ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмної системи обробки заяв на поселення в гуртожиток»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 – Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

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

	Вадим КУЗЬКОВИЙ
(підпис)	*Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:	здобувач вищої освіти гр.КІД-41 Вадим КУЗЬКОВИЙ Ім'я, ПРІЗВИЩЕ
Керівник: *науковий ступінь, вчене звання*	Дмитро РОЗМАЇТИЙ Ім'я, ПРІЗВИЩЕ
Рецензент: *науковий ступінь, вчене звання*	 Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерної інженерії
Ступінь вищої освіти бакалавр
Спеціальність 123 – Комп'ютерна інженерія
Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
_____Ім'я, ПРІЗВИЩЕ
«____» _____2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кузьковий Вадим Вікторович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка програмної системи обробки заяв на поселення в гуртожиток

керівник кваліфікаційної роботи Дмитро РОЗМАЇТИЙ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024р. № 36

2. Строк подання кваліфікаційної роботи «03» червня 2024р.

3. Вихідні дані до кваліфікаційної роботи: axios, notiflix, react, react-dom, react-hook-form, react-icons, react-loader-spinner, react-redux, web-vitals, redux-persist, RestAPI

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Розробка програмної системи
2. Взаємодія з бекендом
3. Рендреринг сторінки

5. Перелік ілюстративного матеріалу: реферат, актуальність, інструменти, які були використані, підключення бібліотек для роботи, DOM, маршрутизація у React, керування логікою у React, фільтрація заяв, рендеринг заяв, висновки

6. Дата видачі завдання «27» лютого 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	26.02.2024-22.03.2024	
2	Обробка матеріалу	23.03.2024-05.04.2024	
3	Огляд інструментів	06.04.2024-10.04.2024	
4	Розробка програмної системи	11.04.2024-30.04.2024	
5	Взаємодія з бекендом	01.05.2024-03.05.2024	
6	Рендреринг сторінки	04.05.2024-08.05.2024	
7	Розробка презентації	09.05.2024-11.05.2024	
8	Передзахист	14.05.2024-17.05.2024	
9	Здача в деканат	25.05.2024-01.06.2024	

Здобувач(ка) вищої освіти _____ Вадим КУЗЬКОВИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Дмитро РОЗМАЇТИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 47 стор., 11 рис., 30 джерел.

Мета роботи – розробити програмну систему обробки заяв на поселення в гуртожиток

Об'єкт дослідження – розробка програмної системи

Предмет дослідження – програмна система

Короткий зміст роботи: - *опис репозиторію, робота із компонентами React та відображення результату на живій сторінці. Створення фільтрації для сортування заяв студентів*

КЛЮЧОВІ СЛОВА: ПРОГРАМНА СИСТЕМА, HTML, CSS, JAVASCRIPT, REACTJS, REDUX, NODEJS, БЕКЕНД, DOM.

ЗМІСТ

ВСТУП

У сучасному світі, де молодь активно вивчає нові спеціальності та шукає можливості для освіти й самореалізації, проблема поселення у гуртожиток набуває особливого значення. Зростаюча кількість студентів та молодих спеціалістів, що потребують комфортного житла під час навчання чи роботи, ставить перед університетами та організаціями, які забезпечують молодь житлом, низку складних завдань.

У цьому контексті, розробка програмної системи для поселення у гуртожиток стає актуальним напрямком, спрямованим на оптимізацію та поліпшення процесів прийому та обслуговування мешканців гуртожитків. Ця робота присвячена розгляду та розробці програмної системи, що спростить та зробить більш ефективним процес поселення студентів у гуртожитки.

Мета дослідження полягає у вивченні поточного стану системи поселення у гуртожитки, а також у розробці та впровадженні інноваційних інструментів, спрямованих на автоматизацію та оптимізацію цього процесу. Впровадження програмної системи має на меті зменшення витрат часу та ресурсів, підвищення зручності для користувачів та покращення загального рівня обслуговування.

Кваліфікаційна робота буде базуватися на аналізі поточних методів та процесів поселення у гуртожитки, а також на дослідженні потреб та вимог користувачів. Результатом дослідження буде створення програмної системи, яка враховуватиме потреби всіх зацікавлених сторін - від адміністраторів гуртожитків до самих мешканців.

Потенційними користувачами розробленої системи є університети, студенти, а також адміністратори гуртожитків. Впровадження програмної системи має на меті спростити процес поселення та зробити його більш прозорим та зручним для всіх учасників.

У подальшому, розроблена програмна система може бути застосована в різних галузях та організаціях, де існує потреба в ефективному управлінні

житловими ресурсами та процесами поселення.

Оскільки я сам проживав у гуртожитку і мав досвід поселення, я вирішив створити програмну систему обробки заяв. Вона буде корисна і студентам, і особам, що відповідають за поселення студентів.

1 ОГЛЯД ІНСТРУМЕНТІВ, ЩО БУЛИ ВИКОРИСТАНІ ДЛЯ РОБОТИ

1.1 HTML

HTML (Hypertext Markup Language) - це мова розмітки, яка використовується для створення веб-сторінок. Вона визначає структуру та зовнішній вигляд веб-сторінки за допомогою тегів і атрибутів.

Отже, основні компоненти HTML включають:

Теги. HTML використовує теги для визначення різних елементів веб-сторінки, таких як заголовки, параграфи, зображення тощо. Теги складаються з відкриваючого та закриваючого елементів. Наприклад:

– *Елементи.* Елементи - це будь-який об'єкт на сторінці, який можна визначити за допомогою HTML-тегів. Вони можуть бути блоковими або вбудованими. Наприклад:

– *Атрибути.* Атрибути надають додаткову інформацію про елементи, таку як ідентифікатор, клас, розмір, посилання тощо. Наприклад:

– *Коментарі.* HTML дозволяє вставляти коментарі у код, які не відображаються на веб-сторінці, але можуть бути корисні для розробника.

– *Структура.* Зазвичай веб-сторінка складається з різних елементів, таких як <head>, <title>, <body>, які визначають заголовок, мета-інформацію, зовнішній вигляд та контент сторінки відповідно.

Безпосередньо у проекті не буде використана html розмітка у документі, що містить кодову назву “index.html”, тому що вся розмітка буде виконуватись динамічно.

1.2 CSS

CSS (Cascading Style Sheets) - це мова стилів, яка використовується для задання вигляду і форматування веб-сторінок, написаних за допомогою HTML або інших мов розмітки. CSS дозволяє розділити структуру та зовнішній вигляд веб-сторінки, що полегшує роботу з дизайном та покращує її маневреність.

Вигляд елементів:

– *Основна функція CSS* - це визначення стилів для різних елементів на веб-сторінці, таких як кольори, шрифти, розміри, відступи, рамки тощо. Наприклад, ви можете використовувати CSS для зміни кольору тексту в параграфі або фонового кольору заголовка.

– *Каскадування та спадкоємність*. У CSS існує принцип каскадування, що дозволяє визначати стилі більш конкретно для окремих елементів або застосовувати їх згідно із загальними правилами. Також існує концепція спадкоємності, коли стилі батьківського елемента успадковуються дочірніми елементами.

– *Селектори.* CSS використовує селектори для вибору елементів, до яких будуть застосовані певні стилі. Селектори можуть бути класами, ідентифікаторами, тегами, а також псевдокласами та псевдоелементами, що дозволяє точно вибирати елементи для стилізації.

– *Модульність та повторне використання*. CSS дозволяє створювати окремі файлі стилів, які можна використовувати на різних сторінках веб-сайту. Це сприяє модульності та полегшує управління стилями, а також забезпечує можливість повторного використання вже налаштованих стилів.

1.3 JavaScript

JavaScript (JS) - це мова програмування, яка використовується для надання динамічного функціоналу веб-сторінкам. Вона є однією з ключових технологій веб-розробки і дозволяє створювати інтерактивні елементи, анімацію, валідацію форм, обробку подій і багато іншого.

Клієнтська скриптова мова. JavaScript виконується на стороні клієнта, тобто веб-браузері користувача. Це дозволяє реагувати на події користувача, такі як натискання кнопок, наведення курсора, а також взаємодію зі структурою HTML і стилями CSS.

Динамічний контент. JavaScript дозволяє додавати, видаляти або змінювати елементи HTML, а також змінювати їх властивості, такі як текст, стилі, атрибути тощо. Це дозволяє створювати динамічний контент, який може адаптуватися до дій користувача без необхідності перезавантаження сторінки.

Обробка подій. JavaScript дозволяє призначати обробники подій, які реагують на різні дії користувача або на зміни в середовищі веб-сторінки. Це дозволяє створювати веб-додатки з реактивною взаємодією, такі як валідація форм, анімація, завантаження динамічного контенту тощо.

Розробка веб-додатків. JavaScript широко використовується для розробки веб-додатків, включаючи односторінкові додатки (SPA), ігри, соціальні мережі, редактори тексту, електронні магазини та багато іншого. Багато веб-фреймворків і бібліотек, таких як React, Angular, Vue.js, допомагають прискорити процес розробки веб-додатків на JavaScript.

1.4 ReactJs

React.js - це бібліотека JavaScript, призначена для розробки інтерфейсів користувача (UI). Вона створена компанією Facebook і використовується для створення веб-додатків з високою реактивністю та швидкістю. React зосереджується на компонентах, які дозволяють розбивати веб-сторінку на невеликі, незалежні частини, що їх легко керувати та перевикористовувати.

Компонентна архітектура. Одним з основних принципів React є компонентна архітектура, що означає, що весь інтерфейс розбивається на невеликі, самодостатні компоненти. Ці компоненти можуть бути складені разом для створення складних інтерфейсів. Це полегшує розробку, тестування та підтримку коду.

Віртуальний DOM. React використовує віртуальний DOM (Document Object Model), що дозволяє ефективно маніпулювати великою кількістю елементів на сторінці. Замість того, щоб маніпулювати реальним DOM напряму, React спочатку змінює віртуальний DOM, а потім автоматично оновлює реальний DOM тільки там, де зміни відбулися.

JSX синтаксис. React використовує спеціальний синтаксис під назвою JSX, який дозволяє вбудовувати HTML-подібний код безпосередньо в JavaScript. Це полегшує розробку та розуміння коду, а також дозволяє створювати компоненти більш зрозумілими та компактними.

Реактивність та швидкість. Завдяки віртуальному DOM та ефективному алгоритму оновлення, React забезпечує швидке відображення змін на сторінці без необхідності перезавантаження. Це дозволяє створювати інтерактивні та реактивні додатки з відмінною продуктивністю.

Розширюваність та екосистема. React має велику та активну спільноту розробників, що сприяє розвитку різноманітних розширень, бібліотек та інструментів для розробки. Також існує багато сторонніх бібліотек, таких як Redux,

React Router, Material-UI, які розширюють можливості React та полегшують розробку.

1.5 NodeJs

Node.js - це середовище виконання JavaScript, побудоване на движку JavaScript V8, що розроблений Google для веб-браузера Chrome. Node.js дозволяє виконувати JavaScript поза контекстом браузера, тобто на сервері. Воно надає можливість розробникам створювати серверні додатки на JavaScript, що спрощує розробку веб-додатків, що використовують як клієнтську, так і серверну частини.

Основні особливості Node.js:

– *Асинхронний та подієвий режим роботи*. Однією з ключових особливостей Node.js є його асинхронний та подієвий режим роботи. Це означає, що Node.js може обробляти багато запитів одночасно, не чекаючи завершення попередніх операцій, що підвищує продуктивність сервера.

– *Платформонезалежність*. Node.js підтримується на різних операційних системах, таких як Windows, macOS та різні дистрибутиви Linux, що дозволяє розробникам створювати крос-платформенні додатки.

– *Широкі можливості розширення*. Node.js має широкий вибір модулів та пакетів, які дозволяють розширювати його функціональність. npm (Node Package Manager) - це централізований репозиторій для пакетів Node.js, який дозволяє розробникам легко встановлювати, оновлювати та управляти залежностями в їхніх проектах.

– *Висока продуктивність*. Завдяки використанню движка V8, Node.js забезпечує високу продуктивність і швидкість виконання JavaScript коду. Це особливо важливо для серверних додатків, які повинні обробляти великий потік запитів.

– *Підтримка WebSocket і HTTP/2.* Node.js підтримує протоколи WebSocket і HTTP/2, що дозволяє створювати веб-додатки з реальним часом, які можуть передавати дані між клієнтом і сервером без затримок.

2 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

2.1 Створення та ініціалізація усіх папок та файлів

Створення файлів та папок, а також їх ініціалізація для роботи з кодом, є важливою частиною розробки програмного забезпечення. Ось детальний огляд цього процесу:

– *Організація проекту*. Створення папок допомагає вам організувати ваш проект. Зазвичай, ви створюєте основну папку проекту, яка містить підпапки для різних складових програми, таких як код, зображення, стилі CSS тощо. Це допомагає зберігати ваші файли організовано та легко знаходити їх у майбутньому.

– *Розділення логічних компонентів*. Файли допомагають у розділенні коду на логічні компоненти. Наприклад, ми можемо мати окремі файли для різних функцій, класів або модулів вашої програми. Це полегшує зміну, тестування та розуміння нашого коду.

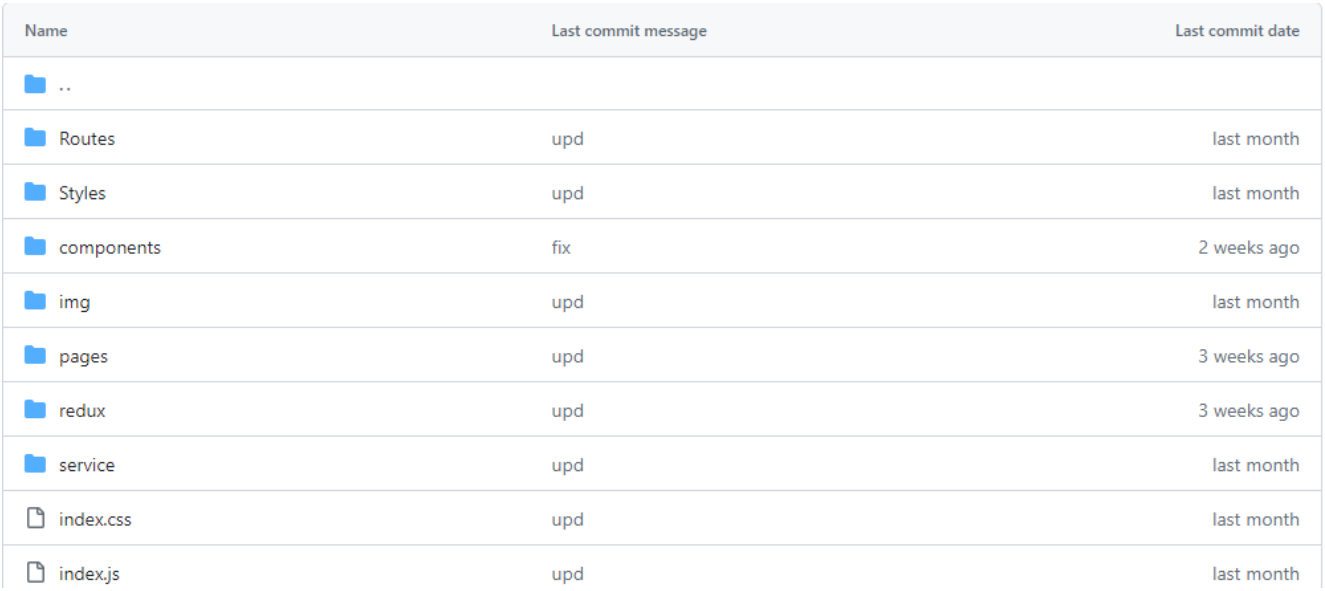
– *Керування версіями*. Папки та файли грають ключову роль у системах керування версіями, таких як Git. Вони використовуються для структурування та зберігання змін вашого коду, що дозволяє нам працювати з різними версіями програми та спільно працювати з іншими розробниками.

– *Модульність та повторне використання коду*. Розділення коду на файли допомагає нам створювати більш модульні програми. Це сприяє повторному використанню коду та полегшує впровадження змін, оскільки ми можемо змінювати лише певні компоненти програми без впливу на інші частини.

– *Підтримка різних середовищ*. Папки та файли можуть допомогти у роботі з різними середовищами розробки та різними версіями програми. Наприклад, ми можемо мати окремі файли конфігурації для різних середовищ, таких як розробка, тестування та виробництво.

Повинно бути надано посилання на головний файл, що буде

використовуватись, як батьківський файл для подальшої розробки. Для цього у файлі “index.html” потрібно надати посилання на файл “index.js”. Отримавши всі потрібні файли, можна розпочинати роботу.



Name	Last commit message	Last commit date
..		
Routes	upd	last month
Styles	upd	last month
components	fix	2 weeks ago
img	upd	last month
pages	upd	3 weeks ago
redux	upd	3 weeks ago
service	upd	last month
index.css	upd	last month
index.js	upd	last month

Рисунок 2.1 – Репозиторій

Ось такий вигляд має репозиторій, у якому будуть знаходитись усі ключові папки та файли, що будуть використані в подальшому для роботи.

2.2 Підключення бібліотек

Розробка повинна початинатись із праці з файлом “index.js”, адже він служить вхідною точкою для React додатка, який буде виконувати функції ініціалізації та монтажу головного компонента нашого додатка.

Такий вигляд матиме файл “index.js” після додавання бібліотек та ініціалізації:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import { App } from 'components/App';
```

```
import { Provider } from 'react-redux';
import { persistor, store } from './redux/store';
import { PersistGate } from 'redux-persist/integration/react';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter basename="/dut_diplom">
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <App />
      </PersistGate>
    </Provider>
  </BrowserRouter>
);
```

Розберемо крок за кроком , що відбувається:

1. *Імпорт бібліотек:*
   - *“React”.* Імпортується основна бібліотека React;
   - *“ReactDOM”.* Імпортується ReactDOM, який використовується для віртуального монтажу React-компонентів в DOM;
   - *“BrowserRouter”.* Імпортується компонент “BrowserRouter” з пакету “react-router-dom", який надає маршрутизацію для додатку;
   - *“App”.* Імпортується головний компонент додатку;
   - *“Provider”.* Імпортується “Provider” з бібліотеки “react-redux”, який дозволяє вам надати Redux store додатку;
   - *“persistor”, “store”.* Імпортуються “persistor” та “store” з файлу “./redux/store”, який містить конфігурацію Redux store.
   - *“PersistGate”.* Імпортується “PersistGate” з пакету “redux-persist/integration/reac”`, який дозволяє очікувати завершення персистентного відновлення.
2. *Монтаж головного компонента:*
   - *“ReactDOM.createRoot(document.getElementById('root')).render(...)”.*

Цей рядок створює кореневий React компонент і монтує його в DOM елемент з ідентифікатором 'root'.

3. Створення маршрутизатора та Redux store:

– *<BrowserRouter basename="/dut_diplom">*. Використовується компонент BrowserRouter, який надає контекст маршрутизатора для додатку з базовим шляхом '/dut_diplom'.

– *<Provider store={store}>*. Використовується “Provider”, щоб надати Redux store додатку.

– *<PersistGate persistor={persistor}>*. Використовується `PersistGate`, щоб забезпечити персистентне відновлення даних перед монтажем головного компонента додатку <App />.

4. Монтаж головного компонента додатку:

– *<App />*. Використовується головний компонент додатку <App />, який монтується всередині “Provider” та “PersistGate”.

2.3 Створення середовищ для подачі заяв та адміністрування

Створимо папку “pages”, у цій папці будуть зберігатись наші сторінки. Для цих сторінко ми створимо окремі папки, “Home” та “Catalog” відповідно.

Папка “Home” буде містити файли, що будуть відповідати за початкову сторінку, де ми зможемо подати заяву. Основний код, який буде виконуватись у поточній папці, має назву “Home.jsx”. Розберемо основні моменти у ньому:

– *Імпорт бібліотек і компонентів.* Цей файл імпортує різні ресурси з React та інших місць, таких як картинки та компоненти модальних вікон.

– *Компонент “Home”.* Це функціональний компонент, який представляє домашню сторінку веб-додатка. Він містить різні елементи, такі як заголовок, кнопки, імпортовані зображення тощо.

– *Стан компоненту за допомогою "useState".* Використовуються хуки стану React для визначення стану компоненту. Наприклад, "openMainModal" та "openAdminModal" вказують, чи відкриті модальні вікна для студента та адміністратора відповідно.

– *Функції "toglMainModal" та "toglAdminModal".* Ці функції змінюють стан "openMainModal" та "openAdminModal", відповідно, при кліку на кнопки.

– *Використання Redux з "useSelector".* Застосування функції "useSelector" дозволяє компоненту зчитувати дані зі стану Redux.

– *Рендеринг модальних вікон.* У компоненті "Home" використовуються компоненти "StudentModal" та "AdminModal" для відображення модальних вікон для студента та адміністратора відповідно.

Простими словами, цей код виконується, щоб створити домашню сторінку веб-додатка, який містить кнопки для подачі заяви та для входу як адміністратор. Крім того, він включає можливість відкривати модальні вікна для подачі заяви студентом або для входу як адміністратор.

Тепер розберемо папку "Catalog", ця папка служить, щоб побачити заяви, які були отримані. У данній папці файл "Catalog.jsx" буде виконувати ці функції. Розберемо код детальніше:

– *Імпорт бібліотек і компонентів*. Код імпортує різні компоненти, такі як "StudentsList", "Loader", "CatalogHeader", "Filter", а також функції та селектори з Redux;

– *Використання Redux з "useDispatch" та "useSelector".* Використовуються хуки "useDispatch" та "useSelector" для взаємодії зі станом Redux. "useDispatch" використовується для виклику action creator, який відправляє запит на отримання заяв на поселення. "useSelector" використовується для вибору певних частин стану Redux для використання в компоненті;

– *Використання ефекту "useEffect".* Використовується хук "useEffect" для того, щоб при монтуванні компонента викликати функцію "allStatementsThunk()", яка завантажує всі заяви на поселення;

– *Фільтрація заяв.* Код містить функції “filteredStatements” та “facultiFilteredStatements”, які відповідають за фільтрацію заяв за певними критеріями, такими як курс і факультет. Після фільтрації дані передаються у компонент “StudentsList”;

– *Рендеринг компонентів*. У компоненті “Catalog” відображаються компоненти “CatalogHeader”, “Filter”, “StudentsList”, а також компонент “Loader”, якщо дані ще завантажуються.

Підсумуємо, що цей код дозволить адміністратору переглянути список заяв на поселення в гуртожиток, фільтрувати їх за різними критеріями, наприклад: курс, кафедра.

У самомі кінці цього блоку ми додамо дві кнопки у папку “Home”, щоб ми могли і подати заяву, і зайти, як адміністратор.

2.4 Маршрутиризація

Маршрутизація в JavaScript - це процес навігації користувача між різними сторінками або відображення різного вмісту на веб-сайті залежно від URL-адреси. Це може бути досягнуто за допомогою різних методів, таких як використання історії браузера, використання хешей в URL-адресі або за допомогою сторонніх бібліотек для маршрутизації. Основна мета маршрутизації - забезпечити користувачеві зручний спосіб переміщення між різними частинами веб-додатка або сайту.

У цьому проекті налаштовуються маршрути по сайту у файлі “App.jsx”.

Демонстрація маршрутиризаці у коді:

```
import { Route, Routes } from 'react-router-dom';
import { StyledContainer } from 'Styles/Container.styled';
import { Suspense, lazy } from 'react';
import { Loader } from './Loader/Loader';
```

```
import PrivateRoute from 'Routes/Private';
import RestrictedRoute from 'Routes/Restricted';

const Home = lazy(() => import('pages/Home/Home'));
const Catalog = lazy(() => import('pages/Catalog/Catalog'));

export const App = () => {
  return (
    <StyledContainer>
      <Suspense fallback={<Loader />}>
        <Routes>
          <Route
            path="/"
            element={
              <PrivateRoute>
                <Home />
              </PrivateRoute>
            }
          />
          <Route
            path="/catalog"
            element={
              <RestrictedRoute>
                <Catalog />
              </RestrictedRoute>
            }
          />
        </Routes>
      </Suspense>
    </StyledContainer>
  );
};
```

Для визначення маршрутів використовується:

– *компонент “Routes” для визначення колекції маршрутів;*

– *два маршрути за допомогою компонента "Route".* Перший маршрут відповідає шляху "/" і відображає компонент "Home" в обгортці "PrivateRoute", що дозволяє доступ тільки авторизованим користувачам. Другий маршрут відповідає шляху "/catalog" і відображає компонент "Catalog" в обгортці "RestrictedRoute", що дозволяє доступ тільки обмеженим користувачам.

2.5 Створення модальних вікон для студента та адміністратора

Після всіх маніпуляцій з кодом, які були описані вище ми створили логіку для подання наших заяв, але нам потрібно оформити сам процес подання заяви і редагування отриманих заяв.

Наразі додаток має ось такий вигляд:

Рисунок 2.1 – Головна сторінка

Отже, розпочинаємо роботу для створення модальних вікон для кнопок. Створюємо папку "modal" у компонентах фреймворку React. Треба розбити модальні вікна, для цього створимо файли "StudentModal.jsx" та "AdminModal.jsx". Перший файл буде відповідати за студентське модальне вікно, інше - за вікно адміна.

Для більш чіткого розуміння роботи коду, який буде далі нам потрібно ознайомитись із структурою DOM дерева нашого документа, адже у коді ми будемо напряму взаємодіяти з нею.

DOM (Document Object Model) - це стандарт, який представляє структуру документа HTML або XML у вигляді дерева об'єктів, які можуть бути маніпульовані за допомогою скриптових мов, таких як JavaScript. DOM представляє кожен елемент сторінки, як об'єкт з властивостями та методами, що дозволяє програмно змінювати вміст, структуру та стилі сторінки. Взаємодія з DOM дозволяє динамічно оновлювати вміст сторінки, реагувати на події користувача та створювати інтерактивні веб-додатки.

Код буде визначати компонент “AdminModal”, що буде відображати модальне вікно для авторизації адміністратора:

1. Обробники подій:

– Функція “hendleKeyDown” обробляє натискання клавіші "Escape" і закриває модальне вікно відповідно.

```
const hendleKeyDown = event => {
    if (event.code === 'Escape') {
      closeModal();
    }
  };
```

– Функція “closeOnBackdrop” закриває модальне вікно, якщо користувач клікає на фоні модального вікна (поза областю контенту).

```
  const closeOnBackdrop = event => {
    if (event.target === event.currentTarget) {
      closeModal();
    }
```

2. Хуки форми “useForm” та методи реєстрації, обробки подання та скидання форми:

– Використовується для створення та керування формою в модальному вікні.

– За допомогою хуків форми здійснюється реєстрація полів, обробка подання форми та скидання її.

```
const {
  register,
  handleSubmit,
  reset,
  formState: { errors },
} = useForm();
```

3. Обробник подання форми “onSubmit”:

– Викликається при поданні форми. Якщо введений пароль дорівнює 'admin', відправляється дія для встановлення статусу адміністратора.

– Також дані форми виводяться у консоль, а форма скидається після подання.

```
const onSubmit = data => {
  if (data.password === 'admin') {
    dispatch(handlIsAdmin());
  }
  console.log(data);
  reset();
};
```

4. Відображення контенту модального вікна:

– Включає в себе заголовок, форму для введення пароля, кнопку відправки форми та індикатор завантаження, який відображається, якщо дані ще завантажуються.

```
return (
    <Overlay onClick={closeOnBackdrop}>
      <ModalContent>
        <ExitBtn onClick={closeModal}>
          <RxCross2 className="cross" />
        </ExitBtn>
```

```
        <p className="admin-title">Авторизація адміністратора</p>
        <AdminForm onSubmit={handleSubmit(onSubmit)}>
          <div className="form__group field">
            <div className="form__label">
              <label>Введіть пароль</label>
              {errors.password   &&   <span   className="error-message">{'
*'}</span>}
            </div>
            <LoginInp
              {...register('password', { required: true })}
              type="password"
            />
          </div>
          <SubmitBtn type="submit" className="send-btn">
            Відправити
          </SubmitBtn>
        </AdminForm>

        {isLoading && (
          <div className="modal-loader">
            <Blocks
              visible={true}
              height="100"
              width="100"
              ariaLabel="blocks-loading"
            />
          </div>
        )}
      </ModalContent>
    </Overlay>
  );
}
```

Отже, після створення цього коду, виникає модальне вікно для входу в режим адміністратора. На живій сторінці це виглядає так:

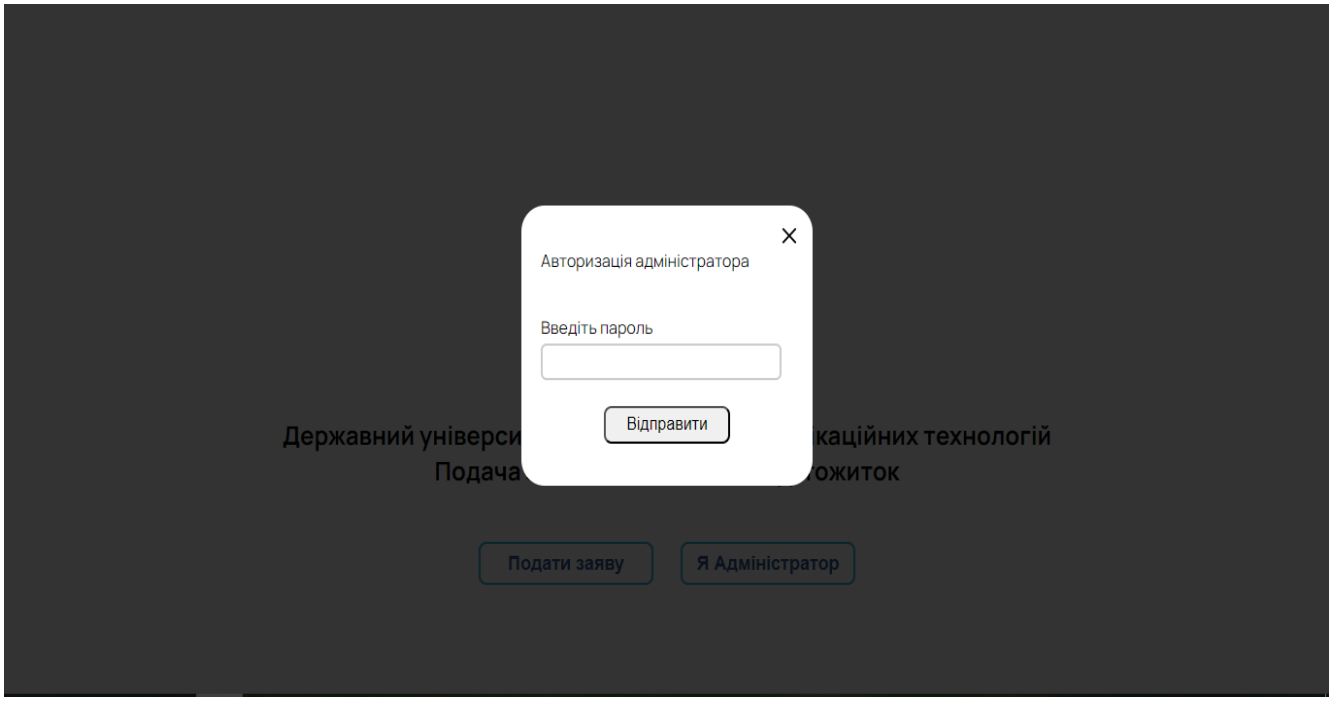


Рисунок 2.2 – Вікно авторизації

Тепер буде виконане модальне вікно для студента. Тут використані ці самі методи, що використовувались для створення модального вікна адміністратора, а саме: слухачі подій, що будуть закривати модальне вікно. Тому не має сенсу писати про це повторно, зупинимось детальніше на коді, що виконуватиметься саме для подачі заяви. Файл “StudentModal.jsx” містить код, щоб вирішити цю задачу:

```
return (
    <Overlay onClick={closeOnBackdrop}>
      <ModalContent>
        <ExitBtn onClick={closeModal}>
          <RxCross2 className="cross" />
        </ExitBtn>

        <RegisterPage close={closeModal} />
```

```
        {isLoading && (
          <div className="modal-loader">
            <Blocks
              visible={true}
              height="100"
              width="100"
              ariaLabel="blocks-loading"
            />
          </div>
        )}
      </ModalContent>
    </Overlay>
  );
}
```

Цей фрагмент коду представляє фрагмент JSX, що відображається під час відкриття модального вікна. Давайте розберемо його:

1. *Компонент “Overlay”:*
   – Це обгортка навколо вмісту модального вікна, яка відповідає за затемнення фону поза областю модального вікна;
   – При кліку на цю область, викликається функція “closeOnBackdrop”, яка закриває модальне вікно.
2. *Компонент “ModalContent”:*
   – Це контейнер, який містить весь вміст модального вікна;
   – У ньому розміщуються кнопка закриття (“ExitBtn”), зображення хрестика (“RxCross2”) та вміст сторінки реєстрації (“RegisterPage”).
3. *Компонент “ExitBtn”:*
   – Це кнопка, яка відповідає за закриття модального вікна;
   – При кліку на неї викликається функція `closeModal`, яка закриває модальне вікно.
4. *Компонент “RegisterPage”:*
   – Це компонент, який містить вміст сторінки реєстрації;

– Зазвичай він містить поля для введення даних користувача та кнопку для підтвердження реєстрації.

5. Умовне відображення завантажувача:

– Якщо змінна “isLoading” має значення “true”, відображається блок з індикатором завантаження;

– Це дозволяє користувачеві знати, що дані завантажуються або обробляються.

2.6 Створення форми для подачі заяви

Зараз у проекті наявні модальні вікна, які будуть вспливати при натисканні на відповідні кнопки, але студент має описати себе за різними крітеріями, наприклад: ПІБ, курс, група, номер телефону тощо. Для цього ми використаємо форми, щоб додати поля для опису студента, які будуть переноситись у модальне вікно.

Щоб реалізувати цю дію у коді, потрібно додати компонент “RegisterPage” у файл “AppelForm.jsx”, де буде відбуватись наша робота з формою. Виконується це таким чином:

```
const {
  register,
  handleSubmit,
  reset,
  watch,
  formState: { errors },
} = useForm();
```

Тепер ми отримали пусту форму, але нам потрібно набір пунктів з інформацією про студента, наприклад: курс, група, факультет тощо.

Перед тим, як задавати динамічно поля з різною інформацією, ми повинні навісити функції, форма могла працювати і змінювати свої значення, як тільки користувач ввів якусь інформацію.

```
const handleInputChange = event => {
    const { value } = event.target;
    const filteredValue = value.replace(/[^0-9+-]/g, '');
    setInputValue(filteredValue);
  };

  const handleFacultyChange = e => {
    const faculty = e.target.value;
    setSelectedFaculty(faculty);
  };

  const onSubmit = data => {
    dispatch(addStatementThunk(data));
    console.log(data);
    reset();
    close();
  };
```

Всього потрібно три функції. Розглянемо їх детальніше:

– *Функція "handleInputChange"* призначена для обробки змін у введеному значенні. Вона отримує значення з події, фільтрує його, залишаючи тільки цифри, а потім встановлює це відфільтроване значення як значення вводу. Це нам знадобиться для полів з числовим форматом, наприклад, щоб ввести номер телефону;

– *Функція "handleFacultyChange"* встановлює обраний факультет при зміні значення вибраного елемента (зазвичай це випадаючий список або інший елемент вводу). Це буде корисно, щоб відфільрувати групи за факультетом, адже у кожного факультету свої унікальні групи і вони не можуть повторюватись;

– *Функція "onSubmit"* відправляє дані форми до магазину Redux за допомогою функції "addStatementThunk(data)", після чого скидає форму за допомогою "reset()" і закриває діалогове вікно чи модальне вікно за допомогою "close()".

Після цього будемо відмалювати поля, використовуючи даний синтаксис:

– Форма для отримання ПІБ особи:

```
return (
    <StyledForm onSubmit={handleSubmit(onSubmit)}>
      {Object.keys(errors).length ? (
        <p className="info">
          Поля відмічені <span className="error-message">*</span> є
          обов'язковими для заповнення
        </p>
      ) : (
        ''
      )}
      <div className="form__group field">
        <div>
          <label className="form__label">ПІБ</label>
          {errors.name && <span className="error-message">{' *'}</span>}
        </div>
         <input
          {...register('name', { required: true })}
          type="text"
          className={errors.name ? 'form-input input-error' : 'form-input'}
        />
      </div>
```

– Форма для отримання електронної пошти:

```
<div className="form__group field">
        <div>
          <label className="form__label">Електронна пошта</label>
          {errors.email && <span className="error-message">{' *'}</span>}
        </div>
```

```
        <input
          {...register('email', { required: true })}
          type="email"
          className={errors.email ? 'form-input input-error' : 'form-input'}
        />
      </div>
```

– Форма для отримання номеру телефона, де використовується перша функція, котру вказали - “handleInputChange”:

```
<div className="form__group field">
        <div>
          <label className="form__label">Номер телефону</label>
          {errors.phone && <span className="error-message">{' *'}</span>}
        </div>
        <input
          {...register('phone', { required: true })}
          type="text"
          value={inputValue}
          onChange={handleInputChange}
          autocomplete="off"
          placeholder="+380953032100"
          maxLength={12}
          className={errors.phone ? 'form-input input-error' : 'form-input'}
        />
      </div>
```

– Вибір факультету, де використовується функція “handlefacultyChange”, що містить випадаючі значення:

```
<div className="form__group field">
        <div>
          <label className="form__label">Факультет</label>
          {errors.specialty    &&    <span    className="error-message">{'
*'}</span>}
        </div>
        <select
          {...register('specialty', { required: true })}
```

```
          onChange={handleFacultyChange}
          className={
            errors.statement ? 'form-select input-error' : 'form-select'
          }
        >
          <option value="">Оберіть факультет</option>
          {faculties.map(option => (
            <option key={option.value} value={option.value}>
              {option.label}
            </option>
          ))}
        </select>
      </div>
```

– Отримання курсу:

```
  <div className="info-wrapper">
          <div className="form__group field">
            <div>
              <label className="form__label">Курс</label>
              {errors.course    &&    <span    className="error-message">{'
  *'}</span>}
            </div>

            <select
              {...register('course', { required: true })}
              className={
                errors.statement ? 'form-select input-error' : 'form-select'
              }
            >
              <option value="">Оберіть курс</option>
              {courses.map(course => (
                <option key={course} value={course}>
                  {course}
                </option>
              ))}
            </select>
          </div>
```

– Вибір групи, який з’являється лише після вибору факультету та курсу:

```
<div className="form__group field">
          <div>
            <label className="form__label">Група</label>
            {errors.group && <span className="error-message">{' *'}</span>}
          </div>
```

```
          <select
            {...register('group', { required: true })}
            className={
              errors.statement ? 'form-select input-error' : 'form-select'
            }
          >
            <option value="">Оберіть групу</option>
            {selectedFaculty &&
              groupsByFacultyAndCourse[selectedFaculty][
                parseInt(watch('course'))
              ]?.map(group => (
                <option key={group} value={group}>
                  {group}
                </option>
              ))}
          </select>
        </div>
      </div>
```

– Фінальна форма, де студент зможе написати текст звернення, щоб отримати кімнату:

```
<div className="form__group field">
        <div>
          <label className="form__label">Текст звернення</label>
          {errors.statement    &&    <span    className="error-message">{'
*'}</span>}
        </div>
        <textarea
          {...register('statement', { required: true })}
          type="text"
          className={
            errors.statement
              ? 'form-input statement input-error'
              : 'form-input statement'
          }
        />
      </div>
```

Тепер модульне вікно відобразить цю розмітку і матиме такий вигляд:

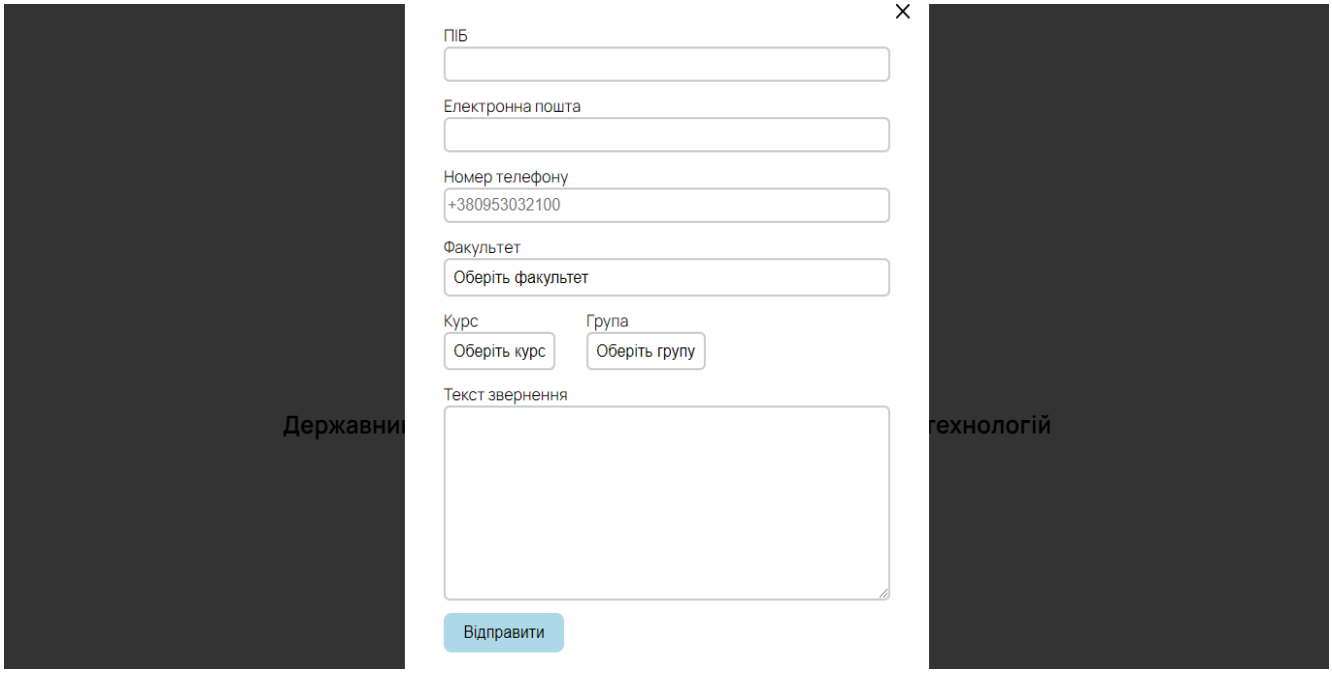


Рисунок 2.3 – Вікно форми реєстрації заявки

3 ВЗАЄМОДІЯ З БЕКЕНДОМ

3.1 Збереження даних

У нашому додатку ми постійно будемо отримувати заяви з боку студентів, отже потрібно їх зберігати, щоб була можливість для подальшої взаємодії з ними. У самому браузері існують дві технології зберігання даних - localStorage та sessionStorage, але у цих технологіях є великий мінус, а саме: зберігання даних всього до 5 – 10 МБ. Ці технології не зможуть впоратись з таким об’ємом інформації, тому для таких типів проектів, де потрібно зберігати дуже великі обсяги даних, такі як інформація про студентів, їхні роботи, інформація про користувачів тощо, використовують реляційні бази даних.

3.2 Керування логікою

Після того, як студент надіслав заявку, вона відправляється на сервер, потім приходить з сервера у базу даних. Це має ось такий вигляд:

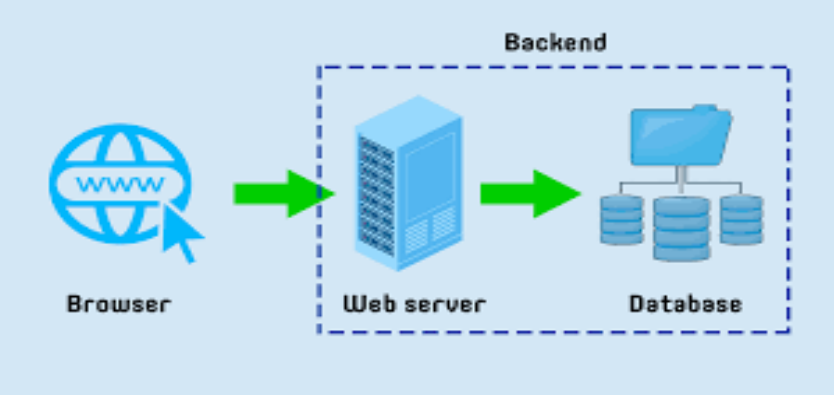


Рисунок 3.1 – Логіка програми

Отже потрібно прописати логіку, що буде відповідати за стан заяви студентів, включаючи завантаження, додавання та видалення заявок, а також

фільтрація інформації всередині заяви. Також потрібна взаємодія з сервером для отримання та оновлення даних через асинхронні запити.

Асинхронні запити - це запити, які виконуються в асинхронному режимі, тобто запити, які не блокують виконання інших операцій в програмі чи на сторінці веб-сайту, поки вони чекають на відповідь від сервера чи іншого джерела даних. Веб-додатки часто взаємодіють з сервером для отримання чи оновлення даних, та для цього вони використовують асинхронні запити.

В якості асинхронних запитів, буде використаний Redux, який є бібліотекою, що відповідає за стан додатків. Він поділяється на основні концепції:

– *Store (Сховище).* Це об'єкт, що містить увесь стан додатку. Всі дані, які використовуються у вашому застосунку, зберігаються в цьому сховищі.

– *Actions (Дії).* Це об'єкти, які вказують, що відбувається в додатку. Вони є єдиним джерелом інформації, яка потрапляє в сховище.

– *Reducers (Редуктори).* Це чисті функції, які приймають поточний стан та дію, і повертають новий стан. Вони відповідають за зміну стану відповідно до дій.

– *Middleware (Проміжне ПЗ).* Це шари, які можна використовувати для реалізації додаткової логіки навколо дій. Наприклад, middleware може використовуватися для обробки асинхронних дій, логування, маршрутизації тощо.

У репозиторії була створена папка redux, яка буде містити всі redux-файли. Потрібно розпочати з store.js, адже це головний файл Redux, який буде служити в якості сховища та передавати дані у локальне сховище браузера.

Код, який буде записаний у store,js:

```
import { combineReducers, configureStore } from '@reduxjs/toolkit';
import {
  persistStore,
  persistReducer,
  FLUSH,
  REHYDRATE,
  PAUSE,
  PERSIST,
```

```
  PURGE,
  REGISTER,
} from 'redux-persist';
import storage from 'redux-persist/lib/storage';

import { studentReducer } from './studentsReduser';
import { adminReducer } from './adminReduser';

const AdminConfig = {
  key: 'admin',
  version: 1,
  storage,
  whitelist: ['isAdmin'],
};

const rootReducer = combineReducers({
  admin: persistReducer(AdminConfig, adminReducer),
  statement: studentReducer,
});

export const store = configureStore({
  reducer: rootReducer,
  middleware: getDefaultMiddleware =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE,
REGISTER],
      },
    }),
});

export let persistor = persistStore(store);
```

У цьому файлі налаштовується Redux-стор для управління станом застосунку та забезпечує його збереження у локальному сховищі браузера за допомогою бібліотек Redux Toolkit та redux-persist. У ньому виконуються наступні дії:

– Імпорт необхідних бібліотек та функцій, таких як “combineReducers”, “configureStore” з Redux Toolkit, а також функцій для збереження стану з “redux-persist”;

– Налаштування конфігурації для збереження стану редуктора адміністратора за допомогою “persistReducer”;

– Об'єднання редукторів у кореневий редуктор за допомогою “combineReducers”;

– Налаштування та створення Redux-стору за допомогою “configureStore”, де вказується кореневий редуктор та middleware;

– Створення об'єкта “persistor”, який зберігає посилання на store.js для подальшого збереження його стану у локальному сховищі.

3.3 Створення редуктора для студентів

Редуктори - це чисті функції, які приймають поточний стан і дію, і повертають новий стан. В контексті Redux, редуктори відповідають за зміну стану додатку відповідно до дій, які відбуваються у додатку.

Основна задача редукторів полягає в управлінні станом додатку. Коли дія спровокована користувачем або програмним кодом, редуктори отримують цю дію та змінюють відповідний фрагмент стану додатку відповідно до логіки, що визначена в них. Це дозволяє зберігати всі дані додатку у одному місці та дозволяє однозначно відстежувати, які дії змінюють стан.

В основі Redux лежить принцип незмінності даних, тобто старий стан не змінюється напряму. Замість цього, редуктори повертають новий об'єкт стану, який є результатом змін у старому стані, зроблених відповідно до прийнятої дії.

Основні функції редукторів:

– *Управління станом*. Редуктори відповідають за управління частинами стану додатку.

– *Чистота та передбачуваність*. Чисті функції редукторів забезпечують передбачувані та детерміновані зміни у стані, що сприяє відтворюваності та легкості відлагодження.

– *Принцип незмінності даних*. Redux використовує принцип незмінності даних, тому редуктори повинні створювати новий об'єкт стану для кожної зміни.

– *Однозначність змін*. Зміни у стані додатку здійснюються через дії, які редуктори інтерпретують однозначно, що дозволяє просто відстежувати та розуміти, які зміни стану відбуваються.

Редуктори в Redux відіграють ключову роль у управлінні станом додатку, забезпечуючи прозору та передбачувану модель зміни даних.

Оскільки робота відбувається зі студентами, тому потрібно створити у папці redux файл studentReducer.js. У фрагменті коду, що знаходиться у studentReducer.js створюється "slice" за допомогою функції “createSlice” з Redux Toolkit. Цей "slice" називається “studentSlice” і містить початковий стан для частини додатку, яка відповідає за заявки студентів.

– Початковий стан. Об'єкт `INITIAL_STATE` містить початковий стан, який включає в себе масив “statements” для зберігання заявок, прапорець “isLoading” для позначення процесу завантаження, поле “error” для зберігання помилок, а також поля для фільтрації заявок.

– Редуктори для зміни стану. Визначені редуктори “handlFiltration” та “handlFacultFiltration”, які відповідають за зміну фільтрів для заявок студентів.

– Додаткові обробники для асинхронних операцій. Використовується “extraReducers”, щоб додати обробники для асинхронних дій, таких як завантаження всіх заявок, додавання нової заявки та видалення заявки.

– Обробка статусів асинхронних операцій. Використовуються обробники для обробки різних статусів асинхронних операцій, таких як “pending”, “fulfilled” та “rejected”, відповідно до різних типів асинхронних дій.

Простими словами цей "slice" забезпечує можливість змінювати стан додатку для операцій з заявками студентів та обробляти асинхронні операції, такі як завантаження, додавання та видалення заявок.

3.4 Додавання та видалення заяв

Щоб додавати та видаляти заяви, був створений файл studentsOperations.js у папці redux. У файлі знаходяться прописані функціх, які виконують цю роботу. Код, що виконує ці дії:

```
import { createAsyncThunk } from '@reduxjs/toolkit';
import {
  requestAllStatements,
  addStatement,
  deleteStatement,
} from 'service/studentsAPI';

export const allStatementsThunk = createAsyncThunk(
  'students/allStatements',
  async (_, thunkAPI) => {
    try {
      const data = await requestAllStatements();
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.message);
    }
```

```
  }
);

export const addStatementThunk = createAsyncThunk(
  'students/addStatement',
  async (student, thunkAPI) => {
    try {
      const data = await addStatement(student);
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.message);
    }
  }
);

export const deleteStatementThunk = createAsyncThunk(
  'students/deleteStatement',
  async (id, thunkAPI) => {
    try {
      const data = await deleteStatement(id);
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.message);
    }
  }
);
```

– Створення асинхронної дії для отримання всіх заявок студентів “allStatementsThunk”:

Використовується функція “createAsyncThunk”, яка створює асинхронну дію з певною назвою та логікою виконання.

Ця асинхронна дія названа “students/allStatements”.

У функції обробника асинхронної дії, яка передається як другий аргумент, виконується запит на сервер за допомогою функції “requestAllStatements()”. У разі успішного виконання запиту, дані повертаються, а у разі помилки - викликається “thunkAPI.rejectWithValue()” з повідомленням про помилку.

– Створення асинхронної дії для додавання нової заявки студента “addStatementThunk”:

Аналогічно, створюється асинхронна дія для додавання нової заявки студента.

Ця дія названа `'students/addStatement'`.

У функції обробника асинхронної дії виконується запит на сервер за допомогою функції “addStatement(student)”. Результат запиту повертається як дані, або в разі помилки - викликається “thunkAPI.rejectWithValue()”.

– Створення асинхронної дії для видалення заявки студента “deleteStatementThunk”:

Аналогічно, створюється асинхронна дія для видалення заявки студента.

Ця дія названа “students/deleteStatement”.

У функції обробника асинхронної дії виконується запит на сервер за допомогою функції “deleteStatement(id)”. Результат запиту повертається як дані, або в разі помилки - викликається “thunkAPI.rejectWithValue()”.

Отже, цей код створює асинхронні дії для взаємодії з сервером при отриманні, додаванні та видаленні заявок студентів.

3.5 Запити на сервер

Після налаштування базових маніпуляцій із заявами, потрібно налаштувати запити, які будуть приходити на сервер. У репозиторії була створена папка service, яка містить файл studentsApi.js. Цей файл містить такий вигляд коду:

```
import axios from 'axios';

export const instance = axios.create({
  baseURL: `https://65f907e2df15145246105050.mockapi.io/students/`,
});

export const requestAllStatements = async () => {
  const { data } = await instance.get(`appeal`);
  return data;
};

export const addStatement = async statement => {
  const { data } = await instance.post(`appeal`, statement);
  return data;
};

export const deleteStatement = async statementId => {
  const { data } = await instance.delete(`appeal/${statementId}`);
  return data;
};
```

У цьому фрагменті коду використовується бібліотека Axios для виконання HTTP-запитів до вказаного API. Більш детально переглянемо код:

а) Створення екземпляру Axios:

1) Викликається функція “create” з Axios для створення екземпляру Axios.

2) Вказується базовий URL, який буде використовуватися для всіх запитів до API.

б) Функція “requestAllStatements”:

1) Виконує GET-запит до ресурсу “appeal” за допомогою методу “instance.get”.

2) Отримані дані повертаються з функції.

в) Функція “addStatement”:

1) Виконує POST-запит до ресурсу “appeal”, передаючи дані заявки як частину запиту.

2) Отримані дані (зазвичай, нова заявка з унікальним ідентифікатором) повертаються з функції.

г) Функція “deleteStatement”:

1) Виконує DELETE-запит до ресурсу “appeal/{statementId}”, де “statementId” - це ідентифікатор конкретної заявки, яку потрібно видалити.

2) Отримані дані (зазвичай, підтвердження про видалення) повертаються з функції.

Підбиваючи підсумки, цей код визначає функції для виконання конкретних типів запитів до API (GET, POST, DELETE) та використовує Axios для виконання цих запитів.

3.6 Створення редуктора для адміністратора

У папці redux створимо файл adminReducer.js, який буде містити код, що буде відповідати за стан адміністратора. Код, який прописаний у файлі:

```
import { createSlice } from '@reduxjs/toolkit';

const INITIAL_STATE = {
  isAdmin: false,
};

const adminSlice = createSlice({
  name: 'admin',
  initialState: INITIAL_STATE,
  reducers: {
    handlIsAdmin(state, action) {
      state.isAdmin = !state.isAdmin;
    },
  },
});
```

```
export const { handlIsAdmin } = adminSlice.actions;
export const adminReducer = adminSlice.reducer;
```

У цьому фрагменті коду використовується функція “createSlice” з Redux Toolkit для створення “slice”, який відповідає за керування статусом адміністратора в додатку. Розберемо цей код:

а) Початковий стан: об'єкт “INITIAL_STATE” містить початковий стан для “slice”, який включає в себе логічне значення “isAdmin”, що вказує, чи є користувач адміністратором.

б) Створення “slice” з допомогою “createSlice”:

1) - Функція “createSlice” використовується для створення “slice”, який містить редуктор для зміни стану та дії для виклику цих змін.

2) - У цьому випадку “slice” названий “adminSlice” та має назву “admin”.

3) - Початковий стан встановлюється в “INITIAL_STATE”.

в) Редуктор для зміни стану:

1) - У визначенні “slice” вказані редуктори для зміни стану. У даному випадку, редуктор “handlIsAdmin” відповідає за зміну значення “isAdmin”. Він переключає значення “isAdmin” на протилежне.

г) Експорт дій та редуктора:

1) - За допомогою деструктуризації експортуються дії “handlIsAdmin”.

2) - Редуктор “adminReducer” також експортується для використання в Redux-сторі.

Отже, цей “slice” дозволяє змінювати стан, що відповідає за адміністративні права, у додатку за допомогою одного редуктора, який переключає значення “isAdmin”.

4 РЕНДЕРИНГ СТОРІНКИ

4.1 Розробка фільтрації заяв

Щоб зробити фільтрацію заяви, у нашому випадку за курсом та факультетом, потрібно створити реактовський компонент. Для цього ми створимо компонент “Filter” і у ньому створимо filter.jsx, що буде відповідати за процеси фільрації. Для того, щоб прописати фільрацію всередині нам потрібно імпортувати функції “handlFiltration” та “handlFacultFiltration”, які знаходяться у redux редукторі. Вони відповідають за фільтрацію по курсу та факультету відповідно. Ось так це виглядає:

```
import {
  handlFiltration,
  handlFacultFiltration,
} from '../../redux/studentsReduser';
```

Далі створимо контрольовану форму, що буде використовувати React-hook “useForm”:

```
const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();

  const onSubmit = data => {
    dispatch(handlFiltration(data));
    dispatch(handlFacultFiltration(data.specialty));
    console.log(data);
  };
```

При відправці форми буде викликатись функція “onSubmit”, яка відправляє дані форми до Redux store для подальшої обробки.

Адміністратор може вибрати діапазон курсів за допомогою двох випадаючих списків: “Від” та “До”. На живій сторінці це буде виглядати ось так:

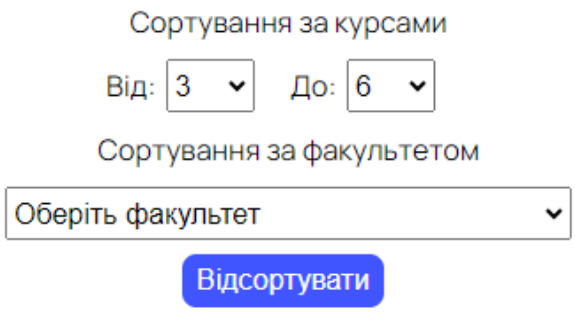


Рисунок 4.1 – Сортування за курсом

Код, що був прописаний для виконання фільтрації:

```
return (
    <Form onSubmit={handleSubmit(onSubmit)}>
      <p className="filter-title">Сортування за курсами</p>
      <div className="filter-title">
        <label className="filter-lable">
          Від:
          <select
            {...register('start')}
            defaultValue={1}
            className="filter-select"
          >
            {[...Array(6)].map((_, index) => (
              <option key={index + 1} value={index + 1}>
                {index + 1}
              </option>
            ))}
          </select>
        </label>
        <label className="filter-lable">
          До:
          <select
            {...register('end')}
            defaultValue={6}
            className="filter-select"
          >
            {[...Array(6)].map((_, index) => (
              <option key={index + 1} value={index + 1}>
                {index + 1}
              </option>
            ))}
          </select>
        </label>
      </div>
```

Сортування за факультетом відбувається аналогічно. Ми отримуємо випадаючий список із переліком факультетів:

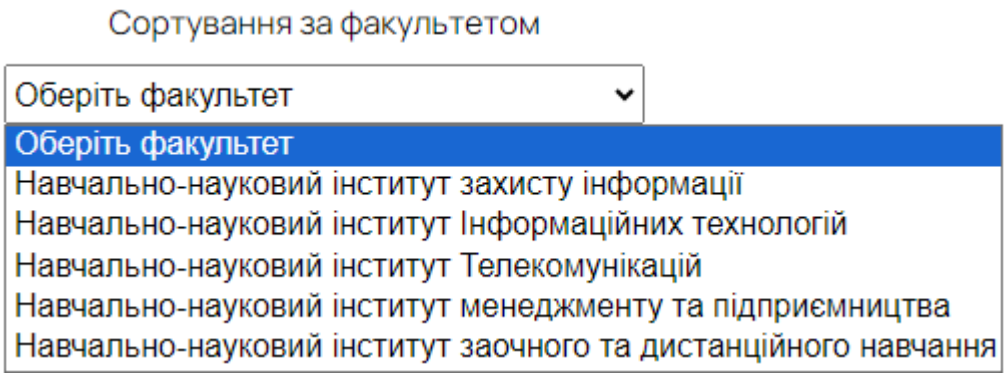


Рисунок 4.1. – Сортування за факультетом

Приклад коду для реалізації фільтрації за факультетом:

```
<p className="filter-title">Сортування за факультетом</p>
      <div className="form-group field">
        <select
          {...register('specialty')}
          className={
            errors.statement ? 'form-select input-error' : 'form-select'
          }
        >
          <option value="">Оберіть факультет</option>
          {faculties.map(option => (
            <option key={option.value} value={option.value}>
              {option.label}
            </option>
          ))}
        </select>
      </div>
      <button type="submit" className="filter-submit">
        Відсортувати
      </button>
    </Form>
  );
};
```

Обрані значення з обох фільтрацій передається до функцій “handlFiltration” та “handlFacultFiltration” та передаються до Redux store для проведення сортування.

Натиснувши кнопку “Відсортувати”, буде викликана функція “handleSubmit”, яка параметром прийме колбек функцію “onSubmit”. Приклад коду:

```
const onSubmit = data => {
```

```
    dispatch(handlFiltration(data));
    dispatch(handlFacultFiltration(data.specialty));
    console.log(data);
  };

  return (
    <Form onSubmit={handleSubmit(onSubmit)}>
```

4.2 Рендеринг заяв

У вікні адміністратора ми будемо приймати заяви, як список, де кожна заява буде окремим елементом списку. Щоб виконати цю дію, нам потрібно створити компонент StudentList.jsx, який буде виконувати роль відображення списку заяв студентів у вигляді окремих елементів списку.

Основне, що буде виконано у коді:

– Перевіряється, чи передані дані у “StudentStatements”. Якщо передані, то вони мапляться, і для кожної заяви створюється окремний елемент списку “StudentItem”.

– Кожна заява отримує дані “StudentsItemData” із масиву “StudentStatements”, також унікальний ключ “key={statement.id}”.

Код, що це реалізовує:

```
import { StudentItem } from 'components/CarsItem/StudentItem';
import { StyledCarList } from './StudentList.styled';
// import { exactCarThunk } from '../../redux/studentsOperations';

export const StudentsList = ({ studentStatements }) => {
  //   const disputch = useDispatch();

  // const vievModal = id => {
  //   // disputch(exactCarThunk(id));
  //   openModal();
  // };

  return (
    <StyledCarList>
      {studentStatements !== null &&
        studentStatements.map(statement => {
          return (
```

```
            <StudentItem
              studentItemData={statement}
              key={statement.id}
              // openModal={() => vievModal(statement.id)}
            />
          );
        })}
    </StyledCarList>
  );
};
```

Після відтворення списку, залишається фінальний крок - робота з елементом списку(заявою). Потрібно реалізувати відображення заяви з усіма даними, що були надіслані студентом. Всі маніпуляції будуть проходити у компоненті StudentItem.jsx.

Прописуємо копіювання та видалення заяви:

```
import { useDispatch } from 'react-redux';
import { deleteStatementThunk } from '../../redux/studentsOperations';
import { StyledStudentItem } from './StudentItem.styled';
import Notiflix from 'notiflix';

export const StudentItem = ({ studentItemData }) => {
  const dispatch = useDispatch();

  const { name, course, specialty, statement, email, id, phone, group } =
    studentItemData;

  const deleteStatement = id => {
    dispatch(deleteStatementThunk(id));
  };

  const copyStatementText = id => {
    const statementText = document.getElementById(id).textContent;
    navigator.clipboard
      .writeText(statementText)
      .then(() => Notiflix.Notify.success('Текст заяви скопійовано!'))
      .catch(error => console.error('Помилка копіювання:', error));
  };
```

Динамічне побудування заяви у кабінеті адміністратора:

```
return (
    <StyledStudentItem>
      <p className="item-description">ПІБ студента:</p>
      <p className="item-text">{name}</p>
      <p className="item-description">Електронна пошта:</p>
      <p className="item-text">{email}</p>
      <p className="item-description">Номер телефону:</p>
      <p className="item-text">{phone}</p>
```

```
        <div className="text-wrapper">
          <div>
            <p className="item-description">Група:</p>
            <p className="item-text">{group}</p>
          </div>
          <div>
            <p className="item-description">Курс:</p>
            <p className="item-text">{course}</p>
          </div>
          <div>
            <p className="item-description">Спеціальність:</p>
            <p className="item-text">{specialty}</p>
          </div>
        </div>
        <p className="item-description">Текст заяви:</p>
        <p className="statement-text" id={id}>
          {statement}
        </p>
        <div className="button-wrapper">
          <button
            onClick={() => deleteStatement(id)}
            type="button"
            className="item-buttons"
          >
            Видалити
          </button>
          <button
            type="button"
            onClick={() => copyStatementText(id)}
            className="item-buttons"
          >
            Копі
          </button>
        </div>
      </StyledStudentItem>
    );
  };
```

4.3 Фінальний результат

При потраплянні на сторінку, студент з’являється на домашній сторінці, де є дві кнопки: “Подати заяву” і “Я адміністратор”.

Рисунок 4.3 – Остаточний вигляд головної сторінки

Якщо користувач є студентом, він може подати заяву, натиснувши кнопку “Подати заяву”, після цього з’явиться модальне вікно, щоб заповнити дані:

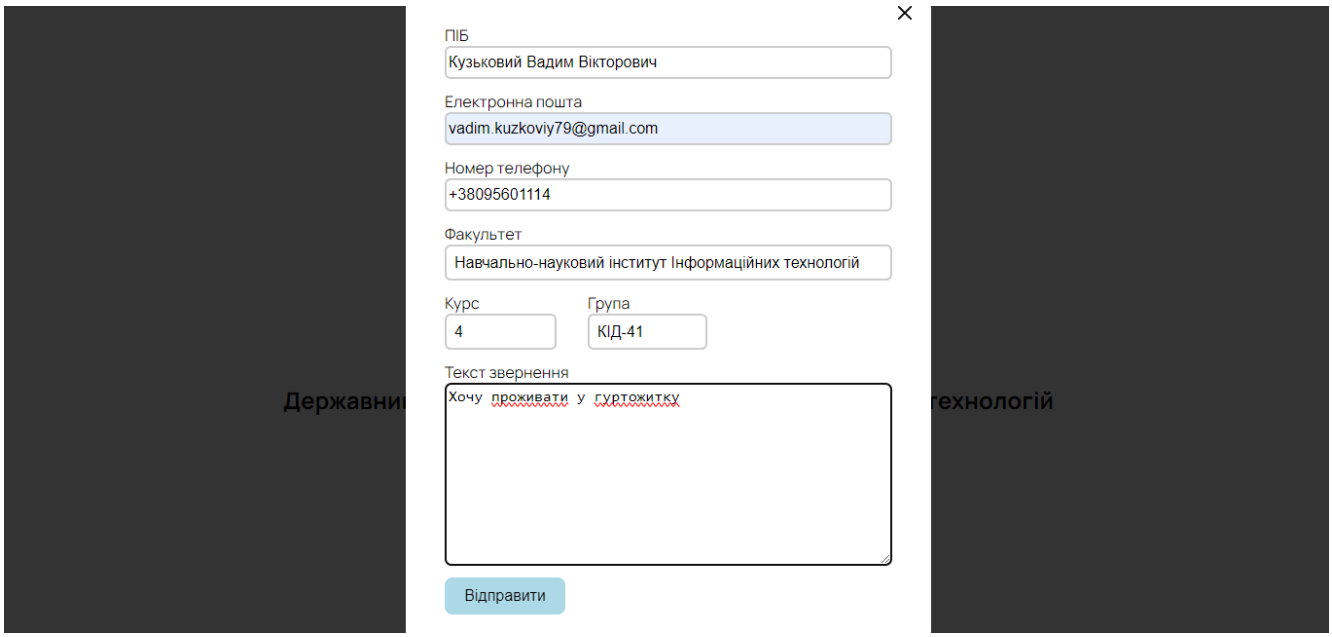


Рисунок 4.4 – Заповнена заява

Коли всі дані були заповнені, натискаємо кнопку “Відправити”, після цієї дії у правому кутку екрану відображається повідомлення, що заява була відправлена:

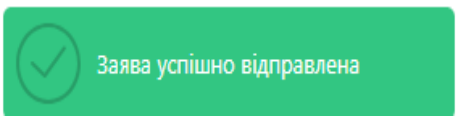


Рисунок 4.5 – Успішна відправка заяви

Потім адміністратор може переглянути заяву, щоб виконати цю дію, потрібно натиснути кнопку “Я адміністратор”, ввести пароль “admin”. Після цього ми знаходимось у кабінеті адміністратора та можемо бачити наші заяви:

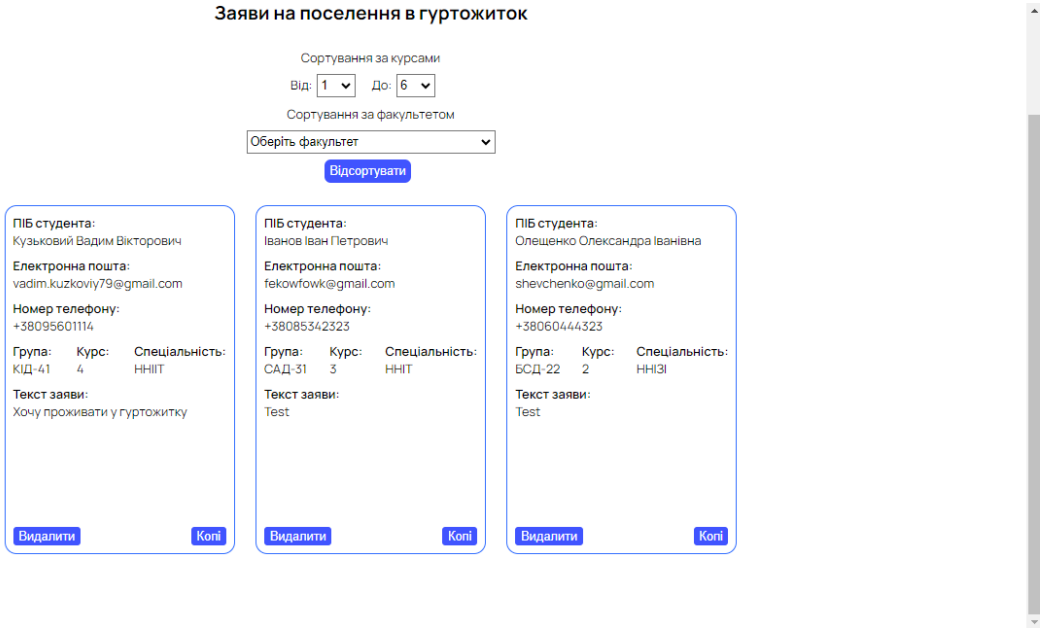


Рисунок 4.6 – Кабінет адміністратора

ВИСНОВКИ

Була розроблена електронна система обробки заяв, яка дала змогу значно підвищити швидкість та ефективність обробки даних. Автоматизація процесів зменшує навантаження на співробітників, знижує ймовірність помилок та забезпечує швидке прийняття рішень. Також веб-система забезпечує доступність для користувачів з будь-якого місця і в будь-який час за умови наявності інтернет-з'єднання. Це особливо важливо для студентів, які подають заяви, оскільки вони можуть робити це зручно і без необхідності відвідування університету.

Використання веб-технологій дозволяє знизити витрати на обробку заяв шляхом автоматизації процесів, зменшення потреби в паперових документах та скорочення часу на обробку даних. Веб-системи сприяють підвищенню прозорості процесів обробки заяв, що дозволяє заявникам відслідковувати статус своїх заяв в режимі реального часу і отримувати своєчасну інформацію про хід їх розгляду.

ПЕРЕЛІК ПОСИЛАНЬ

1. Axios Docs. *Axios*. URL: https://axios-http.com/uk/docs/intro Notiflix Library. *Notiflix.* URL: https://www.npmjs.com/package/notiflix
2. React GettingStart. *React.* URL: https://react.dev/learn
3. Importing and Exporting Components. *React.* URL: https://react.dev/learn/importing-and-exporting-components
4. Writing Markup with JSX. *React.* URL: https://react.dev/learn/writing-markup-with-jsx
5. Passing Props to a Component. *React.* URL: https://react.dev/learn/passing-props-to-a-component
6. React Components. *React.* URL: https://react.dev/learn/your-first-component
7. React DOM. *React.* URL: https://react.dev/learn/understanding-your-ui-as-a-tree
8. React Rendering Lists. *React.* URL: https://react.dev/learn/rendering-lists
9. Деструктиризація у JavaScript. *JavaScript.* URL: https://uk.javascript.info/destructuring-assignment
10. Замикання JS. *JavaScript.* URL: https://foxminded.ua/zamykannia-javascript/
11. Redux React. *React.* URL: https://redux.js.org/
12. React Loader. *React.* URL: https://www.npmjs.com/package/react-loader-spinner
13. React Route. *React.* URL: https://reactrouter.com/en/main
14. React Router. *React.* URL: https://reactrouter.com/en/main/route/route
15. React Route Laze. *React.* https://reactrouter.com/en/main/route/lazy
16. Redux Persist. *React.* URL: https://www.npmjs.com/package/redux-persist
17. Web Vitals. *JavaScript.* URL: https://web.dev/articles/vitals
18. Function Expression. *JavaScript.* URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function

19. Regular Expressions JavaScript. *JavaScript.* URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions

20. Прототипи та наслідування. *JavaScript.* URL: https://uk.javascript.info/prototypes

21. Bundler Vite. *Vite.* URL: https://vitejs.dev/guide/why

22. Фреймоврки JavaScript. *JavaScript.* URl: https://foxminded.ua/freimvorky-javascript/

23. Огляд фреймворків. *JavaScript.* URL: https://dou.ua/forums/topic/34739/

24. Node JS API. *NodeJS.* URL: https://nodejs.org/docs/latest/api/

25. Listener NodeJS. *NodeJS.* URL: https://nodejs.org/docs/latest/api/events.html#emitteraddlistenereventname-listener

26. Rest API. *NodeJS.* URL: https://habr.com/ru/articles/483202/

27. Local Storage. *JavaScript.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

28. Paren Elements. *HTML.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/parent

29. Scrollbars. *JavaScript.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/scrollbars

30. Fetch. *JavaScript.* URL: https://uk.javascript.info/fetch

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Інформаційних технологій
Кафедра Комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

на тему: «Розробка програмної системи обробки заяв на поселення у гуртожиток»

Виконав студент групи КІД-41
Кузьковий Вадим Вікторович

Керівник кваліфікаційної роботи:
Розмаїтий Дмитро Олегович,
викладач кафедри КІ

Київ – 2024

Реферат

- *Мета роботи* – розробити програмну систему обробки заяв на поселення в гуртожиток
- *Об'єкт дослідження – розробка програмної системи*
- *Предмет дослідження – програмна система*
- *Короткий зміст роботи: - опис репозиторію, робота із компонентами React та відображення результату на живій сторінці. Створення фільтрації для сортування заяв студентів*

Актуальність

Робота є актуально за рахуного того, що кожного року до університету вступає багато студентів, велика частина з них не місцеві. Програмна система дасть змогу подавати заяву онлайн. Також вона спростить роботу особам, що відповідають за розселення студентів, адже все буде в єдиному реєстрі, де є функції фільтрації, що допоможуть краще відібрати студентів, котрі будуть проживати у гуртожитку.

Інструменти, які були використані

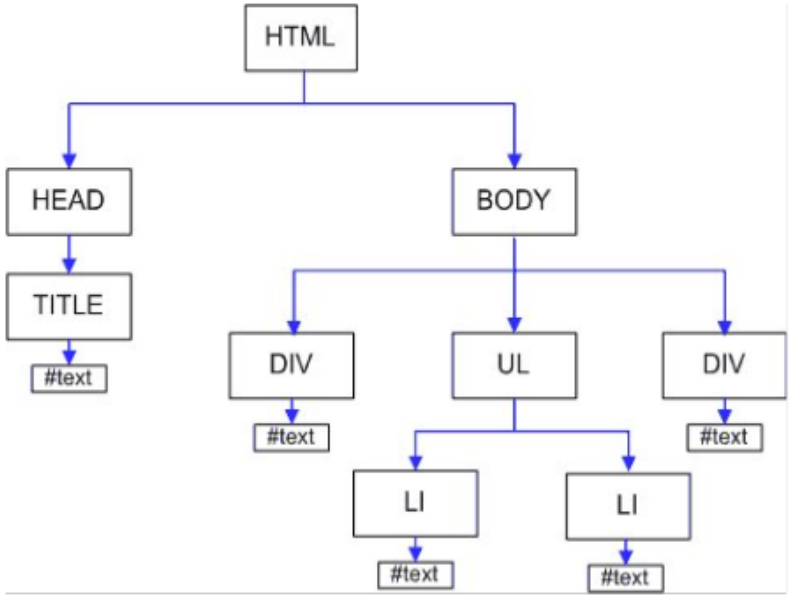
- HTML (HyperText Markup Language) є стандартною мовою розмітки, що використовується для створення веб-сторінок і веб-додатків.
- CSS (Cascading Style Sheets) – це мова стилів, що використовується для опису зовнішнього вигляду та форматування HTML-документів.
- JavaScript (JS) – це мова програмування, яка використовується для додавання інтерактивності та динамічності до веб-сторінок.
- React – це бібліотека JavaScript, розроблена Facebook, яка використовується для створення користувацьких інтерфейсів, особливо односторінкових додатків.
- Node.js – це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код на сервері. Воно побудоване на рушії V8 від Google, що використовується у браузері Chrome.

Підключення бібліотек для роботи

- Зручність у використанні
- Повторне використання коду
- Спрощення складних завдань
- Підвищення продуктивності
- Спільнота та підтримка
- Кросбраузерна сумісність

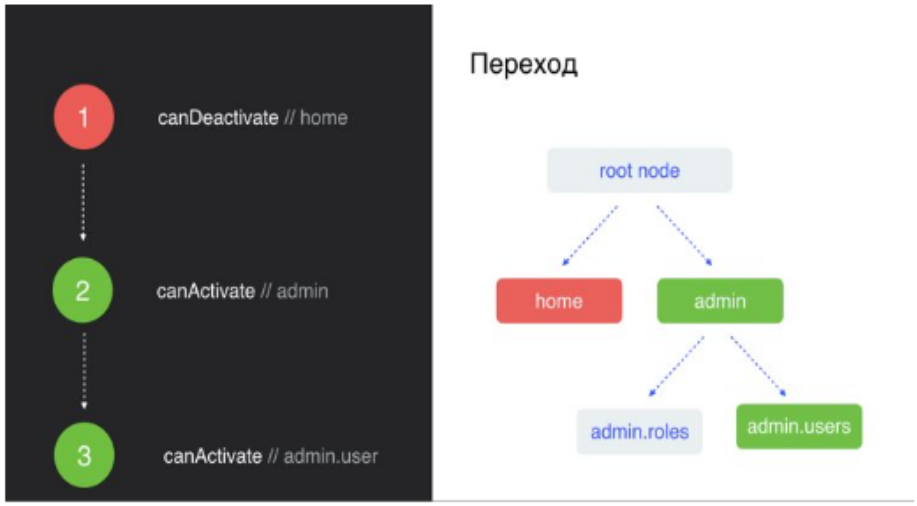
DOM

DOM (Document Object Model) – це програмний інтерфейс для HTML та XML документів. Він представляє структуру документа у вигляді дерева об'єктів, що дозволяє програмам змінювати вміст, структуру та стиль документа.



Маршрутизація у REACT

Маршрутизація у React дозволяє створювати односторінкові додатки з різними "сторінками" чи маршрутами, які динамічно завантажують відповідний вміст без повного перезавантаження сторінки. Основна бібліотека для маршрутизації у React – це `react-router-dom`.



Керування логікою у react

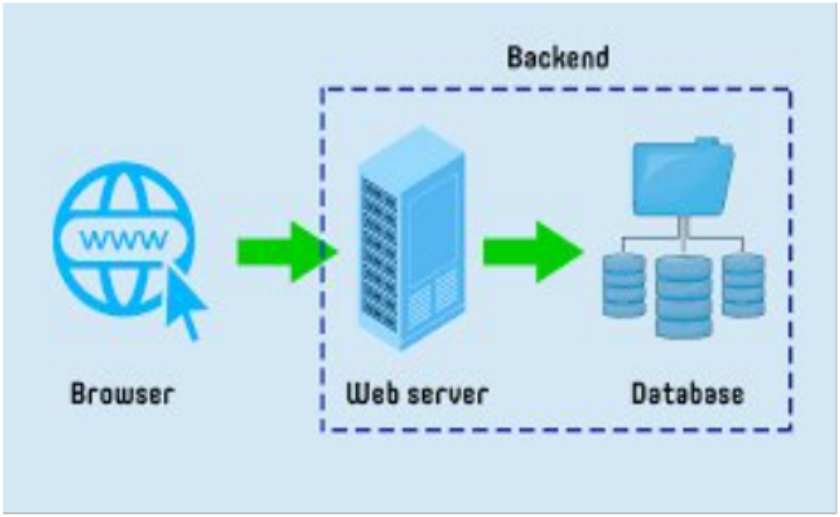
У середовищі React для керування логікою запитів на сервер використовуються різні підходи та бібліотеки:

- **Fetch API**: Вбудований в браузер Fetch API дозволяє виконувати HTTP-запити на сервер та обробляти відповіді. Це простий та зручний спосіб для виконання запитів, особливо для простих сценаріїв.
- **Axios**: Axios – це бібліотека для виконання HTTP-запитів у JavaScript, яка працює як у браузері, так і у середовищі Node.js. Вона забезпечує простий та зрозумілий інтерфейс для роботи з запитами та відповідями.
- **GraphQL**: GraphQL – це мова запитів та середовище виконання, що дозволяє клієнтам запитувати тільки ті дані, які їм потрібні. У React GraphQL може бути інтегрований з допомогою бібліотек, таких як Apollo Client.

Продовження про керування логікою

У дипломній роботі використовувалась бібліотека Axios. Запити отримувались через метод .get.

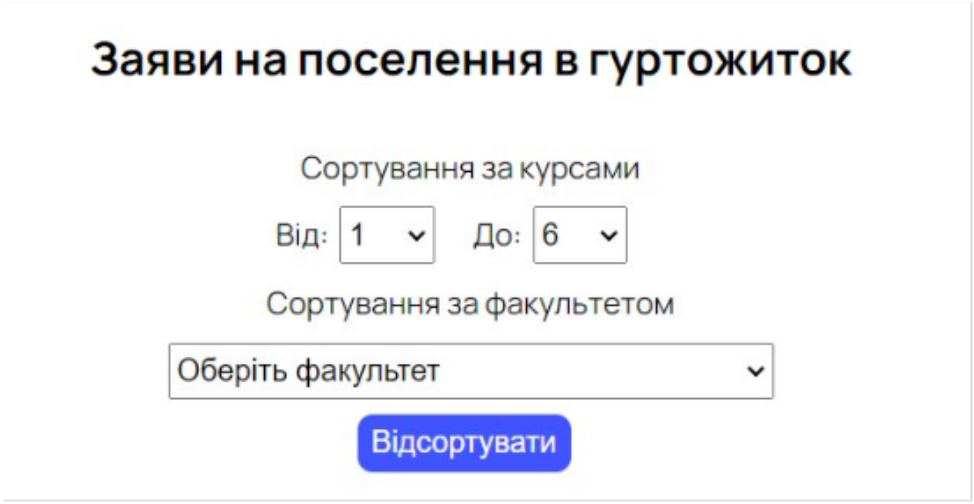
На рисунку візуально представлена логіка запитів на сервер:



Фільтрація заяв

Фільтрація заяв була розроблена для того, щоб особи, які будуть перевіряти заяви, могли відфільтрувати заяви за певними критеріями, а саме: фільтрація за курсом та факультетом.

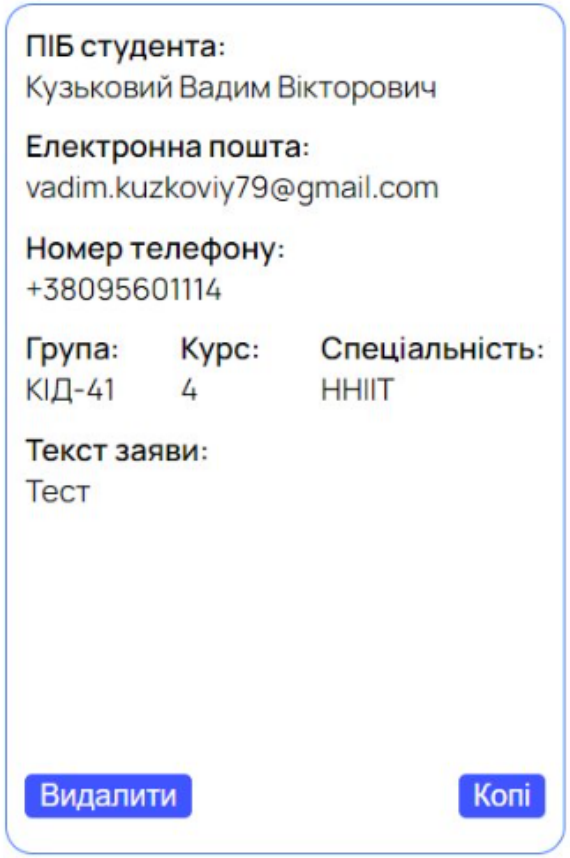
Реалізація на живій сторінці:



Рендеринг заяв

Рендеринг заяв потрібен для того, щоб після подання заяви студентом або після фільтрації заяв, об'єкт з данними студента міг відмалюватись на живій сторінці.

Це матиме такий вигляд:



Висновки

Використання веб-технологій дозволяє знизити витрати на обробку заяв шляхом автоматизації процесів, зменшення потреби в паперових документах та скорочення часу на обробку даних. Веб-системи сприяють підвищенню прозорості процесів обробки заяв, що дозволяє заявникам відслідковувати статус своїх заяв в режимі реального часу і отримувати своєчасну інформацію про хід їх розгляду.