МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «***Розробка клієнтської частини проєкту освітньої платформи***»

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Матвійчук Максим Михайлович

Керівник: ***Красюк Б.В., викладач кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента
Матвійчука Максима Михайловича

1. Тема роботи: "***Розробка клієнтської частини проєкту освітньої платформи"***

Керівник роботи: Красюк Богдан Віталійович, викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: аналіз конкурентів на ринку, репозиторій проєкту taxonomy, репозиторій проєкту highstorm.

4. Перелік завдань, які належить виконати: реалізувати клієнтську частину застосунку використовуючи Shadcn, реалізувати функціонал для роботи з подіями, користувачами. Розробити механізм взаємодії з API для клієнтської частини.

5. Перелік графічного матеріалу: рисунки.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Красюк Б. В.	01.12.2023	01.12.2023
2	Красюк Б. В.	01.12.2023	01.12.2023
3	Красюк Б. В.	01.12.2023	01.12.2023

7. Дата видачі завдання: 01.12.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми проєкту	до 31.10.2023 р.	
2	Постановка технічного завдання.	до 01.12.2023 р.	
3	Ознайомлення з документацією	до 10.12.2023 р.	
4	Написання розділу 1	до 01.02.2024 р.	
5	Написання розділу 2	до 01.03.2024 р.	
6	Написання розділу 3	до 01.04.2024 р.	
7	Тестування застосунку	до 20.04.2024 р.	
8	Виправлення помилок	до 01.05.2024 р.	
9	Попередній захист та перевірка на рівень унікальності кваліфікаційної роботи/проєкту	до 31.05.2024 р.	
10	Здача кваліфікаційної роботи на кафедру	31.05.2024 р.	

Студент: _____ Максим МАТВІЙЧУК

Керівник кваліфікаційної роботи: _____ Богдан КРАСЮК

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: *Розробка клієнтської частини проєкту освітньої платформи*

Автор: *Матвійчук Максим Михайлович*

Науковий керівник: *Красюк Б.В., викладач кафедри ЕММІТ*

Захищена «……»…… 2024 року.

Пояснювальна записка до кваліфікаційної роботи: *59 с., 34 рис., 1 додаток, 30 джерел..*

Ключові слова: *навчальна платформа, вебзастосунок, unit-тестування, meta-framework*

Короткий зміст праці:

Ця кваліфікаційна робота присвячена розробці клієнтської частини проєкту освітньої платформи, ціль якої покращити досвід користування та надати можливість швидкої взаємодії з освітніми подіями. Описана проблематика актуальна, бо через популяризацію освітнього контенту, зростає й попит у платформах, який його надають. У першому розділі роботи йдеться про дослідження та аналіз наявних рішень, виділення їх основних ознак. Опис інструментів для організації роботи. Другий розділ містить у собі опис технологічного стека та структурного рішення застосунку. У третьому розділі описано поетапну реалізацію застосунку та розглянуто такі аспекти як взаємодія із серверним API, використання TypeScript та метафреймворку Next.js. Розглянуто альтернативні технічні рішення та перспективи проєкту на майбутнє. Робота завершується висновками у яких підсумовано усі виконані кроки для реалізації застосунку. Як результат отримано функціональний додаток, який відповідає вимогам клієнтської частини освітньої платформи.

_____

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of the client part of the educational platform*

Author: *Maksym Matviichuk*

Scientific supervisor: *Krasiuk B., lecturer at DEMMIT*

Defensed «……»…… of 2024.

Explanatory note to the qualification work: *59 p., 34 pic., 1 attachment, 30 sources.*

Keywords: *learning platform, web application, unit testing, meta-framework.*

Summary of the paper:

This qualification work is devoted to the development of the client side of an educational platform project, which aims to improve the user experience and provide an opportunity to quickly interact with educational events. The described problem is relevant because due to the popularization of educational content, the demand for platforms that provide it is growing. The first section of the paper deals with the research and analysis of existing solutions, highlighting their main features. It also describes tools for organizing work. The second section describes the technological stack and structural solution of the application. The third section describes the step-by-step implementation of the application and considers such aspects as interaction with the server API, the use of TypeScript and the Next.js meta-framework. Alternative technical solutions and future prospects of the project are considered. The paper ends with conclusions summarizing all the steps taken to implement the application. As a result, a functional application that meets the requirements of the client side of the educational platform has been obtained.

_____

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві, де технології надають нові можливості у всіх сферах життя, освіта стає однією з ключових галузей, яка визначає розвиток суспільства та формує конкурентоспроможних фахівців. З метою підвищення доступності та ефективності освіти, виникає потреба в розробці та вдосконаленні освітніх платформ. Однією з ключових складових таких платформ є їхній клієнтський інтерфейс, який визначає зручність взаємодії користувачів з освітнім контентом.

Актуальність даної роботи обумовлена стрімким розвитком сучасних технологій та їхнім впливом на освітні процеси. Зростання популярності дистанційного навчання та онлайн-освіти підкреслює необхідність створення ефективних та зручних інструментів для взаємодії студентів з навчальним контентом. Розробка клієнтської частини освітньої платформи стає ключовим етапом у розв'язання актуальної проблеми. У сучасних умовах цифровізації, освітні платформи повинні не лише забезпечувати доступ до знань, але й надавати можливості для інтерактивного навчання, яке враховує індивідуальні потреби кожного користувача.

Метою кваліфікаційної роботи є розробка повнофункціонального застосунку для зручного перегляду та взаємодії з подіями на навчальній платформі. Такий застосунок дозволить студентам і викладачам ефективно комунікувати, обмінюватися інформацією та отримувати доступ до навчальних матеріалів у зручному форматі. Розробка такого рішення сприятиме підвищенню якості освіти, забезпечуючи більш інклюзивний та персоналізований підхід до навчання. Об’єктом дослідження є клієнтська частина освітньої платформи. Предметом — оптимізація інтерфейсу для забезпечення ефективного користування освітнім контентом.

Для досягнення мети поставлено кілька завдань, що виступатимуть у ролі плану виконання кваліфікаційної роботи:

1) Здійснити аналіз сучасних клієнтських рішень на ринку EdTech. Це дозволить визначити найкращі практики та інновації, що можуть бути застосовані у розробці власного рішення.

2) Планування функціонала вебзастосунка. На цьому етапі буде визначено ключові функції, які повинні бути реалізовані для забезпечення зручності та ефективності використання платформи.

3) Розробка клієнтської частини рішення. Цей етап включатиме створення інтерфейсу користувача, що буде інтуїтивно зрозумілим.

4) Тестування вебзастосунка та подальша підтримка. Завершальний етап включає перевірку роботи застосунку, виявлення та виправлення помилок.

Таким чином, розробка освітньої платформи з ефективним клієнтським інтерфейсом є актуальним завданням, що відповідає потребам сучасного суспільства в якісній та доступній освіті. Виконання даної роботи сприятиме розвитку освітніх технологій та підвищенню рівня знань і навичок.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Постановка проблеми

У сучасному освітньому середовищі виникає нагальна потреба в покращенні якості та доступності освіти. Зокрема, проблемою є нестача зручних та ефективних інструментів для взаємодії студентів з навчальним контентом на освітніх платформах. Клієнтська частина цих платформ визначає користувацький досвід, і тому важливо розробити версію цієї частини, що відповідала б основним вимогам ергономіки та стандартам веброзробки.

Освітня платформа - це цифровий інструмент, який надає можливості для навчання, сприяє обміну знаннями та взаємодії між вчителями та учнями.

Клієнтська частина - це та частина освітньої платформи, яка взаємодіє безпосередньо з користувачем. Вона містить в собі вебінтерфейс, через який користувачі взаємодіють із платформою.

EdTech — це скорочення від "Educational Technology", що в українському варіанті може перекладатися як "освітні технології". EdTech використовується для опису використання технологій у сфері освіти з метою покращення навчального процесу та забезпечення ефективної та доступної освіти. В Україні EdTech також активно розвивається. З'являються нові українські EdTech-стартапи та платформи, які пропонують інноваційні рішення для освітніх потреб. Як приклад одного з випадків, що збільшили актуальність EdTech, це пандемія у 2020 році, через яку, багато шкіл по всьому світу були змушені закритися, що призвело до того, що все більше учнів початкових класів беруть участь в онлайн-навчанні, а студенти університетів реєструються на онлайн-курси для забезпечення дистанційного навчання. Сфера EdTech продовжує стрімко зростати тому створення продукту у ній є доволі перспективним.

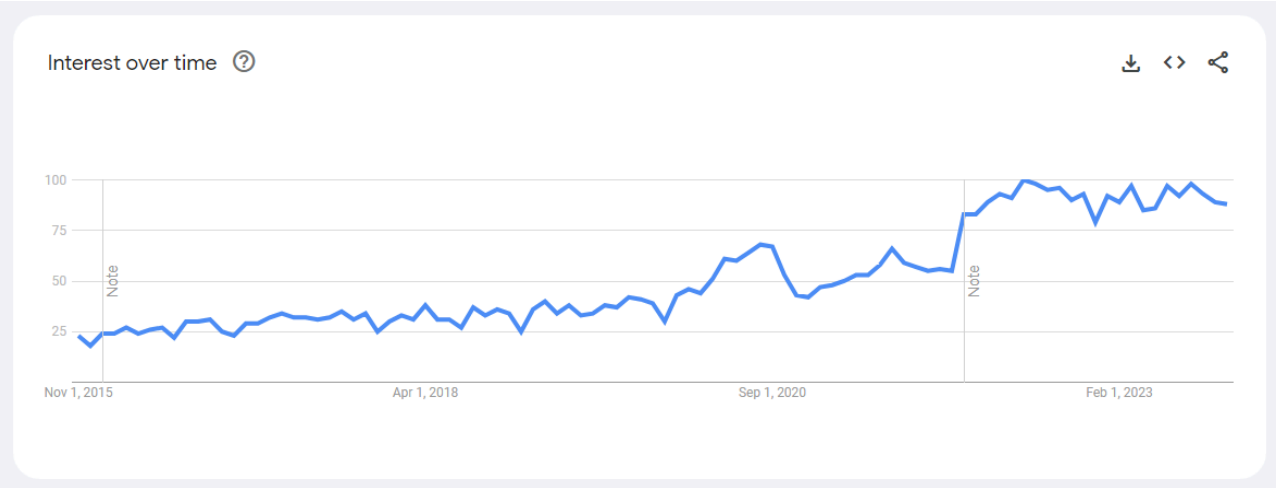

Рис. 1.1. Графік популярності сфери EdTech

Джерело: trends.google.com

1.2. Аналіз продуктів

На ринку освітніх платформ існує багато популярних рішень та їх реалізацій, але основними є наступні продукти: Udemy, Projector Institute, coursera, freeCodeCamp.

Udemy (Рис. 1.2.) це освітня платформа, яка надає широкий спектр онлайн-курсів у різних сферах від вивчення програмування до мистецтва та бізнесу. Основний функціонал включає можливість вибору інструкторів, доступ до відеоуроків та можливість отримання сертифікатів після успішного завершення курсу. Також можлива більш тісна взаємодія з інструктором. Отже, головна ідея полягає у придбанні курсів за певну ціну та їх проходження.

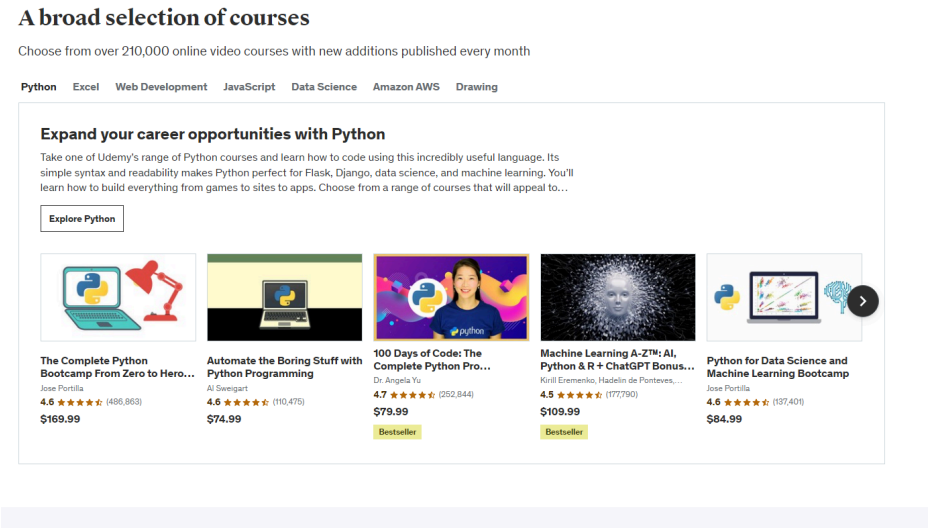

Рис. 1.2. Вигляд UI Udemy

Джерело: Udemy.com

Projector (Рис. 1.3.) — освітній продукт який виступає як приватний інститут з завчасно підготовлених курсів з визначеною тривалістю. Є як курси середньої тривалості, так і “професіуми” які розраховані на рік. Курси мають свою структуру. Ключовими відмінностями є:

1. Курси підготовлені викладачами
2. Курси може додавати тільки команда продукту
3. Взаємодія з викладачами відбувається поза сайтом

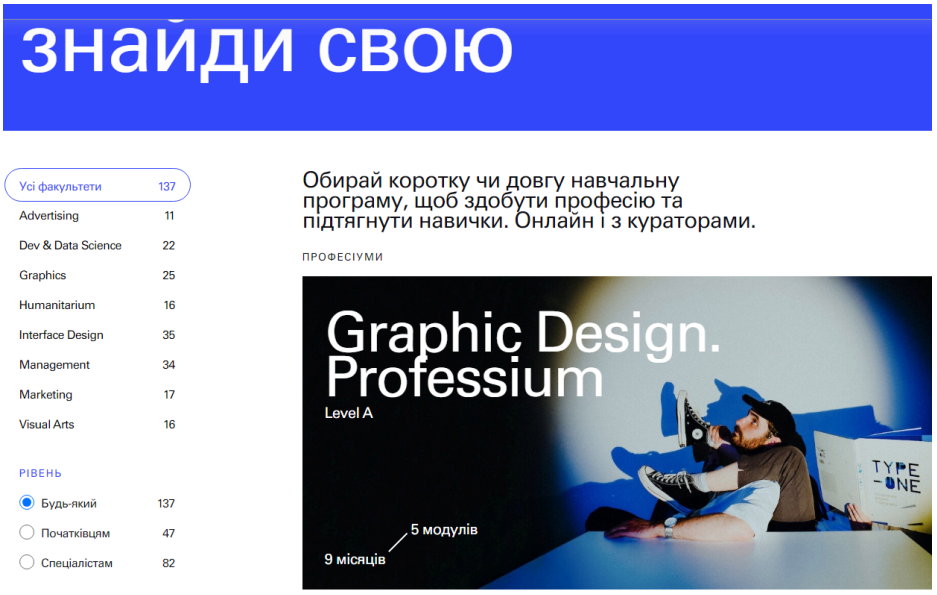

Рис. 1.3. Інтерфейс сайту projector institute

Джерело: prjctr.com

Наступне популярне рішення на ринку це Coursera. Coursera — це освітня платформа, що надає можливість онлайн-курси від університетів та організацій з усього світу. Основний функціонал включає доступ до широкого спектра курсів у різних галузях, можливість вивчення матеріалів за власним графіком, інтерактивні вправи та оцінювання від викладачів. Coursera також надає можливість отримати сертифікати після успішного завершення курсів та працює на засадах гнучкого та доступного навчання для студентів з усього світу.

Основними відмінностями є те що курси на Coursera розробляють викладачі з університетів та організацій у всьому світі. Більшість курсів має академічний підхід та може вести до отримання сертифіката від університету. Також пропонує серйозні академічні курси з гнучким графіком, і більшість вимагає плати за доступ до відеоматеріалів та завдань.

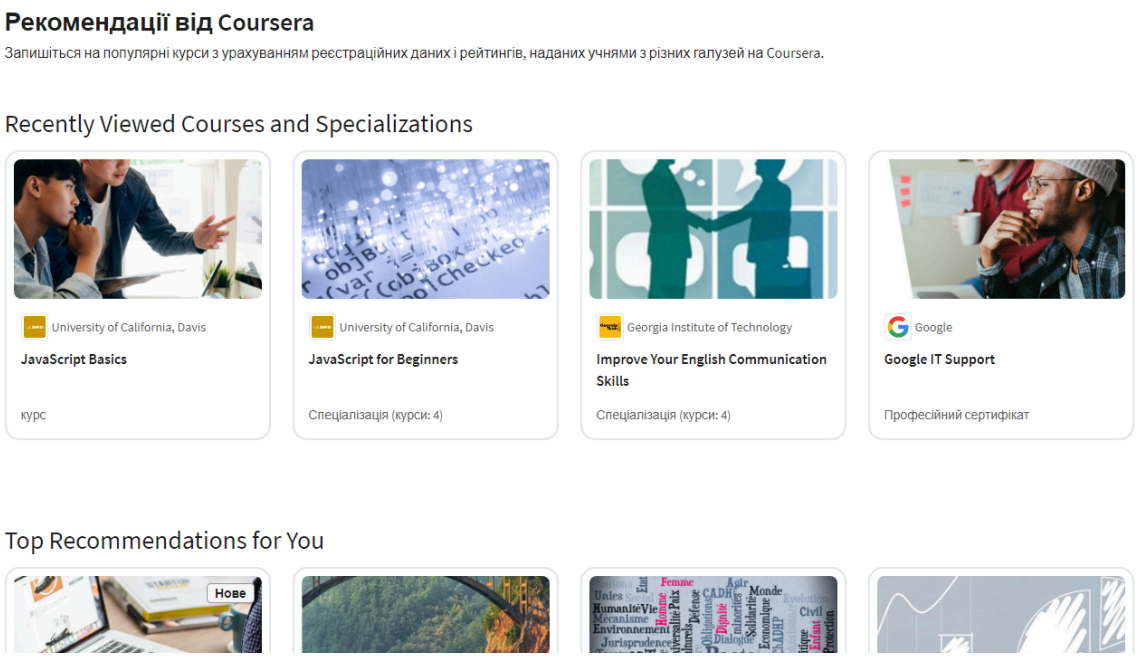

Рис. 1.4. Панель вибору курсів на coursera

Джерело: coursera.org

Ринок характеризується конкурентним середовищем, проте існує потенціал для його подальшого розвитку. За власним підходом маємо наступні цілі:

1. Створення можливості додаткового доходу для студентів та викладачів поза навчальним процесом (основною роботою).
2. Створення більш гнучкого формату навчання для точкових знань/інструментів.

Після порівняння наявних рішень можемо виділити функції які можуть бути перевагами у схожих продуктів:

Загальний функціонал

1. Free пропозиція зрозуміти професію чи обрати курс
2. Консультація, якщо не впевнені в виборі
3. Підписатися на курс (сповіщення)

Структура

1. Картка продукту: відгуки, часті питання, приклади результатів студентів
2. Взаємодія поза системою (пошта, соцмережі, застосунки для спільноти)
3. Взаємодія в системі

Також основною сутність у нашому проєкті буде не курс, а подія, до якої може додатись курс. Тобто різноманітність контенту платформи не буде обмежена лише курсами по відповідних тематиках.

Отже, після аналізу конкурентів та виділивши відмінності, обґрунтувавши цілі проєкту — нашому застосунку необхідно реалізувати схожі та деякі відмінні інтерфейсні рішення.

1.3. Організація роботи

Для спрощення процесу роботи та створення організації контролю проєкту було вирішено використати такі інструменти як Git, GitHub та Postman.

1.3.1. GitHub

GitHub — це вебплатформа для спільної роботи над програмним забезпеченням, яка базується на системі контролю версій Git. Вона надає інструменти для спільної роботи над проєктами, відстеження змін у коді,

управління задачами та автоматизації процесів розробки. При роботі з GitHub варто виділити декілька основних елементів взаємодії.

1. Pull-Request: це механізм в GitHub, що дозволяє розробникам запропонувати зміни в коді та обговорити їх з іншими членами команди перед їх об'єднанням з основною гілкою проєкту. Pull-Request охоплює опис змін, списки змін у коді та може включати коментарі для обговорення.
2. Issues: це інструмент для відстеження завдань, помилок, ідей та інших питань, що виникають під час розробки програмного забезпечення. Користувачі можуть створювати Issues, призначати їх собі або іншим учасникам проекту, обговорювати та вирішувати їх.
3. Actions (GitHub Actions): це інструмент для автоматизації різних процесів в репозиторії GitHub. Вони можуть бути налаштовані для виконання автоматичних тестів, збирання та розгортання програмного забезпечення, виконання регулярних завдань тощо. GitHub Actions дають можливість створювати власні автоматичні робочі процеси з використанням скриптів у синтаксисі YAML.

1.3.2. Git

Git — це розподілена система керування версіями, призначена для відстеження змін у файлах програмного забезпечення та спільної роботи над ними. Вона дозволяє розробникам ефективно керувати кодом, відстежувати його історію, створювати гілки для різних функціональностей та об'єднувати їх. Git надає можливості для роботи як локально, так і з використанням віддалених репозиторіїв, що робить його ідеальним інструментом для розробки програмного забезпечення в команді.

1.3.3. Postman

Postman — це інструмент для розробки, тестування та документування API. Він надає зручне середовище для взаємодії з API, включаючи можливість відправляти HTTP-запити, перевіряти відповіді, автоматизувати тестові сценарії

та створювати колекції запитів для подальшого використання. Використання Postman для тестування API включає наступні кроки:

1. Створення нового запиту: спочатку потрібно створити новий запит і вказати метод (GET, POST, PUT, DELETE тощо) та URL адресу API, з якою ви хочете взаємодіяти.
2. Налаштування параметрів запиту: додавання параметрів, заголовків, тіла запиту тощо відповідно до вимог API.
3. Відправлення запиту: після налаштування запиту ви можете відправити його на сервер API та переглянути відповідь.
4. Перевірка відповіді: після отримання відповіді ви можете перевірити її на предмет правильності та відповідність очікуваним результатам. Postman надає зручні інструменти для цього, такі як перегляд тіла відповіді, перевірка статусу коду, а також можливість написання скриптів для автоматичної перевірки відповіді.
5. Автоматизація тестових сценаріїв: Postman дозволяє створювати колекції запитів та автоматизувати тестові сценарії за допомогою колекцій та середовищ. Ви можете налаштувати запити на виконання послідовності дій та автоматично перевіряти їх результати.

Висновки до розділу 1

У цьому розділі було описано проблему та розказано про популярність такої сфери як EdTech. Проаналізовано основних гравців у цій сфері, представлено їх вигляд та переваги. Також розглянуто основні інструменти для організації роботи — збереження коду проєкту та його зручне розміщення, контроль версій. Показано інструмент для роботи з backend API надалі.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Опис технологічного стека

Бібліотеки й технології, що були використані під час розробки клієнтської частини.

2.1.1. React.js

React.js — це JavaScript бібліотека для створення користувацьких інтерфейсів вебдодатків. Вона базується на компонентній архітектурі та пропонує декларативний підхід до опису UI. Подібно до багатьох інших фреймворків JavaScript, React.js використовує Virtual DOM для ефективної роботи з реальним DOM-деревом. Це дозволяє оптимізувати процес оновлення сторінки, ігноруючи ті частини, які залишилися без змін. У версії React.js 17, було впроваджено підтримку TypeScript, що надає розробникам переваги статичної перевірки типів, що сприяє підвищенню якості та надійності коду.

React пропонує кілька підходів до використання фреймворку — від інтеграції з наявним проєктом до повного створення вебзастосунка з використанням елементів екосистеми React, таких як Redux для глобального стану, серверного рендерингу або навіть створення PWA. Важливою особливість React є його формат для написання коду, який має назву JSX, й усі файли які його використовують матимуть розширення .jsx або .tsx(при використанні TypeScript).

JSX — це розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код безпосередньо в JavaScript. JSX робить код більш зрозумілим та легким для читання, оскільки забезпечує можливість створювати компоненти UI у вигляді вкладених елементів, подібних до HTML-тегів. Під час компіляції JSX перетворюється на виклики функцій React.createElement, які створюють віртуальні DOM-елементи. Це дозволяє React виконувати ефективне управління компонентами та їх оновленнями.

2.1.2. Next.js

Next.js — це метафреймворк для розробки вебдодатків на базі React.js. Він надає швидкий і потужний спосіб створення високопродуктивних додатків, включаючи односторінкові додатки (SPA), статичні сайти (SSG) та серверно-рендерні додатки (SSR). Основні особливості Next.js включають:

1. Універсальність: Next.js підтримує рендеринг як на стороні клієнта, так і на стороні сервера. Це означає, що ви можете рендерити сторінки на сервері під час першого завантаження, а потім використовувати клієнтський рендеринг для подальших відвідувань.
2. Статичне генерування: Next.js дозволяє статично генерувати сторінки в період збірки проєкту. Це підходить для статичних сайтів або сторінок, які майже не змінюються.
3. Серверний рендеринг: Next.js підтримує рендеринг на стороні сервера, що дозволяє генерувати HTML на сервері для кожного запиту. Це дозволяє оптимізувати процес рендерингу та покращує індексацію вебсторінок пошуковими системами.

SSR дозволяє створювати статичні сторінки з динамічним вмістом, що поліпшує індексацію сторінок пошуковими системами та забезпечує кращий досвід користувача, особливо на повільних мережах або пристроях. Візуально приклад роботи Server-Side-Rendering можна зобразити наступним чином

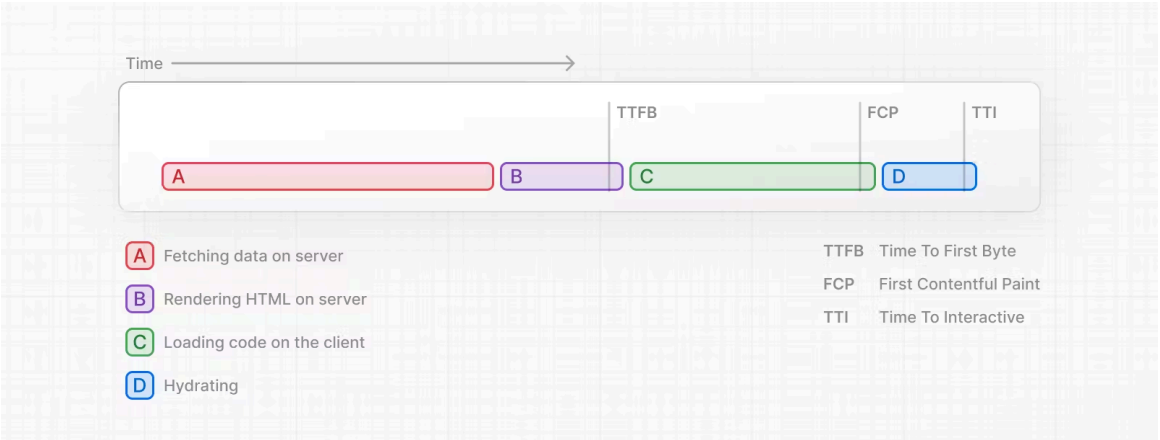

Рис. 2.1. робота SSR

Джерело: документація рендерингу Next.js

4. Розширюваність: Next.js має широкий набір розширень та плагінів, які дозволяють розширювати його функціональність та пристосовувати під конкретні потреби проекту.
5. Роутинг: фреймворк надає простий та зручний маршрутизатор, який дозволяє визначати шляхи та обробляти запити користувачів.
6. Оптимізація проекту: Next.js має вбудовану оптимізацію для швидкості завантаження сторінок, автоматичну підтримку мінімізації та оптимізацію зображень, а також можливість перед передачі даних для покращення швидкодії.

Next.js поділяє компоненти на два типи: серверні та клієнтські. Залежно від обраного типу компонента буде використовуватись відповідний вид рендерингу. Щоб використати клієнтський рендеринг потрібно додати директиву “use client”. Це означає, що, визначивши у файлі “use client”, всі інші модулі, імпортовані до нього, включно з дочірніми компонентами, вважаються частиною клієнтського компонента. Таким чином нам стає доступним використання клієнтських функцій як onClick чи useState. До того ж використання цих функцій у серверному компоненті призведе до помилки, як це зображено на наступній схемі(див. рис. 2.2.).

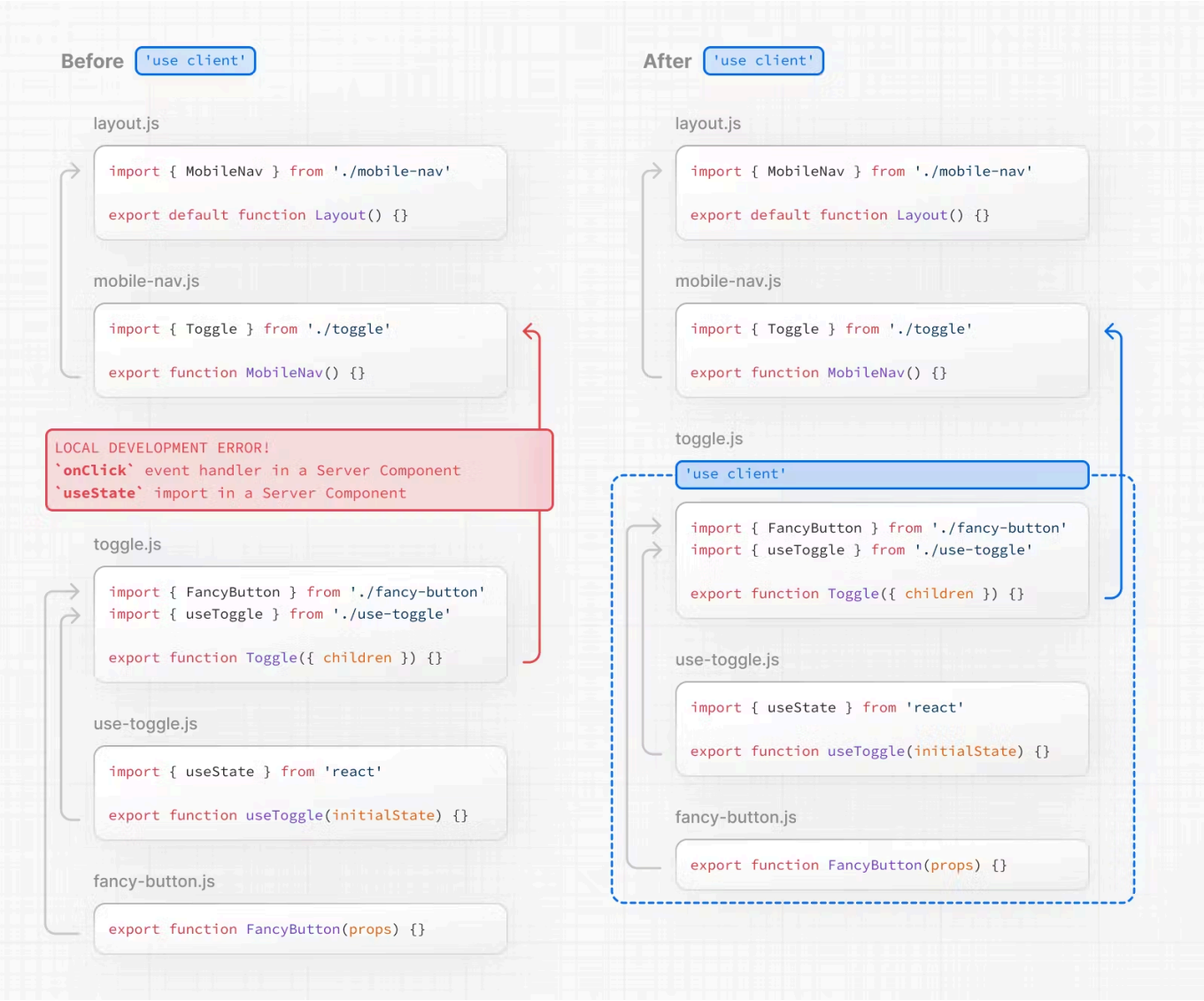

Рис. 2.2. Різниця при використанні директиви “use client”

Джерело: документація рендерингу Next.js

Варто також зазначити, що як модульний пакувальний Next використовує свою розробку - Turbopack. Це інкрементний бандлер, оптимізований для JavaScript і TypeScript, написаний на Rust творцями webpack і Next.js з Vercel.

Секрет продуктивності Turbopack має дві складові: високооптимізований машинний код та низькорівневий механізм інкрементних обчислень, який дозволяє кешувати до рівня окремих функцій. Одного разу виконавши завдання, Turbopack більше ніколи його не виконує.

2.1.3. TypeScript

TypeScript — це надмножина JavaScript, яка додає статичну типізацію до мови.Це означає, що ви можете чітко вказувати типи змінних, функцій та інших елементів коду, що може допомогти покращити надійність та читабельність коду. Існує багато переваг використання TypeScript з React.js:

1. Покращена надійність:
    a. Статична типізація TypeScript може допомогти виявити помилки під час компіляції, а не під час виконання, що може заощадити час і зусилля на налагодження.
    b. TypeScript може допомогти гарантувати, що ви використовуєте правильні типи для даних, що може допомогти уникнути помилок під час виконання.
2. Покращена читабельність:
    a. TypeScript може зробити код більш читабельним, додаючи чітку інформацію про типи до змінних, функцій та інших елементів коду.
    b. Це може бути особливо корисно для великих проєктів з кількома розробниками.
3. Автодоповнення та інструменти:
    a. TypeScript має багату екосистему інструментів, які можуть допомогти писати та підтримувати свій код.
    b. До цих інструментів належать автодоповнення, перевірки типів та лінтери.
4. Легша інтеграція з іншими бібліотеками:
    a. TypeScript може полегшити інтеграцію коду React.js з іншими бібліотеками, які також використовують TypeScript.
    b. Це може заощадити час і зусилля на написання адаптерів або обхідних шляхів.

2.1.4. Prettier

Prettier — це форматор коду, який можна швидко та гнучко налаштувати для роботи майже з будь-яким сучасним фреймворком(від Angular до Svelte).

Метою його використання є підвищення якості та читабельності коду, оскільки задавши правила та відповідну комбінацію клавіш Prettier автоматично розмістить код для найкращого відображення.

2.1.5. ESlint

ESLint — це інструмент для аналізу та виявлення потенційних проблем у JavaScript коді з метою поліпшення якості та стилю коду. Він дозволяє визначати правила, які визначають, який код вважається прийнятним, а який — ні, і надає засоби для автоматичної перевірки цих правил під час розробки. ESLint допомагає підтримувати стандарти коду, зменшувати кількість помилок та підвищувати загальну якість програмного забезпечення.

2.1.6. NPM

NPM (Node Package Manager) — це пакетний менеджер для платформи Node.js, призначений для управління залежностями проєктів та дистрибуції пакетів JavaScript. Він дозволяє встановлювати, оновлювати та видаляти пакети з локального або віддаленого репозиторію npm. npm є невіддільною частиною екосистеми Node.js і використовується мільйонами розробників по всьому світу для управління залежностями та розповсюдження своїх пакетів.

2.1.7. Shadcn

Shadcn — колекція компонентів, які можна з легкістю додати до вебзастосунка. Також має власний CLI, за допомогою якого можливо вибірково додавати компоненти до проєкту, без вимоги встановлювати їх усіх.

2.1.8. Zod

Zod — це бібліотека для валідації даних у JavaScript та TypeScript. Вона надає зручний спосіб визначення схем даних та перевірки, чи відповідають дані цим схемам. Зокрема, Zod дозволяє визначати типи даних, валідатори та обробники помилок. Використання Zod у своїх проєктах допомагає забезпечити правильність та надійність даних, що використовуються в програмі.

2.1.9. React hook form

React Hook Form — це бібліотека для управління формами в React, яка використовує хуки (hooks) для спрощення роботи з формами та уникнення зайвого повторного рендерингу компонентів. Ця бібліотека забезпечує зручний та ефективний спосіб зберігання та валідації даних форм, роблячи розробку форм в React простішою та прозорою. Основним хуком є useForm.

2.1.10. Tailwind

Tailwind CSS — це модульна бібліотека CSS, яка дозволяє швидко створювати стилізовані інтерфейси для вебдодатків. Замість написання власних CSS-стилів, розробник використовує класи, які пропонуються Tailwind, для застосування стилів безпосередньо в HTML-розмітці. Tailwind CSS надає широкий набір готових класів для різних стилів, таких як вирівнювання, розміри, кольори, тіні, границі та інші. Використання Tailwind дозволяє прискорити процес розробки, зменшити кількість CSS-коду.

2.1.11. Jest

Jest — це популярна бібліотека для тестування JavaScript, розроблена для роботи з проєктами на платформі Node.js. Вона забезпечує зручний та потужний інтерфейс для написання, запуску та аналізу тестів у JavaScript-коді. Jest підтримує автоматичне виявлення тестових файлів та запуск усіх тестів у проєкті, а також надає набір корисних утиліт для створення асертів, організації тестових наборів та інших тестових задач.

2.1.12. Zustand

Zustand — це потужна та легка бібліотека для управління станом в React, яка пропонує простий та інтуїтивно зрозумілий API. Вона ідеально підходить для проєктів будь-якого масштабу, забезпечуючи високу продуктивність та гнучкість у роботі з глобальним станом. Завдяки своїй простоті та мінімальному бойлерплейту, Zustand може бути швидко інтегрований та використаний у React-додатках.

2.2. Структурне рішення

Організуємо проєкт за допомогою модульної структури — де кожна директорія відповідатиме за свій функціонал. Але також варто буде врахувати структурні настанови Next.js, а саме правила у “app” директорії. Тому відповідно від потреби будемо видозмінювати структуру, як вказано у правилах роутингу фреймворку.

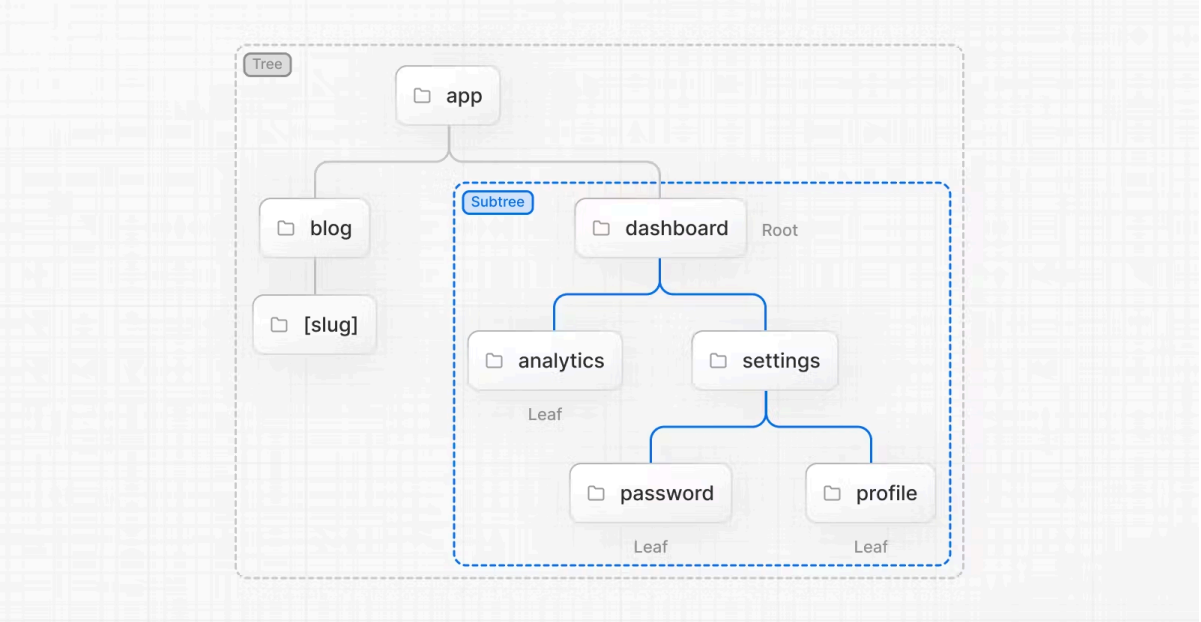

Рис. 2.3. Фундаментальні позначення у “app” директорії

Джерело: next.js routing fundamentals

Tree: конвенція візуалізації ієрархічної структури. Наприклад, дерево компонентів з батьківськими та дочірніми компонентами, структура директорій тощо.

Subtree: частина дерева, що починається з нового кореня (перша) і закінчується листям (остання).

Root: перший вузол у дереві або піддереві, наприклад, кореневий макет.

Leaf: вузли в піддереві, які не мають дочірніх елементів, наприклад, останній сегмент шляху URL-адреси.

2.3. Основні сутності

Оскільки у проєкті використовується TypeScript варто зазначити про основні сутності з якими буде проходити робота та їх кореляція з серверною частиною.

2.3.1. Auth

Для полів авторизації та аутентифікації використаємо стандартну реалізацію, лише для логіну застосуємо функцію Omit(Створює тип, вибираючи всі властивості з Type, а потім видаляючи Keys (рядковий літерал або об'єднання рядкових літералів).

Лістинг 2.3.1. Тип `RegisterFields та LoginFields`.

```
export type RegisterFields = {
 userName: string
 email: string
 password: string
 confirmPassword: string
}
export type LoginFields = Omit<RegisterFields, 'userName' |
'confirmPassword'>
```

2.3.2. Event

Одна з головних сутностей, яку ми отримуємо через запити до сервера. Містить необхідні дані для кожного елементу зі списку подій й має розширену версію для детального ознайомлення на спеціально відведеній сторінці.

Лістинг 2.3.2. Тип `EventCard та ExtendedEvent`.

```
export type EventCard = {
 category: string
 contentType: string
 price: number
 title: string
 thumbnail: string
}
```

```
export type ExtendedEvent = {
 id: number
 title: string
 thumbnail: string
 imageSrc: string
 startDate: string
 endDate: string
 startTime: string
 endTime: string
 price: number
 shortDescription: string
 longDescription: string
 targetedAudience: string
 seatsAvailable: number
 location: string
 additionalResources: string
 isOnline: boolean
 creator: Lector
 eventType: string
 waitingForConfirmation: null | string
 isAllowedToWatchAllContent: null | boolean
 lecturers: Lector[]
}
```

2.3.3. Filter

Також є два типи для фільтрування даних, а саме створення об'єкта filter, який буде відправлено на сервер для надання подій, які відповідають запиту.

Лістинг 2.3.3. Тип `FilterParams` та `FilterRequestParams`.

```
export type FilterParams = {
 category: string | null | undefined,
 format: string | null | undefined,
 lang: string | null | undefined,
}

export type FilterRequestParams = {
 eventTypeId: string | null
 categoryId: string | null
}
```

Висновки до розділу 2

У цьому розділі було описано технологічний стек проєкту, тобто здійснено перелік основних технологій, використані у проєкті, а саме Next.js (на базі React), Typescript, prettier, eslint, npm, shancn, zod, Rhf, tailwind css та Jest. Також розглянуто структурне рішення щодо проєкту, тобто модульна система (як аналог atomic design) та деякі її видозміни відповідно до правил Next.js. Описано основі типи для Typescript, які будуть використовуватись для типізації та роботи з сервером.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Для створення проєкту, як засіб розробки було використане JetBrains WebStorm. JetBrains WebStorm — це інтегроване середовище розробки (IDE), створене спеціально для розробки вебзастосунків. Розроблене JetBrains, відомою компанією-розробником інструментів для програмістів, WebStorm пропонує широкий спектр функцій, які можуть допомогти написати чистий, ефективний та підтримуваний код JavaScript, HTML та CSS.

Ось деякі ключові функції WebStorm, які роблять його улюбленим вибором для багатьох розробників JavaScript:

1) Інтелектуальне автодоповнення та перевірка коду. WebStorm пропонує контекстно-залежне автодоповнення коду, яке допомагає швидше писати код і уникати помилок. Також він включає перевірку коду на льоту, яка виділяє потенційні проблеми у коді, допомагаючи написати чистий та надійний код.
2) Рефакторинг коду**:** WebStorm пропонує широкий спектр функцій рефакторингу коду, які дозволяють легко перейменувати змінні, функції та класи, а також реструктурувати код, не порушуючи його функціональності. Це може допомогти покращити читабельність та підтримуваність коду.
3) Відлагодження**:** WebStorm має потужний інструмент відладки, який дозволяє легко відлагоджувати код JavaScript як у браузері, так і на сервері Node.js. Ви можете встановити точки зупинки, переглянути змінні та стеки викликів, щоб знайти та виправити помилки у коді.
4) Підтримка фреймворків: WebStorm підтримує популярні фреймворки JavaScript, такі як React, Angular та Vue.js. Це означає, що він пропонує спеціальні функції автодоповнення, перевірки коду та відладки для цих фреймворків, що може допомогти працювати з ними більш ефективно.

5) Інтеграція з системами контролю версій: WebStorm тісно інтегрується з популярними системами контролю версій, такими як Git, Subversion та Mercurial. Це дозволяє легко відстежувати зміни у коді, робити коміти й тому подібне.

Переваги використання WebStorm:

1) Підвищена продуктивність**:** функції WebStorm, такі як автодоповнення, перевірка коду та рефакторинг, можуть допомогти написати код швидше та з меншою кількістю помилок.
2) Покращена якість коду**:** WebStorm може допомогти написати чистий, ефективний та підтримуваний код завдяки своїм функціям перевірки коду та рефакторингу.
3) Легше відлагодження**:** потужний інструмент відладки WebStorm може допомогти легко знайти та виправити помилки у коді.

Загалом, JetBrains WebStorm є потужним та багатофункціональним IDE, яке може допомогти стати більш продуктивним та ефективним розробником JavaScript.

3.2. Реалізація вебзастосунка

Застосунок було вирішено створити використовуючи Next.js по низці причин (див. розділ 2, підпункт 2.1.2.).

Для створення базового шаблону варто використати наступну команду у терміналі. Після відповіді на певну кількість питань, буде створено проєкт.

```
npx create-next-app@latest
```

Головним елементом у проєкті є “app” директорія, за допомогою якої відбувається роутинг та у ній розміщуються всі сторінки. Next.js використовує маршрутизатор на основі файлової системи, де директорії використовуються для визначення маршрутів. Кожна директорія являє собою сегмент маршруту, який відображається на сегмент URL-адреси. Щоб створити вкладений

маршрут, ви можете вкладати директорії одна в одну.

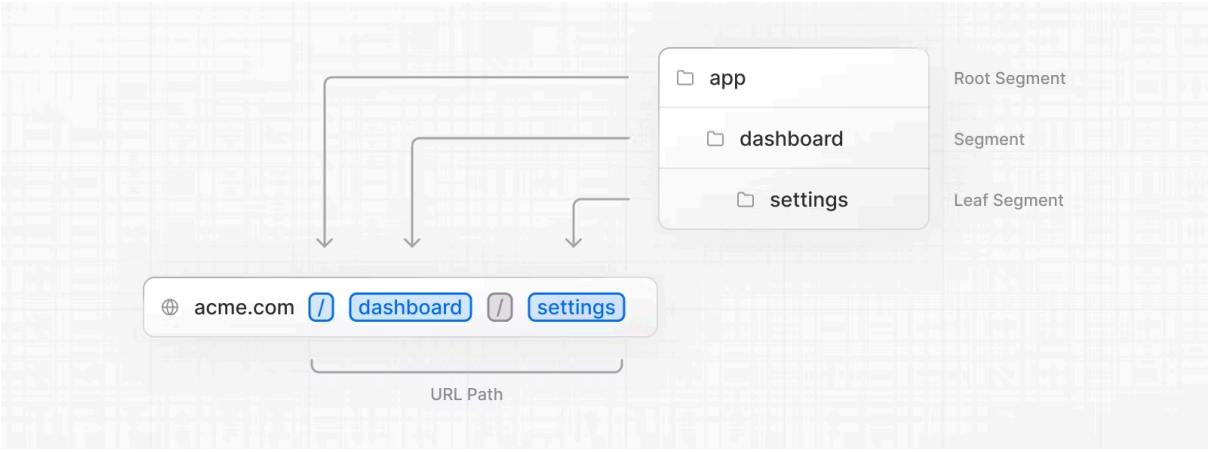

Рис. 3.1. Файловий роутинг

Джерело: документація роутингу Next.js

Визначимо усі необхідні маршрути та створимо для них відповідні директорії. Варто також зазначити, що Next.js дозволяє створювати роутингові групи, у які теж можливо додати файл layout.js.

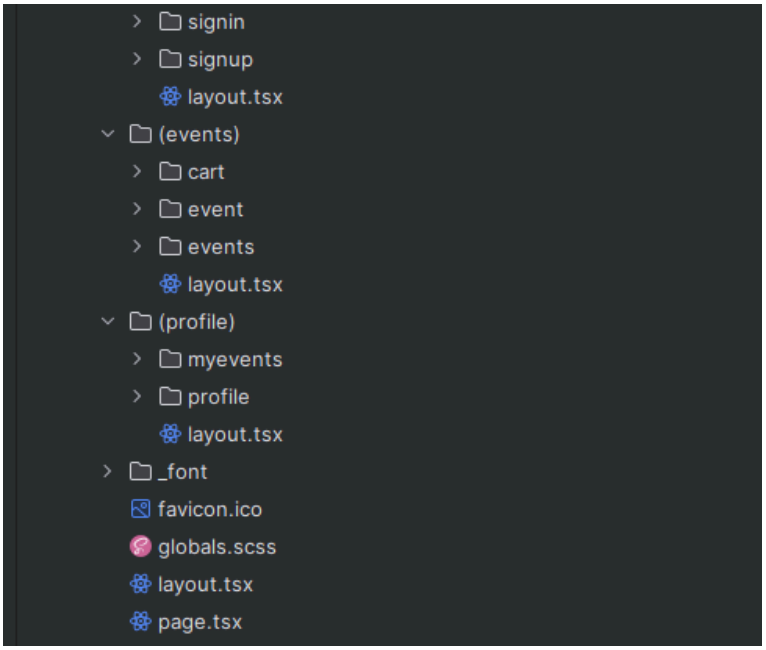

Рис. 3.2. App file tree

Джерело: розроблено автором

Додано shadcn, для доступу до колекції компонентів. Для цього використано CLI, та будемо встановлювати компоненти при потребі.

Лістинг 3.1. Приклад shadcn компонента

```
import { cn } from "@/lib/utils"

function Skeleton({
 className,
 ...props
}: React.HTMLAttributes<HTMLDivElement>) {
 return (
   <div
     className={cn("animate-pulse rounded-md bg-muted", className)}
     {...props}
   />
 )
}

export { Skeleton }
```

Проєкт варто розмістити на GitHub, для збереження кодової бази, та спрощеної роботи. Також це додасть можливість скористатись GitHub Actions(див. розділ 1, підпункт 1.3.).

Рис. 3.3. Вигляд проєкту на GitHub

Джерело: GitHub.com

Почнемо створення з невеликих компонентів, які пізніше об’єднаємо у page.tsx.

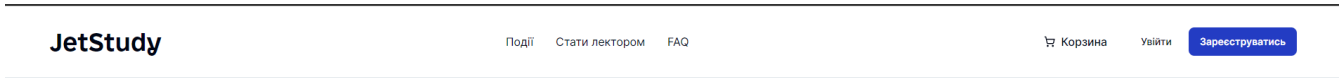

Рис. 3.4. JetStudy Header

Джерело: створено автором

Наступним було створено головну сторінку, де відображаються усі події, відповідно до їх дати:

1. Події цього тижня
2. Події цього місяця
3. Події наступного місяця

Для кожного рядка використовується компонент який отримує необхідні properties.

Лістинг 3.2. Реалізація ContentRow

```
const EventsContentRow = async ({ url, title }: { url: string; title:
string }) => {
 const monthEvents = await fetchData<Event[]>({ url })
 return (
   <div className='mt-[96px]'>
     <div className='flex w-full justify-between'>
       <h2 className={cn('mb-4 text-[30px] font-medium leading-9
tracking-[-0.225px]', eUK.className)}>
         {title}
       </h2>
       <div className='flex cursor-pointer items-center gap-1'>
         <span className='text-sm font-medium leading-[14px]'>Дивитись
усі</span>
         <ChevronRight size={20} />
       </div>
     </div>
     <div className='mt-4 flex gap-6'>
       {monthEvents.map((event, index) => (
         <DefaultEventCard
           title={event.title}
           category={event.eventType}
           price={1200}
           contentType={event.eventType}
```

```
            thumbnail={event.thumbnail}
            key={event.id}
            imgId={index + 1}
            linkId={event.id}
          />
        ))}
      </div>
    </div>
  )
}
```

Як результат отримано наступний вигляд у браузері.

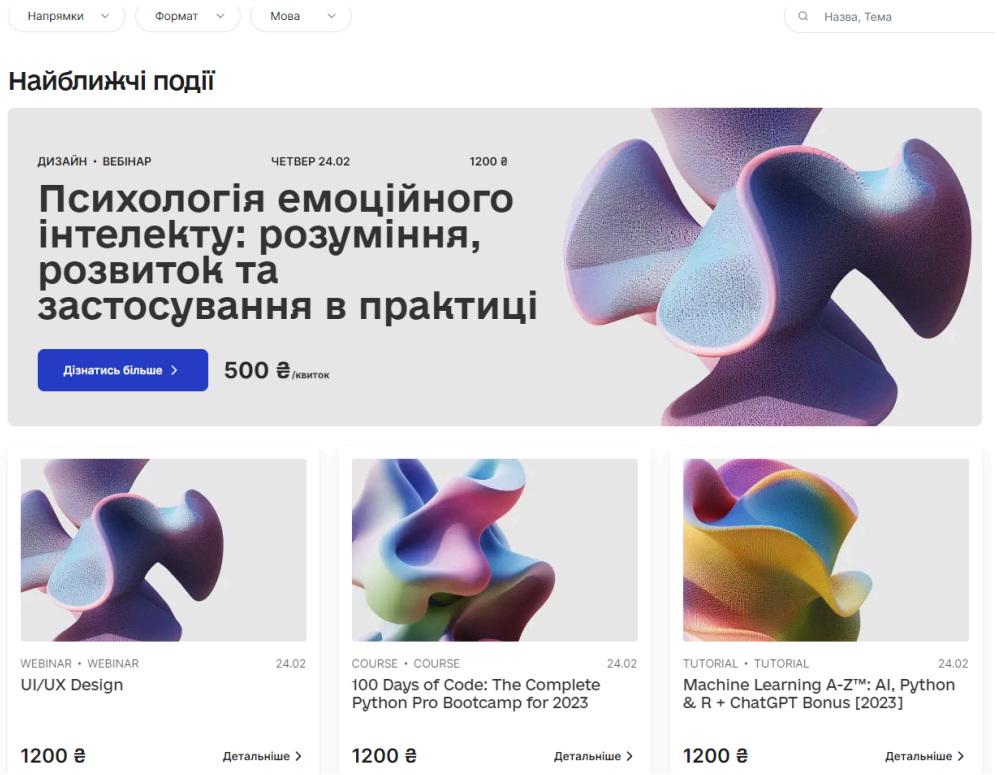

Рис. 3.5. Головна сторінка

Джерело: створено автором

Використавши Select елементи, можемо використовувати фільтрацію — обравши одну з опцій у певні категорії. Наприклад оберемо події з тегом Design.

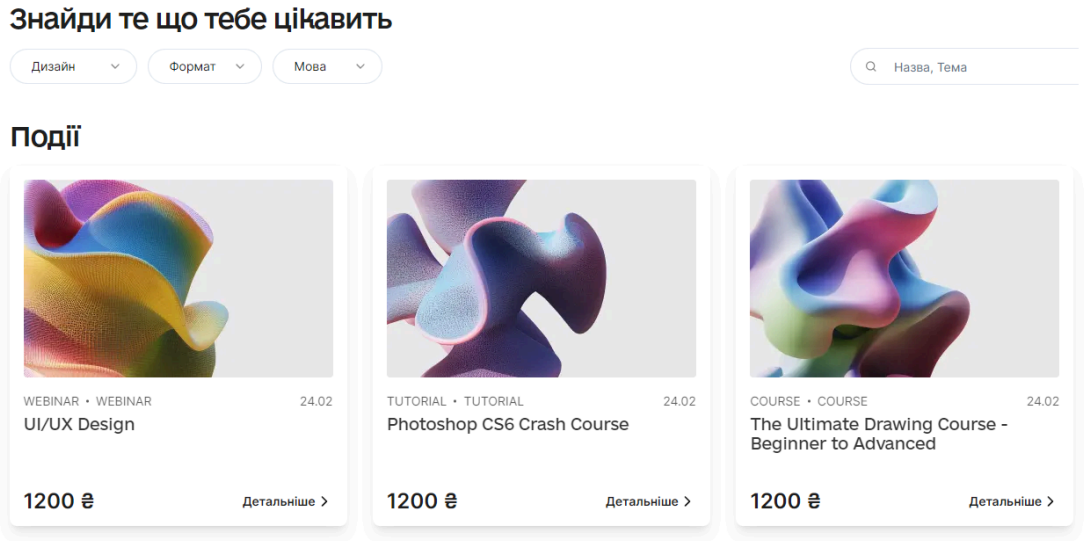

Рис. 3.6. Відфільтровані події

Джерело: створено автором

Фільтрацію реалізовано використовуючи URL як state. Використання URL як стану для фільтрації має декілька переваг порівняно з використанням хука useState. Воно забезпечує постійність стану навіть після перезавантаження сторінки, дозволяє зберігати обрані фільтри в URL, легко обмінюватися посиланнями з іншими користувачами та використовувати стандартну браузерну навігацію. Крім того, цей підхід може покращити SEO, оскільки пошукові системи індексують URL та його параметри.

Лістинг 3.3. Утиліта для створення пошукової стрічки

```
export const generateQueryString = (params:
Partial<FilterRequestParams>): string => {
 const searchParams = new URLSearchParams(
   Object.entries(params)
     .filter(([_, value]) => value !== null && value !== undefined)
     .map(([key, value]) => [key, value] as [string, string]),
 )

 return searchParams.toString() ? `?${searchParams.toString()}` : ''
}
```

Більшість запитів на отримання даних у Next.js працює з використанням серверних компонентів. Next.js розширює fetch Web API, щоб дозволити налаштувати кешування та повторну перевірку для кожного запиту fetch на сервері. React розширює fetch, щоб автоматично запам'ятовувати запити на вибірку під час рендерингу React-компонентів. Тому було вирішено створити невелику функцію утиліту для створення запитів використовуючи нативний fetch API, також додано generic return type, для спрощеної роботи з типізацією отриманих даних.

Лістинг 3.4. Утиліта для запитів

```
export const fetchData = async <T>({ url, cache = 'force-cache' }:
fetchDataOptions): Promise<T> => {
 try {
   const response = await fetch(`${server_url}${url}`, {
     cache,
   })
   return await response.json()
 } catch (e: any) {
   throw new Error(`Error fetching data: ${e.message}`)
 }
}
```

Як результат, тепер при використанні properties, під час методів перебору(та інших) як map, IDE, розуміє який тип ми маємо й буде допомагати в autocomplete.

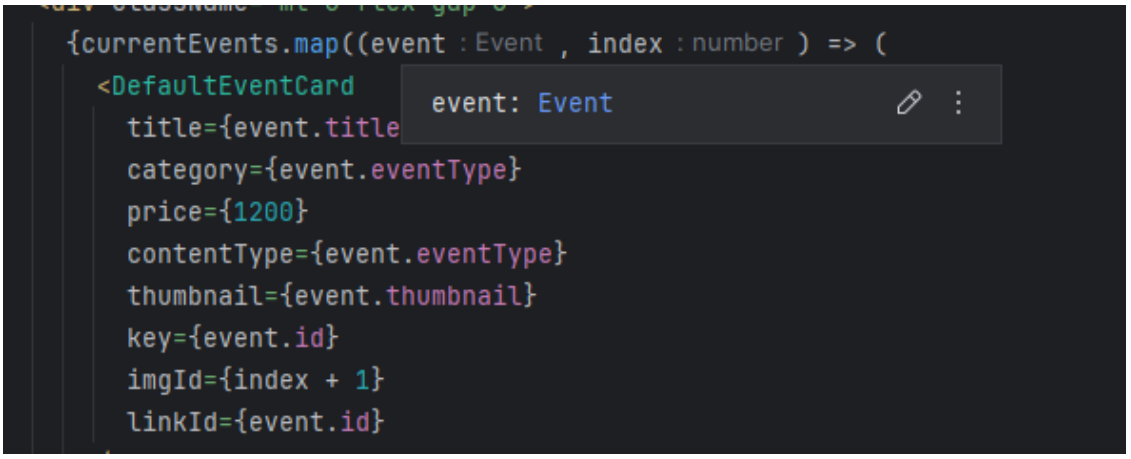

Рис. 3.7. Підказка типів від IDE

Джерело: створено автором

Також, для зручності відображення відфільтрованих подій, чи за спеціально обраний часовий проміжок, було додано пагінацію. Це система поділу великого набору даних (тексту, списку, зображень тощо) на окремі, менші за обсягом, сторінки, а також система нумерації та навігації по цих сторінках. Для реалізації було використано відповідний компонент shadcn та Link для навігації між сторінками.

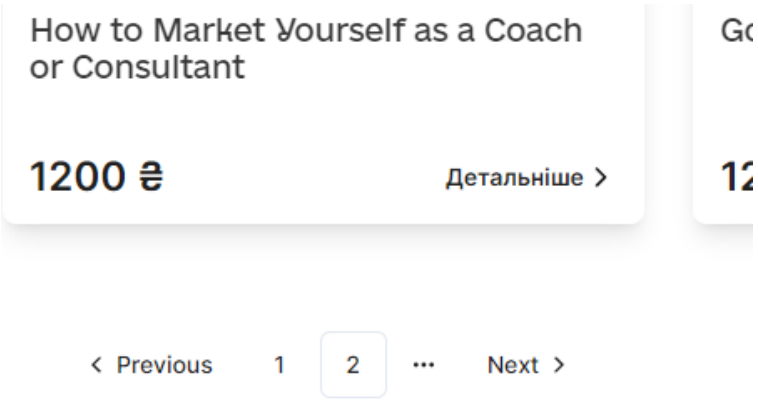

Рис. 3.8. Пагінація подій

Джерело: створено автором

У кожної події є своя детальна сторінка для кращого ознайомлення із нею. Для створення маршруту з динамічною адресою у Next.js є своя реалізація.

Динамічний сегмент можна створити, взявши назву теки у квадратні дужки: [folderName]. Наприклад, [id] або [slug]. Динамічні сегменти передаються як параметри до layout, page, route та generateMetadata.

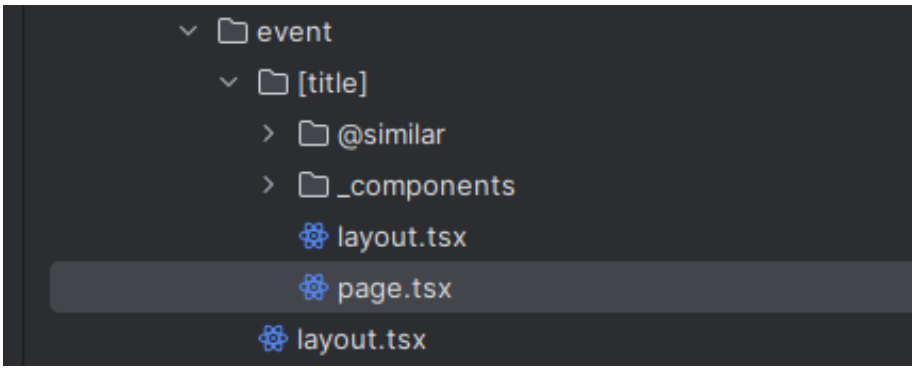

Рис. 3.9. Динамічний сегмент

Джерело: створено автором

Лістинг 3.5. Використання params для отримання даних

```
export default async function DynamicEvent({ params }: { params: {
title: string } }) {
 const detailedEvent = await fetchData<ExtendedEvent>({ url:
`/event/${params.title}` })
```

Створена детальна сторінка має наступний вигляд(див. рис. 3.9.).

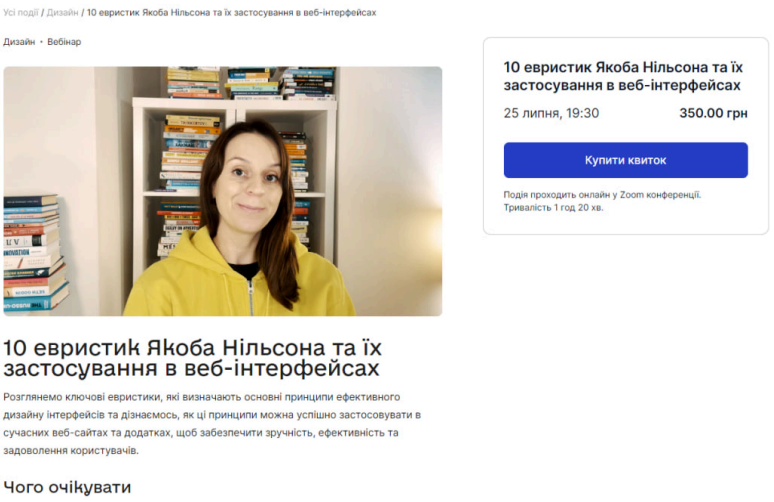

Рис. 3.10. Детальна сторінка

Джерело: створено автором

Нижче, сторінка містить інформацію про лекторів, та контакти

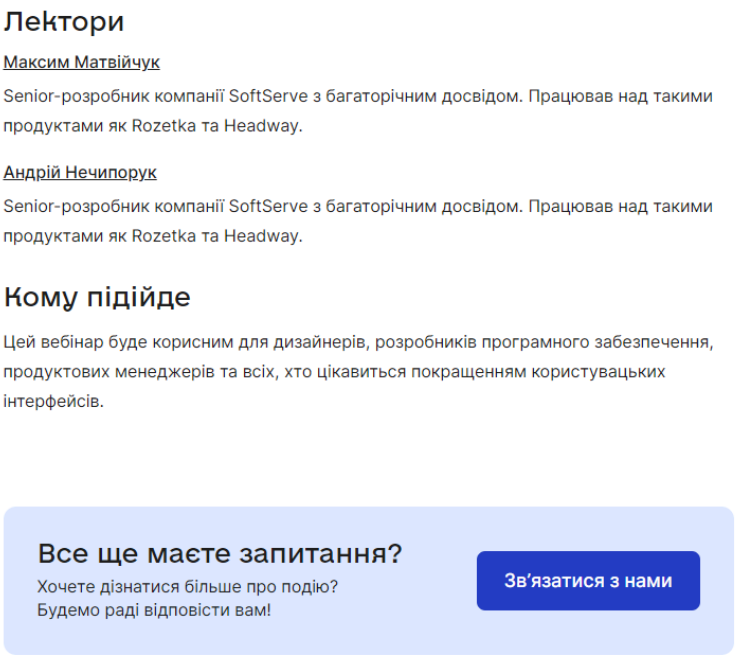

Рис. 3.11. Продовження детальної сторінки

Джерело: створено автором

Детальна сторінка також має секцію “інших подій” у цій самій категорії. Для реалізації окремого завантаження даних було використано Parallel Routes.

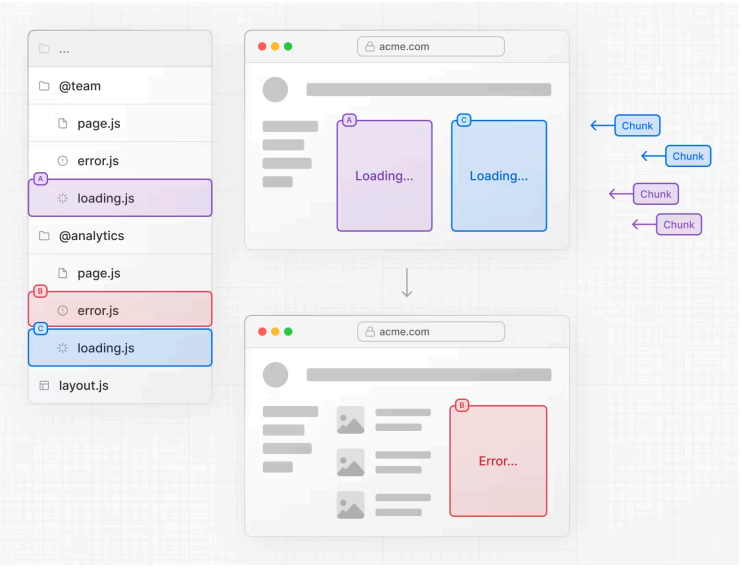

Рис. 3.12. Принцип роботи Parallel Routes

Джерело: Next.js parallel routes documentation

Маршрути завантажуються незалежно одне від одного тому, можемо легко задати сторінку помилки або fallback під час завантаження даних. Як результат маємо два роута на одній сторінці, які працюють незалежно.

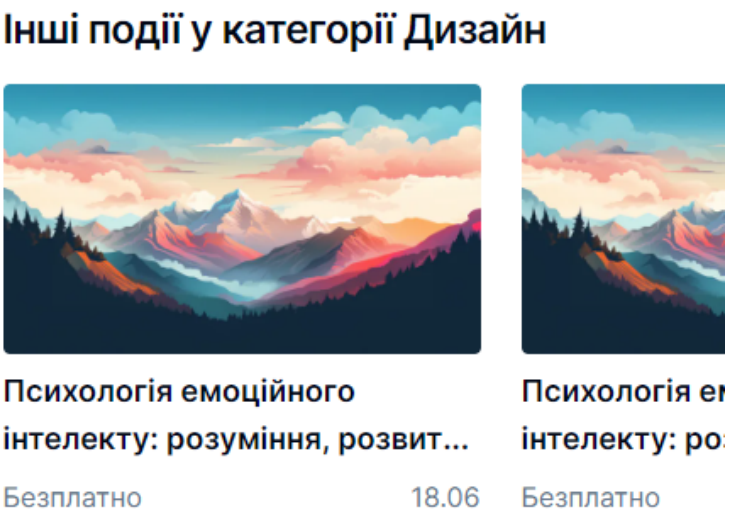

Рис. 3.13. Інші події на сторінці

Джерело: створено автором

Варто зазначити що parallel routes працює через використання Slots. Slots визначаються за допомогою конвенції @folder. Наприклад, наступна структура файлів визначає слот @similar.

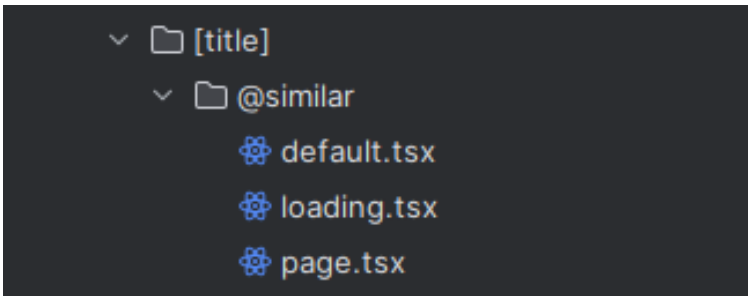

Рис. 3.14. Slot для [title]

Джерело: створено автором

Для того, щоб використати Slot, достатньо отримати його як property у layout.tsx

Лістинг 3.6. Використання Slot як ReactNode у jsx.

```
type LayoutProps = {
 children?: React.ReactNode
 similar?: React.ReactNode
}

export default function Layout({ children, similar }: LayoutProps) {
 return (
   <>
     {children}
     {similar}
   </>
 )
}
```

Наступним, було розроблено сторінки для реєстрації та входу у застосунок. Для роботи з формами було використано react-hook-form та zod як валідація. Приклад використання хука useForm(див. лістинг 3.6.)

Лістинг 3.7. useForm із відповідним типом.

```
const form = useForm<UserAuthType>({
 resolver: zodResolver(formSchema),
 defaultValues: {
   password: '',
   email: '',
 },
})
```

Також можна зазначити що форма використовує resolver, через який працює валідація полів. Для цього створюється спеціальна schema, яка передається як параметр у resolver, після чого правила перевірки успішно працюють у формі.

Лістинг 3.8. Schema для auth форм.

```
import { z } from 'zod'
```

```
const passwordRegex = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*#?&]{8,}$/

export const formSchema = z.object({
 email: z.string().email('Incorrect email'),
 password: z.string().refine((value) => passwordRegex.test(value), {
   message: 'Password must be at least 8 characters long and include
both letters and numbers.',
 }),
})
```

Аби не створювати окремий тип для форми, оскільки useForm вимагає передати як generic тип, що буде відповідати необхідним полям. Скористаймося утилітою zod - infer. Вона дозволяє автоматично створити тип з уже наявної схеми.

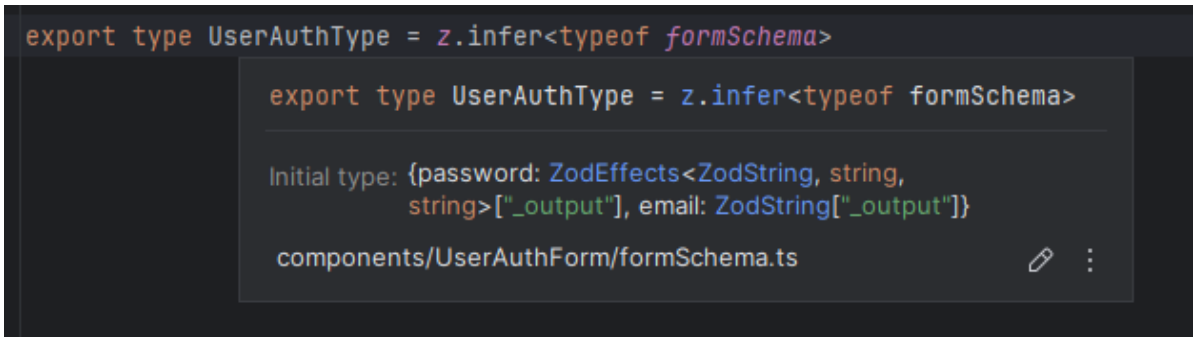

Рис. 3.15. Використання infer

Джерело: створено автором

Як результат у JSX, будемо мати наступний вигляд коду, при застосуванні бібліотеки(див. лістинг 3.7.)

Лістинг 3.9. Використання handleSubmit.

```
<form onSubmit={form.handleSubmit(onSubmit)} className='my-2 flex
flex-col gap-2'>
 <FormField
   control={form.control}
   name='email'
```

```
  render={({ field }) => (
    <FormItem>
      <FormLabel>Електронна пошта</FormLabel>
      <FormControl>
        <Input placeholder='email@gmail.com' {...field} />
      </FormControl>
      <FormMessage />
    </FormItem>
  )}
/>
```

Для реєстрації та входу було також додано взаємодію через Google, тобто використання Google акаунтів. Щоб це реалізувати було використано бібліотеку @react-oauth/google. Вона надає можливість використати хук useGoogleLogin, який допоможе отримати відповідний токен. Отже, сторінка реєстрації має наступний вигляд(див. рис. 3.15.)

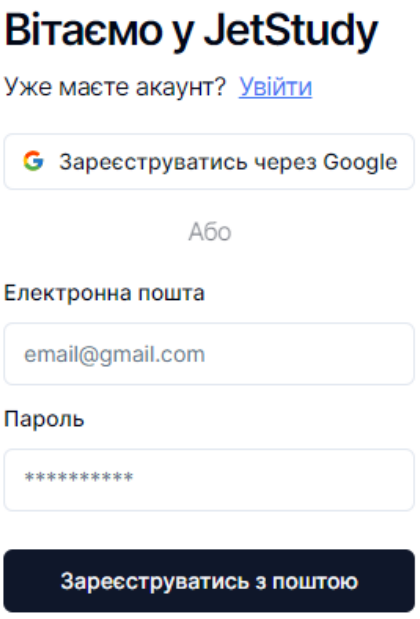

Рис. 3.16. Сторінка реєстрація

Джерело: створено автором

Для роботи з мутаціями даних, вирішено використати бібліотеку swr. SWR - це бібліотека для управління станом даних та кешування вебдодатків, яка

спрощує отримання та оновлення даних з сервера. Вона забезпечує простий та зручний інтерфейс для використання у React-додатках, включаючи автоматичне кешування, керування статусом завантаження та помилок, а також автоматичне оновлення даних при їх зміні на сервері. SWR дозволяє покращити продуктивність та відповідність додатків, зменшити кількість запитів до сервера та зробити код простішим і зрозумілим.

Скористаймося хуком useSWRMutation, у який передамо адресу запиту та функцію fetcher, яка є простою обгорткою над нативним API.

Лістинг 3.10. хук useSWRMutation.

```
const { trigger, isMutating } = useSWRMutation('/auth/register',
registerUser)
```

Надалі, можемо використовувати функцію trigger для відпрацювання мутації, та статус isMutating, для відображення наявності завантаження дії.

Тепер можемо успішно зареєструватися та увійти використовуючи пошту та пароль. Аби мати можливість зробити це через Google, зробимо останній крок для цього та загорнемо auth layout.tsx в GoogleOAuthProvider та надамо google client id який отримали після створення проєкту в google cloud.

Варто зазначити що хуки пов’язані з інтеракцією завжди мають бути використані у клієнтських компонентах, тобто з додаванням директиви ‘use client’. Це зроблено для забезпечення від отримання помилки та дотримання необхідних вимог щодо структури

Рис. 3.17. Приклад помилки

Джерело: створено автором

Після виконання всіх дій можемо спостерігати роботу oauth.

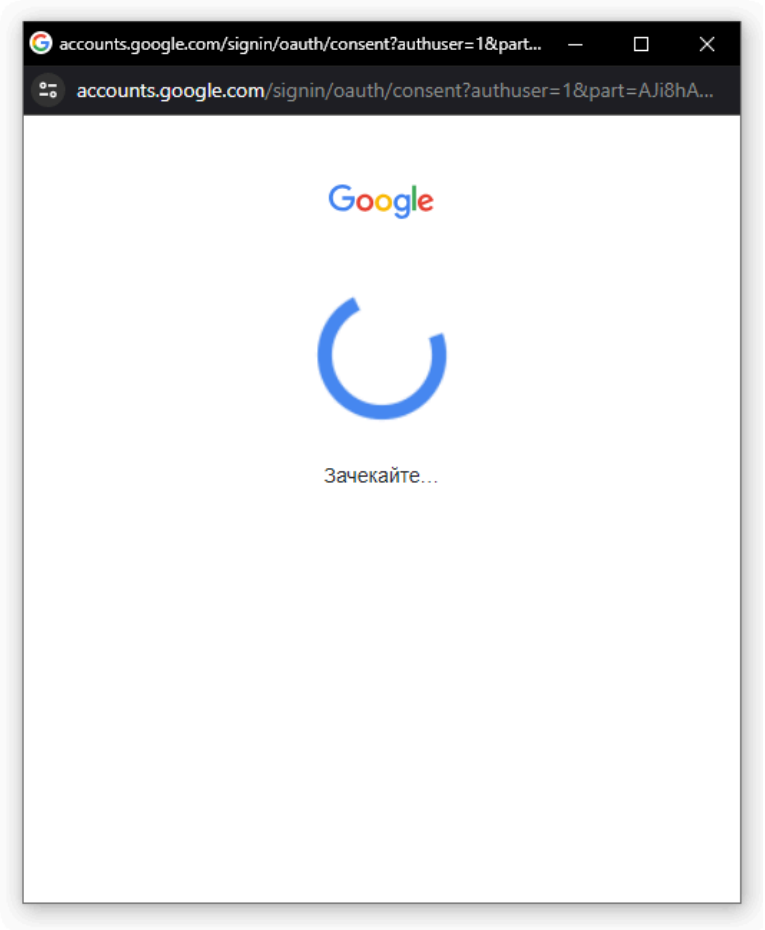

Рис. 3.18. OAuth

Джерело: створено автором

Наступним етапом, було створення сторінок профілю та замовлень. Для них створимо окрему групу роутів, а layout.tsx матиме наступний код(див. лістинг 3.9.)

Лістинг 3.11. layout.tsx для profile.

```
export default async function ProfileLayout({ children }:
ProfileLayoutProps) {
 return (
   <div className='flex min-h-screen flex-col'>
     <Header />
     <main className='flex-1 mt-8'>
```

```
        <div className='container flex gap-6'>
          <ProfileNavigation />
          {children}
        </div>
      </main>
      <Footer />
    </div>
  )
}
```

Після проходження у профіль авторизувавшись, наші дані будуть відображатись у відповідних полях.

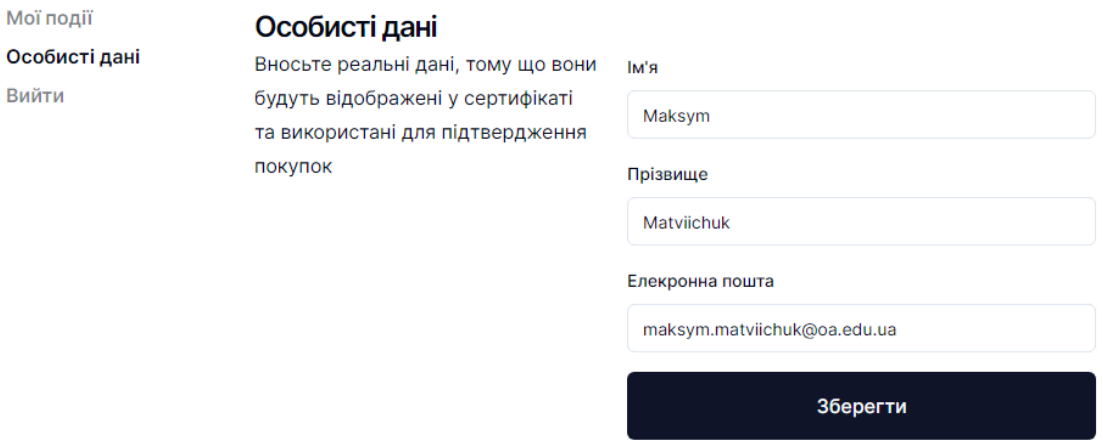

Рис. 3.19. Сторінка профілю
Джерело: створено автором

Також сторінки подій на які користувач зареєстрований.

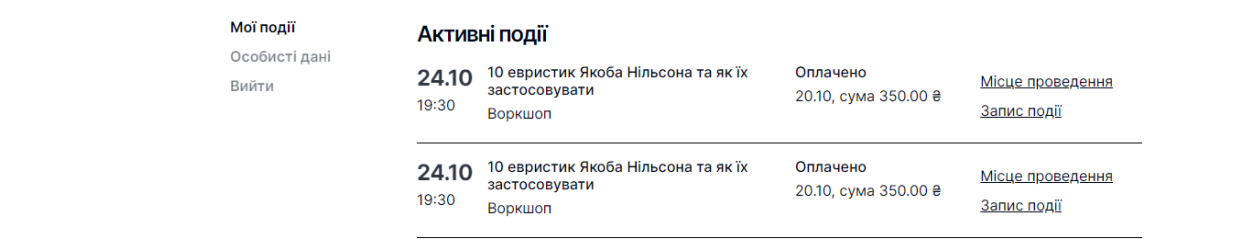

Рис. 3.20. Сторінка подій користувача

Джерело: створено автором

До цього ж створимо сторінку для оформлення замовлення, після додавання товарів у кошик можемо оформити замовлення на спеціально відведеному маршруті.

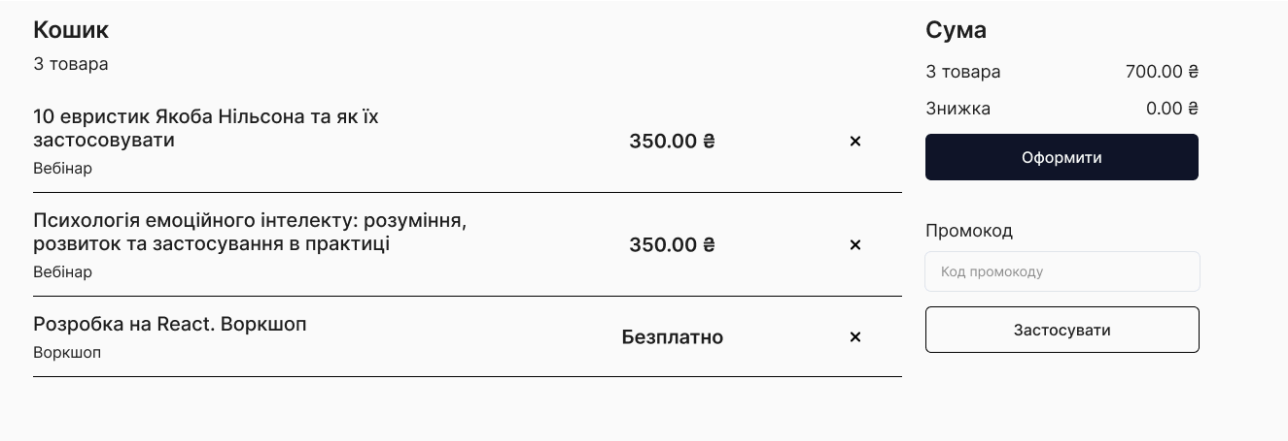

Рис. 3.21. Сторінка підтвердження замовлень

Джерело: створено автором

Самий же кошик було створено використовуючи компонент Sheet, один з його складових має наступну реалізацію (див. лістинг 3.10.).

Лістинг 3.12. SheetContent.

```
const SheetContent = React.forwardRef<
 React.ElementRef<typeof SheetPrimitive.Content>,
 SheetContentProps
>(({ side = "right", className, children, ...props }, ref) => (
 <SheetPortal>
   <SheetOverlay />
   <SheetPrimitive.Content
     ref={ref}
     className={cn(sheetVariants({ side }), className)}
     {...props}>
     {children}
     <SheetPrimitive.Close className="absolute right-4 top-4 rounded-sm
opacity-70 ring-offset-background transition-opacity hover:opacity-100
focus:outline-none focus:ring-2 focus:ring-ring focus:ring-offset-2
disabled:pointer-events-none data-[state=open]:bg-secondary">
       <X className="h-4 w-4" />
       <span className="sr-only">Close</span>
```

```
    </SheetPrimitive.Close>
  </SheetPrimitive.Content>
 </SheetPortal>
))
```

Як результат можемо спостерігати наступний елемент при натисканні на відповідну кнопку (див. рис. 3.21.).

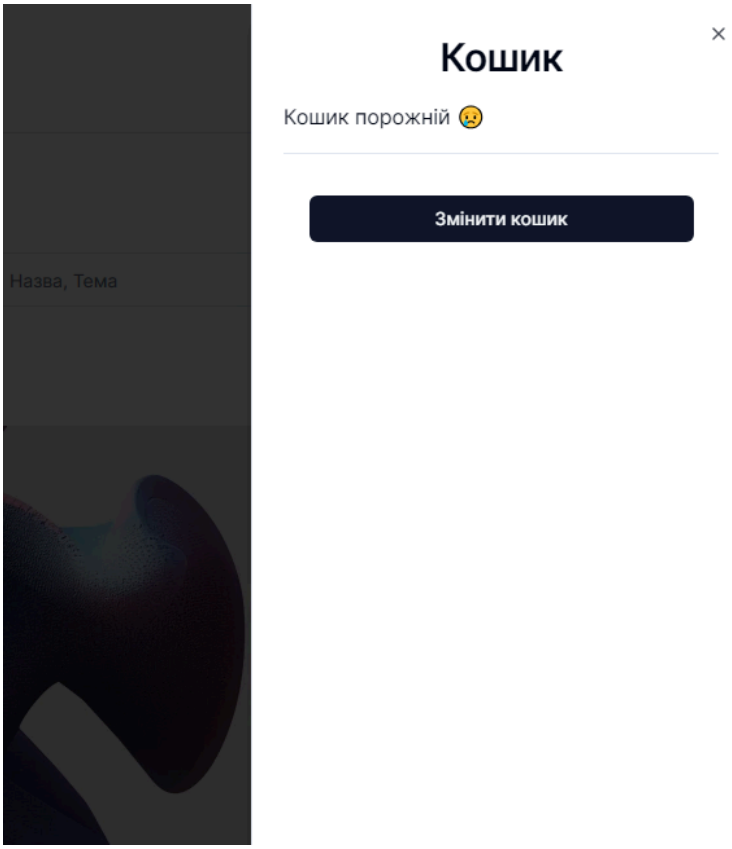

Рис. 3.22. Вигляд кошика

Джерело: створено автором

Наступним було реалізовано можливість додавання подій до кошика. Для цього було використано стейт-менеджер Zustand, який було описано у розділі 2.1. Аби скористатися можливостями цієї бібліотеки потрібно створити файл store.ts, у якому необхідно описати усю логіку управління певними елементами. Імплементація виглядатиме наступним чином:

Лістинг 3.13. Імплементація Zustand Store.

```
export const useBasketStore = create<BasketState>()((set) => ({
 items: [],
 addItemToBasket: (newItem) =>
   set((state) => {
     return {
       items: [...state.items, newItem],
     }
   }),
 setItems: (items) =>
   set(() => {
     return {
       items: [...items],
     }
   }),
 removeItemFromBasket: (eventId) =>
   set((state) => {
     return {
       items: state.items.filter((event) => eventId !== event.eventId),
     }
   }),
}))
```

Після цих дій, стало можливим додавати обрані події у свій кошик. Для цього потрібно зайти на детальну сторінку події, після чого натиснути на відповідну кнопку. Також для кращого розуміння успішності відпрацювання функції додавання було додано сповіщення про дію, або так званий Toast. Його було імпортовано використовуючи shadcn cli.

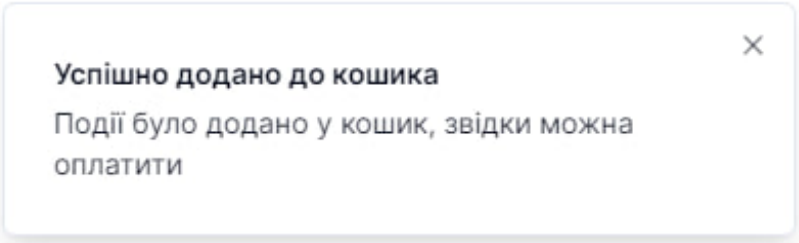

Рис. 3.23. Toast про успішну дію

Джерело: створено автором

Тепер після натискання на відповідну кнопку, обрана подія з’явиться у кошику(див. рис 3.23.).

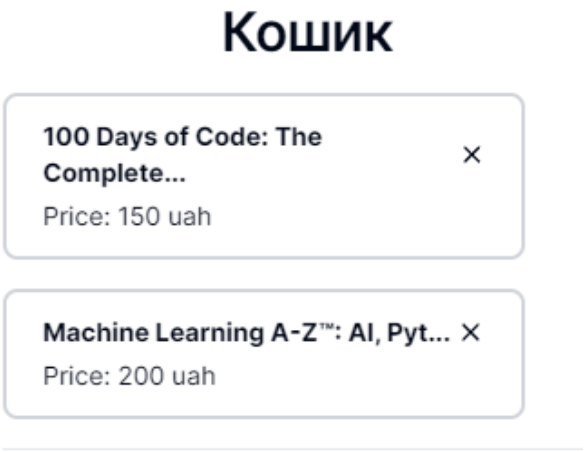

Рис. 3.24. Події додані до кошика

Джерело: створено автором

Після завершення процесу обрання подій, їх також можна видалити напряму через кошик, користувач матиме змогу перейти на сторінку оплати, яку було згадано раніше(див. рис. 3.20.).

Наступним кроком для застосунку було вирішено додати Docker. Docker забезпечить ізольоване середовище для виконання програм, що дозволяє запускати їх на будь-якому комп'ютері або сервері без необхідності налаштування середовища вручну. Це полегшує розгортання та масштабування додатків, поліпшує їх портативність та забезпечує консистентність робочого середовища між різними розробниками та у різних етапах розробки.

Для початку, потрібно локально встановити Docker. Після чого потрібно додати Dockerfile у корінь проєкту. Він містить інструкцію для автоматичної збірки Docker-образу та вказівки з встановлення залежностей, копіювання файлів, налаштування середовища та запуску команд.

Лістинг 3.14. Dockerfile.

```
FROM node:18-alpine
```

```
WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000
```

Наступні елементи, які потрібно додати це dockerignore та docker-compose.yml. Перший використовується для вказівки Docker, які файли та теки слід ігнорувати під час збірки образу, а docker-compose це файл конфігурації, який використовується для визначення та запуску декількох Docker-контейнерів також він описує сервіси, які складають програмне забезпечення, а також їх взаємозв'язки. Цей файл матиме наступні інструкції:

Лістинг 3.15. Інструкція docker-compose.yml.

```
version: '3'
services:
 frontend:
   build: .
   ports:
     - '3000:3000'
   volumes:
     - .:/app
     - /app/node_modules
   environment:
     - NEXT_PUBLIC_DEV_SERVER_URL_DK=${NEXT_PUBLIC_DEV_SERVER_URL_DK}
   command: npm run dev
```

Після того як, усі необхідні вимоги були виконані, необхідно переконатися у тому, чи запущений Docker, після чого потрібно запустити команду docker-compose up у терміналі проєкту. Після чого починається збірка та запуск

контейнера. У разі успішного виконання команди, у програмі Docker з’явиться відповідний проєкт.

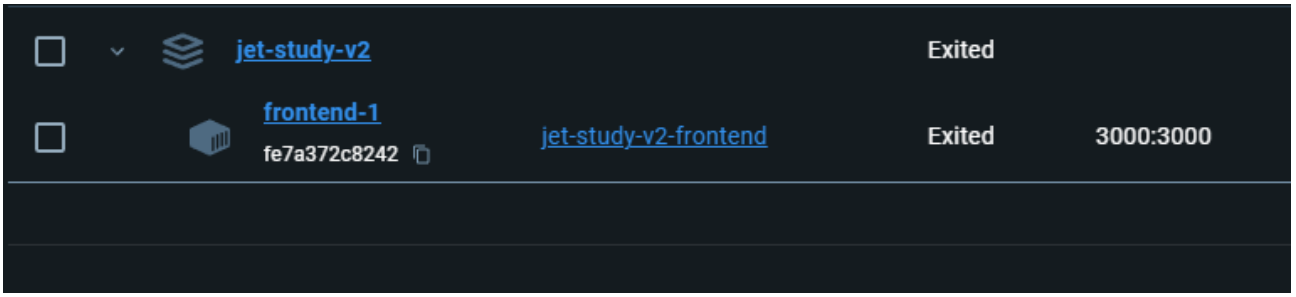

Рис. 3.25. Мультиконтейнер jet-study

Джерело: створено автором

З піднятим контейнером є можливість так само продовжувати розробку над застосунком, але з усіма раніше переліченими перевагами контейнеризації.

Хорошою практикою до більшості написаного функціоналу застосунку буде додавання unit-тестів.

Unit-тестування - це процес написання автоматизованих тестів для невеликих, ізольованих одиниць коду, таких як функції, методи й класи. Ці тести призначені для перевірки того, чи працює код очікуваним чином, і їх можна запускати повторно під час розробки та після неї, щоб гарантувати, що код не регресує.

Переваги використання юніт-тестування:

1. Юніт-тести можуть допомогти виявити помилки на ранніх етапах процесу розробки, коли їх дешевше і простіше виправити.
2. Процес написання юніт-тестів може допомогти краще зрозуміти дизайн коду та покращити його загальну структуру.
3. Юніт-тести дають впевненість, що можливо вносити зміни до свого коду, не порушуючи його функціональність.

Як інструмент тестування було використано Jest, як це зазначається у розділі 1.

Для додавання його у проєкт потрібно встановити необхідні залежності та створити файл конфігурації.

Лістинг 3.16. Конфігурація Jest.

```
import type { Config } from 'jest'
import nextJest from 'next/jest.js'

const createJestConfig = nextJest({
 dir: './',
})

// Add any custom config to be passed to Jest
const config: Config = {
 preset: 'ts-jest',
 coverageProvider: 'v8',
 testEnvironment: 'jsdom',
 moduleNameMapper: {
   '^@/(.*)$': '<rootDir>/$1',
 },
 setupFilesAfterEnv: ['<rootDir>/jest.setup.ts'],
}

// createJestConfig is exported this way to ensure that next/jest can
load the Next.js config which is async
export default createJestConfig(config)
```

Для подальших дій створено директорію __tests__, у якій будуть знаходиться усі файли тестів. Варто зазначити кожен файл тестів має містити у собі .test тобто правильною назвою буде, наприклад: Header.test.tsx. Напишемо кілька простих тестів для компонентів та функцій.

Лістинг 3.17. Конфігурація Jest.

```
describe('generateQueryString', () => {
 it('should generate a query string from an object', () => {
   const params: Partial<FilterRequestParams> = {
     eventTypeId: 'type1',
     categoryId: 'category1',
   }
```

```
    const result = generateQueryString(params)
    expect(result).toEqual('?eventTypeId=type1&categoryId=category1')
  })
  it('should return an empty string when all values are null or
undefined', () => {
    const params: Partial<FilterRequestParams> = {
      eventTypeId: null,
      categoryId: undefined,
    }
    const result = generateQueryString(params)
    expect(result).toEqual('')
  })
})
```

За допомогою npm можемо запустити усі тести та переглянути успішність їх виконання. Для цього у файлі package.json у нас автоматично мають з'являтися відповідні скрипти для запуску.

Як правило це буде npm run test або "test:watch":"jest --watch". Після запуску відповідної команди можемо спостерігати на відпрацювання тестів та чи пройшли елементи сам тест.

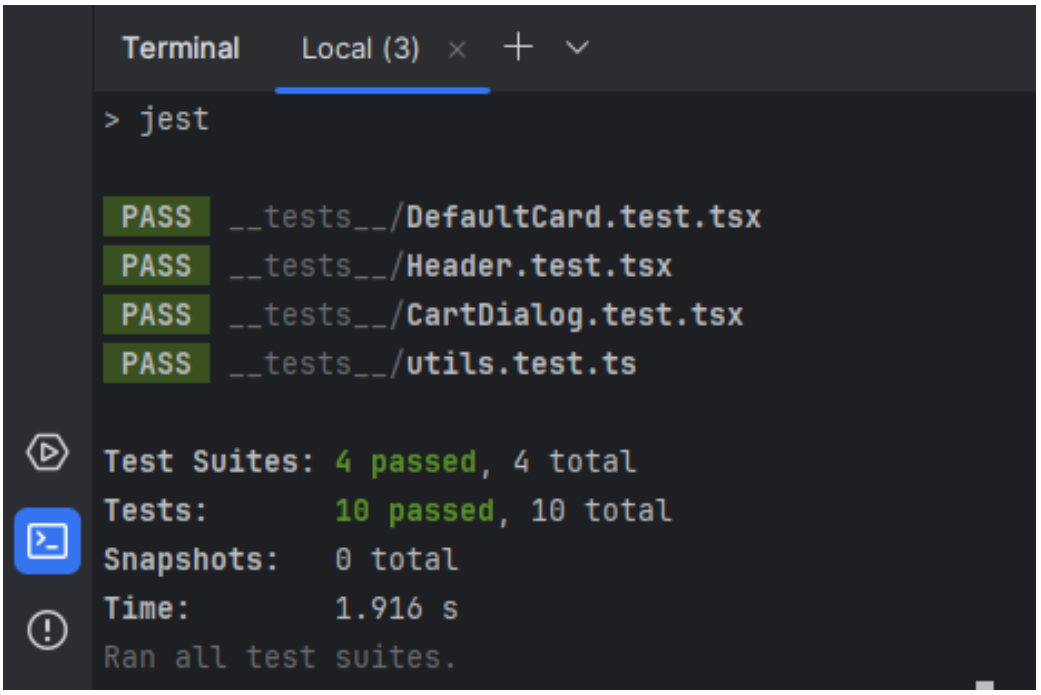

Рис. 3.26. Відпрацювання тестів

Джерело: створено автором

Як було раніше згадано, проєкт розміщено на GitHub, що дає змогу скористатися GitHub Actions, які допомагають в автоматизації процесів та прискорені роботи. Створимо action для запуску всіх тестів під час pull request. Для цього додамо відповідну директорію та файл .yml у якому буде розписана інструкція по діях.

Лістинг 3.18. Tests Action.

```
name: Tests

on:
 pull_request:
   types: [opened, synchronize]

jobs:
 test:
   runs-on: ubuntu-latest

   steps:
     - name: Checkout code
       uses: actions/checkout@v2

     - name: Setup Node.js
       uses: actions/setup-node@v2
       with:
         node-version: '18'

     - name: Install dependencies
       run: npm ci

     - name: Run tests
       run: npm test
```

Тепер при створенні pull requests у репозиторій з проєктом буде запускатись відповідний action виконувати описану дію.

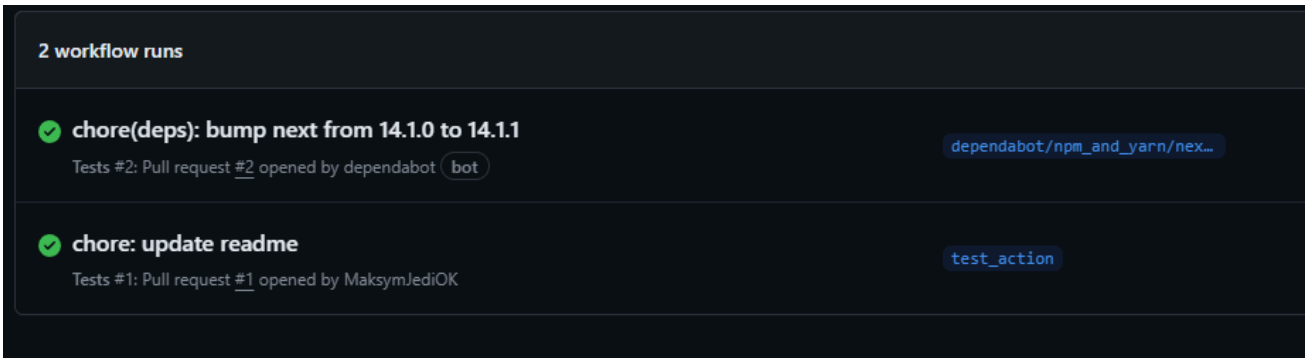

Рис. 3.27. Успішне виконання GitHub Action

Джерело: створено автором

Також є можливість додати інші види actions, які в майбутньому ще краще оптимізують роботу над проєктом.

3.3. Перспективи та альтернативи

Як будь-який додаток JetStudy планує подальше розширення функціонала та додавання нових можливостей до проєкту.

Зі зростанням розміру проєкту буде доцільним збільшити покриття тестами та додати інтеграційні й End-to-End тести. E2E тестування — це вид тестування програмного забезпечення, що перевіряє роботу системи як єдиного цілого, від користувацького інтерфейсу до функціональності. Відмінною рисою є його зосередженість на відтворенні реальних умов використання, дозволяючи виявити проблеми, які можуть виникнути в реальному середовищі. Такий підхід до тестування дозволяє виявити інтеграційні проблеми, підтримувати регресійне тестування та підвищує довіру до продукту завдяки перевірці відповідності його роботи очікуванням користувачів.

Щодо альтернатив, свою увагу заслуговує tRPC. Оскільки сервер JetStudy написаний на .NET доводиться створювати майже ідентичні типи та деякі інші аспекти.А tRPC ж — це модерна бібліотека для створення API у програмуванні на Node.js та TypeScript. Вона пропонує зручний та потужний спосіб створення

серверної логіки та API, використовуючи декларативний підхід до визначення маршрутів та методів. Основні переваги використання tRPC включають:

1. Типізація: tRPC побудовано на TypeScript, що дозволяє забезпечити типізацію як для вхідних, так і вихідних даних API, що сприяє покращенню стабільності та надійності додатків.
2. Декларативний підхід: Визначення маршрутів та методів API здійснюється за допомогою декларативного підходу, що полегшує розробку та підтримку серверної логіки.
3. Інтеграція з різними клієнтами: tRPC підтримує різні способи зв'язку з сервером, включаючи HTTP, WebSocket та бібліотеки для клієнтів JavaScript, TypeScript та React.
4. Підтримка middleware: tRPC надає механізм middleware, що дозволяє реалізувати різноманітні функціональні можливості, такі як автентифікація, авторизація, журналювання тощо.

Загалом, використання tRPC дозволяє створювати сучасні та надійні API з високим рівнем типізації та декларативною структурою, що полегшує розробку та підтримку серверної логіки у програмуванні на Node.js та TypeScript.
Також варто додати що tRPC дуже легко інтегрується з Next.js через особливості його концептів та роботи.

Висновки до розділу 3

У цьому розділі було розглянуто засіб для розробки проєкту, а саме IDE JEtBrains WebStorm, яке має велику кількість встановленого функціоналу та багато інших речей які набагато спрощують розробку та дозволяють сфокусуватись на написанні коду та знаходження рішень.

Описано покрокову реалізацію вебзастосунка з деталізацією інструментів та зображенням результатів та лістингів програми. Також було розглянуто перспективи розвитку застосунку у певних аспектах та додані певних елементів

для покращення якості проєкту та надання більш сприятливого досвіду розробки у випадку колаборації з іншими розробниками.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було визначене технічне завдання та кроки до його виконання. Було проаналізовано основних гравців на ринку EdTech, що займаються дистрибуцією курсів та подій. Організовано роботу за допомогою використання допоміжних сервісів як GitHub, що допомагає швидко та просто керувати кодовою базою проєкту.

Досліджено та обрано оптимальний технічний стек для створення вебзастосунка та підібрано необхідну структуру. Основою проєкту є:

1. TypeScript
2. React.js
3. Next.js
4. Tailwind
5. Jest

Описано основні сутності навколо яких відбувалась основна взаємодія з backend API, який був створений використовуючи .NET Web API.

У останньому розділі описали програмне забезпечення, а саме засіб розробки, що використовувався для написання проєкту, тобто JetBrains WebStorm, який має безліч вбудованого функціонала що дозволяє полегшити та пришвидшити написання коду. Також було показано розробку застосунку крок за кроком та зображено важливі аспекти при написанні проєкту, таких як: розуміння файлової системи роутингу, різниця між клієнтськими та серверними компонентами, найкращі практики для створених певних елементів. Усе було підкріплено зображеннями та лістингами коду програми.

Були описані ідеї для покращення якості проєкту та забезпечення кращої цілісності й підтримки проєкту. Наприклад інтеграції Docker та інші елементи для збільшення якості.

Розказано про альтернативу серверного рішення, яке з простотою можна імплементувати використовуючи Next.js та його можливість працювати як сервер.

Вибір Next.js як основи для проєкту вважаю вдалим, оскільки переваги його використання включають підтримку серверного рендерингу, автоматичну оптимізацію для SEO, зручну роботу з маршрутизацією та вбудовану підтримку TypeScript, що сприяє швидкій та ефективній розробці вебдодатків.

Отже, як результат виконання кваліфікаційної роботи було отримано клієнтську частину застосунку для роботи освітньої платформи. Перегляду її функціоналу та ознайомлення з подіями платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація для бібліотеки React.js. URL: https://react.dev/reference/react. (дата звернення: 20.10.2023 р.).

2. React JS – Architecture. URL: https://shorturl.at/eoxzA. (дата звернення: 27.10.2023 р.).

3. Документація для інструменту ESLint. URL: https://eslint.org/docs. (дата звернення: 28.10.2023 р.).

4. Конфігурація для Prettier. URL: https://shorturl.at/pFBVs. (дата звернення: 29.10.2023 р.).

5. How To Structure React Projects From Beginner To Advanced. URL: https://shorturl.at/PB2gm. (дата звернення: 05.11.2023 р.).

6. MDN web docs for JavaScript. URL: https://shorturl.at/KZgI5. (дата звернення: 07.11.2023 р.).

7. Документація для інструменту react-hook-form. URL: https://react-hook-form.com/docs. (дата звернення: 15.11.2023 р.).

8. Документація для бібліотеки SWR. URL: https://swr.vercel.app/docs/api. (дата звернення: 18.11.2023 р.).

9. Документація та для технології Tailwind CSS. URL: https://tailwindcss.com/docs/configuration. (дата звернення: 20.11.2023 р.).

10. Документація для метафреймворку Next.js. URL: https://nextjs.org/docs (дата звернення: 02.03.2024 р.).

11. Next.js 14.1 update blog. URL: https://nextjs.org/blog/next-14-1 (дата звернення: 04.03.2024 р.)

12. Start a clean Next.js project with TypeScript, ESLint and Prettier. URL: https://shorturl.at/St49L. (дата звернення: 05.03.2024 р.)

13. Репозиторій бібліотеки tailwind-merge. URL: https://shorturl.at/54Vhk. (дата звернення: 07.03.2024 р.)

14. Документація для бібліотеки Zod. URL: https://zod.dev. (дата звернення: 06.03.2024 р.)

15. High performance Node.js image processing with Sharp. URL: https://shorturl.at/GatoB. (дата звернення: 10.03.2024 р.)

16. GitHub репозиторій по PostCSS. URL: https://shorturl.at/qo0d3. (дата звернення: 11.03.2024 р.)

17. GitHub репозиторій для плагіну prettier-plugin-tailwindcss. URL: https://shorturl.at/EwR12. (дата звернення: 13.03.2024 р.)

18. Документація для препроцесора SASS. URL: https://sass-lang.com/documentation. (дата звернення: 14.03.2024 р.)

19. Utility types in TypeScript. URL: https://shorturl.at/oJmxs. (дата звернення: 16.03.2024 р.)

20. React-hook-form validation resolver documentation. URL: https://shorturl.at/DeF0y. (дата звернення: 17.03.2024 р.)

21. Документація бібліотеки перевикористовуваних компонентів. URL: https://ui.shadcn.com/docs. (дата звернення: 20.03.2024 р.)

22. Implementing OAuth 2.0 to React for User Authorization. URL: https://shorturl.at/7fv8l. (дата звернення: 22.03.2024 р.)

23. How to use LocalStorage in JavaScript. URL: https://shorturl.at/KUzv1. (дата звернення: 28.03.2024 р.)

24. Документація по стейт-менеджеру Zustand. URL: https://shorturl.at/ujM4t. (дата звернення: 02.04.2024 р.)

25. Getting started with Jest. URL: https://jestjs.io/docs/getting-started. (дата звернення: 06.04.2024 р.)

26. React testing library intro. URL: https://testing-library.com/docs/react-testing-library/intro. (дата звернення: 08.04.2024 р.)

27. How to configure Jest with TypeScript. URL: https://shorturl.at/3GA8D. (дата звернення: 12.04.2024 р.)

28. Understanding GitHub Actions. URL: https://shorturl.at/tPPkC. (дата звернення: 15.04.2024 р.)

29. How to automatically trigger GitHub Actions workflows. URL: https://t.ly/PkGGw. (дата звернення: 05.05.2024 р.)

30. Довідкова документація по Docker. URL: https://docs.docker.com/reference. (дата звернення: 20.05.2024 р.)

ДОДАТКИ

Додаток А. Лістинг сторінки подій

```
import { SearchInput } from '@/components/ui/input'
import { SelectFilters } from '@/components/SelectFilters'
import { cn } from '@/lib/utils'
import { eUK } from '@/app/_font/eUK'
import { BigEventCard, DefaultEventCard } from '@/components/Cards'
import { Event } from '@/types/event'
import { fetchData } from '@/lib/fetchData'
import { EventsContentRow } from
'@/app/(events)/events/_components/EventsContentRow'
import { AppliedFilter } from
'@/app/(events)/events/_components/AppliedFilter'

export default async function EventsPage({
 searchParams,
}: {
 searchParams: { [key: string]: string | string[] | undefined }
}) {
 let currentEvents = await fetchData<Event[]>({ url: '/event' })
 currentEvents = currentEvents.slice(0, 3)

 return (
   <div className='container mx-auto mt-10 max-w-screen-xl text-main'>
     <h2 className={cn('text-[30px] font-medium leading-9
tracking-tighter', eUK.className)}>
       Знайди те що тебе цікавить
     </h2>
     <div className='mt-4 flex items-center justify-between'>
       <SelectFilters />
       <SearchInput />
     </div>
     {Object.keys(searchParams).length > 0 ? (
       <AppliedFilter {...searchParams} />
     ) : (
       <>
         <div className='mt-10'>
           <h2 className={cn('mb-4 text-[30px] font-medium leading-9
tracking-tighter', eUK.className)}>
             Найближчі події
           </h2>
```

```
            <BigEventCard />
            <div className='mt-6 flex gap-6'>
              {currentEvents.map((event, index) => (
                <DefaultEventCard
                  title={event.title}
                  category={event.eventType}
                  price={1200}
                  contentType={event.eventType}
                  thumbnail={event.thumbnail}
                  key={event.id}
                  imgId={index + 1}
                  linkId={event.id}
                />
              ))}
            </div>
          </div>
          <EventsContentRow title='Цього місяця'
url='/event/get-this-month-events' />
          <EventsContentRow title='Наступного місяця'
url='/event/get-after-month-events' />
        </>
      )}
    </div>
  )
}
```