МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система керування навчанням кондитерській справі»
здобувачки групи ІН-92 Войтенко Каріни Костянтинівни

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

_____ Каріна ВОЙТЕНКО

Керівник,
доцент кафедри комп'ютерних наук,
к.т.н Валентина БОРОВИК _____

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувачки групи ІН-92 Войтенко Каріни Костянтинівни

1. Тема роботи: «Інформаційна система керування навчанням кондитерській справі»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інтернет магазину. 3) Розробка інформаційної системи керування навчанням кондитерській справі. 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	13.04 - 16.04	
2	*Огляд технологій, що використовуються для розробки інформаційної системи*	16.04 - 18.04	
3	*Розробка інформаційної системи керування навчанням навчанням кондитерській справі*	18.04 - 29.05	
4	*Аналіз отриманих результатів*	30.05 - 01.06	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	01.06 - 08.06	

Здобувач вищої освіти _____ Керівник _____

АНОТАЦІЯ

Записка: 62 стр., 34 рис., 4 додатки, 15 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розробці інформаційної системи керування навчанням кондитерській справі, що надасть зручну та ефективну платформу для студентів, зацікавлених у вивченні та покращенні своїх навичок у цій галузі.

Об’єкт дослідження — навчальна платформа для кондитерів.

Мета роботи — розробка інформаційної системи керування навчанням кондитерській справі.

Методи дослідження — алгоритми для навчання кондитерським справам та для створення веб-додатків з обраними інструментами для побудови інформаційних систем.

Результати — розроблена інформаційна система керування навчанням кондитерській справі, яка відповідатиме потребам студентів та викладачів, забезпечуватиме зручну навігацію, ефективну співпрацю та надійне збереження інформації.

ІНФОРМАЦІЙНА СИСТЕМА, КОНДИТЕРСЬКА СПРАВА, ОНЛАЙН НАВЧАННЯ, MERN, JAVA SCRIPT.

ЗМІСТ

ВСТУП

Актуальність. Дана кваліфікаційна робота є актуальною, оскільки дає можливість навчатися кондитерській справі онлайн з будь де і будь коли.

Об’єкт дослідження. Навчальна платформа для кондитерів.

Предмет дослідження. Методологія розробки зручної інформаційної системи навчання керуванням кондитерській справі.

Гіпотеза. Швидке навчання кондитерській справі з будь якої точки світу можна досягти шляхом використання інформаційної технології, що реалізує веб-додаток для розміщення відео-уроків та текстових рецептів-секретів.

Новизна. На відміну від подібних рішень дана інформаційна система дає можливість глибоко та детально вивчити саме кондитерську справу у всіх її розгалуженнях, шляхом детального аналізу та вивчення і реалізації навчальних уроків

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан

У сучасному світі, в час війни, багато хто був вимушений покинути свій дім, рідне місто чи навіть країну, але попри все життя продовжується та розвивається, так само як і всі галузі.

Для нас студентів розвиток технологій і цифрових рішень перетворив спосіб навчання на більш зручний та мобільний, адже неважливо де ти знаходишся, маючи з собою пристрій та доступ до інтернету, можна без проблем виконувати завдання та залишатися на зв’язку з викладачами та студентами. Інтернет та електронні платформи стали важливими інструментами для доступу до навчальних ресурсів у всіх галузях, спілкування з експертами та розвитку професійних навичок. У цьому контексті кондитерська справа, що вимагає високого рівня майстерності та креативності, також потребує сучасного підходу до навчання та обміну знаннями.

Не кожен має можливість прийти у кондитерську школу та навчатись офлайн через різні на те причини. Саме тому правильним рішенням цієї проблеми є онлайн ресурси для навчання кондитерській справі. У будь який час можна зареєструватися на обраний курс та почати навчання онлайн. Не потрібно їхати у інше місто, щоб отримати знання та мати змогу поспілкуватися з викладачем, завдяки інтернету та новим технологіям все можна зробити з будь-якого пристрою, де є підключення до мережі.

1.2 Аналіз аналогічних проєктів

У процесі пошуку та аналізу подібних сервісів було знайдено, насправді мала кількість необхідних рішень, бо даний вид послуг ще не був реалізований та виданий у світ у повному обсязі жодним із конкурентів.

Найбільш з широко розкритим функціоналом є веб-сервіс *“Школа крафтових та кондитерських виробників”* (рис. 1.1) , але цей сервіс спеціалізується не лише в кондитерській справі, а і в виробництві крафтових хлібо-булочних виробів. В ньому можна ознайомитися з інформацією про школу, навчальні матеріали, навчальні модулі, переглянути викладачів.

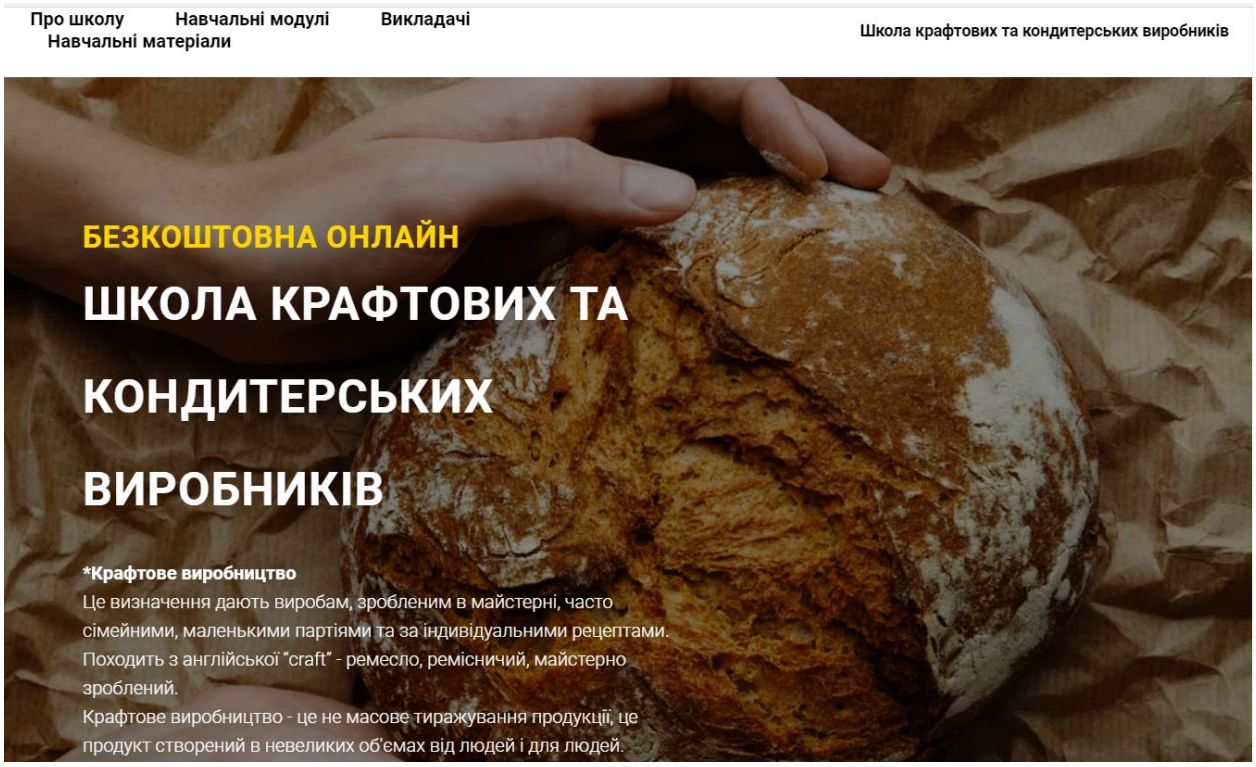

Рисунок 1.1 – Головна сторінка сайту

«Кулінарна академія Євгена Чернухи» (рис 1.2) - це онлайн веб-сервіс, який також спеціалізується на загальній кулінарії та має розділ з кондитерських курсів. Наявні розділи: курси, майстер-класи, корпоративи, про школу.

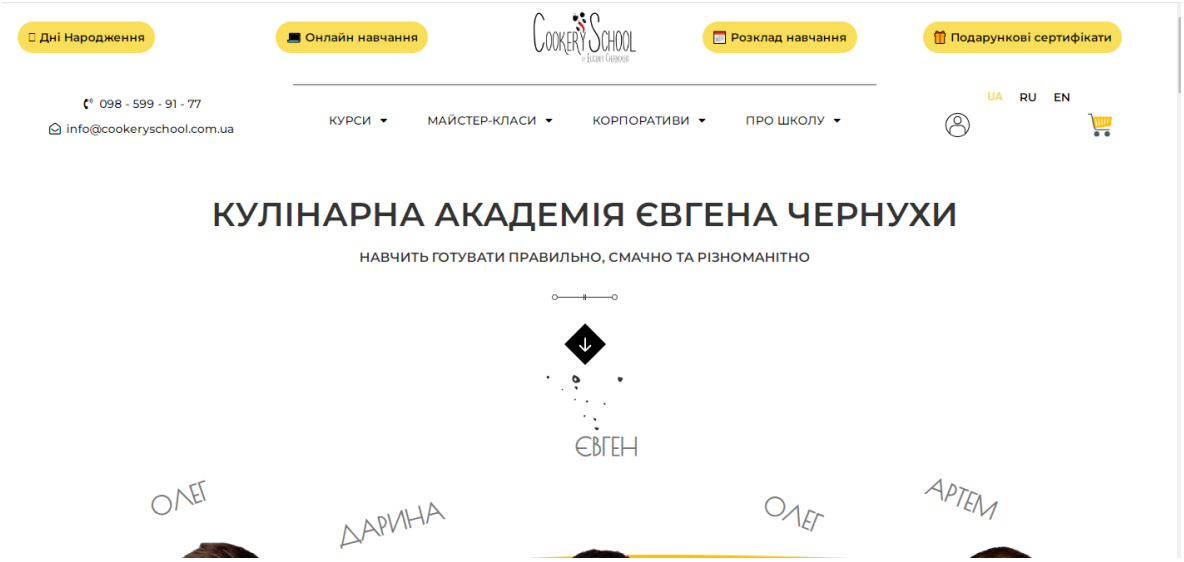

Рисунок 1.2 – Головна сторінка сайту cookeryschool.com.ua

У таблиці 1.1 надано огляд переваг, недоліків та особливостей інформаційної системи керування навчанням кондитерській справі. Вона підкреслює легкий доступ до відеоуроків, можливість завантаження контенту, різні ролі користувачів та зручний інтерфейс, але також зазначає потребу у стабільному Інтернет-підключенні та необхідність актуалізації контенту.

Таблиця 1.1 – Плюси, мінуси та особливості інформаційної системи керування навчанням кондитерській справі (сайту)

Плюси	Мінуси	Особливості
Легкий доступ до відеоуроків	Потребує стабільне підключення до Інтернету	Можливість відстеження прогресу навчання та оцінювання досягнень
Можливість завантаження контенту	Потенційна залежність від серверної інфраструктури	Авторизація для контролю доступу до вмісту
Різні ролі користувачів	Потребує контролю якості та актуалізації контенту	Функція коментування та обміну порадами між користувачами

Зручний інтерфейс	Потребує захисту від несанкціонованого доступу	Можливість стеження за прогресом навчання та ведення власного профілю
	Вимагає викладачів для створення відеоуроків	Можливість розширення функціональності та оновлення вмісту залежно від потреб користувачів

1.3 Розробка, процес створення та інструменти

Метою роботи є створення високоякісної платформи керування навчанням кондитерської справи та її майбутнього розширення.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. **Фаза ідеї та планування:**

- Визначення мети та цілей проєкту.
- Аналіз цільової аудиторії та їх потреб.
- Збір вимог і функціональних вимог до сайту.
- Розробка концепції та дизайну сайту.
- Встановлення бюджету та ресурсів.

2. **Фаза розробки:**

- Вибір технологічного стеку, такого як: мови програмування, фреймворки, бази даних тощо.
- Розробка структури бази даних та схеми сайту.
- Розробка функціональних модулів та компонентів сайту.
- Розробка користувацького інтерфейсу (UI) та користувацького досвіду (UX).
- Інтеграція зовнішніх сервісів та API, якщо необхідно.
- Розробка адміністративної панелі для управління контентом сайту.
- Тестування та усунення помилок (debugging).

- Оптимізація швидкодії та безпеки сайту.

3. Фаза випробування:

- Внутрішнє тестування всіх функцій та модулів сайту.
- Виявлення та виправлення помилок, які знайдені під час тестування.
- Тестування сайту в реальних умовах з малим колом користувачів (beta-тестування).

4. Фаза впровадження:

- Підготовка серверного середовища для розгортання сайту.
- Розгортання коду та бази даних на сервері.
- Налаштування доменного імені та SSL-сертифікату.
- Тестування сайту в реальному середовищі та вирішення проблем, які можуть виникнути.
- Міграція даних з тестового середовища на живий сайт.
- Налаштування аналітики та відстеження показників продуктивності сайту.

5. Фаза підтримки та розвитку:

- Моніторинг роботи сайту та виявлення проблем.
- Регулярне вдосконалення та оновлення функцій сайту.
- Забезпечення безпеки сайту та виявлення потенційних загроз.
- Відповідь на звернення користувачів, підтримка та вирішення проблем.

Цей план є загальним орієнтиром для подальшої розробки проєкту.

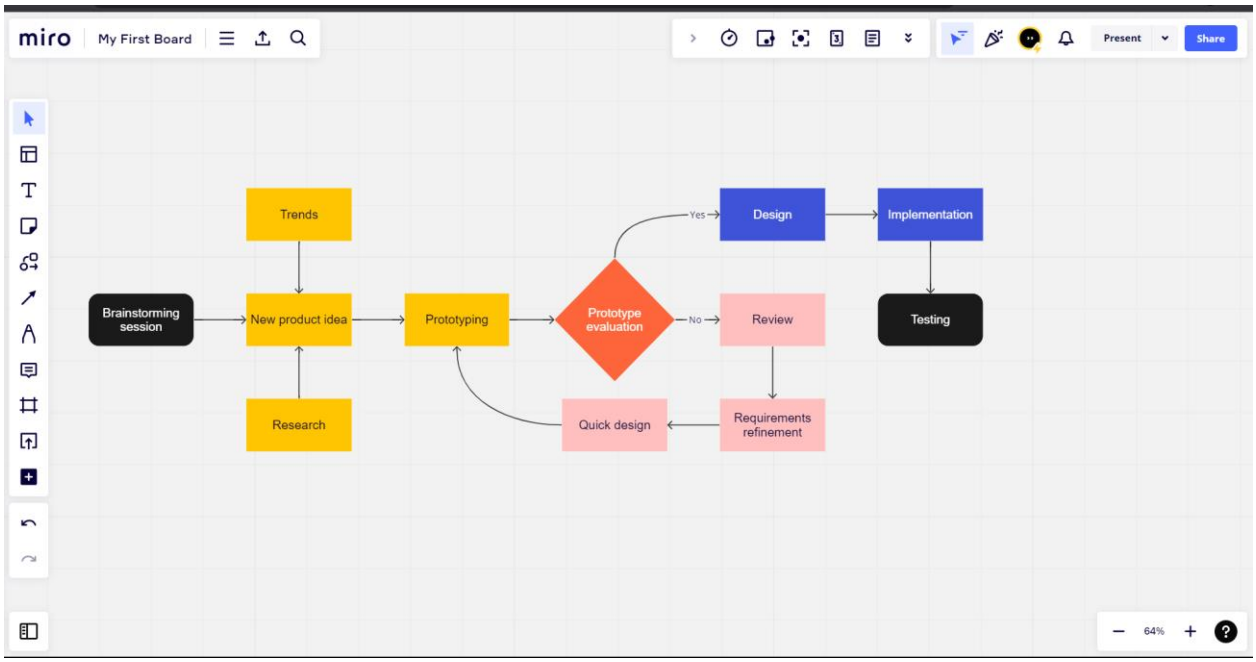

Рисунок. 1.3 – Дорожня карта для розробки

Ці задачі спрямовані на створення високоякісної та функціональної інформаційної системи, яка надає користувачам можливість ефективно навчатись кондитерському мистецтву та розвиватись у цій галузі.

2 ОБРАНИ МЕТОДИ РЕАЛІЗАЦІЇ

2.1 Обрані технології та фреймворки для реалізації

Express:

Express є більш легким та гнучким веб-фреймворком для розробки серверних додатків на Node.js. Він надає широкі можливості для обробки маршрутів, створення обробників запитів та роботи зі статичними файлами. Express спрощує розробку веб-сервера та забезпечує зручність управління HTTP-запитами та відповідями.

React:

React є популярною бібліотекою для розробки користувацького інтерфейсу (UI) на JavaScript. Він дозволяє побудувати ефективні та перевикористані компоненти, які забезпечують швидке відображення змін в інтерфейсі користувача. React пропонує односторінкову архітектуру та використовує віртуальний DOM для оптимізації процесу оновлення сторінки.

Node.js:

Node.js є середовищем виконання JavaScript на серверній стороні. Воно дозволяє розробляти серверні додатки за допомогою JavaScript, що дозволяє забезпечити єдиний мовний стек як на клієнтському, так і на серверному боці. Node.js має необхідні засоби для роботи з мережевими запитами, файловою системою та іншими операціями на сервері.

Переваги цього стеку технологій включають:

1. Єдиноформатний код: використання JavaScript в якості мови програмування як на клієнтській, так і на серверній стороні дозволяє розробникам використовувати один і той самий код для реалізації функціональності на обох частинах додатку, що спрощує розробку та підтримку.

2. Швидкодія: використання Node.js у поєднанні з MongoDB та Express дозволяє створювати швидкі та ефективні веб-додатки, оскільки Node.js має неблокуючий (non-blocking) асинхронний (asynchronous) підхід до обробки запитів.
3. Масштабованість: MongoDB, Express, React та Node.js (MERN) надають гнучкість та масштабованість для розробки та впровадження веб-додатків будь-якої складності. Вони дозволяють легко масштабувати систему для відповіді на зростаючі потреби користувачів.
4. Активна спільнота: усі ці технології мають велику та активну спільноту розробників, що забезпечує підтримку, навчання та розвиток. Існує багато документації, ресурсів та відкритих джерел, які допомагають вирішувати проблеми та вдосконалювати навички розробки.
5. Відкритий код: більшість інструментів у стеку MERN є відкритими та безкоштовними, що дозволяє економити витрати на ліцензії та сприяє розвитку відкритої спільноти.

Загалом, MongoDB, Express, React та Node.js (MERN) надають потужні та гнучкі інструменти для розробки сучасних веб-додатків, які дозволяють розробникам швидко будувати, масштабувати та розширювати систему з використанням єдиноформатного коду та сучасних технологій.

2.2 Платформи та інструменти для розробки

Docker – це програмна платформа для швидкої розробки, тестування та розгортання додатків. Docker упаковує програмне забезпечення (ПЗ) у стандартизовані блоки, які називаються контейнерами. Кожен контейнер містить усе необхідне для роботи застосунку: бібліотеки, системні інструменти, код і середовище виконання. Завдяки Docker можна швидко розгортати і масштабувати додатки в будь-якому середовищі та зберігати впевненість у тому, що код працюватиме.

Використання Docker на AWS надає розробникам і системним адміністраторам надійний і економічний спосіб складання, доставки та запуску розподілених додатків будь-якого масштабу. Docker співпрацює з AWS, щоб допомогти розробникам прискорити доставку сучасних додатків у хмарі. Завдяки такій взаємодії розробники можуть за допомогою Docker Compose і Docker Desktop оптимізувати локальний робочий процес, що застосовується на даний момент, для безперешкодного розгортання додатків на Amazon ECS і AWS Fargate. Docker використовується в різних сферах і має декілька конкретних застосувань:

1. Розробка та тестування: Docker дозволяє розробникам створювати контейнери, які містять усі необхідні компоненти для їх додатків. Це забезпечує однорідність середовища між розробкою та продуктивними серверами, що спрощує виправлення помилок та забезпечує надійну роботу програм.
2. Розгортання мікросервісної архітектури: Docker дозволяє розбити додаток на окремі компоненти, які можуть бути запущені у власних контейнерах. Це сприяє модульності та масштабованості додатків, оскільки окремі сервіси можуть бути розгорнуті та масштабовані незалежно один від одного.
3. Континуальна інтеграція та постійна доставка (CI/CD): Docker використовується для автоматизації процесу збирання, тестування та розгортання додатків. За допомогою Docker можна легко налаштувати середовище для автоматичного тестування, побудувати контейнер з готовим додатком та безпечно розгорнути його на серверах.
4. Хмарні обчислення: Docker є популярним інструментом для розгортання та управління додатками в хмарних середовищах. Він дозволяє легко масштабувати та управляти контейнерами на хмарних платформах, таких як Amazon Web Services (AWS), Microsoft Azure, Google Cloud тощо.

Незважаючи на багато переваг Docker, варто враховувати його потенційні недоліки та ризики.

Деякі з них включають:

1. Складність налаштування: хоча Docker спрощує процес розгортання.
2. Конфігурація та управління можуть бути відносно складними, особливо для новачків. Є необхідність вивчати та розуміти концепції контейнеризації та конфігураційні файли Docker.
3. Потреба в ресурсах: використання Docker може вимагати значних ресурсів, таких як пам'ять та обсяг диску, особливо при розгортанні великих додатків або багатоконтейнерних середовищ.
4. Безпека: несправна конфігурація або недостатні заходи безпеки в Docker можуть відкрити двері до потенційних вразливостей та атак. Важливо правильно налаштувати доступи, контролювати довірені образи та використовувати ізольовані мережі для забезпечення безпеки контейнерів.
5. Сумісність та залежності: деякі програми можуть мати певні вимоги до операційних систем або бібліотек, що може вплинути на сумісність з контейнерами Docker. А ще залежності та версії програм в контейнерах також потребують уваги та керування, щоб уникнути конфліктів.
6. Управління контейнерами: зі зростанням кількості контейнерів, управління та моніторинг може стати складним завданням. Необхідно мати ефективні інструменти та стратегії для керування контейнерами, масштабування ресурсів та вирішення проблем, що можуть виникати. [5]

В цілому, Docker є потужним інструментом для контейнеризації додатків, який дозволяє ефективно розгортати, управляти та масштабувати додатки. Проте, перед використанням Docker варто вивчити його основні концепції та врахувати потенційні недоліки та ризики, пов'язані з використанням Docker.

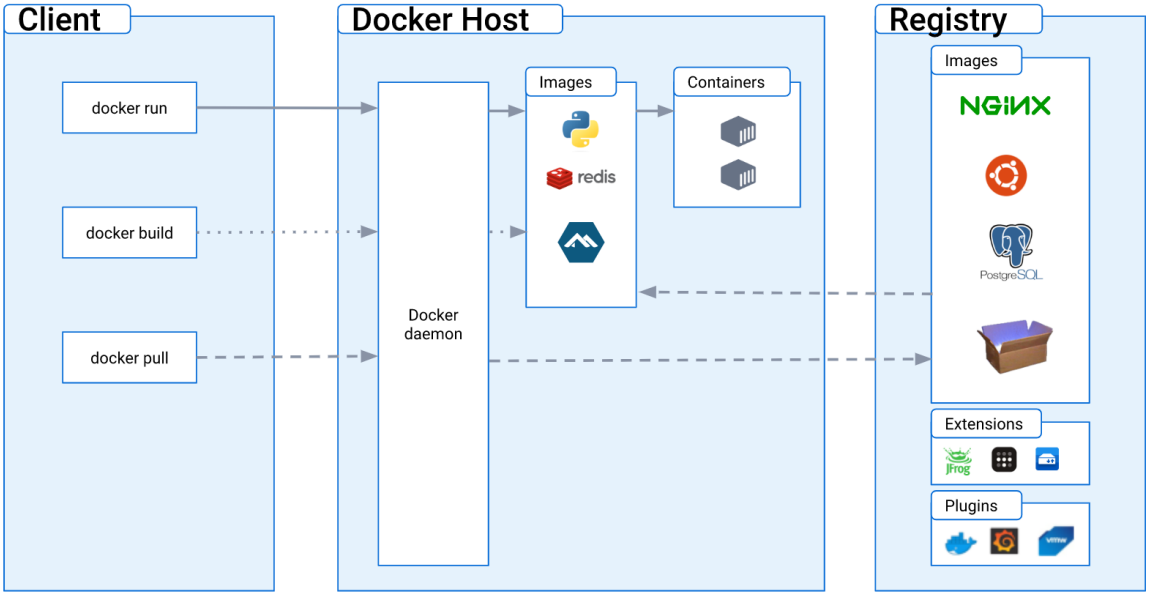

Рисунок. 2.1 – Робота Docker

(https://docs.docker.com/get-started/overview/)

2.3 Сервіс Git

Git - це три сервіси в одному. Цілий комплекс програмних рішень, що використовуються в різних сферах діяльності для відстеження ітерацій розроблюваного продукту. Найчастіше Git використовується в розробці застосунків і сайтів, але він також знаходить застосування і в інших сферах. Пояснити у двох словах всю систему майже нереально, тому що Git містить у собі настільки різноманітні функції та можливості, що підігнати загальну ідею під один з ходу зрозумілий термін не видається можливим. Тому краще розпочати з базового визначення і поступово перейти до конкретних функцій.

Під системою контролю в контексті Git мається на увазі програмний механізм для роботи з контентом. У його функції також входить зберігання, передача даних, відстеження змін та інші аспекти. Як вже було зазначено вище, зазвичай Git використовують для роботи з програмним кодом. Кодери ведуть розробку своїх продуктів, використовуючи систему контролю версій, щоб мати повний контроль над своїм дітищем, над кожною його версією і варіацією. Тобто, в будь-який момент часу, можна повернутися в минуле і

відновити попередню версію програми, якщо новий реліз все зламав. Або коли терміново потрібно подивитися, який вигляд мав код до рефакторингу.

І це стосується не тільки коду. Правилам роботи в системі контролю версій можна підпорядкувати майже будь-який продукт і скрізь від цього буде тільки користь.

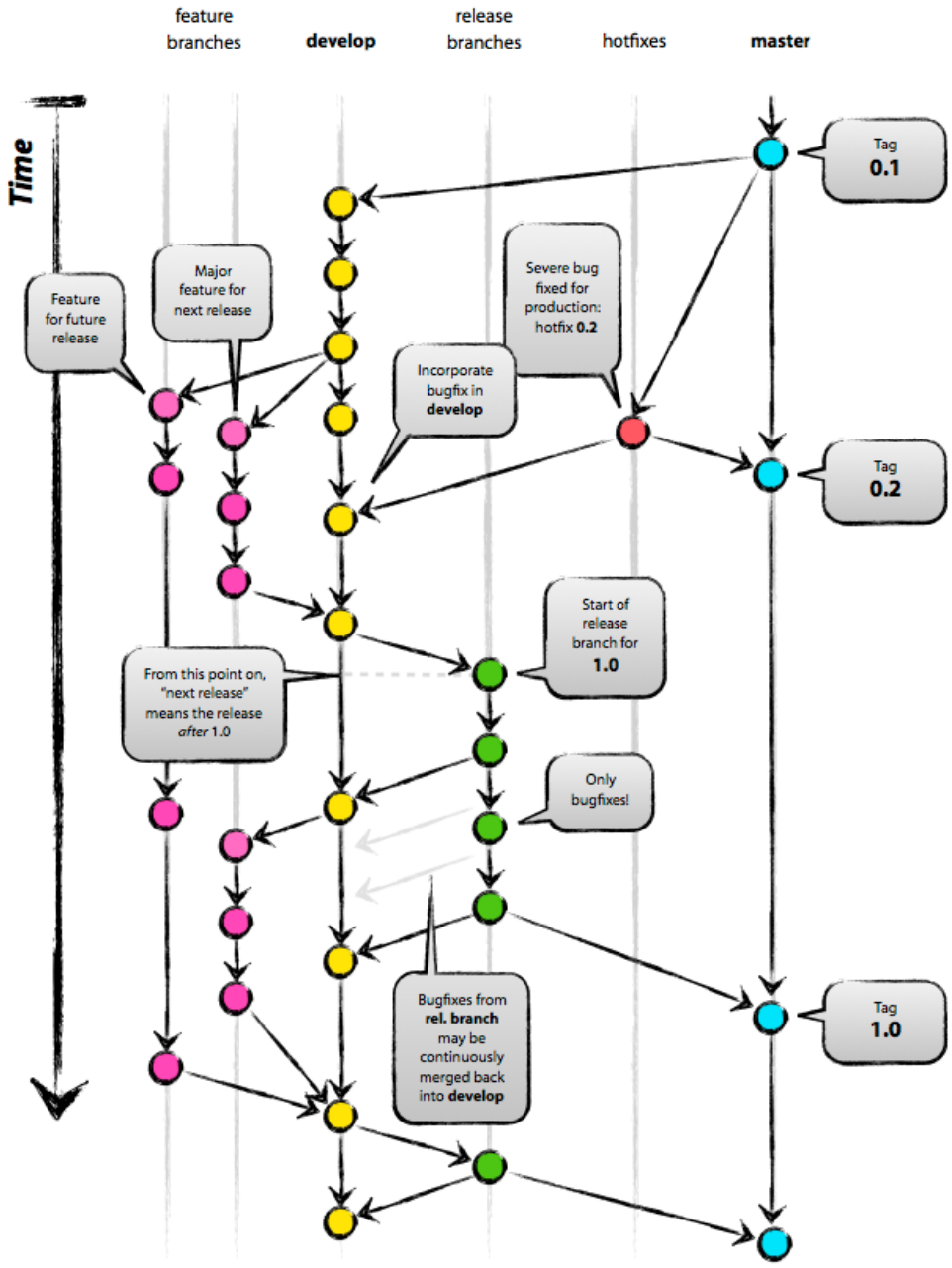

Рисунок. 2.2 – Схема гілок (Git Flow)-(https://geniusee.com/single-blog/everything-you-need-to-know-about-git-flow-branch-model)

Навіщо потрібен Git? Для того, щоб не стався апокаліпсис у світі. Зараз важко уявити собі більш-менш великий додаток, над яким працювала б одна людина. За проектом завжди стоїть ціла команда творців. І щоб їм усім було комфортно працювати разом, потрібна розподілена система контролю версій. Це гарантія відсутності конфліктів у коді та можливість вести розробку кількох функцій ПЗ, не стикаючись одна з одною і загальним кодом. І навіть більше, під час розробки часто змінюються вимоги до проєкту, бачення замовника і стандарти дизайну. Тому повний контроль над усіма ітераціями застосунку дає змогу легко повернути будь-які зміни та вносити поверх них нові.

Розглянемо ще деякі поняття такі як: репозиторіїї і коміти.

Репозиторій – це сховище файлів проєкту, тобто всі компоненти, що відносяться до одного додатка (код, зображення, конфігураційні файли, стилі, скрипти тощо). З ним якраз і працюють люди, які ведуть спільну роботу над одним продуктом. При створенні репозиторію в папці з файлами формується .git-директорія, що містить інформацію, необхідну для коректної роботи з системою контролю версій (кількість комітів, гілок розробки та інших необхідних деталей). За замовчуванням каталог .git прихований, але його бачать git-додатки, наприклад git, Sublime Merge, Gitfox та інші аналоги.

Коміт – це одиниця контенту, що зберігає в собі інформацію про те, які компоненти репозиторію були змінені на поточний момент часу. За рахунок них і працює контроль версій. Припустимо вирішено трохи змінити оформлення сайту. Відкриваєте файл .css, вводиться туди нове значення шрифтів або кольорів, а потім відбувається процес збереження. Щоб ця зміна відобразилася в git, потрібно створити коміт (git commit - m, опис коміту). Тепер є можливість в будь-який момент повернутися до цього етапу в розробці сайту.

Стейджинг-зона – це тимчасовий притулок змінених файлів. Вирушаючи до стейджинг-зони (git add), вони позначаються як проєкт у розробці, але водночас із них усе ще можна вибрати готові файли, щоб безпосередньо їх комітити. Файли зберігаються в стейджинг-зоні (git add) доти, доки користувач

не зробить коміт і не запустить (git push) зміни на сервер, тобто не вивантажить із локального репозиторію у віддалений.

2.4 Розробка Frontend частини

Frontend – це сфера розробки програмного забезпечення, яка відповідає за створення користувацького інтерфейсу (UI) і його взаємодію з користувачем. Він фокусується на тому, який вигляд має додаток або веб-сайт і як користувачі взаємодіють з його інтерфейсом. Frontend-розробники використовують різні мови програмування, такі як HTML (HyperText Markup Language), CSS (Cascading Style Sheets) і JavaScript, для створення структури, стилів і функціональності користувацького інтерфейсу.

HTML використовується для створення структури веб-сторінки, CSS – для оформлення та стилізації елементів інтерфейсу, JavaScript – для додавання інтерактивності та динамічної поведінки на сторінці.

Frontend-розробники також можуть використовувати фреймворки та бібліотеки, такі як: React, Angular або Vue.js, щоб спростити розробку та забезпечити більш ефективне управління станом додатка та взаємодією із сервером. Крім того, Frontend-розробники зазвичай працюють над адаптивним дизайном, щоб забезпечити хорошу роботу інтерфейсу на різних пристроях та екранах, таких як комп'ютери, планшети й мобільні пристрої.

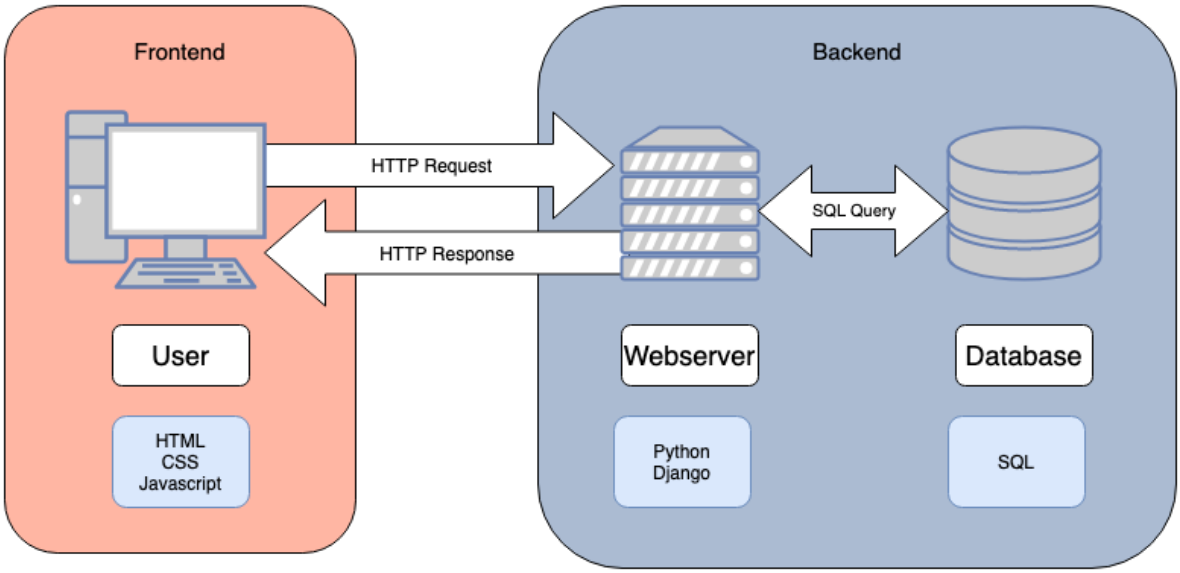

Рисунок. 2.3 – Взаємодія клієнтської частини та серверної (https://www.mohamavey.top/products.aspx?cname=front+y+backend&cid=89)

2.5 Програмні продукти UX і UI

UX і UI – це два скорочення, які використовуються у дизайні продуктів, зокрема веб-сайтів та додатків. Вони відображають два різні аспекти користувальницького досвіду та дизайну.

UX (User Experience) – означає користувацький досвід. Він включає в себе всі враження та емоції, які користувач відчуває при взаємодії з продуктом. Головна мета UX – забезпечити потреби користувачів, зробивши продукт зручним, ефективним та приємним у використанні. UX-дизайнер займається дослідженням потреб користувачів, розробкою концепцій, проектуванням взаємодії та тестуванням, щоб забезпечити оптимальний користувацький попит.

UI (User Interface) – означає інтерфейс користувача. Це включає в себе зовнішній вигляд та взаємодію елементів продукту, таких як кнопки, меню, форми, кольори, шрифти тощо. Головна мета UI – забезпечити ефективну та привабливу взаємодію користувача з продуктом. UI-дизайнер відповідає за створення графічного інтерфейсу, розробку стилів, розміщення елементів та створення унікального вигляду продукту. Узгоджений робочий процес між

UX-дизайнерами та UI-дизайнерами дозволяє створити продукт, який поєднує ефективну взаємодію з привабливим та зручним виглядом, задовольняючи потреби та очікування користувачів. [8]

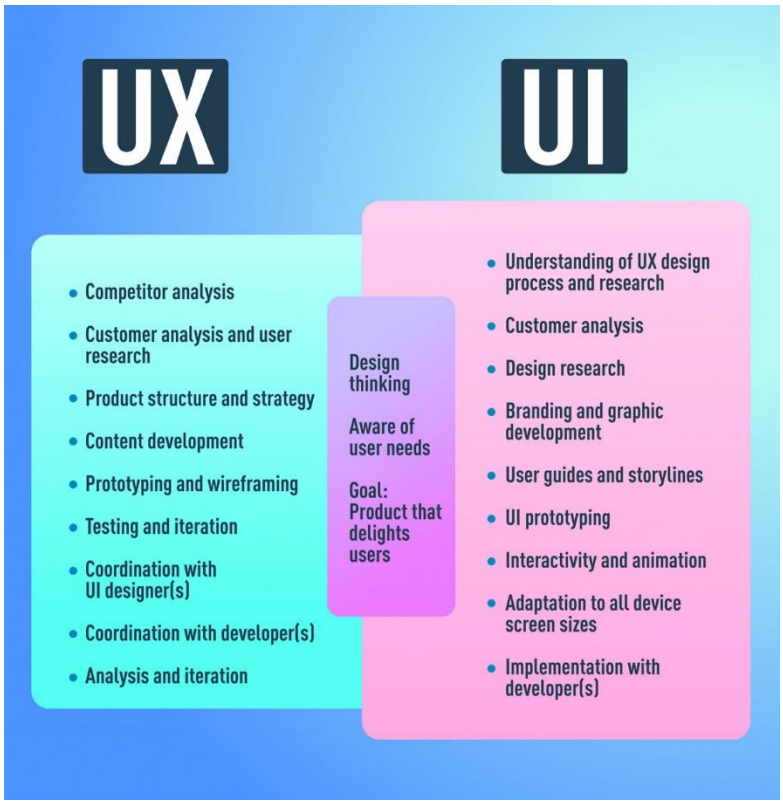

Рисунок. 2.4 – Схема **UX і UI**

(https://demibolt.medium.com/lets-solve-the-ui-ux-equation-46838d78016e)

2.5 Основні можливості ПЗ Backend

Backend – це область розробки програмного забезпечення, яка відповідає за створення і підтримку серверної частини додатків. Вона відповідає за обробку даних, бізнес-логіку та взаємодію з базами даних, а також забезпечує передачу даних між клієнтською (frontend) і серверною (backend) частинами програми.

Backend-розробка включає в себе використання різних мов програмування, таких як Java, Python, PHP, Ruby та інших, для написання серверного

коду. Backend-розробники створюють і підтримують веб-сервери, які обробляють запити від клієнтських додатків і повертають дані або результати операцій. [12]

Основні завдання backend-розробників включають:

1. обробку запитів від клієнтської частини додатка і маршрутизацію запитів до відповідних обробників.
2. роботу з базами даних, включно зі створенням, читанням, оновленням і видаленням даних (CRUD-операції).
3. реалізацію бізнес-логіки та алгоритмів, пов'язаних із функціональністю програми.
4. забезпечення безпеки даних і захист від загроз безпеки.
5. оптимізацію продуктивності серверної частини додатка і масштабування для обробки великих навантажень.

Backend-розробники також можуть використовувати фреймворки та інструменти, такі як Node.js, Django, Ruby on Rails та інші, щоб прискорити розробку, спростити управління базами даних і забезпечити ефективнішу роботу мережевих протоколів і API.

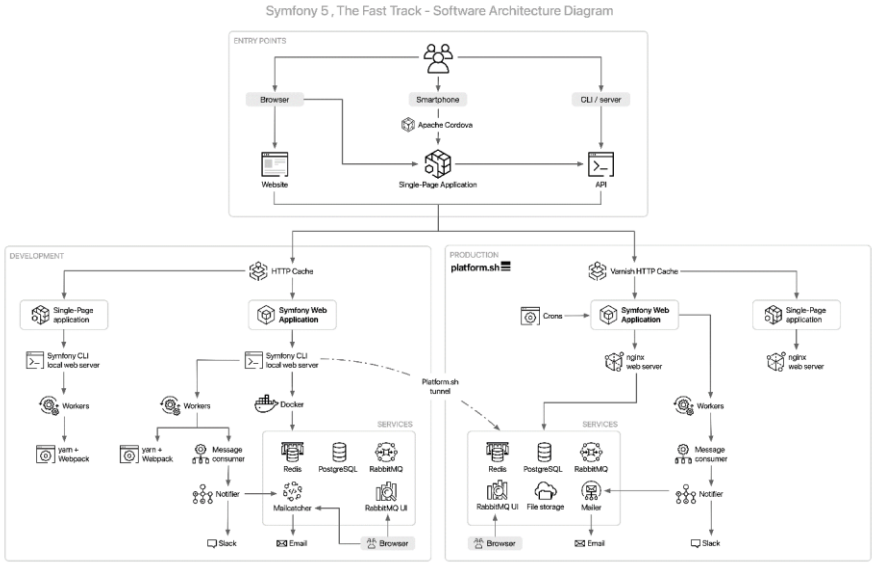

Рисунок. 2.5 – Підсумкова діаграма структури сучасного проекту(https://symfony.com/book)

2.6 Особливості використання Full stack застосунків

Як видно з наведеного вище, різниця між frontend і backend полягає в тому, які частини додатка обробляють і які завдання вирішують.

Frontend і backend взаємодіють один з одним для створення повноцінного веб-додатка. Frontend обробляє візуальну частину додатка і надсилає запити на серверну частину, а backend обробляє ці запити, виконує необхідні операції і надсилає назад дані або результати клієнту. Таким чином, frontend і backend працюють разом для забезпечення функціональності, зовнішнього вигляду та взаємодії веб-додатка з користувачами.

Розгланемо також Full-stack застосунок. Він належить до застосунку, у якому один розробник (або команда розробників) відповідають за створення як його frontend-частини, так і backend-частини. Такий розробник, відомий як full-stack розробник, здатний працювати як із клієнтським, так і з серверним кодом, а також керувати базами даних та іншими аспектами розробки.

Full-stack-розробники мають навички та знання в галузі розробки frontend і backend частин. Вони здатні створювати призначені для користувача інтерфейси з використанням HTML, CSS і JavaScript, а також писати серверний код на мовах програмування, таких як Java, Python, Ruby, PHP та інших. Вони знайомі з концепціями баз даних, роботою з API та мережевими протоколами.

Перевага full-stack-розробників полягає в тому, що вони можуть створювати повноцінні додатки від початку до кінця, не покладаючись на інших фахівців. Це може бути особливо корисно в невеликих командах або стартапах, де є обмежена кількість ресурсів і потрібно ефективно використовувати розробників.

Однак, слід зазначити, що full-stack-розробникам може бути складно охопити всі аспекти проєктування в глибину, оскільки розроблення frontend і backend вимагає різних наборів навичок та експертизи. У великих проєктах або компаніях може бути бажано мати фахівців, що спеціалізуються винятково на

frontend або backend розробці, щоб досягти вищої продуктивності та якості в кожній галузі. [5]

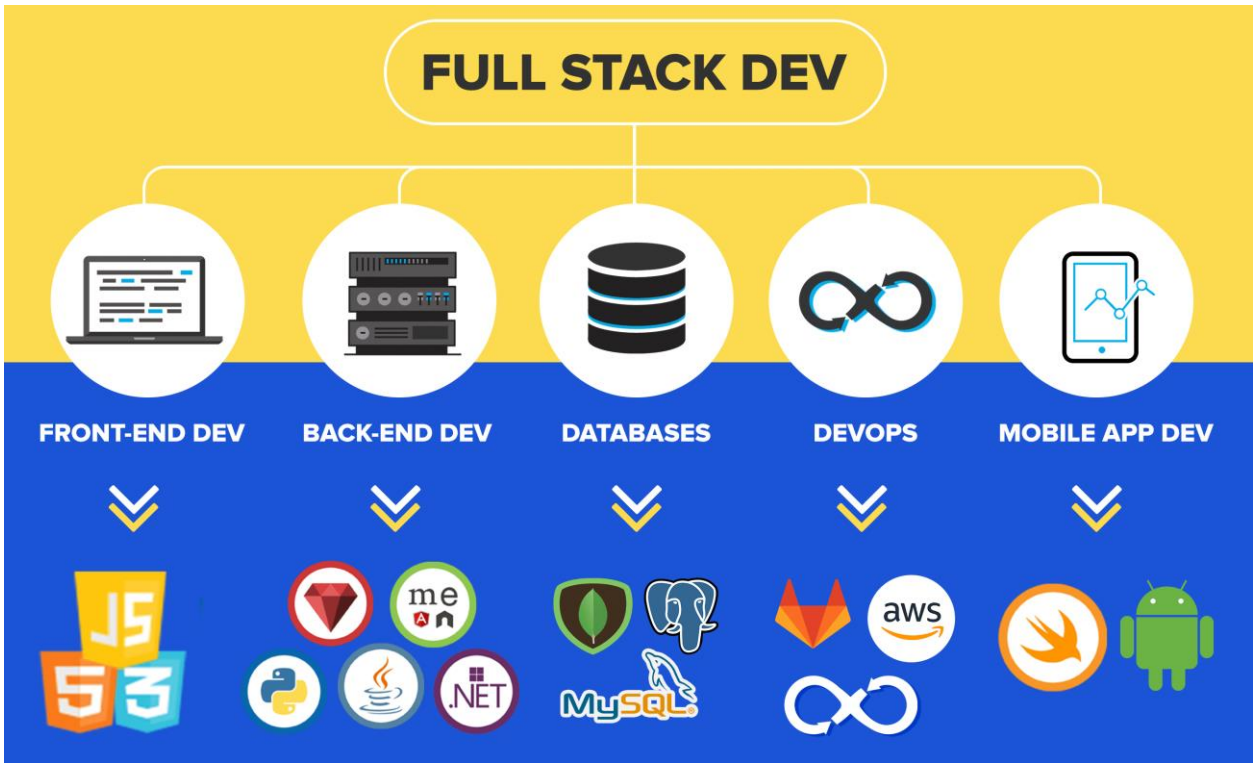

Рисунок. 2.6 – Full Stack Web-Development
(https://loftschool.com/blog/posts/full-stack-razrabotchik-kto-eto)

2.7 Етап тестування програмного продукту

QA (Quality Assurance) – це процес оцінювання та покращення якості програмного забезпечення. Він охоплює діяльність, спрямовану на перевірку та контроль якості розроблення та тестування програмного продукту. Відповідальність QA-фахівців полягає у виявленні дефектів, помилок і недоліків у застосунку, а також у поліпшенні процесів розробки.

Існує кілька видів тестування, які виконуються в рамках QA-процесу. Деякі з них включають:

1. модульне тестування (Unit Testing): Тестування окремих модулів або компонентів додатка для перевірки їхньої працездатності та відповідності заданим специфікаціям.

2. інтеграційне тестування (Integration Testing): Тестування взаємодії різних модулів або компонентів програми, щоб переконатися, що вони працюють разом правильно.
3. системне тестування (System Testing): Тестування додатка загалом, щоб перевірити його функціональність і відповідність вимогам та очікуванням користувачів.
4. приймальне тестування (User Acceptance Testing): Тестування, що проводять кінцеві користувачі або представники замовника, щоб переконатися, що додаток відповідає їхнім потребам і очікуванням.
5. навантажувальне тестування (Load Testing): Тестування додатка за високих навантажень або великого обсягу даних, щоб перевірити його продуктивність і масштабованість.
6. інтерфейсне тестування (UI Testing): Тестування призначеного для користувача інтерфейсу застосунку, щоб переконатися в його зручності використання та відповідності дизайну і стандартам.
7. безпека та захист тестування (Security Testing): Тестування застосунку на наявність вразливостей і перевірка його захищеності від несанкціонованого доступу та атак.

Це лише кілька прикладів типів тестування, а в реальних проєктах може використовуватися комбінація різних методів і підходів до тестування, залежно від вимог і характеристик застосунку. [8]

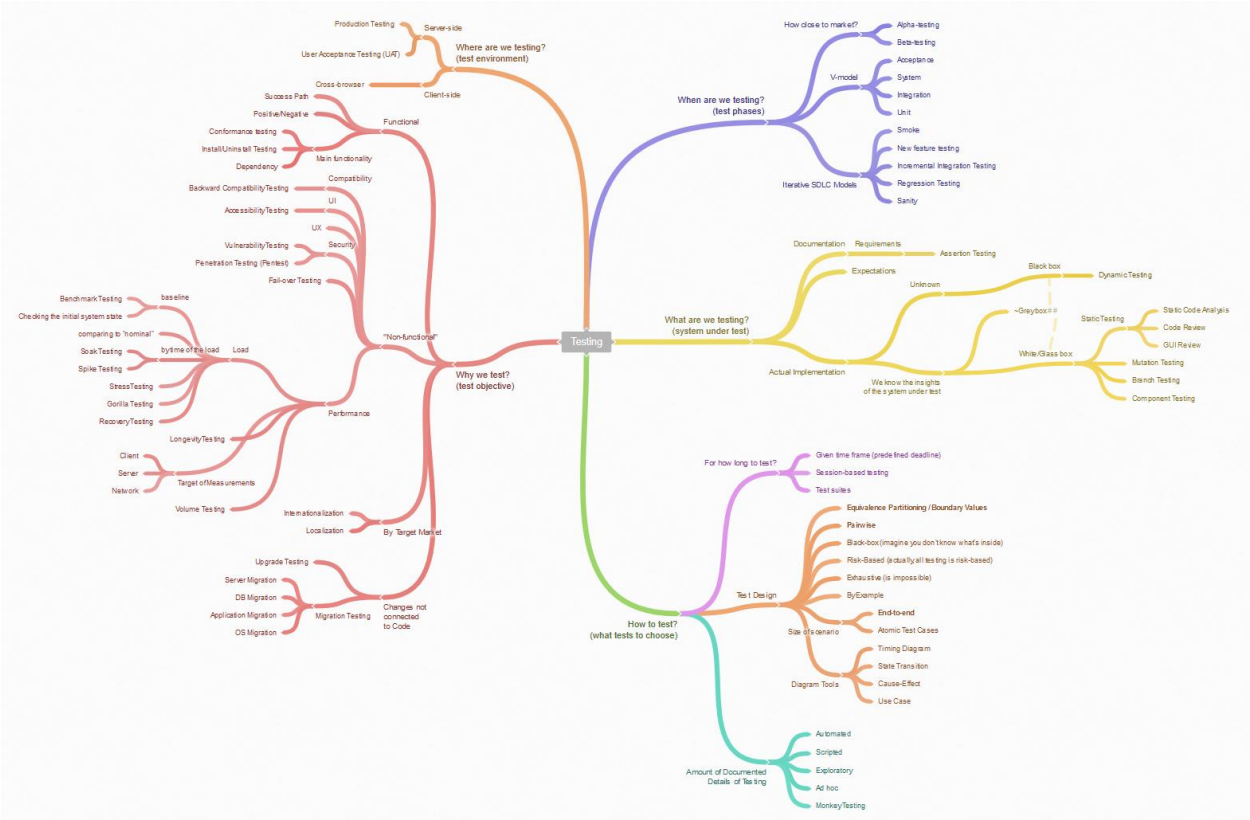

Рисунок. 2.7 – Схема тестувань

(https://coggle.it/gallery)

3 ПРАКТИЧНА РЕЛІЗАЦІЯ

3.1 Налаштування та реалізація бази даних

MongoDB є документоорієнтованою базою даних, яка зберігає їх у форматі JSON-подібних документів. Вона володіє гнучкістю і масштабованістю, дозволяючи легко зберігати та обробляти великі обсяги даних. MongoDB також надає можливість розподіленої реплікації та шардування для забезпечення високої доступності та швидкодії системи. [14]

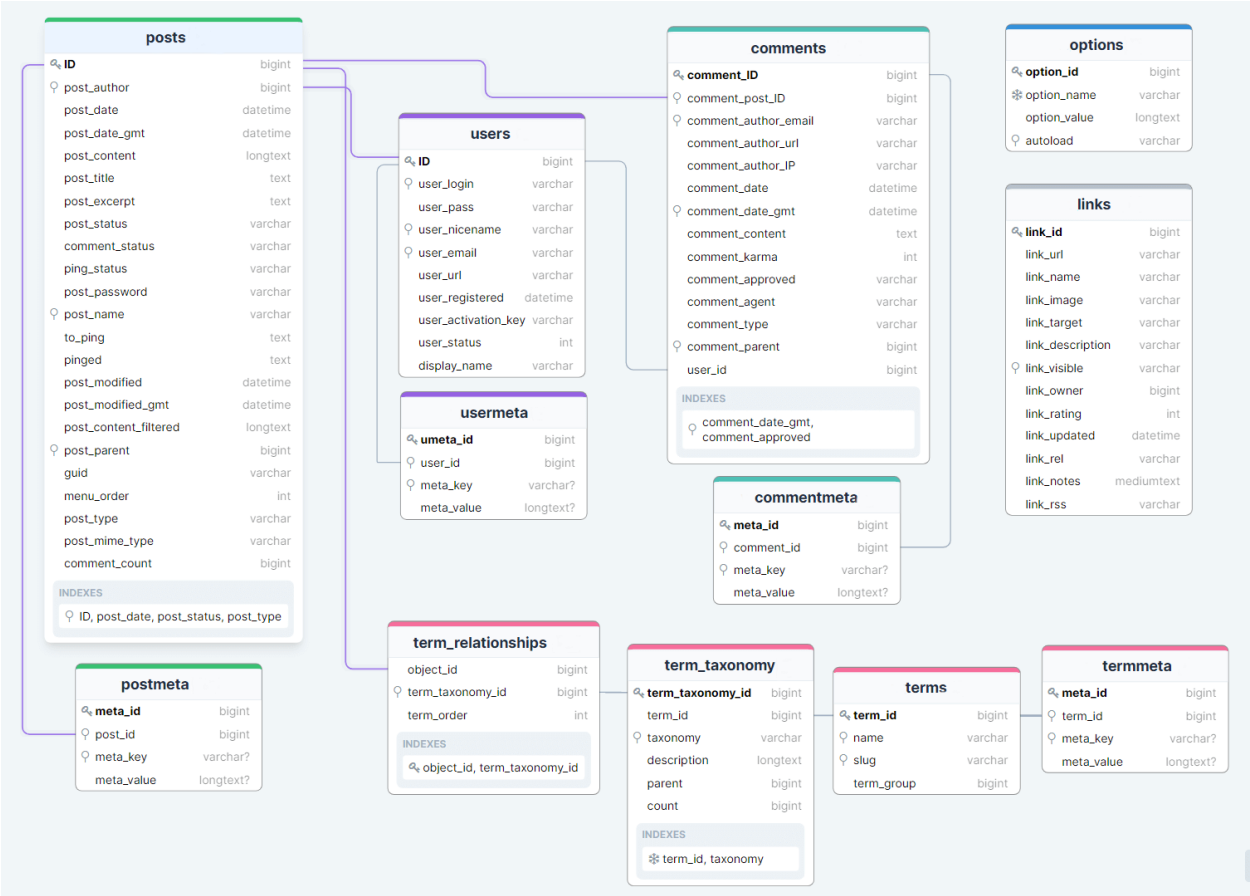

Рисунок 3.1 – ER діаграма

Таблиця 3.1 – Опис сутностей БД

Сутність	**Опис**
posts	Таблиця, куди записуються пости, постійні сторінки, довільні типи записів, вкладення тощо.
postmeta	Доповнює таблицю posts. Зберігає додаткові дані записів (постів) їх ще називають метаполя.
users	Таблиця з даними про зареєстрованих користувачів.
usermeta	Додаткова інформація про користувачів, така як Ім'я, Нік, права та інше.
comments	Таблиця із записами коментарів
commentmeta	Додаткові дані до comments
terms	Таблиця, що містить у собі базову інформацію про кожен елемент
termmeta	Таблиця, що містить у собі додаткові поля для таблиці terms.
term_taxonomy	Таблиця, що пов'язує таксономії з контентом (постами, записами тощо).
options	Таблиця опцій (налаштувань).
links	Таблиця із записами посилань

Дана таблиця 2.1 включає в себе інформацію, відомості та опис про основні сутності(тобто таблиці), які є в бази даних.

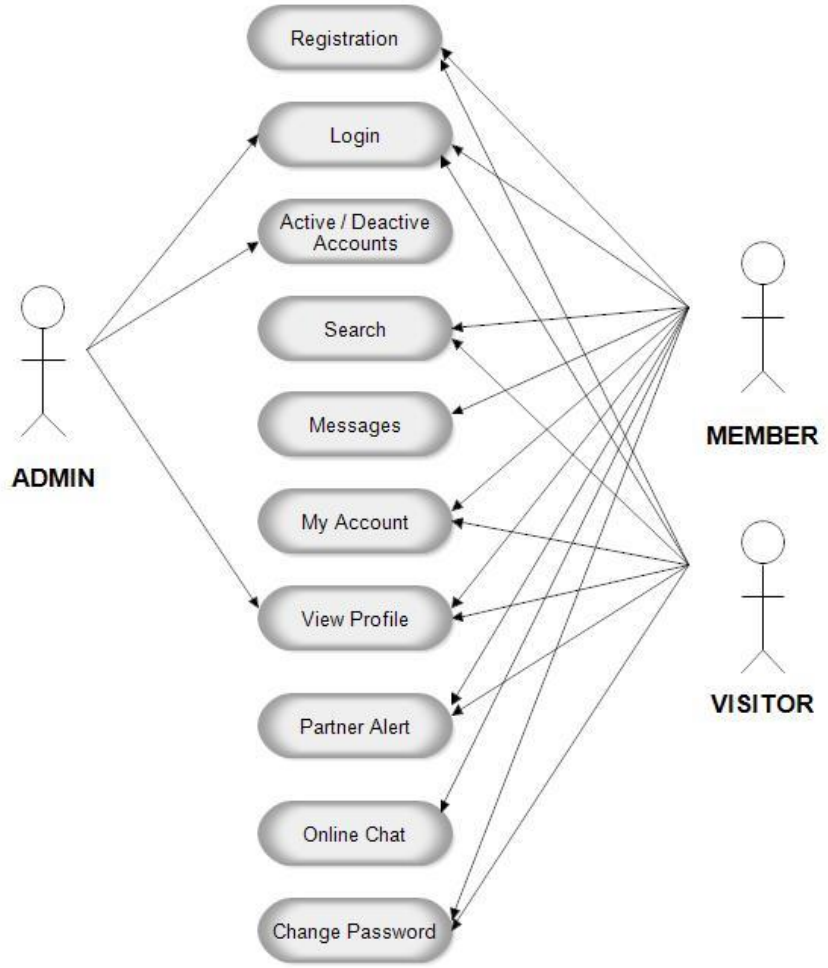

Рисунок 3.2 – Use-case діаграма

На рисунку 2.2 відображено діаграму варіантів-використання, або по іншому use-case діаграму, яка відображає основні процеси та можливості системи та акторів, які можуть виконувати ці дії.

3.2 Налаштування середовища

Для початку роботи користувача необхідно завантажити та налаштувати середовище для розробки, наприклад VS code. Після встановлення потрібно буде завантажити всі необхідні бібліотеки за допомогою npm в package.json.

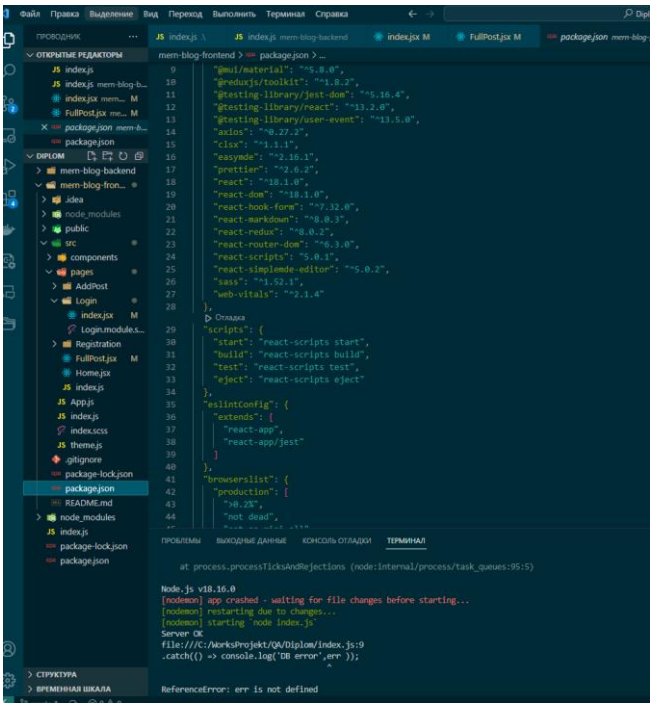

Рисунок. 3.3 – package.json

3.3 Технічна складова та реалізація проєкту

Після налаштування проєкту, розпочинається написання Backend. Спочатку відбувається написання базового коду для реєстрації, авторизації та подальшого адміністрування сайту. Необхідною умовою є можливість додавання картинок, фото і відео на сайт, подальше його адміністрування без зміни коду та перезапуску сервера, просто в онлайн режимі доповнення матеріалу.

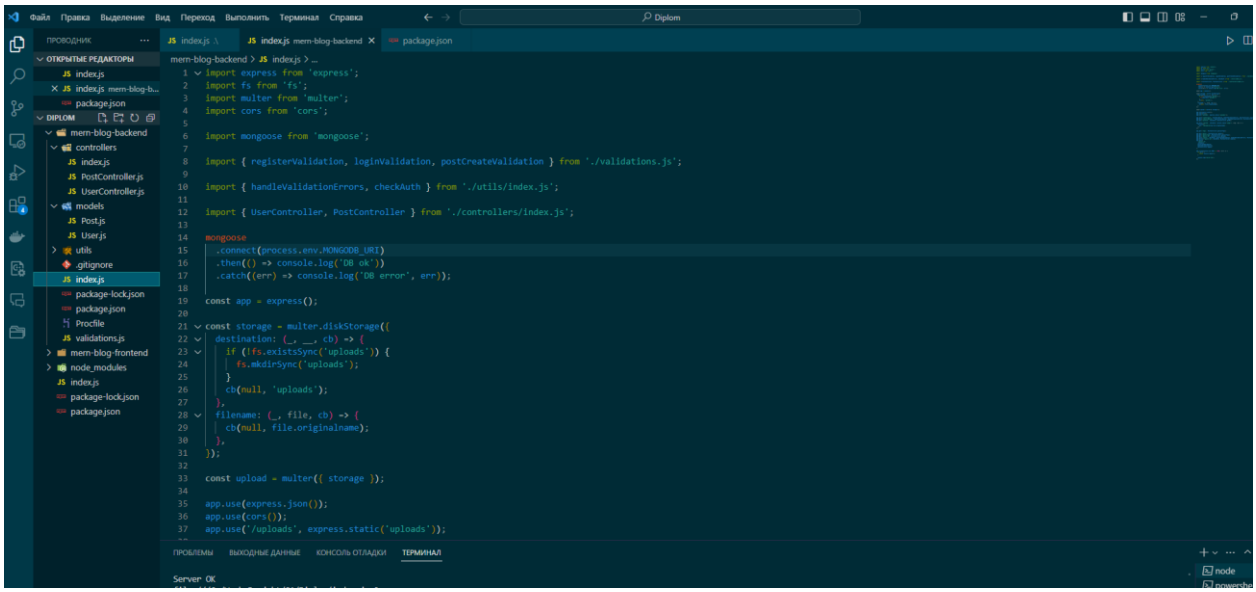

Рисунок. 3.4 – Код

Після написання базового коду, буде відображено текст - Server OK, це означає що сервер працює, отже можна рухатись далі. Після створення Backend, з’являється можливість по його розширенню та додаванню нових функції на сайт.Далі йде підключення підготовленої користувачем бази даних і запуск проєкту.

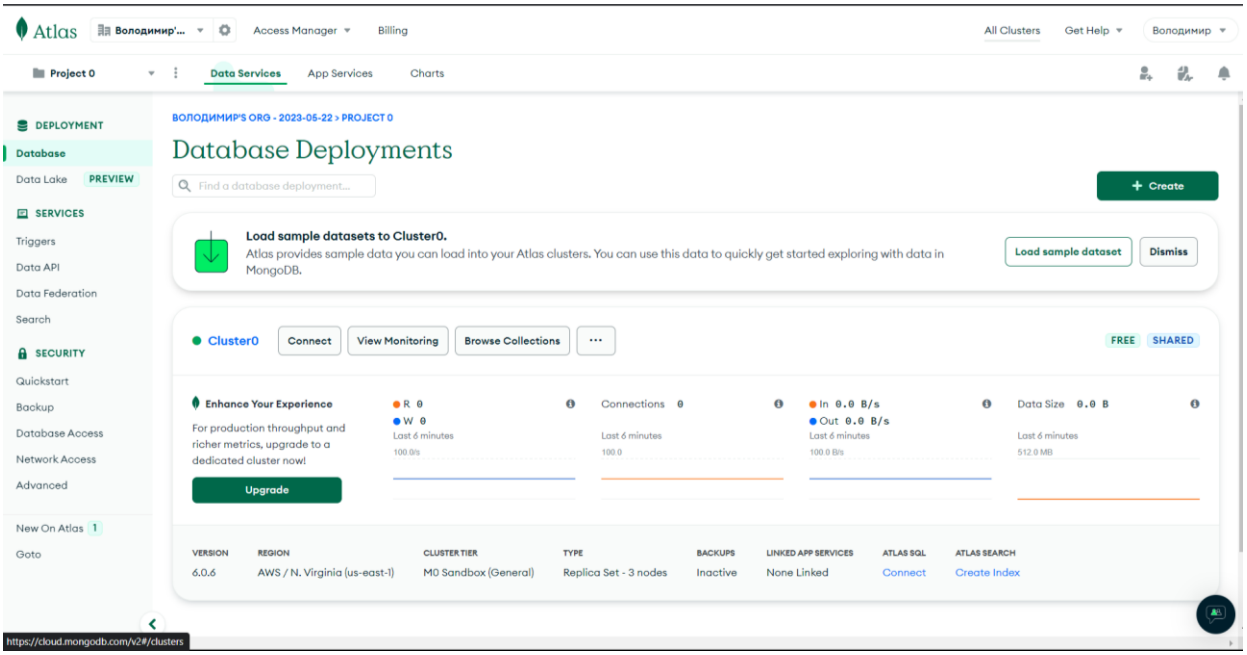

Рисунок. 3.5 – Створення бази даних

Після успішного приєднання бази даних потрібно перевірити роботу сервера, який запускається на Localhost 4444 (можна обрати будь-яку назву з 4 цифр). Тестується його коректна робота і відправляються запити. Для початку GET-запит і, якщо все добре працює, то потім POST-запит.

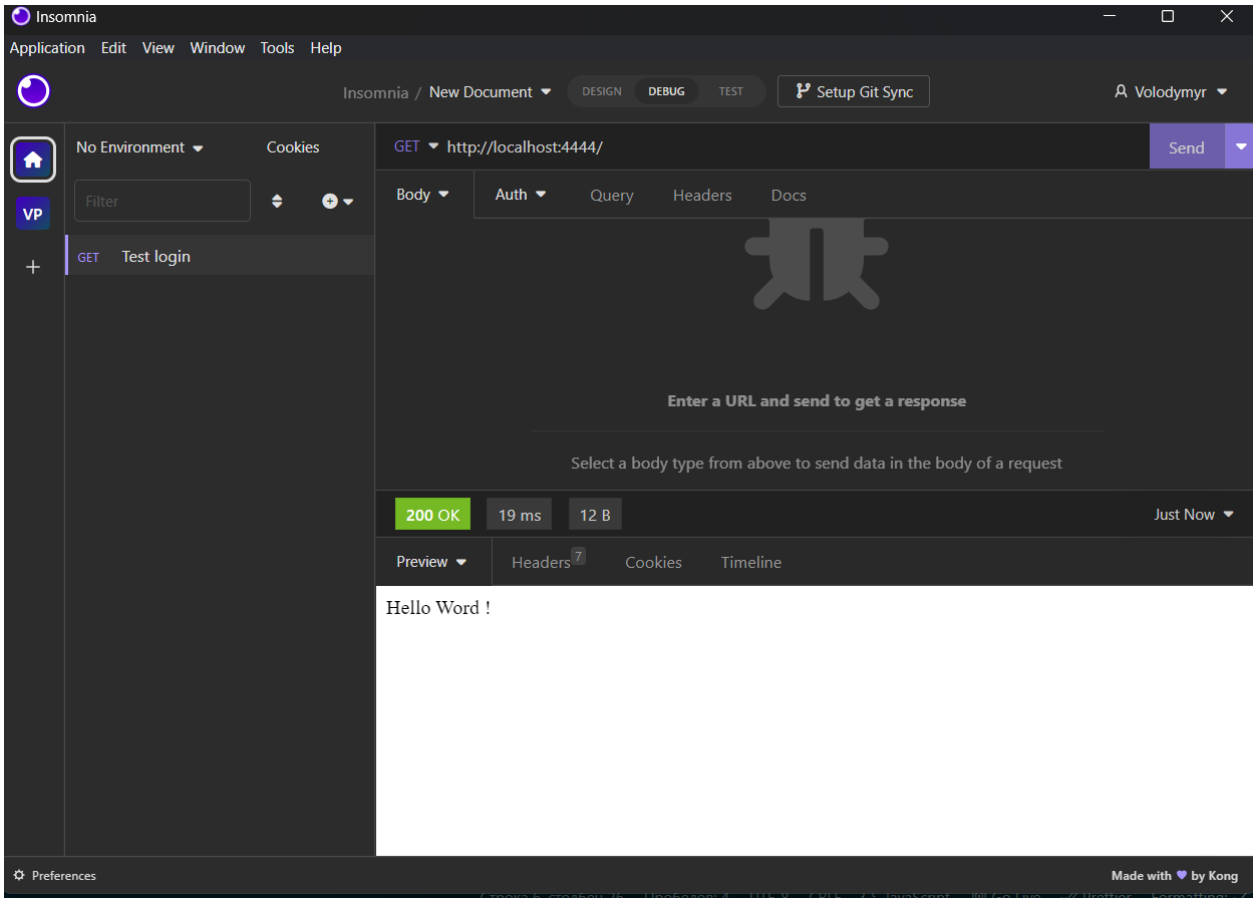

Рисунок. 3.6 – GET запит

Після успішних запитів, продовжується створення реєстрації за допомогою JWT-токену, що дає змогу зробити реєстрацію кращою та полегшити роботу користувача, усунути потенційні проблеми.

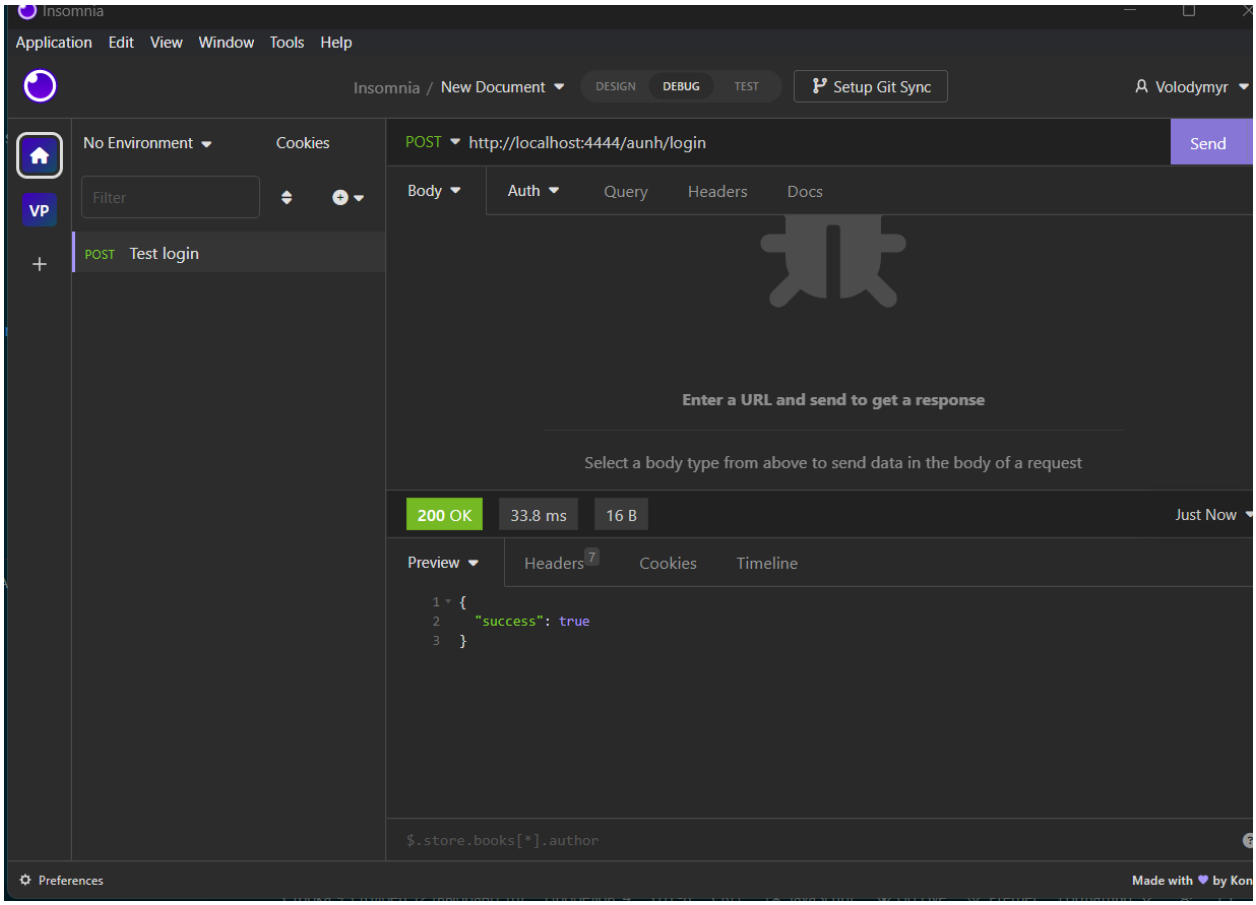

Рисунок. 3.7 – POST-запит

Після перевірки, що все працює, відбувається перехід до створення самого токену.

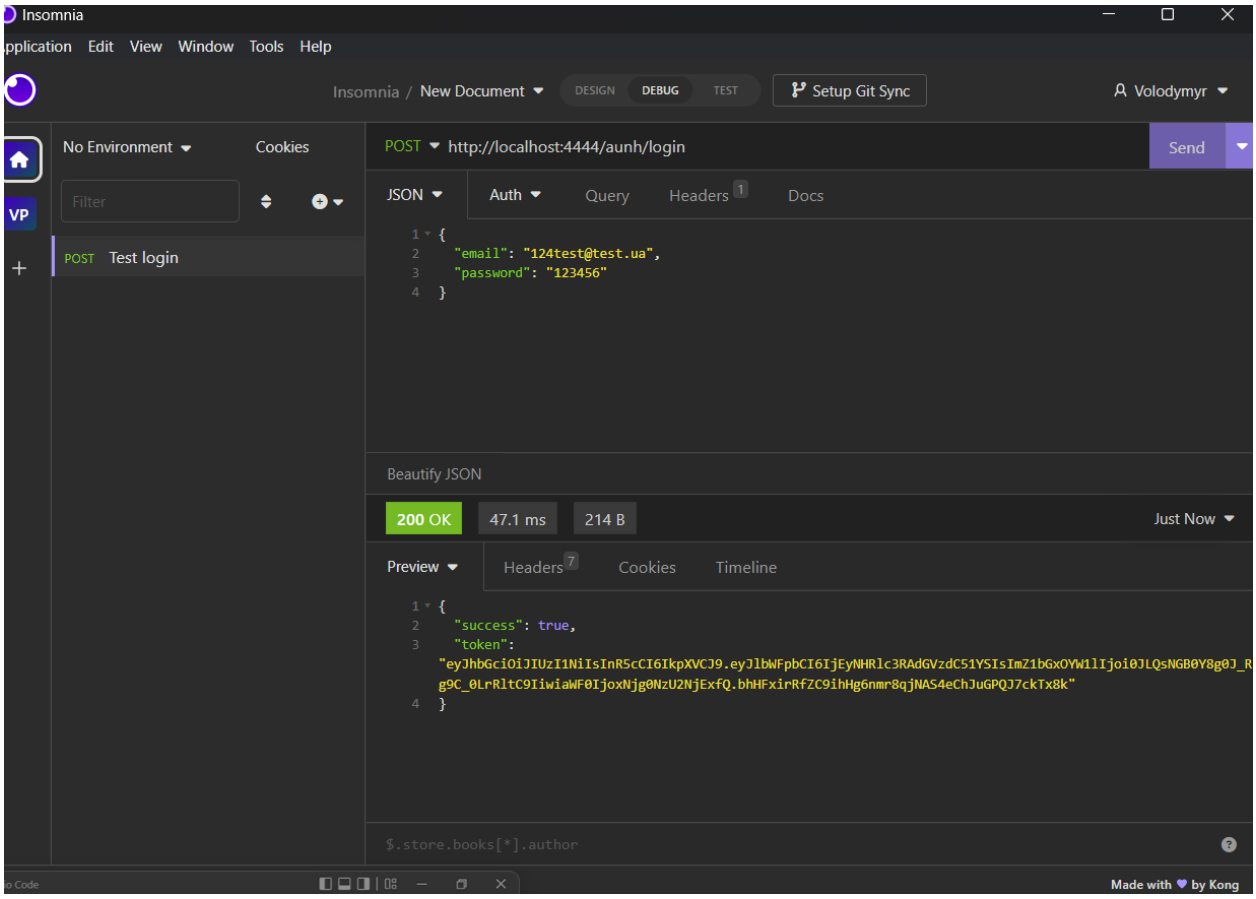

Рисунок. 3.8 – Тестовий логін

Створюється токен і проводяться перші спроби реєстрації та тестування проєкту, а також доповнення перших користувачів у спроєктовану базу даних.

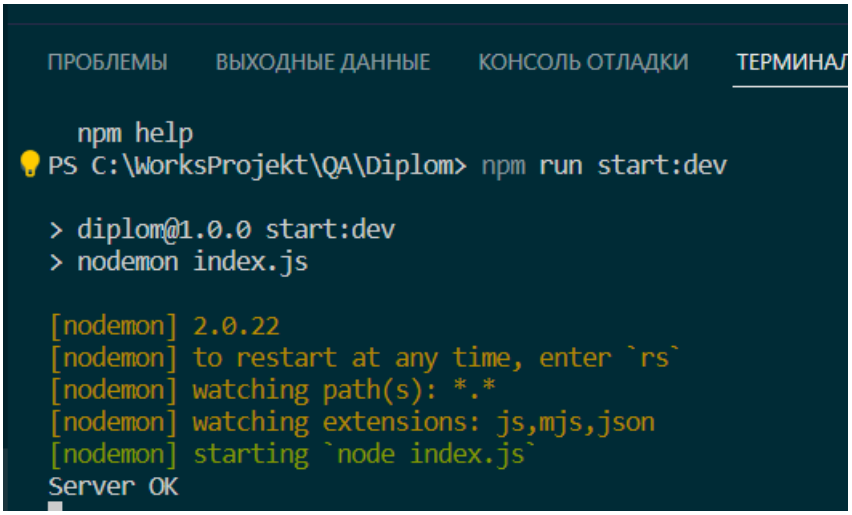

Рисунок. 3.9 – Запуск сервера в робочому режимі

На початку роботи щоразу, щоб щось змінити, необхідно було вимикати і перезавантажувати сервер за допомогою команди npm start, а вона вже запускала головний файл index.js. Потім після перевірки коректної роботи необхідно було знову вимикати, щоб щось додати. Але можна додати одну бібліотеку, яка це все буде робити за користувача за допомогою команди np m ini nodemon. Залишається тільки зробити збереження коду.

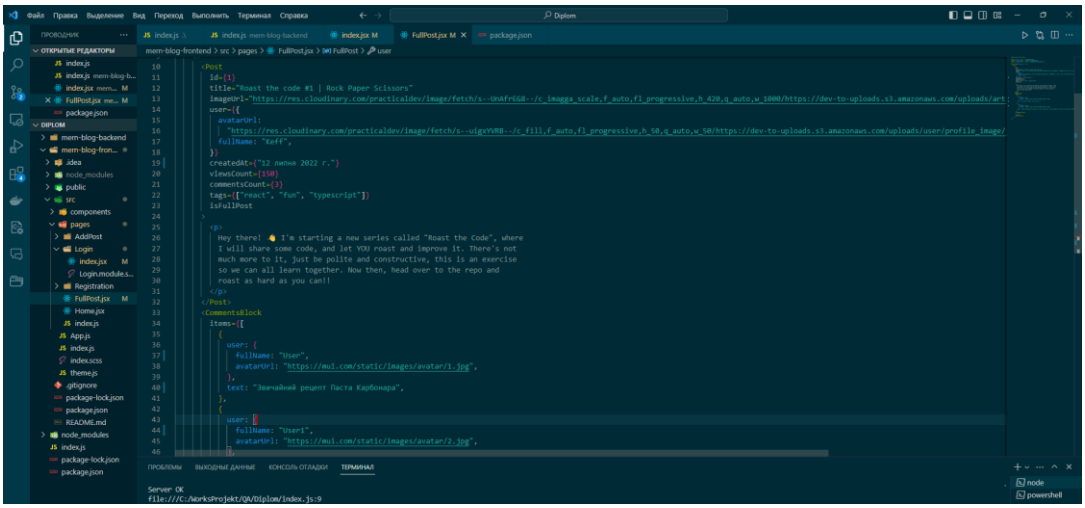

Рисунок. 3.10 – Написання базового Frontend

3.4 Приклади використання системи

Після запуску розробленого сайту на локальному хостингу, з’являється можливість проглянути цей сайт, в якому вже наявний базовий функціонал, так і його взаємодія з контентом.

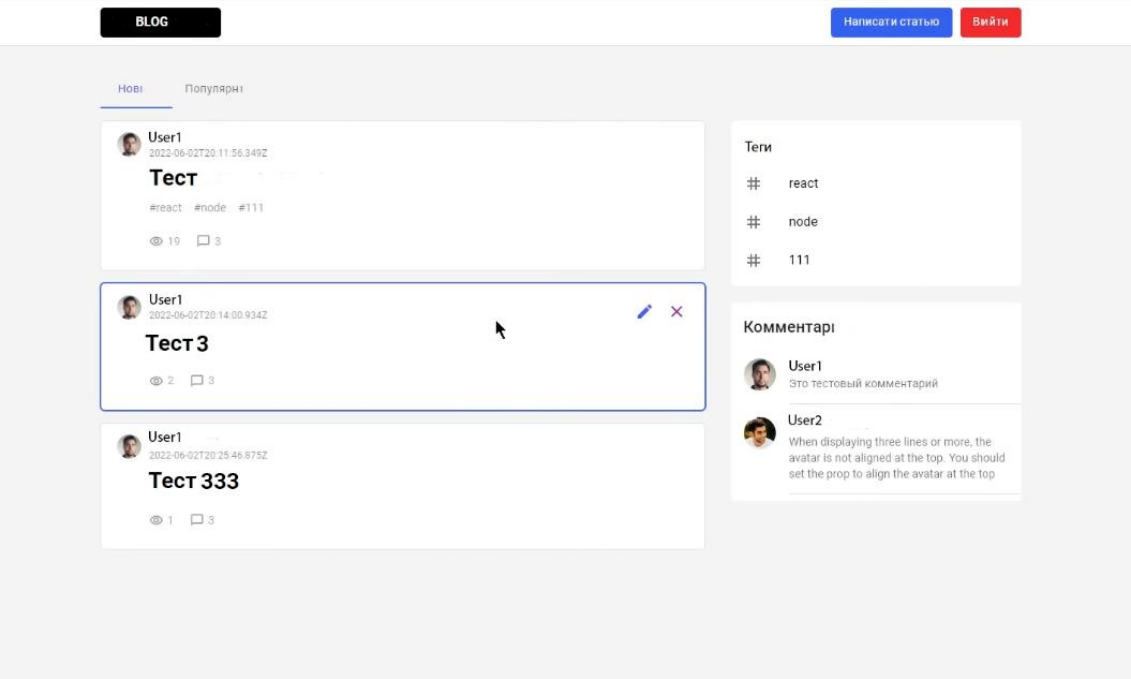

Рисунок. 3.11 – Перший запуск сайту

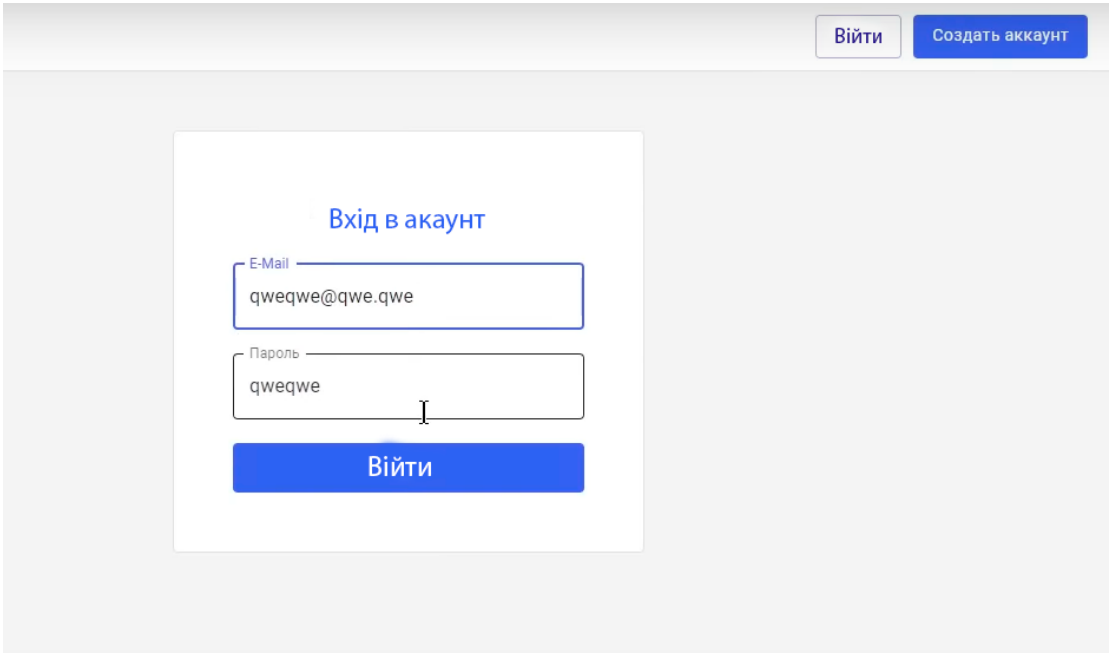

Рисунок. 3.12 – Вхід на сайт

Після входу на сайт є можливість додавання контенту з самого сайту, майже не взаємодіючи з кодом, але деякі аспекти будуть стилізовані та дещо змінені.

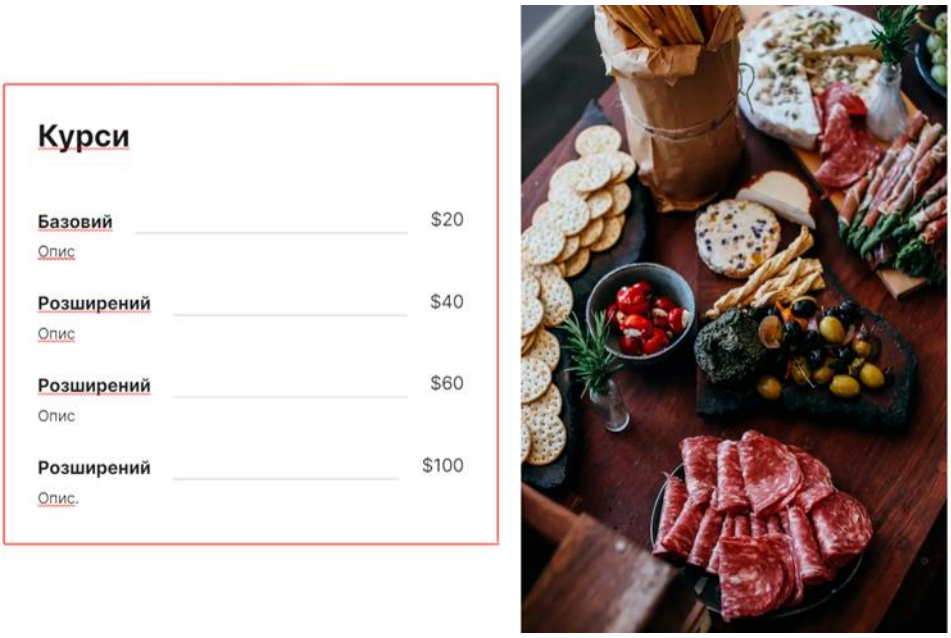

Рисунок. 3.13 – Створення планів курсів та їх ціна

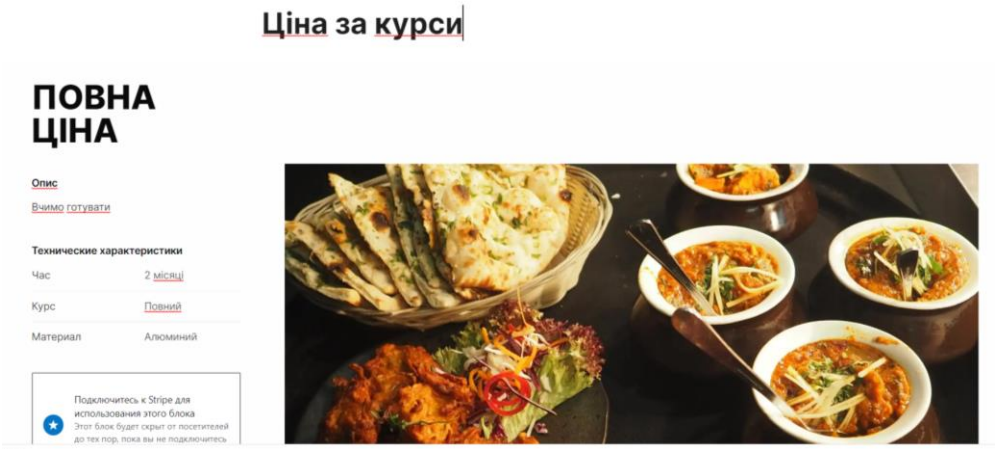

Рисунок. 3.14 – Створення планів курсів та їх ціна

Потім додається декілька сторінок: онлайн-уроки, про нас, контакти та відгуки, головна. Залишається заповнити їх контентом.

Рисунок. 3.15 – Створення онлайн уроків

Заповнюється контент для безкоштовного використання та додається контент, який буде доступний тільки підписникам сайту. Нижче буде показано сам контент сайту, який додається з YouTube, що є у відкритому доступі, а інші будуть надаватись тільки підписникам розробленого сайту. Отже, вони можуть проглядати інформацію, як на даному сайті, так і на самому YouTube-каналі для зручності.

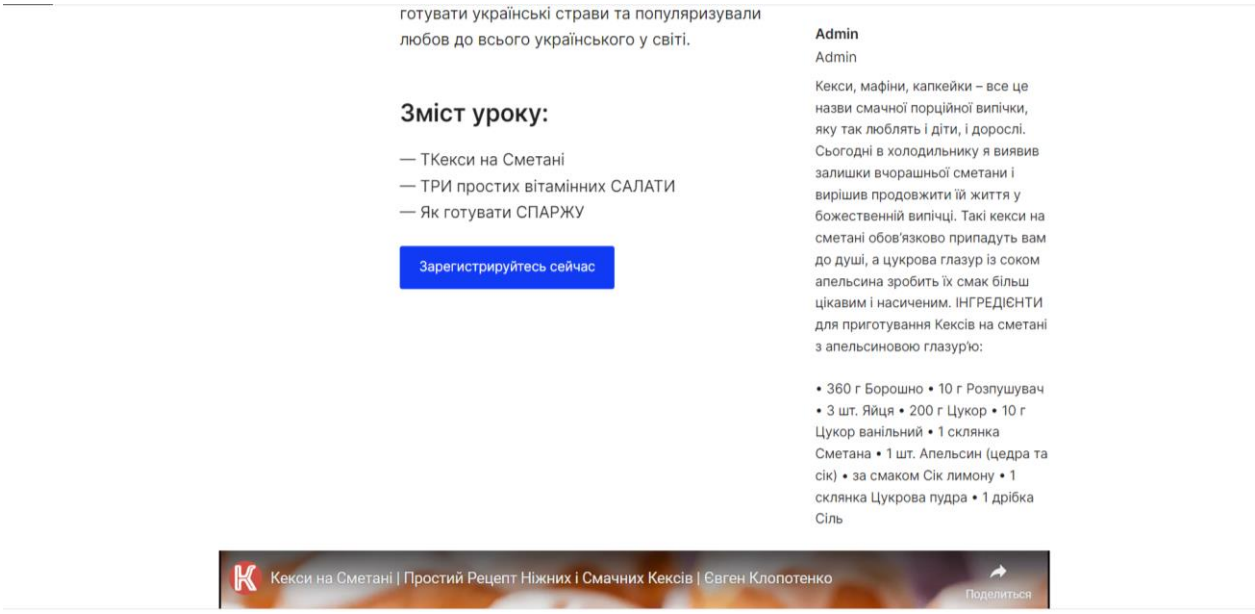

Рисунок. 3.16 – Контент онлайн уроків

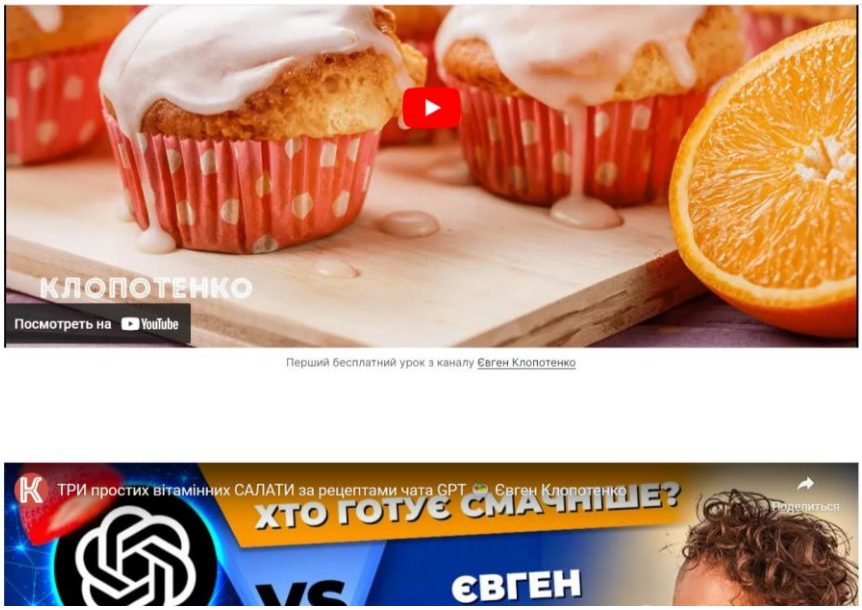

Рисунок. 3.17 – Контент онлайн уроків

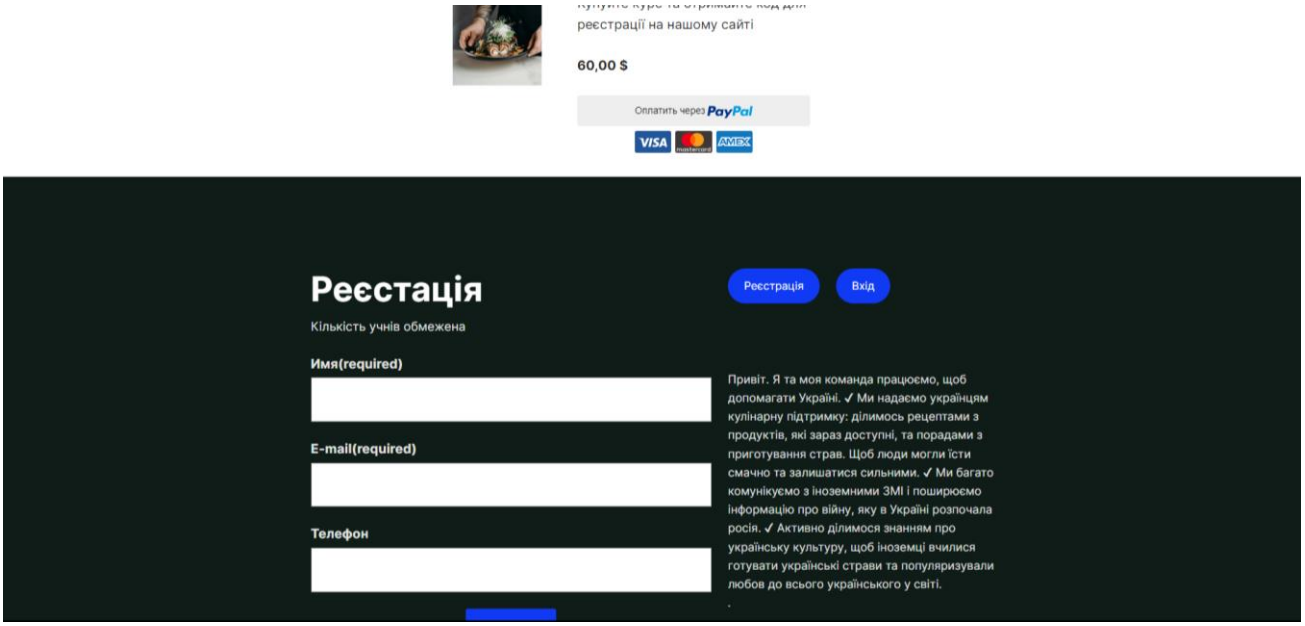

Рисунок. 3.18 – Закритий контент онлайн уроків

Потім заповнюється контент, до якого мають доступ лише Admin та студенти. Студенти можуть додавати свої роботи, а також коментарі до своїх робіт або інших студентів, задавати питання, і отримувати ексклюзивні уроки.

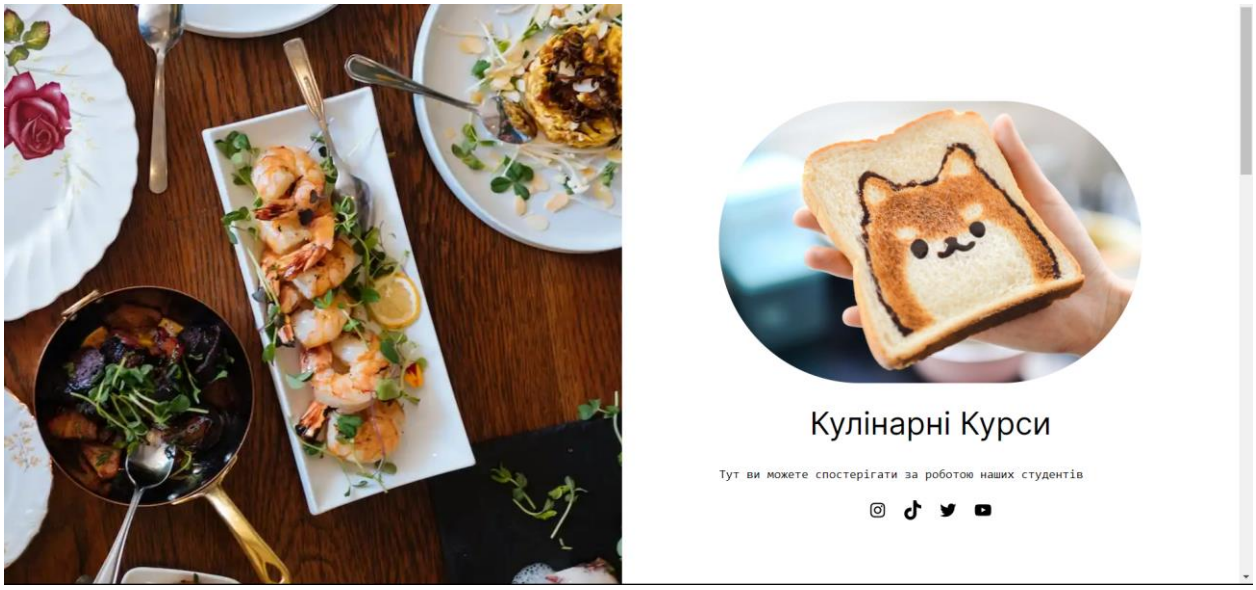

Рисунок. 3.19 – Закритий контент тільки для учнів та Admin

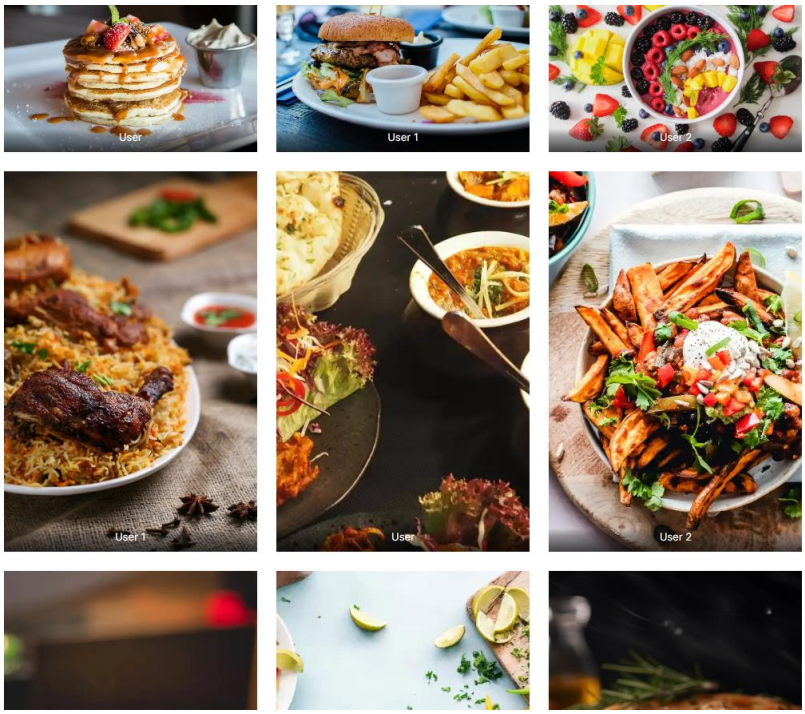

Рисунок. 3.20 – Галерея з роботами студентів.

Створення декількох коментарів, щоб перевірити систему. Видно успішну роботу та відображення того хто виклав роботу чи залишив коментар, дату, час та іншу інформацію. Також коректну роботу галереї.

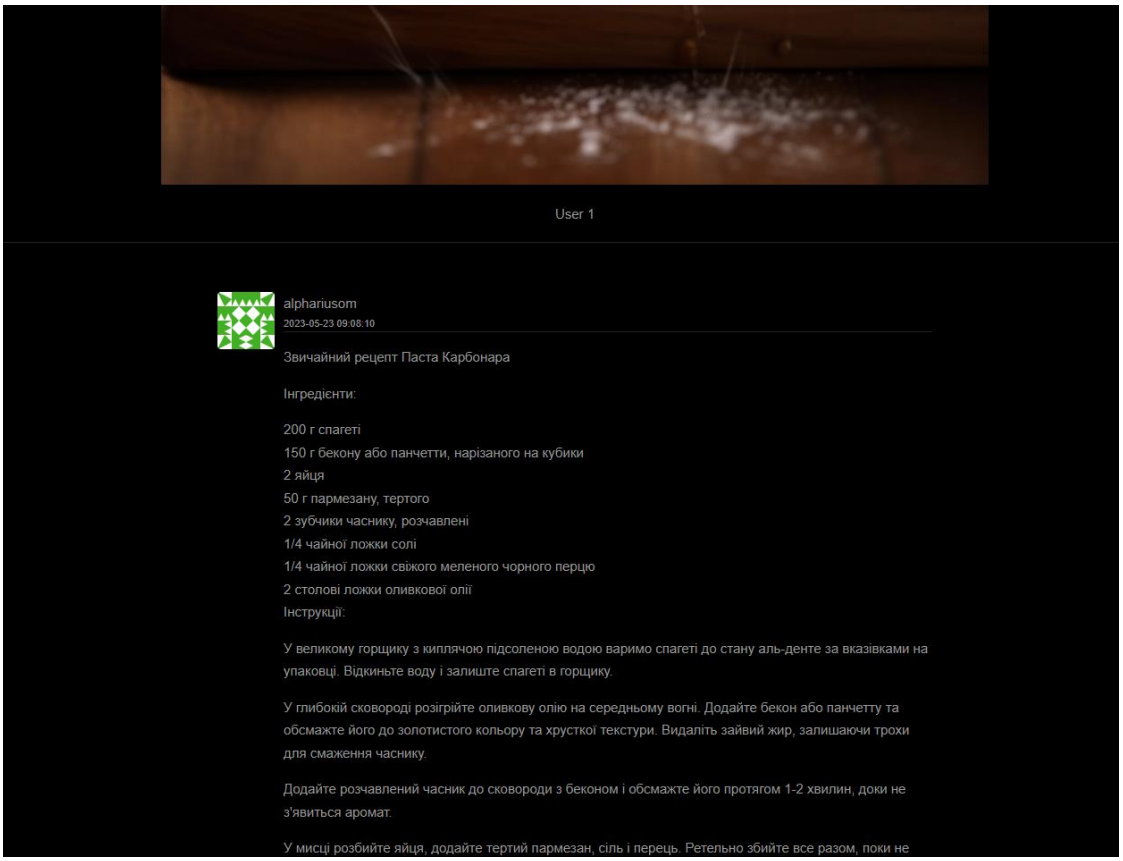

Рисунок. 3.21 – Галерея з роботами студентів.

За допомогою «+» студенти можуть доповнювати галерею та надсилати свої зображення (рис. 3.22).

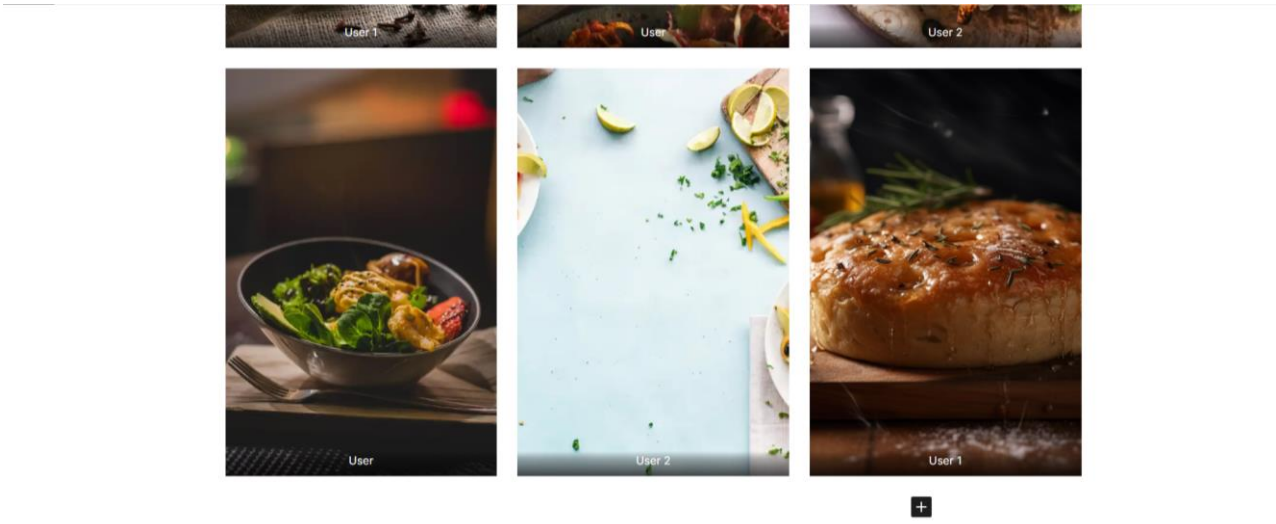

Рисунок. 3.22 – Галерея з роботами студентів.

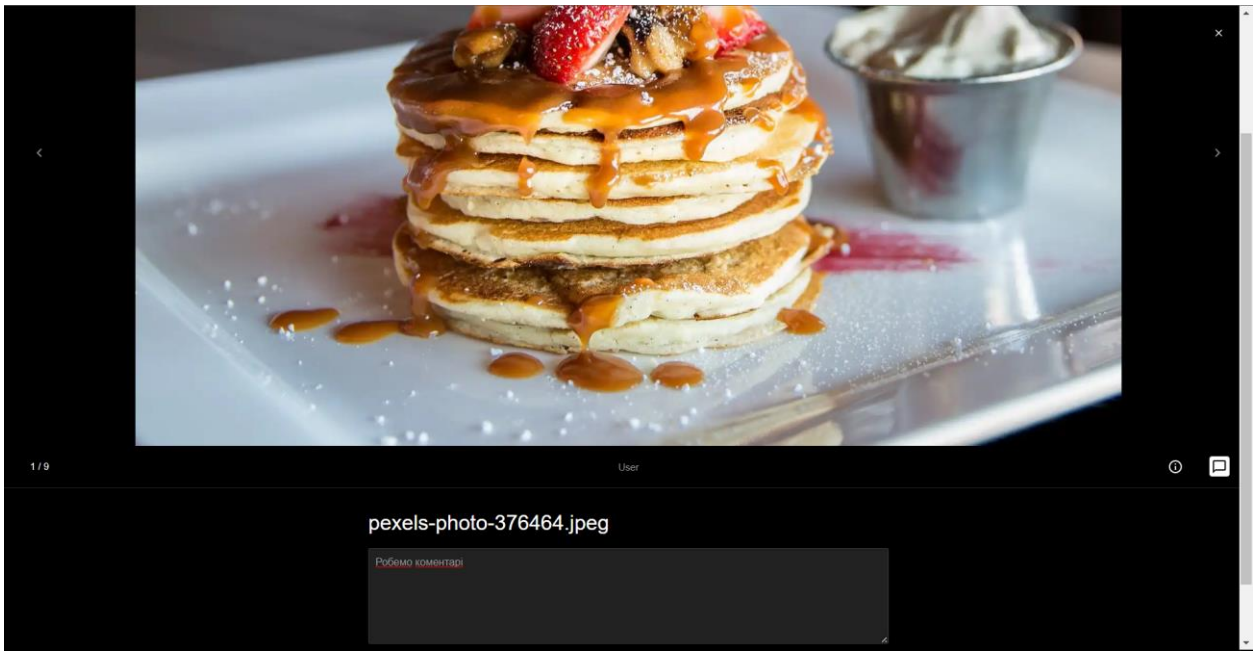

Рисунок. 3.23 – Створення коментаря

Далі створюється новий розділ «Про нас» та заповнюється контентом. Додається особиста інформація та можливість залишати відгуки про курси від User1 та User. Також footer з меню і можливістю підписатися на повідомлення, наступним була додана можливість відправляти платежі через Stripe акаунт.

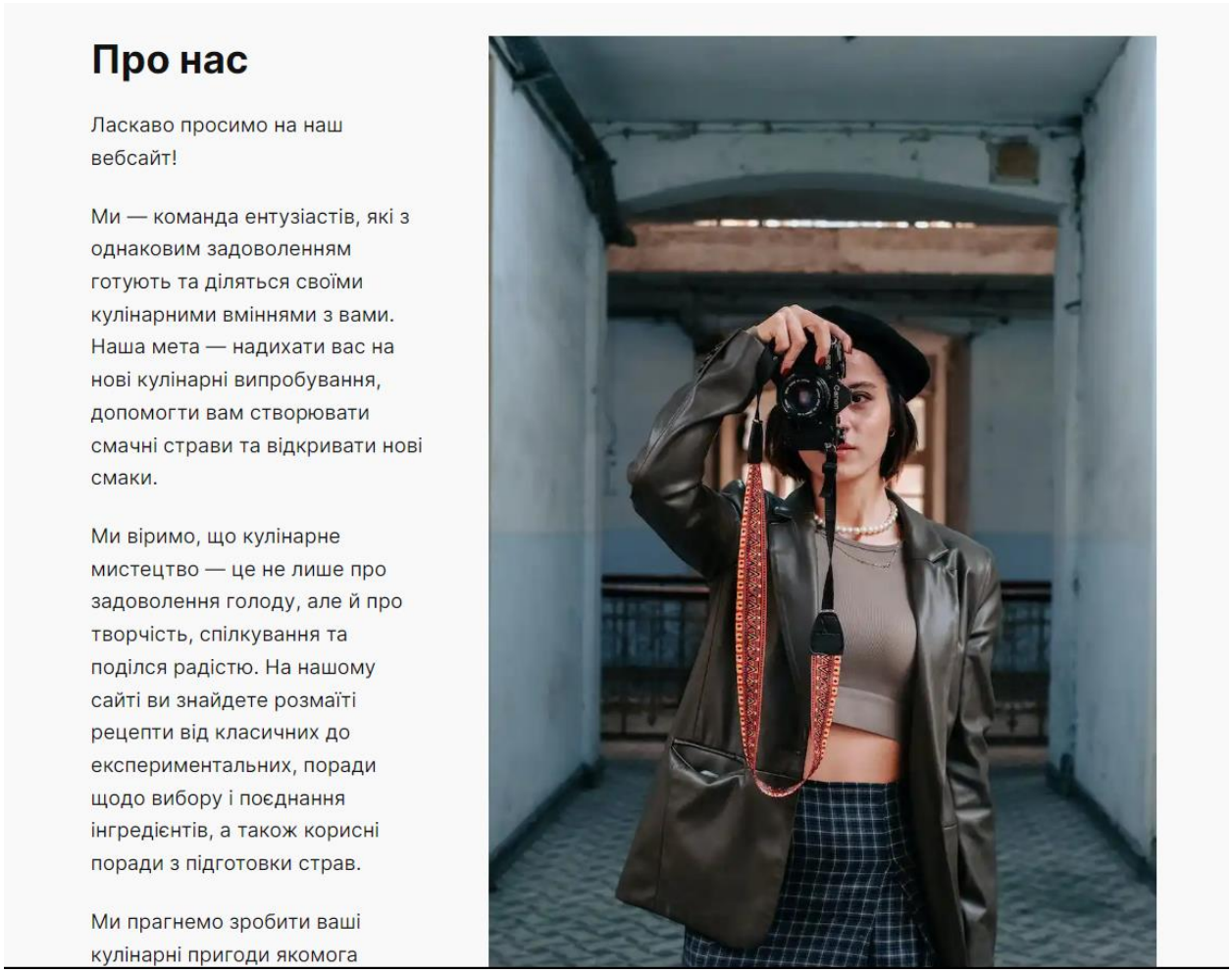

Рисунок. 3.24 – Про нас

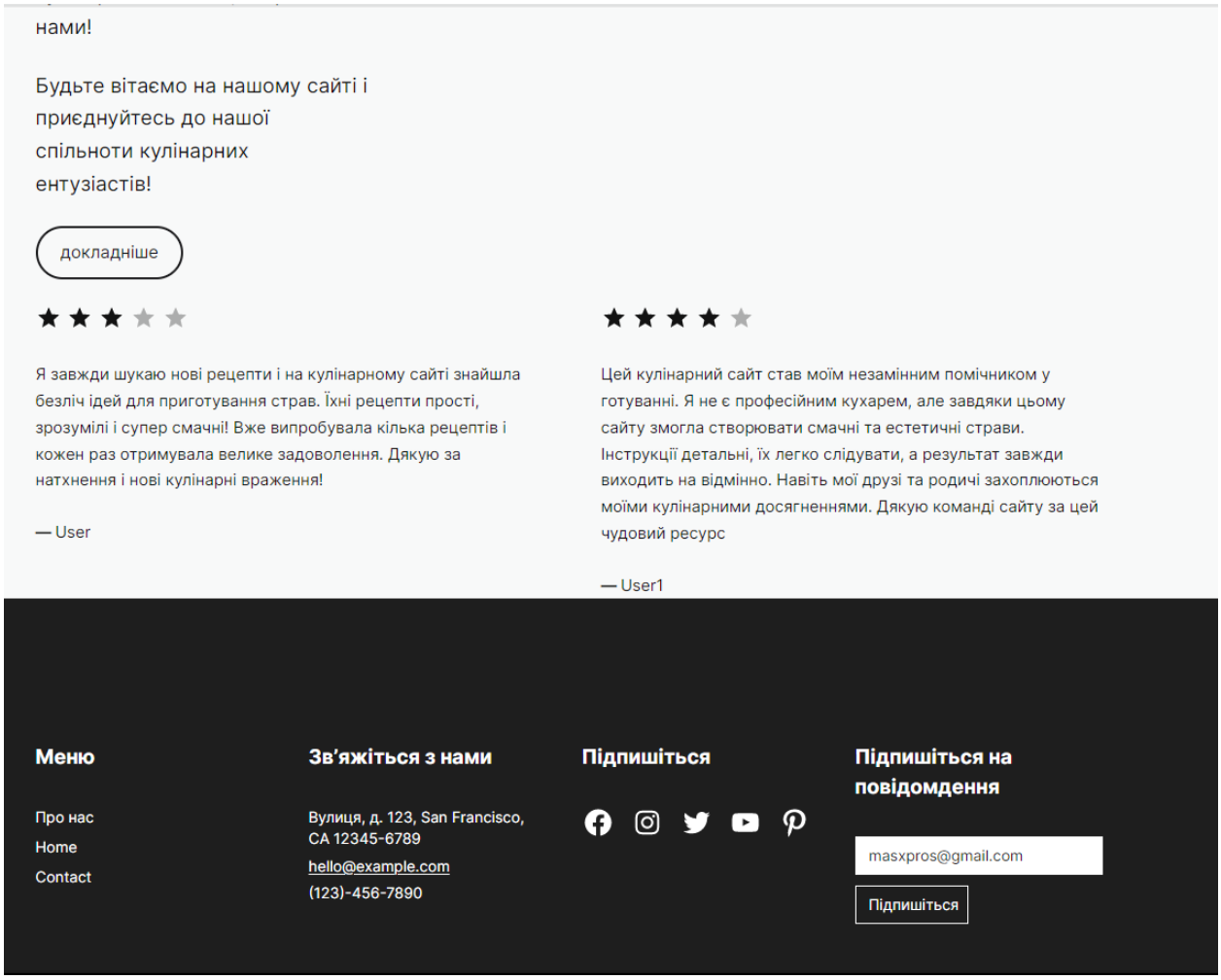

Рисунок. 3.25 – Створення коментаря

ВИСНОВКИ

Результатом виконання даної роботи була розроблена та реалізована інформаційна система керування навчанням кондитерській справі, що включає в себе сайт з відеоуроками з кулінарії, авторизаційною системою, можливістю завантаження контенту та функціоналом коментування і порад. Під час процесу розробки було використано такі технології, як MongoDB для бази даних, Express для створення серверної частини, React для розробки користувацького інтерфейсу та Node.js для серверного середовища.

В результаті проведених досліджень та розробки було досягнуто поставленої мети – створення ефективної та зручної інформаційної системи керування навчанням кондитерській справі. Користувачі мають можливість отримувати доступ до високоякісних відеоуроків, спілкуватись між собою, ділитись порадами та коментарями. Система дозволяє ефективно організувати процес навчання та підвищує доступність знань у сфері кондитерського мистецтва.

Основні задачі які були виконані під час розробки даної роботи:

1) Визначена мета та цілі проєкту, аналіз цільової аудиторії та її потреби.
2) Розроблена концепція сайту.
3) Розроблена структура бази даних та схеми сайту.
4) Розроблені функціональні модулі та компоненти сайту.
5) Розроблена адміністративна панель для управління контентом сайту.
6) Проведено тестування та усунення помилок (debugging).

В цілому, розроблена інформаційна система є значним кроком у напрямку поліпшення процесу керування навчанням кондитерській справі та сприяє розвитку у сфері кулінарії. Дана кваліфікаційна робота відображає важливість технологій у сучасному освітньому процесі та відкриває перспективи для подальших досліджень та розвитку в цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Девід Фленаган JavaScript // Діалектика, 2021. – 174-179 с.
2. «E-Learning and the Science of Instruction» by R. E. Clark та R. K. Mayer. 2021 – 450-455 с.
3. Lewis Coulson, Brett Jephson, Rob Larsen, Matt Park, Marian Zburlea The HTML and CSS Workshop // Packt Publishing, 2019. – 110, 552-557 с.
4. «The Design of Everyday Things» by D. A. Norman. 2021. – 202-215 с.
5. «Agile Software Development: Principles, Patterns, and Practice s» by R. C. Martin. 2020. – 215-217 с.
6. Node, Express - «Web Development with Node and Express» by E. Holmes та S. Armstrong. 2019. – 105, 185-187 с.
7. QA [Електронний ресурс] // URL: https://dou.ua/lenta/articles/qa-engineer-position/
8. MongoDB - «Mastering MongoDB» by S. Dangas.
9. Ben Frain Responsive Web Design with HTML5 and CSS // Packt Publishing, 2020. - 330 с.
10. CSS Tutorial - W3Schools: [Електронний ресурс]. URL: https://www.w3schools.com/css/
11. «The Impact of Online Learning on Student Performance» by M. S. Smith та R. W. Ferguson. 2022. 225, 244 с.
12. Reactjs [Електронний ресурс] // URL: https://legacy.reactjs.org/docs/getting-started.html
13. Nextjs [Електронний ресур] // URL: https://nextjs.org/docs
14. MongoDB [Електронний ресурс] // URL: https://www.mongodb.com/docs/
15. Nodejs [Електронний ресурс] // URL: https://nodejs.org/en/docs

Додаток А

PostController

```
import PostModel from '../models/Post.js';

export const getLastTags = async (req, res) => {
  try {
    const posts = await PostModel.find().limit(5).exec();

    const tags = posts
      .map((obj) => obj.tags)
      .flat()
      .slice(0, 5);

    res.json(tags);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось отримати теги',
    });
  }
};

export const getAll = async (req, res) => {
  try {
    const posts = await PostModel.find().populate('user').exec();
    res.json(posts);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не  вдалось отримати статьті',
    });
  }
};

export const getOne = async (req, res) => {
  try {
```

```
    const postId = req.params.id;

    PostModel.findOneAndUpdate(
      {
        _id: postId,
      },
      {
        $inc: { viewsCount: 1 },
      },
      {
        returnDocument: 'after',
      },
      (err, doc) => {
        if (err) {
          console.log(err);
          return res.status(500).json({
            message: 'Не  вдалось отримати статьті',
          });
        }

        if (!doc) {
          return res.status(404).json({
            message: 'Статья не найдена',
          });
        }

        res.json(doc);
      },
    ).populate('user');
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось отримати статьті',
    });
  }
};

export const remove = async (req, res) => {
  try {
    const postId = req.params.id;

    PostModel.findOneAndDelete(
      {
        _id: postId,
      },
      (err, doc) => {
        if (err) {
          console.log(err);
          return res.status(500).json({
```

```
            message: 'Не вдалось видалити статью',
          });
        }

        if (!doc) {
          return res.status(404).json({
            message: 'Статья не знайдена',
          });
        }

        res.json({
          success: true,
        });
      },
    );
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось отримати статьті',
    });
  }
};

export const create = async (req, res) => {
  try {
    const doc = new PostModel({
      title: req.body.title,
      text: req.body.text,
      imageUrl: req.body.imageUrl,
      tags: req.body.tags.split(','),
      user: req.userId,
    });

    const post = await doc.save();

    res.json(post);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось створити статью',
    });
  }
};

export const update = async (req, res) => {
  try {
    const postId = req.params.id;

    await PostModel.updateOne(
```

```
      {
        _id: postId,
      },
      {
        title: req.body.title,
        text: req.body.text,
        imageUrl: req.body.imageUrl,
        user: req.userId,
        tags: req.body.tags.split(','),
      },
    );

    res.json({
      success: true,
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось оновити статью',
    });
  }
};
```

Додаток Б

Index.js

```
import express from 'express';
import fs from 'fs';
import multer from 'multer';
import cors from 'cors';

import mongoose from 'mongoose';

import { registerValidation, loginValidation, postCreateValidation } from
'./validations.js';

import { handleValidationErrors, checkAuth } from './utils/index.js';

import { UserController, PostController } from './controllers/index.js';

mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => console.log('DB ok'))
  .catch((err) => console.log('DB error', err));

const app = express();

const storage = multer.diskStorage({
  destination: (_, __, cb) => {
    if (!fs.existsSync('uploads')) {
      fs.mkdirSync('uploads');
    }
    cb(null, 'uploads');
  },
  filename: (_, file, cb) => {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage });

app.use(express.json());
app.use(cors());
app.use('/uploads', express.static('uploads'));

app.post('/auth/login', loginValidation, handleValidationErrors,
UserController.login);
app.post('/auth/register', registerValidation, handleValidationErrors,
UserController.register);
app.get('/auth/me', checkAuth, UserController.getMe);
```

```
app.post('/upload', checkAuth, upload.single('image'), (req, res) => {
  res.json({
    url: `/uploads/${req.file.originalname}`,
  });
});

app.get('/tags', PostController.getLastTags);

app.get('/posts', PostController.getAll);
app.get('/posts/tags', PostController.getLastTags);
app.get('/posts/:id', PostController.getOne);
app.post('/posts', checkAuth, postCreateValidation, handleValidationErrors,
PostController.create);
app.delete('/posts/:id', checkAuth, PostController.remove);
app.patch(
  '/posts/:id',
  checkAuth,
  postCreateValidation,
  handleValidationErrors,
  PostController.update,
);

app.listen(process.env.PORT || 4444, (err) => {
  if (err) {
    return console.log(err);
  }

  console.log('Server OK');
});
```

Додаток В

Post.js

```
import mongoose from 'mongoose';

const PostSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
    },
    text: {
      type: String,
      required: true,
      unique: true,
    },
    tags: {
      type: Array,
      default: [],
    },
    viewsCount: {
      type: Number,
      default: 0,
    },
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true,
    },
    imageUrl: String,
  },
  {
    timestamps: true,
  },
);

export default mongoose.model('Post', PostSchema);
```

Додаток Г

User.js

```
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema(
  {
    fullName: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    passwordHash: {
      type: String,
      required: true,
    },
    avatarUrl: String,
  },
  {
    timestamps: true,
  },
);

export default mongoose.model('User', UserSchema);
```

Пояснення

CI/CD: Continuous Integration/Continuous Deployment - це методологія розробки програмного забезпечення, яка передбачає автоматичну інтеграцію змін у коді та безперервну поставку оновлень до продуктивного середовища.

UX/UI: User Experience/User Interface - це поняття, яке охоплює дизайн та взаємодію користувачів з продуктом або інтерфейсом.

FRONT: веб-фронтенд – це частина програмного забезпечення, яка відповідає за відображення інформації та взаємодію з користувачем у браузері.

BACKEND: веб-бекенд – це частина програмного забезпечення, яка відповідає за обробку запитів, доступ до бази даних та бізнес-логіку за кулісами.

GIT: розподілена система керування версіями, яка дозволяє зберігати та керувати кодом проекту.

API: Application Programming Interface – це набір правил та протоколів, які визначають, які функції та можливості доступні для взаємодії з певним програмним продуктом чи сервісом.

SSL: Secure Socket Layer – це криптографічний протокол, який забезпечує безпеку передачі даних через Інтернет шляхом шифрування з'єднання.

SLA: Service Level Agreement – це угода між постачальником послуг та клієнтом, в якій встановлюються параметри рівня обслуговування, такі як час реакції, доступність тощо.

IP: Internet Protocol – це протокол, який визначає правила для передачі даних через мережу Інтернет.

HTTP: Hypertext Transfer Protocol – це протокол передачі гіпертекстових документів через мережу.

HTTPS: Hypertext Transfer Protocol Secure – це захищений протокол передачі даних через мережу, який використовує SSL для шифрування з'єднання.

DOM: Document Object Model – це стандартна модель програмування, яка представляє HTML-документ як дерево об'єктів, що дозволяє маніпулювати та змінювати його з допомогою скриптів.

VS: Visual Studio – це інтегроване середовище розробки (IDE), яке використовується для створення програмного забезпечення.

SEO: Search Engine Optimization – це процес оптимізації веб-сайту з метою покращення його видимості та рейтингу у пошукових системах.

КЕШ: кешування – це техніка збереження проміжних даних, що дозволяє прискорити доступ до них у майбутньому.

UNIT-TEST: юніт-тестування – це процес виконання автоматичних тестів на найменших окремих одиницях коду (юнітах) з метою перевірки їх правильності та функціональності.

NPM: Node Package Manager – це менеджер пакетів для платформи Node.js, який дозволяє швидко та зручно установлювати, оновлювати та керувати залежностями проекту.

CLL: Call Level Interface – це програмний інтерфейс, який дозволяє програмам взаємодіяти з системою управління базами даних за допомогою SQL-запитів.

SCOPUS: SCOPUS – це міжнародна база даних наукових статей та рецензованих матеріалів конференцій.

QA: Quality Assurance - це процес контролю та забезпечення якості програмного забезпечення, який включає тестування, аналіз помилок та вдосконалення процесів розробки.

ТЗ: технічне завдання – це документ, що визначає вимоги до розробки програмного продукту або проекту.

AWS: Amazon Web Services – це платформа хмарних обчислень, яка надає різноманітні послуги, такі як обчислення, зберігання даних, бази даних, мережі та інші.

CRUD: Create, Read, Update, Delete – це базові операції для управління даними в базі даних або системі керування вмістом.

JWT: JSON Web Token – це стандарт для створення місцевих токенів автентифікації, які використовуються для передачі інформації про користувача між сторонами у форматі JSON.