@index([userId])” присутні для створення індексів на відповідні поля таблиць бази даних. Індеси служать для прискорення виконання запитів, що використовують ці поля.

Після налаштування файлу schema.prisma необхідно застосувати міграції для створення таблиць у базі даних. Для цього виконано команду “npx prisma migrate dev --name init”.

Останнім етапом є встановлення Prisma Client, який використовується для виконання запитів до бази даних у коді. Для цього виконано команду “npm install @prisma/client” та редаговано файл db.ts у каталозі “src/lib”. Результуючий код файлу db.ts наведено у лістингу 3.5.

Лістинг 3.5 – Код файлу db.ts

```
import { PrismaClient } from '@prisma/client';

declare global {
  var prisma: PrismaClient;
}

let prisma: PrismaClient;

if (process.env.NODE_ENV === 'production') {
  prisma = new PrismaClient();
} else {
  if (!global.prisma) {
    global.prisma = new PrismaClient();
  }
  prisma = global.prisma;
}

export default prisma;
```

Зміст файлу, зображеного в лістингу 3.5, створює екземпляр PrismaClient. Він налаштований таким чином, щоб уникати створення нових екземплярів клієнта у середовищі розробки, що запобігає проблемам з перевантаженням клієнтів при гарячому перезавантаженні модулів.

3.4 Розробка користувацького інтерфейсу

Розробка користувацького інтерфейсу (UI) є важливим етапом у створенні вебсайту для пошуку рецептів. Цей процес включає налаштування загального макету сайту, створення загальних компонентів, таких як Header та Footer, а також розробку специфічних компонентів UI та сторінок.

3.4.1 Налаштування загального макету сайту

Першим кроком є налаштування загального макету сайту у файлі layout.tsx. Цей файл відповідає за структуру сторінок і забезпечує загальне оформлення для всіх сторінок вебсайту. Макет зазвичай включає загальні компоненти, такі як Header та Footer, а також контентну область для вмісту кожної сторінки. Лістинг 3.6 містить код файлу layout.tsx.

Лістинг 3.6 – Код файлу layout.tsx

```
const font = Nunito_Sans({
  subsets: ['latin'],
  weight: ['400', '500', '600', '700'],
});

export const metadata: Metadata = {
  title: 'Menu mastermind',
  description: 'Recipes for everyone',
};

export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang='en'>
      <body className={font.className} id='body'>
        <div className='wrapper'>
          <Header></Header>
          <div className='wrapper__pageContent'>{children}</div>
          <Footer></Footer>
        </div>
      </body>
    </html>
  );
}
```

На початку лістингу 3.6 відбувається імпорт і налаштування найбільш поширеного сімейства шрифту, яке буде використовуватись на вебсайті. Далі змінна font буде використовуватись для визначення сімейства шрифту за

замовчуванням для тега `body`.

Основним компонентом макета є функція `RootLayout`, яка обгортає весь вміст сторінки. Вона встановлює мову документа як англійську за допомогою тега `html` з атрибутом `lang`. Тег `body` застосовує клас шрифту та встановлює ідентифікатор `id`, що дозволяє легко застосовувати стилі до всього тіла документа.

Внутрішня структура сторінки міститься у елементі `div` з класом `wrapper`, який включає компоненти `Header` і `Footer`. Ці компоненти відповідають за відображення заголовка та підвалу сторінки відповідно. Основний контент сторінки передається як дочірні елементи до елементу `div` з класом `“wrapper__pageContent”`, що дозволяє динамічно змінювати вміст сторінки.

Слід зазначити що цей компонент визначає структуру усіх сторінок на вебсайті. Компоненти `Header` та `Footer` залишаються однаковими для усіх сторінок, в свою чергу пропс `“children”` визначає контент певної сторінки який динамічно підставляється.

3.4.2 Створення загальних компонентів `Header` та `Footer`

Компонент `Header` відповідає за відображення заголовка сторінки, включаючи логотип та навігаційні посилання. Код даного компоненту зображено у лістингу 3.7.

Спершу, у компоненті визначаються дві кнопки: `SignInButton` та `ProfileButton`. Кнопка `SignInButton` містить посилання на сторінку входу (`/sign-in`) та обгортає компонент `Button`. Кнопка `ProfileButton` містить посилання на сторінку профілю (`/profile/profileData`) та обгортає компонент `Button`, який включає відображення імені користувача та іконки профілю.

Лістинг 3.7 – Код компоненту `Header`

```
const Header = async () => {
  const SignInButton = (
    <Link href={'/sign-in'} className={styles.nav__link}>
      <Button>Sign in</Button>
    </Link>
  );
  const ProfileButton = (
    <Link href={'/profile/profileData'} className={styles.nav__link}>
      <Button>
        <div className={styles.signIn}>
          <span>Username</span>
        </div>
      </Button>
    </Link>
  );
}
```

```

        <Image src={'/profile_placeholder.svg'} height={25} width={25}
alt='profile' />
      </div>
    </Button>
  </Link>
);
return (
  <header className={styles.header}>
    <div className={styles.header__body}>
      <Link href={'/'} className={styles.nav__link}>
        <div className={styles.logo}>
          <Image src={'/logo.png'} height={35} width={39} alt='logo'
/>
          MenuMastermind
        </div>
      </Link>
      <nav className={styles.nav}>
        {userAuthorized && (
          <Link href={'/sign-up'} className={styles.nav__link}>
            Sign up
          </Link>
        )}
        {userAuthorized ? ProfileButton : SignInButton}
      </nav>
    </div>
  </header>
);
};

```

Основна частина компонента повертає JSX, який описує структуру заголовка. У заголовку міститься контейнер з класом “header__body”, всередині якого знаходиться логотип і навігаційне меню. Логотип є посиланням на головну сторінку (/) і містить зображення логотипу та текст “MenuMastermind”.

Навігаційне меню (nav) динамічно відображає різні кнопки залежно від стану автентифікації користувача. Якщо користувач авторизований (userAuthorized), відображається посилання на сторінку реєстрації (/sign-up) та кнопка профілю. Якщо користувач не авторизований, відображається кнопка входу.

Слід зазначити що на даному етапі автентифікація не реалізована, тому все визначає тимчасова змінна “userAuthorized”.

Компонент Footer відображає структуру підвалу вебсайту, що включає два основні елементи: текст про авторські права та список посилань на правові документи. Усі ці елементи містяться в контейнері з класом “footer__body”. Код самого компоненту Footer зображено у лістингу 3.8.

Лістинг 3.8 – Код компоненту Footer

```
const Footer = () => {
  return (
    <footer className={styles.footer}>
      <div className={styles.footer__body}>
        <div className={styles.footerLeft}>© 2024
MenuMastermind</div>
        <ul className={styles.footerRight}>
          <li>Legal Terms</li>
          <li>Privacy policy</li>
        </ul>
      </div>
    </footer>
  );
};
```

Компонент починається з визначення функції Footer, яка повертає JSX, що описує структуру підвалу. Вся структура обгорнута в тег <footer> з класом styles.footer, який відповідає за основні стилі підвалу.

Внутрішній контейнер з класом “footer__body” ділить підвал на дві частини: ліву і праву. Ліва частина (footerLeft) містить текст з авторськими правами: “© 2024 MenuMastermind”. Цей текст інформує користувачів про права на контент сайту.

Права частина підвалу представлена у вигляді списку (footerRight), який містить два елементи: “Legal Terms” та “Privacy policy”. Ці елементи можуть бути використані для створення посилань на відповідні сторінки з правовою інформацією, що важливо для забезпечення користувачів інформацією про умови використання сайту та політику конфіденційності.

3.4.3 Створення компонентів UI

Компонент Button визначений як функціональний компонент React. Він приймає пропси, які дозволяють налаштовувати його поведінку та вміст. Зокрема, пропси включають children, type, та onClick. Код компоненту зображено у лістингу 3.9.

Лістинг 3.9 – Функціональний компонент Button

```
import React from 'react';
import styles from './button.module.scss';

interface ButtonProps {
  children: React.ReactElement | String;
  type?: 'button' | 'submit' | 'reset';
  onClick?: () => void;
```

```

}

const Button = ({ children, type = 'button', onClick }: ButtonProps) => {
  return (
    <button type={type} className={styles.button} onClick={onClick}>
      {children}
    </button>
  );
};

export default Button;

```

Детальніше про пропси:

- `children`: вставляється в компонент як обов’язковий зміст кнопки, може бути елементом React або текстовим рядком;
- `type`: необов’язковий параметр який визначає тип кнопки (`button`, `submit`, або `reset`), за замовчуванням – `button`;
- `onClick`: необов’язкова функція-обробник, яка викликається при натисканні кнопки.

На даному прикладі можна розгляну використання Css-модулів в Next.js. Спочатку вміст файлу `button.module.scss` імпортується та зберігається як `styles`, після чого значення поля даного об’єкту, еквівалентного назві класу в файлі стилів, присвоюється атрибуту “`className`” необхідно блоку.

На прикладі лістингу 3.10, у якому зображені стилі для компоненту `Button` можна виділити два ключових прийома: використання `css` змінних та функції `clamp`. Змінні беруться з файлу `variables.scss`, який автоматично імпортується до кожного файлу стилів, а функція `clamp` допомагає визначити максимально адаптивне значення шрифту для тексту кнопки.

Лістинг 3.10 – Стилi компоненту Button

```

.button {
  padding: 10px;
  border-radius: var(--border-radius);
  background-color: var(--accent-color-secondary);

  display: flex;
  justify-content: center;
  align-items: center;

  //14-16px
  font-size: clamp(0.88rem, calc(0.83rem + 0.21vw), 1rem);
  font-weight: 700;
}

```

```

    color: #fafaf9;
}

```

Функція `clamp` приймає 3 аргумента: мінімальне значення, рекомендоване та максимальне. Підраховане значення буде дорівнювати рекомендованому якщо мінімальне чи максимальне значення його не обмежують.

Для визначення розміру шрифту використано калькулятор для обчислення адаптивного значення `Observablehq` [32]. Над визначенням розміру шрифту можна помітити коментар “14-16px”, це мінімальне та максимальне значення, задані в калькуляторі. Також калькулятор вимагає визначити мінімальну та максимальну ширину екрану, для проєктованого вебсайту це 375px та 1280px відповідно. На практиці це задає значення шрифту 16px при ширині екрану 1280px, при зменшенні розміру екрану користувача розмір шрифту буде плавно зменшуватись, поки не стане рівним 14px на телефоні з шириною екрану 375px.

Також слід зазначити, що дане значення обчислюється не в px, а в rem. Це відносна одиниця розміру, на яку впливає стандартний розмір шрифту у тегу `body` або налаштування браузера. Таким чином це допомагає підвищити доступність вебсайту, щоб користувачі із вадами зору завжди могли зручно користуватися вебсайтом.

Компонент `RecipeCard`, код якого зображено у лістингу 3.11, відповідає за відображення картки рецепту, яка включає посилання на рецепт, назву, зображення, час приготування та кількість калорій.

Компонент визначений як функціональний компонент `React`, який приймає пропси, що описуються інтерфейсом `RecipeCardProps`. Пропси включають `href` для посилання на сторінку рецепту, `title` для назви рецепту, `image` для URL зображення, `numberInMinutes` для часу приготування в хвилинах і `calories` для кількості калорій.

Лістинг 3.11 – Компонент `RecipeCard`

```

interface RecipeCardProps {
  href: string;
  title: string;
  image: string;
  numberInMinutes: number;
  calories: number;
}

```

```

    }

    const RecipeCard = ({ href, title, image, numberInMinutes, calories }:
RecipeCardProps) => {
    return (
      <Link href={href} className={s.recipeCard}>
        <Image src={image} height={196} width={270} alt='recipe-img'
className={s.recipeCard__img} />

        <div className={s.recipeCard__textContent}>
          <div className={s.recipeCard__title}>{title}</div>
          <div className={s.recipeCardParams}>
            <Param imgSrc='/clock.svg' text={` ${numberInMinutes}
minutes`} />
            <Param imgSrc='/calories.svg' text={` ${calories}
calories`} />
          </div>
        </div>
      </Link>
    );
  };

```

Компонент повертає JSX, який визначає структуру картки. Він містить елемент `Link`, який обгортає всю картку і надає посилання на сторінку рецепту. Внутрішньо містить компонент `Image`, що відображає зображення рецепту з вказаними розмірами. Текстовий вміст картки включає `div` з класом “`recipeCard__textContent`”, який містить назву рецепту (`recipeCard__title`) і параметри (`recipeCardParams`). Параметри відображаються за допомогою компонента `Param`, який приймає зображення і текст для кожного параметра, такого як час приготування та калорії.

Компонент `Review` призначений для відображення огляду користувача, включаючи його аватар, ім'я автора, дату публікації, рейтинг і коментар. Компонент використовує інтерфейс `ReviewProps` для визначення типів даних, які він приймає.

Компонент визначений як функціональний і приймає наступні пропси:

- `avatarSrc`: URL зображення аватара автора огляду;
- `author`: ім'я автора огляду;
- `publishDate`: дата публікації огляду;
- `rating`: рейтинг, наданий автором;
- `comment`: текст коментаря.

Компонент повертає JSX, який визначає структуру огляду. Основний

контейнер має клас `review`, що включає кілька підконтейнерів для відображення різних частин огляду.

Перший підконтейнер `“review__img”` містить зображення аватара автора, відображене за допомогою компонента `Image` з вказаними висотою та шириною. Другий підконтейнер `“review__title”` містить ім'я автора, обгорнуте в заголовок (`<h6>`), та дату публікації, відображену як параграф (`<p>`).

Третій підконтейнер `“review__starRating”` відображає рейтинг огляду, а четвертий `“review__comment”` містить текст коментаря.

Код компоненту `Review` наведено у лістингу 3.12.

Лістинг 3.12 – Компонент `Review`

```
const Review = ({ avatarSrc, author, publishDate, rating, comment }: ReviewProps)
=> {
  return (
    <div className={s.review}>
      <div className={s.review__img}>
        <Image src={avatarSrc} height={30} width={30} alt='avatar'
      />
      </div>
      <div className={s.review__title}>
        <h6>{author}</h6>
        <p>{publishDate}</p>
      </div>
      <div className={s.review__starRating}>{rating}</div>
      <div className={s.review__comment}>{comment}</div>
    </div>
  );
};
```

Компонент `Searchbar` призначений для надання користувачам можливості пошуку вмісту на вебсайті. Він приймає наступні пропси:

- `defaultValue`: початкове значення для пошукового поля (необов'язковий);
- `placeholder`: текст плейсхолдера для пошукового поля (необов'язковий);
- `onSubmit`: функція, яка викликається при поданні форми з введеним значенням.

Так як компонент має відслідковувати дії користувача та відповідно реагувати на них то для `Next.js` його визначено як клієнтський компонент: першим рядком у файлі компоненту написано `“use client”`.

Компонент використовує хук `useState` для керування станом введеного

значення, а також хук `useEffect`, щоб оновлювати значення поля вводу при зміні пропса `defaultValue`. Код компоненту `Searchbar` наведено у лістингу 3.13.

Лістинг 3.13 – Компонент `Searchbar`

```
interface SearchbarProps {
  defaultValue?: string;
  placeholder?: string;
  onSubmit: (value: string) => void;
}

const Searchbar = ({ defaultValue, placeholder, onSubmit }: SearchbarProps) => {
  const [inputValue, setInputValue] = useState<string>(defaultValue || '');

  useEffect(() => {
    setInputValue(defaultValue || '');
  }, [defaultValue]);

  function onChangeHandler(e: React.ChangeEvent<HTMLInputElement>) {
    setInputValue(e.target.value);
  }

  return (
    <form
      className={styles.searchbar}
      onSubmit={e => {
        e.preventDefault();
        if (inputValue !== '') onSubmit(inputValue);
      }}
    >
      <input
        className={styles.searchbar__input}
        type='text'
        name='search'
        placeholder={placeholder ? placeholder : 'Search'}
        value={inputValue}
        onChange={e => onChangeHandler(e)}
      />

      <div className={styles.searchbar__imgSection}>
        <Image src={'/faq-icon.svg'} height={35} width={35}
alt='faq-icon' />
      </div>
    </form>
  );
};
```

Функція `onChangeHandler` оновлює стан `inputValue` при кожній зміні значення в полі вводу.

JSX розмітка компонента включає форму (`<form>`), яка обробляє подію `onSubmit` для запобігання перезавантаженню сторінки і викликає функцію `onSubmit`, якщо поле вводу не порожнє.

3.4.4 Створення компонентів сторінок

Відповідно до вимог проекту, необхідно створити головну сторінку з полем для пошуку та слайдером випадкових рецептів, сторінку пошуку, сторінку окремого рецепту, сторінки для автентифікації та реєстрації, а також сторінку профілю користувача.

Головна сторінка має забезпечити перше враження користувача про вебсайт. Вона включає поле для пошуку рецептів та слайдер із випадковими рецептами. Розмітка сторінки наведена у лістингу 3.14.

Лістинг 3.14 – Розмітка головної сторінки

```
<main className={styles.home}>
  <section className={styles.home__section}>
    <Image src={'/home-bg_2.png'} alt='home-bg' layout='fill'
objectFit='cover' className={styles.sectionBackground} />
    <div className={styles.overlay}></div>
    <Searchbar onSubmit={onSearchbarSubmit} />
  </section>

  <section className={styles.sliderSection}>
    <div className={styles.sliderSection__title}>
      <h2>Culinary Delights</h2>
      <h5>Feast your eyes on dishes from around the world.</h5>
    </div>

    <div className={styles.sliderSection__slider}>
      <HomeSlider recipes={randomRecipes} />
    </div>
  </section>
</main>
```

Як зображено у лістингу вище, контент усієї сторінки поділений на дві секції, перша з яких на всю висоту екрану користувача і містить у собі поле для пошуку. Нижня секція містить компонент HomeSlider до якого передається масив із зображеннями випадкових рецептів. Цей компонент відображає слайдер із зображень, кожне з яких міститься усередині посилання яке веде на сторінку відповідного рецепту.

Сторінка пошуку призначена для відображення результатів пошукових запитів користувача. Вона включає список рецептів, які відповідають критеріям пошуку, та надає користувачам можливість робити нові пошукові запити.

Як можна побачити на лістингу 3.15, дана сторінка пошуку насамперед

відображає поле для пошуку із зазначеним пошуковим параметром. Під полем розташовується результат пошуку у вигляді масиву раніше описаного компонента RecipeCard, якій надається інформація про окремий рецепт.

Лістинг 3.15 – Розмітка сторінки пошуку рецептів

```
<main className={s.searchPage}>
  <div className={s.searchPageContainer}>
    <LocalSearchbar defaultValue={query} />

    <div className={s.cardsContainer}>
      {recipesArr.length > 0 ? (
        recipesArr.map(recipe => (
          <RecipeCard
            key={recipe.id}
            href={`recipe/${recipe.id}`}
            title={recipe.title}
            image={recipe.image}
            numberInMinutes={recipe.readyInMinutes}
            calories={recipe.calories}
          />
        ))
      ) : (
        <p>Recipes not found</p>
      )}
    </div>
  </div>
</main>
```

У місці відображення карток рецептів використано умовний рендеринг за допомогою тернарного оператора. Якщо результат пошуку налічує щонайменше 1 рецепт, то користувачеві буде показано даний результат. У випадку, якщо рецептів не знайдено, користувач побачить параграф із текстом, у якому буде зазначено, що рецептів не знайдено.

Сторінка окремого рецепту забезпечує детальну інформацію про обраний рецепт. Вона включає інгредієнти, інструкції з приготування, час приготування, калорійність та інші важливі деталі, а також можливість залишати відгуки та оцінювати рецепт.

Дана сторінка складається з трьох основних частин обгорнутих обмежувачами по ширині та вирівнюваними по центру блоками:

- власне рецепту;
- списку відгуків;
- форми щоб залишати відгук.

Розмітка секції з детальною інформацією про рецепт зображена у лістингу 3.16.

Лістинг 3.16 – Розмітка секції рецепту

```
<div className={s.recipe}>
  <div className={s.recipe__img}>
    <Image src={recipeInfo.image} height={350} width={350}
alt='recipe-img' />
  </div>
  <div className={s.recipe__mainInfo}>
    <div className={s.flexSpaceBetween}>
      <h3 className={s.title}>{recipeInfo.title}</h3>
      {session?.user && FavoriteButtonBlock}
    </div>
    <div className={s.row}>
      {recipeInfo.categories.map((category, id) => (
        <Category text={category} key={id} />
      ))}
    </div>
    <div className={s.starRating}>
      <span>{stringifiedRating}</span>
      <span>({reviewsNumber})</span>
    </div>
    <div className={s.row}>
      <div className={s.param}>{`Ready in:
${recipeInfo.readyInMinutes}min`}</div>
      {recipeInfo.preparationMinutes > 0 && (
        <div className={s.param}>{`Prep time:
${recipeInfo.preparationMinutes}min`}</div>
      )}
      {recipeInfo.cookingMinutes > 0 && (
        <div className={s.param}>{`Cooking time:
${recipeInfo.cookingMinutes}min`}</div>
      )}
    </div>
    <div className={s.description} dangerouslySetInnerHTML={{
__html: recipeInfo.summary }}></div>
  </div>
  <div className={s.recipe__ingredients}>
    <h3 className={s.title}>Ingredients</h3>
    <ul>
      {recipeInfo.ingredients.map((ingredient, id) => (
        <li key={id}>{ingredient.name}</li>
      ))}
    </ul>
  </div>
  <div className={s.recipe__cookProcess}>
    <h3 className={s.title}>Cooking process</h3>
    <p>{recipeInfo.instructions}</p>
  </div>
</div>
```

Основний контейнер з класом “recipe” містить кілька підрозділів інформації про рецепт:

- зображення блюда;

- назва блюда поруч з якою відображається кнопка “додати до списку улюблених” (лише для авторизованих користувачів);
- список категорій до яких блюдо належить;
- рейтинг серед користувачів;
- час на приготування;
- короткий опис блюда;
- список інгредієнтів;
- процес приготування.

Список відгуків відображається як декілька компонентів Review якому, у якості пропсів, передаються дані відгуку.

Сторінки аторизації (Sign in) та реєстрації (Sign up) мають майже ідентичну структуру та представляють собою форми із заголовком та полями для введення даних із слідуючою за ними кнопкою надсилання форми. Нижче буде розглянуто форму реєстрації, так як вона містить у собі ті ж самі елементи що і форма для автентифікації.

Почати можна з того, що обидві форми є клієнтськими компонентами, розташованими по центру сторінки. Це дозволяє відслідковувати дії користувача та виконувати валідацію полів форми з боку клієнту. Код компоненту форми реєстрації наведено у додатку А лістингу А.1.

Для виконання валідації використовується атрибут pattern:

- “^(?!\\s*\$).{3,}\$” для поля з ім’ям користувача;
- “^[a-zA-Z0-9]+(?:\\.[a-zA-Z0-9]+)*@[a-zA-Z0-9]+(?:\\.[a-zA-Z0-9]+)*\$” для поля з електронною поштою;
- “. {6,}” для поля з паролем відповідно.

Ім'я повинно містити принаймні 3 символи. Поле не може бути порожнім або складатися лише з пробілів. Електронна пошта повинна відповідати стандартному формату email-адреси: має містити літери та/або цифри, за якими може йти крапка, та потім знак @ з доменом. Наприклад, example@example.com. Пароль повинен містити принаймні 6 символів, дозволяються усі типи символів.

У випадку непроходження валідації поле вводу буде виділене як некоректне

вбудованими засобами браузера, а функція `onSubmit` не надішле запит на сервер доки користувач не введе валідні дані.

Компонент `ProfilePage` представляє сторінку профілю користувача з можливістю перемикання між двома вкладками: профільними даними та списком улюблених рецептів. Головний контейнер `main` має клас `“profilePage”` і містить внутрішній контейнер `“profilePage__container”`. Код даної сторінки наведено у додатку А лістингу А.2.

На верхній частині сторінки розташовані вкладки (`tabs`), які дозволяють користувачеві перемикатися між вкладками `“Profile”` та `“Favorites”`. При натисканні на вкладку викликається функція `handleTabClick`, яка змінює активну вкладку.

У нижній частині сторінки розташовані контейнери для вмісту вкладок (`tabContentContainer`). Перший контейнер з ID `tab1` відповідає за вміст вкладки `"Profile"`. У ньому відображається блок профілю (`profileBlock`), який містить заголовок та форми для редагування імені та електронної пошти користувача. Користувач також може натиснути на посилання для зміни пароля (`changePassword`) та кнопку для виходу з облікового запису (`Button`), що викликає функцію `signOut`.

Другий контейнер з id `tab2` відповідає за вміст вкладки `“Favorites”`. Якщо у користувача є улюблені рецепти, вони відображаються у вигляді компонентів `RecipeCardFavorite`. Якщо список улюблених рецептів порожній, відображається повідомлення `“Your favorites-list is empty”`.

3.5 Імплементація реєстрації та автентифікації

Для імплементації реєстрації створено таблицю у базі даних з полями для зберігання інформації про користувачів. Таблиця містить наступні поля: `id`, `email` (унікальне), `username` (унікальне), `password` та `avatarUrl` (необов'язкове).

Процес реєстрації реалізовано за допомогою роута, розташованого в папці

“app/user/register”. Код файлу даного роута наведено у лістингу 3.17.

Ліістинг 3.17 – Код роута для реєстрації користувача

```
export async function POST(req: Request, res: Response) {
  try {
    const body = await req.json();
    const { email, username, password } = body;
    const emailExist = await prisma.users.findUnique({
      where: {
        email: email,
      },
    });
    if (emailExist) {
      return NextResponse.json({
        success: false,
        message: 'Email already exist',
      });
    }
    const usernameExist = await prisma.users.findUnique({
      where: {
        username: username,
      },
    });
    if (usernameExist) {
      return NextResponse.json({
        success: false,
        message: 'Username already exist',
      });
    }
    const hashedPassword = await hash(password, 10);
    const newUser = await prisma.users.create({
      data: {
        username: username,
        email: email,
        password: hashedPassword,
      },
    });
    return NextResponse.json({
      success: true,
      message: 'User created successfully',
    });
  } catch (error) {
    return NextResponse.json({
      success: false,
      message: 'Something went wrong',
    });
  }
}
```

Важливим буде зазначити, що при реєстрації в таблицю бази даних заносяться усі дані в їх початковому стані, окрім паролю. Для забезпечення безпеки користувачів в базі даних зберігається хеш паролю. Таким чином у випадку, якщо база даних з пароллями буде зламана або втрачена, хакери не зможуть отримати доступ до реальних паролів користувачів.

Для реалізації автентифікації користувача у проекті було використано бібліотеку NextAuth, яка забезпечує простий та зручний спосіб додавання автентифікації до Next.js додатків. Процес реалізації включає створення роута для обробки запитів на авторизацію, а також налаштування опцій NextAuth.

Роут для обробки запитів автентифікації розташований у файлі `api/[...nextauth]/route.ts`. Код цього файлу наведено у лістингу 3.18.

Лістинг 3.18 – Код роуту автентифікації

```
import { authOptions } from '@lib/auth';
import NextAuth from 'next-auth';

const handler = NextAuth(authOptions);

export { handler as GET, handler as POST };
```

Код даного роуту імпортує опції автентифікації з файлу `auth.ts` і передає їх у функцію `NextAuth`, яка повертає обробник запитів. Цей обробник використовується для обробки як GET, так і POST запитів до роуту.

Файл `auth.ts` у директорії `app/lib` містить конфігурацію `NextAuth`, яка визначає, як саме відбувається автентифікація користувачів. Код конфігурації наведено у лістингу 3.19.

Лістинг 3.19 – Код конфігурації NextAuth

```
export const authOptions: NextAuthOptions = {
  adapter: PrismaAdapter(prisma),
  secret: process.env.NEXTAUTH_SECRET,
  session: {
    strategy: 'jwt',
  },
  pages: {
    signIn: '/sign-in',
  },
  providers: [
    CredentialsProvider({
      name: 'Credentials',
      credentials: {
        email: { label: 'Email', type: 'text', placeholder: 'Enter your email' },
        password: { label: 'Password', type: 'password' },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) return null;

        const existingUser = await prisma.users.findUnique({
          where: {
```

```

        email: credentials.email,
      },
    });
    if (!existingUser) return null;
    const passwordMatch = await compare(credentials.password,
existingUser.password);
    if (!passwordMatch) return null;
    return {
      id: existingUser.id + '', // id value should be a
string
      username: existingUser.username,
      email: existingUser.email,
    };
  },
  }),
],
callbacks: {
  async jwt({ token, user }) {
    if (user) {
      return {
        ...token,
        username: user.username,
      };
    }
    return token;
  },
  async session({ session, token }) {
    return {
      ...session,
      user: {
        ...session.user,
        username: token.username,
      },
    };
  },
},
};

```

Дана конфігурація NextAuth використовує кілька ключових компонентів:

- PrismaAdapter дозволяє NextAuth взаємодіяти з базою даних через Prisma, забезпечуючи зручну інтеграцію між NextAuth і базою даних;
- значення NEXTAUTH_SECRET використовується для підпису токенів, що забезпечує безпеку та цілісність даних сесій;
- для збереження сесій використовується стратегія JWT (JSON Web Token), що дозволяє зберігати сесії у форматі токенів, які легко передавати між клієнтом і сервером.

Конфігурація також вказує на кастомну сторінку автентифікації, що дозволяє налаштувати інтерфейс автентифікації відповідно до вимог проекту. Провайдер CredentialsProvider використовується для автентифікації за допомогою електронної пошти та пароля. При цьому, коли користувач вводить свої облікові дані, функція

authorize перевіряє їхню коректність, використовуючи бібліотеку bcrypt для порівняння введеного пароля з хешованим паролем у базі даних.

Callbacks використовуються для модифікації JWT токенів та сесійних об'єктів, додаючи до них ім'я користувача. Це дозволяє зберігати додаткову інформацію про користувача у сесії та забезпечує більш персоналізований досвід користувача.

Код надсилання запиту на авторизацію наведено у лістингу 3.20. При натисканні кнопки "Submit", викликається асинхронна функція onSubmit, яка спочатку запобігає стандартному перезавантаженню сторінки за допомогою e.preventDefault(). Після чого відбувається спроба авторизації: виконується функція бібліотеки NextAuth signIn. У випадку успішної авторизації користувач направляється на головну сторінку де він одразу може здійснити запит на пошук рецептів, у іншому випадку користувачу показується повідомлення про помилку.

Лістинг 3.20 – Код запиту на авторизацію

```
onSubmit={async e => {
  e.preventDefault();
  setInputsError(false);
  try {
    const signInData = await signIn('credentials', {
      email: emailRef.current?.value,
      password: passwordRef.current?.value,
      redirect: false,
    });
    if (signInData?.error) {
      console.log(signInData.error);
      setInputsError(true);
    } else {
      router.push('/');
    }
  } catch (error) {
    setShowServerErrorMsg(true);
  }
}}
```

3.6 Динамічне формування контенту

Next.js надає можливості обирати тип компонента: серверний або

клієнтський. Серверні компоненти дозволяють визначати контент на боці сервера, що забезпечує швидшу реакцію на запити користувачів та оптимізувати файли які відправляються клієнту, залишаючи лише те що йому потрібно.

У більшості випадків це використовується для генерації динамічного контенту сторінок. Так головна сторінка має функцію `getRandomRecipes`, яка отримує дані 5 випадкових рецептів, на основі яких потім формується слайдер головної сторінки. Код цієї функції наведено у лістингу 3.21.

Лістинг 3.21 – Код функції `getRandomRecipes`

```
const getRandomRecipes = async () => {
  try {
    const response = await
    axios.get('https://api.spoonacular.com/recipes/random', {
      params: {
        number: 5,
        apiKey: process.env.SPOONACULAR_API_KEY,
      },
    });

    const recipes: Recipe[] = response.data.recipes.map((recipe: any) =>
    ({
      id: recipe.id,
      image: recipe.image.replace('-556x370', '-636x393'),
    }));

    return recipes;
  } catch (error) {
    console.error('Error fetching random recipes:', error);
    throw error;
  }
};
```

В залежності від статусу автентифікації користувача компонент `Header` рендерить кнопку для відвідування сторінки профілю користувача, або ж кнопки реєстрації та автентифікації. Для отримання цього статусу використовується функція бібліотеки `NextAuth` `getSession` в яку передається параметр `authOptions`.

Сторінка рецепту, розташована за маршрутом `app/recipe/[recipeId]/page.tsx`, використовує динамічний підхід для визначення і відображення контенту на основі унікального ідентифікатора рецепту (`recipeId`). Основна функціональність сторінки включає отримання даних про рецепт із API `spoonacular`, завантаження відгуків із бази даних, а також динамічне відображення інформації про рецепт, його

інгредієнти, інструкції та відгуки користувачів. Код серверної частини сторінки наведено у додатку А лістингу А.3.

При завантаженні сторінки, спочатку виконується функція `getServerSession`, яка отримує поточну сесію користувача, що допомагає визначити, чи авторизований користувач. Потім, використовуючи значення параметра `recipeId`, сторінка викликає функцію `getRecipeById`, яка робить запит до зовнішнього API `spoonacular` для отримання детальної інформації про рецепт. Після успішного отримання даних, інформація про рецепт, включаючи зображення, заголовок, категорії, опис, інгредієнти та інструкції з приготування, зберігається у змінній `recipeInfo`.

Наступним кроком є завантаження відгуків про рецепт з бази даних за допомогою функції `getReviewsFromBD`, яка використовує ORM `Prisma` для отримання відгуків, включаючи інформацію про користувачів, які залишили ці відгуки. Відгуки використовуються для обчислення середньої оцінки рецепту і їх відображення на сторінці.

4 ВИПРОБУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Випробування користувацького інтерфейсу

Головною метою при побудові інтерфейсу було забезпечення коректного відображення контенту сторінок на усіх сучасних пристроях. При перевірці адаптивності вебсайту буде порівняно десктопну та мобільну версію.

Десктопну та мобільну версію головної сторінки наведено на рисунках 4.1 та 4.2.

Головна сторінка складається з двох головних секцій: повноекранне вікно з рядком пошуку рецептів та слайдером із випадковими зображеннями рецептів. При адаптації даної секції під мобільні пристрої зменшено розмір основних елементів та, задля акцентування уваги користувача на рядку пошуку, зображенню на фоні

додано темний напівпрозорий фільтр.

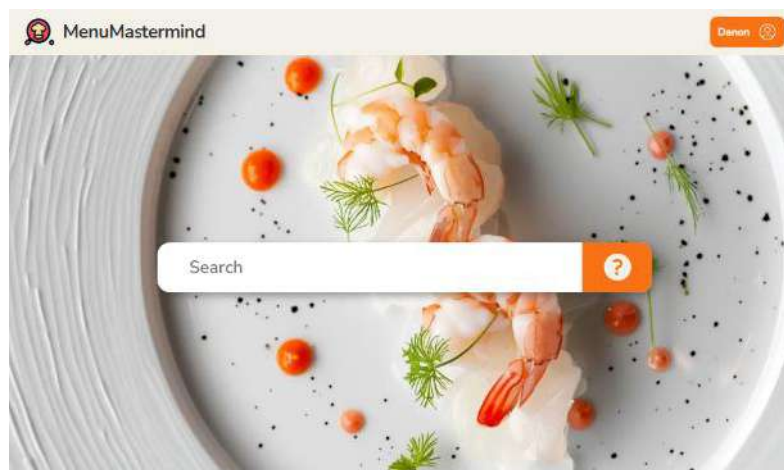


Рисунок 4.1 – Десктопна версія головної сторінки

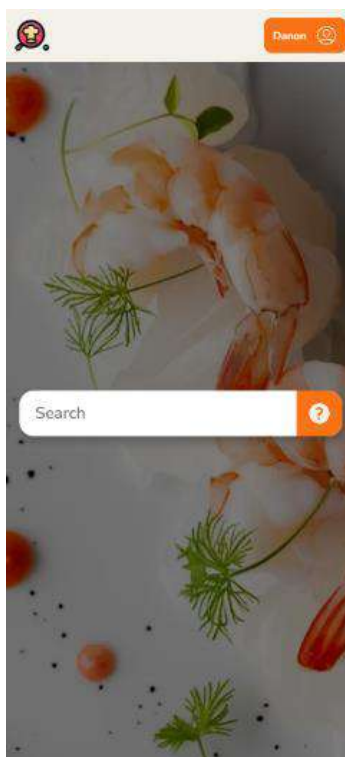


Рисунок 4.2 – Мобільна версія головної сторінки

Секція із слайдером адаптувалась наступним чином: при зменшенні екрану автоматично обирається оптимальна кількість слайдів для показу щоб задіяти максимум простору. Зображення даної секції наведено у рисунках 4.3 та 4.4.



Рисунок 4.3 – Десктопна версія слайдеру головної сторінки

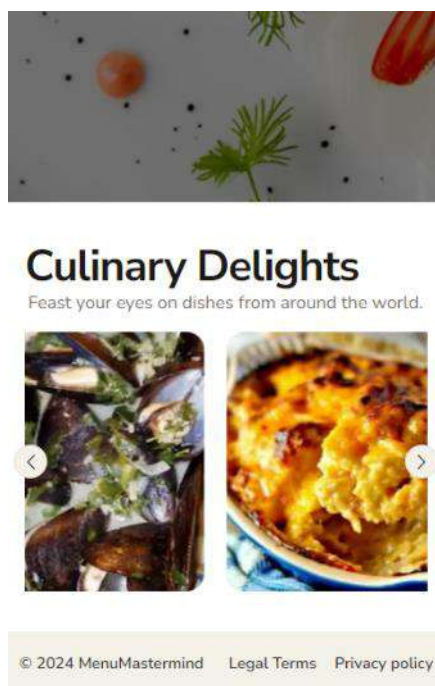


Рисунок 4.4 – Мобільна версія слайдеру головної сторінки

Адаптація сторінки пошуку рецептів при зменшенні розміру екрану відбувається за рахунок зміни розташування та розмірів елементів інтерфейсу. На десктопній версії (рис 4.5) результати пошуку відображаються у вигляді grid сітки з трьох колонок та довільної кількості рядків. На мобільній версії (рис. 4.6) кількість карток у рядку зменшується до однієї, що дозволяє зберегти читабельність та зручність користування.

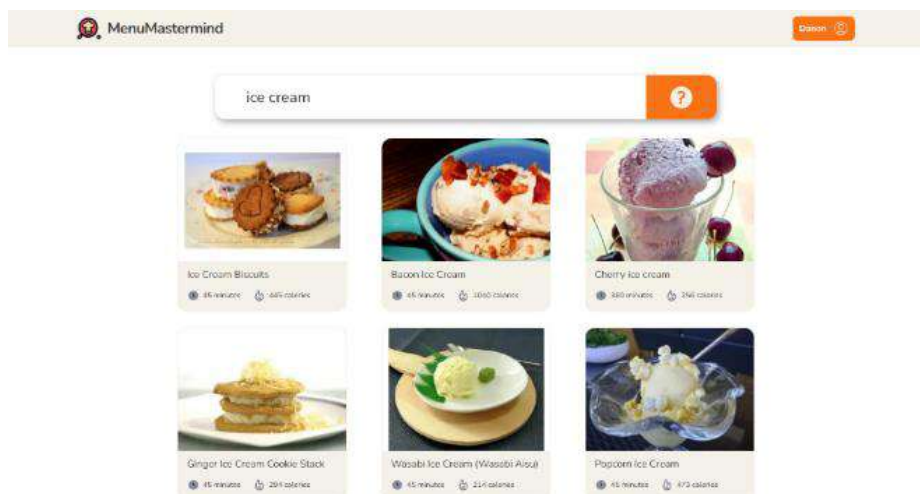


Рисунок 4.5 – Десктопна версія сторінки пошуку рецептів



Рисунок 4.6 – Мобільна версія сторінки пошуку рецептів

Адаптація сторінки рецепту при зменшенні розміру екрану відбувається через зміну розташування та розмірів елементів інтерфейсу. На десктопній версії (рис. 4.7) основні елементи розташовані горизонтально: зображення рецепту зліва, а текстова інформація праворуч.



Рисунок 4.7 – Десктопна версія сторінки рецепту

На мобільній версії (рис. 4.8) зображення рецепту розташоване зверху, а текстова інформація під ним. Це дозволяє зберегти читабельність і зручність користування навіть на невеликих екранах. Всі елементи залишаються доступними, але їх розташування стає вертикальним для забезпечення оптимального перегляду.



Рисунок 4.8 – Мобільна версія сторінки рецепту

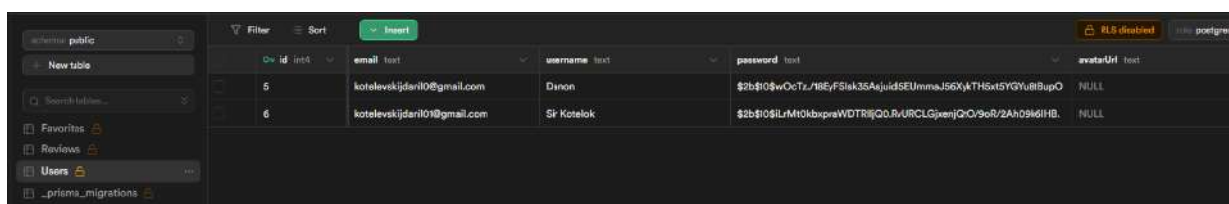
4.2 Функціональне випробування

Під час перевірки інтерфейсу головної сторінки, сторінок пошуку рецептів та власне рецепту було також перевірено їх функціональність. Тому у даному підрозділі буде опущено перевірку генерації зображень та пошукових можливостей даних сторінок. Таким чином перевірка стосуватиметься наступних функціональностей:

- реєстрації та автентифікації;
- відображення існуючих відгуків та додавання нових;
- додавання рецепта до списку улюблених та перевірка цього на сторінці профілю.

Процес перевірки реєстрації та автентифікації розрочато з введення даних у форму реєстрації. Введено ім'я користувача “Danon”, електронну пошту “kotelevskijdanil0@gmail.com” та пароль “helloworld”. Надіслано запит на реєстрацію.

Після успішного надсилання запиту перевірено їх присутність у базі даних. Зображення створеного запису в базі даних наведено на рисунку 4.9. Як можна побачити, пароль користувача зберігся у шифрованому виді.



id	email	username	password	avatarUrl
5	kotelevskijdanil0@gmail.com	Danon	\$2b\$10\$wOcTz/18E/F5iak35Aajuid5EUmmaJ56XyKTH5xt5YGy8BupO	NULL
6	kotelevskijdanil0@gmail.com	Sir Kotelok	\$2b\$10\$ILrMtOkbpnraWDTRIjQ0.R/URCLGjxvjQO/9eR/2Ah09k6IHb.	NULL

Рисунок 4.9 – Створений запис користувача у базі даних

Введено дані з реєстрації у форму автентифікації. Після натискання кнопки “Sign in” відбулася успішна автентифікація, про що свідчить автоматична зміна сторінки на головну та присутнє посилання на сторінку профілю користувача. Зображення головної сторінки користувача, що пройшов автентифікацію зображено на рисунку 4.10.

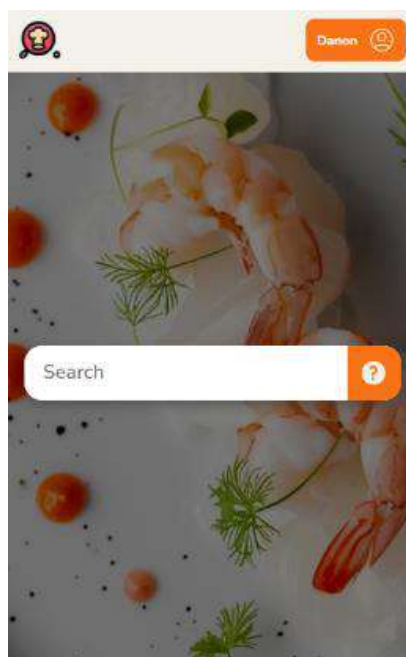


Рисунок 4.10 – Успішне проходження автентифікації користувачем

Таким чином, було підтверджено коректну роботу системи реєстрації та автентифікації, яка забезпечує правильне додавання нових користувачів та їх успішний вхід до системи.

Для перевірки функціональності списку улюблених рецептів було виконано наступні кроки. Спочатку до списку улюблених додано три випадкових рецепта та перевірено їх наявність у відповідній таблиці бази даних. Виконано перехід на сторінку профілю та відкрито вкладку “favorites”. На цій вкладці відобразилися всі три додані рецепти, підтверджуючи, що система коректно зберігає та відображає улюблені рецепти користувача. Зображення записів у базі даних та вкладки “favorites” наведено на рисунках 4.11 та 4.12.

schema: public	Filter	Sort	Insert	
+ New table				
Search tables...				
Favorites				
Reviews				
Users				
_prisma_migrations				

	id int4	userid int4	recipeld int4	+
☐	3	5	664959	
☐	4	5	756817	
☐	6	5	633314	

Рисунок 4.11 – Записи таблиці “favorites” бази даних

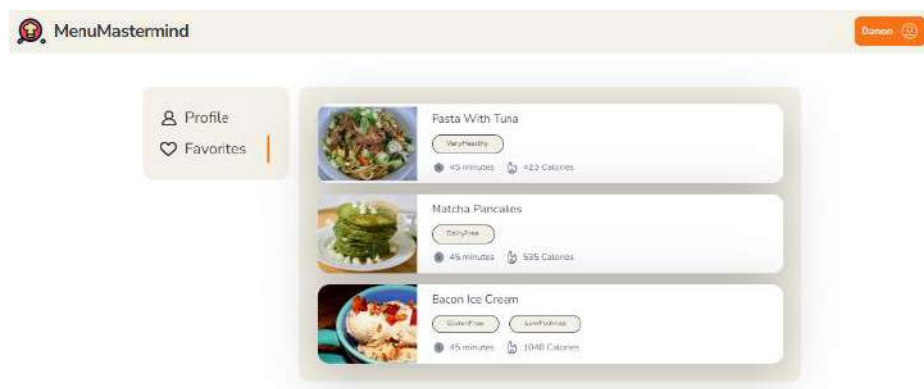


Рисунок 4.12 – Вкладка “favorites” у профілі користувача

Для перевірки функціональності відображення відгуків спочатку здійснено перехід на сторінку з рецептом морозива. На даній сторінці перевірено відсутність відгуків.

Далі залишено новий відгук для цього рецепту із тестовим текстом та виставлено оцінку 3 зірки. Відгук надіслано та перевірено оновлену сторінку. Зображення оновленої сторінки наведено на рисунку 4.13.

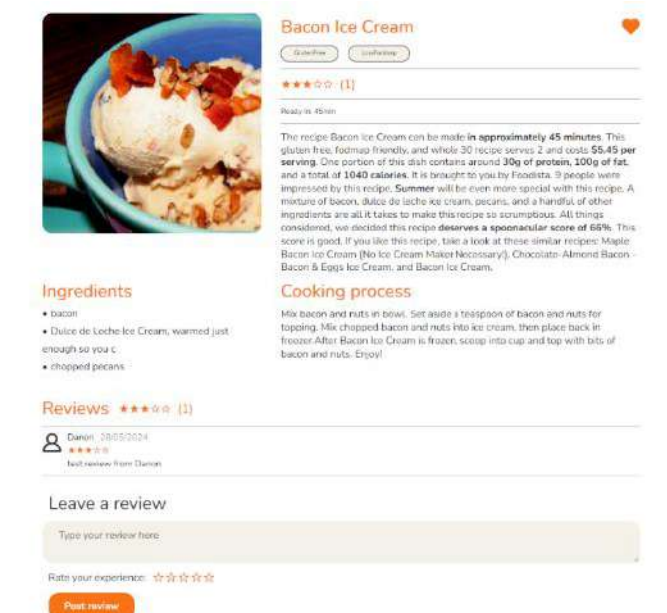


Рисунок 4.13 – Оновлена сторінка рецепту з відгуком

ВИСНОВКИ

В результаті дипломної роботи виконано проектування програмного забезпечення для вебсайту з пошуку рецептів. Проведено аналіз предметної області, сформовано вимоги до ПЗ, за якими було спроектовано систему.

Протягом виконання роботи проаналізовано аналогічні системи, сформовано специфікацію вимог, спроектовано архітектуру додатку з базою даних.

У результаті виконання дипломної роботи спроектовано та реалізовано програмне забезпечення для вебсайту з пошуку рецептів, яке забезпечує зручний інтерфейс для користувачів, дозволяє реєструватися та автентифікуватися, залишати відгуки, додавати рецепти до списку улюблених та переглядати збережені рецепти у профілі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Software Architecture - Definition. URL: <https://www.techopedia.com/definition/24596/software-architecture> (дата звернення: 24.04.2024);
2. Our Work: Software Architecture. URL: <https://www.sei.cmu.edu/our-work/software-architecture/> (дата звернення: 24.04.2024);
3. Monolithic vs Microservices Architecture. URL: <https://www.lambdatest.com/blog/monolithic-vs-microservices-architecture/> (дата звернення: 24.04.2024);
4. Microservices: Advantages and Disadvantages. URL: <https://www.orientsoftware.com/blog/microservices-advantages-and-disadvantages/> (дата звернення: 24.04.2024);
5. What is Client-Server Architecture? URL: <https://www.simplilearn.com/what-is-client-server-architecture-article> (дата звернення: 24.04.2024);
6. Why Use Serverless? URL: <https://www.cloudflare.com/learning/serverless/why-use-serverless/> (дата звернення: 24.04.2024);
7. ReactJS - Introduction. URL: <https://www.geeksforgeeks.org/reactjs-introduction/> (дата звернення: 05.05.2024);
8. What does React.js do? URL: <https://blog.hubspot.com/website/react-js#> (дата звернення: 05.05.2024);
9. What is Angular? URL: <https://angular.io/guide/what-is-angular> (дата звернення: 05.05.2024);
10. How does Angular compare to other frameworks. URL: <https://www.sanity.io/glossary/angular> (дата звернення: 05.05.2024);
11. Vue.js - The Progressive JavaScript Framework. URL: <https://vuejs.org/guide/introduction> (дата звернення: 05.05.2024);

12. Django. URL: <https://www.djangoproject.com/> (дата звернення: 24.04.2024);
13. Що таке Laravel? URL: <https://probotiuk.com/blog/sho-take-laravel> (дата звернення: 05.05.2024);
14. Documentation - Next.js. URL: <https://nextjs.org/docs> (дата звернення: 24.04.2024);
15. Nuxt.js - Glossary. URL: <https://www.sanity.io/glossary/nuxt> (дата звернення: 05.05.2024);
16. Relational Model in DBMS. URL: <https://www.geeksforgeeks.org/relational-model-in-dbms/> (дата звернення: 05.05.2024);
17. Introduction to NoSQL. URL: <https://www.geeksforgeeks.org/introduction-to-nosql/> (дата звернення: 05.05.2024);
18. Що таке база даних? URL: <https://apeps.kpi.ua/shcho-take-basa-danykh> (дата звернення: 05.05.2024);
19. Наукова публікація про популярність СУБД Oracle. URL: <https://javarush.com/ua/groups/posts/uk.251.ndeks-pypl-naypopuljarnsha-subd---oracle-a-firebase-roste-ljakajuche-shvidko> (дата звернення: 05.05.2024);
20. Oracle SQL Developer: Overview. URL: <https://www.oracle.com/database/sqldeveloper/> (дата звернення: 24.05.2024);
21. MySQL Workbench: Product Overview. URL: <https://www.mysql.com/products/workbench/> (дата звернення: 24.05.2024);
22. What is SQL Server? URL: <https://www.techtarget.com/searchdatamanagement/definition/SQL-Server> (дата звернення: 24.05.2024);
23. What is PostgreSQL? URL: <https://foxminded.ua/postgresql-shcho-tse/> (дата звернення: 24.05.2024);
24. UML Use Case Diagram Tutorial. URL: <https://www.lucidchart.com/pages/uml-use-case-diagram> (дата звернення: 24.05.2024);

25. UML Sequence Diagram Tutorial. URL: <https://www.lucidchart.com/pages/uml-sequence-diagram> (дата звернення: 24.05.2024);

26. Next.js integrating with Sass URL: <https://nextjs.org/docs/app/building-your-application/styling/sass> (дата звернення: 24.05.2024);

27. Using CSS custom properties (variables) URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties (дата звернення: 24.05.2024);

28. How to use Sass Variables URL: https://www.w3schools.com/sass/sass_variables.php (дата звернення: 24.05.2024);

29. Configure your Next.js project to import SVG as React components in your application. URL: <https://react-svgr.com/docs/next/> (дата звернення: 24.05.2024);

30. Start a fresh project from scratch URL: <https://www.prisma.io/docs/getting-started/setup-prisma/start-from-scratch> (дата звернення: 24.05.2024);

31. Supabase Documentation URL: <https://supabase.com/docs> (дата звернення: 24.05.2024);

32. Clamp Based CSS Property Generator for Fluid Typography URL: <https://observablehq.com/@saneef/fluid-typography-css-clamp-value-generator> (дата звернення: 24.05.2024);

ДОДАТОК А

ОСНОВНІ КОДИ ПРОГРАМНИХ МОДУЛІВ

Лістинг А.1 – Код компоненту форми реєстрації користувача

```
'use client';
import Image from 'next/image';
import React, { useRef, useState } from 'react';
import Input from './Input';
import s from './form.module.scss';
import { useRouter } from 'next/navigation';

interface InputState {
  label: string;
  showError: boolean;
}

interface InputStates {
  username: InputState;
  email: InputState;
  password: InputState;
}

const SignUpForm = () => {
  const router = useRouter();

  const defaultInputsState = {
    username: { label: 'Nickname', showError: false },
    email: { label: 'Email', showError: false },
    password: { label: 'Password', showError: false },
  };

  const [inputStates, setInputStates] =
    useState<InputStates>(defaultInputsState);
  const [showServerErrorMsg, setShowServerErrorMsg] = useState<boolean>(false);

  const nameRef = useRef<HTMLInputElement>(null);
  const emailRef = useRef<HTMLInputElement>(null);
  const passwordRef = useRef<HTMLInputElement>(null);

  const handleError = (message: string) => {
    const errorStates: { [key: string]: InputStates } = {
      'Email already exist': {
        ...defaultInputsState,
        email: { label: 'This email address is already in use', showError:
true },
      },
      'Username already exist': {
        ...defaultInputsState,
        username: { label: 'This username is already in use', showError:
true },
      },
    };

    setInputStates(errorStates[message] || defaultInputsState);
    if (!errorStates[message]) {
      setShowServerErrorMsg(true);
    }
  };
};
```

```

};

return (
  <form
    className={s.form}
    onSubmit={async e => {
      e.preventDefault();
      setShowServerErrorMsg(false);

      const response = await fetch('/api/user/', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          username: nameRef.current?.value,
          email: emailRef.current?.value,
          password: passwordRef.current?.value,
        }),
      });

      if (response.ok) {
        const data = await response.json();

        if (data.success) {
          setInputStates(defaultInputsState);
          router.push('/sign-in');
        } else {
          handleError(data.message);
        }
      } else {
        setInputStates(defaultInputsState);
        setShowServerErrorMsg(true);
      }
    }}
  >
    <div className={s.form__title}>Sign up</div>

    <button type='button' onClick={() => console.log('Google')}
      className={s.googleButton}>
      <Image src={'/google-icon.svg'} height={24} width={24}
        alt='google-img' />
      <span>Continue with Google</span>
    </button>

    <div className={s.form__dividingLine}>
      <span></span>
      or
      <span></span>
    </div>

    <div className={s.form__fields}>
      <Input
        ref={nameRef}
        type='text'
        name='text'
        pattern='^(?!\\s*$).{3,}$'
        placeholder='Enter your nickname'
        isRequired={true}
        label={inputStates.username.label}
        showError={inputStates.username.showError}
      />

      <Input

```

```

        ref={emailRef}
        type='email'
        name='email'
        pattern='^[a-zA-Z0-9]+(?:\\.[a-zA-Z0-9]+)*@[a-zA-Z0-9]+(?:\\.[a-
zA-Z0-9]+)*$'
        placeholder='Enter your email'
        isRequired={true}
        label={inputStates.email.label}
        showError={inputStates.email.showError}
      />

      <Input
        ref={passwordRef}
        type='password'
        name='password'
        placeholder='Create Password'
        pattern='.{6,}'
        isRequired={true}
        label={inputStates.password.label}
        showError={inputStates.password.showError}
      />

      <button type='submit' className={s.formSubmitButton}>
        Signup with Email
      </button>

      {showServerErrorMsg && <p className={s.serverErrorMsg}>Server
error occurred, try again later</p>}
    </div>
  </form>
);
};

export default SignUpForm;

```

Лістинг А.2 – Код клієнтської частини сторінки профілю

```

'use client';
import Image from 'next/image';
import React, { useEffect, useState } from 'react';
import s from './page.module.scss';
import RecipeCardFavorite from '@components/recipe-card/RecipeCardFavorite';
import Button from '@components/button/Button';
import { useRouter } from 'next/navigation';
import { signOut } from 'next-auth/react';

const tabsList = ['profileData', 'favorites'];

const ProfilePage = ({ tab, favorites }: { tab: string; favorites: any[] }) => {
  const router = useRouter();
  const [activeTabIndex, setActiveTabIndex] = useState<number>(0);

  useEffect(() => {
    if (tab === 'favorites') {
      setActiveTabIndex(1);
    } else if (tab === 'profileData') {
      setActiveTabIndex(0);
    }

    console.log(favorites);
  }, [tab, favorites]);

```

```

}, [tab]);

const handleTabClick = (index: number, tabName: string) => {
  setActiveTabIndex(index);
  router.push(`/profile/${tabName}`);
};

return (
  <main className={s.profilePage}>
    <div className={s.profilePage__container}>
      <div className={s.tabs}>
        <a
          className={` ${s.tabs__link} ${activeTabIndex === 0 ?
s.active : ''}`}
          onClick={e => {
            e.preventDefault();
            handleTabClick(0, 'profileData');
          }}
        >
          <Image src='/profile.svg' height={23} width={23}
alt='profile-img' />
          <p>Profile</p>
        </a>
        <a
          className={` ${s.tabs__link} ${activeTabIndex === 1 ?
s.active : ''}`}
          onClick={e => {
            e.preventDefault();
            handleTabClick(1, 'favorites');
          }}
        >
          <Image src='/favorite.svg' height={23} width={23}
alt='favorites-img' />
          <p>Favorites</p>
        </a>
      </div>

      <div className={s.tabContentContainer}>
        <div id='tab1' className={` ${s.tabContentContainer__tab}
${activeTabIndex === 0 ? s.active : ''}`>
          <div className={s.profileBlock}>
            <h3 className={s.profileBlock__title}>You can edit
your profile data here</h3>

            <div className={s.profileBlock__fields}>
              <label className={s.input}>
                <p>Name</p>
                <input type='text' name='name'
defaultValue={'SirKotelok'} />
              </label>
              <label className={s.input}>
                <p>Email</p>
                <input type='email' name='name'
defaultValue={'kot@gmail.com'} />
              </label>
            </div>

            <a href='#' className={s.changePassword}>
              Change Password
            </a>

            <Button
              onClick={() => {
                signOut({

```

```

                                redirect: true,
                                callbackUrl: window.location.origin,
                                });
                                }}
                                >
                                Log out
                                </Button>
                                </div>
                                </div>

                                <div id='tab2' className={` ${s.tabContentContainer__tab}
                                ${activeTabIndex === 1 ? s.active : ''}`}>
                                    {favorites.length > 0 ? (
                                        favorites.map(favorite => (
                                            <RecipeCardFavorite
                                                key={favorite.id}
                                                imageUrl={favorite.image}
                                                recipeId={favorite.id}
                                                title={favorite.title}
                                                category={favorite.categories}
                                                time={` ${favorite.readyInMinutes} minutes`}
                                                calories={` ${favorite.calories} Calories`}
                                            />
                                        ))
                                    ) : (
                                        <p>Your favorites-list is empty</p>
                                    )}
                                </div>
                                </div>
                                </div>
                                </main>
                                );
                                };

                                export default ProfilePage;

```

Лістинг А.3 – Код серверної частини сторінки рецепту

```

const session = await getServerSession(authOptions);

const recipeId = params.recipeId;

const getRecipeById = async (id: number): Promise<Recipe> => {
    try {
        const response = await
        axios.get(`https://api.spoonacular.com/recipes/${id}/information`, {
            params: {
                apiKey: process.env.SPOONACULAR_API_KEY,
            },
        });

        const data = response.data;

        const ingredients: Ingredient[] =
        data.extendedIngredients.map((ingredient: any) => ({
            name: ingredient.originalName,
            amount: ingredient.amount,
            unit: ingredient.unit,
        }));
    }

```

```

        const instructions: string =
data.analyzedInstructions[0]?.steps.map((step: any) => step.step).join(' ') || '';

        const recipe: Recipe = {
            image: data.image,
            title: data.title,
            categories: categoriesKeysToCheck.filter(key => data[key] ===
true),
            summary: data.summary,
            readyInMinutes: data.readyInMinutes,
            preparationMinutes: data.preparationMinutes || 0,
            cookingMinutes: data.cookingMinutes || 0,
            ingredients,
            instructions,
        };

        return recipe;
    } catch (error) {
        console.error('Error fetching recipe:', error);
        throw error;
    }
};

const recipeInfo = await getRecipeById(recipeId);

async function getReviewsFromBD(recipeId: number) {
    const reviews = await prisma.reviews.findMany({
        where: {
            recipeId: recipeId,
        },
        include: {
            user: {
                select: {
                    username: true,
                },
            },
        },
    });

    return reviews.map(review => ({
        id: review.id,
        userId: review.userId,
        date: review.date,
        text: review.text,
        rating: review.rating,
        recipeId: review.recipeId,
        username: review.user.username,
    }));
}

function ratingToString(rating: number) {
    if (rating === 0) return '☆☆☆☆☆';
    const roundedRating = Math.round(rating);
    const filledStars = '★'.repeat(roundedRating);
    const emptyStars = '☆'.repeat(5 - roundedRating);
    return filledStars + emptyStars;
}

const reviews = await getReviewsFromBD(Math.trunc(recipeId));

function getAverageRating(reviews: Review[]): number {
    if (reviews.length === 0) return 0;

```

```

    const totalRating = reviews.reduce((sum, review) => sum + review.rating,
0);
    const averageRating = totalRating / reviews.length;

    return averageRating;
}

let stringifiedRating: string;

if (reviews.length > 0) {
    const averageRating = getAverageRating(reviews);
    stringifiedRating = ratingToString(averageRating);
} else {
    stringifiedRating = '☆☆☆☆';
}

const reviewsNumber = reviews.length;

function formatDate(dateString: Date) {
    const date = new Date(dateString);
    return new Intl.DateTimeFormat('en-GB', {
        day: '2-digit',
        month: '2-digit',
        year: 'numeric',
    }).format(date);
}

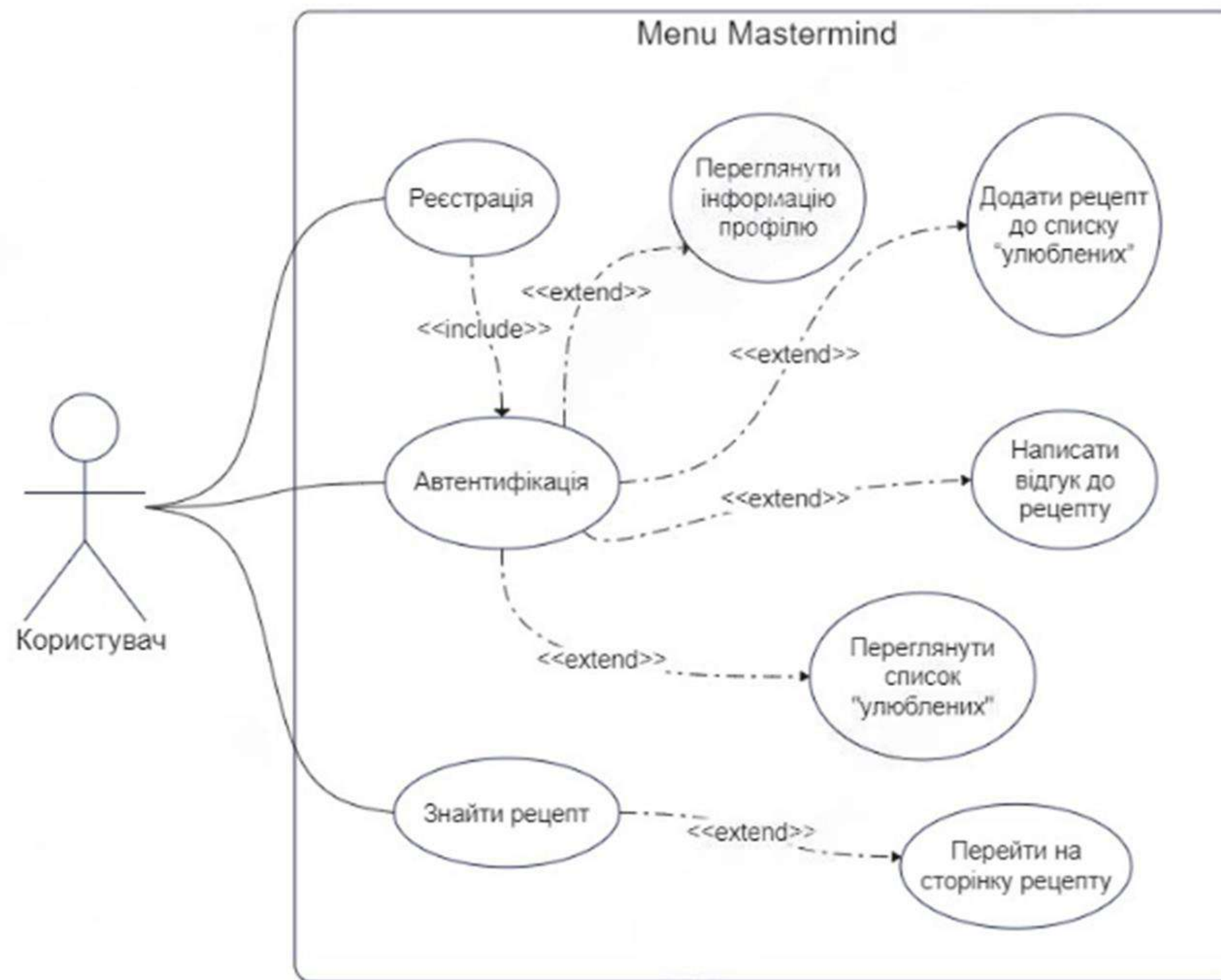
let FavoriteButtonBlock = null;

if (session?.user.username) {
    const user = await prisma.users.findFirst({
        where: { username: session?.user.username },
    });

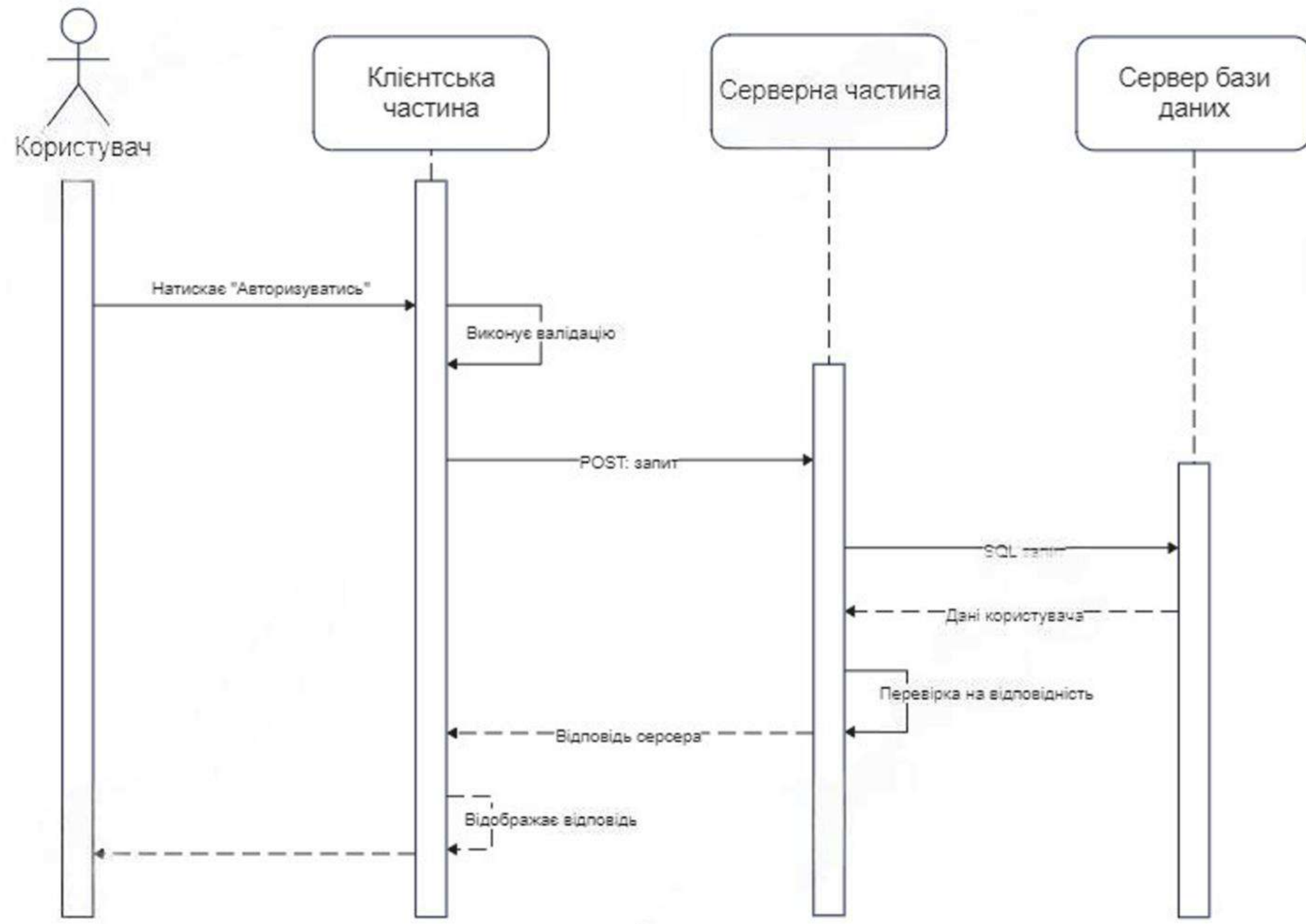
    const favorite = await prisma.favorites.findFirst({
        where: {
            userId: user?.id,
            recipeId: Math.trunc(recipeId),
        },
    });

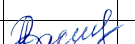
    FavoriteButtonBlock = (
        <FavoriteButton recipeId={recipeId} username={session.user.username}
defaultIsActive={!!favorite} />
    );
}

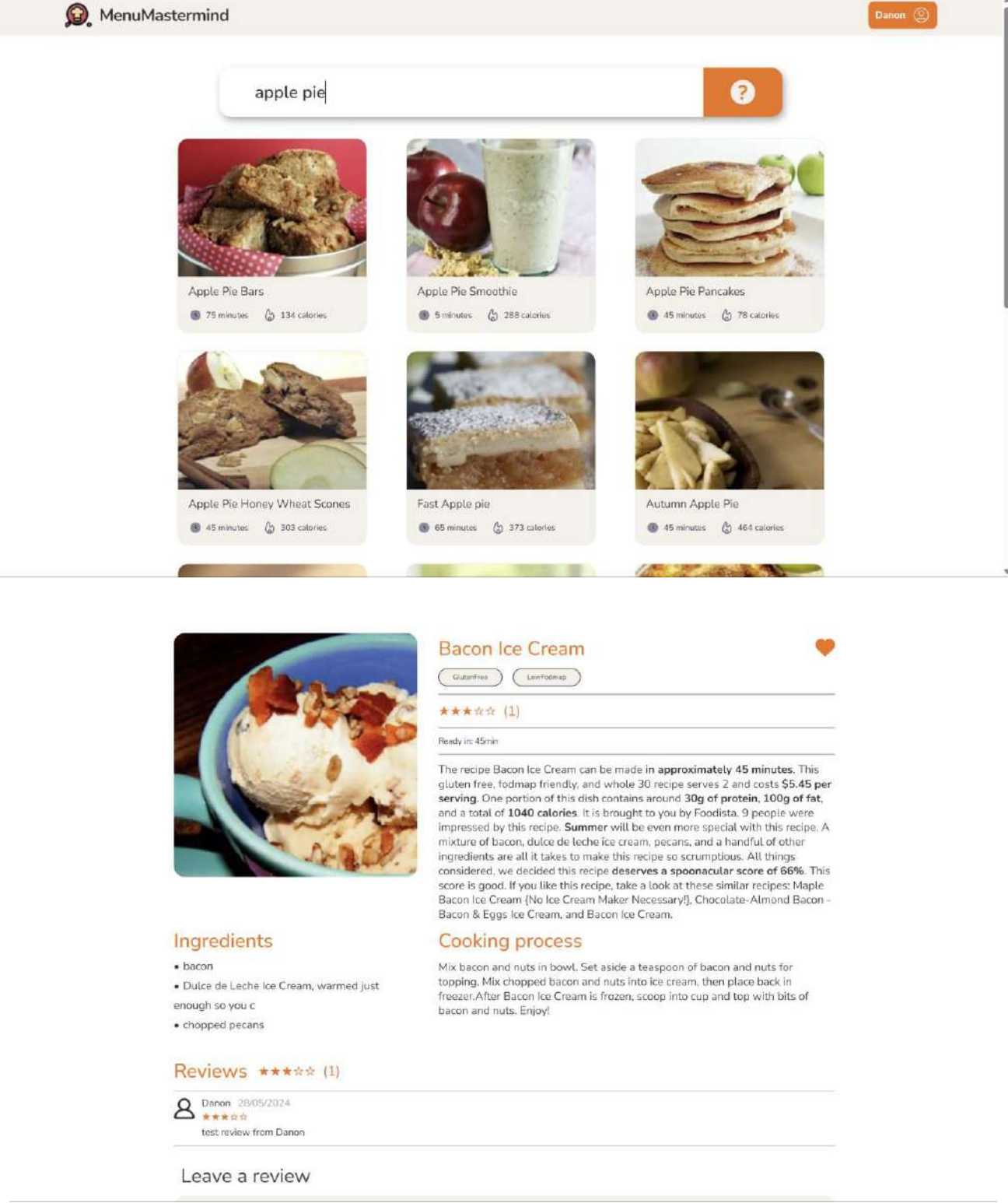
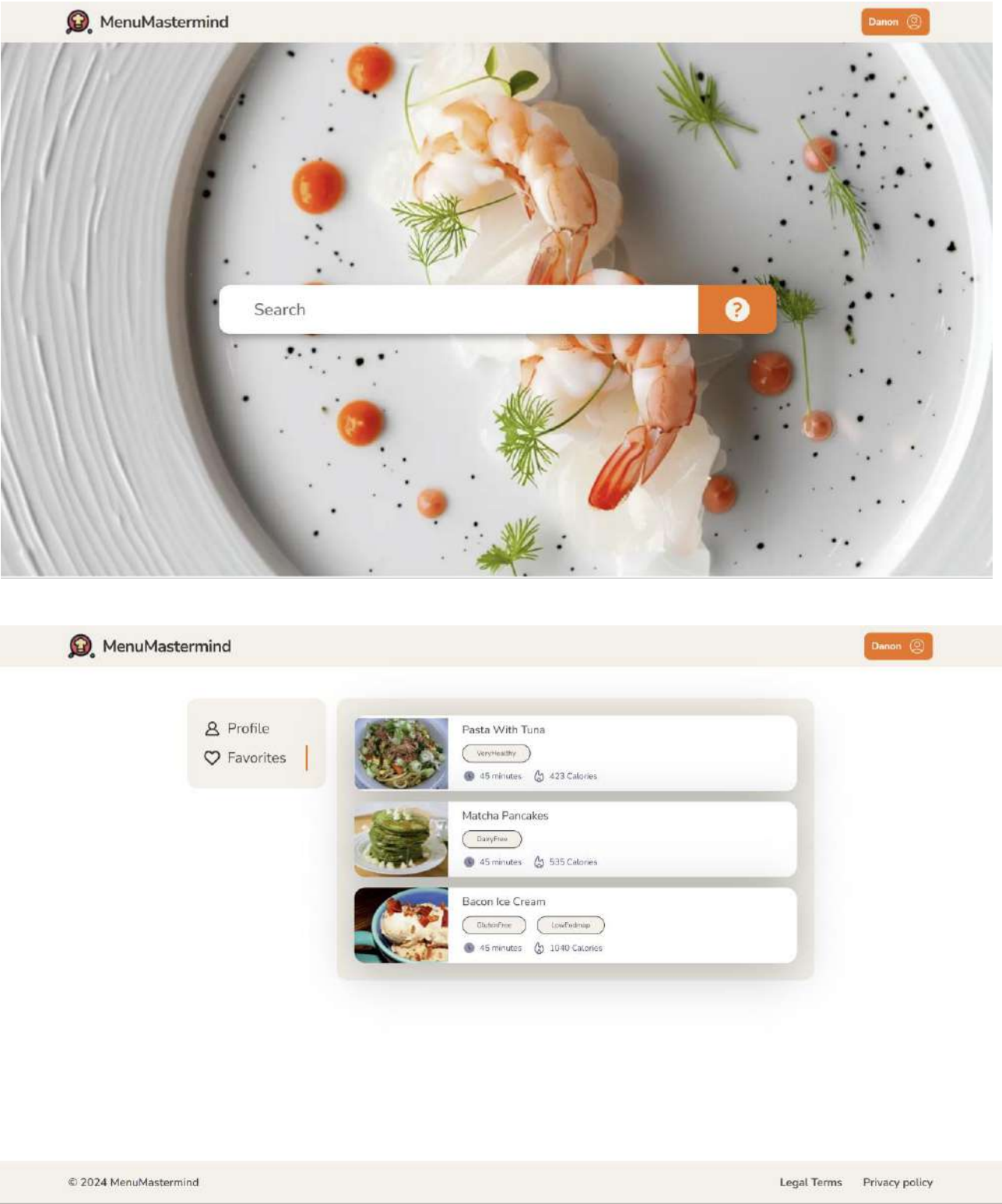
```

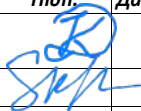
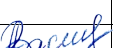
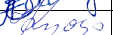


					13.02070849.00042 ПЛ1			
					Розробка комп'ютерної системи з пошуку рецептів			
Зам.	Лист.	№ докум.	Підп.	Дата				
Розроб.		Котелевський Д.С.						
Перев.		Скрупський С.Ю.						
Т.контр.								
Н.контр.		Польська О.В.			Діаграма використання			
Затв.		Кудерметов Р.К.						
						Лист 1	Листів 1	
					НУ «Запорізька політехніка» КНТ-520			



					13.02070849.00042 ПЛ2													
					Розробка комп'ютерної системи з пошуку рецептів						Лист.		Маса		Масшт.			
Зам.	Лист.	№ докум.	Підп.	Дата														
Розроб.		Котелевський Д.С.																
Перев.		Скрупський С.Ю.																
Т.контр.																		
											Лист 1		Листів 1					
Н.контр.		Польська О.В.			Діаграма послідовності обробки запиту на автентифікацію						НУ «Запорізька політехніка» КНТ-520							
Затв.		Кудерметов Р.К.																



					13.02070849.00042 ПЛЗ											
					Розробка комп'ютерної системи з пошуку рецептів					Лист.		Маса		Масшт.		
Зам.	Лист.	№ докум.	Підп.	Дата						Лист 1		Листів 1				
Розроб.		Котелевський Д.С.														
Перев.		Скрупський С.Ю														
Т.контр.																
Н.контр.		Польська О.В.														
Затв.		Кудерметов Р.К.			Інтерфейс вебсайту					НУ «Запорізька політехніка» КНТ-520						