КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

На здобуття ступеня бакалавра

За освітньо-професійною програмою «Інформатика»

Спеціальність 122 «Комп'ютерні науки»

на тему:

РОЗРОБКА КЛІЄНТСЬКОГО ІНТЕРФЕЙСУ СИСТЕМИ ПІДТРИМКИ ІНТЕГРОВАНОГО НАВЧАЛЬНОГО КУРСУ

Виконав студент 4-го курсу бакалаврату

Пінковський Дмитро Олександрович _____

(підпис)

Науковий керівник:

Доцент, кандидат фіз.-мат. наук

Панченко Тарас Володимирович _____

(підпис)

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

Студент _____

(підпис)

Київ – 2021

ЗМІСТ

Перелік прийнятих скорочень

ПК	Персональний комп’ютер
MVC	Model, View, Controller; програмна архітектура, структура системи, яка відокремлює бізнес-логіку від іншої частини програми, логічно відокремлюючи додаток на 3 частини: Модель, Представлення и Контролер, відповідно скороченню
JS	JavaScript
DOM	Document Object Model, об’єктна модель документу
CSS	Cascading Style Sheets, каскадні таблиці стилів
Роутинг	Функціонал, котрий зіставляє набрану користувачем URL-адресу з дією програмної логіки WEB-інтерфейсу
API	Application Programming Interface, програмний інтерфейс додатку
JSON	JavaScript Object Notation, текстовий формат даних, який використовується для представлення об’єктів в JavaScript

РЕФЕРАТ

Бакалаврська робота складається із вступу, трьох розділів, висновків, списку використаних джерел (14 найменувань). Робота містить 22 малюнків. Загальний обсяг становить 32 сторінки, основний текст роботи викладено на 20 сторінках.

Ключові слова: КЛІЄНТСЬКИЙ ІНТЕРФЕЙС, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, ІНФОРМАЦІЙНІ СИСТЕМИ.

В ході роботі вирішуються наступні задачі:

1. Розкриття поняття «клієнтського інтерфейсу» та його типи, вимоги до створення клієнтського інтерфейсу.
2. Дослідження сучасних WEB-технологій.
3. Створення клієнтського інтерфейсу системи інтегрованого курсу.

Отримано новий досвід у технологіях, які використовуються при створення клієнтського інтерфейсу.

На основі дослідженні сучасних WEB-технологій, розробити клієнтський інтерфейс системи підтримки інтегрованого навчального курсу.

Було розроблено клієнтську частину додатку, який задовольняє критеріям мінімалізму, інтуїтивності та дозволяє користувачу легко виконувати свою роботу.

Інтерфейс повинен забезпечувати:

1. Можливість додавати/корегувати/видаляти курси та групи;
2. Можливість створювати інформативні пости для студентів та окремої групи від викладача;
3. Можливість виставлення викладачем оцінок за лабораторні роботи;
4. Можливість приватного спілкування у чаті.

ВСТУП

Оцінка сучасного стану об'єкта розробки. З кожним роком інформаційні технології все більшу роль в житті людей. Усі підприємства, освітні та державні заклади, мають свою особисту систему, яка несе в собі увесь об'єм інформації для підтримки прямої діяльності даної організації. В даний час існує велика кількість різних інформаційних систем, кожна з яких має свої особливості. Для того пришвидшити та дати змогу клієнту легко користуватися цими системами, вони повинні буди побудовані на однакових принципах.

Актуальність роботи та підстави для її виконання. Саме клієнтський інтерфейс бачить користувач і саме завдяки інтерфейсу, він оцінює додаток, а іноді і увесь продукт загалом.

Звичайно, зараз є велика кількість систем, які допомагають студентам та викладачам виконувати наукову діяльність, але кожна з систем-конкурента має один або декілька недоліків інтерфейсу (перевантаженість, не інтуїтивність, відсутність засобу комунікації між користувачами). Тому з'являється необхідність створення клієнтського інтерфейсу, який буде відповідати більшості потреб.

Мета і завдання роботи. Метою кваліфікаційної роботи є створення клієнтського інтерфейсу для інформаційної системи на основі дослідження найпопулярніших програмних засобів та вимог до клієнтського інтерфейсу. Для досягнення цієї мети поставлено такі задачі:

1. Ознайомитись з вимогами до створення клієнтського інтерфейсу;
2. Дослідити найпопулярніші технології;
3. Розробити клієнтський інтерфейс системи підтримки інтегрованого курсу.

Об'єкт, предмет, методи й засоби розроблення. Об'єктом кваліфікаційної роботи є інтерфейс WEB-систем і їх розробка.

Предмет роботи — розробка клієнтського інтерфейсу системи на прикладі GoogleClass.

В якості інструменту створення клієнтського інтерфейсу було обрано VS Code – інтегроване середовище розробки (IDE). В якості бібліотеки використовувався ReactJS – JavaScript-бібліотека для написання клієнтської сторони додатку.

Можливі сфери застосування. Результат кваліфікаційної роботи може використовувати кожен студент та викладач для спрощення процесів наукової діяльності.

РОЗДІЛ 1. КЛІЄНТСЬКИЙ ІНТЕРФЕЙС

1.1 Вимоги до клієнтського інтерфейсу

В умовах використання інформаційних технологій постало дуже актуальне питання стосовно взаємодії людини (користувача) з технічними та програмними засобами. Таку взаємодію забезпечує клієнтський інтерфейс.

В інформаційних технологіях інтерфейс – це сукупність засобів та правил, які забезпечують взаємодію пристроїв, програм.

Клієнтський інтерфейс – це елементи й компоненти програми, які здатні впливати на взаємодію користувача з програмним забезпеченням.

Клієнтський інтерфейс складається з 3 частин:

- спілкування програмного додатку з користувачем;
- спілкування користувача з програмним додатком;
- мова спілкування:

Мову спілкування обирається розробником програмного додатку. Інтерфейс користувача програмного додатку включає в себе:

- засоби відображення інформації, відображену інформацію, формати та коди;
- командні режими, мову «користувач - інтерфейс»;
- пристрої та технології введення даних;
- діалоги, взаємодії та транзакції між користувачем та комп'ютером, зворотній зв'язок з користувачем;
- підтримку прийняття рішень в конкретній предметній області;
- порядок використання програми та документацію на неї.

Клієнтський інтерфейс доволі часто приймають, як зовнішній вид програми. Але, насправді, користувач сприймає через нього усю програму загалом, а значить, що таке визначення є «душе вузьким».

Насправді, клієнтський інтерфейс об'єднує в собі усі елементи й компоненти програми, котрі здатні впливати на взаємодію користувача з програмним забезпеченням(ПЗ), це не тільки екран, який бачить користувач.

До цих елементів відноситься:

- набір завдань користувача, які він вирішує за допомогою системи;
- елементи управління системою;
- навігація між блоками системи;
- візуальний (і не тільки) дизайн екранів програми;
- засоби відображення інформації, відображена інформація та формати;
- прилади й технології введення даний;

Для успішної взаємодії користувача з програмним додатком, необхідно, щоб програмний додаток мав наступні риси:

1. *Доступність* – найбільш важливий елемент дизайну. Ціль клієнтського інтерфейсу полягає в тому, щоб надати можливість користувачеві взаємодіяти з системою. Якщо людина не може зрозуміти, як працює застосунок, він буде тільки заплутаний та, у підсумку, розчарований. Ось чому, розробляючи інтерфейс застосунку чи веб-сайту, необхідно врахувати, що він був інтуїтивно зрозумілим користувачеві.
2. *Мінімалізм*. Велика завантаженість – ворог гарного клієнтського інтерфейсу. Легко попасти у пастку надлишкової доступності – додаючи все більше і більше керівницьких елементів, так як це сильно навантажує інтерфейс. Він зростає, і користувачеві буде необхідно багато читати, щоб зрозуміти, де та для чого розташовано.
3. *Впевненість*. Багато дизайнерів прагнуть розробити інтерфейси «інтуїтивно зрозумілим». Це означає, що користувачі повинні інтуїтивно розуміти й осмислювати можливості проекту.
4. *Чуйність* означає декілька речей. Інтерфейс веб-сайту повинен працювати дуже швидко. Термін очікування завантаження сторінок дуже роздратовує користувача. Також чуйність означає постійну форму взаємодії з клієнтом. Інтерфейс повинен інформувати користувача о процесах виконання завдань. Кори клієнт бачить процес

виконання, він відчуває себе спокійніше. Особливо це дуже добре працює в повільних інтернет-каналах.

5. *Відповідність контенту.* При виборі визначених рішень при створенні дизайну необхідно брати у рахунок тип змісту сторінки. Різні сторінки можуть містити контент різного типу. Адаптація кожної сторінки під відповідний контент дозволить зробити роботу користувача простіше та зручніше.
6. *Привабливість.* Хороший інтерфейс повинен буди привабливим. Такий інтерфейс робить роботу більш приємною. Неможливо зробити інтерфейс, який буде подобатися усім, але можливо провести відповідну градацію клієнтів и на основі цього створити інтерфейс.
7. *Ефективність.* Клієнтський інтерфейс – це інструмент управління. Він надає доступ до різних функцій застосунку чи веб-сайту. Гарний інтерфейс повинен давати змогу користувачу з найменшими зусиллями виконати дії, які його цікавлять. Ще дуже важливо зрозуміти, що найчастіше користувач хоче виконати на окремій сторінці. Не потрібно виводити список усіх можливих застосунків, найчастіше користувачу цікава тільки невелика його частина. При цьому клієнт повинен дуже швидко знайти найбільш корисніші функції, це дуже спростить його спілкування з проектом.
8. *Поблажливість.* Програміст повинен буди готовий до того, що користувачі будуть робити помилки при роботі з проектом. Це може буду як через розробника, так і через клієнта. Необхідно дуже грамотно опрацювати усі можливі помилки – це є одним із найголовніших показників якості проекту. Важливим показником поблажливості інтерфейсу, є можливість зберігати дані від випадкових дій клієнта. Наприклад, якщо хтось видалив важливу інформацію, необхідна можливість її відновлення.

Працюючи над досягненням однієї з цих характеристик, можна створити проблеми для створення іншої. Наприклад, прагнучі створити інтерфейс більш

зрозумілим, додаючи багато описів і пояснень, що у кінцевому випадку робить інтерфейс ще більше громіздким й незручним. Або видаляючи контент для досягнення мінімалізму, можна створити додаток, який буде незрозумілий звичайному користувачеві. Що актуально для одного, для іншого може буди недопустимим.

Таким чином, маючі усі перераховані властивості, інформаційна система зможе оптимально працювати буть-де.

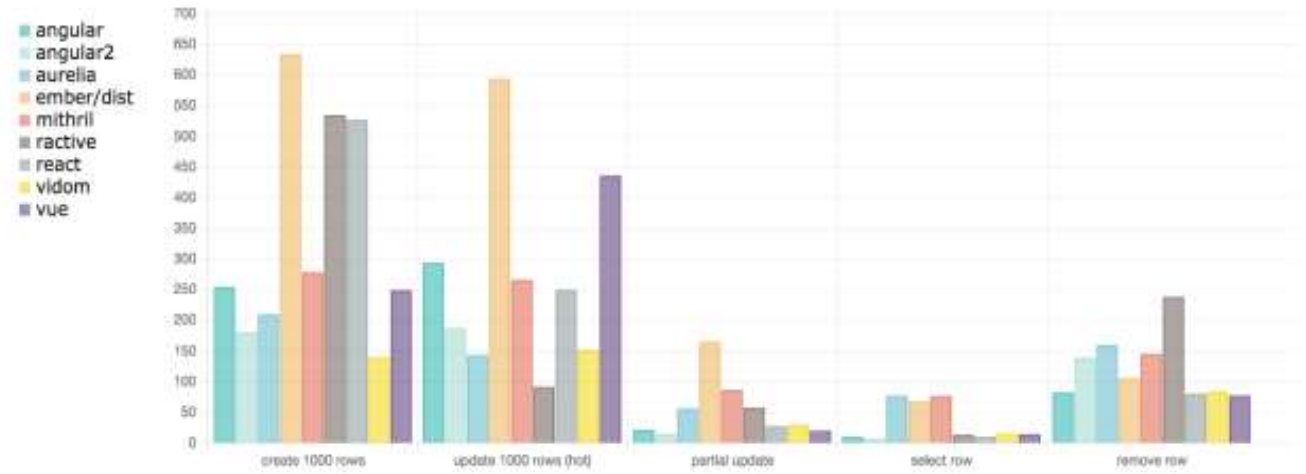
РОЗДІЛ 2. ДОСЛІДЖЕННЯ СУЧАСНИХ WEB-ТЕХНОЛОГІЙ ДЛЯ НАПИСАННЯ КЛІЄНТСЬКОГО ІНТЕРФЕЙСУ

2.1 Опис стеку технологій WEB-додатку.

Стек технологій, використаний при розробці WEB-додатку, включає в себе фреймворки і бібліотеки, які потрібні для ефективної та швидкої розробки клієнтської частини системи.

2.1.1 JavaScript-бібліотеки

Розглянемо основні JavaScript-бібліотеки і фреймворки, які використовуються в сучасному WEB-програмування. В роботі [1] наведено порівняння їх продуктивності (відповідно до малюнку 1) в браузері Google Chrome. В даному випадку тестується створення 1000 рядків відразу після завантаження сторінки ("create 1000 rows"), оновлення 1000 рядків в таблиці після 5 ітерацій «розігріву» JavaScript-движка ("update 1000 rows (hot) "), часткове оновлення рядків в таблиці після 5 ітерацій «розігріву» JavaScript-движка (додавання точки в кінці кожного 10-го рядку, "partial update"), вибір рядка після 5 ітерацій «розігріву» JavaScript-движка (візуальне виділення рядка, "select row"), видалення рядки після 5 ітерацій «розігріву» JavaScript-движка ("remove row"). Як видно з гістограми на рис. 1, найкраще показує себе бібліотека VueJS, крім відновлення рядків в таблиці. Хоча клієнтський інтерфейс системи інтегрованого курсу припускає візуальне оновлення даних у реальному часі(чат), а створення візуалізації курсів та груп створюється лише раз, тому треба було обрати бібліотеку, яка є продуктивною при оновленнях даних і добре поєднується з реалізацією клієнтського роутінгу. Такою бібліотекою є ReactJS.



Малюнок 1 – Гістограма продуктивності JS-фреймворків (мс)

React - розроблена компаніями Facebook і Instagram JavaScript-бібліотека з відкритим вихідним кодом для створення користувацьких інтерфейсів. Більшість користувачів React вважають його Виставою в ідеології MVC. React був розроблений, щоб вирішити велику проблему: написання великих додатків з даними, які змінюються з плином часу.

React - це бібліотека на мові JavaScript для створення користувацьких інтерфейсів. React дозволяє описувати елементи, тобто є декларативним. За допомогою даної технології можна безболісно створювати інтерактивні інтерфейси. Розробник може спроектувати прості уявлення для кожного стану майбутнього WEB-додатки, і React зможе ефективно і продуктивно оновити і перемалювати тільки ті компоненти, які були порушені зміною даних. Декларативні подання роблять код більш передбачуваним при виконанні і більш зручним при налагодженні.

Розробник також може створювати інкапсульовані компоненти, які керують своїм власним станом, а потім з'єднувати їх для подальшого використання і створення складних складових призначених для користувача інтерфейсів. Так як логіка компонентів написана на JavaScript, а не на мові-шаблонизаторі, програміст може легко передавати досить великий набір даних зі складною структурою по всьому додатком, зберігаючи стан поза DOM [2].

Завдяки використанню даної бібліотеки з'являється можливість завантажити клієнту (наприклад, в WEB-браузер) відразу всі можливі «Уявлення». Тобто для

кожної дії користувача в WEB-клієнті знайдеться підходяще графічне WEB-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 WEB-клієнт завантажується порівняно великий файл (близько 200-300 КБ для великих WEB-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 подання та Інтернет-ресурси користувача. В результаті спостерігається висока швидкість роботи WEB-сервісу, поліпшується чуйність інтерфейсу, прискорюється перехід між сторінками завдяки клієнтського роутингу, а шаблонізація інтерфейсу значно спрощує доопрацювання клієнтської частини WEB-додатку.

React побудований на концепції компонентів. Він відрізняється від таких фреймворків, як Angular або Ember, які використовують двосторонню прив'язку даних для оновлення HTML сторінки. На погляд багатьох фронтенд-розробників [3], React простіше для вивчення, ніж Angular або Ember - він набагато менше і добре працює з jQuery і іншими фреймворками. Він, до того ж, надзвичайно швидкий, так як використовує віртуальний DOM і оновлює тільки змінені частини сторінки (звернення до DOM досі є самою повільною частиною сучасних WEB-додатків, тому дана бібліотека і отримує перевагу в продуктивності, оптимізуючи його).

Клієнтський роутінг в цьому випадку можна реалізувати за допомогою бібліотеки react-router [4]. Дана бібліотека дозволяє пов'язувати клієнтські роути з React-компонентами, таким чином, всі можливі стани WEB-інтерфейсу будуть міститися в JavaScript-файлі проекту, де оголошені роути (в точці входу додатки, у відповідність з малюнком 2).

Малюнок 2 - Код JS-файлу точки входу додатку, що містить клієнтський роутінг

Так як більшість браузерів, що використовуються сьогодні, не підтримують сучасні стандарти ECMAScript [5], в браузерах необхідно використовувати старий стандарт ECMAScript 5, який підтримується всіма сучасними браузерами, включаючи Internet Explorer 11 [6], що дозволяє значно розширити можливість використання розроблюваного WEB-інтерфейсу. Однак, відмовляючись від новітніх стандартів мови програмування, розробник також відмовляється від можливостей використовувати новий синтаксичний цукор і деякі оптимізаційні рішення нових стандартів. Для цього необхідно транслювати JavaScript більш “свіжого” стандарту [5] в JavaScript стандарту ECMAScript 5, з цим завданням справляється Babel (JavaScript-компілятор, на сайті [7] розташовується його документація).

У підсумку виходить досить багато залежностей, які потрібно імпортувати в більшості файлів, також, при розробці програми, код розростається і з'являється

безліч файлів. Для складання JavaScript-файлів і файлів стилів в один JavaScript-файл і файл стилів відповідно, знадобиться система збирання коду і пакетів в WEB-фронтенді, Webpack, документація якого розташована на сайті [8].

Для візуалізації було обрано бібліотеку React Material UI [9]. Ця бібліотека знімає частину обов'язків з розробника та дозволяє обрати готові компоненти, які можна кастомізувати під свої стильові вподобання.

Сучасні технології WEB-програмування дозволяють реалізувати WEB-додатки, що задовольняють, як відсутності надмірних обчислень і оптимізаційних операцій при малому потоці даних, так і відсутності видимих затримок обчислень при великому потоці даних. Різні завдання розробок вимагають використання різних технологій, методів і підходів до обробки інформації і оптимізації даного процесу.

РОЗДІЛ 3. РОЗРОБКА КЛІЄНТСЬКОГО ІНТЕРФЕЙСУ СИСТЕМИ ПІДТРИМКИ ІНТЕГРОВАНОГО НАВЧАЛЬНОГО КУРСУ

3.1 Огляд аналогів

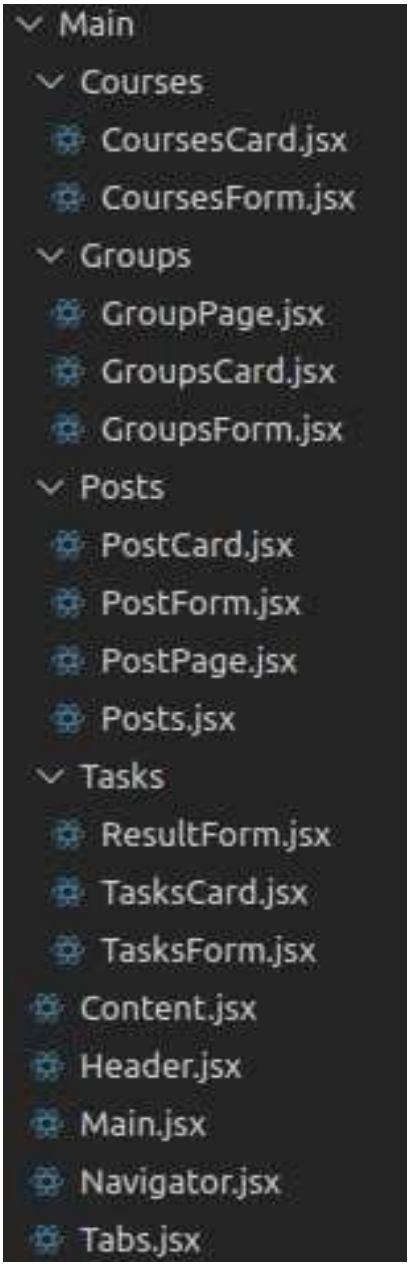
Дослідивши декілька систем-конкурентів, було виявлено декілька основних недоліків, які роблять систему неконкурентоспроможною, а саме:

1. Більшість систем розроблені на технологіях, які фізично не можуть обробляти велику кількість інформації;
2. Деякі системи розробляються, як щось дуже серйозне, тому в результаті вони стають інформативно перенавантажені, що не дає можливості комфортно працювати;
3. Навігація не є інтуїтивною, що також не дає можливості комфортно працювати;
4. Одне з найголовніших – це можливість приватного спілкування з викладачем чи одногрупником в інтегрованому чаті є в одиницях систем.

3.2 Розробка клієнтського інтерфейсу системи

Архітектура клієнтської частини WEB-додатку складається з:

- Кореня додатку, яке забезпечує клієнтський роутінг (SPA – Single Page Application) та дозволяє змінювати візуальне представлення браузера без повного перезавантаження сторінки. (у відповідності з малюнком 2)
- Компонентів та стилів, які реалізують візуалізацію даних в клієнтському браузері (у відповідності з малюнком 3)



Малюнок 3 – Структура директорій React-компонентів

Спочатку треба було створити сам проект та завантажити усі залежності, які необхідні для розробки клієнтської частини. Середовищем для розробки було обрано VS Code, який дуже добре взаємодіє з ReactJS та підсвічує код, щоб розробнику було простіше працювати та орієнтуватися у коді.

Розробка клієнтського інтерфейсу складається з трьох частин:

1. Розробка сторінки Реєстрації/Авторизації;
2. Розробка клієнтського інтерфейсу Додатку;
3. Розробка сторінки Чату.

Перед початком розробки було сформовано UML-діаграму взаємодії студента та викладача(у відповідності з малюнком 4).

Малюнок 4 — UML-діаграма студента та викладача

Під час розробки першої частини не було використано бібліотеки Material UI, натомість, за стилі відповідав препроцесор SCSS. Основною метою при розробці компонента Авторизації було написання логіки, яка дозволить зберігання даних користувача в Cookie, для того, щоб забезпечити сесії користування. Ця мета була досягнута таким чином (у відповідності з малюнком 5)

```
import axios from "axios";
import React from "react";
import loginImg from "../../login.svg";
import Cookies from 'universal-cookie';
import base64 from 'base-64';
export class Login extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      password: '',
      uid: ''
    }
  }

  myChangeHandleUID = e => {
    this.setState({uid: e.target.value})
  }

  myChangeHandlePSWD = e => {
    this.setState({password: e.target.value})
  }

  handleSubmit = e => {
    e.preventDefault();

    axios.defaults.baseURL = 'https://service-ems.herokuapp.com/'
    axios.defaults.withCredentials = true;
    axios.post('/login',
        {
          uid: this.state.uid,
          password: base64.encode(this.state.password)
        }
    ).then(response => {
      const cookies = new Cookies();
      cookies.set('SESSIONID', response.data.sessionId, {path: '/'});
      cookies.set('ROLE', response.data.role, {path: '/'});
      localStorage.setItem('uid',this.state.uid);
      localStorage.setItem('role',response.data.role);
    });

  }
```

Малюнок 5 – Зберігання даних користувача в Cookie

Частина коду на малюнку №5 спілкується через метод POST з сервером та зберігає данні користувача в обраним ним браузером, але лише на моменти сеансу.

Компонент Реєстрації зчитує дані, які користувач записав у формі реєстрації, та відправляє на сервер за допомогою axios та методу POST (у відповідності з малюнком 6). Після цього користувач може авторизуватись та перейти до головної сторінки.

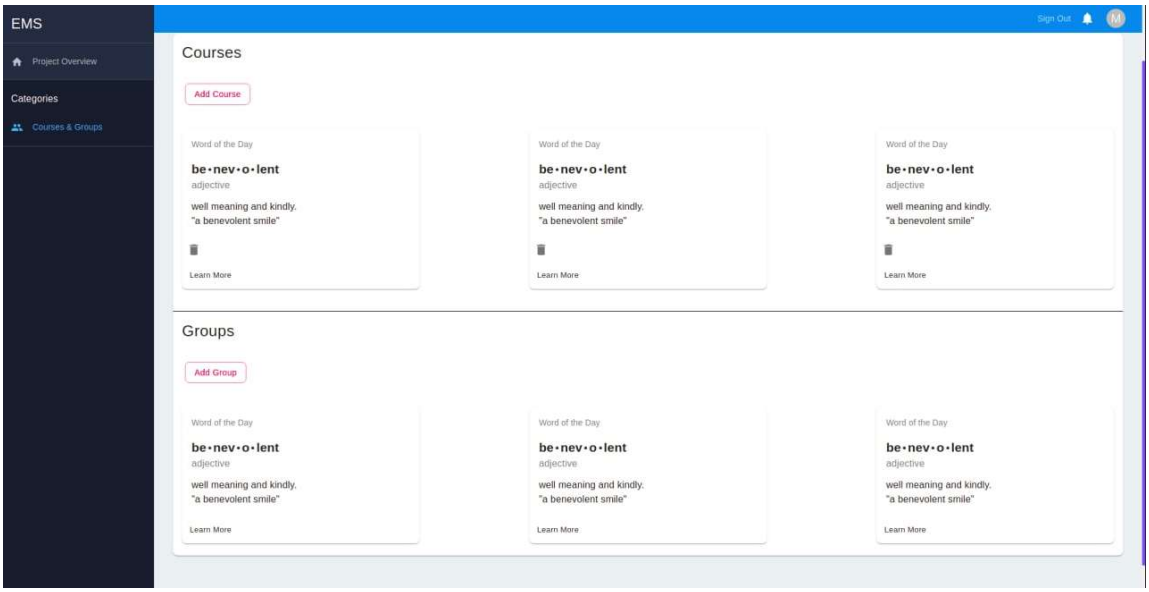
Наступним та найбільшим етапом було створення Основної частини клієнтського інтерфейсу. Основний інтерфейс цієї частини складається з трьох компонентів:

1. Header (Шапка), який відповідає за верхню частину інтерфейсу;
2. Navigator (Навігація), який відповідає за переключання між частинами додатку;
3. Content (Наповнення), який відповідає за основну інформацію на сайті.

Третій компонент включає в себе інформацію про Курси, Групи та їх сторінки.

```
import React from "react";
import loginImg from "../../login.svg";
import axios from "axios";
import base64 from "base-64";

export class Register extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      firstName: '',
      lastName: '',
      password: '',
      uid: '',
      role: 'Student'
    }
  }

  myChangeHandleUID = e => {
    this.setState({uid: e.target.value})
  }

  myChangeHandlePSWD = e => {
    this.setState({password: e.target.value})
  }

  myChangeHandleFirstName = e => {
    this.setState({firstName: e.target.value})
  }

  myChangeHandleLastName = e => {
    this.setState({lastName: e.target.value})
  }
  //TODO fix role handling???
  myChangeHandleRole = e => {
    this.setState({role: e.target.value})
  }

  handleRegister = e => {
    e.preventDefault();

    axios.defaults.baseURL = 'https://service-ems.herokuapp.com/'
    axios.defaults.withCredentials = true;
    axios.post('/register',
        {
          firstName: this.state.firstName,
          lastName: this.state.lastName,
          password:  base64.encode(this.state.password),
          uid: this.state.uid,
          role: this.state.role
        }
    );
```

Малюнок 6 – Зчитування даних користувача з форми та відправлення на сервер



Малюнок 7 – Головна сторінка системи

Компонент Content включає в себе логіку відображання компонентів Groups та Courses за допомогою «хука» станів (useState). Цей хук дозволяє використовувати стан та інші можливості React без написання класів. Також завдяки методу GET можна запросити усі картки груп та курсів з сервера(у відповідності з малюнком 8).

```
function Content(props) {
    const {classes} = props;
    const [isLoadingCourses, setLoadingCourses] = React.useState(true);
    const [isLoadingGroup, setLoadingGroup] = React.useState(true);
    const [courseData, setCourseData] = React.useState([]);
    const [groupData, setGroupData] = React.useState([]);

    axios.defaults.baseURL = 'https://service-ems.herokuapp.com/'
    axios.defaults.withCredentials = true;
    if (isLoadingCourses) {
        axios.get('/all-courses').then(function (res) {
            let courseGrids = [];
            res.data.forEach(function (item) {
                let card = React.createElement(CoursesCard, {
                    courseName: item.name,
                    courseUid: item.uid,
                    courseLecturer: item.lecturer
                });
                let grid = React.createElement(Grid, {item, className: classes.course, md: 3}, card);
                courseGrids.push(grid);
            });
            setCourseData(courseGrids);
            setLoadingCourses(false);
        });
    }
    if (isLoadingGroup) {
        axios.get('/all-groups').then(function (res) {
            let grids = [];
            res.data.forEach(function (item) {
                let card = React.createElement(GroupsCard, {
                    groupName: item.name,
                    groupUid: item.uid,
                    groupLecturer: item.lecturer
                });
                let grid = React.createElement(Grid, {item, className: classes.course, md: 3}, card);
                grids.push(grid);
            });
            setGroupData(grids);
            setLoadingGroup(false);
        });
    }
```

Малюнок 8 – Використання методу GET та хука useState

Далі відображання усієї інформації подається у наступному вигляді (у відповідності з малюнком 9). Використання компонентів бібліотеки Material UI дозволяє вставляти свою інформацію, як у тег HTML, що забезпечує легку інтеграцію великої інформації в код React додатку.

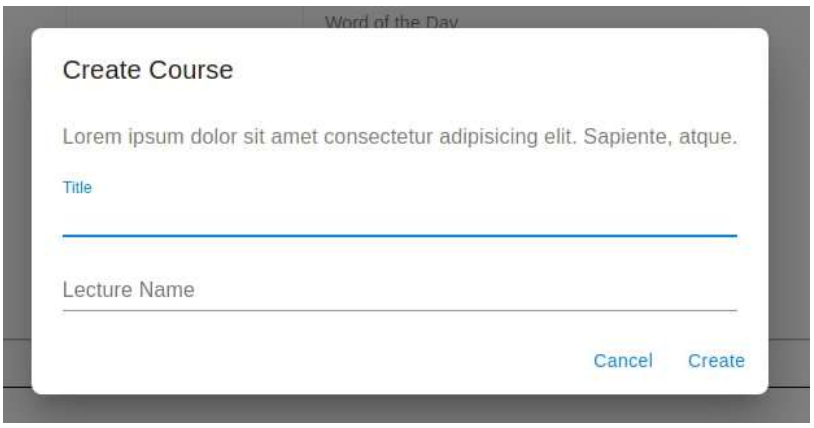
```
    return (
        <div>
            {}
            <Paper className={classes.paper}>
                {}
                {}
                <div>
                    <Typography color="inherit" variant="h5" component="h1" style={{margin: 15}}>
                        Courses
                    </Typography>
                    <CoursesForm/>
                    <Grid container spacing={24} justify="space-between">
                        {courseData}
                    </Grid>
                    <Typography
                        type="title"
                        color="inherit"
                        style={{borderBottom: '0.1em solid black', padding: '0.5em'}}
                    >

                    </Typography>
                    <Typography color="inherit" variant="h5" component="h1" style={{margin: 15}}>
                        Groups
                    </Typography>
                    <GroupsForm/>
                    <Grid container spacing={24} justify="space-between">
                        {groupData}
                    </Grid>
                </div>
            </Paper>
        </div>
    );
}
```

Малюнок 8 – Відображання інформації, яка отримана з сервера

Після переходу на головну сторінку користувач може обрати декілька варіантів використання клієнтський інтерфейсом:

1. Користувач може зробити Sign Out натиснувши на однойменну кнопку в шапці сторінки, тим самим він вернеться на сторінку Авторизації/Реєстрації;
2. Створити, видалити або перейти на сторінку Курсу чи Групи

Створення нового Курсу чи Групи можна за допомогою форми, яка відкривається при натисканні на кнопку «Add Course» чи «Add Group»(у відповідності з малюнком 10).



Малюнок 10 – Форма створення Курсу

Код реалізації логіки створення Курсу та відправлення інформації на сервер за допомогою методу POST реалізовано, як це показано на малюнку 11. Таким самим чином реалізовано створення нової Групи. Далі натиснувши на кнопку «Learn more» можна перейти на сторінку Курсу чи Групи відповідно. На сторінці Курсу можна побачити реалізацію Посту, який дозволяє учасникам прочитати

важливу інформацію від викладача. Такий саме пост викладач може створити для окремої групи використовуючи дуже схожу форму, яка створює Групу. Видалення та виправлення постів виконується за допомогою HTTP запитів PUT та DELETE.

Представлення постів виконується за допомогою вже відомого нам хуку useState. Найцікавішою функцією на сторінці Курсу – є представлення компоненту Tabs, який дозволяє поєднати багато сторінок на одній не перевантажуючи додаток. На малюнках 12 та 13 можна побачити, як за допомогою хуку useState та вбудованої можливості цього компоненту можна відображати дві сторінки з різним контентом.

```
import React from 'react';
import Button from '@material-ui/core/Button';
import TextField from '@material-ui/core/TextField';
import Dialog from '@material-ui/core/Dialog';
import DialogActions from '@material-ui/core/DialogActions';
import DialogContent from '@material-ui/core/DialogContent';
import DialogContentText from '@material-ui/core/DialogContentText';
import DialogTitle from '@material-ui/core/DialogTitle';
import axios from "axios";

export default function FormDialog() {
    const [open, setOpen] = React.useState(false);

    let description = '';
    let name = '';

    const handleClickOpen = () => {
        setOpen(true);
    };

    const handleClose = () => {
        setOpen(false);
    };

    const changeDescription = e => {
        description = e.target.value;
    }

    const changeName = e => {
        name = e.target.value;
    }

    const handleCreate = () => {
        axios.defaults.baseURL = 'https://service-ems.herokuapp.com//'
        axios.defaults.withCredentials = true;
        axios.post('add?userUid='+localStorage.getItem('uid'),
            {
                "description": description,
                "name": name
            }
        ).then();
        setOpen(false);
    }
```

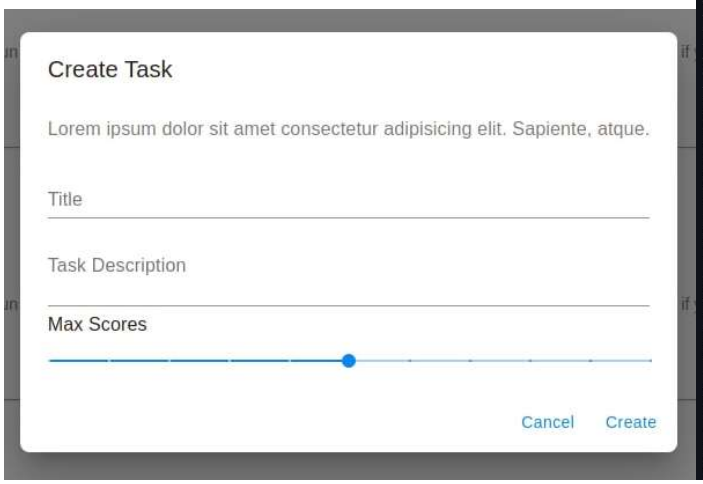
Малюнок 11 – Реалізація створення нового Курсу

```jsx
function TabPanel(props) {
  const { children, value, index, ...other } = props;

  return (
    <div
      role="tabpanel"
      hidden={value !== index}
      id={`simple-tabpanel-${index}`}
      aria-labelledby={`simple-tab-${index}`}
      {...other}
    >
      {value === index && (
        <Box p={3}>
          <Typography>{children}</Typography>
        </Box>
      )}
    </div>
  );
}

TabPanel.propTypes = {
  children: PropTypes.node,
  index: PropTypes.any.isRequired,
  value: PropTypes.any.isRequired,
};

function a11yProps(index) {
  return {
    id: `simple-tab-${index}`,
    'aria-controls': `simple-tabpanel-${index}`,
  };
}
```

```jsx
return (
  <div className={classes.root}>
    <AppBar position="static">
      <Tabs value={value} onChange={handleChange} aria-label="simple tabs example">
        <Tab label="Posts" {...a11yProps(0)} />
        <Tab label="Tasks" {...a11yProps(1)} />
      </Tabs>
    </AppBar>
    <TabPanel value={value} index={0}>
      <PostForm/>
      {posts.map((t,index)=>{
          return(
              <PostCard title={t.title} nickName={t.nickName}/>
          )
      })}
    </TabPanel>
    <TabPanel value={value} index={1}>
      <Box display="flex">
        <TasksForm/>
        <ResultForm />
      </Box>
      {tasks.map((t,index)=>{
          return(
              <TasksCard title={t.title} score={t.score}/>
          )
      })}
    </TabPanel>
  </div>
);
}
```

Малюнки 12 та 13 – Використання хуку useState для відображання контенту

Завдяки компоненту Tabs користувач може переглянути сторінку Завдань. На цій сторінці викладач може створити нове завдання для курсу та зазначити максимальну кількість балів, яку може отримати студент(у відповідності з малюнком 14). На малюнку 15 можна побачити, як програмно відображена ця форма з використання синтаксису бібліотеки Material UI.




```jsx
return (
  <div>
    <Button variant="outlined" color="secondary" onClick={handleClickOpen} style={{margin: 20}}>
      Add Task
    </Button>
    <Dialog open={open} onClose={handleClose} aria-labelledby="form-dialog-title">
      <DialogTitle id="form-dialog-title">Create Task</DialogTitle>
      <DialogContent>
        <DialogContentText>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Sapiente, atque.
        </DialogContentText>
        <TextField
          autoFocus
          margin="dense"
          id="title"
          label="Title"
          type="text"
          fullWidth
        />
        <TextField
          margin="dense"
          id="Task Description"
          label="Task Description"
          type="text"
          multiline
          rows={2}
          rowsMax={400}
          fullWidth
        />
        <Typography id="discrete-slider" gutterBottom>
          Max Scores
          </Typography>
          <Slider
              defaultValue={20}
              aria-labelledby="discrete-slider"
              valueLabelDisplay="auto"
              step={10}
              marks
              min={0}
              max={100}
          />
      </DialogContent>
      <DialogActions>
        <Button onClick={handleClose} color="primary">
          Cancel
        </Button>
        <Button onClick={handleClose} color="primary">
          Create
        </Button>
      </DialogActions>
    </Dialog>
  </div>
);
}
```

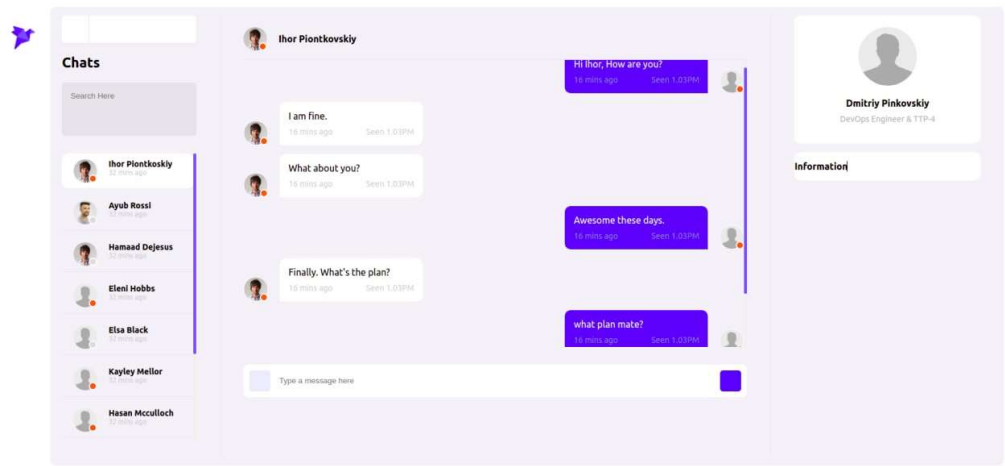
Малюнки 14 та 15 – Форма створення Завдання та її програмна реалізація

Для відправлення результатів завдання студент використовує форму результатів, щоб додати посилання на сховище. Це посилання відправляється на поштову скриньку викладача для перевірки.

Також для спілкування користувачів було реалізовано інтерфейс чату. Цей інтерфейс дозволяє знайти потрібного користувача, щоб почати з ним приватне листування. Реалізація надсилання повідомлень реалізована за допомогою хуку componentDidMount, який викликається лише раз в усьому життєвому циклу компонента й сигналізує, що компонент та усі його дочірні компоненти відмальовані без помилок (у відповідності з малюнком 16). Сам інтерфейс чату зазначений на малюнку 17.

```
componentDidMount() {
  window.addEventListener("keydown", (e) => {
    if (e.keyCode == 13) {
      if (this.state.msg != "") {
        this.chatItms.push({
          key: 1,
          type: "",
          msg: this.state.msg,
          image:
            "https://www.paintingcontest.org/components/com_djclassifieds/assets/images/default_profile.png",
        });
        this.setState({ chat: [...this.chatItms] });
        this.scrollToBottom();
        this.setState({ msg: "" });
      }
    }
  });
  this.scrollToBottom();
}
onStateChange = (e) => {
  this.setState({ msg: e.target.value });
};
```

Малюнок 16 – Реалізація хуку componentDidMount



Малюнок 17 – Сторінка Чату

По закінченю розробки клієнтского інтерфейсу було сформовано фінальний варіант React-компоненту Main.jsx, який містить в собі збірні компоненти додатку(у відповідності з малюнками 18 та 19).

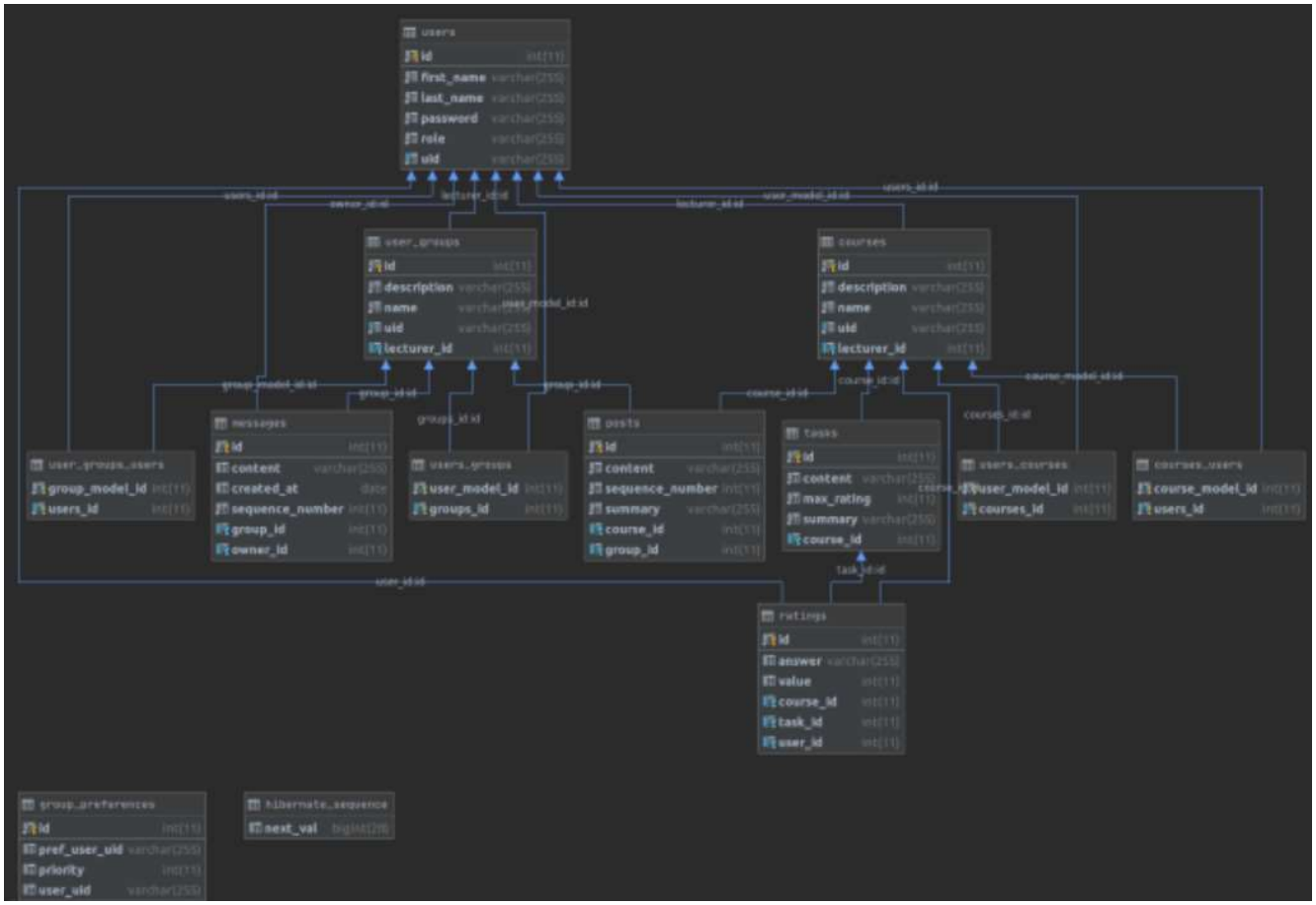
```jsx
import React from 'react'
import PropTypes from 'prop-types'
import { createMuiTheme, ThemeProvider, withStyles } from '@material-ui/core/styles'
import CssBaseline from '@material-ui/core/CssBaseline'
import Hidden from '@material-ui/core/Hidden'
import Typography from '@material-ui/core/Typography'
import Navigator from './Navigator'
import Content from './Content'
import Header from './Header'
import { Switch, Route } from 'react-router-dom'
import Post from './Posts/PostPage'
import Group from './Groups/GroupPage'

function Copyright() {
    return (
        <Typography variant="body2" color="textSecondary" align="center">
            {'Copyright © Dream Fifth Group '}
            {new Date().getFullYear()}
            {'.'}
        </Typography>
    )
}
```

```jsx
function Paperbase(props) {
    const { classes } = props
    const [mobileOpen, setMobileOpen] = React.useState(false)

    const handleDrawerToggle = () => {
        setMobileOpen(!mobileOpen)
    }

    return (
        <ThemeProvider theme={theme}>
            <div className={classes.root}>
                <CssBaseline />
                <nav className={classes.drawer}>
                    <Hidden smUp implementation="js">
                        <Navigator PaperProps={{ style: { width: drawerWidth } }} variant="temporary" open={mobileOpen} onClose={handleDrawerToggle} />
                    </Hidden>
                    <Hidden xsDown implementation="css">
                        <Navigator PaperProps={{ style: { width: drawerWidth } }} />
                    </Hidden>
                </nav>
                <div className={classes.app}>
                    <Header login={props.login} onDrawerToggle={handleDrawerToggle} />
                    <main className={classes.main}>
                        <Switch>
                            <Route
                                exact
                                path="/main"
                                render={() => {
                                    return <Content />
                                }}
                            />
                            <Route
                                exact
                                path="/main/posts"
                                render={props => {
                                    return <Post {...props.match.params} />
                                }}
                            />
                            <Route
                                exact
                                path="/main/groups"
                                render={props => {
                                    return <Group {...props.match.params} />
                                }}
                            />
                        </Switch>
                    </main>
                    <footer className={classes.footer}>
                        <Copyright />
                    </footer>
                </div>
            </div>
        </ThemeProvider>
    )
}
```

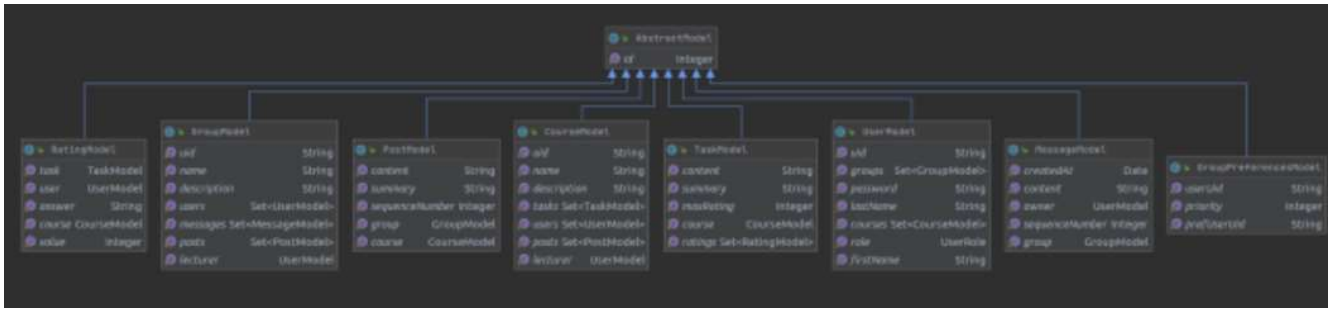
Малюнок 18 та 19 — Зміст React-компоненту Main.jsx

Під час розробки клієнтського інтерфейсу моєму колезі було запропоновано створити серверну частину на мові програмування Java (Spring) та реляційну базу даних MySQL (ClearSQL).

Після того, як були обрані основні компоненти серверної частини, ми почали проектувати модельний шар додатку, а саме: було спроектовано ER діаграму для БД(у відповідності з малюнком 19), по ній побудовано Java класи(у відповідності з малюнком 20), а потім, за допомогою JPA, було згенеровано схему БД, як для локального, так і для віддаленого середовища.



Малюнок 19 – ER діаграма бази даних



Малюнок 20 – Діаграма класів модельного шару

Під час розробки клієнтського інтерфейсу було сформовано файл package.json із залежностями, які дозволяють створювати майже будь-який клієнтський інтерфейс(у відповідності з малюнком нище).

```
{} package.json > ...
{
  "name": "group-app",
  "version": "0.1.0",
  "private": true,
  "homepage": "hhttps://front-ems.herokuapp.com/",
  "dependencies": {
    "@material-ui/core": "^4.11.3",
    "@material-ui/icons": "^4.11.2",
    "@testing-library/jest-dom": "^5.11.9",
    "@testing-library/react": "^11.2.5",
    "@testing-library/user-event": "^12.7.1",
    "axios": "^0.21.1",
    "base-64": "^1.0.0",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.2",
    "sass": "^1.32.8",
    "universal-cookie": "^4.0.4",
    "utf8": "^3.0.0",
    "web-vitals": "^1.1.0"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "node-sass": "^5.0.0",
    "webpack-cli": "^4.5.0"
  }
}
```

Також, дуже важливим показати HTTP-запит для комунікації інтерфейсу з сервером(у відповідності з малюнком 22).

```
axios.defaults.baseURL = 'https://service-ems.herokuapp.com/'
axios.defaults.withCredentials = true;
if (isLoadingCourses) {
    axios.get('/all-courses').then(function (res) {
        let courseGrids = [];
        res.data.forEach(function (item) {
            let card = React.createElement(CoursesCard, {
                courseName: item.name,
                courseUid: item.uid,
                courseLecturer: item.lecturer
            });
            let grid = React.createElement(Grid, {item, className: classes.course, md: 3}, card);
            courseGrids.push(grid);
        });
        setCourseData(courseGrids);
        setLoadingCourses(false);
    });
}
```

Малюнок 22 — Приклад HTTP-запиту

ВИСНОВОК

У результаті виконання бакалаврської роботи була досягнута поставлена мета та виконані поставлені задачі, а саме:

- Було виконано дослідження та аналіз сучасних WEB-технологій;
- Розкрите поняття клієнтського інтерфейсу;
- Розроблено клієнтський інтерфейс системи підтримки інтегрованого курсу, який задовольняє більшості потреб користувача.

У результаті дослідження були виявлені основні проблеми при розробці клієнтського інтерфейсу, проведений аналіз та порівняння сучасних WEB-технологій, які дозволяють вирішити дані проблеми.

Після виконання поставленого завдання клієнтський інтерфейс дозволяє користувачу дізнаватися нову інформацію через пости Груп та Курсів. Отримувати завдання й надсилати посилання на виконану роботу. Спілкуватися з однокурсниками та викладачами за допомогою вбудованого чату.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. JS web frameworks benchmark. [Електроний ресурс]. – http://www.stefankrause.net/wp/?p=191
2. A Javascript library for building user interfaces – React. [Електроний ресурс]. –https://ru.reactjs.org/
3. 5 практических примеров для изучения фреймворка React. [Електроний ресурс]. – https://habrahabr.ru/post/229629/
4. Github - ReactTraining/react-router: Declarative routing for React. [Електроний ресурс]. – https://reactrouter.com/web/guides/quick-start
5. ECMAScript 5 compatibility table. [Електроний ресурс]. – http://kangax.github.io/compat-table/es5/
6. Babel · The compiler for writing next generation JavaScript. [Електроний ресурс]. — https://babeljs.io/
7. Webpack module bundler. [Електроний ресурс]. – https://webpack.github.io/
8. Material UI. [Електроний ресурс]. – https://material-ui.com/ru/
9. Set Up Your Build Tools | Web | Google Developers. [Електроний ресурс]. – https://developers.google.com/web/tools/setup/setup-buildtools#dont_trip_up_with_vendor_prefixes
10. Node.js [Електроний ресурс]. – https://nodejs.org/en/
11. npm [Електроний ресурс]. – https://www.npmjs.com/
12. Git [Електроний ресурс]. – https://git-scm.com/
13. Інтерфейс користувача [Електроний ресурс]. – https://znaimo.com.ua/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0

14. Carlos R. React Cookbook: Create dynamic web apps with React using Redux, Webpack, Node.js, and GraphQL. — Packt Publishing, 2018. — 580 p.
15. Gorgon Z. React Explained: Your Step-by-Step Guide to React. – Amazon Digital Services LLC, 2019. — 305 p.
16. Wieruch R. The Road to learn React: Your journey to master plain yet pragmatic React.js. — Amazon Digital Services LLC, 2017. — 202 p.