МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Інформаційна системи підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТдн-84жт Лисюк Олександр Анатолійович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____«__» _____2022 р.

Науковий керівник _____ к.т.н., доц., Парфененко Ю.В.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент_____
(підпис)

Суми-2022

Сумський державний університет
Центр заочної та дистанційної форми навчання
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. кафедри ІТ

_____ В. В. Шендрик
«06» жовтня 2021 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Лисюк Олександр Анатолійович

1 Тема роботи *Інформаційна система підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання*

керівник роботи *Парфененко Юлія Вікторівна, к.т.н., доцент*,

затверджені наказом по університету від «10» червня 2022 р. № 0434-VI

2 Строк подання студентом роботи «20» червня 2022 р.

3 Вхідні дані до роботи Результати аналізу предметної області, вимоги державних стандартів

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області, актуальність досліджуваної задачі, аналіз сучасних тенденцій, постановка задачі, моделювання предметної області, засоби розробки системи, опис програмної реалізації додатку, висновки, використані джерела, додаток А, додаток Б, додаток В.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Ілюстрації з інтерфейсом, кодом та функціоналом програми

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 6 жовтня 2021

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	06.12.21	
1.1	Актуальність досліджуваної задачі	17.12.21	
1.2	Аналіз сучасних тенденцій	22.12.21	
1.3	Постановка задачі	26.12.21	
2	Моделювання предметної області	02.01.22	
2.1	UML діаграми	12.01.22	
3	Засоби розробки системи	18.01.22	
4	Опис програмної реалізації вебдодатку	20.01.22	
4.1	Реалізація клієнтської частини	05.04.22	
4.2	Реалізація серверної частини	15.05.22	

Студент _____ Лисюк О.А.
(підпис)

Керівник роботи _____ к.т.н., доц. Парфененко Ю.В.
(підпис)

РЕФЕРАТ

Тема роботи «Інформаційна система підтримки взаємодії учасників навчального процесу під час очного онлайн-навчання»

Пояснювальна записка – 72 сторінки, кількість розділів 4, кількість рисунків - 72, кількість таблиць - 4, кількість використаних джерел – 10.

Метою дипломного проекту є розробка «Web-додаток підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання»

Проведено аналіз предметної області наявних рішень.

Під час роботі над кваліфікаційною роботою досліджено предметну область додатка.

У роботі розглянуто інформаційну систему підтримки взаємодії учасників навчального процесу, по принципу e-learning систем.

В результаті розроблено web-додаток для керування навчальним процесом в режимі онлайн

Ключові слова: Webstorm, ReactJs, ChakraUi.

ЗМІСТ

ВСТУП

E-learning – це скорочення від електронного навчання та охоплює будь-який тип навчання чи освіти, які відбуваються на цифровій платформі. Сфера електронного навчання різноманітна, вона може включати 20-хвилинний курс, який ви закінчите на своєму телефоні, університетську освіту, яку ви проходите на відстані.

Ви можете навчатися в будь-який час і на будь-якому електронному пристрої, включаючи: настільний комп'ютер, ноутбук або мобільний телефон, що робить його гнучким і легким для включення в наше повсякденне життя. Сьогодні люди спочатку шукають свої запити в Інтернеті, а не в книгах. Отже, це призвело до важливості електронного навчання в освіті. Наприклад, лекції в прямому ефірі є одним із засобів, які дозволяють учасникам висловити свою думку на певну тему, а потім обговорити їх детальніше. Також доступні статичні сторінки з матеріалами курсу.

Електронне навчання швидко розширюється, тому що воно гнучке, швидке та дає виняткові результати та пропонує простоту використання для створювачів навчального матеріалу, експертів предметної області, менеджерів. Навчальна програма може бути з легкістю оновлена, від тонкого налаштування слів до наповнення новим контентом та візуальних матеріалів. Це означає, що ваші матеріали завжди залишаються актуальними. Також він забезпечує набагато кращу масштабованість для широкої аудиторії.

Електронне навчання підвищує продуктивність та потребує на 40-60% менше часу студентів, у порівнянні з навчанням під керівництвом інструктора, яке дуже часто відриває людей від роботи на декілька годин, або днів та може потребувати транспортних витрат.

Електронне навчання закріплює пам'ять, тому що воно має візуальні та інтерактивні елементи. Окрім цього, дослідження показують, що учасники

електронного навчання вивчають в п’ять разів більше нового матеріалу, ніж при вивченні традиційним методом, без збільшення часу, що витрачається на навчання. Воно представляє цінні показники, які можуть використовуватися керівниками для перевірки та підвищення ефективності навчання. Окрім цього, студенти та їх менеджери можуть відстежувати індивідуальний прогрес та продуктивність.

Метою даного дослідження є розробка інформаційної системи підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання.

Для досягнення мети дипломної роботи ми маємо виконати наступні задачі:

- дослідити предметну область;
- порівняти наш майбутній продукт із його аналогами;
- вибрати архітектуру нашого проєкту.
- вибрати інструменти для реалізації нашого майбутнього веб застосунку;
- розробити візуальну та функціональну частину додатка;
- протестувати роботу застосунку.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність досліджуваної задачі

Система навчання, заснована на формалізованому навчанні, але з використанням електронних ресурсів відома як електронне навчання. У той час як навчання може проводитися в класі або поза ним, використання комп'ютерів та Інтернету є основним компонентом електронного навчання. Електронне навчання можна також назвати мережевою передачею навичок і знань, а навчання надається великій кількості одержувачів в один і той же або в різний час. Раніше це не приймалося повністю, оскільки передбачалося, що в цій системі немає людського фактора, необхідного для навчання.

Проте, завдяки швидкому розвитку технологій та розвитку систем навчання, нині він охоплений масами. Впровадження комп'ютерів стало основою цієї революції, і з часом, коли ми звикли до смартфонів, планшетів і т. д., ці пристрої тепер займають важливе місце в класах для навчання. Книги поступово замінюються на електронні навчальні матеріали, такі як оптичні диски або флешки. Знаннями також можна ділитися через Інтернет, який доступний 24 години на добу, 7 днів на тиждень, у будь-якому місці та у будь-який час.

Електронне навчання зарекомендувало себе як найкращий засіб у корпоративному секторі, особливо коли ТНК проводять навчальні програми для професіоналів по всьому світу, і співробітники можуть набувати важливих навичок, сидячи в залі засідань або відвідуючи семінари, які проводяться для працівників однієї чи різних організацій під одним дахом. Школи, що використовують технології електронного навчання, на крок попереду тих, у яких досі зберігається традиційний підхід до навчання.

Без сумніву, не менш важливо просувати концепцію неелектронного навчання за допомогою книг та лекцій, але важливість та ефективність навчання на

основі технологій не можна сприймати легковажно чи повністю ігнорувати.

Вважається, що людський мозок може легко запам'ятовувати та пов'язувати побачене і почуте за допомогою зображень, що рухаються, або відео. Також було виявлено, що візуальні образи не лише утримують увагу учня, а й утримуються мозком протягом тривалого часу. Різні сектори, включаючи сільське господарство, медицину, освіту, послуги, бізнес та державні установи, адаптуються до концепції електронного навчання, що сприяє прогресу нації.

1.2 Аналіз сучасних тенденцій

1.2.1 Ключові переваги

Переваги, які більшість людей пов'язують із електронним навчанням у порівнянні зі звичайним навчанням:

- Швидший потенційний темп доставлення
- Найменший вплив на навколишнє середовище через меншу кількість друку та поїздок для учнів
- Завдяки онлайн-навчанню студенти мають змогу отримувати доступ з будь-якої точки світу та в будь-який час. Електронне навчання також рентабельно; компанії значно економлять на проїзді, проживанні студентів та викладачів.
- Сучасні учні надають перевагу саме інтерактивному контенту. Інструменти електронного навчання дозволяють дизайнерам, що навчаються, роботи контент більш інтерактивним. Чим цікавіший зміст, тим краще учні запам’ятовують інформацію.
- На очних заняттях кожен викладач має свій власний метод навчання. Кожен відрізняється за підходом і стилем та схильний до помилок. Цю проблему вирішує саме онлайн-навчання, яке щоразу забезпечує стандартизоване та

упорядковане навчання. Кожен учень проходить через той самий досвід, незалежно від того, коли й де він або вона проходить курс.

- Онлайн-навчання масштабується. Його можна розвернути для будь-якої кількості студентів, що є одноразовою інвестицією.
- Кожен учень має унікальні цілі та переваги навчання. Електронне навчання дає змогу задовольнити індивідуальні потреби. Це дозволяє учням вибирати свій шлях навчання та орієнтуватися у своєму власному темпі. [8]

1.2.2 Наявні аналоги систем підтримки взаємодії учасників навчального процесу.

- Coursera –сервіс який співпрацює з музеями, університетами та іншими установами, щоб запропонувати студентам безплатні заняття на виняткову різноманітність тем. Студенти можуть переглянути список доступних тем, а коли вони відповідають на це запитання, їх переведуть до списку доступних курсів на цю тему. Студенти, які нервують через те, що влаштовані через голову, можуть розслабитися. Серсів Coursera пропонує багато інформації про кожен клас. Це включає:
    - Програма курсу.
    - Формат курсу.
    - Рекомендований досвід та досвід.
    - Необхідні матеріали.
    - Коротка інформація про курс.
    - Студенти, які закінчили курс, часто можуть отримати від викладача заяву про досягнення.
- W-3 Schools - це безплатний вебсайт електронного навчання, який присвячений навчанню студентів різним аспектам вебдизайна. Студенти вибирають те, чому вони хочуть навчатись, наприклад:

- HTML.
- PHP.
- SQL.
- JQuery.

Щодо кожної концепції, яку студенти хочуть освоїти, вони проходять різноманітні онлайн-підручники, складають тести та в кінцевому підсумку завершують кожен курс. Студенти можуть скласти підсумковий іспит, щоб підтвердити свою майстерність, і якщо вони сплатять додатковий внесок, то вони отримають сертифікат про закінчення.

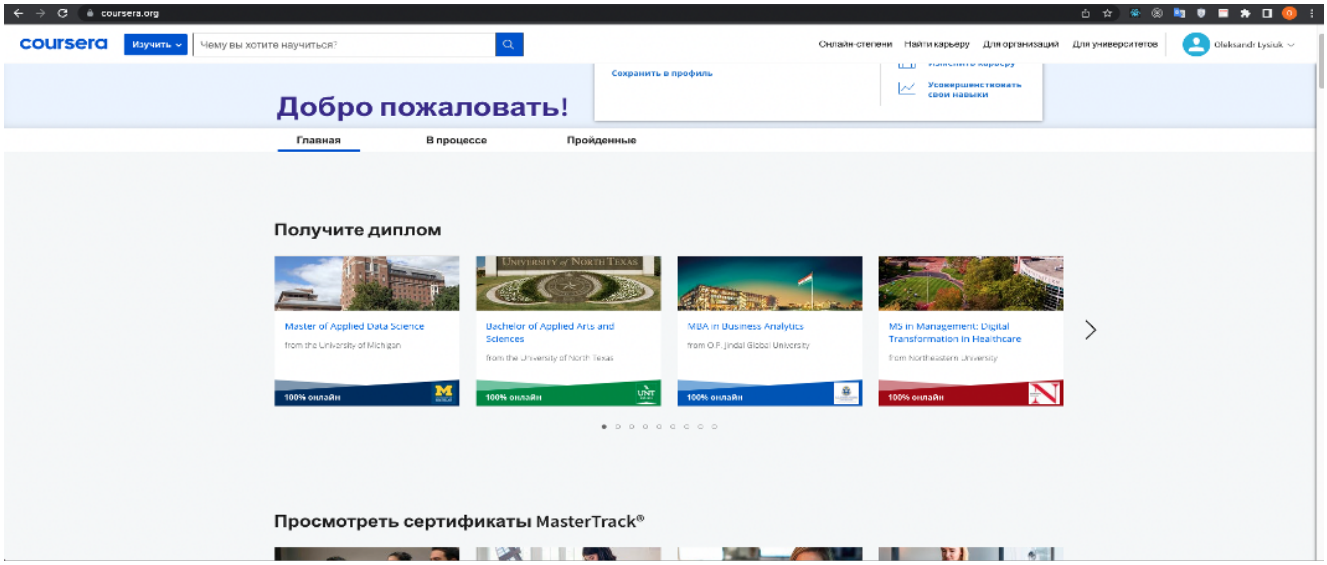


Рисунок 1.1 – Вигляд головної сторінки веб-застосунку coursera.org

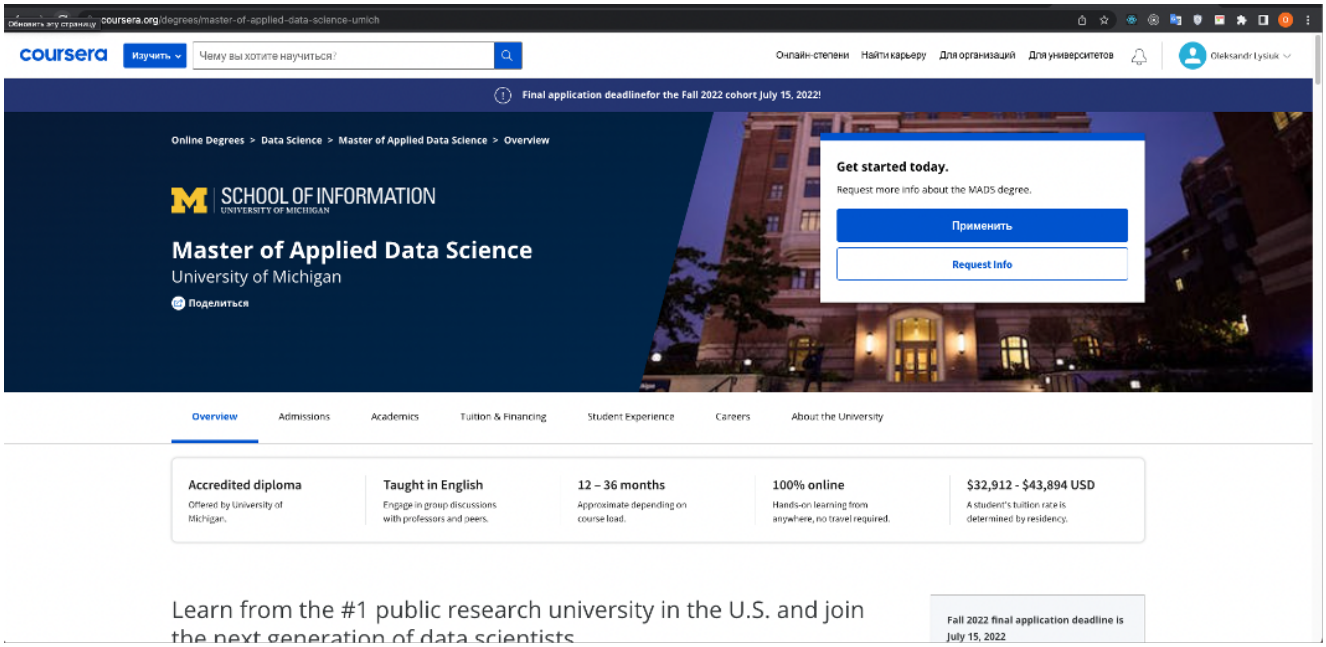


Рисунок 1.2 – Вигляд сторінки навчального закладу

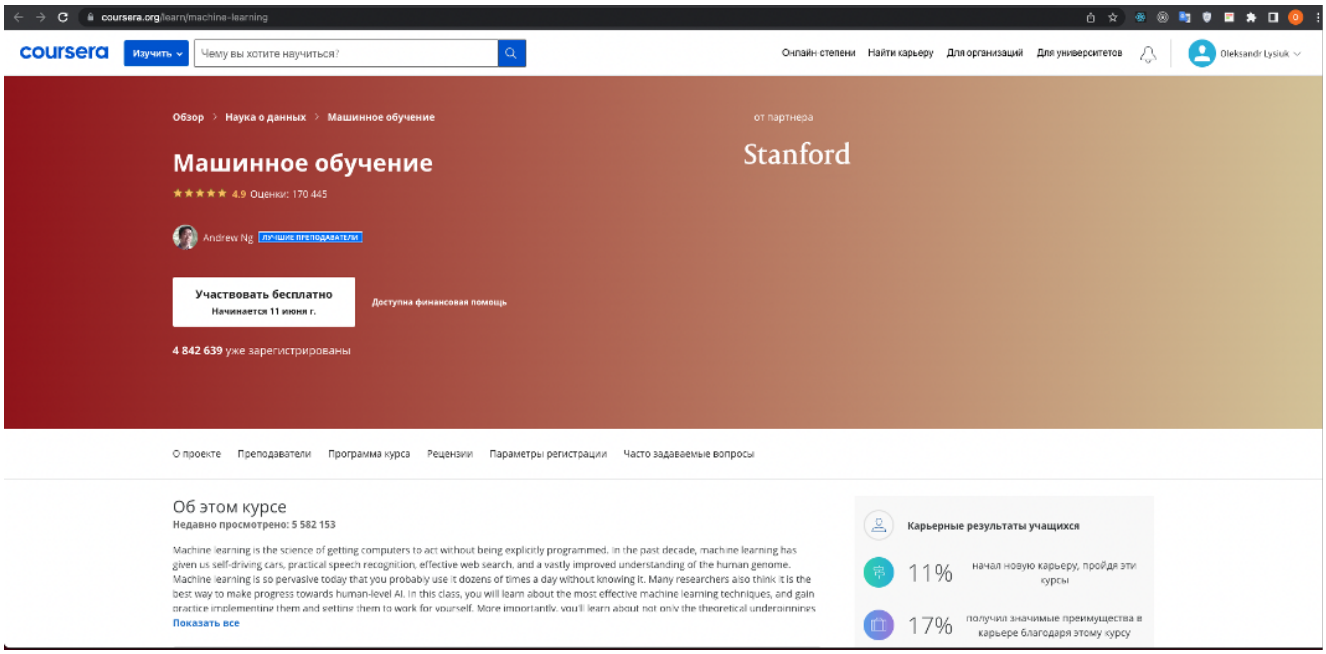


Рисунок 1.3 – Сторінка навчального курсу

- Codeacademy сервіс дозволяє студентам вибрати свою мету навчання, а потім рекомендує відповідний курс для цього студента. Codecademy працює, оскільки робить кодування доступним для будь-якого зацікавленого студента, надає практичні рекомендації для студентів, які хочуть навчитися програмувати, але

не розуміють, як ці нові навички можуть застосовуватися до їхньої поточної роботи. Codeacademy має широкий спектр пропозицій, які містять навчальні посібники по HTML, CSS, Sass, Rails, JavaScript, ReactJs, SQL, Java, Ruby. Окрім того, є наявні курси по веб-дизайну, веб-розробці, структурам даних, розробці мобільних додатків та навіть ігор.

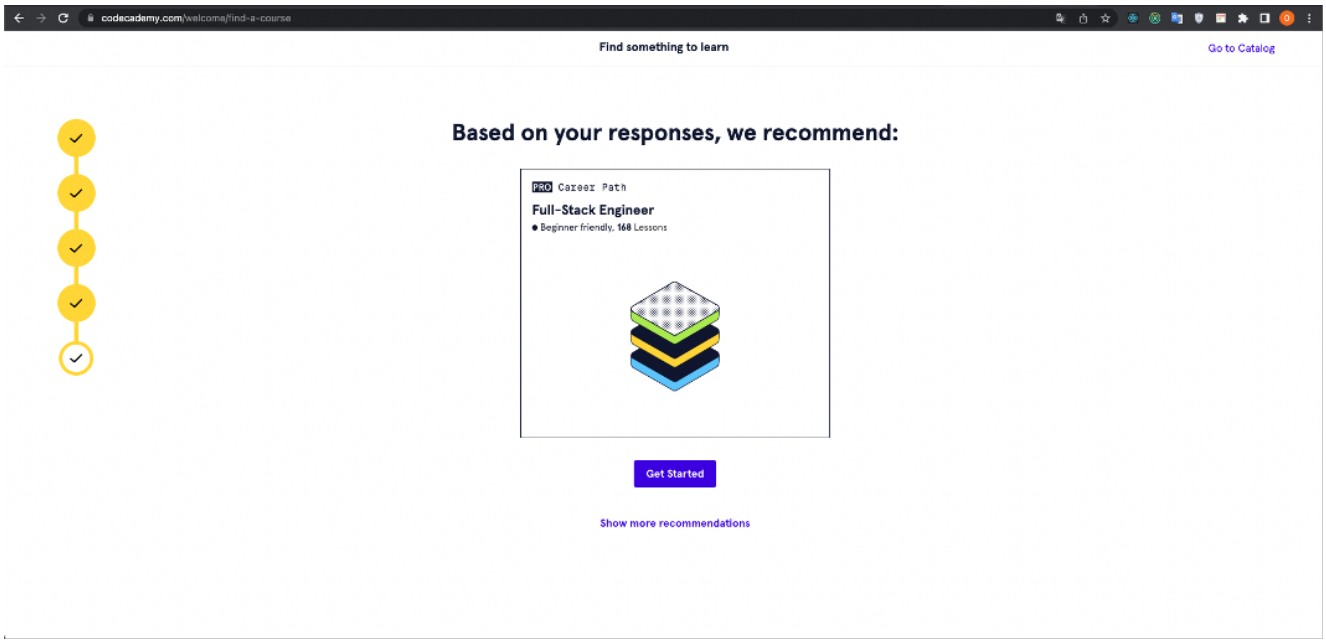


Рисунок 1.4 – Сторінка з рекомендованими курсами

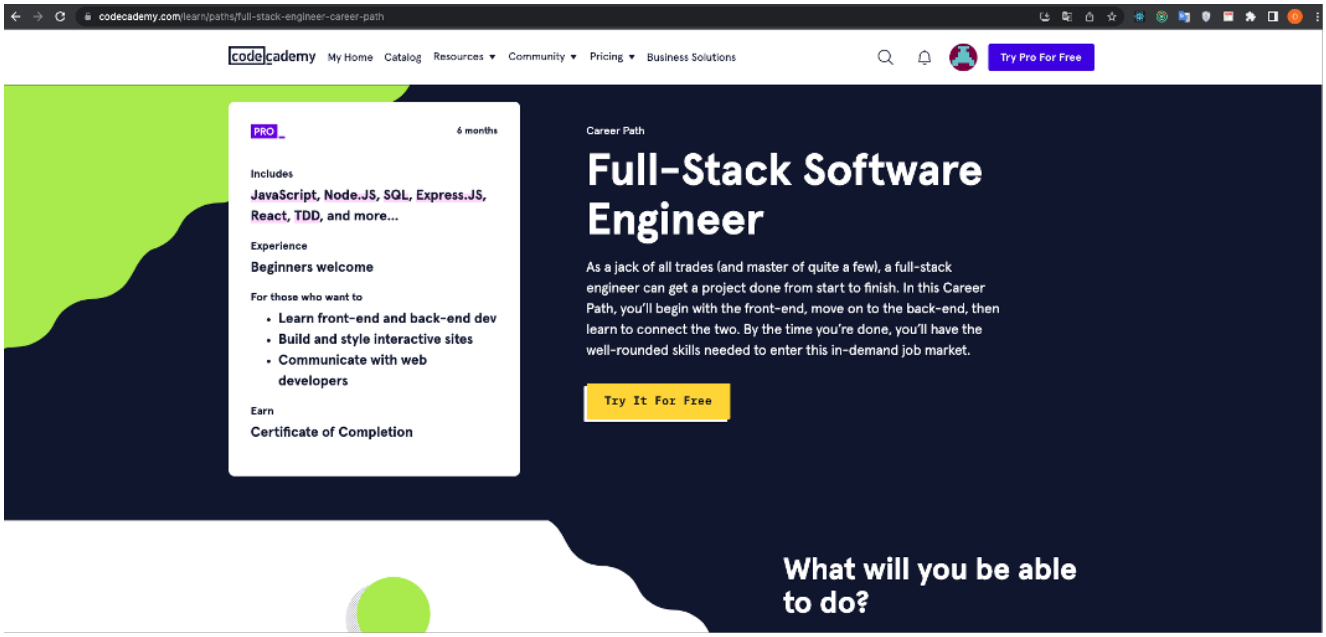


Рисунок 1.5 – Вигляд сторінки певного курсу

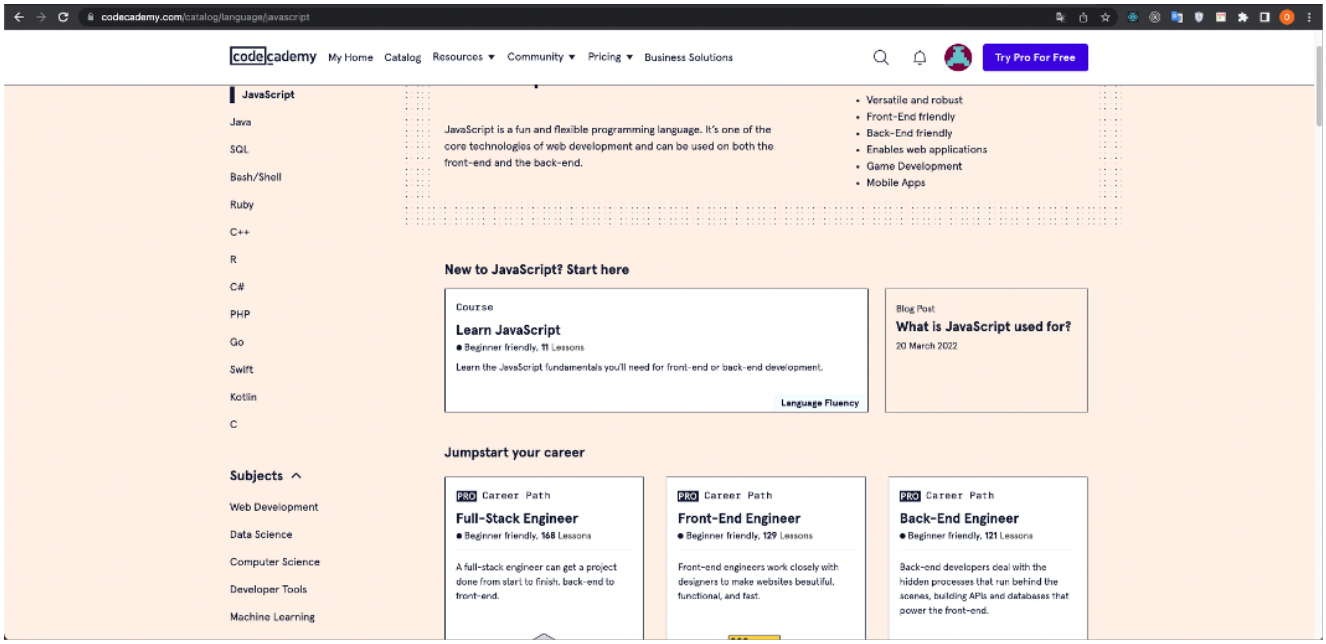


Рисунок 1.6 – Сторінка всіх доступних курсів за обраною технологією

Рисунок 1.7 – Огляд сторінки з матеріалами курсу

- Elucidat – це повністю хмарна та потужна авторська платформа, що дозволяє будь-якому користувачеві проводити високоякісне онлайн навчання на будь-якому пристрої. Завдяки інтелектуальному, але простому інтерфейсу,

оптимізованим процесам та інструментам, які виконують важку роботу, ви можете проводити навчання в декілька разів швидше і простіше, ніж будь-коли раніше. Даний сервіс надає можливість прослуховування подкастів, перегляду матеріалу у вигляді звичайного тексту.

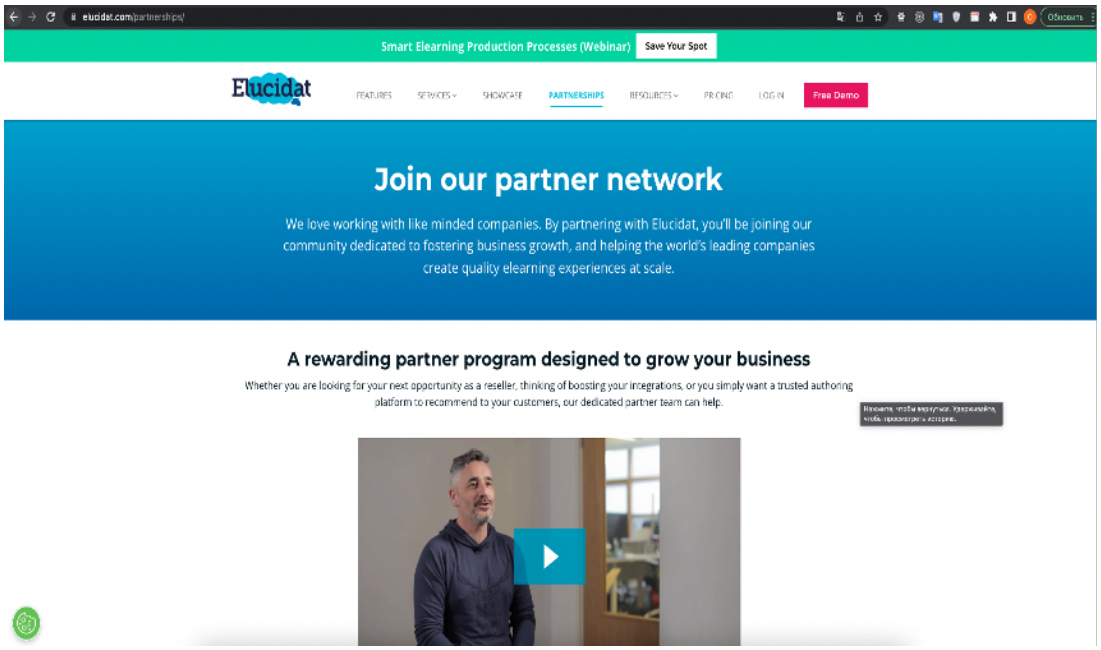


Рисунок 1.8 – Огляд головної сторінки

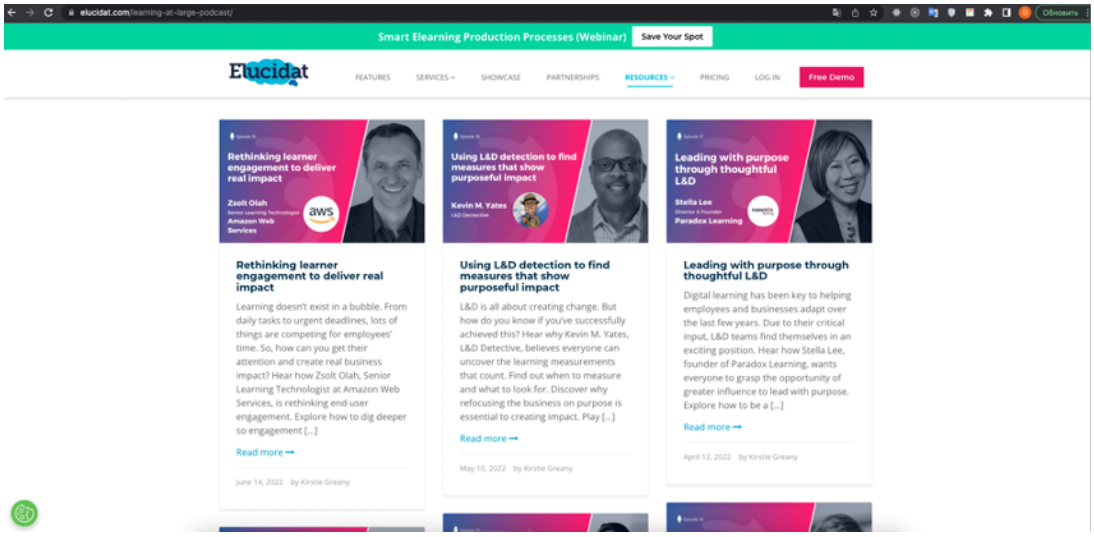


Рисунок 1.9 – Огляд сторінки подкастів

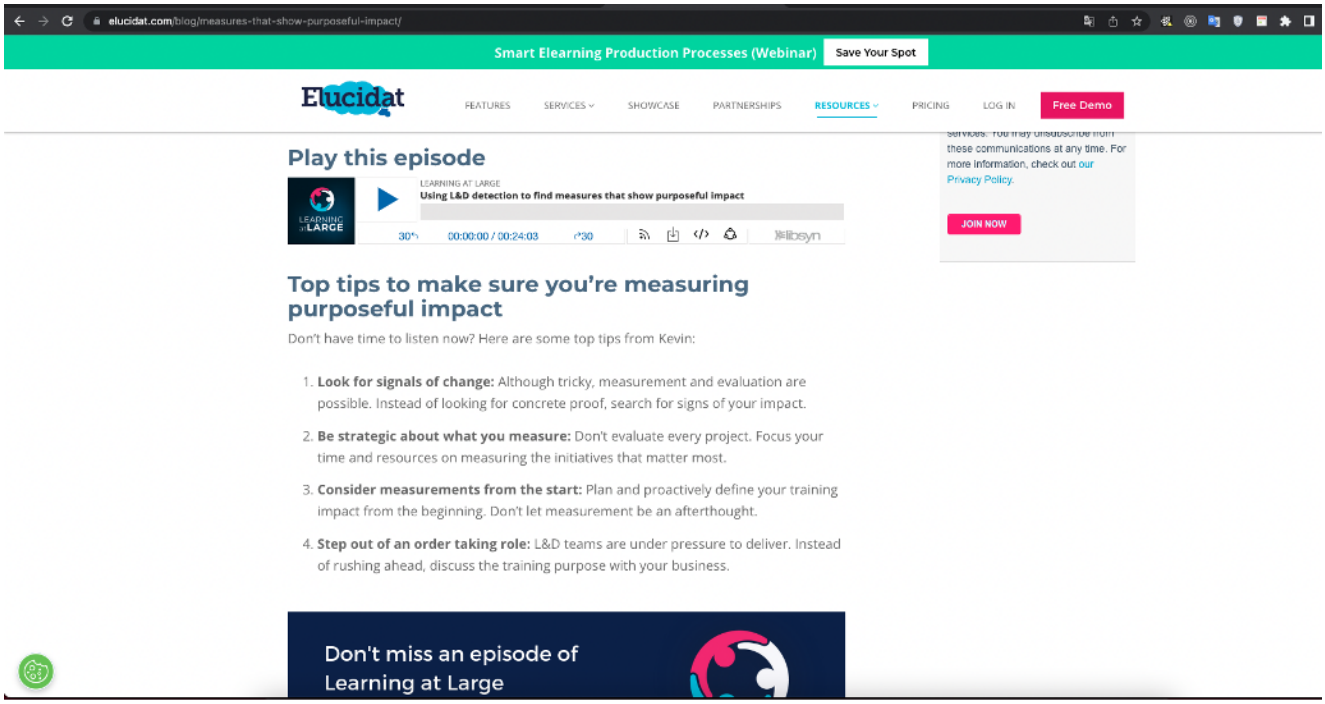


Рисунок 1.10 – Огляд сторінки подкаста з навчальним матеріалом

Таблиця 1.1 - Порівняльна таблиця характеристик аналогів веб-застосунків

Характеристика	“coursera”	“codeacademy”	“elucidat”
Зручний інтерфейс	+	+-	-
Навігація	+	+	+
Гнучкість конфігурування	-	+-	+
Швидкість	+	-	+
Функціональність	+-	+	+
Інтерактивність	+	+-	+
Сучасний дизайн	+	+	+-
Наявність обов’язкової реєстрації	+	+	-

1.3 Постановка задачі

Розроблюваний вебдодаток складається з двох основних частин: SPA-додаток, та store, які в свою чергу працюють одночасно і використовують так звану FLUX архітектуру.

Етапи розробки веб-додатку:

- Створення загального уявлення про роботу додатку.
- Встановлення необхідних модулів для розроблення додатку.
- Створення логічної структури проекту.
- Розробка UI, тобто інтерфейсу додатку додатку.
- Написання логіки для правильного функціонування UI частини.
- Написання логіки для взаємодії з вхідними даними.
- Підключення Store(Redux-toolkit).
- Створення slice, actions, selectors, reducers, utils.
- Підключення запитів до сервера за допомогою Redux-thunk.
- Налаштування взаємодії між додатком та стором.
- Чистка так званого «мертвого коду» для більш швидкої роботи додатку.
- Пошук та усунення проблем з PSI за допомогою Google PageSpeedInside.
- Фінальне тестування додатку.
- Завантаження на хостинг.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Перехід із когнітивної лінгвістики до світу програмної інженерії в цілому та об’єктноорієнтованого програмування зокрема має свої переваги. Обидва світи спираються на конструкції концептуалізації, які, хоча й мають різну реалізацію та контекст, мають спільні властивості.

Предметна область – це організоване та структуроване зображення проблеми. Вона повинна представляти словниковий запас і ключові поняття проблемної області, а також визначати відносини між усіма сутностями в межах області

Модель предметної області є чудовим інструментом, щоб створити спільну мову та фундаментальну структуру, важливу для аналізу особливостей та епохи. Предметна область визначається та постійно перебудовується в міру того, як знання про модель покращуються, а функціональність системи розвивається.

2.1 Проектування інформаційної системи

UML – це спосіб візуалізації програмної програми за допомогою набору діаграм. Позначення розвинулося на основі робіт Грейді Буча, Джеймса Рамбо, Івара Джейкобсона та Rational Software Corporation для використання для об’єктно-орієнтованого проектування, але відтоді воно було розширено, щоб охопити ширший спектр проектів програмної інженерії. Сьогодні UML прийнято Групою управління об’єктами (OMG) як стандарт розробки програмного забезпечення для моделювання.

UML означає Unified Modeling Language. UML 2.0 допоміг розширити оригінальну специфікацію UML, щоб охопити ширшу частину зусиль з розробки програмного забезпечення, включаючи гнучкі методи.

Покращена інтеграція між структурними моделями, такими як діаграми класів, і моделями поведінки, такими як діаграми діяльності.

Додано можливість визначати ієрархію та де компонувати програмну систему на компоненти та під компоненти.

Оригінальний UML визначав дев'ять діаграм; UML 2.x доводить це число до 13. Чотири нові діаграми називаються: комунікаційна діаграма, складена структура структури, діаграма огляду взаємодії та часова діаграма. Він також перейменував діаграми діаграм станів на діаграми станів, також відомі як діаграми станів.

Ці діаграми організовані у дві окремі групи: структурні діаграми та діаграми поведінки чи взаємодії.

Структурні діаграми UML:

- Схема класів.
- Схема пакета.
- Діаграма об’єкта.
- Схема компонента.
- Схема розгортання.

Поведінкові діаграми UML:

- Діаграма діяльності.
- Схема послідовності.
- Схема варіантів використання.
- Діаграма стану.
- Схема зв’язку .
- Оглядова діаграма взаємодії.
- Часова діаграма.

2.1.1 Робота додатку на прикладі діаграм.

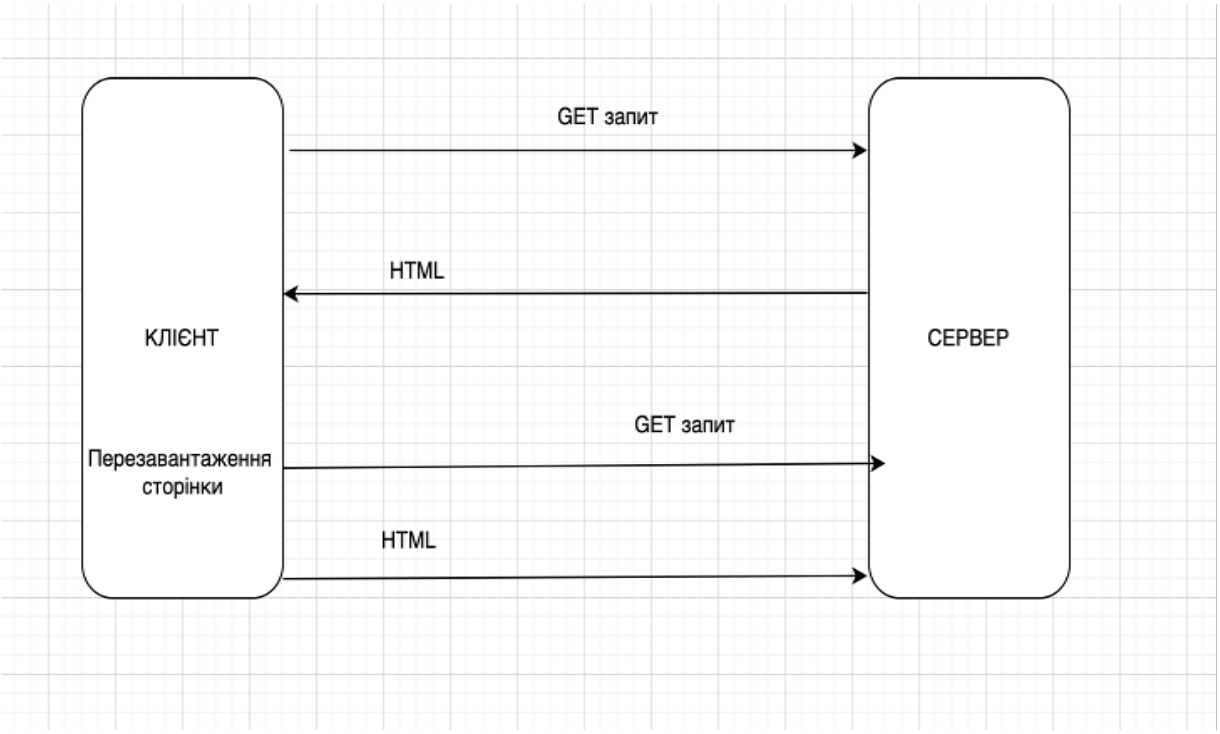


Рисунок 2.1. - Діаграма роботи традиційного вебсайту

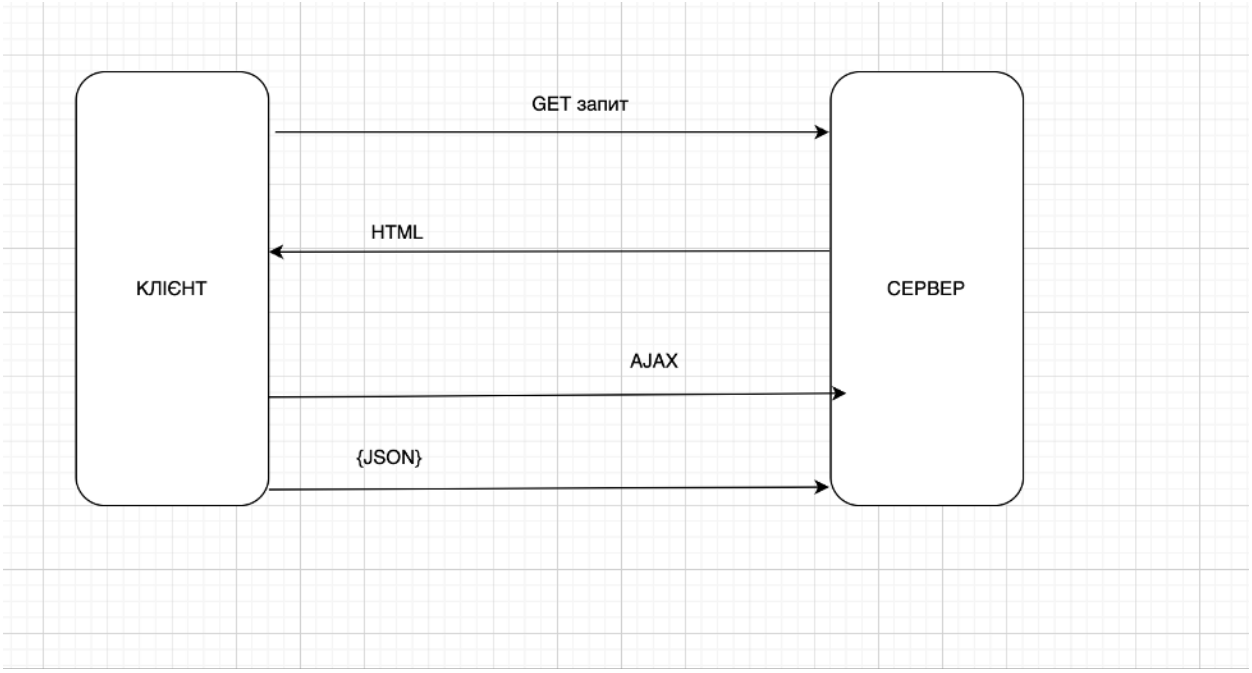


Рисунок 2.2 - Діаграма роботи SPA додатку

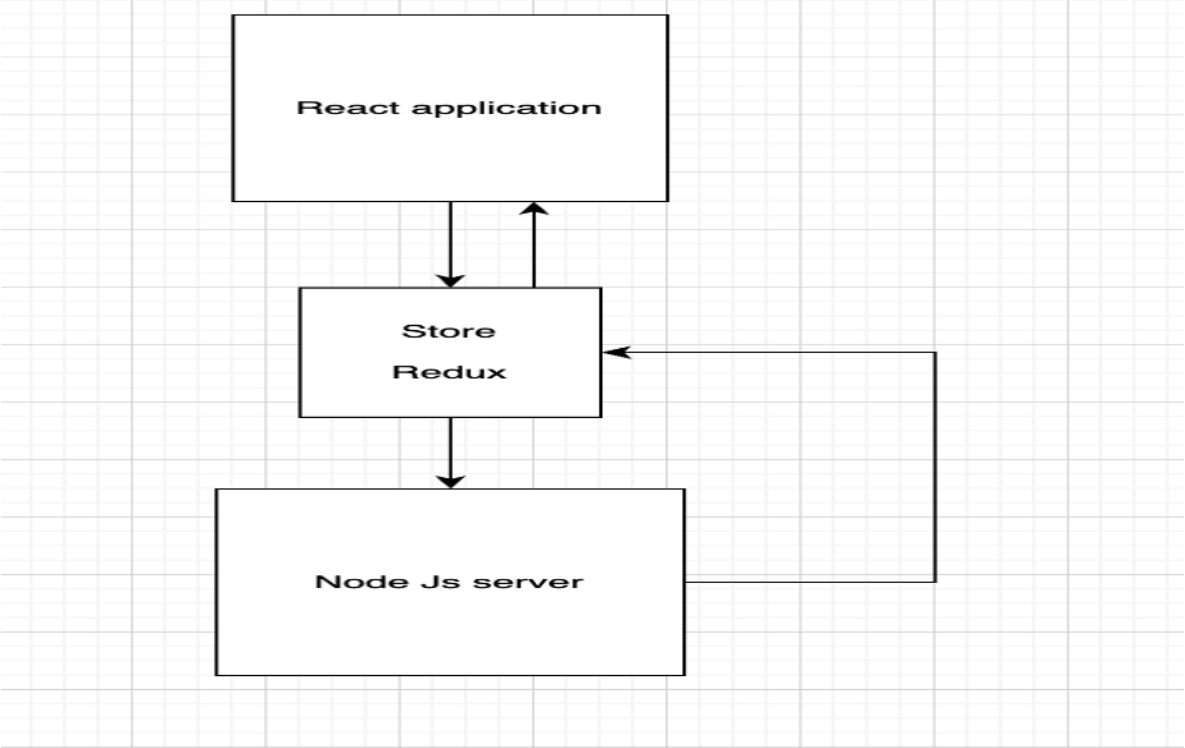


Рисунок 2.3 - Діаграма компонентів розробленого SPA додатку

На діаграмі компонентів 13 рисунку, ми можемо побачити наступні складові системи:

- SPA додаток на React
- Redux store
- Сервер на NodeJs

Опис портів сервісів:

- SPA додаток - 3000;
- NodeJs сервер - 5000.

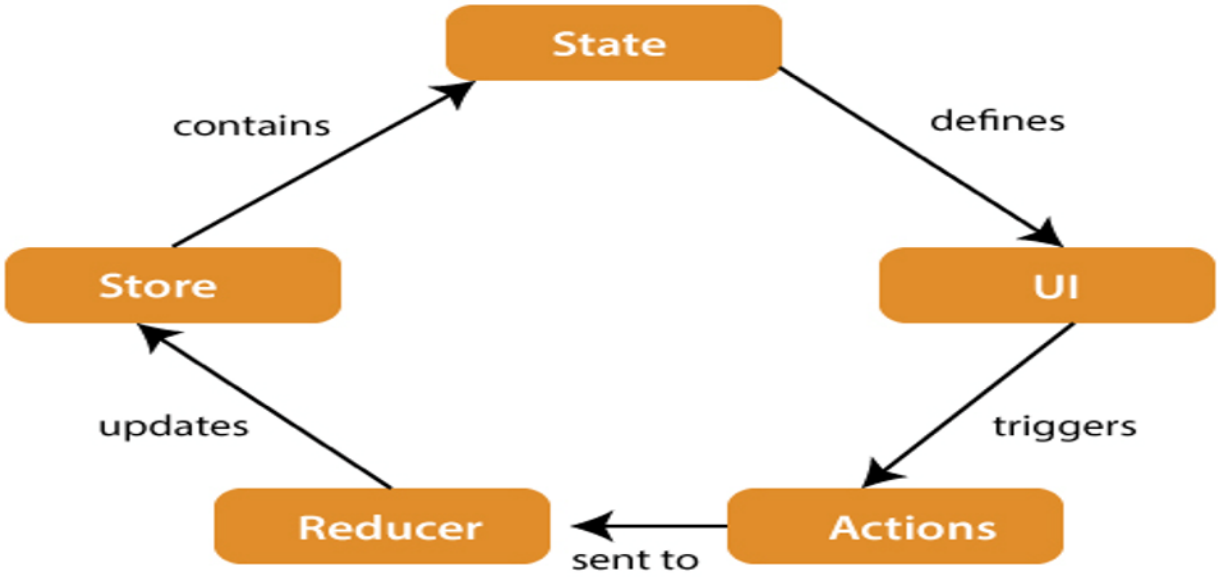


Рисунок 2.4 - Діаграма роботи React та Redux

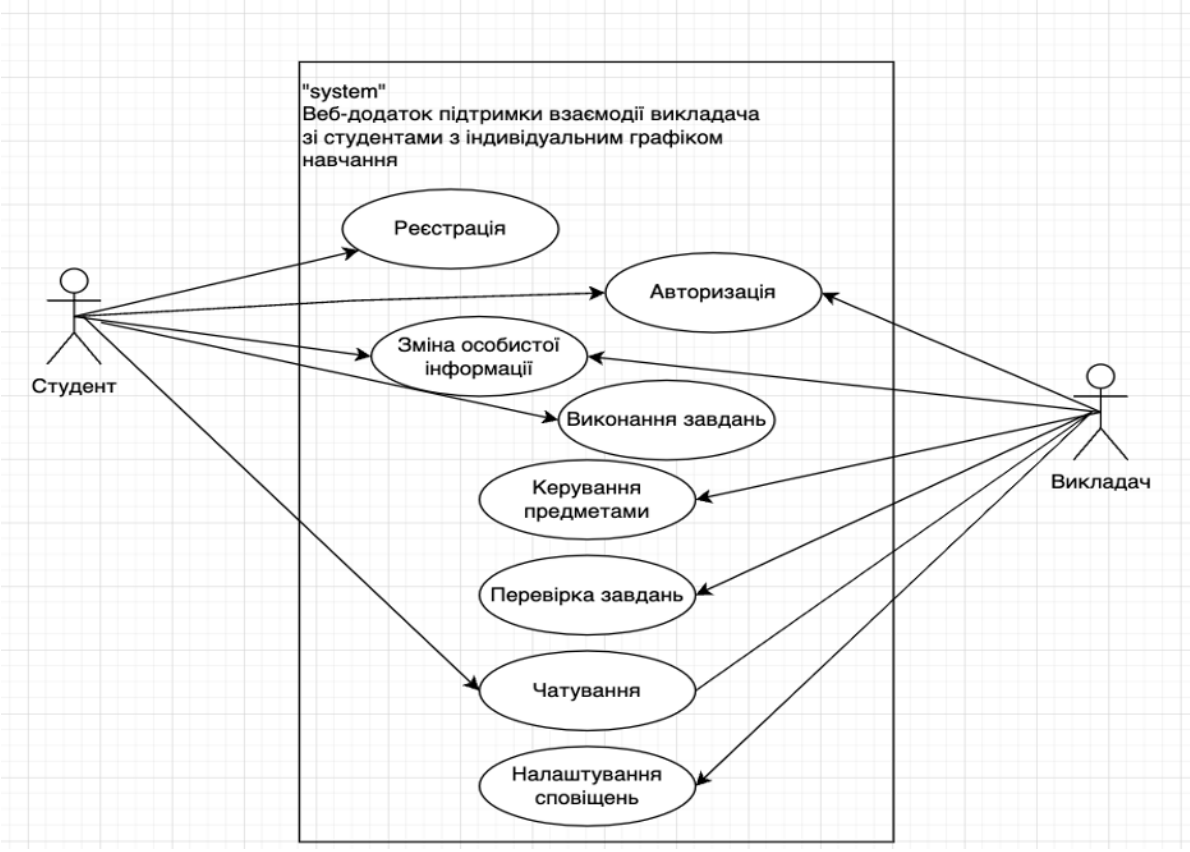


Рисунок 2.5 - Use case діаграма користувача та адміністратора

Починаючи використовувати додаток, користувач має можливість зареєструватися, або ж авторизуватися у створений раніше профіль. Реєстрація/авторизація може відбуватися за допомогою електронної пошти.

Додаток має декілька типів користувачів, а саме: звичайний користувач, адміністратор.

2.1.2. Функціональне моделювання веб-додатку в IDEF0

IDEF – це методологія графічного моделювання процесів, яка використовується для реалізації програмного забезпечення та систем інженерів. Ці методи використовуються для функціонального моделювання даних та об’єктно орієнтованого аналізу та отримання знань

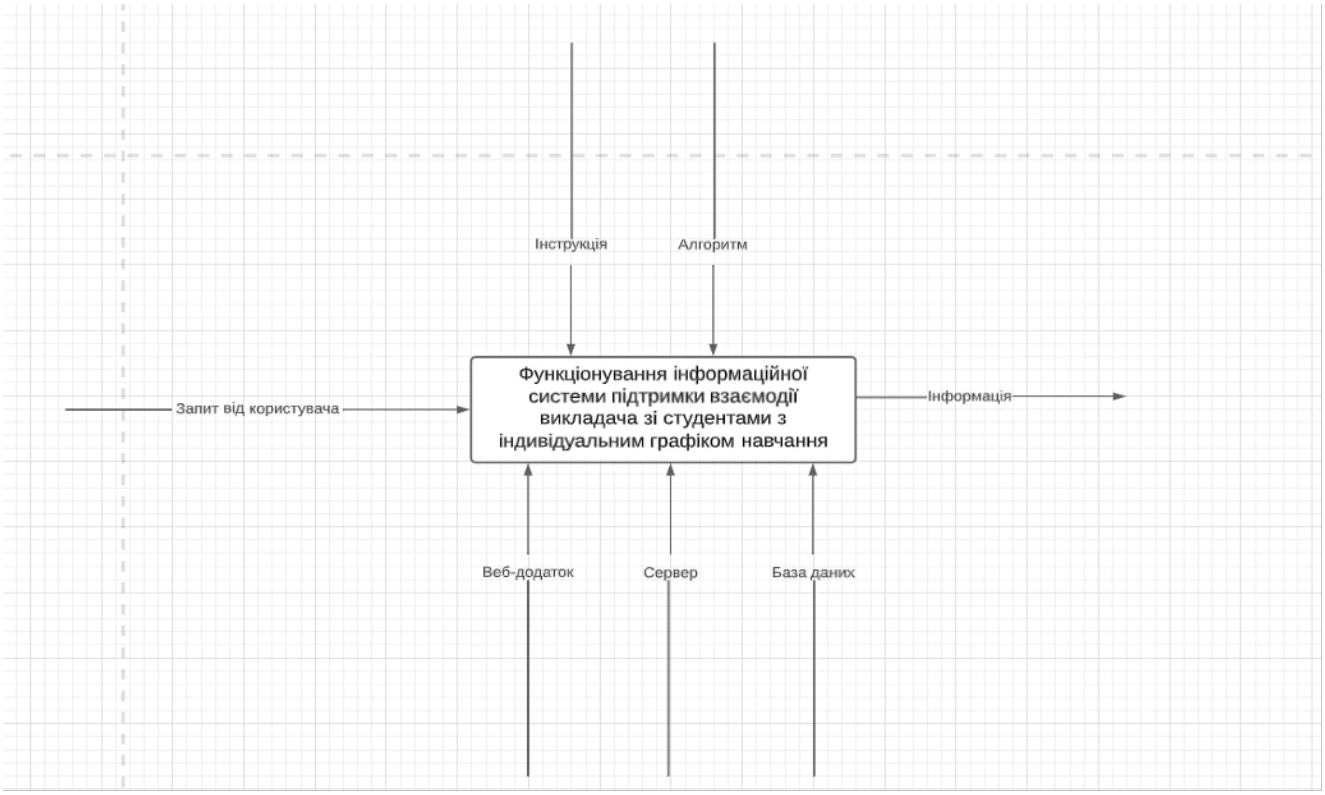


Рисунок 2.6 - Контекстна діаграма

На рисунку 2.7 представлена декомпозиція контекстної діаграми веб-додатку підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання у нотації IDEF0.

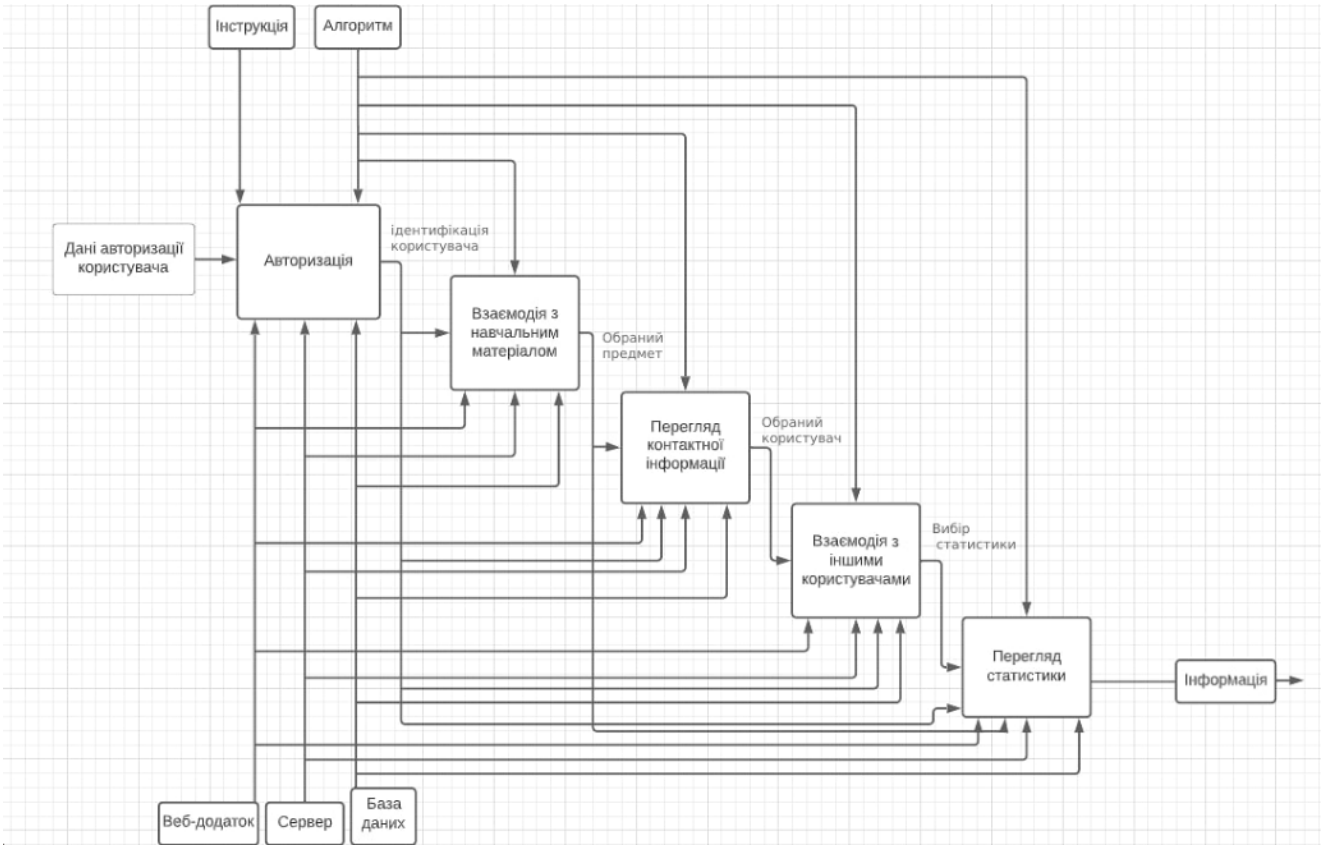


Рисунок 2.7 - Діаграма декомпозиції

Таблиця 2.1 – Можливості по ролям

Роль	Адміністратор	Користувач
Опис	Має можливість створювати предмети, додавати та редагувати вже існуючі, приймати завдання на перевірку та інше.	Має змогу ознайомлюватись з матеріалом, листуватися з адміністраторами веб-додатку, та здавати завдання на перевірку.
Відповідальність	Додавання, зміна, видалення, а також редагуваня предметів, матеріалів до них.	Перегляд предметів та матеріалів до них.

Таблиця 2.2 - Позиціювання продукту

Необхідно	Інформаційну систему підтримки взаємодії учасників навчального процесу.
Деталі	Використані технології: SPA-додаток був розроблений за допомогою JavaScript бібліотеки React, та Redux-toolkit. UI частина була оформлена бібліотекою ChakraUI.
Огляд аналогів	Було оглянуто велику кількість веб-додатків та вебсайтів, але ні один із них не мав весь функціонал який буде реалізовано в нашому додатку.

2.2 Проектування бази даних

Бази даних є основою всіх сучасних інформаційних систем. Оскільки комп’ютери зберігають сучасні бази даних, дані можуть бути будь-якого розміру та складності. Існує багато способів збору й упорядкування даних залежно від використання та типу даних.

База даних інформаційної системи складається з таких таблиць:

- Таблиця «users» - таблиця яка містить в собі інформацію про користувачів нашого застосунку, роль.
- Таблиця «chats» - таблиця яка зберігає наявні чати, користувачів які в ньому присутні, а також ідентифікатори повідомлень.
- Таблиця «messages» - містить повідомлення, час та дату створення, а також ідентифікатор відправника.
- Таблиця «files» - зберігає файл, часу та дату його завантаження.

Структура бази даних проілюстрована на рисунку 2.8

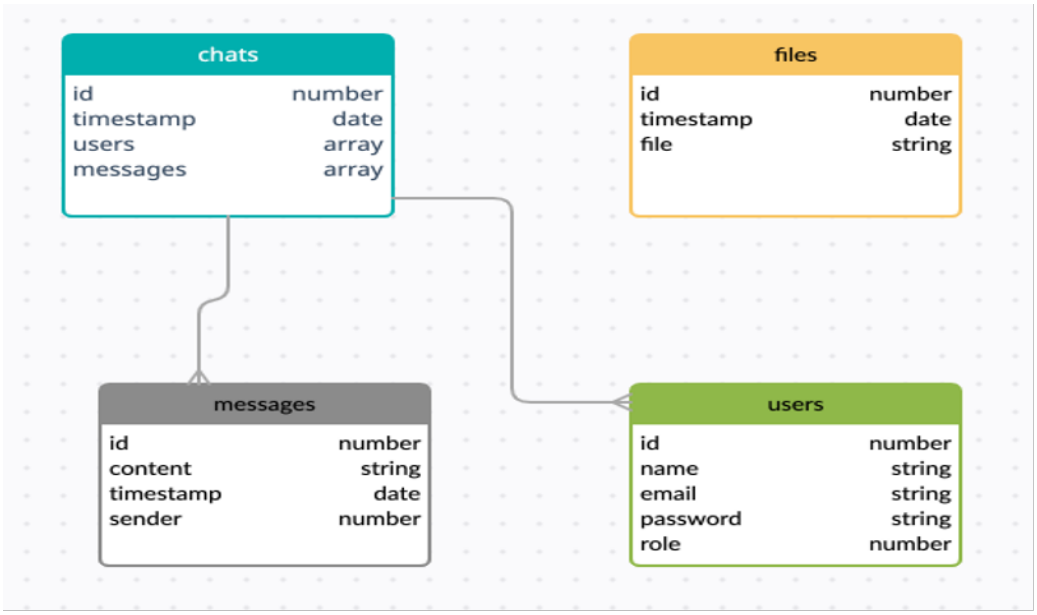


Рисунок 2.8 – Структура бази даних

РОЗДІЛ 3
ЗАСОБИ РОЗРОБКИ СИСТЕМИ

Для розробки веб-додатку було використано MERN стек.

Для початку, MERN стек означає:

- MongoDb
- ExpressJs
- ReactJs
- Node.js

Стек MERN був створений, щоб дати розробникам повного стека можливість розробляти сайт від початку до кінця, не маючи знання або використання інших навичок. Таким чином, розробники можуть створювати веб-сайт або веб-додаток від початку до кінця і керувати як інтерфейсом, так і сервером. Стек MERN

забезпечує можливість освоєння як алгоритмічної, так і логічної серверної частини, а також дизайну, користувацького досвіду та анімаційних компонентів передньої частини.

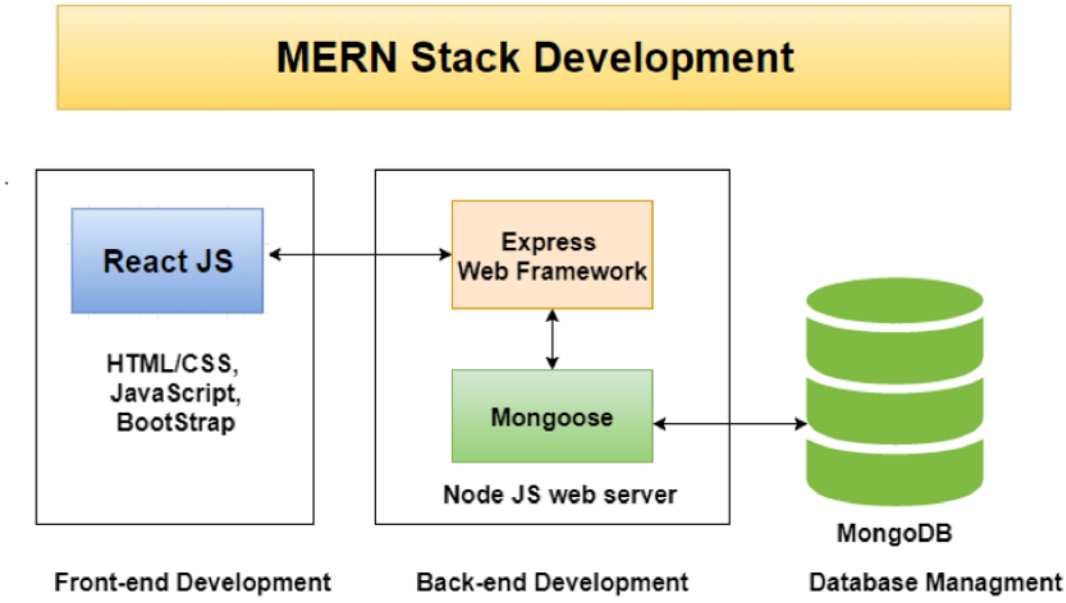


Рисунок 3.1 - Технології MERN стеку

Для реалізації вебдодатку було використано:

- IDE Webstorm - інтегроване середовище розробки для JavaScript, HTML, CSS;
- Javascript - мова програмування з динамічною типізацією;
- ReactJs - відкрита javascript бібліотека для створення інтерфейсів користувача;
- Redux-toolkit - javascript бібліотека для управління станом програм;
- Node.js - платформа з відкритим кодом для виконання; високопродуктивних мережевих стосунків, написаних на мові Javascript;
- Express.js - javascript фреймворк для Node.js застосунків;
- MongoDb - документо-орієнтована СУБД;
- Git - система керування версіями файлів для спільної роботи;
- TypeScript - мова програмування, яка поширює можливості javascript, методом додання суворої типізації;
- Webpack - пакувальник з відкритим кодом статичних модулів;

- Chakra UI - UI бібліотека.
- Axios - бібліотека для виконання запитів до серверу.

Середовищем розробки було обрано IDE Webstorm через його широкий спектр необхідного функціоналу та зручність.

Мовою програмування було обрано Javascript та Typescript, що дозволяє використовувати сувору типізацію, яка значно покращує якісь нашого коду.

Для адекватного керування даними в додатку, було обрано state-manager Redux-toolkit.

Пакувальником модулів було обрано Webpack який аналізує модулі, створює графи залежностей, та збирає їх в правильному порядку.

3.1 IDE Webstorm

Webstorm – це інтегроване середовище для розробки на JavaScript та пов'язаних з ним технологіях. Як і інші IDE JetBrains, WebStorm дозволяє автоматизувати рутинну роботу та легко справлятися зі складними завданнями, роблячи розробку цікавішою. Розширення які будуть використовуватися для написання проєкту:

- .env files support - необхідний для підтримки .env файлів, які використовуються для приватної інформації.
- GitToolBox - для зручної роботи з git.
- Import cost - зображення ваги імпортів в наших React компонентах.
- Prettier - форматує код.

Рисунок 3.2 - Розширення IDE Webstorm

3.2 React.js

React — це бібліотека, яка допомагає розробникам створювати користувацькі інтерфейси (UI) у вигляді дерева невеликих частин, які називаються компонентами. Компонент — це суміш HTML і JavaScript, яка фіксує всю логіку, необхідну для зображення невеликого розділу більшого інтерфейсу користувача. Кожен з цих компонентів можна об’єднати в послідовні складні частини програми. Решта лише деталі. У цьому першому розділі ви розглянете React, оскільки він пов’язаний з JavaScript і HTML. Для тих, хто працює з кодом, цей розділ допоможе підготувати основу для деяких концепцій наступних розділів. Якщо ви не працюєте з кодом, це

нормально; ви можете пропустити цей розділ і перейти до більш концептуальних розділів. [3]

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Привіт, {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Петро" />,
  document.getElementById('hello-example'),
);
```

Рисунок 3.3 - Приклад простого класового компоненту

3.3 Node.js

Node.js використовує архітектуру "Single Threaded Event Loop" для одночасної обробки кількох клієнтів. Щоб зрозуміти, чим це відрізняється від інших середовищ виконання, нам потрібно зрозуміти, як багатопотокові паралельні клієнти обробляються в таких мовах, як Java.

У багатопотоковій моделі запит-відповідь кілька клієнтів надсилають запит, і сервер обробляє кожен з них, перш ніж надіслати відповідь назад. Однак, для обробки одночасних дзвінків використовується кілька потоків. Ці потоки визначені в пулі потоків, і кожного разу, коли надходить запит, для його обробки призначається окремий потік [9].

Особливості:

- Масштабованість - забезпечує широку масштабованість додатків. Node.js, будучи одно поточний, здатний обробляти безліч одночасних підключень з високою пропускною здатністю.
- Швидкість – деблокуючи виконання потоків робить Node.js ще швидше та ефективніше.
- Пакети. Доступний широкий набір пакетів Node.js з відкритим вихідним кодом, які можуть спростити вашу роботу. Сьогодні в екосистемі NPM налічується понад мільйон пакетів.
- Сильний сервер - Node.js написаний на C і C++, що робить його швидким і додає такі функції, як підтримка мережі.
- Мультиплатформність – кросплатформова підтримка дозволяє створювати вебсайти SaaS, настільні програми та навіть мобільні програми з використанням Node.js.
- Зручний у супроводі – Node.js – це простий вибір для розробників, оскільки інтерфейс та серверна частина можуть керуватися за допомогою JavaScript як однієї мови.

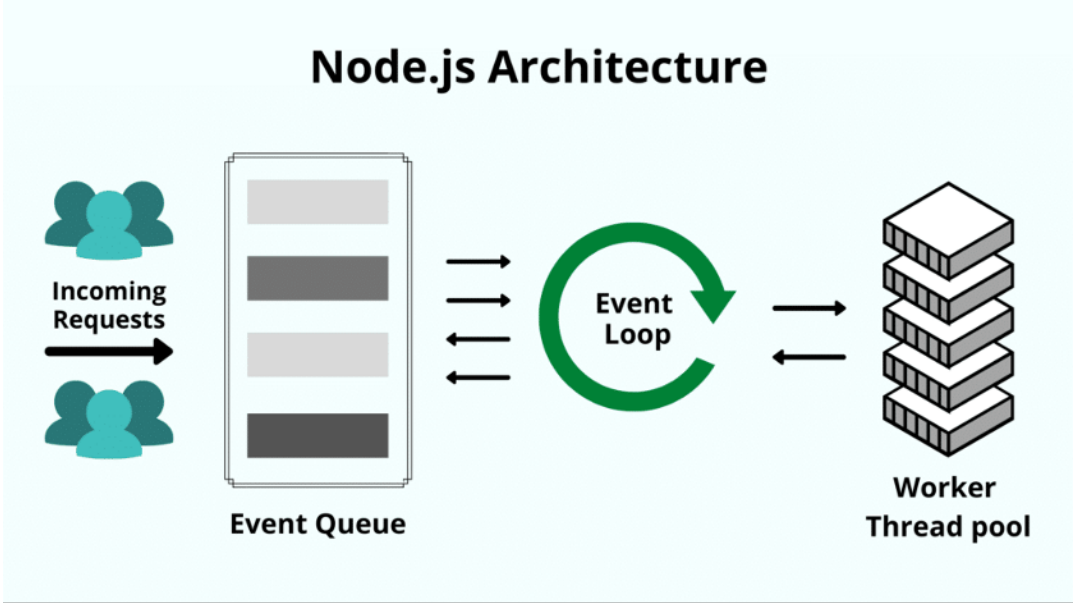


Рисунок 3.4 - Як працює Node.js

3.4 Redux.js

Redux — одна з найпопулярніших бібліотек у front-end розробці сьогодні. Однак багато людей спантеличені тим, що це таке і які його переваги. Як зазначено в документації, Redux — це передбачуваний контейнер стану для програм JavaScript. Перефразовуючи це, це архітектура потоку даних програми, а не традиційна бібліотека чи фреймворк, як Underscore.js та AngularJS.

Flux — це архітектура програми, яку Facebook використовує для створення клієнтських вебдодатків. Він доповнює компоновані компоненти перегляду React, використовуючи односпрямований потік даних. Це скоріше шаблон, ніж формальна структура, і ви можете почати використовувати Flux відразу без великої кількості нового коду. [4]

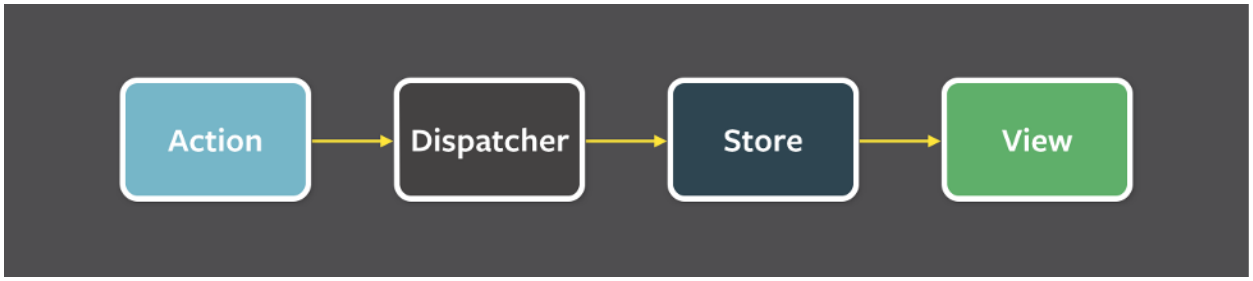


Рисунок 3.5 - Flux

Щоб дати деяку перспективу, візьмемо класичний шаблон MVC. В архітектурі MVC є чітке розділення між даними (моделлю), презентацією (подання) і логікою (контролер). У цьому є одна проблема, особливо у великомасштабних програмах: потік даних є двонапрямний. Це означає, що одна зміна (введення користувача або відповідь API) може вплинути на стан програми в багатьох місцях коду — наприклад, двостороннє прив'язування даних. Це може бути важко підтримувати та налагоджувати.

Flux дуже схожий на Redux. Основна відмінність полягає в тому, що Flux має кілька сховищ, які змінюють стан програми, і він транслює ці зміни як події. Компоненти можуть підписатися на ці події для синхронізації з поточним станом. Redux не має диспетчера, який у Flux використовується для передачі корисних даних зареєстрованим зворотним викликам. Інша відмінність у Flux полягає в тому, що доступно багато різновидів, і це створює деяку плутанину та непослідовність.

3.5 Typescript

TypeScript — це над набір JavaScript, що означає, що він містить усі функції JavaScript, а потім деякі. Таким чином, будь-яка програма, написана на допустимому JavaScript, також працюватиме, як очікується, у TypeScript. Фактично, TypeScript компілюється просто у звичайний JavaScript. Отже, яка різниця? TypeScript пропонує нам більше контролю над нашим кодом за допомогою анотацій типів, інтерфейсів і класів. Я поясню кожен із них пізніше.

TypeScript був створений Microsoft і був випущений у 2012 році після двох років розробки. Він був створений, щоб дозволити додаткову статичну перевірку типів, що особливо корисно при розробці великомасштабних програм. [10]

3.6 Express.js

Express — це структура вебдодатків node js, яка надає широкі можливості для створення веб та мобільних додатків. Він використовується для створення односторінковох, багатосторінкових та гібридних вебдодатків.

Це шар, побудований на вершині Node js, який допомагає керувати серверами та маршрутами.

Переваги Express.js:

- Express створено для створення API та вебдодатків з легкістю,
- Це заощаджує багато часу на кодування майже вдвічі і все одно робить веб і мобільні додатки ефективні.
- Іншою причиною використання express є те, що він написаний на javascript, оскільки javascript є легкою мовою, навіть якщо у вас немає попереднього знання будь-якої мови. Express дозволяє безлічі нових розробників увійти у сферу веб розробки.

Особливості Express.js:

- Швидка розробка на стороні сервера - функції node js допомагають заощадити багато часу.
- Проміжне програмне забезпечення - це обробник запитів, який має доступ до циклу запиту-відповіді програми.
- Маршрутизація - це стосується того, як URL-адреси кінцевої точки програми відповідають на запити клієнта.

3.7 Chakra UI

Chakra UI — це проста, модульна та доступна бібліотека компонентів, яка надає вам будівельні блоки, необхідні для створення ваших програм React.

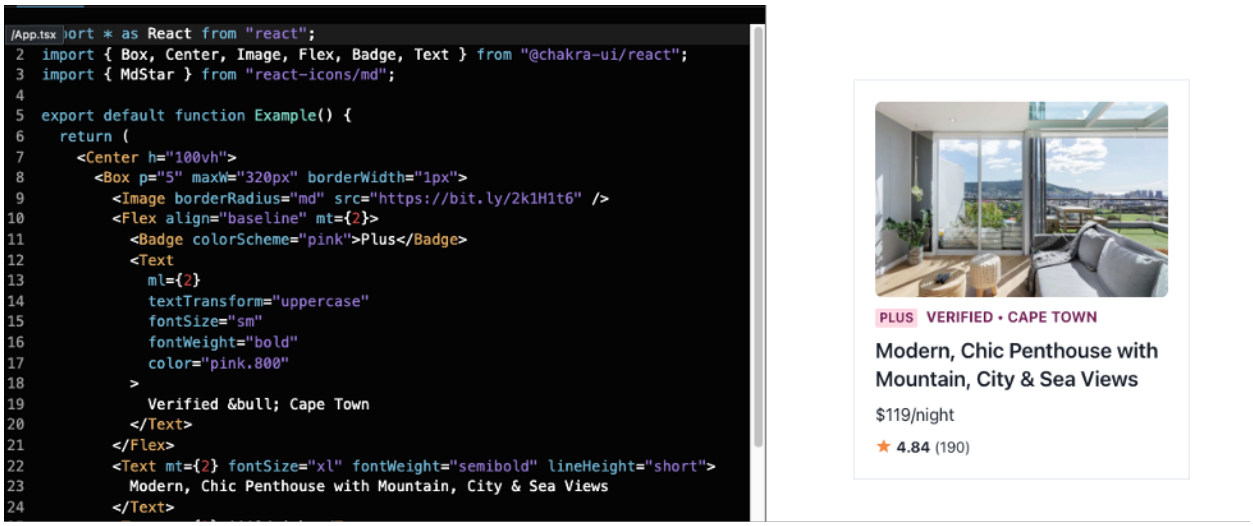


Рисунок 3.6 - Приклад використання Chakra UI

Кастомізація та можливості:

- Стильовий реквізит - інтерфейс Chakra підтримує Reactjs, і кожен компонент можна налаштувати за допомогою параметрів Style. Вони показуються майже з усіма доступними властивостями CSS. Наприклад, для margin-top у CSS ви б написали це як<Text mt={8} >. Це встановить верхнє поле вибраного елемента в 8 пікселів. Інтерфейс Chakra створено на основі палітри кольорів TailwindCSS, тож ви можете знайти всі свої улюблені кольори!
- Ви можете замінити тему інтерфейсу Chakra за замовчуванням і створити власну тему з кольорами на ваш вибір. Ви можете зробити це за допомогою змінних CSS Chakra UI.

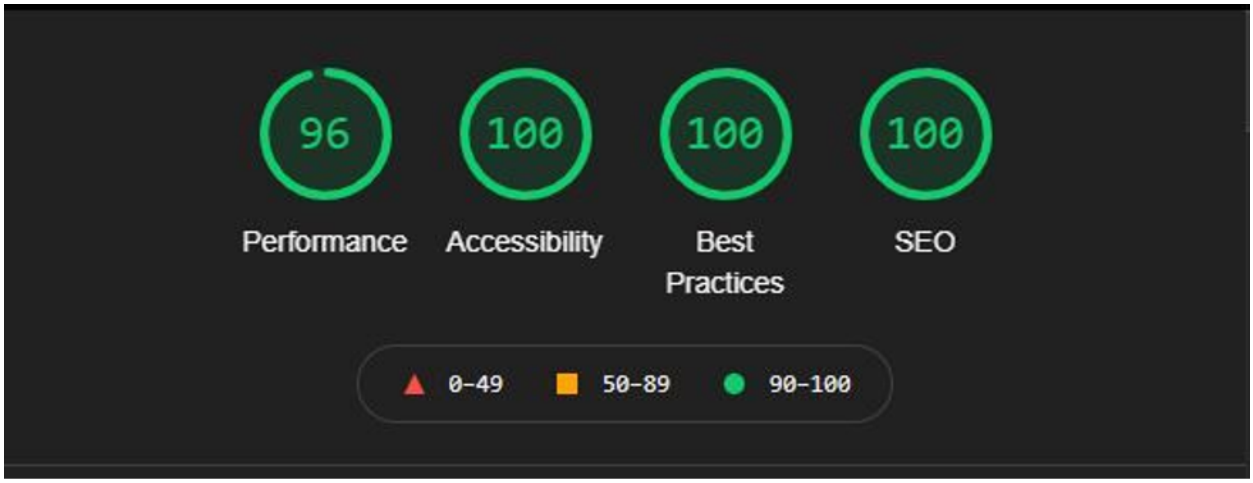


Рисунок 3.7 - Результати PageSpeedInside при використанні даної бібліотеки.

3.8 Webpack

Webpack - це збирач статичних модулів для програм JavaScript - він бере весь код програми та робить його придатним для використання у браузері.

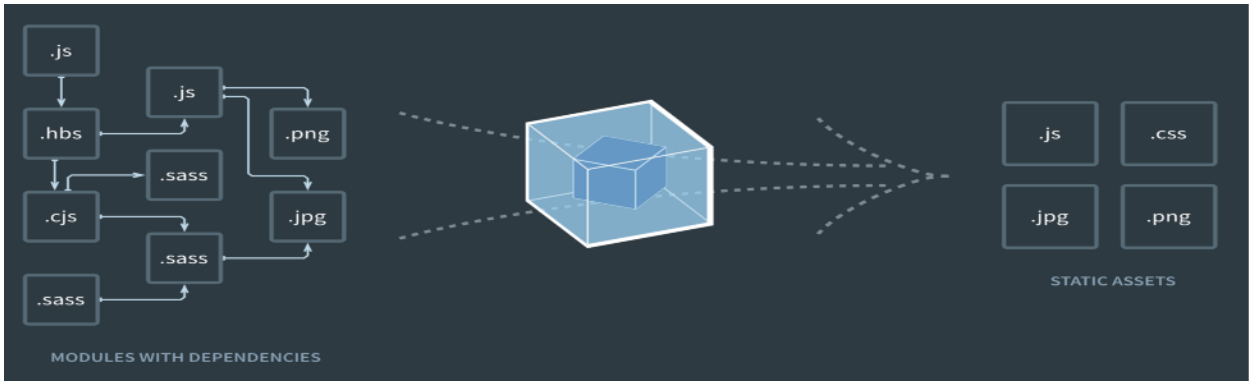


Рисунок 3.8 - Вебпак

Коли Webpack обробляє вашу програму, він будує графік залежностей, який показує модулі, які потрібні вашому проєкту, і генерує один або кілька пакетів. Пакет — це окреме угруповання підключеного коду, який був скомпільований і перетворений для браузера.

Якщо один файл залежить від іншого (він використовує код з окремого файлу), Webpack розглядає це як залежність. Webpack також бере ваші не кодові

активи (зображення, шрифти, стилі тощо) і перетворює їх у залежності для вашої програми.

Webpack можна розбити на ці 5 принципів:

- Вхід.
- Вихід.
- Навантажувачі.
- Плагіни.
- Режим.

Webpack визначає, які інші модулі залежать від точки входу, як прямо, так і опосередковано. Точкою входу за замовчуванням в сучасних програмах JavaScript є ./src/index.js, але її можна встановити на будь-який файл на ваш вибір. Також можна мати кілька точок входу.

```
// Multiple Entry points for Multiple Dependency Graphs
module.exports = {
    entry: {
        pageOne: './src/pageOne/index.js',
        pageTwo: './src/pageTwo/index.js',
        pageThree: './src/pageThree/index.js'
    }
}
```

Рисунок 3.9 - Приклад запису

Точка виведення — це місце, де файли мають бути записані на диск із назвою файлів. Основний вихідний файл записується як ./dist/main.js, а будь-які інші файли додаються до каталогу dist. Вихід встановлюється в тому ж місці, що й точка входу. Вихідні дані також можуть використовувати хеш або імена фрагментів для вмісту, що дозволяє динамічно оновлюватися при зміні коду. Це гарантує, що ви зможете обслуговувати правильний код.

За замовчуванням Webpack знає лише, як обробляти файли .js або .json, що може бути обмеженим. За допомогою завантажувачів Webpack може розширити

функціональність для обробки інших типів файлів, перетворивши їх у модулі для вашої програми.

3.9 Axios

Axios — це HTTP-клієнт на основі обіцянок, який працює як у браузері, так і в середовищі Node.js. Він надає єдиний API для роботи з XMLHttpRequests і http-інтерфейсом вузла. Майже будь-який динамічний проєкт, який ви створюєте, потребує взаємодії з API REST у певний момент, і використання Axios — це простий спосіб зробити це.

```
axios
.get("https://jsonplaceholder.typicode.com/posts")
.then(data => console.log(data.data))
.catch(error => console.log(error));
};
```

Рисунок 3.10 - Приклад використання Axios

3.10 Git

Git є найбільш часто використовуваною системою контролю версій. Git відстежує зміни, які ви вносите у файли, тож у вас є запис про те, що було зроблено, і ви можете повернутися до певних версій, якщо вам знадобиться. Git також спрощує співпрацю, дозволяючи всі зміни, внесені кількома людьми, об’єднувати в одне джерело. Тож незалежно від того, чи пишете ви код, який Бачите лише ви, чи працюєте в команді, Git буде корисним для вас.

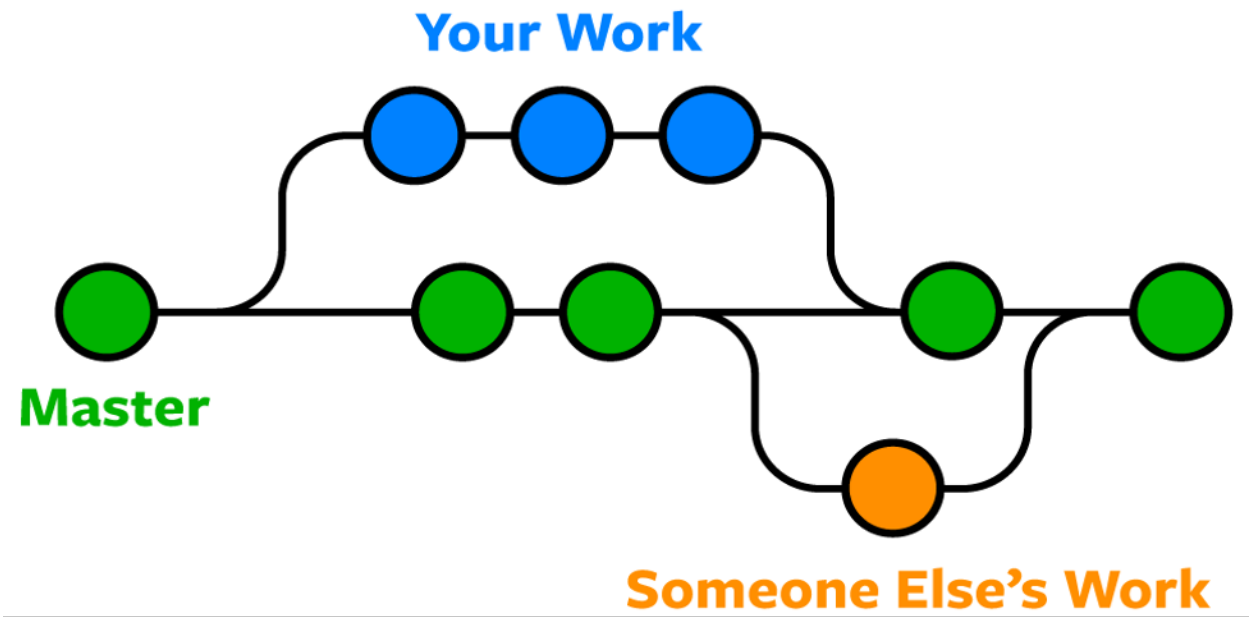


Рисунок 3.11 - Git

Git — це програмне забезпечення, яке працює локально. Ваші файли та їхня історія зберігаються на вашому комп'ютері. Ви також можете використовувати онлайн-хости (наприклад, GitHub або Bitbucket) для зберігання копій файлів та їх історії переглядів.

Репозиторій Git (або скорочено репозиторій) містить усі файли проєкту та всю історію редакцій. Ви візьмете звичайну теку файлів (наприклад, кореневу теку вебсайту) і скажете Git зробити її сховищем. Це створює вкладену теку .git, яка містить усі метадані Git для відстеження змін.

В операційних системах на базі Unix, таких як macOS, файли та теки, які починаються з крапки (.), приховані, тому ви не побачите теку .git у Finder macOS, якщо не покажете приховані файли, але вона є! Ви можете побачити це в деяких редакторах коду.

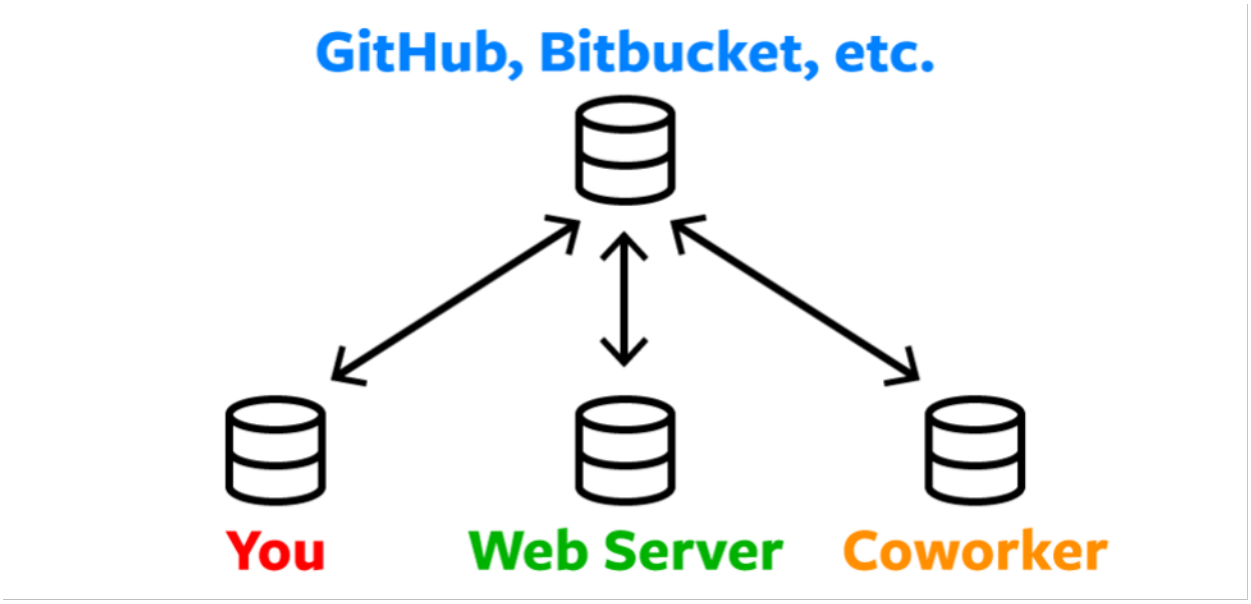


Рисунок 3.12 - Github, etc

Pull-запити – це спосіб обговорення змін, перш ніж об’єднати їх у свою кодову базу. Скажімо, ви керуєте проєктом. Розробник вносить зміни в нову гілку і хоче об’єднати цю гілку з головною. Вони можуть створити запит на витяг, щоб сповістити вас про перегляд їх коду. Ви можете обговорити зміни та вирішити, чи Хочете ви об’єднати їх чи ні.

РОЗДІЛ 4
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ

Під час розробки додатка, сервер клієнтської та серверної частини розвертається локально на наступних url:

- http://localhost:3000 - React;
- http://localhost:5000 - Node.js.

4.1 Реалізація клієнтської частини

Як було описано в попередніх розділах, клієнтська частина була реалізована за допомогою бібліотеки React, з використанням Typescript.

4.1.1 Розгортання проєкту, налаштування Webpack

Create React App — це комфортний шлях щоб почати будувати нові односторінкові додатки за допомогою React.

Він встановлює осередок для розробки таким чином, щоб ми могли використовувати найновіші можливості JavaScript, робить розробку комфортнішою, а також оптимізує ваш додаток для продакшну. Для цього нам знадобиться Node версії >= 8.10 та npm версії >= 5.6.

```
npx create-react-app my-app
cd my-app
npm start
```

Рисунок 4.1 - Команда для створення чистого React-додатку, без конфігурації Webpack.

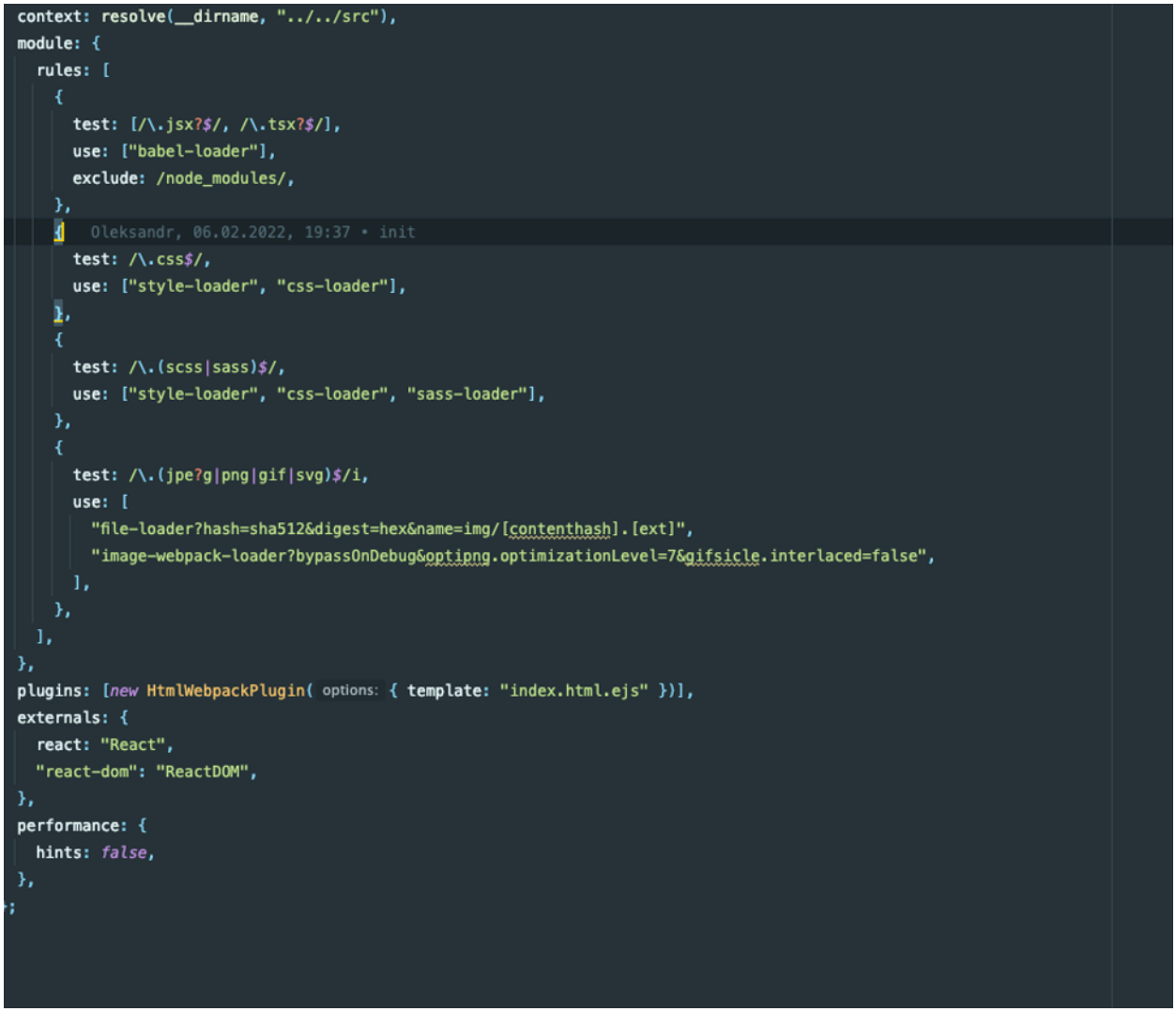
Create React App не керує серверною логіку чи логікою баз даних, а лише надає команди для побудови клієнтської частини, тому ви можете використовувати його з будь-яким сервером. Під капотом він використовує Babel та webpack.

Після попередньо описаних дій необхідно встановити node-модулі для роботи з вебпаком.

```
module.exports = {
  resolve: {
    extensions: [".js", ".jsx", ".ts", ".tsx"],
    alias: {
      "@shared": resolve(__dirname, "../../src/shared-components"),
      "@page": resolve(__dirname, "../../src/page-components"),
      "@assets": resolve(__dirname, "../../src/assets"),
      "@constants": resolve(__dirname, "../../src/constants"),
      "@utils": resolve(__dirname, "../../src/utils"),
      "@interfaces": resolve(__dirname, "../../src/interfaces"),
    },
  },
```

Рисунок 4.2 - Вебпак common конфігурація. Path alias, extensions

На скриншоті вище, ми можемо побачити масив extensions який описує допустимі розширення файлів, та об’єкт alias, який описує шляхи до основних тек, які будуть використовуваться для імпорту компонентів, констант, утиліт, інтерфейсів і тд.



```
context: resolve(__dirname, "../../src"),
module: {
  rules: [
    {
      test: [/\.jsx?$/, /\.tsx?$/],
      use: ["babel-loader"],
      exclude: /node_modules/,
    },
    {   Oleksandr, 06.02.2022, 19:37 • init
      test: /\.css$/,
      use: ["style-loader", "css-loader"],
    },
    {
      test: /\.(scss|sass)$/,
      use: ["style-loader", "css-loader", "sass-loader"],
    },
    {
      test: /\.(jpe?g|png|gif|svg)$/i,
      use: [
        "file-loader?hash=sha512&digest=hex&name=img/[contenthash].[ext]",
        "image-webpack-loader?bypassOnDebug&optipng.optimizationLevel=7&gifsicle.interlaced=false",
      ],
    },
  ],
},
plugins: [new HtmlWebpackPlugin( options: { template: "index.html.ejs" })],
externals: {
  react: "React",
  "react-dom": "ReactDOM",
},
performance: {
  hints: false,
},
;
```

Рисунок 4.3 - Webpack common конфігурація. Style-loader, sass-loader, file-loader конфігурація

Далі необхідно налаштувати так званий development config, який буде запускати localhost.

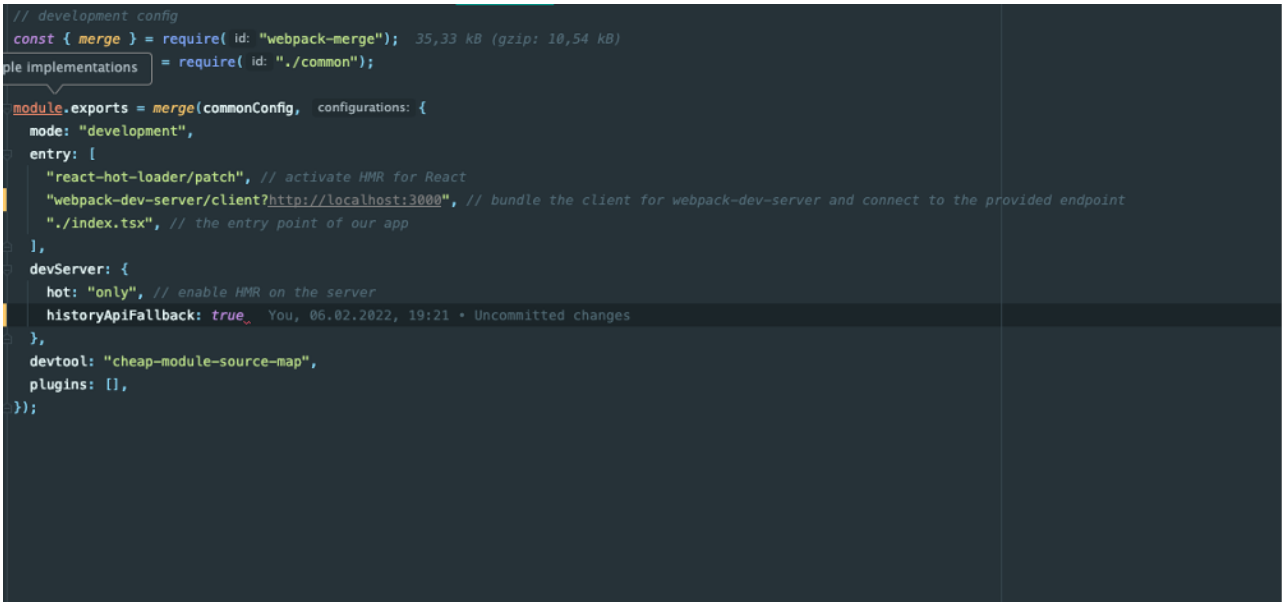


Рисунок 4.4 - dev.js файл, налаштування dev config

Тут ми налаштували порт localhost, вказали index.tsx файл, як корінь нашого додатка, та під'єднали react-hot-loader/patch бібліотеку, яка буде перезавантажувати сторінку автоматично, після змін в IDE Webstorm. Також створюється production config, який є схожим на той, що був описаний вище.

4.1.2 Налаштування package.json, додавання необхідних залежностей, модулів, та команд.

Файл package.json - це файл JSON, який існує в корені проекту Javascript/Node. Він містить метадані, що мають відношення до проекту, і використовується для керування залежностями проекту, сценаріями, версіями та багатьма іншими.

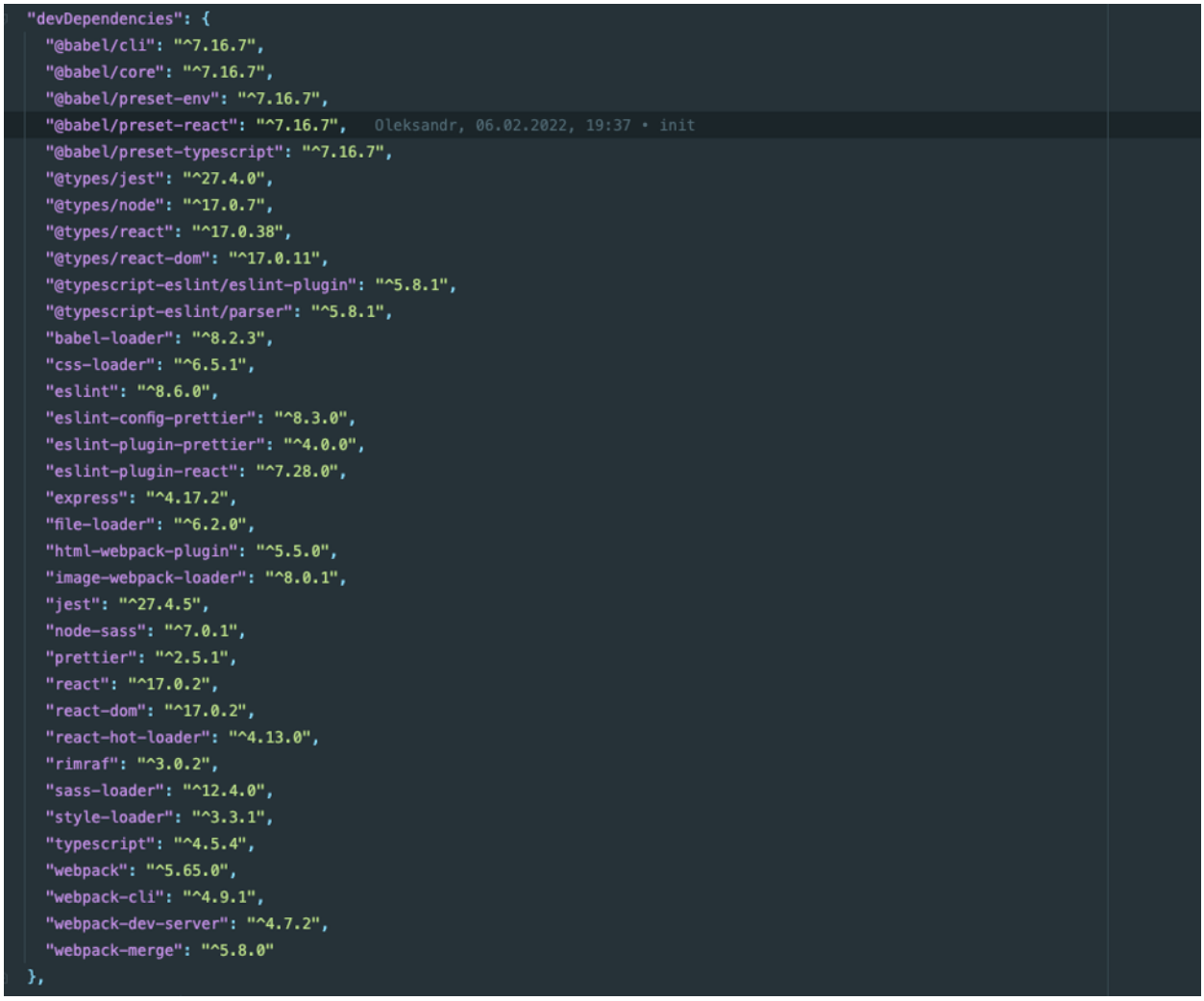
В файл package.json необхідно додати скрипти, тобто команди, які будуть використовуватися для запуску localhost, тестів та ін.

```
"scripts": {
  "build": "yarn run clean-dist && webpack --config=configs/webpack/prod.js",
  "clean-dist": "rimraf dist/*",
  "lint": "eslint './src/**/*.{js,ts,tsx}' --quiet",
  "start": "yarn run start-dev",
  "start-dev": "webpack serve --config=configs/webpack/dev.js",
  "start-prod": "yarn run build && node express.js",
  "test": "jest --coverage --watchAll --config=configs/jest.json"
},
```

Рисунок 4.5 - Скрипти які будуть використовуватися в нашому додатку

Опишімо основні скрипти, які необхідні на початку:

- build - створює мінімізовану, та готову до використання production версію додатка.
- lint - використовується для форматування коду за допомогою бібліотеки eslint - про яку поговоримо трішки пізніше.
- start - запускає localhost.
- test - запускає unit-тести, в нашому випадку використовує бібліотеку jest.



```
"devDependencies": {
  "@babel/cli": "^7.16.7",
  "@babel/core": "^7.16.7",
  "@babel/preset-env": "^7.16.7",
  "@babel/preset-react": "^7.16.7",   Oleksandr, 06.02.2022, 19:37 • init
  "@babel/preset-typescript": "^7.16.7",
  "@types/jest": "^27.4.0",
  "@types/node": "^17.0.7",
  "@types/react": "^17.0.38",
  "@types/react-dom": "^17.0.11",
  "@typescript-eslint/eslint-plugin": "^5.8.1",
  "@typescript-eslint/parser": "^5.8.1",
  "babel-loader": "^8.2.3",
  "css-loader": "^6.5.1",
  "eslint": "^8.6.0",
  "eslint-config-prettier": "^8.3.0",
  "eslint-plugin-prettier": "^4.0.0",
  "eslint-plugin-react": "^7.28.0",
  "express": "^4.17.2",
  "file-loader": "^6.2.0",
  "html-webpack-plugin": "^5.5.0",
  "image-webpack-loader": "^8.0.1",
  "jest": "^27.4.5",
  "node-sass": "^7.0.1",
  "prettier": "^2.5.1",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-hot-loader": "^4.13.0",
  "rimraf": "^3.0.2",
  "sass-loader": "^12.4.0",
  "style-loader": "^3.3.1",
  "typescript": "^4.5.4",
  "webpack": "^5.65.0",
  "webpack-cli": "^4.9.1",
  "webpack-dev-server": "^4.7.2",
  "webpack-merge": "^5.8.0"
},
```

Рисунок 4.6 - Development залежності

Dev-залежності - це пакети, які використовуються у файлах або запускаються як двійкові файли на етапі розробки. Це пакети, які необхідні лише під час розробки і не потрібні для виробничої збірки.

Тепер нам необхідно встановити основні залежності, які використовуються на всіх етапах.

```
"dependencies": {
  "@chakra-ui/icons": "^1.1.5",
  "@chakra-ui/react": "^1.8.3",
  "@emotion/react": "^11",
  "@emotion/styled": "^11",
  "axios": "^0.27.2",
  "date-fns": "^2.28.0",
  "framer-motion": "^5",
  "lodash": "^4.17.21",
  "react-intl": "^5.25.1",
  "redux-toolkit": "^1.1.2"
}
```

Рисунок 4.7 - Dependencies

- @chakra-ui - UI бібліотека, яка була використана для оформлення User Interface, тобто візуальної частини.
- axios - бібліотека яка заміняє нам нативний fetch, для запитів до сервера. Має більш зручний синтаксис, та при отриманні даних одразу форматування json формату в об’єкт.
- date-fns - бібліотека для керування датами, тобто її форматування. Має різні корисні допоміжні функції, наприклад isToday - для перевірки чи є дата яку ми передали сьогоднішним днем.
- lodash - бібліотека з допоміжними функціями, для стандартної логіки, яку немає сенсу дублювати.
- react-intl - бібліотека для інтернаціоналізації, тобто додає можливість використовувати абсолютно нескінченну кількість мов.
- redux-toolkit - наш state manager(store) про який ми вже говорили. Потрібен для зберігання даних, їх керування та виконання асинхронних запитів до сервера.

4.1.3 Prettier, eslint, typescript налаштування

ESLint виконує автоматичне сканування ваших файлів JavaScript на предмет поширених синтаксичних і стильових помилок. Prettier сканує ваші файли на предмет проблем зі стилем і автоматично переформатує ваш код, щоб забезпечити дотримання послідовних правил щодо відступів, інтервалів, крапки з комою, одинарних і подвійних лапок тощо.

Рисунок 4.8 - Prettier конфіг

Рисунок 4.9 - Eslint конфіг

Ці налаштування необхідні для знаходження синтаксичних на інших помилок, а також форматування коду - це дуже важливо в сучасній розробці.

Рисунок 4.10 - Typescript конфіг

За допомогою файлу tsconfig.json можна налаштувати проєкт під TypeScript. Зокрема, цей файл виконує такі завдання:

- встановлює кореневий каталог проєкту TypeScript
- виконує налаштування параметрів компіляції
- встановлює файли проєкту

4.1.4 Загальна структура нашого додатка.

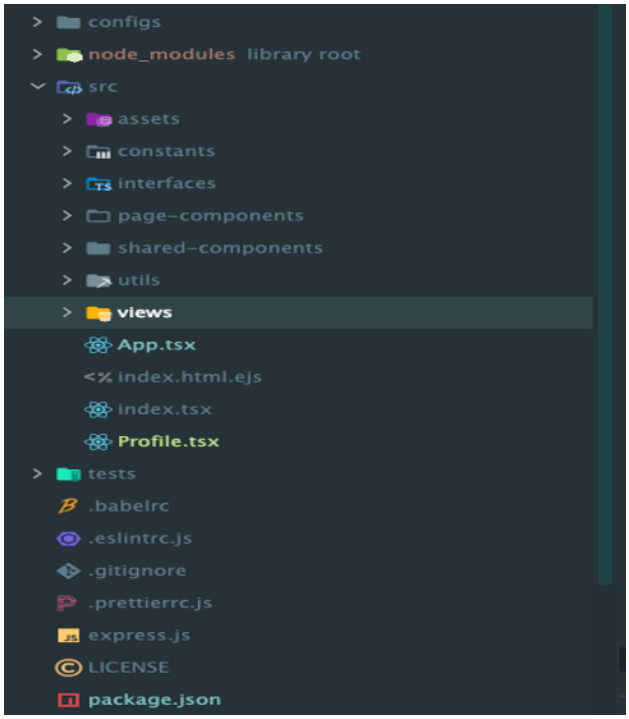


Рисунок 4.11 - Структура додатку

Коротко про теки:

- Assets - іконки, картинки та ін.
- Constants - константи проєкту, тобто незмінні значення.
- Interfaces - Інтерфейси, тобто описання типів TS.
- Page-components - компоненти наших сторінок.
- Shared-components - загальні компоненти проєкту, які можуть бути використані в будь-якому місці.
- Utils – допоміжні функції, тобто функції які немає сенсу тримати в компонентах, бо вони можуть бути використані в різних місцях нашого додатка.
- Views - наші сторінки.
- Node_modules - пакети необхідні для розробки додатка.

4.1.5 Приклад реалізації сторінки авторизації

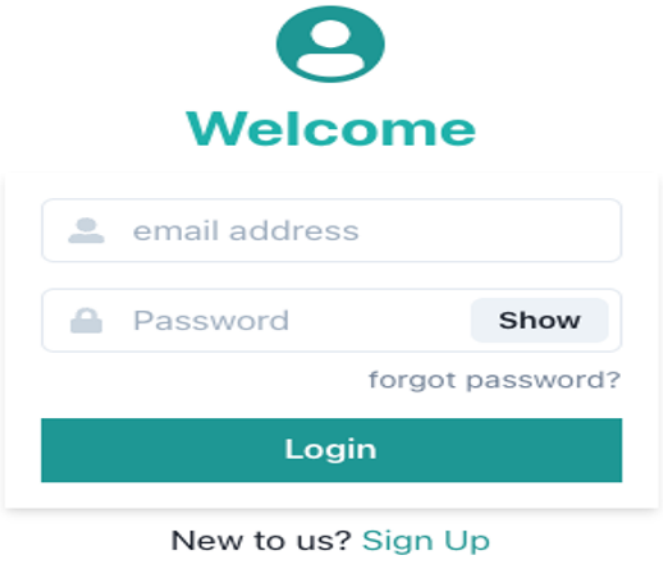


Рисунок 4.12 - Сторінка авторизації

Розглянемо як виглядає наша сторінка авторизації в IDE.

Рисунок 4.13 - Імпорти

На даному скриншоті ми можемо побачити імпортування компонентів та іконок, які використовуються на даній сторінці. Їх використання значно прискорює розробку, бо вони містять в собі певні розв’язання стандартних проблем.

Рисунок 4.14 - Форма авторизації

Вона містить два поля - пошта, пароль, та кнопку для авторизації

Всі поля форм в даному проєкту проходять валідацію, тобто перевірку, перед тим як відправлятись на сервер, для того, щоб уникнути зайвих запитів, та

виникнення можливих помилок. Для валідації керування формами, та їх валідації використовуються наступні бібліотеки - formik, yup.

auth.ts

```ts
import { ACCESS_TOKEN_KEY, REMEMBER_ME_KEY } from 'constants/auth';

export const resetUserAuthStorage = () => {
  localStorage.removeItem(ACCESS_TOKEN_KEY);
  localStorage.removeItem(REMEMBER_ME_KEY);
  sessionStorage.removeItem(ACCESS_TOKEN_KEY);
};

export const checkIsAccessTokenExist = () => {
  const token = (localStorage.getItem(REMEMBER_ME_KEY) ? localStorage : sessionStorage).getItem(ACCESS_TOKEN_KEY);

  return token;
};

export const setUserAccess = (remember_me: string | boolean, access_token: string) => {
  if (remember_me) {
    localStorage.setItem(ACCESS_TOKEN_KEY, access_token);
    localStorage.setItem(REMEMBER_ME_KEY, '1');
  } else {
    sessionStorage.setItem(ACCESS_TOKEN_KEY, access_token);
  }
};

export const get2FacHeaders = (token?: string) => {
  return {
    headers: {
      Authorization: `Bearer ${token || sessionStorage.getItem( key: 'two_auth_access')}`,
    },
  };
};
```

Рисунок 4.15 - Приклад використання утилітних функцій

На даному скриншоті можемо побачити декілька функцій які виконують роль - очищення localStorage/sessionStorage, які використовуються для зберігання JWT токена, необхідного для ідентифікування користувача. Також спостерігаємо за функцією для перевірки на наявність токена в локальних сховищах, надання доступу для користувача, та створення так званих headers, об’єкт який відправляється в запитах на сервер, та в нашому випадку містить в собі Bearer JWT токен.

```
export const LOGIN_FIELDS: AuthField[] = [
  {
    name: 'email',
    placeholder: 'Email',
    type: 'text',
    validation: Yup.string().required('The email field is required').email('Invalid email address'),
  },
  {
    name: 'password',
    placeholder: 'Password',
    type: 'password',
    validation: Yup.string().required('The password field is required'),
  },
];
```

Рисунок 4.16 - Константи з полями форми авторизації

На рисунку вище ми спостерігаємо за константою з полями нашої форми. Це масив об’єктів, кожен з яких має 4 обов’язкових поля. Останнім з яких є валідація.

Після введення інформації в форму, нам необхідно виконати запит, та отримати дані в разі валідності даних. На наступних рисунках розглянемо підключення Redux-toolkit яка і буде виконувати цю роль.

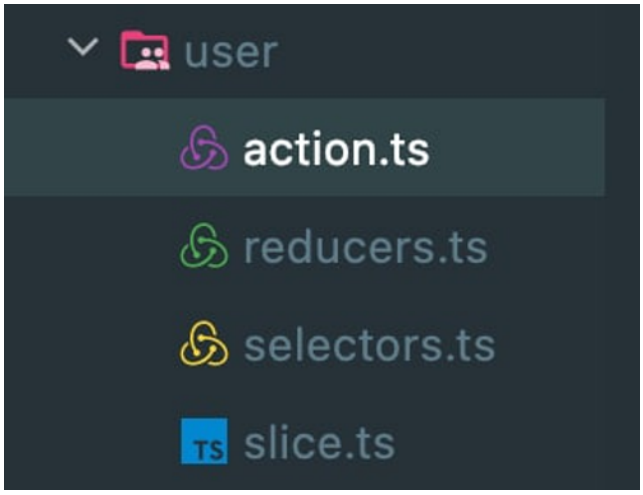


Рисунок 4.17 - Структура modules

- actions.ts - наші actions, які будуть виконувати запити.
- reducers.ts - функції які будуть змінювати наше сховище.
- selectors.ts - функції які виконують функцію вибірки даних із сховище.
- slice.ts - так званий корінь нашого redux-toolkit модуля.

Розглянемо підключення redux-toolkit до проекту.

```
const rootReducer = combineReducers( reducers: {
  userReducer,

});

export const store = configureStore( options: {
  reducer: rootReducer,
  middleware: getDefaultMiddleware => getDefaultMiddleware(),
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
```

Рисунок 4.18 - Підключення redux-toolkit.

- Допоміжна функція combineReducers перетворює об’єкт, значення якого є різними функціям, в одну функцію, яку можна передати до createStore.
- configureStore - зручна абстракція у порівнянні зі стандартною функцією Redux createStore, яка додає параметри за замовчуванням до налаштування сховища для кращого досвіду розробки.
- useAppDispatch - типізована версія хука useDispatch.
- useAppSelector - типізована версія хука useSelector.

Тепер давайте розглянемо більш детально кожну раніше описану частину редаксу на прикладі.

Рисунок 4.19 - Action AsyncLogin

На рисунку вище описаний процес відправлення даних на авторизацію, та отримання відповіді від сервера з подальшим наданням доступу.

Рисунок 4.20 - getUserInfo action

На рисунку ми бачимо запит на отримання інформації про користувача.

Рисунок 4.21 - asyncUpdateUser

На рисунку вище зображено запит на оновлення інформації по користувачу, згідно з полями які описані в типах.

```typescript
export interface User {
  name: string;
  email: string;
  twoFact: boolean;
  confirmed: boolean;
  image?: string;
  description?: string;
  birth_date: string;
}

export interface LoginUser extends Omit<User, "email"> {
  password: string;
}
```

Рисунок 4.22 - Інтерфейс для користувача

Один з основних принципів TypeScript полягає в тому, що перевірка типу фокусується на формі, яку мають значення. Це іноді називають «качиним типом» або «структурним підтипом». У TypeScript інтерфейси виконують роль іменування цих типів і є потужним способом визначення контрактів у вашому коді, а також контрактів із кодом за межами вашого проєкту.

На рисунку вище, ми можемо побачити описання базової інформації про користувача та описання типу для відправки даних на сервер.

```typescript
export function logoutReducer(builder: ActionReducerMapBuilder<UserState>) {
  builder.addCase(userLogout, reducer: () => ({
    ...initialState,
    isLoaded: false,
  }));
}
export function isAuthReducer(builder: ActionReducerMapBuilder<UserState>) {
  builder.addCase(setIsAuth, reducer: (state, action: PayloadAction<boolean>) => {
    state.isAuth = action.payload;
  });
}
```

Рисунок 4.23 Приклад редюсера для виходу з аккаунту та перевірки на те, чи авторизований користувач

Перший із них змінює поле isLoaded - за цим полем ми слідкуємо в нашому компоненті, та умовно показуємо, що йде завантаження.

```
export const initialState: UserState = {
  error: '',
  isAuth: null,
  isLoaded: false,
  isLoading: false,
  isResended: false,
  user: initialUserInfo,
};

export const userSlice = createSlice( options: {
  name: USER_SLICE_NAME,
  initialState,
  reducers: {},
  extraReducers: builder => {
    loginReducer(builder);
    userInfoReducer(builder);
    logoutReducer(builder);
    isAuthReducer(builder);
```

Рисунок 4.24 - Слайс user модуля

Тут можем спостерігати за початковим станом нашого слайса, та редюсерами які ми підключили до нього.

```
const ProfileSettings: FC = () => {
  return (
    <SettingsContainer>
      <SettingsWrapper>
        <SettingsTop>
          <Button>Settings</Button>
        </SettingsTop>
        <SettingsContent>
          {PROFILE_SETTINGS_FIELDS.map(item => (
            <SettingsForm {...item} key={item.title} />
          ))}
        </SettingsContent>
      </SettingsWrapper>
    </SettingsContainer>
  );
};

export default ProfileSettings;
```

Рисунок 4.25 - Сторінка редагування профілю

Вона складається із заголовку, та форми, яка створюється із константи.

Як було зазначено раніше, для керування формою використовується formik + yup.

Formik - це невелика бібліотека, яка допоможе вам з 3 найбільш дратівливими частинами:

- Отримання значень у стані форми та з неї.

- Перевірка та повідомлення про помилки.
- Обробка подачі форми.

Змістивши все перераховане вище в одному місці, Formik забезпечить організованість – полегшить тестування і міркування про ваші форми.

```
const formik = useFormik( {validateOnChange, validateOnBlur, validateOnMount, isInitialValid, enableReinitialize, onSubmit, ...rest}: {
  initialValues: getFormikInitialValues(formikLoginFields),
  onSubmit: values => {
    dispatch(
      asyncLogin({
        ...values,
        device_name: DEVICE_NAME,
      } as LoginPayload),
    );
  },
  validationSchema: getFormikValidationSchema(formikLoginFields),
  validateOnBlur: false,
  validateOnChange: false,
});
const formikProps = getFormikProps(formik);
```

Рисунок 4.26 - Приклад використання бібліотеки formik

На рисунку ми викликаємо хук useFormik, в який передаємо наступні параметри:

- initialValues - початкові значення полів форми.
- onSubmit - функція, яка буде викликана на підтвердження форми.
- validationSchema - yup схема валідації для полів.
- validateOnBlur - приймає булеве значення, яке визначає, чи має виконуватись валідація на так званий «blur» поля.
- validationOnChange - ідентична роль як у параметра вище, за умови, що перевірки виконується на зміну значення поля форми.

4.1.6 Приклади реалізації hooks, shared-components.

useOutsideClick.tsx

```tsx
import { useEffect, RefObject } from 'react';  8,25 kB (gzip: 3,33 kB)

type Event = MouseEvent | TouchEvent;

const useOutsideClick = <T extends HTMLElement = HTMLElement>(
  ref: RefObject<T> | null,
  handler: (event: Event) => void,
): void => {
  useEffect( effect: () => {
    const listener = (event: Event) => {
      const el = ref?.current;
      if (!el || el.contains((event?.target as Node) || null)) {
        return;
      }

      handler(event);
    };

    document.addEventListener( type: 'mousedown', listener);
    document.addEventListener( type: 'touchstart', listener);

    return () => {
      document.removeEventListener( type: 'mousedown', listener);
      document.removeEventListener( type: 'touchstart', listener);
    };
  }, deps: [ref, handler]);
};
export default useOutsideClick;
```

Рисунок 4.27 - Приклад реалізації хука useOutsideClick

Він використовується для того, щоб при виконанні натиску на ліву кнопку миші, поза межі вибраного нами блоку, виконувалась певна функція, яку ми самі прописуємо. Часто використовується з модальними вікнами, боковими меню і т.д.

```
useOutsideClick( ref: isDisableOutsideClick ? null : modalRef, onClose);

return ReactDOM.createPortal(
  <Overlay isOpen={isOpen}>
    {isLoading && <LoadingLine />}
    <ModalWrapper ref={modalRef}>
      <Top>
        <h4>{title}</h4>
        <FontAwesomeIcon icon={faClose as IconDefinition} onClick={onClose} />
      </Top>
      <Content>{children}</Content>
      <Bottom align={bottomAlign}>
        {!isCloseHidden && (
          <Button onClick={onClose} backgroundColor={Colors.GRAY}>
            Close
          </Button>
        )}
        {additionalButtons?.length &&
          additionalButtons.map(({ text, action, disabled }) => (
            <Button key={text} onClick={action} disabled={disabled}>
              {text}
            </Button>
          ))}
        {onSubmit && (
          <Button onClick={handleSubmit} backgroundColor={Colors.BLUE}>
            {submitBtnText || 'Save changes'}
          </Button>
        )}
      </Bottom>
    </ModalWrapper>
  </Overlay>,
  document.getElementById( elementId: 'modal') as HTMLElement,
);
```

Рисунок 4.28 - Реалізація модального вікна

Воно складається з контенту, який ми прокидаємо в середину, та кнопок - cancel, submit які ми можемо приховати за необхідності.

4.1.7 Приклади компонентів додатку.

Розглянемо як виглядають деякі з компонентів нашого додатку.

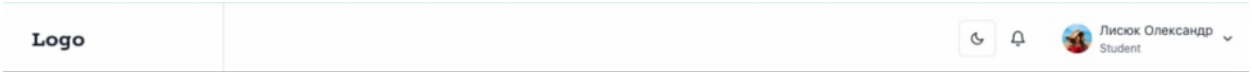


Рисунок 4.29 - Шапка застосунка

Шапка додатку складається з:

- Логотип, при натиску на який виконується перенаправлення на головну сторінку.
- Ім’я користувача

- Кнопка у вигляді місяця, для зміни теми на чорну/білу.
- Кнопка для переміщення на сторінку успішності.
- Фото користувача, яке відкриває dropdown зображений нижче.

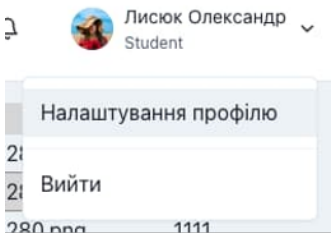


Рисунок 4.30 - Dropdown користувача

Складається з фото профілю, імені користувача, кнопок для переадресації, та виходу з профілю.

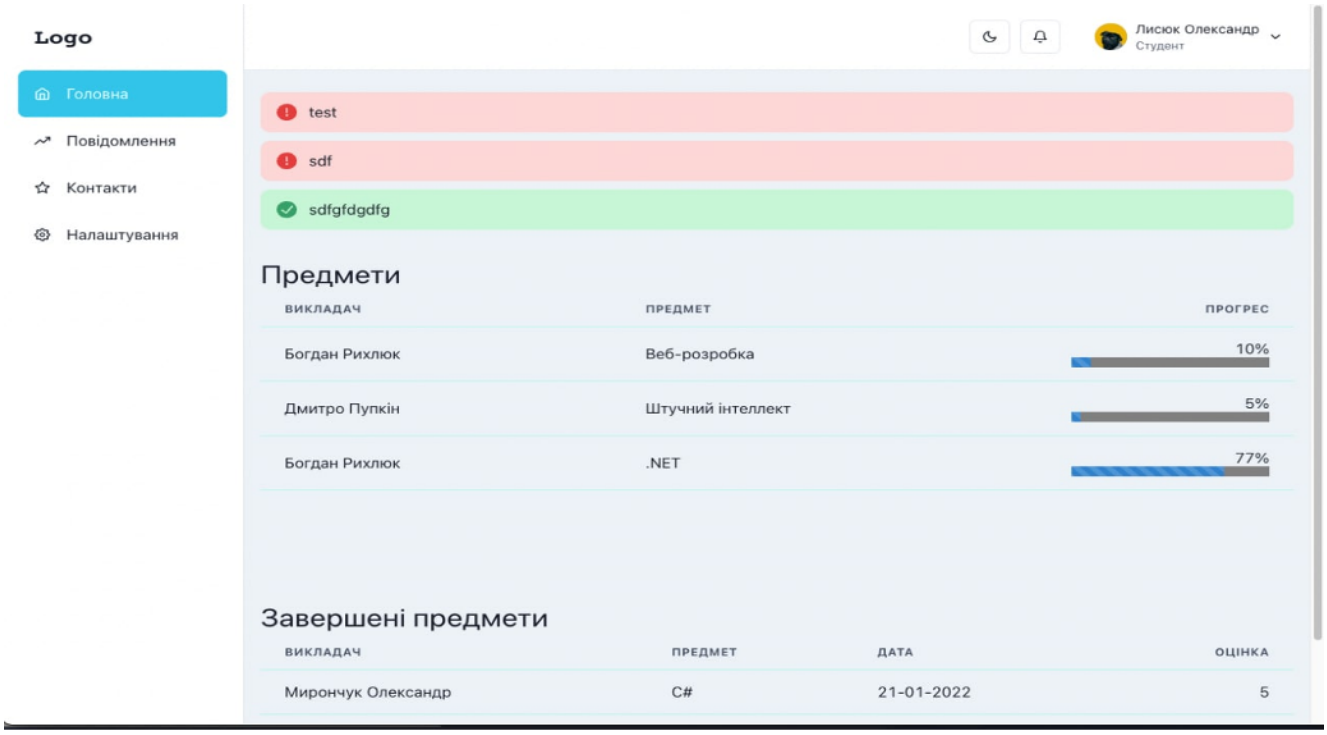


Рисунок 4.31- Головна сторінка, з таблицею яка містить інформацію по предметах зі ввімкненою темною темою

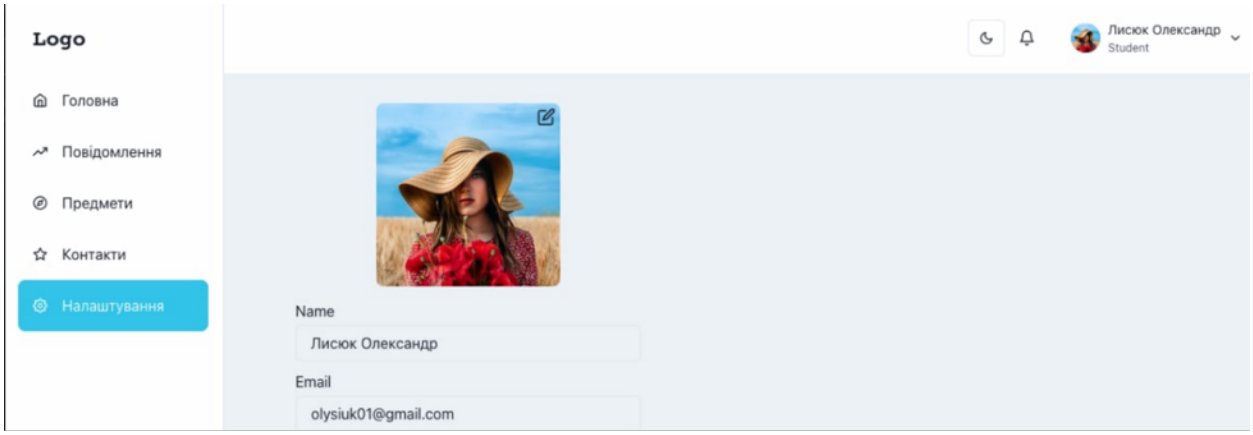


Рисунок 4.32 - Форма зміни даних користувача

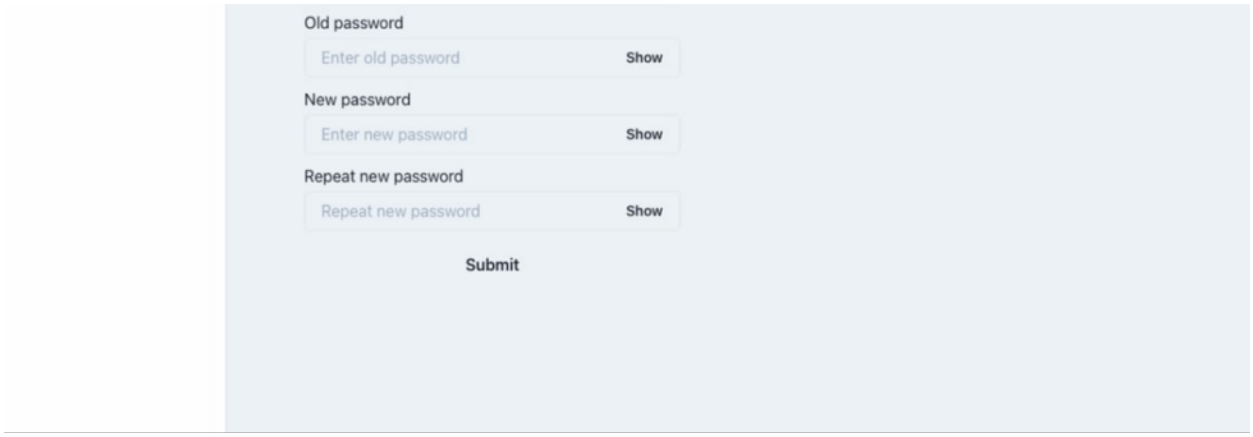


Рисунок 4.33 - Форма зміни пароля

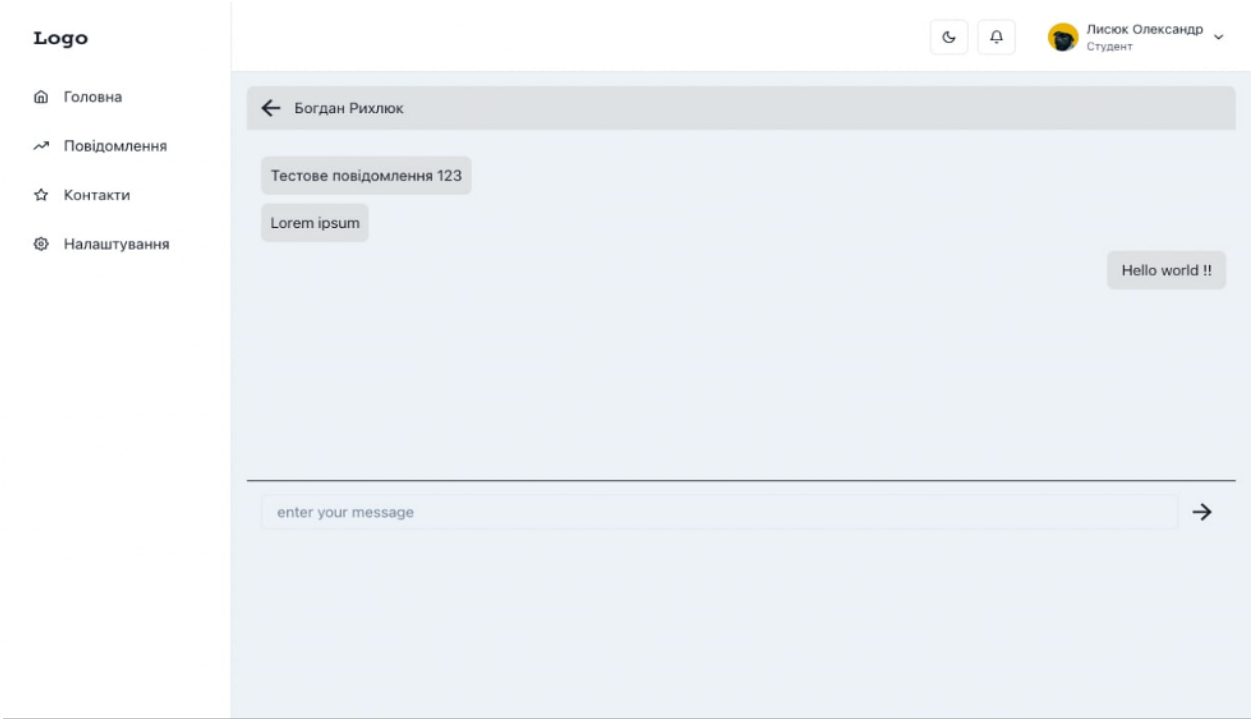


Рисунок 4.34 - Чат з іншим користувачем

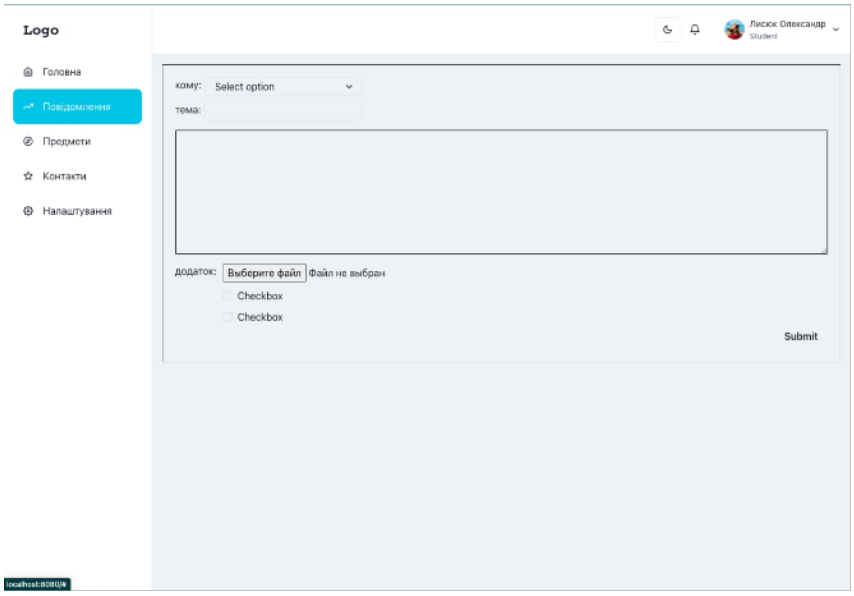


Рисунок 4.35 - Сторінка для завантаження завдань

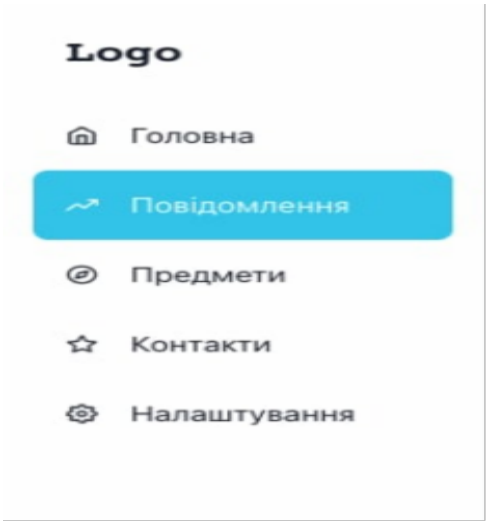


Рисунок 4.36 Бокова панель навігації

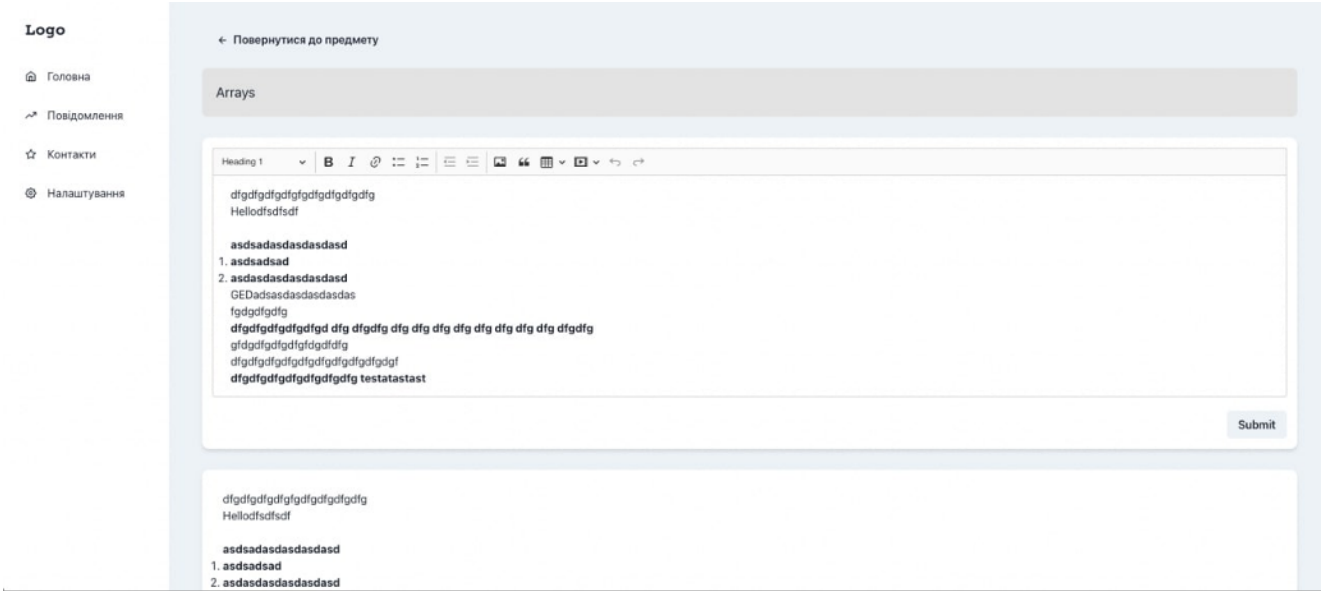


Рисунок 4.37 – Сторінка з лекційним матеріалом та завданнями до теми

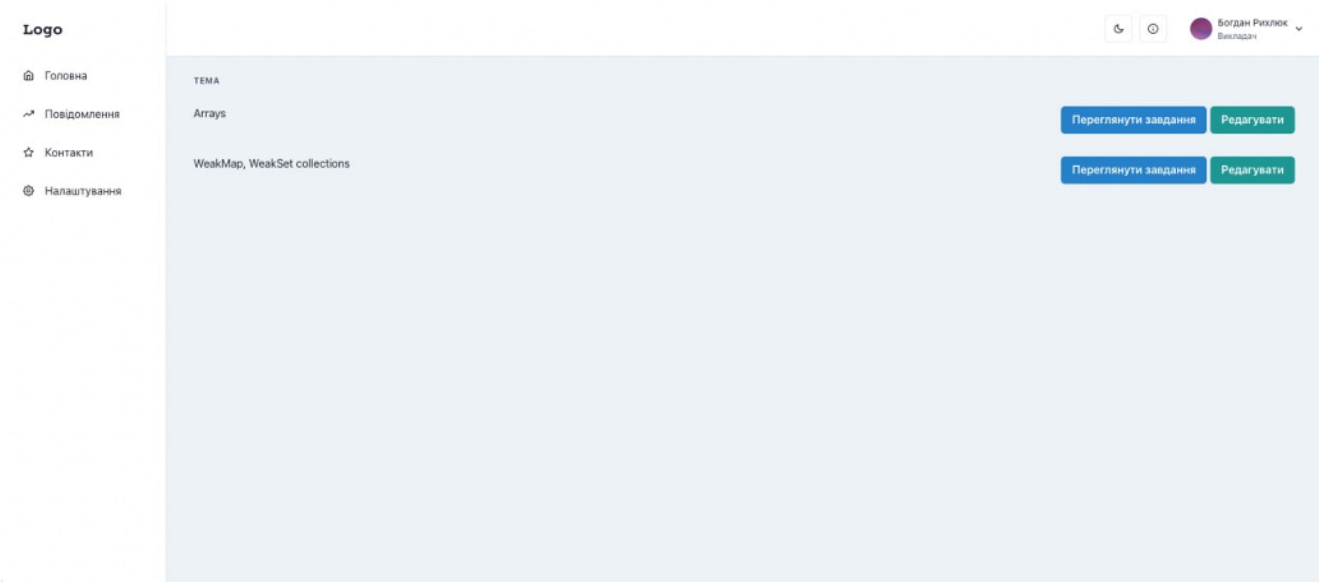


Рисунок. 4.38 – Список тем по предмету

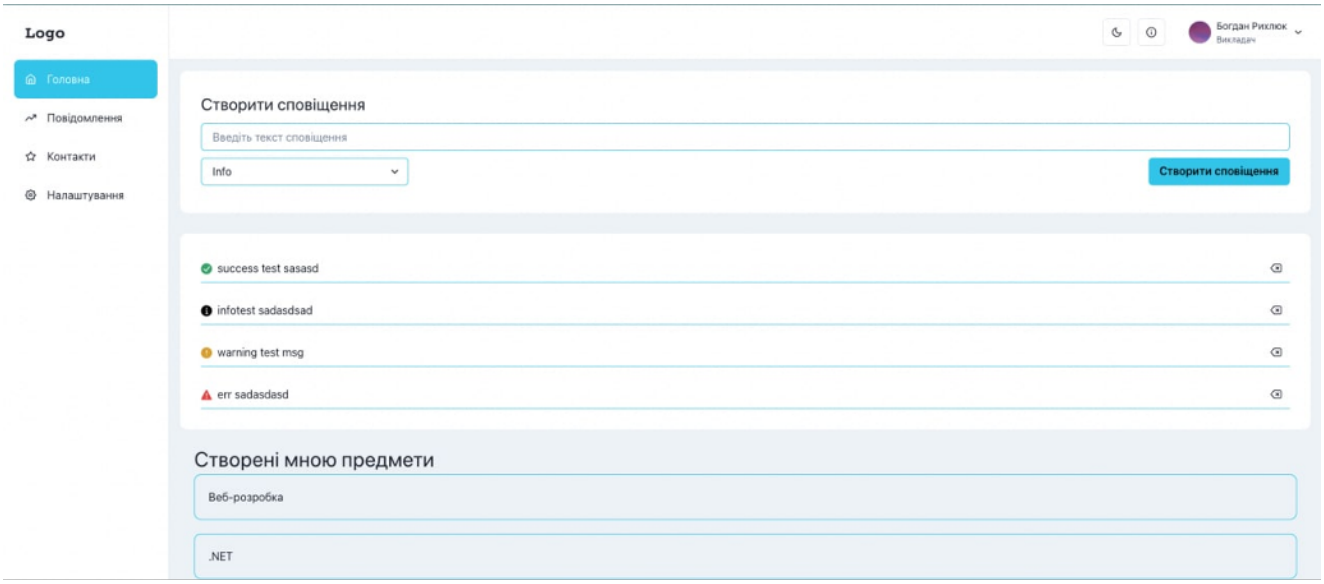


Рисунок 4.39 – Головна сторінка адміністратора

На рисунку вище ми можемо побачити голову сторінку викладача, яка містить:

- Форма створення глобального сповіщення, яке можемj мати чотири статуси.
- Список сповіщень, з можливістю видалення.
- Створені адміністратором предмети.

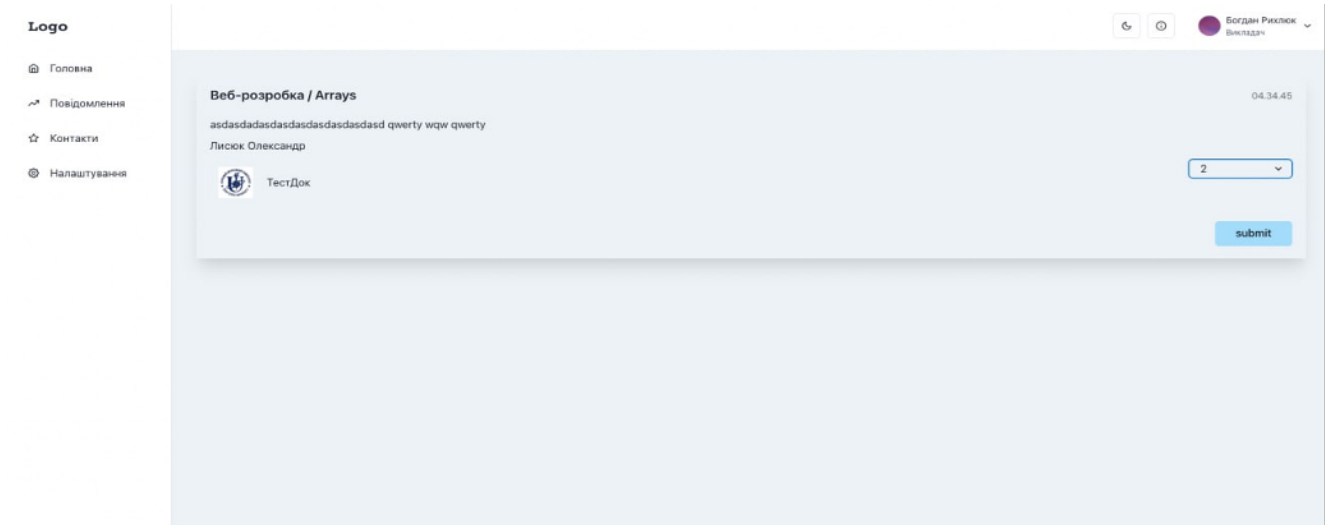


Рисунок 4.40 – Вигляд сторінки перевірки завдань

На попередньому рисунку ми бачимо сторінку, на якій викладач може перевірити завдання, яке було завантажено студентом на перевірку. Є можливість завантажування прикріплених файлів, та виставлення оцінки.

4.2 Реалізація серверної частини

Серверна частина була реалізована за допомогою Node.js, та бібліотеки Express, яка значно прискорює розробку.

Express.js — це фреймворк вебдодатків для Node.js. Він надає різні функції, які роблять розробку вебдодатків швидкою та легкою, що в іншому випадку займає більше часу, використовуючи лише Node.js.

REST API (також відомий як RESTful API) — це інтерфейс програмування прикладних програм (API або WEBAPI), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з вебсервісами RESTful. REST розшифровується як репрезентаційна передача стану і був створений комп’ютерним науковцем Роєм Філдінгом.

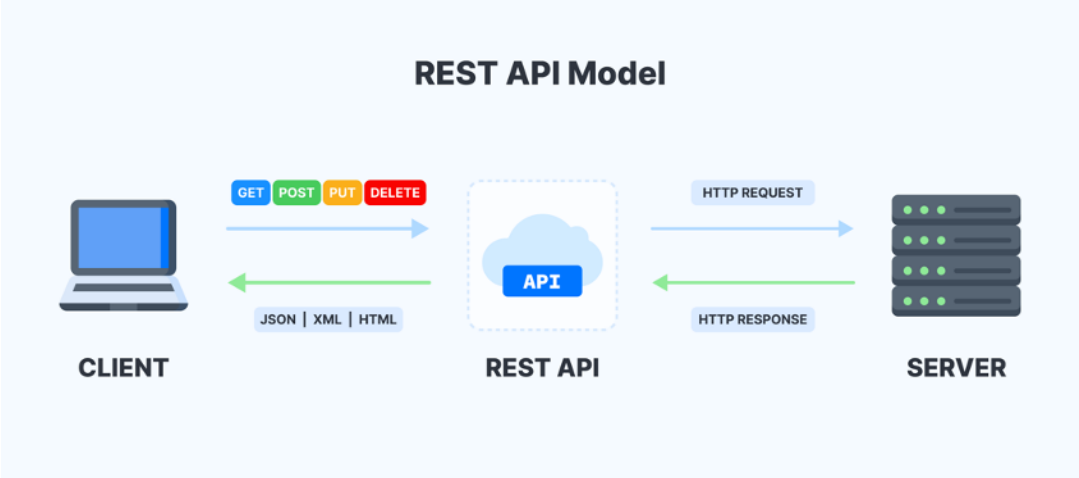


Рисунок 4.41 - Модель роботи REST API

4.2.1 Загальна структура серверної частини.

Рисунок 4.42 - Структура

Короткий опис структури:

- Controllers - в контролерах виконується виклик сервісів, або ж напряму описана логіка кожного з ендпоїнту.
- Models - тут описані наші моделі для збереження в базу даних, тобто юзер, предмет, і так далі.
- Routes - тут виконується логіка, яка пов’язана із формуванням запитів
- Uploads - завантажені нами файли, тобто фото профілю, і тд.
- Server.js - корінний файл нашого сервера.

4.2.2 Реалізація серверної частини.

Розглянемо кожен із вище описаних пунктів більш детально.

```
const app = express();
const PORT = process.env.PORT || 8000;

const allowCrossDomain = function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Methods", "GET,PUT,POST,DELETE");
  res.header("Access-Control-Allow-Headers", "Content-Type");
  next();
};
require("dotenv").config();
app.use(morgan("dev"));
app.use("/uploads", express.static( root: "uploads"));
app.use(cors());
app.use(allowCrossDomain);
app.use(express.json());

app.use("/auth", authRoutes);
app.use("/user", userRoutes);
app.use("/article", articleRoutes);

mongoose
  .connect(process.env.MONGODB, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useFindAndModify: false,
    useCreateIndex: true,
  })
  .then(() => {
    console.log("MongoDB connected");
  });

app.listen(PORT,  hostname: (err) => {
  if (err) {
    console.log(err);
  }
  console.log("server is running");
```

Рисунок 4.43 - Корінний файл сервера

Тут ми виконуємо запуск нашого сервера, за портом вказаним в змінній PORT, з налаштуванням headers, cors, mongoose, та інших модулів, які сприяють коректній роботі сервера.

```
module.exports.login = async (req, res) => {
  try {
    const candidate = await User.findOne( object: { email: req.body.email });
    if (candidate) {
      if (!candidate.isVerified) {
        res.status(409).json({
          message: "Access is denied. Please, confirm your email.",
        });
      }
      You, Moments ago • Uncommitted changes
      const passwordResult = crypto(req.body.password, candidate.password);
      if (passwordResult) {
        const token = signToken(candidate._id);
        res.status(200).json({
          message: "You are successfully logged in",
          token: `Bearer ${token}`,
          user: {
            ...candidate,
            isAuth: true,
          },
        });
      } else {
        res.status(401).json({
          message: "Passwords does not match. Please, try again.",
        });
      }
```

Рисунок 4.44 - Login controller

Цей контролер реалізує авторизацію користувача в декілька етапів. Спочатку виконується перевірка на наявність користувача в базі даних, якщо користувача було знайдено ми виконуємо перевірку паролю, який хешуємо за допомогою crypto. Якщо паролі сходяться - створюємо JWT токен, який натомість надсилаємо в response нашого запиту, із 200 статус кодом, який говорить про позитивну відповідь серверної частини на запит. В інших випадках ми повертаємо помилку.

```
1  const express = require('express')
2  const authController = require('../controllers/authController')
3
4  const router = express.Router()
5
6  router.get('/verify', authController.verifyPassword)
7  router.post('/login', authController.login)
8  router.post('/register', authController.register)
9  router.post('/forgotPassword', authController.forgotPassword)
10 router.post('/resetPassword', authController.changePassword)
11 module.exports = router
```

Рисунок 4.45 Login controller.

Тут ми можемо побачити опис кожного з кінцевих точок, та присвоєння кожному з них контролера, також є можливість використання там званих middleware, які виконують роль проміжного етапу, наприклад для перевірок на валідність токена та інше.

Описувати всі інші приклади використання будь-якої частини не має сенсу, бо всі дії схожі, і виконують одну і ту ж саму роль, тобто отримання запиту, звернення до БД, виконання певних операцій з даними які були надіслані, або ж одразу повернення певної інформації на query і т.д.

ВИСНОВКИ

У випускному кваліфікаційному проєкті представлено результати практичних досліджень, що полягають у створенні інформаційної системи підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання.

Перед тим як почати розробку нашого застосунку, було проведено аналіз предметної області. Проаналізувавши аналоги веб-додатків було виявлено негативні та позитивні сторони, які потрібно було уникнути на етапі створення нашого застосунка. Було сформовано чіткі вимоги до програмного продукту.

В процесі створення застосунку було застосовано Model-View-Controller патерн, що значно полегшило управління тяжкими структурами шляхом розподілення на контролер, представлення та модель.

Вебдодаток було створено за використання бібліотеки React.js та мови програмування TypeScript. Даний застосунок дозволяє студентам ознайомлюватися з навчальним матеріалом, виконувати завдання та здавати їх на перевірку викладачеві, який в свою чергу має змогу їх перевіряти. Також була реалізована можливість листування між учасниками навчального процесу, перегляду детальної інформації про кожного з них. Викладачам надані права на створення предметів та тем, керування глобальними сповіщеннями та інше.

Після створення додатку було проведення фінальне тестування, яке показало, що система має весь необхідний функціонал для взаємодії між учасниками навчального процесу під час індивідуального навчання та готова до впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація бібліотеки react-redux[Електронний ресурс] – Режим доступу до ресурсу: https://react-redux.js.org/
2. Довідник по мові програмування JavaScript[Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/
3. Документація бібліотеки React[Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/
4. Документація до бібліотеки redux-toolkit[Електронний ресурс] – Режим доступу до ресурсу: https://redux-toolkit.js.org/
5. Документація до бібліотеки yup[Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/yup
6. Документація до бібліотеки formik[Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/formik
7. Документація до бібліотеки date-fns[Електронний ресурс] – Режим доступу до ресурсу: https://date-fns.org/
8. Довідник до e-learning системи навчання[Електронний ресурс] – Режим доступу до ресурсу: https://www.td.org/talent-development-glossary-terms/what-is-e-learning
9. Документація до платформи Node.Js[Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/
10. Документація до мови програмування TypeScript[Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

ДОДАТКИ

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Технічне завдання
на створення програмного продукту

«Інформаційна системи підтримки взаємодії викладача зі студентами з індивідуальним графіком навчання»

2022

1. Призначення й мета створення веб-застосунку.

1.1 Призначення веб-застосунку

Додаток повинен надавати можливість студентам тримати зв’язок із викладачами. Проходити навчальний матеріал, та здавати його на перевірку.

1.2 Мета створення веб-застосунку

Метою даного дослідження є створення вебзастосунку для підтримки взаємодії учасників навчального процесу під час очного онлайн-навчання

1.3 Цільова аудиторія

У цільовій аудиторії інформаційної системи ми можемо відокремити такі групи:

1. Студенти.
2. Викладачі.
3. Співробітники будь-яких компаній, які бажають навчатись.

2. Вимоги до веб-застосунку.

2.1 Вимоги до структури й функціонування веб-застосунку.

Застосунок має бути представлений у вигляді вебсайту, який складається із сторінок: «Налаштування профілю», «Контакти», «Головна сторінка зі списком предметів», «Лекції та завдання», «Сторінка завантаження завдань», «Повідомлення». Цей додаток має надавати можливість студентам проходити навчальний матеріал, здавати його на перевірку, та отримувати за це оцінку. Також

має бути можливість перегляду предметів, листування з іншими користувачами, та інше.

2.2 Структура веб-застосунку

2.2.1 Головна сторінка.

Головна сторінка, як і будь-яка інша складається з шапки сайту, яка в свою чергу складається з логотипа, імені користувача, та його фото, при натиску на яке у нас з’явиться меню, з кнопками – виходу з профілю, та переходу на сторінку редагування профілю. В тілі сторінки, ми можемо побачити список предметів, з оцінками до кожного з них.

2.2.2 Сторінка редагування профілю.

Має включати в себе фото студента, та його основні дані, та поля для їх редагування, наприклад такі як – старий пароль, новий пароль, підтвердження нового паролю, адрес електронної пошти.

2.2.3 Сторінка з лекціями та завданнями.

Ця сторінка має містити себе перелік лекцій та практичних завдань до них, з можливістю переходу на сторінку завдання, інші ресурси за необхідністю.

2.2.4 Сторінка завантаження завдань.

Повинна містити тему завдання, кнопку підтвердження, та наступні поля: текстове, для завантаження файлів.

2.2.5 Сторінка повідомлень.

Містить список користувачів, при натиску на яке – має виконуватись редірект на сторінку для чатування з ним.

2.2.6 Сторінка чатування.

Має містити ім’я користувача з яким ми переписуємось, його статус – онлайн/офлайн, повідомлення обох сторін, поле для вводу повідомлення, а також кнопка для його відправлення.

2.3 Вимоги до видів забезпечення.

2.3.1 Вимоги до інформаційного забезпечення

Реалізація додатку відбувається за допомогою наступних технологій:

- HTML
- CSS
- JavaScript
- ReactJs
- NodeJs
- ExpressJs
- Chakra UI
- Typescript

ДОДАТОК Б

Система навчання, заснована на формалізованому навчанні, але з використанням електронних ресурсів відома як електронне навчання. У той час як навчання може проводитися в класі або поза ним, використання комп'ютерів та Інтернету є основним компонентом електронного навчання. Електронне навчання можна також назвати мережевою передачею навичок і знань, а навчання надається великій кількості одержувачів в один і той же або в різний час. Раніше це не приймалося повністю, оскільки передбачалося, що в цій системі немає людського фактора, необхідного для навчання.

Деталізація мети проєкту методом SMART**.** Для збільшення успішності проєкту необхідно правильно встановити мету, це можна зробити використовуючи метод SMART

Таблиця Б.1 – Деталізація мети проекту SMART методом

Specific (конкретна)	Розробити всі компоненти web-застосунку та протестувати їх роботу
Measurable (вимірювана)	Розробити визначені компоненти веб-застосунку до кінця 4 курсу, використовуючи мінімальний обсяг ресурсів необхідний для виконання даного завдання.
Achievable (досяжна, узгоджена)	Для розробки проекту необхідні знання та досвід у використанні наступних технологій: HTML, CSS, JavaScript, Typescript, Redux, React та його екосистеми
Relevant (реалістична)	Створений web-застосунок має надавати функціонал для підтримки взаємодії учасників навчального процесу під час очного онлайн-навчання

Time-framed (обмежена в часі)	Термін досягнення мети даного проекту визначено - до кінця червня 2022 року.

Планування змісту робіт**.** Результатом проєкту, що розбиває командну роботу на певні керовані частини – це WBS (Work Breakdown Structure). Іншими словами – це опис обсягу роботи, яку буде виконано в рамках проєкту. Тобто, це ієрархія завдань, яка представляє деяке розуміння учасників певної робочої групи проєкту щодо складу проекту, а також вартості, тривалості, розміру кожного окремого компонента або завдання.

Структурно, найвищий рівень розміщує кінцевий результат проєкту. Другий рівень представляє з себе різноманітні заходи та дії, які необхідні для досягнення мети нашого проекту. Виходячи з цього, декомпозиція відбувається ще до моменту, коли дії мають однозначний та чіткий результат і мають одного виконавця для якого є можливість розрахувати часові затрати, та витрати на працю.

WBS розробки веб-застосунку зображено на рисунку Б.1

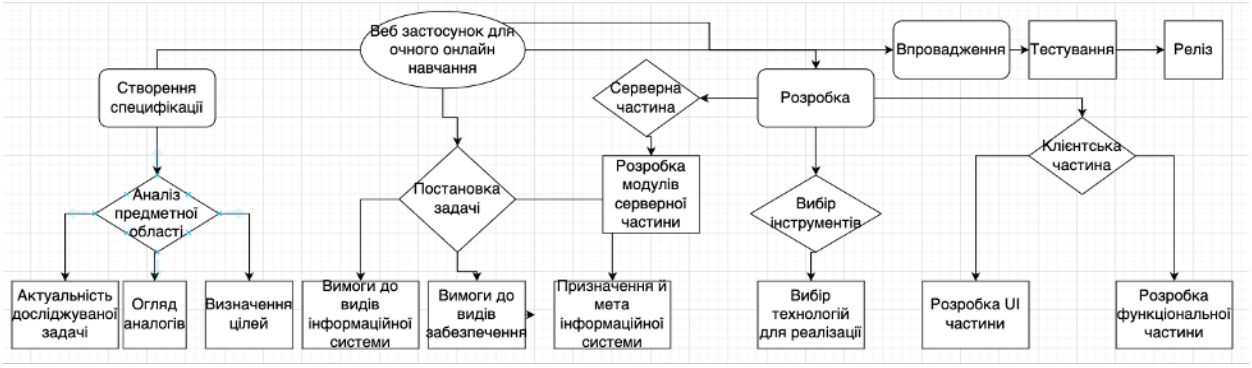


Рисунок Б.1 – WBS структура проекту

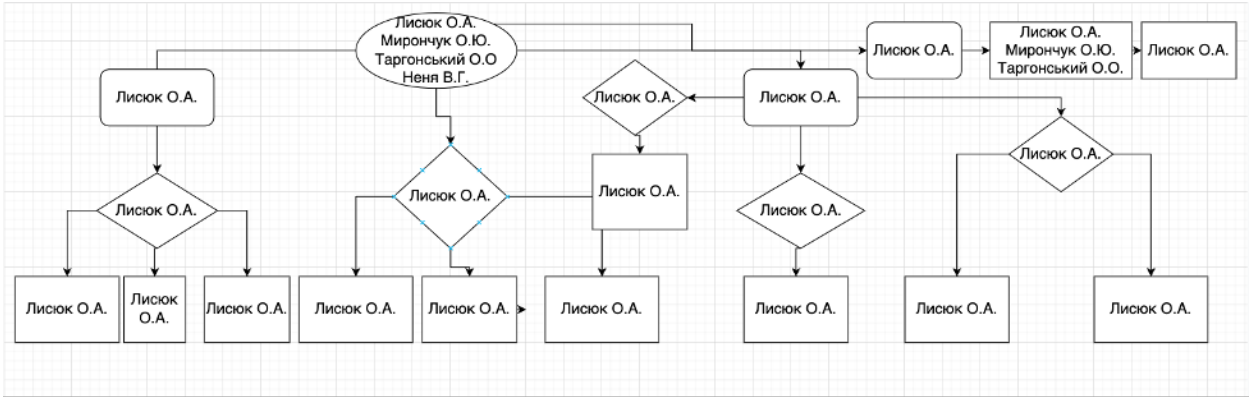


Рисунок Б.2 – OBS структура проекту

Список виконавців проекту описано в таблиці Б.2

Таблиця Б.2 – Виконавці проекту

Роль	**Ім'я**	**Роль на проекті**
Розробник	Лисюк О.А.	Розробка всього веб-застосунку з необхідним функціоналом
Проектувальник	Лисюк О.А.	Проектування структури веб-застосунку
Тестувальник	Лисюк О.А. Мирончук О.Ю. Таргонський О.О	Тестування візуальної та функціональної частини проекту.
Менеджер	Лисюк О.А.	Розподіляє ресурси на виконання завдань та керує всіма процесами.
Керівник проекту	Парфененко Ю.В.	Видає завдання на розробку застосунку.

Діаграма Ганта — це популярна діаграма стовпчастого типу, яка використовується для ілюстрації плану, графіка робіт за певним проєктом. Завдяки їй можливо отримати певне уявлення про обмеження у ресурсах, тривалість процесів з урахуванням не робочих днів. Календарний графік проєкту можемо спостерігати на рисунку Б.3.

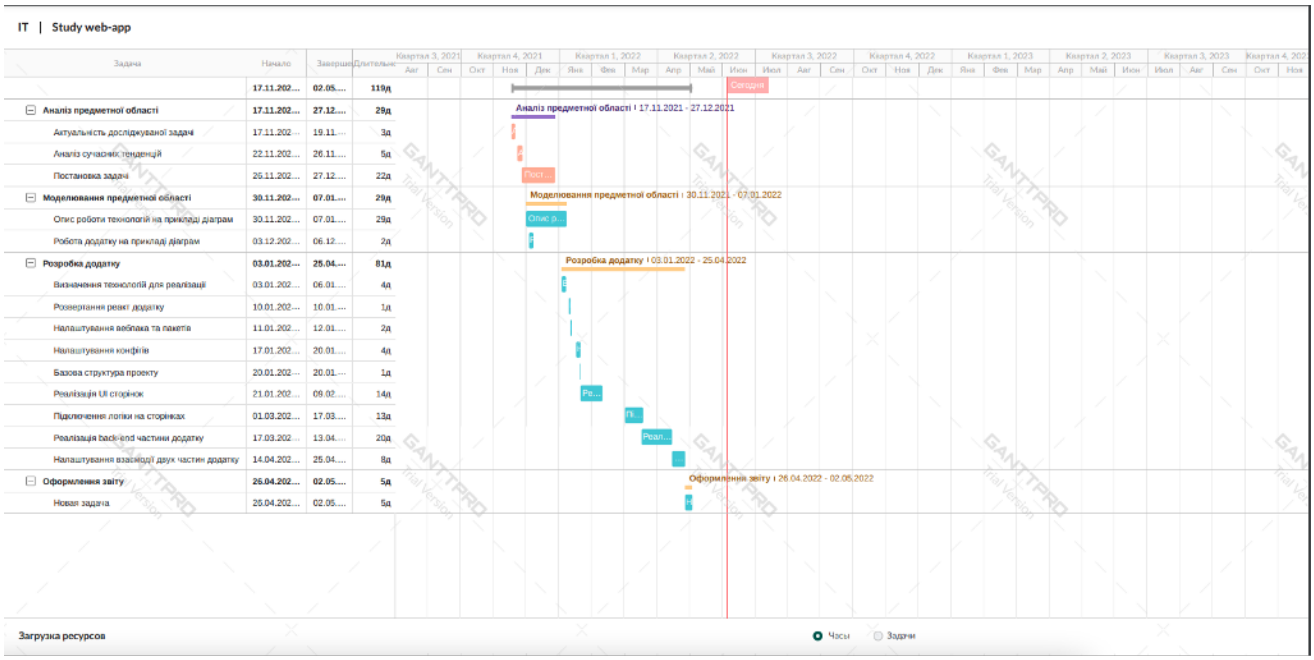


Рисунок Б.3. – Діаграма Ганта.

ДОДАТОК В

Лістинг файла base.apis.ts

```
import { AxiosRequestConfig } from 'axios';
import { httpApiClientV3 } from 'configs/http-client';

export class BaseApi {
  constructor(protected url: string) {}

  getList<R>(page?: number, config?: AxiosRequestConfig) {
    return httpApiClientV3.get<R>(`${this.url}?page=${page || 1}`, config);
  }

  deleteItem(id: number, config?: AxiosRequestConfig) {
    return httpApiClientV3.delete(`${this.url}/${id}`, config);
  }

  createItem<T, R>(data: T, config?: AxiosRequestConfig) {
    return httpApiClientV3.post<R>(this.url, data, config);
  }

  editItem<T extends { id: number }, R>(data: T, config?: AxiosRequestConfig) {
    const { id, ...payload } = data;
    return httpApiClientV3.put<R>(`${this.url}/${id}`, payload, config);
  }
}
```

Лістинг файла http-client.ts

```
import axios, { AxiosInstance } from 'axios';
import { ACCESS_TOKEN_KEY, REMEMBER_ME_KEY } from 'constants/auth';
import { resetUserAuthStorage } from 'utils';

const { REACT_APP_NEXT_PUBLIC_API_URL = '' } = process.env;

function createHttpClient(endPoint = ''): AxiosInstance {
  const instance = axios.create({
    baseURL: `${REACT_APP_NEXT_PUBLIC_API_URL}${endPoint}`,
    timeout: 0,
  });

  instance.interceptors.request.use(config => {
    const isRemember = !!localStorage.getItem(REMEMBER_ME_KEY);
    const token = (isRemember ? localStorage :
sessionStorage).getItem(ACCESS_TOKEN_KEY);
    if (token) {
      config.headers = {
        Authorization: `Bearer ${token}`,
      };
    }
    return config;
  });

  instance.interceptors.response.use(
    response => {
```

```
      return response;
    },
    error => {
      if (error?.response?.status === 401 && window.navigator.onLine) {
        resetUserAuthStorage();
      }
      return error;
    },
  );

  return instance;
}

export const httpApiClient = createHttpClient('/api/v1');
export const httpApiClientV2 = createHttpClient('/api/v2');
export const httpApiClientV3 = createHttpClient('/api/v3');
```

Лістинг файла useOutsideClick.ts

```
import { useEffect, RefObject } from 'react';

type Event = MouseEvent | TouchEvent;

const useOutsideClick = <T extends HTMLElement = HTMLElement>(
  ref: RefObject<T> | null,
  handler: (event: Event) => void,
): void => {
  useEffect(() => {
    const listener = (event: Event) => {
```

```
      const el = ref?.current;
      if (!el || el.contains((event?.target as Node) || null)) {
        return;
      }

      handler(event);
    };

    document.addEventListener('mousedown', listener);
    document.addEventListener('touchstart', listener);

    return () => {
      document.removeEventListener('mousedown', listener);
      document.removeEventListener('touchstart', listener);
    };
  }, [ref, handler]);
};
export default useOutsideClick;
```

Лістинг файла SidebarWithHeader.tsx

```
import React, { useEffect, useState } from 'react';
import { NavLink, Link, useLocation, useNavigate } from 'react-router-dom';
import {
  IconButton,
  Avatar,
  Box,
  CloseButton,
  Flex,
```

```
  HStack,
  VStack,
  Icon,
  useColorModeValue,
  Drawer,
  DrawerContent,
  Text,
  useDisclosure,
  Menu,
  MenuButton,
  MenuDivider,
  MenuItem,
  MenuList,
  useColorMode,
} from '@chakra-ui/react';
import { FiHome, FiTrendingUp, FiStar, FiSettings, FiMenu, FiInfo, FiChevronDown,
FiMoon } from 'react-icons/fi';

const LinkItems = [
  { name: 'Головна', icon: FiHome, path: '/' },
  { name: 'Повідомлення', icon: FiTrendingUp, path: '/messages' },
  { name: 'Контакти', icon: FiStar, path: '/contacts' },
  { name: 'Налаштування', icon: FiSettings, path: '/profile' },
];

export default function SidebarWithHeader({ children }) {
  const { isOpen, onOpen, onClose } = useDisclosure();

  return (
```

```
    <div style={{ minHeight: '100vh' }}>
      <Box minH="100vh" bg={useColorModeValue('gray.100', 'gray.900')}>
        <SidebarContent onClose={() => onClose} display={{ base: 'none', md: 'block' }}
/>
        <Drawer
          autoFocus={false}
          isOpen={isOpen}
          placement="left"
          onClose={onClose}
          returnFocusOnClose={false}
          onOverlayClick={onClose}
          size="full"
        >
          <DrawerContent>
            <SidebarContent onClose={onClose} />
          </DrawerContent>
        </Drawer>
        {/* mobilenav */}
        <MobileNav onOpen={onOpen} />
        <Box ml={{ base: 0, md: 60 }} p="4">
          {children}
        </Box>
      </Box>
    </div>
  );
}
```

Лістинг файла SidebarContent.tsx

```
const SidebarContent = ({ onClose, ...rest }) => {
  return (
    <Box
      transition="3s ease"
      bg={useColorModeValue('white', 'gray.900')}
      borderRight="1px"
      borderRightColor={useColorModeValue('gray.200', 'gray.700')}
      w={{ base: 'full', md: 60 }}
      pos="fixed"
      h="full"
      {...rest}
    >
      <Flex h="20" alignItems="center" mx="8" justifyContent="space-between">
        <Text fontSize="2xl" fontFamily="monospace" fontWeight="bold">
          Logo
        </Text>
        <CloseButton display={{ base: 'flex', md: 'none' }} onClick={onClose} />
      </Flex>
      {[...LinkItems].map((link) => (
        <NavItem key={link.name} icon={link.icon} path={link.path}>
          {link.name}
        </NavItem>
      ))}
    </Box>
  );
};
```

Лістинг файла NavItem.tsx

```
const NavItem = ({ icon, children, path, ...rest }) => {
  const location = useLocation();
  return (
    <NavLink to={path} _focus={{ boxShadow: 'none' }}>
      <Flex
        align="center"
        p="4"
        mx="4"
        borderRadius="lg"
        role="group"
        cursor="pointer"
        _hover={{
          bg: 'cyan.400',
          color: 'white',
        }}
        style={{
          background: location.pathname === path ? '#0BC5EA' : 'inherit',
          color: location.pathname === path ? 'white' : 'inherit',
        }}
        {...rest}
      >
        {icon && (
          <Icon
            mr="4"
            fontSize="16"
            _groupHover={{
              color: location.pathname === path ? 'white' : 'inherit',
            }}
            as={icon}
```

```
        />
       )}
       {children}
     </Flex>
   </NavLink>
  );
};
```

Лістинг файла MobileNav.tsx

```
const MobileNav = ({ onOpen, user,...rest }) => {
  const { toggleColorMode } = useColorMode();
  const navigate = useNavigate();

  const [n, setN] = useState(Boolean(localStorage.getItem(${user.id}/n)));

  const toggleNotify = () => {
    setN(!n);
  };
return (
    <Flex
      ml={{ base: 0, md: 60 }}
      px={{ base: 4, md: 4 }}
      height="20"
      alignItems="center"
      bg={useColorModeValue('white', 'gray.900')}
      borderBottomWidth="1px"
      borderBottomColor={useColorModeValue('gray.200', 'gray.700')}
      justifyContent={{ base: 'space-between', md: 'flex-end' }}
```

```
    {...rest}
  >
    <IconButton
      display={{ base: 'flex', md: 'none' }}
      onClick={onOpen}
      variant="outline"
      aria-label="open menu"
      icon={<FiMenu />}
    />
    <Text display={{ base: 'flex', md: 'none' }} fontSize="2xl" fontFamily="monospace" fontWeight="bold">
      SUMDU
    </Text>
    <IconButton onClick={toggleColorMode} variant="outline" aria-label="test" icon={<FiMoon />} />
    <HStack spacing={{ base: '0', md: '6' }} marginLeft="10px" marginRight="10px">
      <IconButton
        onClick={() => navigate('/chart')}
        variant={n ? 'outline' : 'ghost'}
        aria-label="open menu"
        icon={<FiInfo />}
      />

      <Flex alignItems={'center'}>
        <Menu>
          <MenuButton py={2} transition="all 0.3s" _focus={{ boxShadow: 'none' }}>
            <HStack>
              <Avatar size={'sm'} src={user.url} marginLeft="10px" />
```

```
            <VStack display={{ base: 'none', md: 'flex' }} alignItems="flex-start" spacing="1px" ml="2">
              <Text fontSize="sm">{user.name}</Text>
              <Text fontSize="xs" color="gray.600">
                {user?.role === 'admin' ? 'Викладач' : 'Студент'}
              </Text>
            </VStack>
            <Box display={{ base: 'none', md: 'flex' }}>
              <FiChevronDown />
            </Box>
          </HStack>
        </MenuButton>
        <MenuList
          bg={useColorModeValue('white', 'gray.900')}
          borderColor={useColorModeValue('gray.200', 'gray.700')}
        >
          <MenuItem>
            <Link to="/profile">Налаштування профілю</Link>
          </MenuItem>
          <MenuDivider />
          <MenuItem>
            <Link to="/login">Вийти</Link>
          </MenuItem>
        </MenuList>
      </Menu>
    </Flex>
  </HStack>
</Flex>
);
```

```
};
```

Лістинг файла Profile.tsx

```
import { Box, Text, Button, Flex, Image } from '@chakra-ui/react';
import { useNavigate, useParams } from 'react-router-dom';
import Header from '../components/Header';
import { UserInterface } from ‘interfaces’

export default ({item}: {item: UserInterface}) => {
  const params = useParams();
  const navigate = useNavigate();

  const { name, email, role, url } = item || {};

  return (
    <Header>
      <Flex flexDir="column" p="15px" maxW="600px">
        <Flex position="relative" flexDir={{ base: 'column', sm: 'row' }} gap="20px">
          <Image
            alignSelf={{ base: 'center', sm: 'flex-start' }}
            borderRadius="8px"
            boxSize="200px"
            objectFit="cover"
            src={url}
            alt=""
```

```
        />
        <Box w="100%">
          <Flex w="100%" justifyContent={{ base: 'center', sm: 'flex-end' }}>
            <Button type="button" onClick={() => navigate('/messages/1')}>
              Написати повідомлення
            </Button>
          </Flex>
          <Text pb={'3px'} mt="8px" textAlign="left" borderBottom="1px" borderColor="#0BC5EA">
            Name: {name}
          </Text>
          <Text pb={'3px'} mt="8px" textAlign="left" borderBottom="1px" borderColor="#0BC5EA">
            Email: {email}
          </Text>
          <Text pb={'3px'} mt="8px" textAlign="left" borderBottom="1px" borderColor="#0BC5EA">
            Role: {role}
          </Text>
        </Box>
      </Flex>
      { item?.details && <Box bg="#e3e3e3" h="100px" p="10px" mt="15px" borderRadius="8px">
        <Text textAlign="left">Детальна інформація про користувача прихована</Text>
      </Box> }
    </Flex>
  </Header>
 );
```

```
};
```

Лістинг файла Login.tsx

```
import React, { useEffect, useState } from 'react';
import {
  Flex,
  Heading,
  Input,
  Button,
  InputGroup,
  Stack,
  chakra,
  Box,
  Link,
  Avatar,
  FormControl,
  FormHelperText,
  InputRightElement,
  FormErrorMessage,
} from '@chakra-ui/react';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import { useNavigate } from 'react-router-dom';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { useAppDispatch, asyncLoginUserAction } from ‘store’

export const VALIDATION_SCHEMA = Yup.object().shape({
```

```
  email: Yup.string().required('required').email('Email should be valid'),
  password: Yup.string()
    .required('Field is required')
    .min(5, 'Should be at least 5 characters')
    .max(15, 'Should be less than 15 characters'),
});

const Login = () => {
  const navigate = useNavigate();
  const dispatch = useAppDispatch();

  const [showPassword, setShowPassword] = useState(false);
  const handleShowClick = () => setShowPassword(!showPassword);

  const formik = useFormik({
    initialValues: {
      email: '',
      password: '',
    },
    validateOnChange: false,
    validationOnBlur: false,
    validationSchema: VALIDATION_SCHEMA,
    onSubmit: (values) => {
        dispatch(asyncLoginUserAction())
    }
  });
```

```
return (
   <Flex flexDirection="column" height="100vh" backgroundColor="gray.200" justifyContent="center" alignItems="center">
     <ToastContainer />
     <Stack flexDir="column" mb="2" justifyContent="center" alignItems="center">
       <Avatar bg="teal.500" />
       <Heading color="teal.400">Welcome</Heading>
       <Box minW={{ base: '90%', md: '468px' }}>
         <form onSubmit={formik.handleSubmit}>
           <Stack spacing={4} p="1rem" backgroundColor="whiteAlpha.900" boxShadow="md">
             <FormControl isInvalid={!!formik?.errors.email}>
               <InputGroup>
                 <Flex flexDir="column" width="100%">
                   <Input
                     width="full"
                     type="text"
                     placeholder="E-mail"
                     name={'email'}
                     onChange={formik.handleChange}
                     value={formik.values.email}
                   />
                   {formik.errors.email && <FormErrorMessage>{formik.errors.email}</FormErrorMessage>}
                 </Flex>
               </InputGroup>
             </FormControl>
             <FormControl isInvalid={!!formik?.errors.password}>
               <InputGroup>
```

```
            <Flex flexDir="column" width="100%">
              <Input
                type={showPassword ? 'text' : 'password'}
                name={'password'}
                onChange={formik.handleChange}
                value={formik.values.password}
                placeholder="Пароль"
              />
              {formik.errors.password &&
<FormErrorMessage>{formik.errors.password}</FormErrorMessage>}
            </Flex>
            <InputRightElement width="4.5rem">
              <Button h="1.75rem" size="sm" onClick={handleShowClick}>
                {showPassword ? 'Сховати' : 'Показати'}
              </Button>
            </InputRightElement>
          </InputGroup>
          <FormHelperText textAlign="right">
            <Link to=’/forgot’>Забули пароль?</Link>
          </FormHelperText>
        </FormControl>
        <Button
          isLoading={isLoading}
          borderRadius={0}
          type="submit"
          variant="solid"
          colorScheme="teal"
          width="full"
        >
```

```
            Авторизуватися
          </Button>
        </Stack>
      </form>
    </Box>
  </Stack>
  <Box>
    Не маєте аккаунту?
    <Link color="teal.500" href="/registration">
      Зареєструватися
    </Link>
  </Box>
 </Flex>
 );
};

export default Login;
```

Лістинг файла App.ts

```
import Home from './views/Home';
import UserProfile from './views/UserPage';
import { UsersBlock } from './views/Users';
import Messages from './views/Messages';
import UploadSubject from './views/UploadSubject';
import Subject from './views/Subject';
import Register from './views/Register';
import { Chat } from './views/Chat';
import ProfilePage from './views/ProfilePage';
```

```
import Lesson from './views/Lesson';
import AdminTheme from './views/AdminTheme';
import Chart from './views/Chart';
import Login from './views/Login';
import './style.css';
import SubjectsCreate from './views/SubjectsCreate';
import ThemeEdit from './views/ThemeEdit';
import {useAppDispatch, useAppSelector, selectUser} from '@modules/store'
import {getUserInfo} from '@modules/auth/actions';

function App() {
  const [isAuth, setIsAuth] = useState(null);
  const navigate = useNavigate();
  const dispatch = useAppDispatch();

const user = useAppSelector(selectUser)

  useEffect(() => {
    dispatch(getUserInfo())
  }, [user?.id, dispatch]);

useEffect(() => {
   if(user?.id) setIsAuth(true)
},[userInfo])

  return (
    <ChakraProvider>
      <Routes>
        <Route path="/" element={<Home />} />
```

```
        <Route path="/profile" element={<UserProfile />} />
        <Route path="/contacts" element={<UsersBlock />} />
        <Route path="/messages" element={<Messages />} />
        <Route path="/subjects/:id/upload/:themeId" element={<UploadSubject />} />
        <Route path="/subjects/:id/themes/:themeId" element={<Lesson />} />
        <Route path="/subjects/:id/finish/:themeId" element={<AdminTheme />} />
        <Route path="/subjects/:id" element={<Subject />} />
        <Route path="/messages/:id" element={<Chat />} />
        <Route path="/subjects/create" element={<SubjectsCreate />} />
        <Route path="/themes/create" element={<ThemeEdit />} />

        {isAuth !== null && !isAuth && (
          <>
            <Route path="/register" element={<Register />} />
            <Route path="/login" element={<Login />} />
          </>
        )}
        <Route path="/contacts/:id" element={<ProfilePage />} />
        <Route path="/chart" element={<Chart />} />
      </Routes>
    </ChakraProvider>
  );
}

export default App;
```

Лістинг файла Users.tsx

```
import { Box, Text, Image, Flex } from '@chakra-ui/react';
```

```
import { EmailIcon, InfoIcon } from '@chakra-ui/icons';
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import {getUserInfo} from '@modules/auth/actions';
import Header from '../components/Header';
import axios from 'axios';

export const UsersBlock: React.FC = () => {
  const navigate = useNavigate();

  const [list, setList] = useState([]);

const user = useAppSelector(selectUser)

  useEffect(() => {
    axios.get('https://lysiukapi.herokuapp.com/users/list').then((res) => {
      setList(res.data?.filter((i) => i.id !== user?.id) || []);
    });
  }, []);

  return (
    <Header>
      <Box>
        <Box>
          <Flex background="cyan.500" borderRadius="5px" w="100%" h="40px"
pos="relative" alignItems={'center'}>
            <Image
              pos="absolute"
              top="-1"
```

```
    left="-1"
    boxSize="27px"
    objectFit="cover"
    src="https://flyclipart.com/thumb2/user-icon-133449.png"
    alt="Dan Abramov"
   />
   <Text color="white" textAlign="left" pl="30px">
    Контакти
   </Text>
  </Flex>
  <Flex>
   {list?.length
    ? list.map(({ name, id , src}) => (
       <Box py="20px" px="20px" w="130px" key={id}>
        <Image
         boxSize="120px"
         objectFit="cover"
         src={src ||
          “”}
        alt={name}  />
        <Text
         style={{
          textOverflow: 'ellipsis',
          whiteSpace: 'nowrap',
          overflow: 'hidden',
          width: '100%',
         }}
        >
         {name}
```

```
                </Text>
                <Flex justifyContent="center">
                  <EmailIcon
                    onClick={() => {
                      axios
                        .post(`https://lysiukapi.herokuapp.com/messages/start-chat`, {
                          userIds: [Number(id), Number(user?.id)],
                        })
                        .catch(() => {})
                        .finally(() => navigate(`/messages/${id}`));
                    }}
                    cursor="pointer"
                    mr="5px"
                  />
                  <InfoIcon onClick={() => navigate(`/contacts/${id}`)} cursor="pointer" />
                </Flex>
              </Box>
            ))
          : null}
        </Flex>
      </Box>
    </Box>
  </Header>
 );
};
```

Лістинг файла SubjectsCreate.tsx

```
import React, { useState } from 'react';
```

```
import Header from '../components/Header';
import { Box, Button, Checkbox, Flex, Input, Progress, Text, Textarea } from '@chakra-ui/react';
import { ToastContainer, toast } from 'react-toastify';
import { useFormik } from 'formik';
import { format } from 'date-fns';
import { useNavigate } from 'react-router-dom';
import { useAppDispatch} from '@modules/store';
import {SubjectInterface} from '@interfaces';
import {CreateSubject} from '@store/modules/actions';

const SubjectsCreate = () => {
  const [isLoading, setIsLoading] = useState(false);
  const user = JSON.parse(localStorage.getItem('user'));
  const navigate = useNavigate();
  const dispatch = useAppDispatch();

  const formik = useFormik({
    initialValues: {
      subject: '',
    },
    onSubmit: (values) => {
      setIsLoading(true);
     dispatch(createSubject(values as SubjectInterface))
      toast.success(`Предмет ${values.subject} було успішно створено`);
      setIsLoading(false);
    },
  });
```

```
 return (
  <Header>
    {isLoading && <Progress size="xs" colorScheme="cyan" isIndeterminate />}
    <ToastContainer />
    <form onSubmit={formik.handleSubmit}>
     <Text fontSize="3xl">Створити предмет</Text>
     <Box p="20px">
      <Box w="80%">
       <Flex mt="10px">
        <Input
         borderColor="black"
         ml="5px"
         h="30px"
         name="subject"
         onChange={formik.handleChange}
         value={formik.values.subject}
         placeholder="Назва нового предмета"
        />
       </Flex>
      </Box>

      <Flex>
       <Button isLoading={isLoading} mt="10px" ml="5px" type="submit"
colorScheme="blue">
         Submit
       </Button>
      </Flex>
      <Flex>
```

```
          <Button onClick={() => navigate('/themes/create')} mt="10px" ml="5px"
colorScheme="blue">
            Create new theme
          </Button>
        </Flex>
      </Box>
    </form>
  </Header>
 );
};

export default SubjectsCreate;
```

Лістинг файла AdminTable.tsx

```
import { useEffect, useState } from 'react';
import { useLocation, useNavigate } from 'react-router-dom';
import { Text, Box, List, ListItem } from '@chakra-ui/react';
import {useAppSelector, useAppDispatch, selectUser} from ‘@modules/store’

const TableComponent = () => {
  const navigate = useNavigate();
  const dispatch = useAppDispatch();
  const { state } = useLocation();

const selector = useAppSelector(selectSubjects)

const [list, setList] = useState(selector);
```

```
  const [a, sa] = useState(0);

const user = useAppSelector(selectUser);

useEffect(() => {
    dispatch(setList(selector))
  }, [selector]);

  useEffect(() => {
    dispatch(getSubjects(state?.id as number))
  }, [dispatch, getSubjects]);

  return (
    <Box p="20px">
      <Text fontSize="3xl">Створені мною предмети</Text>
      <List>
        {list?.length
          ? list.map((i) => (
              <ListItem
                key={i.id}
                cursor="pointer"
                onClick={() => navigate(`/subjects/${i.id}`)}
                p="20px"
                mb="20px"
                borderRadius="10px"
                style={{ border: '1px solid rgb(11, 197, 234)' }}
              >
                {i.subject}
              </ListItem>
```

```
            ))
          : null}
      </List>
    </Box>
  );
};

export default TableComponent;
```

Лістинг файла server.js

```
const express = require('express')
const http = require('http')
const path = require('path')
const cors = require('cors')
const formData = require('express-form-data')

require('dotenv').config()

const app = express()
const server = http.createServer(app)
app.use(cors({}))

const PORT = process.env.PORT || 5000

try {
  require('./config/database')()

  app.use(express.urlencoded({ extended: true }))
```

```
  app.use(express.json())

  app.use(formData.parse())

  require('./src/routes')(app)

  server.listen(PORT, () => {
    console.log(`Server listening on port ${PORT}`)
  })
} catch (e) {
  console.log('Error:', e)
}

module.exports = app
```

Лістинг файла files.js

```
const path = require('path')

const FilesService = require('../services/files')
const { saveFile } = require('../utils')

exports.upload = async (req, res) => {
  try {
    const { imageUrl, filename, tempFileName, buffer } = await saveFile(
      req.files.file,
    )
    const upload = await FilesService.upload({
      imageUrl,
```

```
    })

    return res.status(200).json(upload)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.download = async (req, res) => {
  try {
    const file = await FilesService.download({ id: req.params.file })
    const fpath = path.join(__dirname, '../..', '/' + file)

    return res.download(fpath)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.getFiles = async (req, res) => {
  try {
    const list = await FilesService.getFiles()

    return res.status(200).json(list)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
```

```
  }
}
```

Лістинг файла messages.js

```
const MessagesService = require('../services/messages')
const { get, isNil } = require('lodash')

exports.sendMessage = async (req, res) => {
  try {
    const send = await MessagesService.sendMessage(req.body)

    return res.status(200).json(send)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.getMessages = async (req, res) => {
  try {
    const messages = await MessagesService.getMessages(req.query)

    return res.status(200).json(messages)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}
```

```
exports.startChat = async (req, res) => {
  try {
    const start = await MessagesService.startChat(req.body)

    return res.status(200).json(start)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.getChatList = async (req, res) => {
  try {
    const getList = await MessagesService.getChatList(req.query)

    return res.status(200).json(getList)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}
```

Лістинг файла users.js

```
const { get } = require('lodash')
const UsersService = require('../services/users')
const { saveFile } = require('../utils')

exports.register = async (req, res) => {
```

```
  try {
    const created = await UsersService.register(req.body)

    return res.status(200).json(created)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.login = async (req, res) => {
  try {
    const token = await UsersService.login(req.body)

    return res.status(200).json(token)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.getUserById = async (req, res) => {
  try {
    const id = get(req, 'params.id', null)
    const user = await UsersService.getUserById(id)

    return res.status(200).json(user)
  } catch (e) {
    console.log(e)
```

```
    return res.status(500).json(e)
  }
}
exports.list = async (req, res) => {
  try {
    const { limit, offset } = req.query
    const users = await UsersService.getUsersList({
      limit: Number(limit),
      offset: Number(offset),
    })

    return res.status(200).json(users)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}

exports.updateUser = async (req, res) => {
  try {
    const update = await UsersService.updateUser(req.body)

    return res.status(200).json(update)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}
```

```
exports.updateAvatar = async (req, res) => {
  try {
    const { buffer } = await saveFile(req.files.avatar)
    const userId = get(req, 'body.id')

    const update = await UsersService.updateAvatar({ userId, buffer })

    return res.status(200).json(update)
  } catch (e) {
    console.log(e)
    return res.status(500).json(e)
  }
}
```

Лістинг файла models/chats.js

```
const { Schema, model } = require('mongoose')

const chats = new Schema({
  id: {
    type: Number,
    unique: true,
  },
  timestamp: {
    type: Number,
  },
  users: [
    {
      ref: 'users',
```

```
    type: 'ObjectId',
   },
  ],
  messages: [
   {
    ref: 'messages',
    type: 'ObjectId',
   },
  ],
})

module.exports = model('Chats', chats)
```

Лістинг файла models/files.js

```
const { Schema, model } = require('mongoose')

const files = new Schema({
  id: {
   type: Number,
   unique: true,
  },
  timestamp: {
   type: Number,
  },
  file: {
   type: String,
  },
})
```

```
module.exports = model('Files', files)
```

Лістинг файла models/messages.js

```
const { Schema, model } = require('mongoose')

const messages = new Schema({
  id: {
    type: Number,
    unique: true,
  },
  content: {
    type: String,
  },
  timestamp: {
    type: Number,
  },
  sender: {
    ref: 'users',
    type: 'ObjectId',
  },
})

module.exports = model('Messages', messages)
```

Лістинг файла models/users.js

```
const { Schema, model } = require('mongoose')
```

```
const users = new Schema({
  id: {
    type: String,
    unique: true,
  },
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    unique: true,
    required: true,
  },
  password: {
    type: String,
    required: true,
  },
  avatar: {
    type: Buffer,
    contentType: String,
  },
  role: {
    type: Number,
    default: 2,
  },
})

module.exports = model('Users', users)
```

Лістинг файла routes/users.js

```
const router = require('express').Router()
const multer = require('multer')

const Users = require('../controllers/users')

try {
  router.post('/register', Users.register)
  router.post('/login', Users.login)

  router.patch('/update', Users.updateUser)
  router.patch('/avatar', Users.updateAvatar)

  router.get('/list', Users.list)
  router.get('/:id', Users.getUserById)
} catch (e) {
  console.log(e)
}

module.exports = router
```

Лістинг файла routes/files.js

```
 const router = require('express').Router()

const Files = require('../controllers/files')
```

```
router.post('/upload', Files.upload)

router.get('/download/:file', Files.download)
router.get('/list', Files.getFiles)

module.exports = router
```

Лістинг файла services/auth.js

```
const jwt = require('jsonwebtoken')

exports.auth = async (req, res, next) => {
  const header = req.headers['authorization'] || req.headers['Authorization']
  if (!header) return res.sendStatus(401)

  const token = header.split(' ')[1]
  if (!token) return res.sendStatus(401)

  jwt.verify(token, process.env.JWT_TOKEN, async (err, user) => {
    if (err) return res.sendStatus(401)

    req.user = user
    next()
  })
}

exports.generateAccessToken = ({ id, email, date }) =>
  jwt.sign({ id, email, date }, process.env.JWT_TOKEN)
```

Лістинг файла errors.js

```
module.exports = {
  E1: {
    description: 'User already exists',
    statusCode: 400,
    errorCode: 'E1',
  },
  E2: {
    description: 'Invalid email',
    statusCode: 400,
    errorCode: 'E2',
  },
  E3: {
    description: 'Invalid password. (From 5 to 15 symbols allowed)',
    statusCode: 400,
    errorCode: 'E3',
  },
  E4: {
    description: 'User not found',
    statusCode: 403,
    errorCode: 'E4',
  },

  E5: {
    description: 'Invalid credentials',
    statusCode: 401,
```

```
    errorCode: 'E5',
  },
  E6: {
    description: 'Id is missing',
    statusCode: 400,
    errorCode: 'E6',
  },
  E7: {
    description: 'Name is missing',
    statusCode: 400,
    errorCode: 'E7',
  },
  E8: {
    description: 'Old password is incorrect',
    statusCode: 403,
    errorCode: 'E8',
  },
  E9: {
    description: 'Chat already exists',
    statusCode: 400,
    errorCode: 'E9',
  },
  E10: {
    description: 'File not exists',
    statusCode: 400,
    errorCode: 'E10',
  },
}
```

Лістинг файла services/files.js

```
const createError = require('http-errors')
const Files = require('../models/files')
const { dateToMs, generateId } = require('../utils')
const errors = require('./errors')

exports.upload = async ({ imageUrl }) => {
  const message = await Files.create({
    id: generateId(999999999, 1),
    timestamp: dateToMs(new Date()),
    file: imageUrl,
  })

  return message
}

exports.download = async ({ id }) => {
  const file = await Files.findOne({ id })
  if (!file) throw createError(errors['E10'])

  return file.file
}

exports.getFiles = async () => {
  const files = await Files.find({})
```

```
  return files.file
}
```

Лістинг файла services/messages.js

```
const createError = require('http-errors')
const { map } = require('lodash')

const Chats = require('../models/chats')
const Messages = require('../models/messages')
const Users = require('../models/users')
const { dateToMs, generateId } = require('../utils')
const errors = require('./errors')

exports.sendMessage = async ({ userId, chatId, content }) => {
  const sender = await Users.findOne({ id: userId })

  const forCreate = {
    sender: sender._id,
    content,
    id: generateId(999999999, 1),
    timestamp: dateToMs(new Date()),
  }

  const message = await Messages.create({ id: userId, ...forCreate })

  const chat = await Chats.updateOne(
    { id: chatId },
    { $push: { messages: message._id } },
```

```
  )

  return { message, chat }
}

exports.getMessages = async ({ chatId }) => {
  const chat = await Chats.findOne({ id: chatId })
    .populate({
      path: 'messages',
      model: Messages,
    })
    .populate({ path: 'users', model: Users })

  return chat
}

exports.startChat = async ({ userIds = [] }) => {
  const users = await Users.find({ id: { $in: userIds } }, { _id: 1 })
  const ids = map(users, (el) => el._id)

  const isExists = await Chats.findOne({ users: ids })

  if (isExists)
    throw createError({ ...errors['E9'], existingChatId: isExists.id })

  const start = await Chats.create({
    id: generateId(999999999, 1),
    timestamp: dateToMs(new Date()),
    users: ids,
```

```
    messages: [],
  })

  return start
}

exports.getChatList = async ({ userId }) => {
  const user = await Users.findOne({ id: userId }, { _id: 1 })

  const chat = await Chats.find({ users: user._id })
    .populate({
      path: 'users',
      model: Users,
    })
    .populate({ path: 'messages', model: Messages })

  return chat
}
```

Лістинг файла services/users.js

```
const sharp = require('sharp')
const path = require('path')
const createError = require('http-errors')
const { replace, get, isEmpty, assign, isNil } = require('lodash')
const bcrypt = require('bcrypt')

const errors = require('./errors')
const Users = require('../models/users')
```

```
const { generateAccessToken } = require('./auth')
const { dateToMs, isValidEmail, generateId } = require('../utils')

const salt = 10

exports.register = async ({ email, password, name }) => {
  const isExists = await Users.findOne({ email })

  if (isExists) throw createError(errors['E1'])
  if (!isValidEmail(email)) throw createError(errors['E2'])
  if (!name) throw createError(errors['E7'])

  replace(password, ' ', '')
  if (password.length < 5 || password.length > 15)
    throw createError(errors['E3'])

  const hash = await bcrypt.hash(password, salt)
  const created = await Users.create({
    id: generateId(999999999, 1),
    email,
    name,
    password: hash,
  })

  return created.toObject()
}

exports.login = async ({ email, password }) => {
  const user = await Users.findOne({ email })
```

```
  if (!user) throw createError(errors['E4'])

  const verify = bcrypt.compareSync(password, user.password)
  if (!verify) throw createError(errors['E5'])

  const id = get(user, 'id', null)
  const token = generateAccessToken({
    id,
    email,
    date: dateToMs(new Date()),
  })

  return { token, user }
}

exports.getUserById = async (id) => {
  if (!id) throw createError(errors['E6'])

  const user = await Users.findOne({ id })
  if (!user) throw createError(errors['E4'])

  return user.toObject()
}

exports.getUsersList = async ({ limit = 10, offset = 0 }) => {
  const arr = await Users.find().skip(offset).limit(limit)

  return arr
}
```

```
exports.updateUser = async ({ id, info, passwords }) => {
  if (!id) throw createError(errors['E6'])

  const user = await Users.findOne({ id })
  if (!user) throw createError(errors['E4'])

  const forUpdate = {}
  if (!isEmpty(passwords)) {
    const { oldPassword, newPassword } = passwords

    const verify = bcrypt.compareSync(oldPassword, user.password)
    if (!verify) throw createError(errors['E8'])

    const password = await bcrypt.hash(newPassword, salt)
    forUpdate['password'] = password
  }
  if (!isEmpty(info)) {
    const { email, name } = info
    if (!isValidEmail(email)) throw createError(errors['E2'])
    if (!isNil(email)) forUpdate['email'] = email
    if (!isNil(name)) forUpdate['name'] = name
  }
  const update = await Users.updateOne({ id }, forUpdate)

  return update
}
exports.updateAvatar = async ({ buffer, userId }) => {
  const user = await Users.updateOne({ id: userId }, { avatar: buffer })
```

```
  return user
}
```