МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ

КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

**Пояснювальна записка**

**до дипломної роботи**

**\_\_\_\_\_\_\_\_бакалавр \_\_\_\_\_\_\_\_**

(освітньо-кваліфікаційний рівень)

**на тему «Розробка клієнтської частини навчальної платформи на базі React.js»**

Виконала: студентка 4 курсу, групи ІТ-651

напряму підготовки 6.050103 „Програмна інженерія”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кізілова А.Є.

(підпис)

Керівник,

доцент, к.т.н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фесенко Т.М.

(підпис)

Рецензент,

доцент каф. ПМ,

к.ф.-м.н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ковальов Ю.Г.

(підпис)

СЄВЄРОДОНЕЦЬК – 2019

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

бакалаврської роботи студентки гр. ІТ-651 Кізілової А.Є.

Науковий керівник

Доцент, к.т.н. Фесенко Т.М.

ПІБ, посада

Оцінка наукового керівника:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Рецензент Ковальов Ю.Г., к.ф.-м.н, доцент каф. ПМ СНУ ім. В. Даля

ПІБ, місто роботи, посада

Оцінка рецензента:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кінцева оцінка за результатами захисту: \_\_\_\_\_\_\_\_\_\_\_

Голова ЕК,

Зав. кафедри ПМ

д.т.н., доцент \_\_\_\_\_\_\_\_\_\_\_\_ Лифар В.О

підпис

**СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ВОЛОДИМИРА ДАЛЯ**

Факультет інформаційних технологій та електроніки

Кафедра програмування та математики

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050103 „Програмна інженерія”

|  |
| --- |
| ЗАТВЕРДЖУЮ  Завідувач кафедри ПМ,  д.т.н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_ Лифар В.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019 р. |

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кізілової Аліни Євгенівни

**1. Тема роботи Розробка клієнтської частини навчальної платформи на базі React.js.**

**керівник роботи к.т.н., доцент Фесенко Тетяна Миколаївна**

затверджені наказом вищого навчального закладу від “23” квітня 2019 року №68/14.04

2. Строк подання студентом роботи 6 червня 2019 р.

3. Вихідні дані до роботи

Об'єктом досліджень є клієнт-серверні технології на основі веб-додатку навчальної платформи.

3.1 Літературні джерела:

The Road to learn React. [Robin Wieruch](https://leanpub.com/u/rwieruch). – 2018, 203 с.

The Road to GraphQL. Robin Wieruch. – 2018, 355 с.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ

4.2 Аналіз предметної галузі (огляд літератури), з висвітленням наступних питань:

Актуальність навчальних платформ

Порівняння існуючих аналогів

Формування вимог до навчальної платформи

4.3 Основна частина, в якої висвітлити:

Архітектура клієнт-серверної взаємодії

Аналіз та проектування веб-додатка

Вибір стеку технологій

Розробка клієнтської частини веб-додатка

4.4 Висновки

4.4 Перелік використаних джерел

5. Перелік графічного матеріалу немає

6. Дата видачі завдання 11 березня 2019 року.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Примітка |
| 1 | Одержання завдання на виконання роботи | 23.04.19 |  |
| 2 | Укладання і погодження з керівником плану і етапів виконання роботи | 23.04.19 |  |
| 3 | Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі» | 26.04.19 |  |
| 4 | Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху | 02.05.19 |  |
| 5 | Проектування задачі що реалізується. | 09.05.19 |  |
| 6 | Укладання програмного продукту | 11.05.19 |  |
| 7 | Укладання, оформлення та погодження пояснювальної записки з керівником | 25.05.19 |  |
| 8 | Здача готової пояснювальної записки на кафедру | 06.06.19 |  |
| 9 | Укладання доповіді і презентації | 07.06.19 |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Кізілова А.Є.

(підпис)

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ Фесенко Т.М.

(підпис)

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

UI (User Interface) - Інтерфейс користувача

SPA (Single-Page Application) - Односторінковий додаток

SOAP (Simple Object Access Protocol) - Протокол обміну структурованими повідомленнями

REST (Representational State Transfer) - Передача репрезентативного стану

JSON (JavaScript Object Notation) - запис об'єктів JavaScript

API (Application Programming Interface) - Прикладний програмний інтерфейс

ПЗ – програмне забезпечення

HTML (Hypertext Markup Language ) - Мова розмітки гіпертекстових документів

DOM (Document Object Model) - Об'єктна модель документа

РЕФЕРАТ

Текст – 55 с., рис. – 28, табл. – 1, додатків – 0, літературних джерел – 18

У ході виконання даної дипломної роботи було проведено дослідження

предметної області, в якості якого виступав безпосередній процес розробки клієнтьскої частини веб-додатка навчальної платформи.

Були виявлені проблеми існуючих рішень, розроблені бізнес завдання, узгоджені вимоги до навчальної платформи, і на підставі проведеної аналітики було визначено модель проектування і вибір найбільш доступних і оптимальних технологій.

В якості результатів всіх досліджень була розроблена клієнтська частина веб-додатку навчальної платформи.

Ключові слова: веб-додаток, SPA, GraphQL, React.js, Apollo Client.

ЗМІСТ

[ВСТУП 9](#_Toc11390100)

[РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД 11](#_Toc11390101)

[1.1 Актуальність навчальних платформ 11](#_Toc11390102)

[1.2 Існуючі навчальні платформи з Node.js 11](#_Toc11390103)

[1.3 Недоліки сучасних сервісів і шляхи вирішення проблем 14](#_Toc11390104)

[1.4 Формування вимог до навчальної платформи 14](#_Toc11390105)

[РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ТЕХНОЛОГІЙ КЛІЄНТСЬКОЇ ЧАСТИНИ 16](#_Toc11390106)

[2.1 Архітектура клієнт-серверної взаємодії 16](#_Toc11390107)

[2.2 Клієнтська частина веб-додатку 17](#_Toc11390108)

[2.3 Аналіз та проектування 18](#_Toc11390109)

[2.4 Створення додатку у вигляді SPA 21](#_Toc11390110)

[2.5 Вибір стеку технологій 23](#_Toc11390111)

[2.5.1 React 23](#_Toc11390112)

[2.5.2 GraphQL 26](#_Toc11390113)

[2.5.3 Apollo Client 28](#_Toc11390114)

[2.5.4 Reactstrap 29](#_Toc11390115)

[РОЗДІЛ 3. РЕЗУЛЬТАТ РОБОТИ 31](#_Toc11390116)

[ВИСНОВКИ 53](#_Toc11390120)

[СПИСОК ЛІТЕРАТУРИ 54](#_Toc11390121)

# ВСТУП

Актуальність досліджень:

У сучасному світі все більш поширюються ідеї отримання освіти у сфері програмування. Класична форма навчанняне встигає за сучасними темпами розвитку інтернет технологій. Саме тому збільшується роль навчальних онлайн платформ, що надають найбільш сучасні та необхідні для фахівця даної галузі знання.

Сучасна навчальна онлайн платформа повинна мати актуальну інформацію, бути доступною для всіх, незалежно від достатку, віку та часу. Рішення, що існують на даний момент, не завжди відповідають критеріям, що перелічені вище.

У сучасному програмуванні існує багато web-технологій, серед яких особливо виділяється Node.js, яка швидко розвивається, має відмінну продуктивність та є простою в освоєнні.

Наявні онлайн курси з технології Node.js є або обмеженими за часом, або не мають української локалізації, або не є фінансово доступними. В українському сегменті мережі Інтернет не вистачає актуальних даних з цієї технології.

Розробка навчальної платформи, яка відповідає заданим критеріям та не має перелічених недоліків є актуальною задачею.

Об’єкт досліджень: Клієнт-серверні технології на основі веб-додатку навчальної платформи

Предмет досліджень: Клієнтська частина веб-додатку

Мета дослідження: Розробка клієнтської частини веб-додатку навчальної платформи

Завдання дослідження:

а) Проаналізувати існуючі рішення

б) Провести дослідження технологій і механізмів взаємодії клієнт-серверної архітектури.

в) Розробити клієнтську частину веб-додатка

Методи дослідження: Технології які використовуються для створення веб-додатку.

# РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

## Актуальність навчальних платформ

Знання – головна рушійна сила сучасного суспільства, джерело сили і конкурентних переваг у роботі й житті. Однак традиційна система освіти не може дати студентам знання, яких їм вистачить на все життя. Безперервний процес отримання нових знань - єдиний правильний підхід до освіти у сучасному світі для людини будь-якого віку. Особливо це стосується людей, які обрали спеціальність пов’язану з інформаційними технологіями.

У темпі розвитку сучасних інформаційних технологій самоосвіта є невід’ємною частиною життя будь-якого програміста, який хоче залишатися конкурентоспроможним на ринку праці. Від фахівця завжди вимагатимуть глибоких знань сучасних технологій, розуміння тенденцій їх розвитку та вміння швидко адаптуватися під нові потреби ринку. Часи, коли навчання було лише дитячо-юнацьким етапом в житті людини, закінчуються. Тепер, щоб добитися успіху, потрібно вчитися протягом усього життя.

Технології оновлюються з такою швидкістю, що кожні 3-5 років спеціальність, яка відноситься до інформаційних технологій, може кардинально змінитися. Саме тому збільшується роль навчальних онлайн платформ, що надають найбільш сучасні та необхідні для фахівця даної галузі знання. Подібні навчальні платформи розраховані як на студентів різних рівнів підготовки, так і на досвідчених фахівців, яким треба освіжити пам’ять або вивчити нові технології.

## Існуючі навчальні платформи з Node.js

В мережі Інтернет існує декілька варіантів навчальних онлайн платформ та веб ресурсів, які надають можливість опанувати таку технологію, як Node.js. Серед них можна виділити Codecademy, Udemy та w3schools.org.

**Codecademy** пропонує безкоштовні курси програмування на таких мовах, як Python, JS, PHP, Ruby, HTML і CSS, Java і т.д. Ресурс ідеально підходить для тих, хто бажає ознайомитись з мовами програмування і веб-розробкою. Але всі найскладніші деталі і нюанси доведеться вивчати самостійно. Codecademy також пропонує курс для ознайомлення з Node.js, який може бути доступним після оформлення платної підписки.



Рисунок 1.1. - Курс з Node.js на сайті Codecademy

**Udemy** має безліч різноманітних платних курсів, в тому числі і з Node.js. Курси на зазначеному ресурсі більш схожі на те, що всі звикли називати туторіалами. Величезний плюс цих курсів в тому, що дані курси готують і записують практикуючі фахівці.

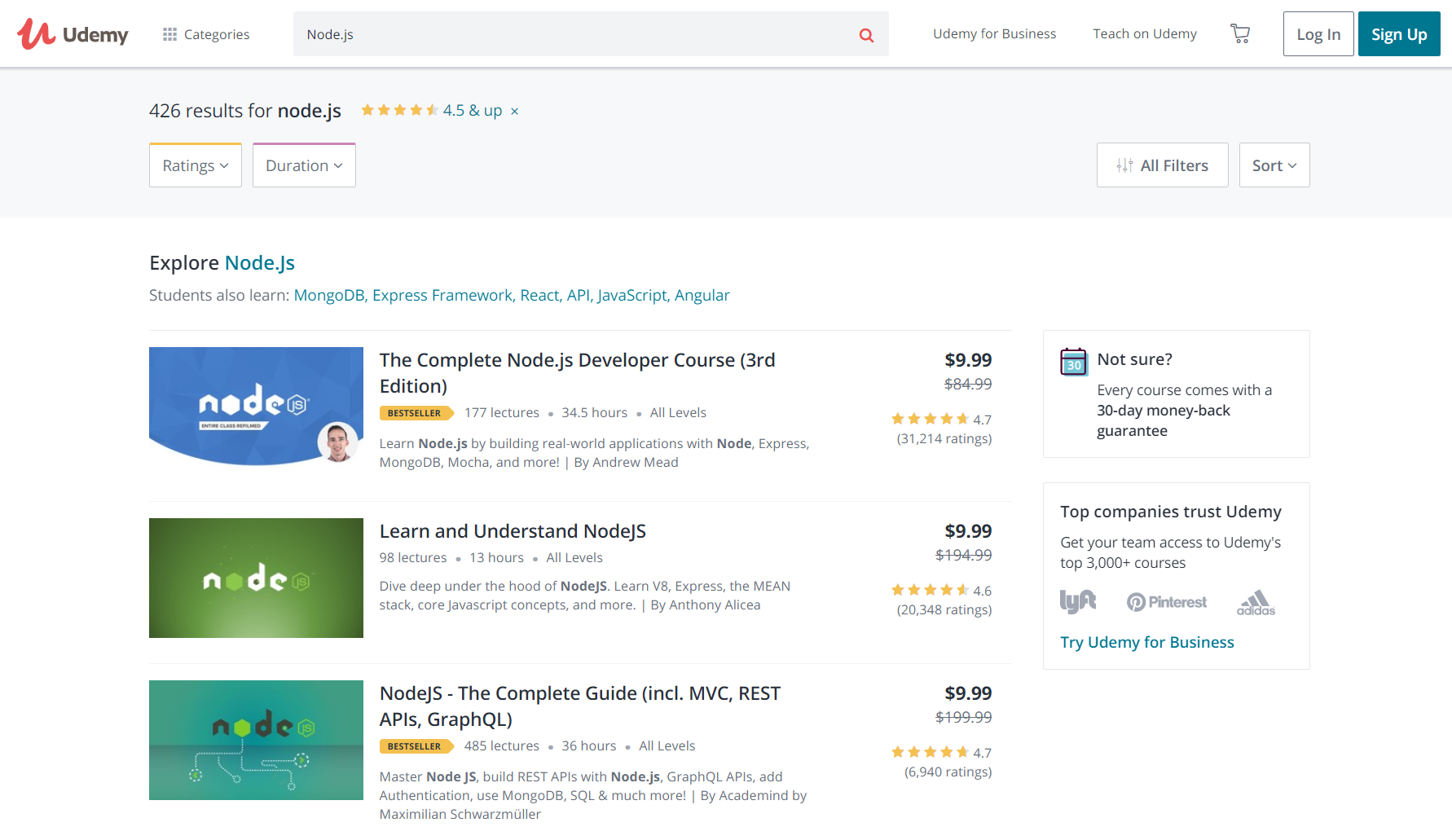


Рисунок 1.2. - Курси з Node.js на сайті Udemy

**W3Schools** це сайт з інструкціями, але вони більш стиснуті і нагадують довідки. Інформація, яка подана в такому вигляді, стане у нагоді якщо буде потрібно швидко щось пригадати.

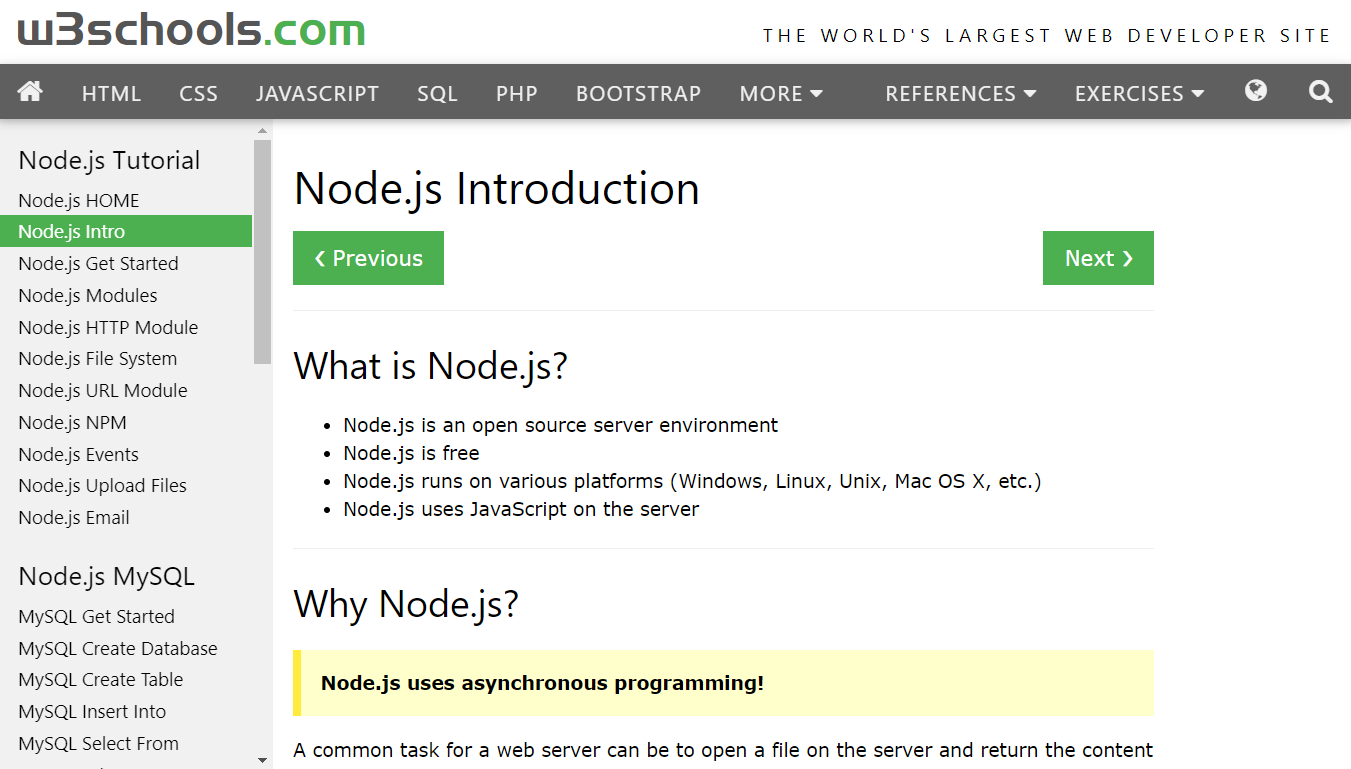


Рисунок 1.3. - Курс з Node.js на сайті w3schools.com

Всі вище перелічені навчальні платформи відносяться до іноземного сегмента Інтернету. Більшість з них платні, або містять обмежену частину курсу, яка доступна безкоштовно. Якщо брати український сегмент, то можна розраховувати тільки на окремі статті призначені для ознайомлення з технологією або для вирішення окремих випадків з якими зіткаються фахівці.

## 1.3 Недоліки сучасних сервісів і шляхи вирішення проблем

В мережі з'являється все більш і більш сервісів спрямованих на онлайн навчання. Але, як було підмічено раніше, наявні онлайн курси з технології Node.js мають декілька вагомих недоліків:

* Не мають української локалізації. Всі курси подані англійською мовою без можливості перейти на українську мову.
* Не є фінансово доступними. Деякі з перелічених платформ пропонують ознайомитись лише з першою частиною курсу, пропонуючи платну підписку для перегляду всього курсу, або пропонують оплатити курс одноразово.
* Мають обмежену інформацію, якої недостатньо для повного вивчення цієї технології. Для освоєння потребується знаходити інформацію на сторонніх ресурсах.

В зв‘язку з тим, що жоден з вище перелічених навчальних платформ не відповідає всім поставленим вимогам, з’являється потреба в створенні нової, яка би включала в себе всі існуючі можливості та задовольняла необхідні потреби.

Для вирішення проблеми сучасних навчальних платформ щодо отримання знань з технології Node.js було прийнято рішення:

* сформувати вимоги до навчальної платформи
* сформувати бізнес-логіку навчальної платформи;
* реалізувати навчальну платформу у вигляді веб-додатку

## 1.4 Формування вимог до навчальної платформи

Сучасна навчальна онлайн платформа повинна відповідати наступним вимогам:

1. Вона повинна бути безкоштовною.
2. Мати українську локалізацію.
3. Поданий матеріал повинен надавати всю необхідну та актуальну інформацію для вивчення обраної технології
4. Дані користувача мають бути надійно збережені.

Ці всі вимоги спрямовані саме на властивості самої навчальної платформи. Однак, під час користування та підтримки веб-додатка з’являються й інші. Їх створюють користувачі та персонал, який займається підтримкою. До таких вимог можна віднести такі:

1. Можливість реєстрації
2. Легкий процес автентифікації
3. Захист від доступу до певного функціоналу неавтентифікованими користувачами
4. Проходження тестів для перевірки вивченого матеріалу
5. Отримання та збереження результатів тестів у кабінеті користувача
6. Адміністратор повинен мати змогу додавати нові тести та видаляти застарілі
7. Адміністратор повинен мати змогу додавати новий матеріал для вивчення

# РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ТЕХНОЛОГІЙ КЛІЄНТСЬКОЇ ЧАСТИНИ

## 2.1 Архітектура клієнт-серверної взаємодії

Веб-додаток - це програма, яка працює у веб-браузері. Подібно до програмного забезпечення, яке працює на настільному комп'ютері, веб-додаток дозволяє взаємодіяти з користувачем і може бути розроблений для різних цілей [1].

Веб-додаток складається з клієнтської і серверної частини, тим самим реалізуючи «клієнт-сервер» архітектуру. Клієнт-серверна модель розподіляє завдання і навантаження між серверами, які постачають ресурси і послуги, та клієнтами, які запитують ці послуги. Як правило вони розташовані на різних обчислювальних машинах, які потребують різне апаратне та програмне забезпечення. Між собою вони обмінюються даними по мережі за допомогою протоколів [2].

Коли користувачі натискають на посилання, заповнюють форму або виконують пошук, клієнт відправляє на сервер запит. Сервер очікує повідомлень із запитами від клієнта, обробляє їх, коли вони приходять і видає відповідь. Відповідь містить код статусу, який показує, чи був запит успішним (наприклад, «200 OK» означає успіх, «404 Not Found», якщо ресурс не може бути знайдений, «403 Forbidden», якщо користувач не має права переглядати ресурс, і т.д.). Тіло успішної відповіді на запит буде містити запитуваний ресурс [3]. Графічно це виглядає так, як показано на Рисунку 2.1.

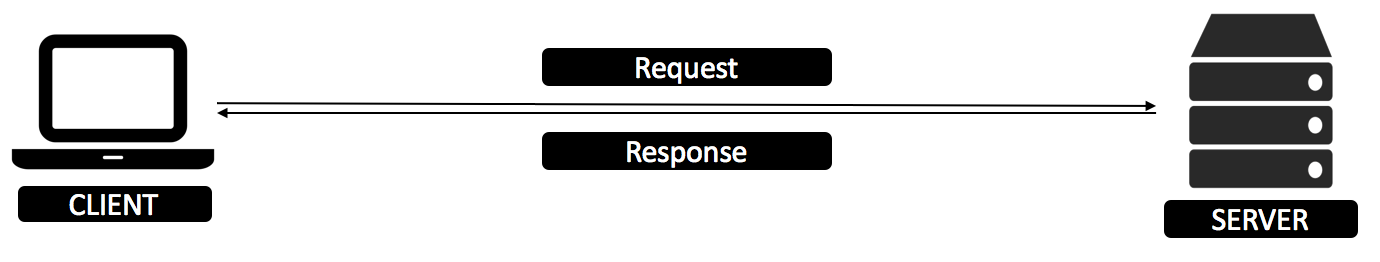


Рисунок 2.1. - Модель взаємодії клієнт-сервер

Перевагою моделі взаємодії клієнт-сервер є те, що програмний код клієнтського і серверного додатка розділений.

На сьогоднішній день існує декілька інтерфейсів комунікації клієнта і сервера. Вони визначаються способом відправлення запиту, форматом переданих даних і форматом одержуваних даних. Найбільш популярними інтерфейсами комунікації клієнт-серверного додатка є SOAP, REST, JSON API [4].

## 2.2 Клієнтська частина веб-додатку

Враховуючи поставлені вимоги до ПЗ, клієнтську частину варто розробляти як окремий компонент. Такий підхід реалізовує певну автономність, що в свою чергу забезпечує більшу швидкодію, стійкість та масштабованість. Окрім переліченого, процес розробки в такому випадку простіше організувати для команди.

Клієнтська частина веб-додатка повинна виконувати наступні функції:

* ініціалізовувати запити до сервера
* обробляти результати запитів, отриманих від сервера
* представляти результати запиту користувачеві у формі інтерфейсу

користувача

Інтерфейс веб-додатку повинен забезпечувати інтуїтивно зрозумілу структуру розміщеної на ньому інформації та мати швидкий і логічний перехід до сторінок. Навігаційні елементи повинні забезпечувати однозначне розуміння користувачів їхнього змісту: посилання на сторінки повинні бути забезпечені заголовками, умовні позначення відповідати загальноприйнятим. Система повинна забезпечувати навігацію по всім доступним користувачеві ресурсам і відображати відповідну інформацію. Також інтерфейс повинен мати єдину концепцію графічного відображення (дизайну).

Проаналізувавши вимоги до навчальної платформи у Розділі 1, було сформовано вимоги до клієнтської частини веб-додатку:

1. Реалізувати інтерфейс сайту, який доступний для всіх і включає в себе:
   1. Головну сторінку, яка містить посилання на весь навчальний матеріал
   2. Сторінку, яка містить інформацію певного розділа
   3. Форму реєстрації користувача у системі
   4. Форму авторизації користувача
2. Реалізувати інтерфейс сайту, який доступний для авторизованих користувачів і включає в себе:
   1. Сторінку зі списком тестів доступних для проходження
   2. Сторінку, яка відображає окремий тест
   3. Кабінет користувача
3. Реалізувати інтерфейс сайту, який доступний тільки для адміністратора і включає в себе:
   1. Форму для додавання навчальних матеріалів
   2. Форму для додавання та видалення тестів

## 2.3 Аналіз та проектування

З метою проектування і аналізу вимог буде потрібно максимально покрити список функціоналу, який розробляється, для того, щоб вірно побудувати пріоритети робіт, вибудувати ітерації в логічному ланцюжку і не повертатися до базових речей.

Розділимо кожне бізнес завдання на складові технічні завдання по реалізації відповідного функціоналу:

1. Головна сторінка повинна містити:
   1. Навігацію, яка доступна на кожній сторінці. В залежності від стану користувача в системі, навігація повинна мати посилання на перехід до реєстрації, авторизації, кабінету користувача, перевірку знань, вихід із системи.
   2. Список посилань на навчальні статті. Щоб отримати список статей клієнту необхідно сформувати запит на отримання всіх статей і відправити його на сервер. Сервер має повернути наступні дані:
      1. Ключ, унікальний для кожної статті
      2. Назву статей
2. Сторінка перегляду статті має містити назву статті та її зміст. Щоб отримати статтю, необхідно сформувати запит на отримання певної статті і відправити його на сервер. Сервер має повернути наступні дані:
   * 1. Унікальний ключ
     2. Назву статті
     3. Уніфікований ідентифікатор ресурсу
3. Форма реєстрації користувача повинна містити:
   1. Поля: username, email, password, confirm password
   2. Кнопку для відправки форми

Для відправки даних о новому користувачеві до сервера, треба сформувати запит на реєстрацію, який повертає тимчасовий ключ доступу.

1. Форма авторизації користувача повинна містити:
   1. Поля: username, password
   2. Кнопку для відправки форми

Для авторизації користувача треба сформувати запит, який передає логін та пароль на перевірку до сервера та повертає тимчасовий ключ доступу при вдалій автентифікації.

1. Сторінка з тестами має містити:
   1. Список тестів. Щоб отримати список тестів клієнту необхідно сформувати запит на отримання всіх тестів і відправити його на сервер. Сервер має повертати наступні дані:
      1. Ключ, унікальний для кожного теста
      2. Назву тестів

В залежності від того, чи є користувач адміністратором, у кожного теста повинна бути кнопка для видалення. Видалення тесту на клієнті відбувається після того, як запит на видалення тесту повертає його унікальний ключ.

1. Сторінка проходження тесту має містити:
   1. Назву теста
   2. Питання з чотирма можливими варіантами відповідей
   3. Кнопку для відправки форми

Для отримання та збереження результату тестування, клієнт відправляє на сервер запит для перевірки відповідей, які обрав користувач.

1. Кабінет має містити:
   1. Інформацію користувача
   2. Результати проходження тестів у вигляді таблиці

В залежності від того, чи є користувач адміністратором, в кабінеті повинні бути посилання на додавання тесту та додавання статті.

1. Форма для додавання теста має містити:
   1. Поля: title, question, answer1, answer2, answer3, correct answer
   2. Кнопку для відправки форми

Щоб додати тест до бази даних треба сформувати запит, який передає назву тесту і питання з варіантами відповідей. Сервер має повертати наступні дані:

* + 1. Унікальний ключ кожного теста
    2. Назву теста
    3. Список питань:
       1. Унікальний ключ кожного питання
       2. Текст питання
       3. Список відповідей:
          1. Унікальний ключ кожної відповіді
          2. Текст відповіді
          3. Правильність відповіді

1. Форма для додавання статті
   1. Поля: title, uri
   2. Кнопку для відправки форми

Щоб додати статтю до бази даних треба сформувати запит, який передає назву статті і уніфікований ідентифікатор ресурсу. Сервер повертає наступні дані:

* + 1. Унікальний ключ статті
    2. Назву статті
    3. Уніфікований ідентифікатор ресурсу

## 2.4 Створення додатку у вигляді SPA

Враховуючи дані вимоги, можна стверджувати, що в якості інтерфейсу варто розробити односторінковий додаток (SPA), також відомий як односторінковий інтерфейс (SPI) - це тип веб-додатку чи веб-сайту, який потребує лише одну сторінку для роботи у браузері. Це забезпечує користувачу досвід близький до користування настільною програмою. За даних умов реалізація серверного рендерингу буде не оптимальним рішенням [5].

Односторінкові веб-додатки будуються навколо концепції динамічного перезапису вмісту однієї сторінки. Це відрізняється від завантаження попередньо візуалізованих сторінок з сервера. Використовуючи цей підхід, веб-додатки з однією сторінкою уникають переривання, викликаного візуалізацією сторінок на сервері.

У традиційному додатку, кожен раз, коли програма викликає сервер, сервер відтворює всю HTML-сторінку. Клієнт отримує візуалізовану сторінку і запускає оновлення сторінки.

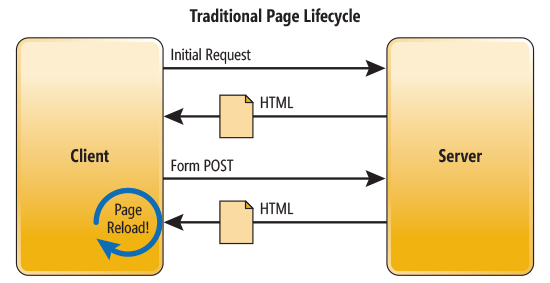


Рисунок 2.2. – Життєвий цикл традиційного додатку

В односторінковому додатку весь необхідний код (HTML, JavaScript, та CSS) завантажується разом із сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Додаток використовує дані JSON для динамічного оновлення сторінки. Однак сторінка ніколи не оновлюється і не перенаправляє користувача на іншу сторінку в процесі роботи з нею. Взаємодія з односторінковим додатком часто включає в себе динамічний зв'язок з веб-сервером [6].

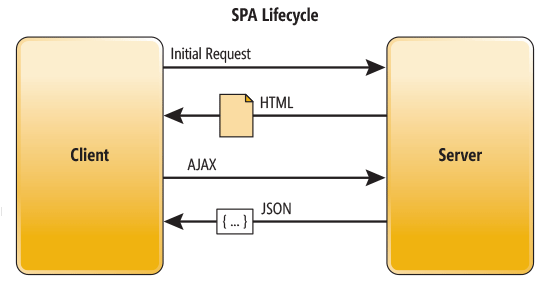


Рисунок 2.3. – Життєвий цикл односторінкового додатку

Розробка такого типу додатка - досить класичне завдання для front-end розробника.

Переваги односторінкових додатків

* Висока швидкість. Оскільки SPA не оновлює всю сторінку, а лише необхідну частину, це значно підвищує швидкість роботи додатку.
* Висока швидкість розробки. Готові бібліотеки та фреймворки забезпечують потужні інструменти та велику кількість готових рішень для розробки веб-програм.
* Мобільні додатки. SPA дозволяє створювати мобільні додатки, оскільки розробник може повторно використати той же код бекенда для веб-додатка та власної мобільної програми.
* SPA може ефективно кешувати будь-які дані. Програма надсилає лише один запит, кешує всі дані, після чого може їх використовувати і навіть працювати у режимі офлайн [7].

## 2.5 Вибір стеку технологій

Стек технологій, використаний при розробці клієнтської частини веб-додатка навчальної платформи, включає в себе фреймворки і бібліотеки, які потребуються для ефективної та швидкої розробки.

### 2.5.1 React

Існує широке коло фреймворків для створення SPA. Якщо подивитися на дати релізів, то більшість з них відносяться до першого покоління SPA: Angular (2010 року), Backbone (2010 року) і Ember (2011 року).

React був випущений Facebook в 2013 році. React - це не SPA-фреймворк, а бібліотека на мові JavaScript, яка призначена для створення інтерфейсів користувача. Це тільки уявлення, буква V в абревіатурі MVC (Model View Controller). Однак ціла екосистема навколо React дозволяє створювати односторінкові додатки [8].

Такі технології, як React і Angular значно полегшили життя розробникам: повторне використання, рефакторинг, поділ на модулі - це лише частина плюсів від використання цих технологій. Але якщо порівнювати популярність їх використання, то React.js буде більш популярним способом створення односторінкових додатків (Рисунок 2.1.).

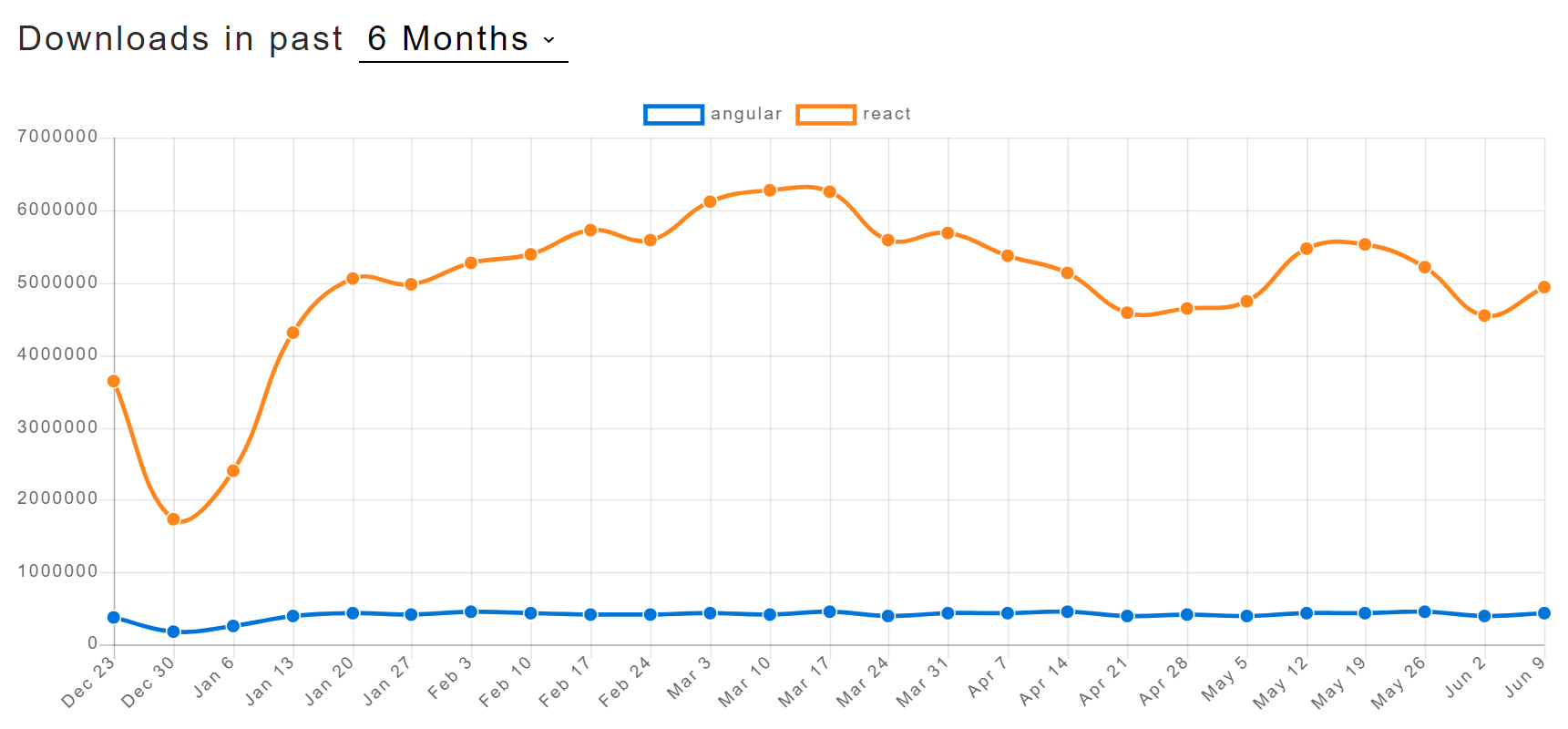


Рисунок 2.4. – Порівняння популярності технологій на сайті npmtrends.com

Таблиця 2.1.

|  |  |  |
| --- | --- | --- |
| **Angular** | **Особливості** | **React** |
| Google | Розробник | Facebook |
| Фреймворк | Тип | Бібліотека |
| Високий | Поріг входу | Низький |
| HTML + TypeScript | Шаблони | JSX + JS (ES5/ES6) |
| Двостороннє | Зв'язування даних | Одностороннє |
| Звичайний | DOM | Віртуальний |
| Висока | Абстракція | Середня |

Порівнюючи ці дві технології (Таблиця 2.1.), було вирішено обрати для розробки клієнтської частини веб-додатка навчальної платформи саме React. Він простіший для вивчення, ніж Angular. До того ж, він надзвичайно швидкий, так як використовує віртуальний DOM і оновлює тільки змінені частини сторінки. Virtual DOM виконує оптимізоване порівняння свого внутрішнього стану з DOM-моделями браузерів і вносить мінімальні зміни, необхідні для збереження узгодженості призначеного для користувача інтерфейсу. Припустимо, якщо потрібно оновити дату народження користувача всередині блоку HTML-тегів, то віртуальний DOM буде оновлювати необхідну частину, порівнюючи відмінності між попередньою і поточною версією HTML, а звичайний DOM - буде оновлювати все дерево HTML-тегів, поки не досягне дати народження. Звернення до DOM досі є самою повільною частиною сучасних веб-додатків, тому дана бібліотека і отримує перевагу в продуктивності, оптимізуючи його.

React надається у вигляді бібліотеки JavaScript з компілятором JSX і пов'язаних з ними інструментів розробки. JSX - це перетворювач і компілятор, що сприймає розширений синтаксис JavaScript. Знайома HTML-подібна форма запису JSX значно спрощує кодування і обслуговування компонентів React. JSX дозволяє розмістити логіку призначеного для користувача інтерфейсу разом з відповідним структурним описом в одному файлі, що допомагає підвищити продуктивність праці і зменшити кількість помилок в великих проектах[10]. При використанні Angular може знадобитися синхронізувати в три рази більше файлів: файли шаблону, обробника коду JavaScript і структурного опису HTML.

React дозволяє описувати елементи, тобто є декларативним. За допомогою даної технології можна легко створювати інтерактивні інтерфейси. Розробник може описати, як різні частини інтерфейсу виглядають у кожному стані додатку і React ефективно оновить та відрендерить лише потрібні компоненти, коли дані зміняться. Декларативні інтерфейси роблять код більш передбачуваним і його набагато легше налагоджувати.

Розробник також може створювати інкапсульовані компоненти, які керують власним станом, а потім з'єднувати їх для подальшого використання і створення складних інтерфейсів користувача. Так як логіка компонентів написана на JavaScript, замість шаблонів, програміст може легко передавати досить великий набір даних зі складною структурою по всьому додатку, зберігаючи стан окремо від DOM [9].

Завдяки використанню даного фреймворка з'являється можливість завантажити клієнту (наприклад, в веб-браузер) відразу всі можливі «Уявлення». Тобто для кожної дії користувача у веб-клієнті знайдеться відповідне графічне веб-уявлення, а з сервера потрібно завантажити лише те, що змінилося в даних (наприклад, якісь числові дані в моніторингу, або будь-яка інша інформація в зручному вигляді, наприклад, JSON). Завдяки такому підходу, при початковому завантаженні сторінки на веб-клієнт завантажується порівняно великий файл (близько 200-300 КБ для великих веб-сервісів), однак при подальших запитах в поточній сесії користувачеві завантажується від 1 Б до ~2-3 КБ (в залежності від розміру змінених даних), при цьому на екрані перемальовується тільки змінена частина, що економить час завантаження необхідного уявлення та Інтернет-ресурси користувача. В результаті спостерігається висока швидкість роботи веб-сервісу, поліпшується чуйність інтерфейсу, прискорюється перехід між сторінками завдяки клієнтському роутингу, а шаблонізація інтерфейсу значно спрощує доопрацювання клієнтської частини веб-додатку.

Роутинг в браузері працює за допомогою DOM API, а саме, window.history. Це API дозволяє дивитися історію відвідування поточної вкладки браузера, додавати нові шляхи в історію вкладки, замінювати поточний шлях, а також переходити назад і вперед по історії. Клієнтський роутинг в цьому випадку можна реалізувати за допомогою бібліотеки react-router [11]. Дана бібліотека дозволяє пов'язувати клієнтські роути з React-компонентами, таким чином, всі можливі стани веб-інтерфейсу будуть міститися в JavaScript-файлі проекту, де оголошені роути (в точці входу веб-додатка).

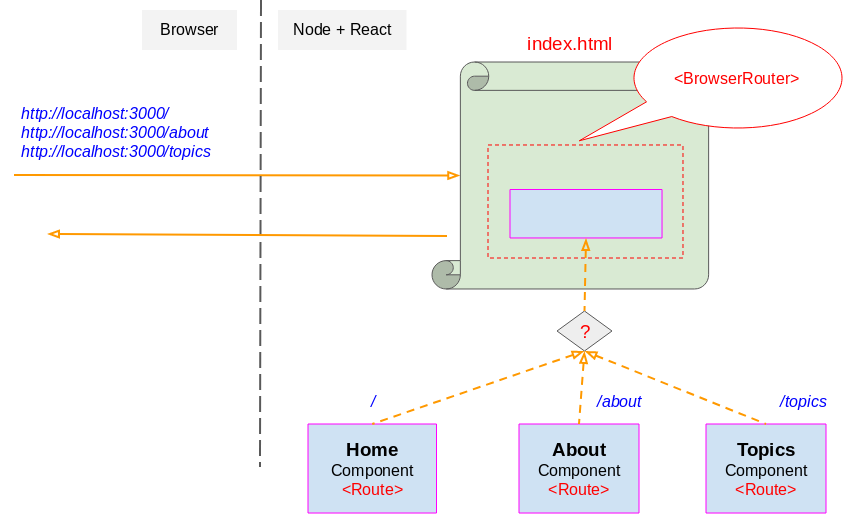


Рисунок 2.5. – Принцип роботи react-router

Бібліотека надає можливість описувати роутинг веб-додатку використовуючи компоненти, HOC і контекст. Працювати з контекстом безпосередньо не потрібно, вся робота буде зроблена компонентами помічниками.

### 2.5.2 GraphQL

Як було зазначено раніше, найбільш популярними інтерфейсами комунікації клієнт-серверного додатка є SOAP, REST, JSON API. Всі ці інтерфейси мають свої переваги і недоліки. Однак в еру розвитку мобільних технологій з'явилась нова вимога: швидкість відображення даних для мобільного інтернету, який часто не завжди має велику швидкість передачі даних. Більш швидкого завантаження можна досягти не тільки збільшенням швидкості передачі даних, але і зменшенням кількості переданих даних і зниженням кількості запитів до сервера. Для вирішення даного завдання була розроблена технологія GraphQL, що надає новий гнучкий інтерфейс комунікації клієнта з сервером.

Технологія GraphQL, розроблена компанією Facebook в 2012 році, є альтернативою REST-архітектурі. Особливість технології GraphQL полягає в тому, що вона дозволяє клієнту визначати структуру даних, яку він хоче отримати [12].

Технологія GraphQL розроблялася для вирішення наступних проблем:

1. Проблема безлічі запитів. Припустимо, з сервера необхідно отримати список всіх користувачів і інформацію про них. У REST-архітектурі для цього буде потрібно виконати один запит для отримання списку всіх користувачів і, в залежності від виконання API сервера, ще один або безліч запитів для отримання інформації про них.
2. Проблема зайвих даних. Припустимо, треба відобразити тільки ім'я і прізвище користувача, а в REST-архітектурі сервер повертає абсолютно всю інформацію про нього: прізвище, ім'я, по батькові, дату народження і т. д.
3. Залежність архітектури від протоколу передачі даних. Архітектура REST прив'язана до HTTP і використовує властивості, характерні тільки для HTTP [13].
4. Незалежність коду клієнта від сервера. Для усунення проблем безлічі запитів і зайвих даних можна створювати нові методи і логіку в програмі. Але для цього треба одночасно змінювати і клієнтський код, і серверний.

GraphQL - це стандарт декларування структури даних і способів отримання даних, який виступає додатковим шаром між клієнтом і сервером. Однією з основних відмінних рис GraphQL є те, що структура і обсяг даних визначається клієнтським додатком [13].

На відміну від стандартного REST-підходу, що має кілька точок входу, що виконують прості завдання, GraphQL містить тільки одну точку входу, яка працює з усіма запитами. Фактично, GraphQL знаходиться між клієнтом і одним або декількома джерелами даних; він приймає запити клієнтів, перенаправляє ці запити за потрібне обробникам і повертає необхідні дані відповідно до переданих інструкціями.

### 2.5.3 Apollo Client

Apollo Client - це найкращий спосіб використовувати GraphQL для створення клієнтських додатків. Клієнт розроблений, щоб допомогти нам швидко створювати інтерфейс користувача, який отримує дані з кінцевої точки GraphQL і піклується про весь цикл запиту [14] (Рисунок 2.2).

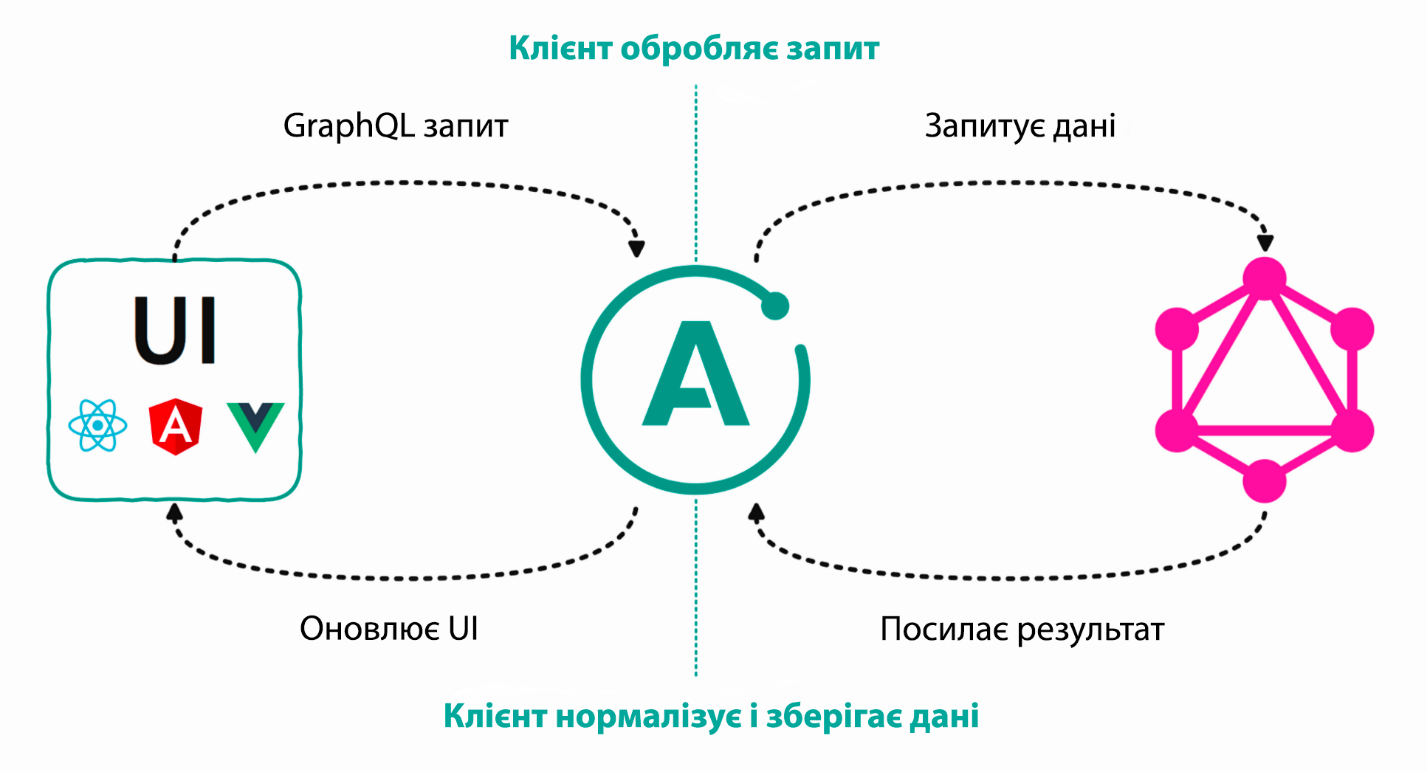


Рисунок 2.6. - Схема, що пояснює потік даних в Apollo Client

1. Інтерфейс користувача (будь-який сучасний інтерфейс) використовує запит GraphQL для спілкування з клієнтом Apollo.
2. Клієнт Apollo використовує запит для запиту даних з будь-якої кінцевої точки (GraphQL-сервер, REST API, мікросервіси,…).
3. API відправляє результат, який клієнт Apollo ловить.
4. Клієнт Apollo посилає результат назад в інтерфейс користувача, щоб він міг оновлюватись відповідно.

Apollo Сlient пропонує лаконічний спосіб роботи з даними у веб-додатках на базі React. У більшості випадків все те, що ми звикли робити за допомогою Redux, набагато простіше зробити за допомогою Apollo Client. Незважаючи на те, що redux це фантастична бібліотека, вона також приносить багато шаблонів коду та речей, які потрібно налаштовувати вручну. Однак це не означає, що клієнт Apollo повинен замінити Redux.

Apollo зробили багато чудових інструментів і технологій для роботи в зв'язці з GraphQL. Звичайно ж вони не могли пройти повз популярної екосистеми React і розробили свій інструмент react-apollo.

Найпростіший спосіб розпочати роботу з Apollo Client - це використання стартового комплекту apollo-boost для швидкого налаштування [15]. Він включає в себе пакети, які є необхідними для створення додатку. І хоча кінцева точка сервера GraphQL є єдиним параметром конфігурації, який потрібен для початку роботи, до стартового комплекту також включені пакети, які дозволяють швидко реалізувати такі функції, як локальне керування станом, автентифікація, та обробка помилок.

### 2.5.4 Reactstrap

Ця бібліотека містить компоненти React Bootstrap 4, які надають перевагу композиції та контролю. Бібліотека не залежить від jQuery або Bootstrap JavaScript [16].

Reactstrap забезпечує попередньо побудовані компоненти, які забезпечують велику гнучкість і готову валідацію. Це дозволяє швидко створювати красиві форми, які гарантовано будуть забезпечувати інтуїтивно зрозумілий досвід користувачу.

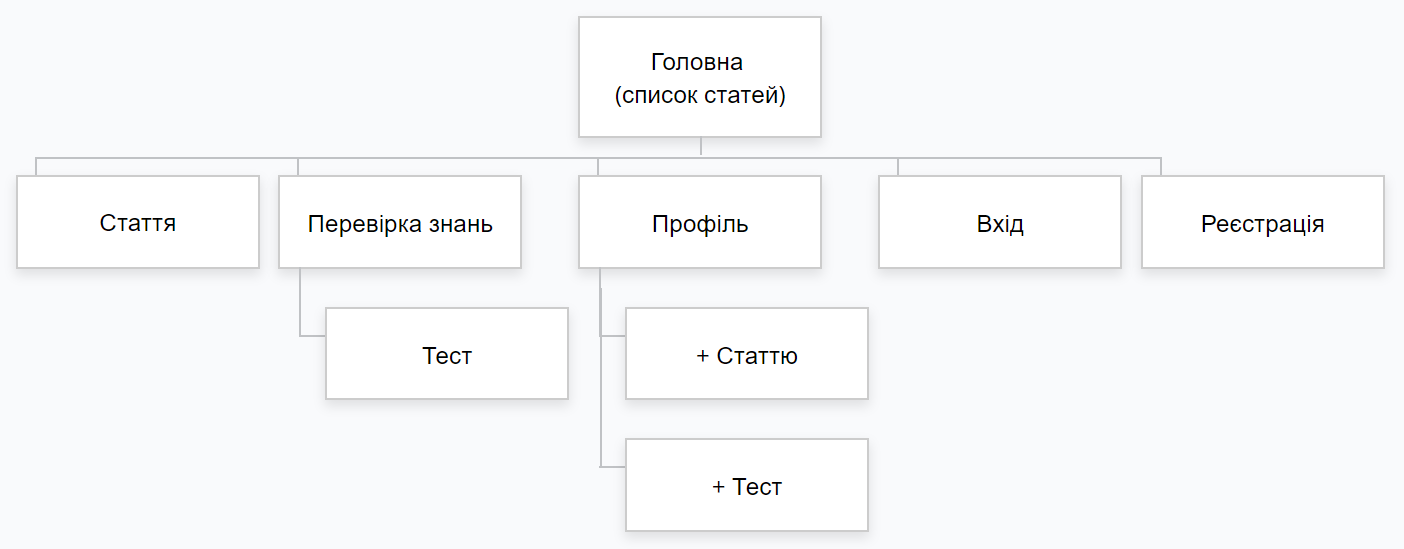
Reactstrap має компонент Form, який включає в себе все від стандартного вводу тексту до більш просунутого завантаження файлів. У створенні інтерфейсу користувача для навчального проекту, цей компонент буде використовуватись найбільше.

# РОЗДІЛ 3. РЕЗУЛЬТАТ РОБОТИ

Головним завданням було створення зручного та зрозумілого інтерфейсу, який би задовільнив всі функціональні вимоги навчальної платформи «NodeUA». Для цього були розроблені такі компоненти:

1. Компонент, який відображає список статей (ChapterList.js)
2. Компонент для відображення певної статті (ChapterPage.js)
3. Компонент реєстрації користувача у системі (SignupUser.js)
4. Компонент авторизації користувача (SigninUser.js)
5. Компонент, який відображає список тестів (TestList.js)
6. Компонент для відображення тесту (TestPage.js)
7. Профіль користувача (Profile.js)
8. Форма для додавання навчальних матеріалів (AddChapter.js)
9. Форма для додавання та видалення тестів (AddTest.js)

Перед розробкою проекту була створена карта веб-додатку:



## Рисунок 3.1. – Карта сайту

Розробка проводилась у Visual Studio Code, безкоштовному засобі для створення та редагування сучасних [веб-застосунків](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) (Рисунок 3.2).

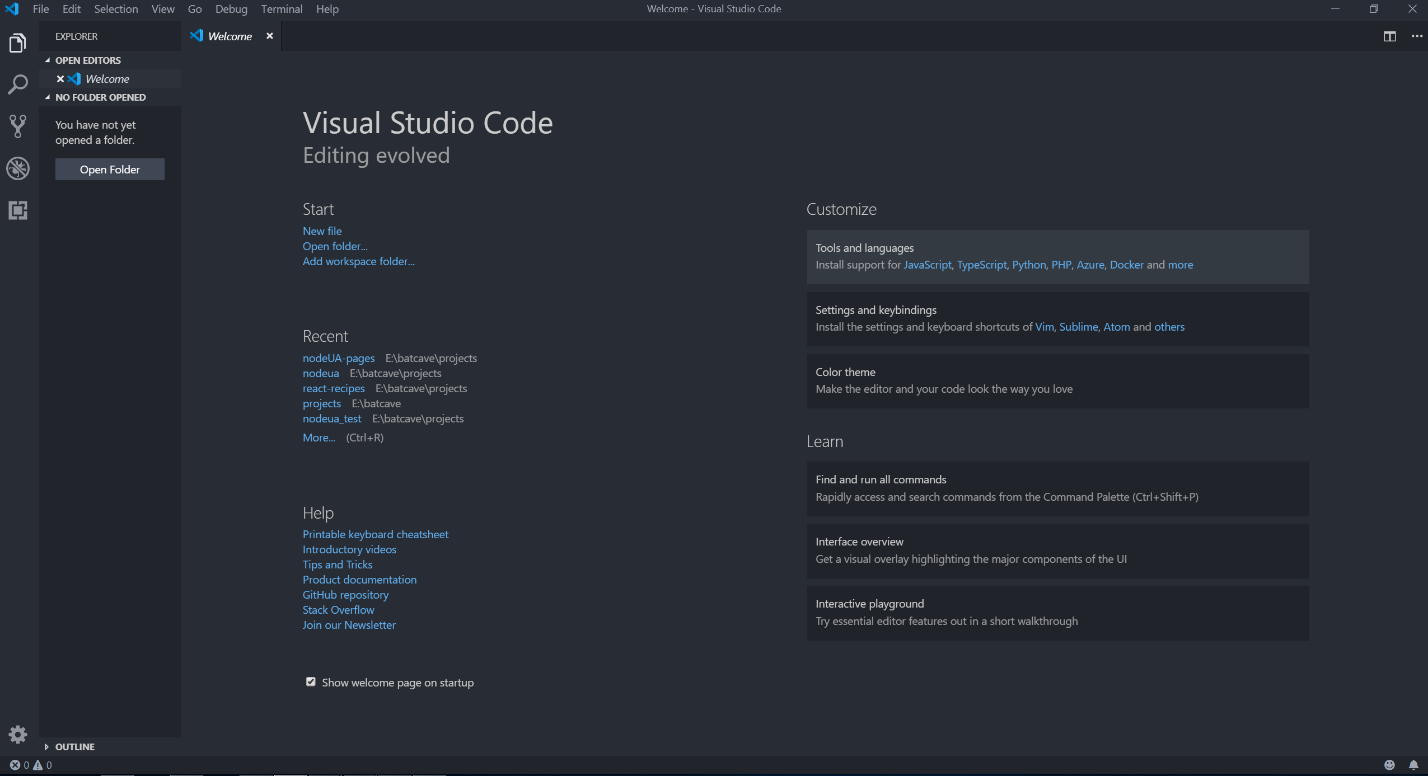


Рисунок 3.2 – Visual Studio Code

В якості системи контролю версій була обрана система Git [17], яка є поширеною серед веб-розробників, а також має зручний графічний інтерфейс і функціональну командну консоль. Github [18] був обраний як сервер для системи контролю версій так як даний сервер дозволяє безкоштовно зберігати приватні репозиторії коду, а також зручно об'єднувати репозиторії в проекти і створювати команди розробників у веб-інтерфейсі (Рисунок 3.3.).

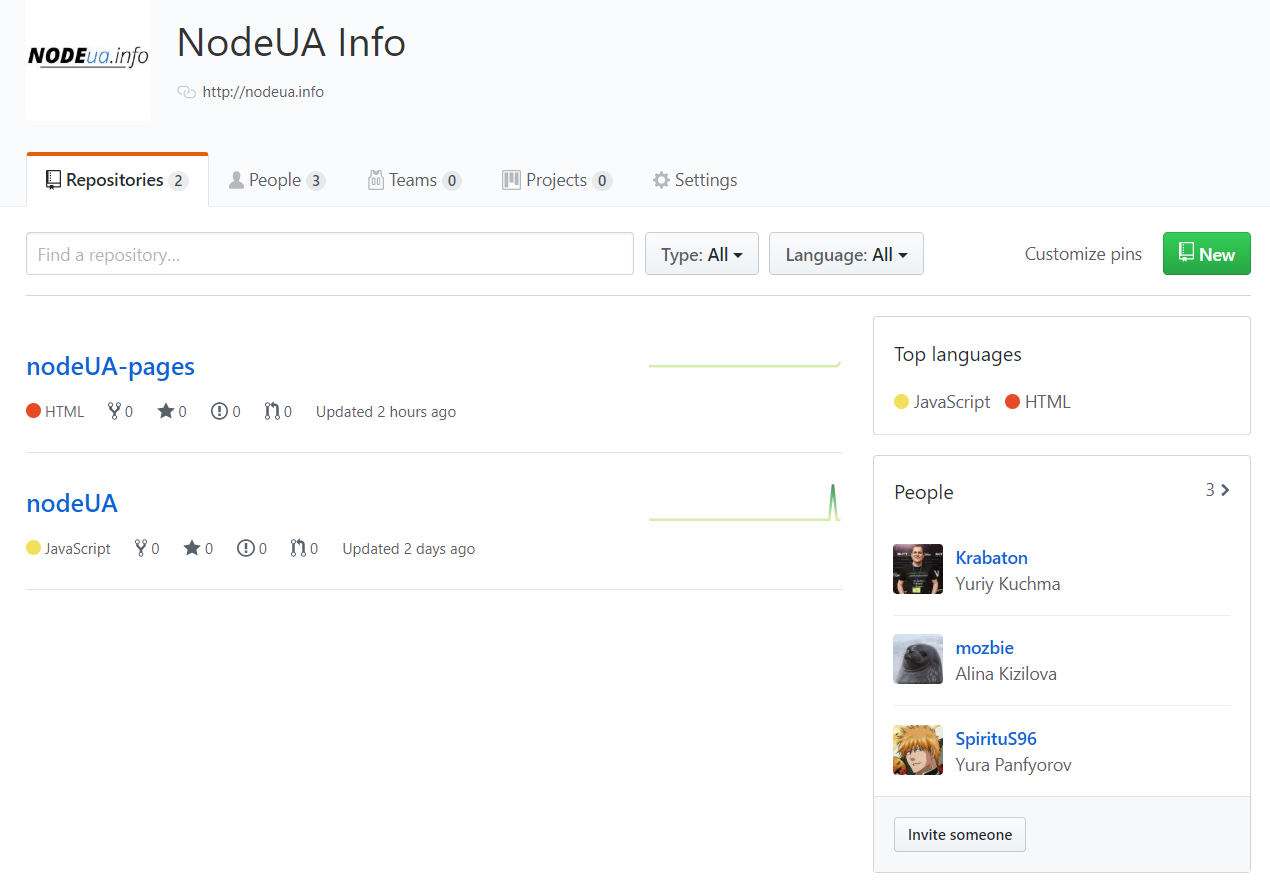


Рисунок 3.3. – NodeUA організація на Github

Для початку розробки клієнтської частини веб-додатку навчальної платформи було вирішено використовувати create-react-app. Create React App - відмінний інструмент для створення базового шаблону стандартних React-додатків. Він економить час на налаштування Webpack, Babel і інших звичних інструментів. Вони заздалегідь налаштовані і заховані, так що можна сфокусуватися на коді і бізнес-логіці програми.

Для створення нового проекту в терміналі виконується наступна команда:

npx create-react-app nodeua

Структура всієї клієнтської частини зображена на Рисунку 3.4.

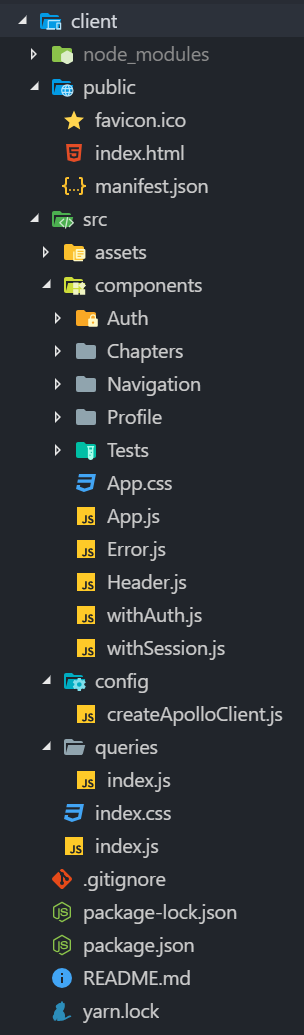


Рисунок 3.4. – Структура проекту

Коренева директорія клієнтської частини містить каталог src, в якому було створено наступні каталоги:

* assets – містить всі медіа файли
* components – містить всі компоненти
* config – містить файл createApolloClient.js
* queries – містить файл index.js з усіма GraphQL запитами та мутаціями (додаток 2)

Файл createApolloClient.js містить наступний код:

import ApolloClient from "apollo-boost";

**const** client = **new** ApolloClient({

uri: "http://localhost:4444/graphql",

fetchOptions: {

credentials: "include"

},

request: operation => {

**const** token = localStorage.getItem("token");

operation.setContext({

headers: {

authorization: token

}

});

},

onError: ({ networkError }) => {

**if** (networkError) {

console.log("Network Error", networkError);

**if** (networkError.statusCode === 401) {

localStorage.removeItem("token");

}

}

}

});

export **default** client;

У цьому файлі відбувається ініціалізація Apollo Client і отримання тимчасового ключа доступу з localStorage.

Клієнтський роутинг описано у кореневому файлі index.js за допомогою бібліотеки react-router:

**const** Root = ({ refetch, session }) => (

<Router>

<Fragment>

<Navigation session={session} />

<Switch>

<Route path="/" exact component={App} />

<Route path="/signin" render={() => <SignIn refetch={refetch} />} />

<Route path="/signup" render={() => <SignUp refetch={refetch} />} />

<Route path="/test/add" render={() => <AddTest refetch={refetch} />} />

<Route path="/chapter/add" render={() => <AddChapter refetch={refetch} />} />

<Route path="/chapters/:\_id" component={ChapterPage} />

<Route path="/profile" render={() => <Profile session={session} />} />

<Route path="/tests" render={() => <TestList session={session} />} />

<Route path="/test/:\_id" render={() => <TestPage session={session} />} />

<Redirect to="/" />

</Switch>

</Fragment>

</Router>

);

Перейдемо безпосереднього до візуального опису функціоналу закладеного в інтерфейс користувача, і приведемо демонстрацію основних елементів з їх описом.

Головна сторінка зі списком статей (Рисунок 3.5):

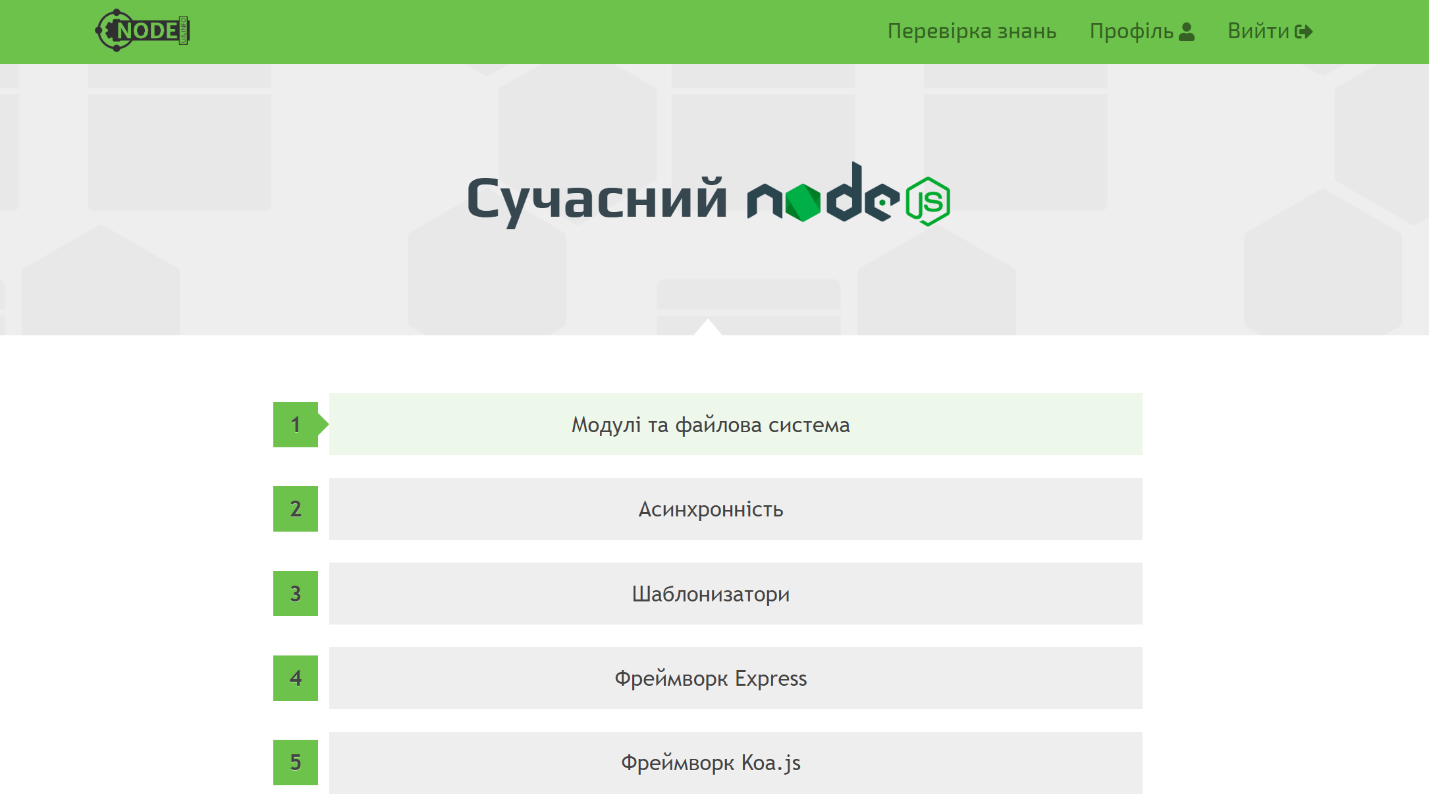


Рисунок 3.5. – Головна сторінка NodeUA

Для отримання всіх статей був створений запит, який вказує, які дані необхідно отримати, використовуючи декларативну, графоподібну структуру, що нагадує формат JSON.

export **const** GET\_CHAPTERS = gql`

query {

getChapters {

\_id

title

}

}

`;

Для відображення цього списку необхідно було отримати унікальний ключ та назву тесту. Щоб отримані дані були доступні компоненту ChapterList потрібно обгорнути його у Query компонент і передати через query prop відповідний запит. Ми також повинні надати функцію до нашого компонента Query, який покаже React, що ми хочемо зробити. Ми можемо використовувати властивості, які надає нам компонент Query, щоб відображати різні інтерфейси в залежності від стану нашого запиту. Наш готовий компонент виглядає наступним чином:

import React from "react";

import { Query } from "react-apollo";

import { GET\_CHAPTERS } from "../../queries";

import Chapter from "./Chapter";

**const** ChapterList = () => {

**return** (

<Query query={GET\_CHAPTERS}>

{({ loading, data, error }) => {

**if** (loading) **return** <div>Loading</div>;

**if** (error) **return** <div>Error</div>;

console.log(data);

**if** (!loading && data.getChapters.length > 0) {

**return** (

<div className="chapter\_list\_\_container">

<ol className="rectangle-list">

{data.getChapters.map(chapter => (

<Chapter key={chapter.\_id} {...chapter} />

))}

</ol>

</div>

);

} **else** {

**return** (

<div>

<h3>No chapters available</h3>

</div>

)}

}}

</Query>

);

};

export **default** ChapterList;

Сторінка, яка відображає певну статтю (Рисунок 3.6.):

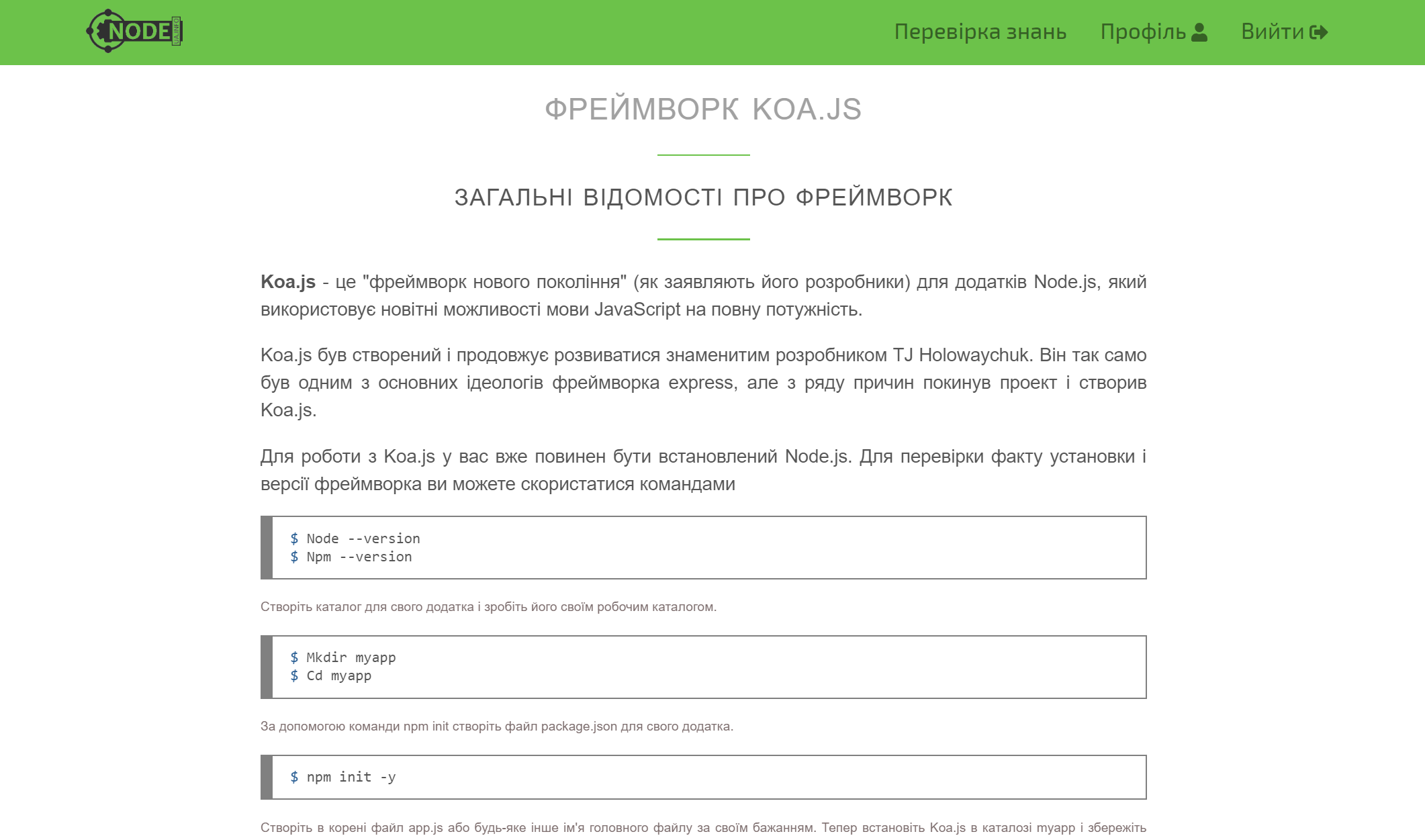


Рисунок 3.6. – Сторінка відображення статті

Крім отримання даних, GraphQL підтримує функції їх створення і зміни. Для цього існують обробники, звані мутаціями. Принцип їх роботи схожий на розглянутий вище, тільки дані не зчитуються, а оновлюються.

Форма реєстрації користувача (Рисунок 3.7):

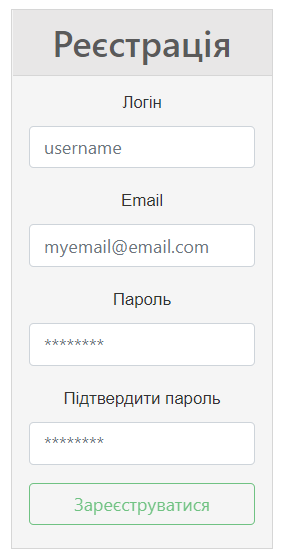


Рисунок 3.7. – Форма реєстрації

Для реєстрації нового користувача, треба сформувати мутацію SIGNUP\_USER, яка приймає у вигляді аргументів логін, пароль та електрону адресу:

export **const** SIGNUP\_USER = gql`

mutation($username: String!, $email: String!, $password: String!) {

signupUser(username: $username, email: $email, password: $password) {

token

}

}

`;

Мутація повертає тимчасовий ключ доступу.

Мутація у компоненті SignupUser має наступний вигляд:

<Mutation mutation={SIGNUP\_USER} variables={{ username, email, password }}>

{(signupUser, { data, loading, error }) => {

**return** (

<Form

className="form"

onSubmit={event => **this**.handleSubmit(event, signupUser)}

>

<Col>

<FormGroup>

<Label>Логін</Label>

<Input

type="text"

name="username"

placeholder="username"

value={username}

onChange={**this**.handleChange}

/>

</FormGroup>

</Col>

<Col>

<FormGroup>

<Label>Email</Label>

<Input

type="email"

name="email"

placeholder="myemail@email.com"

value={email}

*// valid={emailState === "has-success"}*

invalid={emailState === "has-danger"}

onChange={e => {

**this**.validateEmail(e);

**this**.handleChange(e);

}}

/>

</FormGroup>

</Col>

<Col>

<FormGroup>

<Label **for**="examplePassword">Пароль</Label>

<Input

type="password"

name="password"

placeholder="\*\*\*\*\*\*\*\*"

value={password}

onChange={**this**.handleChange}

/>

</FormGroup>

</Col>

<Col>

<FormGroup>

<Label **for**="examplePassword">Підтвердити пароль</Label>

<Input

type="password"

name="passwordConfirmation"

placeholder="\*\*\*\*\*\*\*\*"

value={passwordConfirmation}

onChange={**this**.handleChange}

/>

</FormGroup>

</Col>

<Button

className="form\_btn"

outline

color="success"

disabled={loading || **this**.validateForm()}

>

Зареєструватися

</Button>

{error && <Error error={error} />}

</Form>

);

}}

</Mutation>

Форма авторизації користувача (Рисунок 3.8):

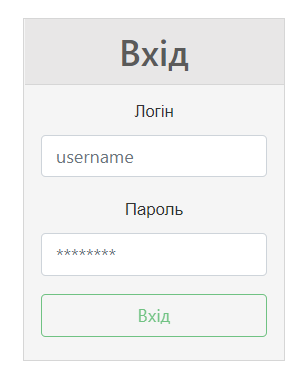


Рисунок 3.8. – Форма авторизації

Формування запиту і використання отриманих даних відбувається так само, як і у випадку з реєстрацією користувача, але тепер в аргументах мутації ми передаємо тільки логін та пароль для автентифікації на сервері.

Реалізацію виходу з системи було описано у SignoutUser компоненті:

import React from "react";

import { withRouter } from "react-router-dom";

import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

import { ApolloConsumer } from "react-apollo";

**const** handleSignOut = (client, history) => {

localStorage.removeItem("token");

client.resetStore();

history.push("/");

};

**const** SignOut = ({ history }) => (

<ApolloConsumer>

{client => {

**return** (

<span onClick={() => handleSignOut(client, history)}>

Вийти <FontAwesomeIcon className="awesome\_icon" icon="sign-out-alt" />

</span>

);

}}

</ApolloConsumer>

);

export **default** withRouter(SignOut);

Оскільки сторінки «Профіль» і «Перевірка знань» доступні лише авторизованим користувачам, був створений додатковий HOC withSession. За допомогою GET\_CURRENT\_USER запиту, компонент withSession отримує всі дані поточного користувача:

import React from "react";

import { Query } from "react-apollo";

import { GET\_CURRENT\_USER } from "../queries";

**const** withSession = Component => props => (

<Query query={GET\_CURRENT\_USER}>

{({ data, loading, refetch }) => {

**if** (loading) **return** **null**;

*// console.log(data);*

**return** <Component {...props} refetch={refetch} session={data} />;

}}

</Query>

);

export **default** withSession;

Від стану користувача у системі залежить те, яка навігація буде відображена користувачу (Рисунки 3.9, 3.10). У Navigation.js:

{session && session.getCurrentUser ? (

<NavbarAuth />

) : (

<NavbarUnAuth />

)}

 Рисунок 3.9. – Навігація для неавторизованого користувача

Рисунок 3.10. – Навігація для авторизованого користувача

Сторінка, яка відображає всі тести (Рисунок 3.11):

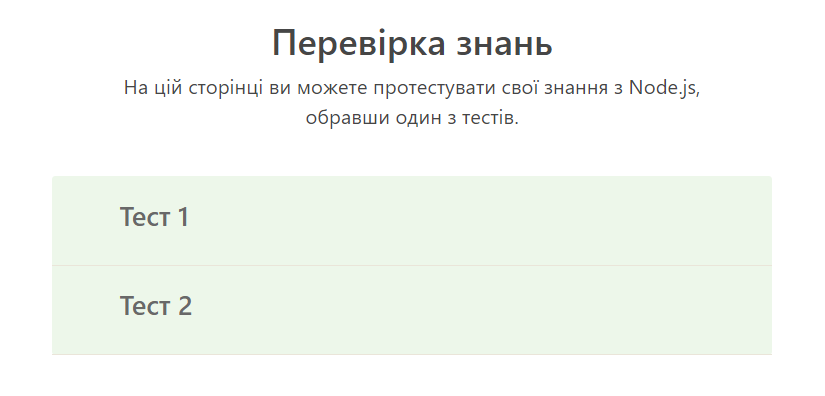


Рисунок 3.11. – Список тестів на сторінці «Перевірка знань»

Формування запиту і використання отриманих даних відбувається так само, як і при отриманні всіх статей для виводу на головну сторінку.

Якщо поточний користувач є адміністратором, то у нього з’являється можливість видалити тест:

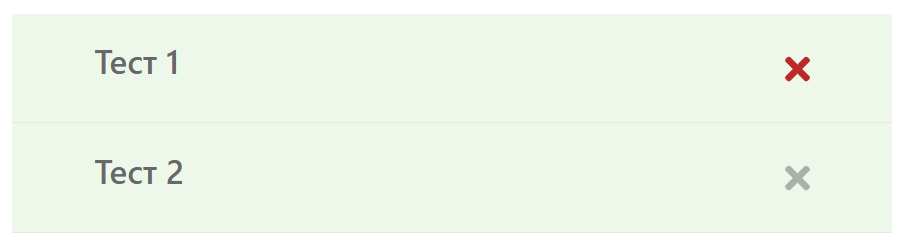


Рисунок 3.12. – Список тестів адміністратора

Видалення тесту відбувається за допомогою мутації DELETE\_TEST:

export **const** DELETE\_TEST = gql`

mutation($\_id: ID!) {

deleteTest(\_id: $\_id) {

\_id

}

}

`;

У компоненті Test.js робимо перевірку і відображаємо кнопку для видалення:

<div>

{session.getCurrentUser.role === "admin" ? (

<Mutation

mutation={DELETE\_TEST}

variables={{ \_id }}

key={\_id}

update={(cache, { data: { deleteTest } }) => {

**const** { getTests } = cache.readQuery({

query: GET\_TESTS

});

cache.writeQuery({

query: GET\_TESTS,

data: {

getTests: getTests.filter(test => test.\_id !== deleteTest.\_id)

}

});

}}

>

{deleteTest => {

**return** (

<span

className="delete-button"

onClick={() => handleDelete(deleteTest)}

>

<FontAwesomeIcon icon="times" />

</span>

);

}}

</Mutation>

) : **null**}

</div>

Сторінка, яка відображає тест для його проходження (Рисунок 3.13.):

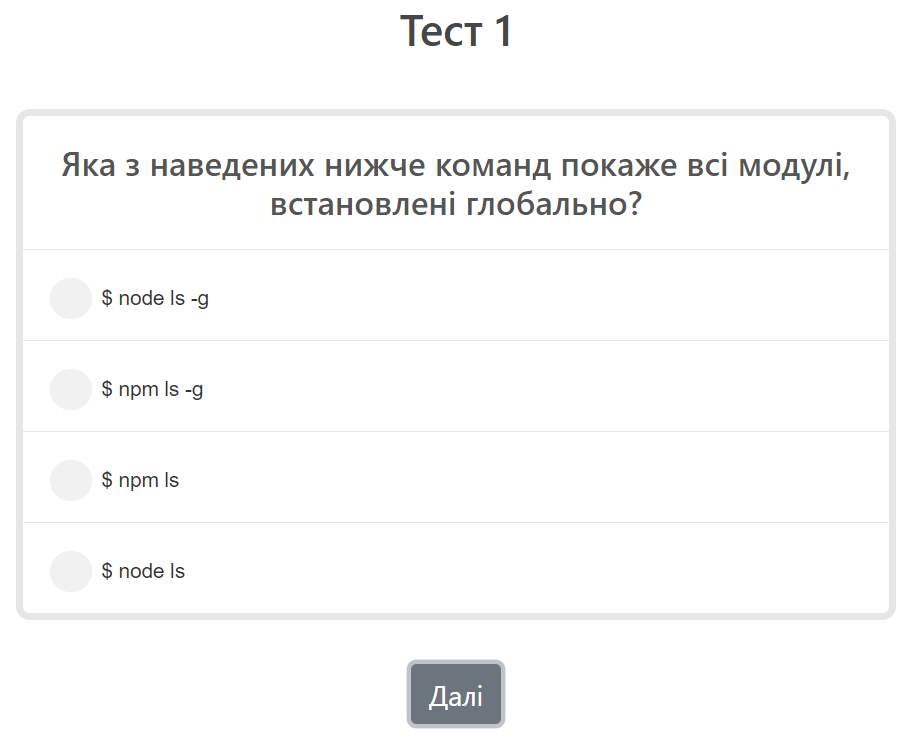


Рисунок 3.13. – Сторінка з тестом

Щоб отримати результати проходження тесту, була створена мутація, яка відправляє масив з питаннями та обраними користувачем відповідями на сервер для перевірки:

export **const** CHECK\_TEST = gql`

mutation($\_id: ID!, $questions: [QuestionInput]!, $title: String!) {

checkTest(\_id: $\_id, questions: $questions, title: $title) {

results

}

}

`;

Коли сервер повертає масив з правильними і неправильними відповідями, клієнт обробляє інформацію і відображає результат користувачу, як показано на Рисунку 3.14.

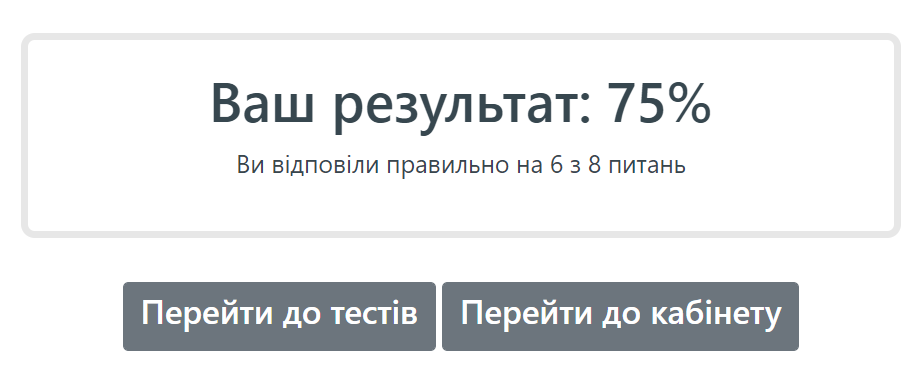


Рисунок 3.14. – Відображення результату

Сторінка профіля містить результати тестування у вигляді таблиці:

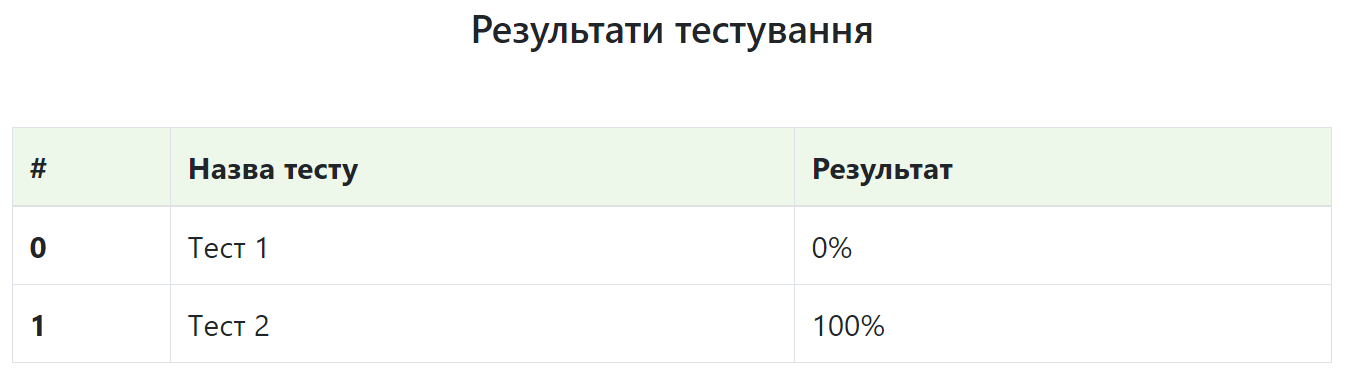


Рисунок 3.15 – Результати тестування

Щоб отримати результати проходження користувачем тестів, було використано компонент withSession, який надає інформацію поточного користувача.

**const** Profile = ({ session }) => {

**const** { testResults } = session.getCurrentUser;

**return** (

<div className="profile\_container">

<h3 className="profile\_results\_title">Результати тестування</h3>

<Table bordered hover responsive className="table\_result">

<thead>

<tr>

<th>#</th>

<th>Назва тесту</th>

<th>Результат</th>

</tr>

</thead>

<tbody>

{testResults.map((result, index) => (

<tr key={index}>

<th scope="row">{index}</th>

<td>{result.testName}</td>

<td>{result.score}%</td>

</tr>

))}

</tbody>

</Table>

{!testResults.length && (

<p className="profile\_attention">У вас поки що немає результатів.</p>

)}

</div>

);

};

Якщо поточний користувач є адміністратором, у його профілі наявні посилання на додавання статті та на додавання тесту:

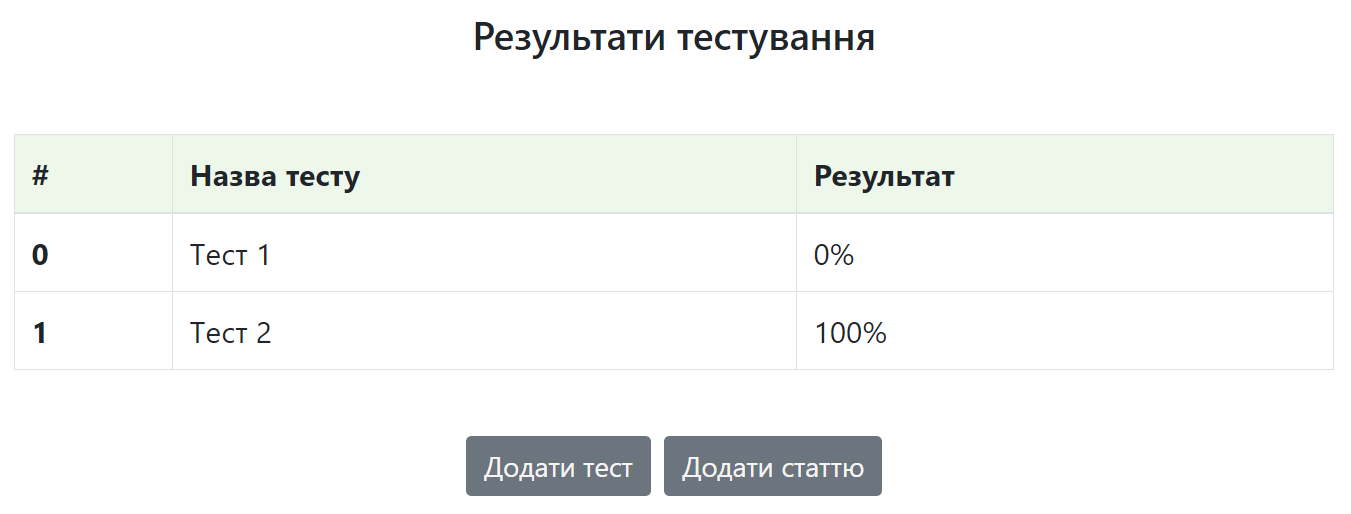


Рисунок 3.16. – Профіль адміністратора

Цього можна досягти виконавши просту перевірку:

{session.getCurrentUser.role === "admin" ? (

<div className="admin\_links">

<Button className="btn\_admin">

<Link className="admin\_link" to="/test/add">

Додати тест

</Link>

</Button>

<Button>

<Link className="admin\_link" to="/chapter/add">

Додати статтю

</Link>

</Button>

</div>

) : **null**}

Форма на додавання статті має наступний вигляд:

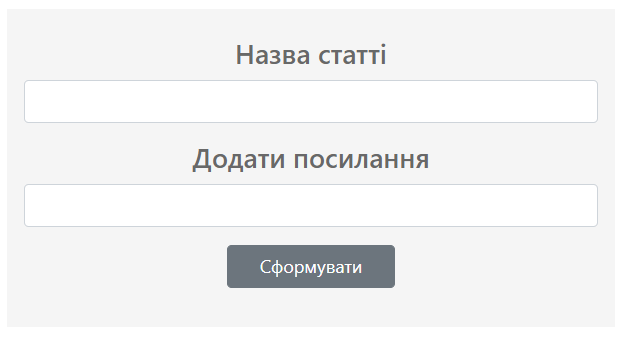


Рисунок 3.17. - Форма на додавання статті

Мутація на додавання статті передає назву статті та уніфікований індентифікатор ресурсу:

export **const** ADD\_CHAPTER = gql`

mutation($title: String!, $uri: String!) {

addChapter(title: $title, uri: $uri) {

\_id

title

uri

}

}

`;

Компонент AddChapter має наступний вигляд:

import React, { Component } from "react";

import { withRouter } from "react-router-dom";

import withAuth from "../withAuth";

import { Container, Button, Form, FormGroup, Input } from "reactstrap";

import { Mutation } from "react-apollo";

import { ADD\_CHAPTER, GET\_CHAPTERS } from "../../queries";

import Error from "../Error";

**const** initialState = {

title: "",

uri: ""

};

class AddChapter extends Component {

state = { ...initialState };

clearState = () => {

**this**.setState({ ...initialState });

};

handleChange = event => {

**const** { name, value } = event.target;

**this**.setState({ [name]: value });

};

handleSubmit = (e, addChapter) => {

e.preventDefault();

addChapter().then(({ data }) => {

console.log(data);

});

**this**.props.history.push("/");

};

updateCache = (cache, { data: { addChapter } }) => {

**const** { getChapters } = cache.readQuery({ query: GET\_CHAPTERS });

cache.writeQuery({

query: GET\_CHAPTERS,

data: {

getChapters: [...getChapters, addChapter]

}

});

};

render() {

**const** { title, uri } = **this**.state;

**return** (

<Mutation

mutation={ADD\_CHAPTER}

variables={{ uri, title }}

update={**this**.updateCache}

>

{(addChapter, { data, loading, error }) => {

**return** (

<div className="App">

<Container className="form\_container\_test">

<Form

className="form\_test"

onSubmit={e => **this**.handleSubmit(e, addChapter)}

>

<FormGroup>

<h4 className="test\_title">Назва статті</h4>

<Input

type="text"

name="title"

onChange={**this**.handleChange}

value={title}

/>

</FormGroup>

<FormGroup>

<h4 className="test\_title">Додати посилання</h4>

<Input

type="text"

name="uri"

onChange={**this**.handleChange}

value={uri}

/>

</FormGroup>

<Button disabled={loading} type="submit" className="test\_btn">

Сформувати

</Button>

{error && <Error error={error} />}

</Form>

</Container>

</div>

);

}}

</Mutation>

);

}

}

export **default** withAuth(

session => session && session.getCurrentUser.role === "admin"

)(withRouter(AddChapter));

Для того щоб після додавання статті відбулась переадресація на головну сторінку, було використано withRouter компонент з бібліотеки react-router. Компонент withAuth, в даному випадку, дає дозвіл на перехід до цієї сторінки тільки адміністратору.

Компонент withAuth має наступний вигляд:

import React from "react";

import { Query } from "react-apollo";

import { Redirect } from "react-router-dom";

import { GET\_CURRENT\_USER } from "../queries";

**const** withAuth = conditionFunc => Component => props => (

<Query query={GET\_CURRENT\_USER}>

{({ data, loading }) => {

**if** (loading) **return** **null**;

**return** conditionFunc(data) ? (

<Component {...props} />

) : (

<Redirect to="/" />

);

}}

</Query>

);

export **default** withAuth;

Щоб на головній сторінці, після переадресації, ми побачили оновлений список статей, було використано Optimistic UI, який описано функцією updateCache.

Optimistic UI - це шаблон, який можна використовувати для моделювання результатів мутації та оновлення інтерфейсу користувача ще до отримання відповіді від сервера. Після того, як відповідь отримано від сервера, оптимістичний результат видаляється і замінюється фактичним результатом.

Форма на додавання тесту має наступний вигляд:

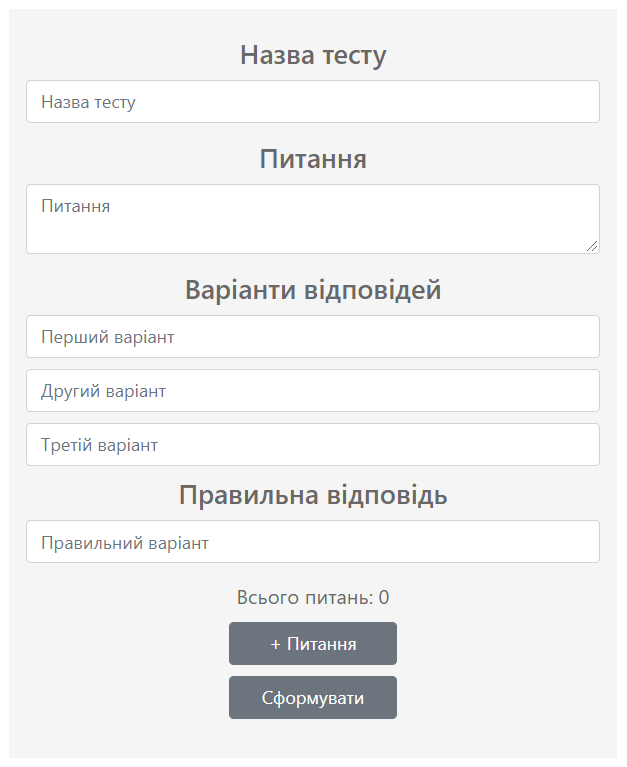


Рисунок 3.18. - Форма на додавання тесту

Для того щоб додати тест потрібно виконати мутацію ADD\_TEST, яка передає масив сформованих питань з відповідями та назву тесту:

export **const** ADD\_TEST = gql`

mutation($questions: [QuestionInput]!, $title: String!) {

addTest(questions: $questions, title: $title) {

\_id

title

questions {

\_id

questionText

answers {

\_id

answerText

isValid

isChecked

}

}

}

}

`;

Компонент AddTest буде мати структуру аналогічну AddChapter. Він так само буде переадресовувати користувача за допомогою компонента withRouter, оновлювати список тестів на сторінці «Перевірка знань» за допомогою Optimistic UI та захищати сторінку від звичайних і неавторизованих користувачів за допомогою withAuth.

# ВИСНОВКИ

Дипломним проектом в якості предмета дослідження виступав безпосередній процес розробки клієнтської частини веб-додатка навчальної платформи. В роботі було детально розглянуто процес розробки клієнтської частини проекту. Для початка були виявлені проблеми і бізнес завдання, що стоять перед клієнтом, після цього були узгоджені вимоги, що висуваються до майбутньої реалізації клієнтської частини веб-додатка, і вже на підставі проведеної аналітики було визначено модель проектування і вибір найбільш доступних і оптимальних технологій та інструментів з урахуванням специфіки проекту.

В результаті досліджень, в якості засобу створення SPA було обрано React.js, а для взаємодії з сервером - GraphQL, так як вони найкраще підходять для розробки клієнтської частини, що відповідала б поставленим вимогам.

В якості результатів всіх досліджень була розроблена клієнтська частина веб-додатку навчальної платформи.

# СПИСОК ЛІТЕРАТУРИ

* + - 1. Web application [Електронний ресурс]. – Режим доступу: https://www.computerhope.com/jargon/w/web-application.htm. – Дата звернення: 19.05.2019 р.
      2. Клієнт-серверна архітектура [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Клієнт-серверна\_архітектура. – Дата звернення: 19.05.2019 р.
      3. Клиент-сервер [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Server-side/First\_steps/Client-Server\_overview. – Дата звернення: 19.05.2019 р.
      4. REST is the new SOAP [Електронний ресурс]. – Режим доступу: https://www.freecodecamp.org/news/rest-is-the-new-soap-97ff6c09896d/. – Дата звернення: 19.05.2019 р.
      5. Односторінковий застосунок [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Односторінковий\_застосунок. – Дата звернення: 25.05.2019 р.
      6. Single-Page Applications: Build Modern, Responsive Web Apps [Електронний ресурс]. – Режим доступу: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>. – Дата звернення: 25.05.2019 р.
      7. Багатосторінкові та односторінкові додатки, їх переваги та недоліки [Електронний ресурс]. – Режим доступу: <https://dou.ua/forums/topic/25444/>. – Дата звернення: 25.05.2019 р.
      8. Introduction to React [Електронний ресурс]. – Режим доступу: <https://github.com/the-road-to-learn-react/the-road-to-learn-react/blob/master/manuscript/chapter1.md>. – Дата звернення: 01.06.2019 р.
      9. A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: <https://reactjs.org/>. – Дата звернення: 01.06.2019 р.
      10. Introducing JSX [Електронний ресурс]. – Режим доступу: <https://reactjs.org/docs/introducing-jsx.html>. – Дата звернення: 01.06.2019 р.
      11. Понимание ReactJS Router с базовым примером (NodeJS) [Електронний ресурс]. – Режим доступу: <https://o7planning.org/ru/12139/undertanding-react-router-with-a-basic-example>. – Дата звернення: 01.06.2019 р.
      12. Introduction to GraphQL [Електронний ресурс]. – Режим доступу: https://graphql.org/learn/. – Дата звернення: 01.06.2019 р.
      13. Что же такое этот GraphQL? [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/326986/. – Дата звернення: 01.06.2019 р.
      14. What is Apollo Client and what does it do? [Електронний ресурс]. – Режим доступу: https://www.apollographql.com/docs/react/. – Дата звернення: 01.06.2019 р.
      15. Apollo Boost [Електронний ресурс]. – Режим доступу: https://www.apollographql.com/docs/react/essentials/get-started/#apollo-boost. – Дата звернення: 01.06.2019 р.
      16. Easy to use React Bootstrap 4 components [Електронний ресурс]. – Режим доступу: <https://reactstrap.github.io/>. – Дата звернення: 01.06.2019 р.
      17. Git [Електронний ресурс]. – Режим доступу: https://git-scm.com/. – Дата звернення: 01.06.2019 р.
      18. Github [Електронний ресурс]. – Режим доступу: <https://github.com/>. – Дата звернення: 01.06.2019 р.