МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Сайт для Курінського НВК»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Тиркусова Н.В.**

Студента групи ІН – 64-8 **Варчак В.М.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК, СЕКЦІЇ ІКТ

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 р.

ЗАВДАННЯ до випускної роботи

Студента четвертого курсу, групи ІН-64-8 спеціальності “Інформатика” денної форми навчання Варчака Валерія Михайловича.

Тема: “ Сайт для Курінського НВК”

Затверджена наказом по СумДУ

№_____ от _____2020 р.

Зміст пояснювальної записки: 1) огляд існуючих рішень; 2) вибір методу рішення; 3) інформаційне та програмне забезпечення системи.

Дата видачі завдання “_____”_____2020 р.

Керівник випускної роботи _____ Тиркусова Н.В.

Завдання прийняв до виконання _____ Варчак В.М.

РЕФЕРАТ

Записка: 61 сторінок, 27 рисунків, 8 джерел.

Об’єкт дослідження — Сайт для Курінського НВК

Мета роботи — Розробка сайту для Курінського НВК.

Методи дослідження — односторінковий сайт на основі технологій React, MongoDB, Node.js, Express.

Результати — Створено односторінковий веб-сайт для загальноосвітнього навчального закладу Курінський НВК, де було створено можливість зареєструватися на сайті, створено функціонал управління новинами, класами та користувачами. Готовий веб-сайт завантажено на хостинг .

ВЕБ-САЙТ ШКОЛИ, WEB, MERN, MONGODB,
EXPRESS, REACT, NODE.JS.

ЗМІСТ

ВСТУП

Використання веб-технологій помітно зростає у сфері освіти. Багато шкіл та загальноосвітніх закладів мають свій сайт.

По-перше, сайт є представництвом школи в мережі Інтернет, джерелом інформації про неї, про її викладачів і учнів і т.п.

По-друге, сайт може бути використаний і безпосередньо як інструмент навчального процесу, значно полегшити задачу вчителя. І якщо в процес навчання сайти тільки починають вводити, то як візитну картку школи сайти використовують щосили.

Що стосується того, "кому все це треба і взагалі навіщо?" то відповідь очевидна. Треба, власне, тим, хто має безпосереднє відношення до шкільного життя.

Практична значимість роботи полягає в розробці і створенні шкільного сайту для загальноосвітньої школи Курінь НВК.

Мати школі свій власний сайт це мати власну ЗМІ, і часто краще, ніж звичайні газети, тому що сайт оперативніше, з набагато більшими можливостями. Шкільний сайт - візитна картка школи, елемент освітньої Інтернет-системи, точка взаємодії. Сайт створює нові можливості взаємодії в освітньому процесі. Шкільний сайт - інструмент для всіх хто безпосередньо має відношення до школи. Сайт сприяє підвищенню престижу освітнього закладу.

Актуальність роботи полягає в доступі інформації про школу через веб-сторінку, викликає довіру з боку батьків, підвищує серед школярів престиж власної школи, виховує у школярів патріотизм і почуття приналежності до своєї школи.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд існуючих рішень

На сьогоднішній день інформаційний ринок, що поставляються програмні рішення в обраній сфері пропонує безліч продуктів, дозволяють досягти поставленої мети - розробки web-сайту. Передбачається, що розробляється система повинна бути на багато користувачів і в ній буде передбачено поділ користувачів по ролям.Так як розробляється система призначена для забезпечення інформаційної підтримки, необхідно реалізувати її з урахуванням максимальної доступності звернення до неї.

При такій постановці завдання, найбільш вигідним варіантом реалізації системи є створення веб-сайту або модуля існуючої системи. В цьому випадку з метою зручності наповнення та управління інформацією має здійснюватися через веб-інтерфейс. Також необхідно прагнути до досягнення максимальної автоматизації операцій в системі, так як це економить час доступу до інформації і спрощує роботу користувачів. Даний підхід найбільш раціональний зважаючи забезпечення, таким чином, змоги не закріплюватися за одним робочим місцем. Звернутися до системи, а також керувати нею при наявності відповідних прав можна з будь-якого приміщення, в якому є комп'ютер при наявності доступу в мережу Інтернет

Головне призначення сайту Курінський НВК по бути суті віртуальною школою.

Для користувачів буде доступне:

- Новини – описи усіх подій, які трапилися в школі за останній час;
- Успіхи учнів які будуть з’являться в новинах – талановиті учні, які вигравали предметні олімпіади, брали участь у різноманітних розважальних і спортивних заходах;
- Вчителі – імена вчителів та предмети, котрі вони викладають;

- Інформація про учня його клас та класного керівника;
- Для батьків – розклад уроків їх дітей;
- Розклад уроків для учнів;
- Контакти – місцезнаходження школи.
- Корисні посилання – посилання на різноманітні навчальні сайти;
- Слайдер фото – містить фотографії школи.

1.2 Постановка задачі

Метою роботи є створення сайту для школи.

Основні вимоги до проекту:

- Реалізація у вигляді сайту;
- Авторизація користувача або реєстрація;
- Користувачі повинні мати, принаймні, одну роль з переліку: ученик, викладач, модератор, адміністратор;

Під час реєстрації потрібно буде заповнити потрібну інформацію для користувача.

Під час авторизації необхідно підтвердити свої дані на право користування деякими компонентами сайту

В особистому кабінеті ученика необхідно розмістити данні його аккаунта які він може змінювати, інформація про його клас, класного керівника та поштовий адрес класного керівника.

Функціонал звичайного користувача:

- Можливість перегляду новин;
- Перегляд розкладу;
- Перегляду інформації про вчителів.

Функціонал ролі «Ученик»:

- Можливість перегляду новин;

- Перегляд розкладу;
- Перегляду інформації про вчителів;
- Перегляд інформації про аккаунт.

Функціонал ролі «Вчитель»:

- Можливість перегляду новин;
- Перегляд розкладу;
- Перегляду інформації про вчителів;
- Може назначатися класним керівником;
- Перегляд інформації про аккаунт.

Функціонал ролі «модератор»:

- Можливість перегляду, редагування, зміни, видалення новин;
- Можливість назначення класного керівника для класу, редагування класу, видалення класу;
- Перегляд розкладу;
- Перегляду інформації про вчителів;
- Перегляд інформації про аккаунт;
- Може назначатися класним керівником.

Функціонал ролі «адміністратор»:

- Можливість перегляду, редагування, зміни, видалення новин;
- Можливість назначення класного керівника для класу, редагування класу, видалення класу;
- Перегляд розкладу;
- Перегляду інформації про вчителів;
- Перегляд інформації про аккаунт;
- Може назначатися класним керівником;
- Повинен надавати ролі користувачам, та керувати ними.

Графічну структуру сайту можна побачити на рисунку 1.1.

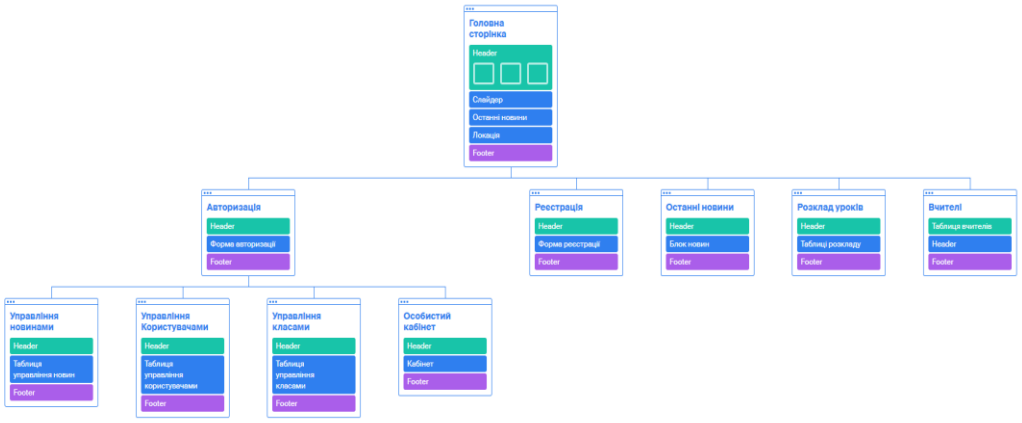

Рисунок 1.1 — Графічна структура сайту

Ескіз головної сторінки можна побачити на рисунку 1.2.

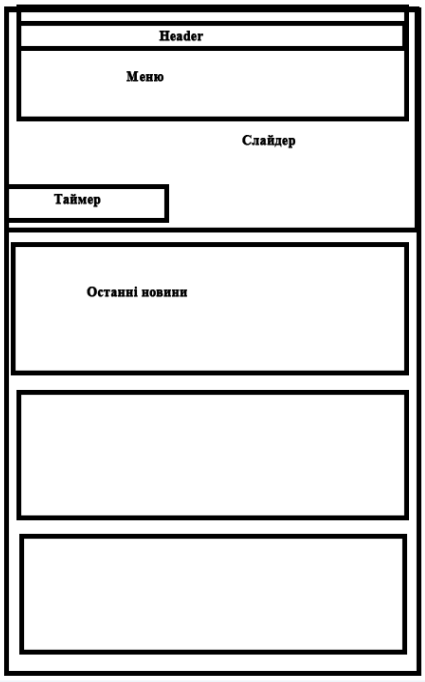

Рисунок 1.2 — Ескіз головної сторінки.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір технологій

Мова програмування, як і будь-яка інша мова призначена для комунікації, тобто зв'язку між що говорить і слухає. У програмуванні промовистою є програміст, а слухачем - інтерпретатор мови, деяка комп'ютерна програма, що розуміє цю мову і виконує дії відповідно до того, що вона зрозуміла. Був час, коли вважалося, що для полегшення спілкування з комп'ютером необхідно створити мову, досить близьку до природньої. Ця ідея, в кінцевому рахунку, не витримала випробувань часом, хоча і породила кілька прекрасних мов програмування. Часто буває, що побічні ефекти якоїсь діяльності перевершують очікування. Для клієнтської частини даного поекту я вирішив використати мову JavaScript. JavaScript, незважаючи на його численні недоліки, можливо, є найкращим мовою програмування для початківців розробників.

Одним з унікальних переваг JavaScript є його поширеність. Ця мова можна зустріти буквально всюди. Він підтримується на всіх операційних системах, у всіх видах браузерів, і на настільних комп'ютерах, і на мобільних пристроях. Дуже важливо ще й те, що JavaScript-додатки працюють без установки їх на комп'ютери користувачів. Зіткнувшись з подібними кошмарами, ви цілком можете оцінити привабливість JavaScript.

Для розробки WEB-сервера була обрана технологія NodeJS тому вона дозволяє прискорити процес реалізації внаслідок того, що використовується один і той же мова програмування, що і в WEB інтерфейсе (JavaScript). Структура WEB-сервера не є складною так як сервер являє собою набір обробників HTTP-запитів і звернень до бази даних. Сервер надає призначеному для користувача інтерфейсу API для безпечного зміни або збереження даних в базі даних.

2.2 Опис технологій для клієнтської частини

JavaScript досить простий у вивченні, аналогічно HTML. Сценарій можна безпосередньо вставляти в документ HTML. Використовуючи JavaScript можна вирішити завдання:

- додавання в документ рядків, що біжать і повідомлень про його зміну;
- зміни форми введення даних і виконання необхідних обчислень;
- відображення повідомлень, призначених для користувача (як на самій сторінці, так і в окремому діалоговому вікні);
- створення анімованих зображень, що змінюються при наведенні на них мишки;
- додавання інтерактивного банера;
- визначення використовуваного браузера і настройки відповідно до нього веб-сторінки;
- виявлення використовуваних впроваджуваних модулів і повідомлення користувача про їх статус.

Наведено лише деякі із загальних завдань, які дозволяє виконати JavaScript. Насправді ж, за допомогою нього вирішуються і більш складні завдання аж до створення окремих додатків. Для більш коммфортного його використання існує дуже багато бібліотек для створення більш сучасного сайту, для цього проекту я використовував бабліотеки React и Redux.

React - библиотека для рендеринга. Бібліотека React є зручним інструментом для розробки web-додатків, який завдяки своїм алгоритмам дозволяє ефективно і швидко відображати користувачеві інформацію.

Кілька років тому розробка інтерфейсних програм здавалася чимось несерйозним. Тепер від таких додатків вимагають більшого, тому складність розробки зросла. Це можна порівняти з ситуацією, коли кіт більше не

поміщається у звичайну коробку, і тепер потрібно зробити для нього нове місце проживання, а це вимагає більше зусиль. Хоча всі процеси так влаштовані: чим далі і більше, тим складніше.

Принципи Redux Згідно з документацією, Redux - це передбачуваний контейнер стану для JavaScript-додатків. Так, це звучить дивно і незрозуміло. Насправді суть Redux в тому, щоб приручити того самого кота з коробки, а точніше зробити так, щоб програми працювали послідовно. Крім цього, у нього є багато інших переваг.

2.3 Опис технологій для серверної частини

Node.JS - серверна JavaScript платформа, що включає інтерпретатор JavaScript, вбудований сервер і базовий набір бібліотек. Платформа надає повністю асинхронну роботу з файлами і мережевими пристроями. За допомогою Node.js ви зможете створювати високопродуктивні масштабовані клієнтські і серверні додатки і сервіси. Особливістю Node.js є каркас, який виконується не в браузері клієнта, а на стороні сервера.

Один з найважливіших плюсів - це асинхронність. Також платформа передбачає власні інструменти, наприклад, тут немає браузерних API, coockie або DOM, зате присутні власні бібліотеки. В цілому використовуються можливості JavaScript. Node.JS від самого початку мав великий набір можливостей, також кожного разу платформа поліпшується. NPM дозволяє постійно розвивати Node.

Але замість очікування виконання коду Node.js він може зайнятися іншими завданнями. Таким чином, програма може звернутися до бази даних сервера, а поки очікується відповідь, обробити інший функції. В результаті одночасно обробляються багато завдань,з малою витратою часу. Це швидко прискорює обмін данних з сервером.

Фреймворк Express.js надає широкий спектр утиліт HTTP, і може похвалитися високою швидкістю і невеликою вагою. Це середовище Node.js

використовується для всіх видів додатків і надає додаткові функції. В основному розробники використовують цей інструмент для живого чату і потокових додатків, оскільки він забезпечує відмінну продуктивність в режимі реального часу.

Таким чином, цей інструмент може бути легко використаний для додавання каналів професійної підтримки на веб-сайт і ефективного спілкування з клієнтами. Більш того, Express може створювати API REST і має просту інтеграцію з базою даних.

Вирішимо наступний важливе питання, яку систему керування базами даних нам краще (зручніше) всього використовувати в нашій розробці інформаційної системи. Мій вибір зупинився на MongoDB.

MongoDB - документоорієнтованих система керування базами з вихідним кодом, яка не потребує опису схеми таблиць. Написана на мові C ++.

СУБД MongoDB проектувалася для зберігання гігантських обсягів даних. Під час налаштування сервера Mongo перевага віддається узгодженості після операції записи всі наступні операції читання витягнуть одне і те ж значення (до наступного оновлення). Ця особливість робить MongoDB привабливою альтернативою для тих, хто має досвід роботи з РСУБД. Крім того, MongoDB підтримує атомарні операції читання-запису, в тому числі инкрементирование значення і запити до вкладених документів. Завдяки використанню JavaScript в якості мови запитів MongoDB підтримує як прості запити, так і складні завдання з розподілом-редукцією.

2.4 Інші фреймворки та інструменти

Javascript фреймворк - це набір методів і функцій, які написані на мові Javascript і готові до використання розробниками. Ці функції спрощують взаємодію веб-додатки з сервером і прискорюють маніпулювання елементами веб-сторінці або в додатку. Основною перевагою використання фреймворків є той факт, що вони забезпечують миттєвий зворотний зв'язок з тими, хто в

даний час взаємодіє з вашим сайтом. На традиційних веб-сайтах без фреймворків початковий контент зберігається на сервері, і будь-який новий контент, Який необхідно завантажити, вимагає перезавантаження страніци.А при використанні фреймворком перезавантажуються тільки потрібні блоки сайту. Такий підхід зараз вживають соціальні мережі, наприклад, Facebook. Якщо ви перебуваєте на сторінці «Новини» і переходите на сторінку «Повідомлення», перевантаження сторінки не відбувається, натомість змінюється кілька блоків на сайті. Тому я вирішив використовувати CSS фреймворк Material UI.

Середовище розробки було вибрано Visual Studio Code - повнофункціональний редактор коду, доступний на Windows, Linux і Mac OS X. VS Code є розширюваним open-source редактором, який можна налаштувати під будь-яке завдання. VS Code побудований на Electron, тому у нього є ті ж переваги і недоліки.

Npm - це менеджер пакетів, що входить до складу Node.js. Він використовує клієнт командного рядка і базу даних, що складається з загальнодоступних і приватних пакетів, відомої як npm registry. Користувачі можуть отримати доступ до бази через сайт або через консоль.

Складальник проекту було обрано Webpack так як він чудово підходить до мого односторінкового сайту.

Webpack дозволяє позбутися від bower і gulp / grunt в додатку, і замінити їх одним інструментом. Замість bower'a для установки і управління клієнтськими залежностями, можна використовувати стандартний Node Package Manager (npm) для установки і управління всіма фронтенд-залежностями. Вебпак також може виконувати більшість завдань grunt / gulp'a.

Sass - це препроцесор CSS, за допомогою якого можна знизити кількість повторюваного CSS коду і заощадити час. Це більш стабільне розширення CSS, чітко і структурно описує стилі документа.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Реалізація клієнтської частини

SPA - концепція односторінкового веб-ресурсу, що припускає використання архітектурного рішення "Товстий клієнт" (надає собою функціональний інтерфейс користувача з виконанням бізнес-логіки) і API серверної частини для обміну даними. Особливість "Толстого клієнта" в даному випадку полягає в тому, що практично всі проміжні дії для отримання документа відбуваються на клієнті і тільки дані принципових дій відправляються на сервер, де в якості відповіді приходять необхідні дані для досягнення користувальницької мети. Таким чином, це зменшує залежність від якості зв'язку між клієнтом і сервером.

Без звернення клієнта на сервер, зміна уявлень користувацького інтерфейсу змінюється практично миттєво згідно UX. У разі звернення, обмін даними відбувається в асинхронному режимі, що дозволяє продовжувати роботу користувача без блокування клієнта в очікуванні відповіді сервера.

За рахунок винесення частини бізнес-логіки й обробки подань на клієнта, а також відділення уявлення від даних, - досягається слабка зв'язаність клієнтської і серверної частини веб-ресурсу. Таким чином, в порівнянні з традиційним підходом організації ВП, робота клієнтської частини ВП характеризується високою швидкодією (проте має залежність від апаратних характеристик клієнта) і можливістю коректної роботи при неякісному з'єднанні.

Клієнтська частина сайту відобразить всі плюси технологій React, Redux та концепції Single Page Application. Даній ресурс має гарну продуктивність, високу захищеність і динамічний інтерфейс клієнта.

На даному сайті можна:

- Зареєструватися;
- Увійти;

- Переглянути головну сторінку
- Переглядати новини
- Створювати новини
- Редагувати новини
- Видаляти новини
- Створювати класи
- Редагувати класи
- Видаляти класи
- Давати ролі користувачам
- Редагувати дані про користувача
- Забанити користувача
- Редагувати дані профілю;

Перейшовши на сайт, буде відображено голов сторінку яка відображена на рисунку 3.1.

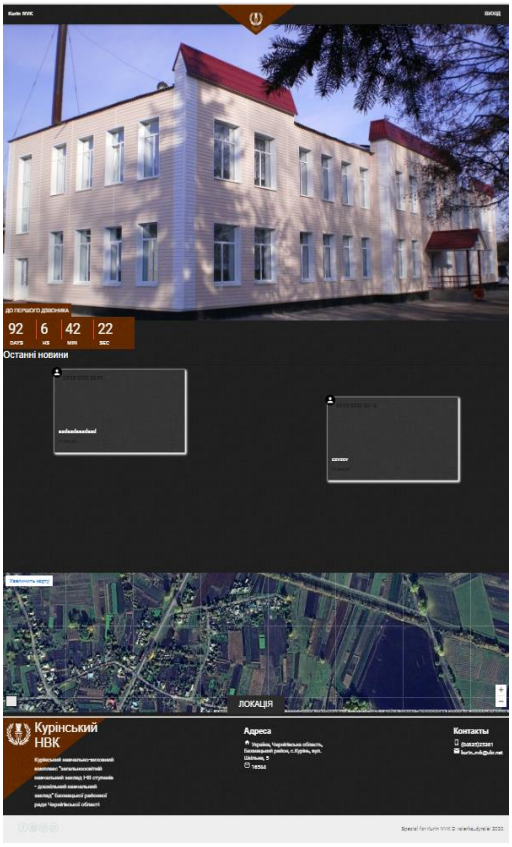

Рисунок 3.1 – Головна сторінка сайта.

Для логічної постановки всіх сторінок веб ресурсу було зроблено логічну структуру сайту яка зображена на рисунку 3.2.

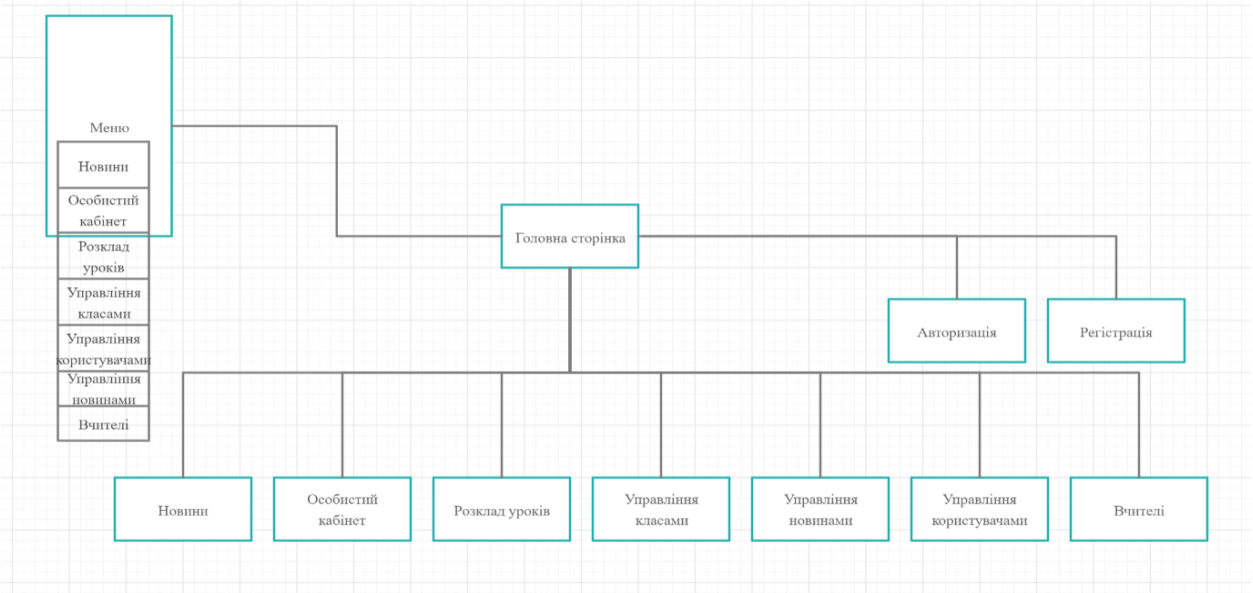

Рисунок 3.2 — Логічна структура сайту

Однією з діаграм, що застосовуються на етапі проектування логічної моделі ІС, є діаграма варіантів використання яка зображена на рисунку 3.3.

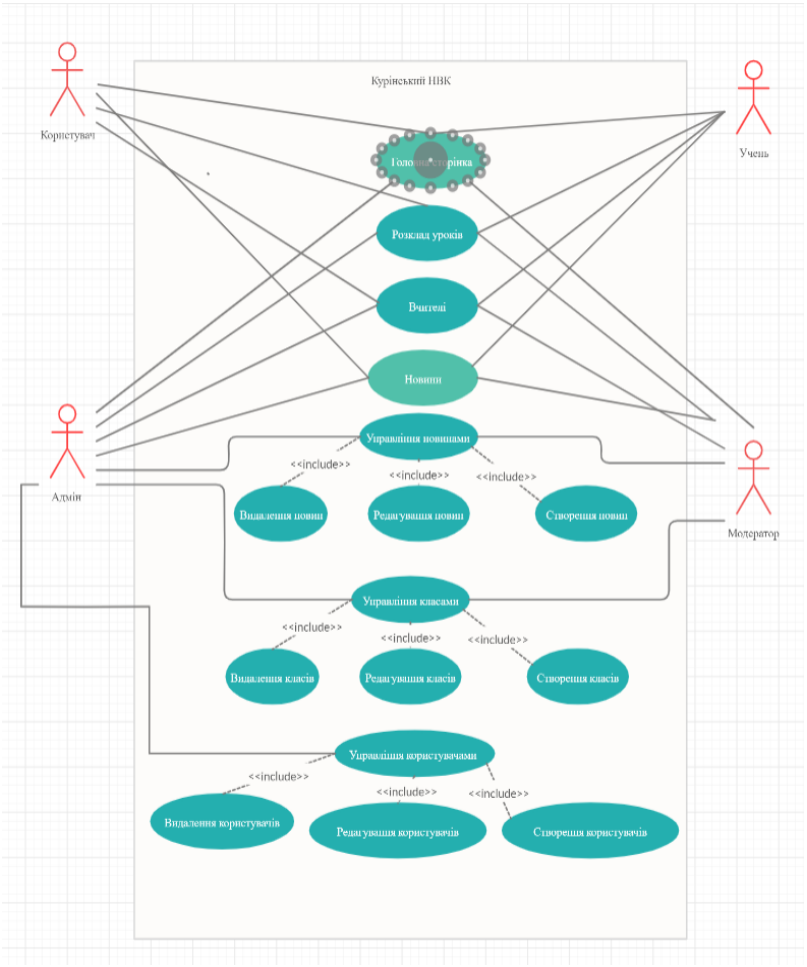

Рисунок 3.3 — Модель варіантів використання

Так як я використовую Redux, то на відміну від useState() єдиний спосіб відновити стан компоненту це сказати про це reducer:

Розглянемо код файлу auth.reducer.js

```
const authReducer = (state = initialState, action) => {
    switch (action.type) {
    case "LOGIN_REQUEST":
        return loginRequest(state);
    case "LOGIN_SUCCESS":
        return loginSuccess(state, action);
    case "LOGIN_ERROR":
        return loginError(state, action);
    case "LOGOUT":
        return logout(state);
    case "REGISTRATION_REQUEST":
        return registrationRequest(state);
    case "REGISTRATION_SUCCESS":
        return registrationSuccess(state);
    case "REGISTRATION_ERROR":
        return registrationError(state, action);
```

```
        case "CLEAR_AUTH_ERROR":
            return clearAuthError(state);
        default:
            return state;
        }
};
```

3.2 Реалізація серверної частини

Архітектура серверної частини побудована на основі фреймворку Express. він інкапсулює обробку високорівневих об'єктів додатки, наприклад, такі як об'єкти запиту і відповіді, надаючи їх в розпорядження розробнику. Для структурування серверної частини веб-ресурсу були взяті принципи архітектурного типового рішення MVC. Розглянемо основні компонентні уявлення серверної частини ВП, які були розроблені.

Маршрутизатор (Route) - відповідно до маршруту делегує подальшу обробку запиту керуючим об'єктів нижчого рівня: контролеру або API.

Контролер (Controller) - відповідає за обробку запиту, визначає HTML для користувацького інтерфнйсу. Містить методи для реалізації функціональних вимог до веб-ресурсу. Використовує модель для отримання даних предметної області.

Модель (Model) - відповідає за бізнес-логіку. Є моделлю суті. Надає інтерфейс для роботи з сутностями. Інкапсулює обробку даних відповідної їй суті.

Mongoose - являє "Обгортку" для моделей, надає функціональність для роботи з моделями. Шаблон (Template) - уявлення користувацького інтерфейсу у вигляді HTML розмітки.

Вид (View) - уявлення, код який необхідно обробити сервером додатка.

API - стандартизований інтерфейс для роботи з даними предметної області.

Згідно з принципом замикання класи були об'єднані в пакети, враховуючи прийняте рішення MVC, діаграма пакетів серверної частини зображено на рисунку 3.4.

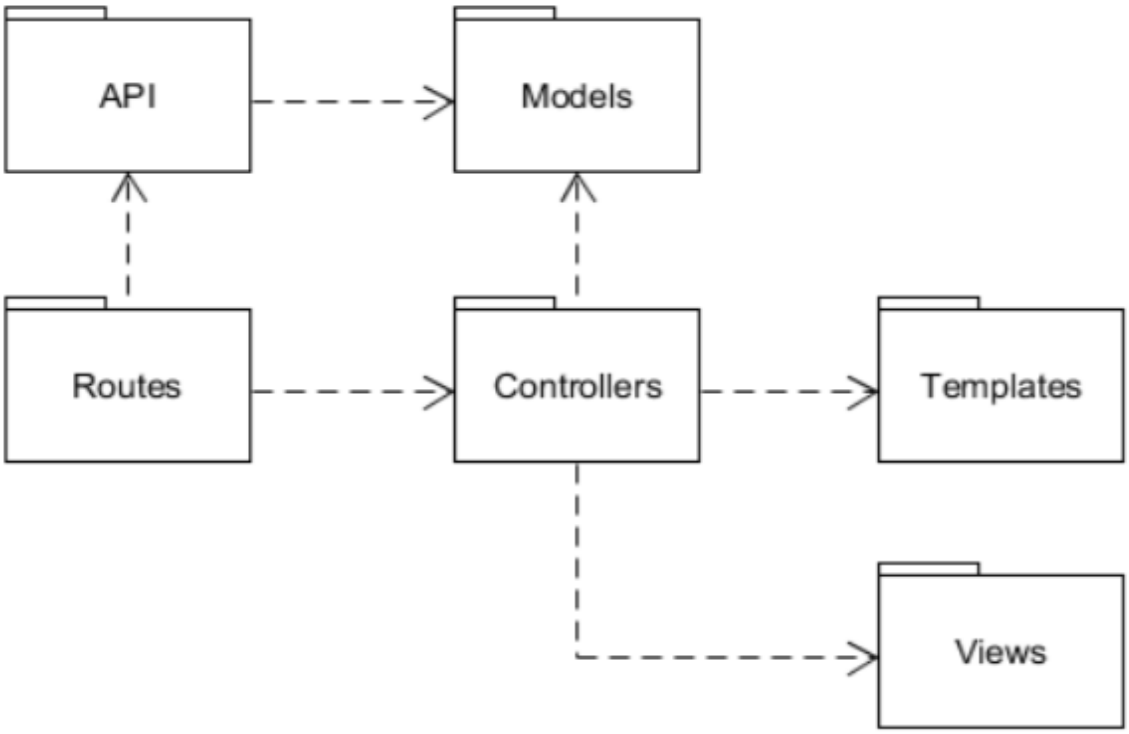

Рисунок 3.4 — Діаграма пакетів серверної частини веб-додатку.

Розглянемо код файлу user.js

```
const express = require('express');
const passport = require('passport');
const controller = require('../controllers/user');
const hasAccess = require('../middleware/access');
const { ROLES } = require('../roles-and-access');
const upload = require('../middleware/upload');

const router = express.Router();

router.get(
    '/',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.ADMIN),
    controller.getAll,
);
router.get(
    '/:id',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.USER),
    controller.getById,
);
router.delete(
    '/:id',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.ADMIN),
    controller.delete,
```

```
);
router.patch('/:id',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.USER),
    upload.single('avatar'),
    controller.update);

module.exports = router;
```

Зайшовши на сайт, користувач може або відправити в POST-запиті дані для авторизації, або перейти на сторінку реєстрації і відправити звідти POST-запит з даними для реєстрації. пройшовши етап аутентифікації, користувач може отримати список своїх каналів за допомогою GET-запиту або створити новий канал за допомогою POST-запиту. Відкривши сторінку якогось конкретного каналу розсилки, користувач може підписатися на нього за допомогою PUT-запиту. Методи GET і POST дозволяють подивитися історію повідомлень і відправити повідомлення передплатниками каналу відповідно. Усе дії, крім реєстрації, входу в систему та підписки на канал, вимагають аутентифікації користувача.

3.3 Інструкція користувача

Веб-сайт розроблено на мові програмування JavaScript і може експлуатуватися під управління будь- яких операційних систем та браузерів.

Для того щоб перейти на сайт потрібно в адресний рядок а ввести "https://kyrin-nvk.herokuapp.com/" відкриється головна сторінка веб-сайту як показано на рисунку 3.5

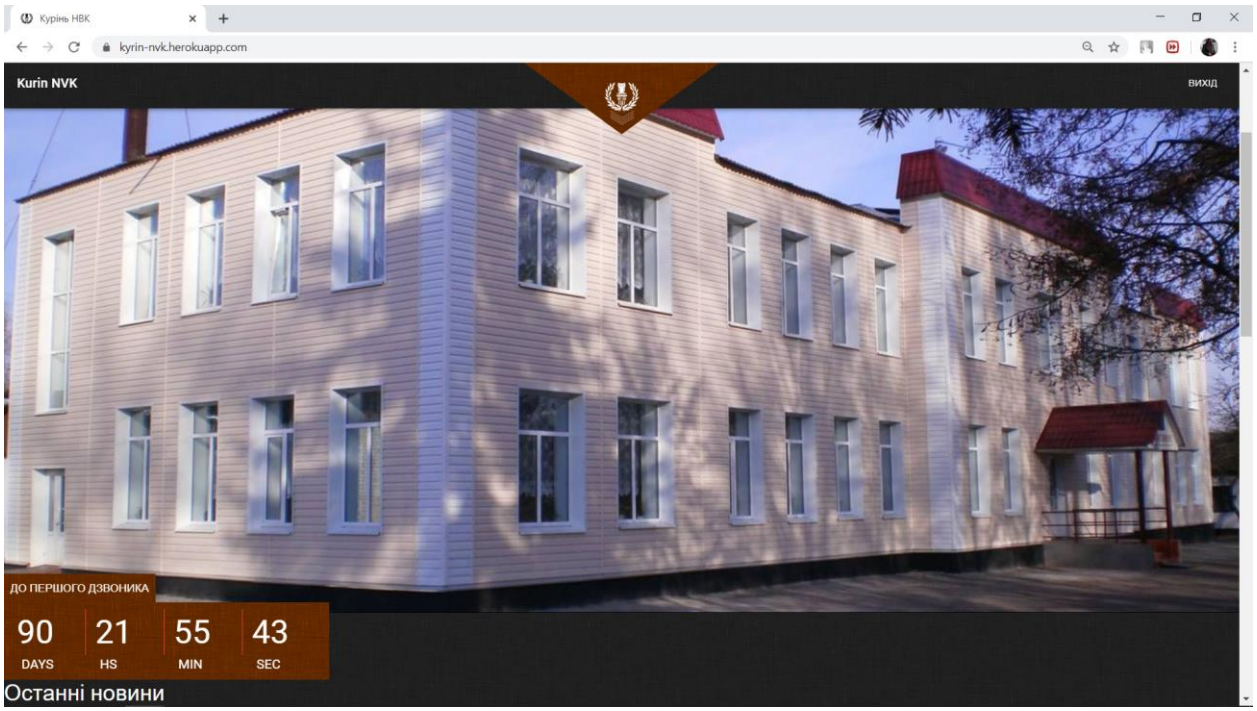

Рисунок 3.5 — Головна сторінка сайту

Для того щоб перейти на сторінку авторизації потрібно в верхньому правому кутку натиснути кнопку "ВХІД", потім відкриється сторінка з формою авторизації як показано на рисунку 3.6

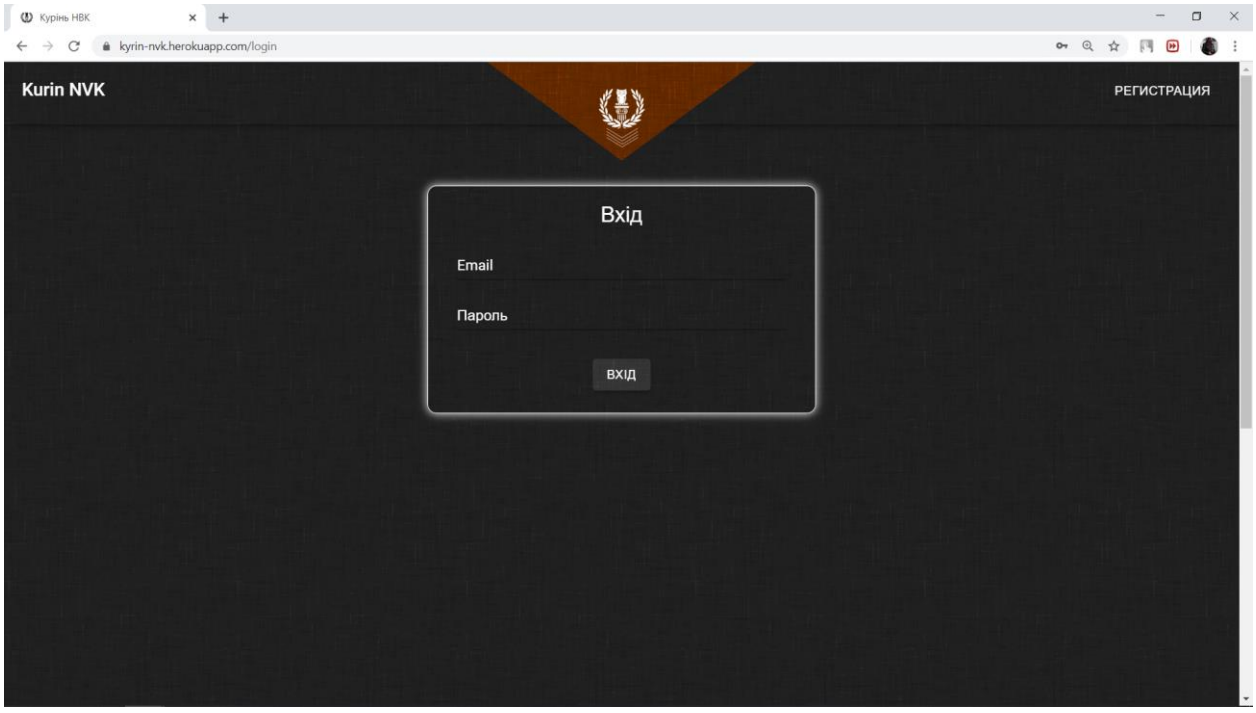

Рисунок 3.6 — Сторінка авторизації

Якщо в користувача немає облікового запису він може зареєструватися натиснувши кнопку "РЕЄСТРАЦІЯ" яка знаходиться на сторінці входу і йому відкриється форма реєстрації як показано на рисунку 3.7

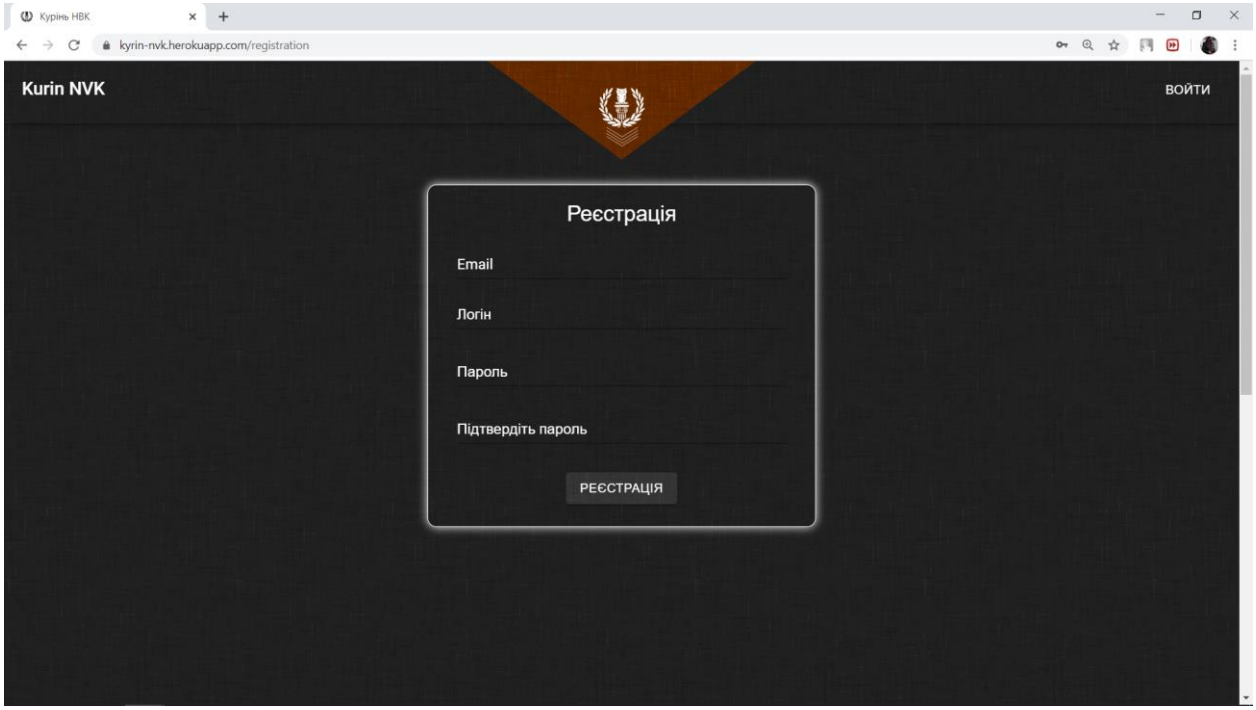

Рисунок 3.7 — Сторінка реєстрації

Після підтвердження входу в користувача в меню з’являться нові елементи списку. Якщо користувач не підтвердив вхід в меню будуть елементи "Новини", "Розклад уроків", "Вчителі". Якщо користувач підтвердив вхід до списку меню додається елемент "Особистий кабінет". Якщо у користувача назначена роль "Модератор" до списку меню додаються елементи "Управління класами" та "Управління новинами". Якщо у користувача роль "Адміністратор" то він має в меню всі елементи списку як показано на рисунку 3.8

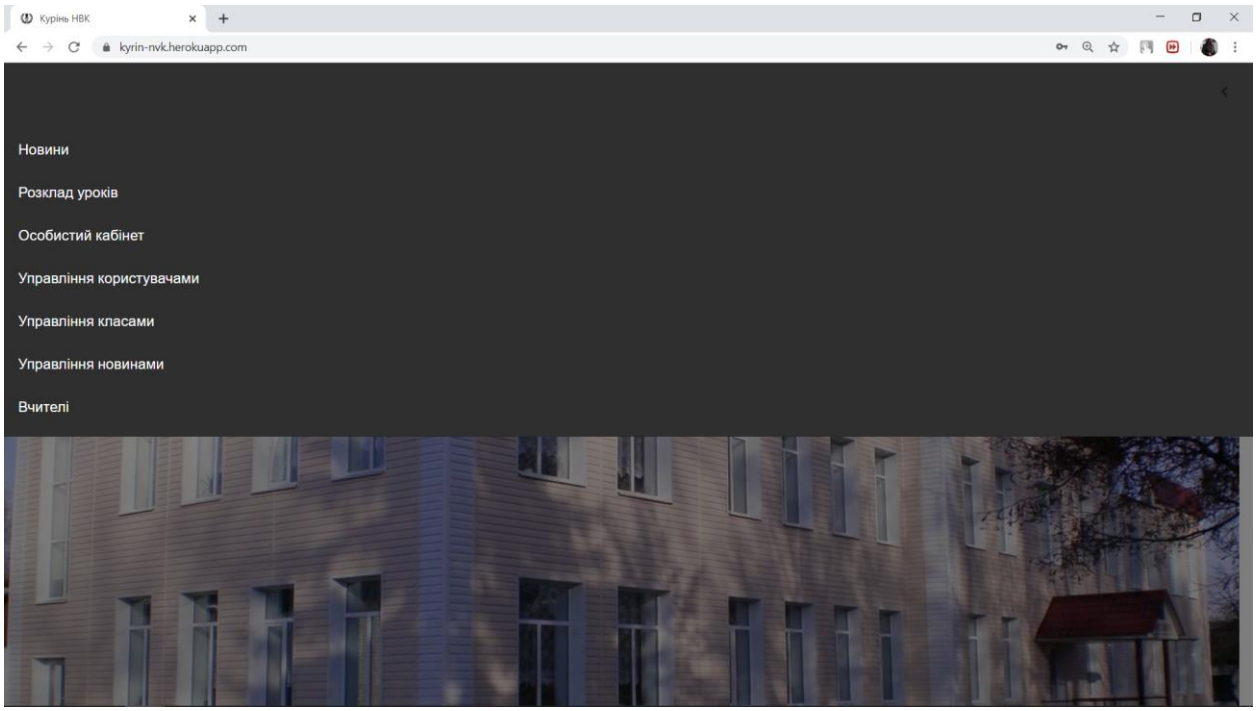

Рисунок 3.8 — Список меню ролі "Адміністратор"

Після того як натиснути на елемент списку меню "Новини" в користувача відкриється сторінка всіх новин як показано на рисунку 3.9, якщо натиснути на будь-яку новину відкриється сторінка про детальну інформацію вибраної новини як показано на рисунку 3.10

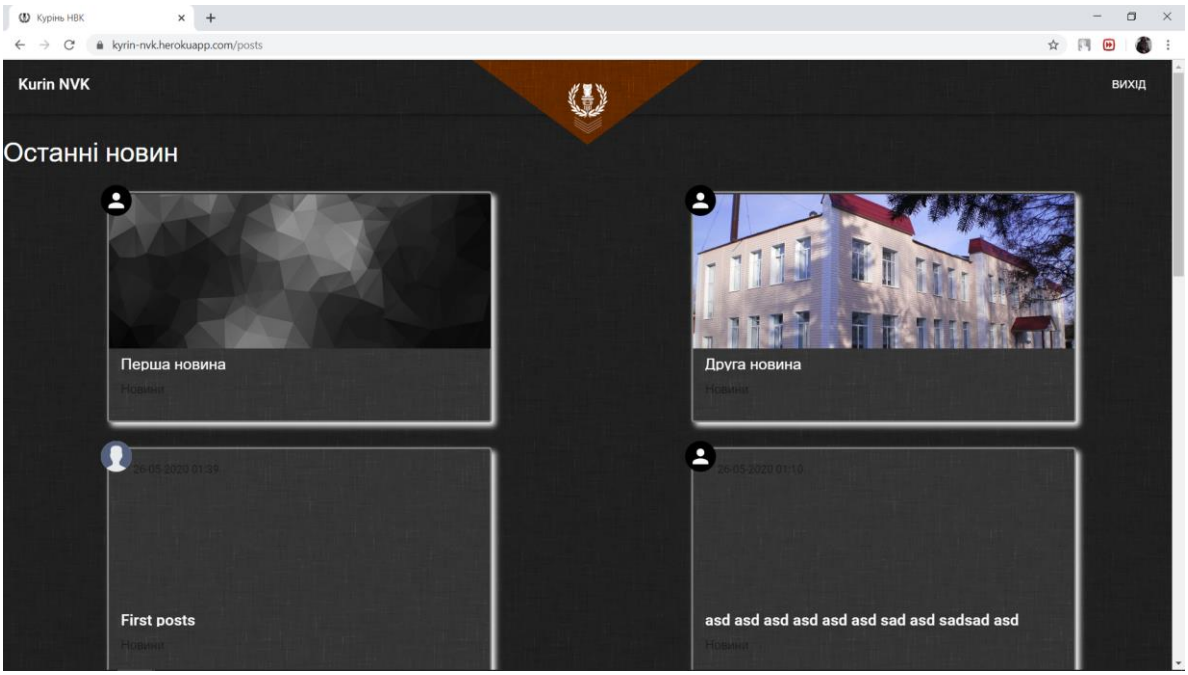

Рисунок 3.9 — Сторінка новин

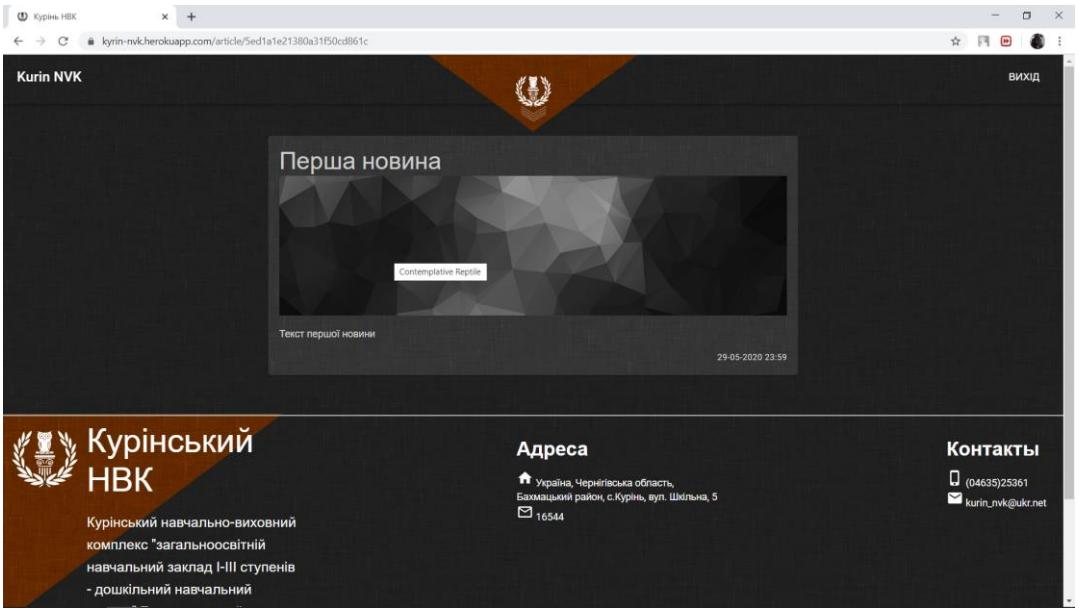

Рисунок 3.10 — Сторінка вибраної новини

Після того як натиснути на елемент списку меню "Розклад уроків" в користувача відкриється сторінка розкладу уроків як показано на рисунку 3.11

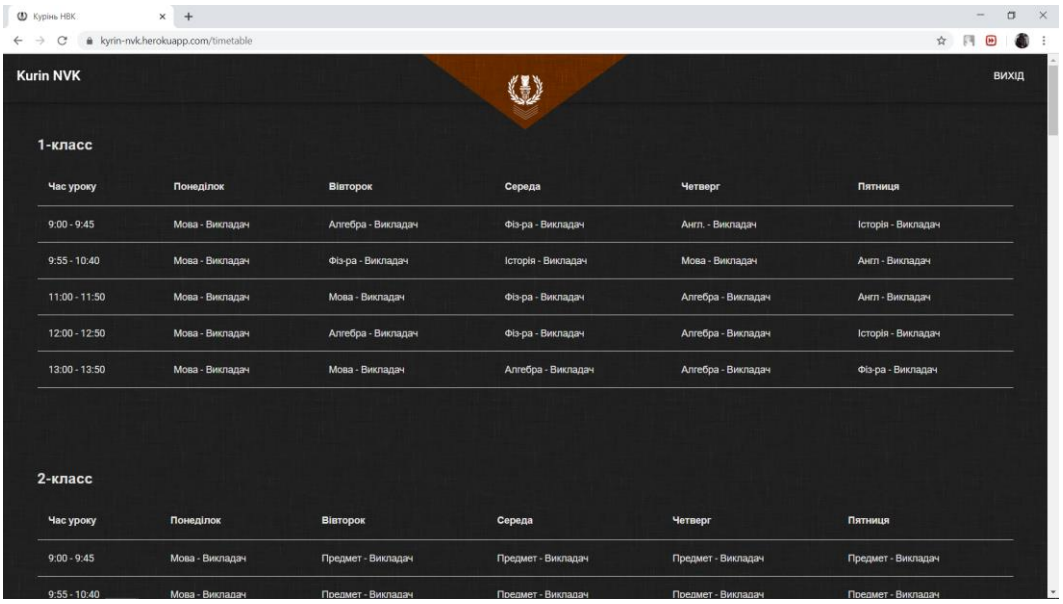

Рисунок 3.11 — Сторінка розкладу уроків

Після того як натиснути на елемент списку меню "Вчителі" в користувача відкриється сторінка з вчителями як показано на рисунку 3.12.

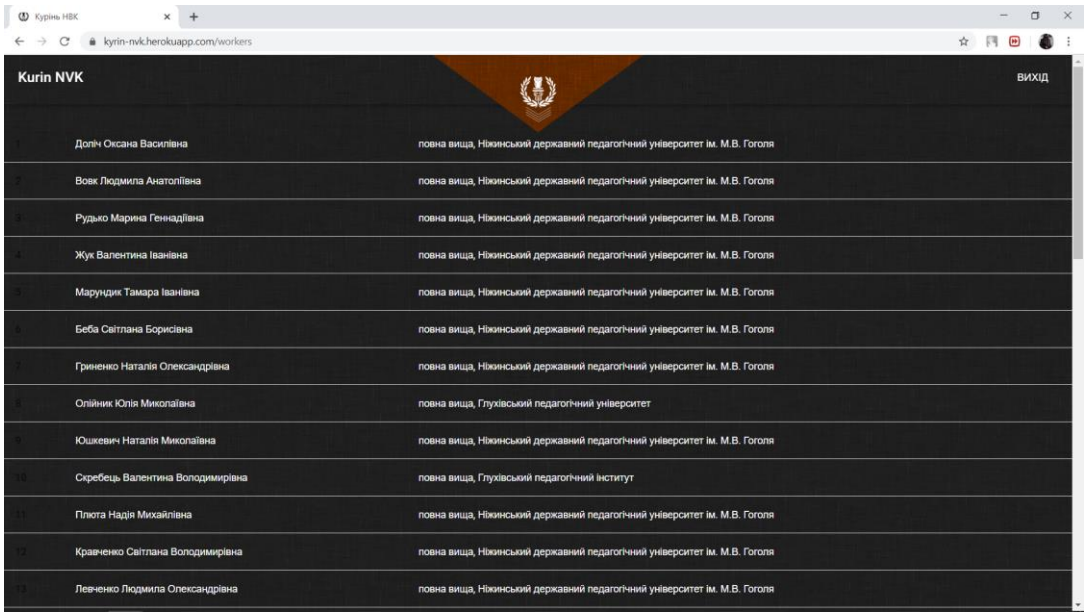

Рисунок 3.12 — Сторінка вчителів

Після того як натиснути на елемент списку меню "Особистий кабінет" в користувача відкриється сторінка особистого кабінету як показано на рисунку 3.13

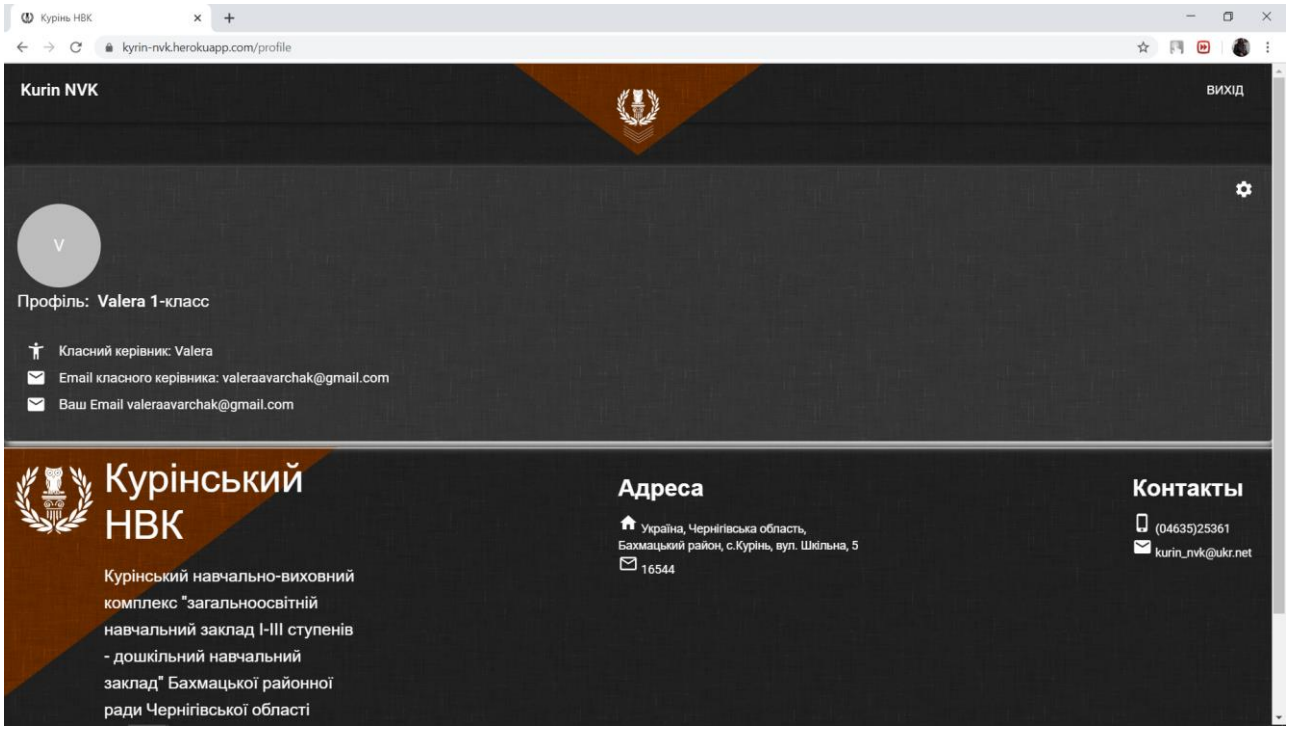

Рисунок 3.13 — Особистий кабінет

Зареєстрований користувач може змінювати інформацію про себе якщо натиснути на кнопку в верхньому правому куті як показано на рисунку 3.14

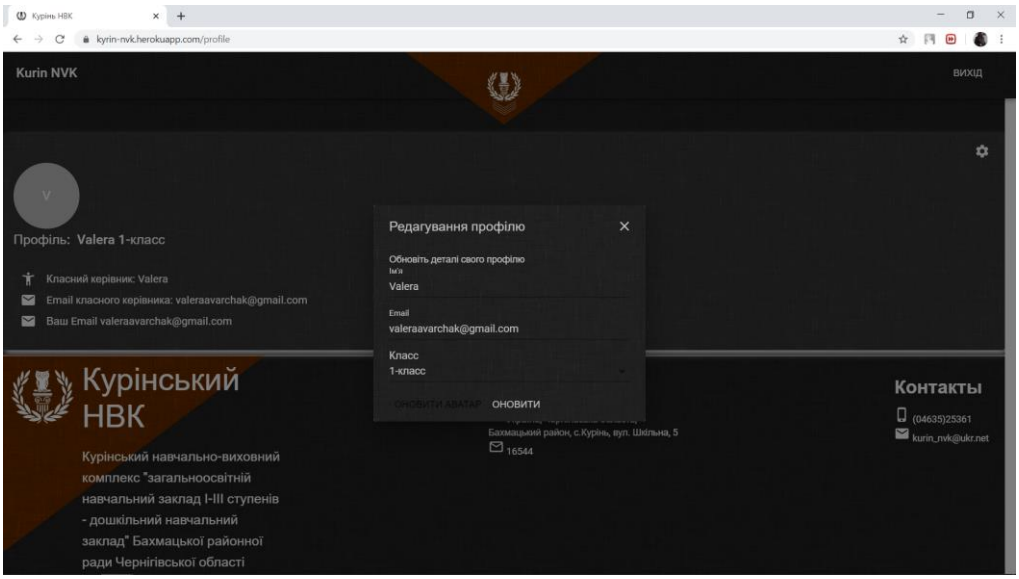

Рисунок 3.14 — Вікно для редагування профілю

Якщо в користувача роль "Модератор" він має право перейти на сторінку "Управління класами" як показано на рисунку 3.15

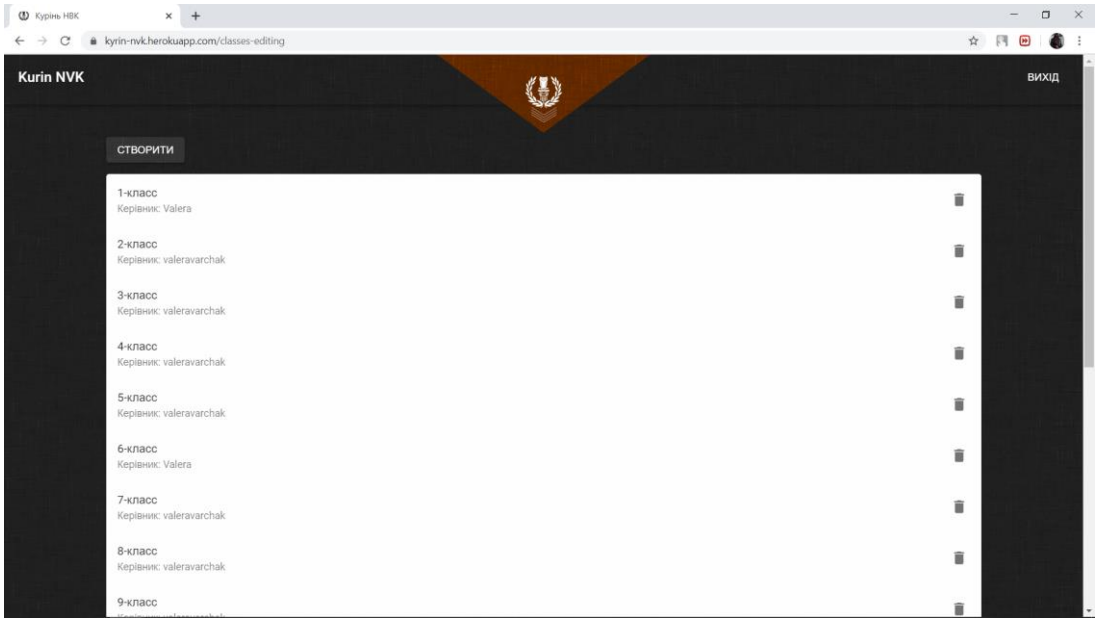

Рисунок 3.15 — Сторінка управління класами

Модератор зможе створити клас, коли натисне на кнопку "Створити" в нього відкриється вікно як показано на рисунку 3.16, редагувати клас можна при натисненні на будь-який елемент списку класів відкриється вікно як показано на рисунку 3.17, при натисненні на іконку видалення класу відкриється вікно для підтвердження як показано на рисунку 3.18

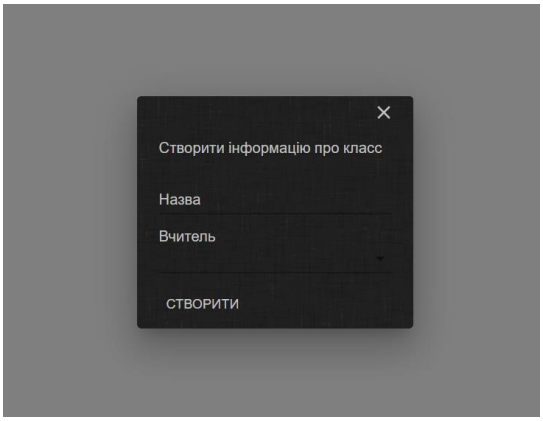

Рисунок 3.16 — Вікно створення класу

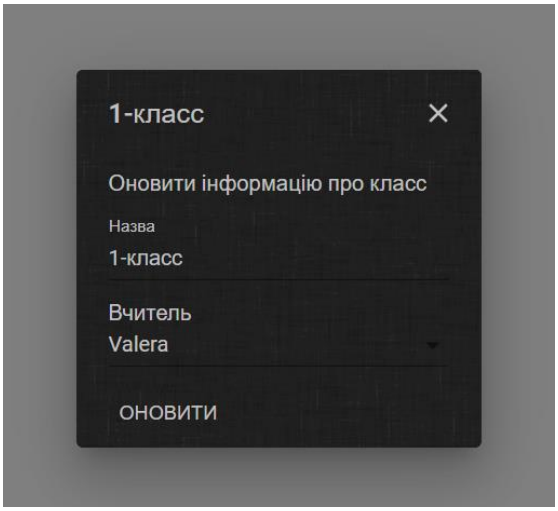

Рисунок 3.17 — Вікно редагування класу

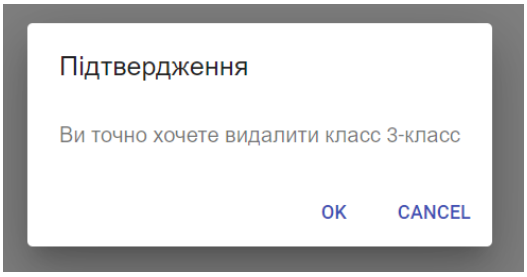

Рисунок 3.18 — Вікно підтвердження видалення класу

Також модератор має право перейти на сторінку "Управління новинами" як показано на рисунку 3.19.

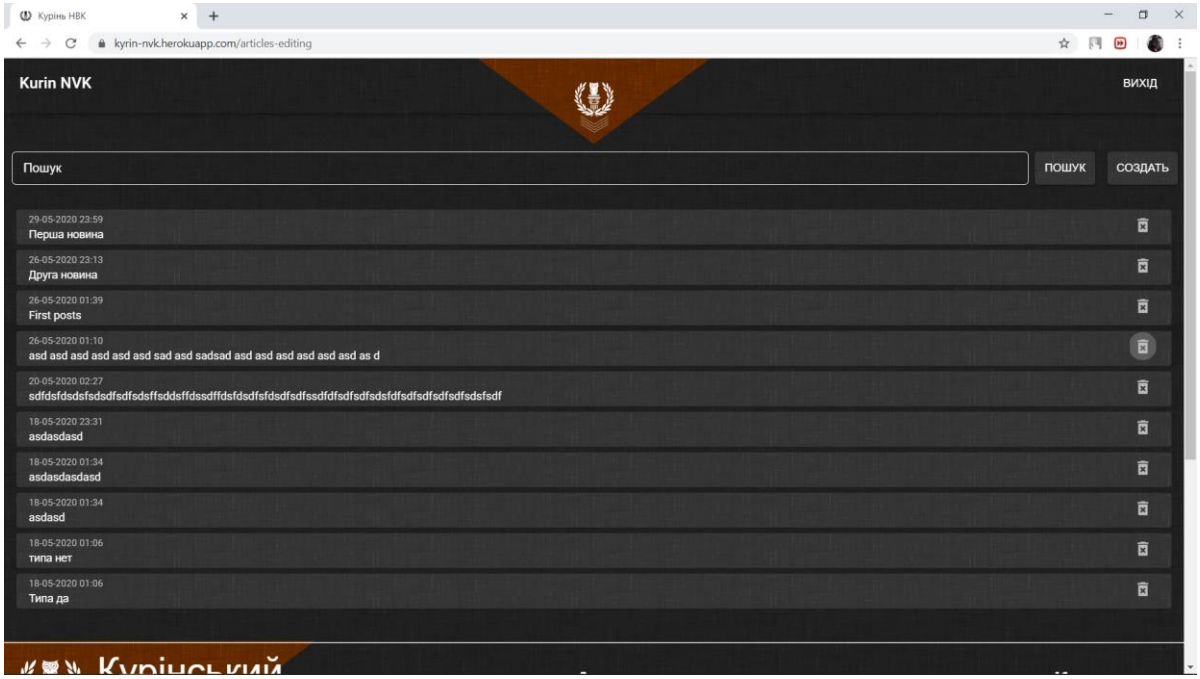

Рисунок 3.19 — Сторінка управління новинами

Модератор зможе створити новину, коли натисне на кнопку "Створити" в нього відкриється сторінка з формою як показано на рисунку 3.20, редагувати можна при натисненні на будь-який елемент списку новин після цього у користувача відкриється сторінка з формою як показано на рисунку 3.21, при натисненні на іконку видалення новини відкриється вікно для підтвердження як показано на рисунку 3.22.

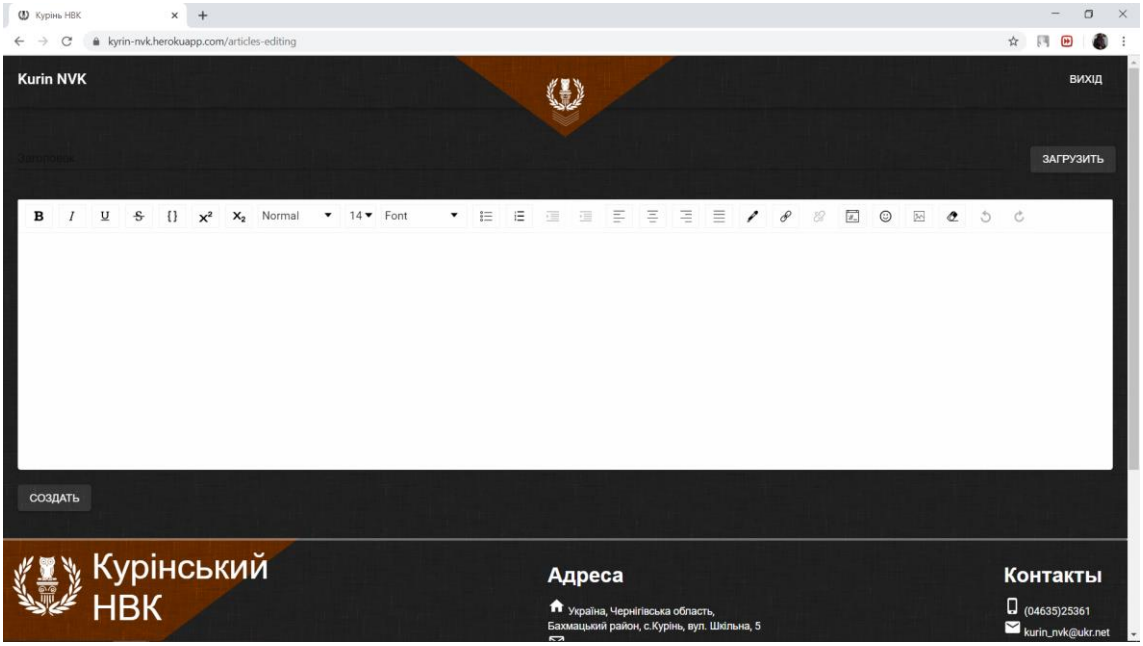

Рисунок 3.20 — Сторінка створення новини

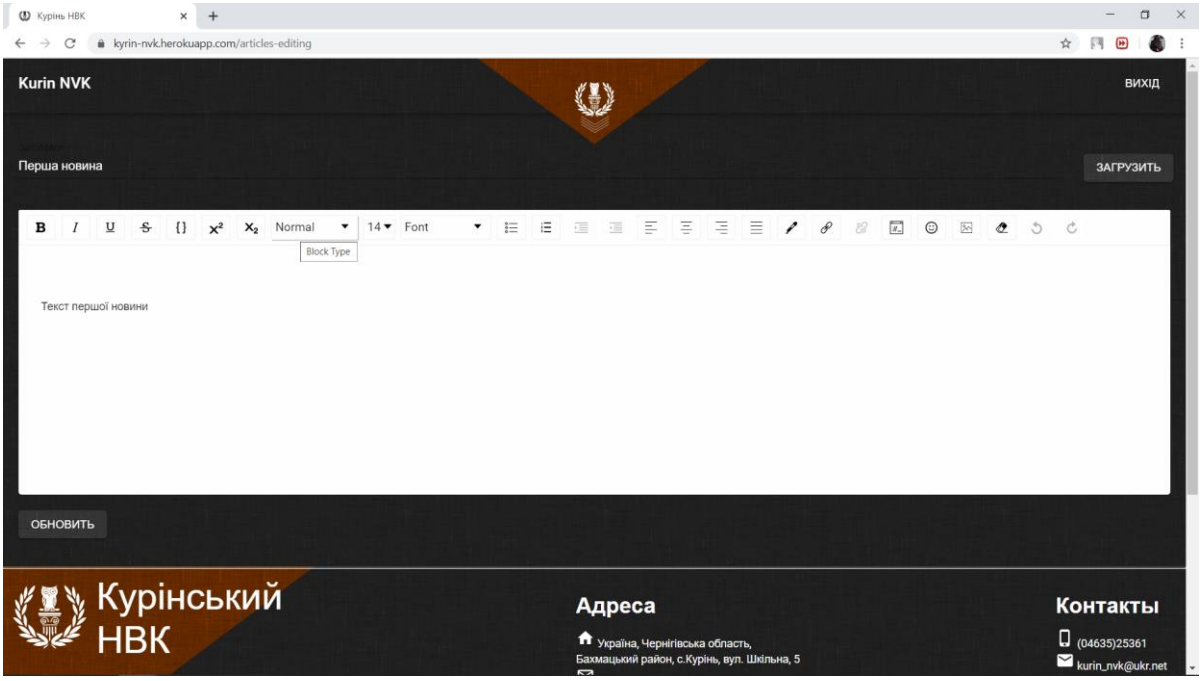

Рисунок 3.21 — Сторінка редагування новини

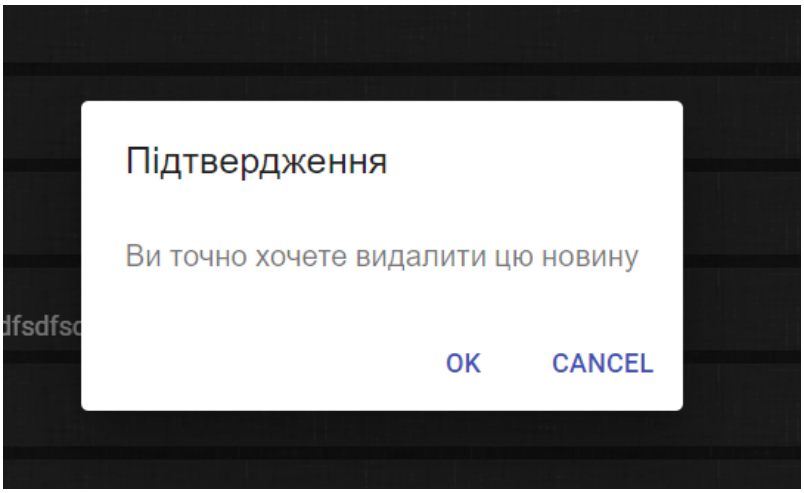

Рисунок 3.22 — Вікно для підтвердження видалення.

Якщо у користувача роль "Адміністратор" він має право перейти на сторінку "Управління користувачами" як показано на рисунку 3.23.

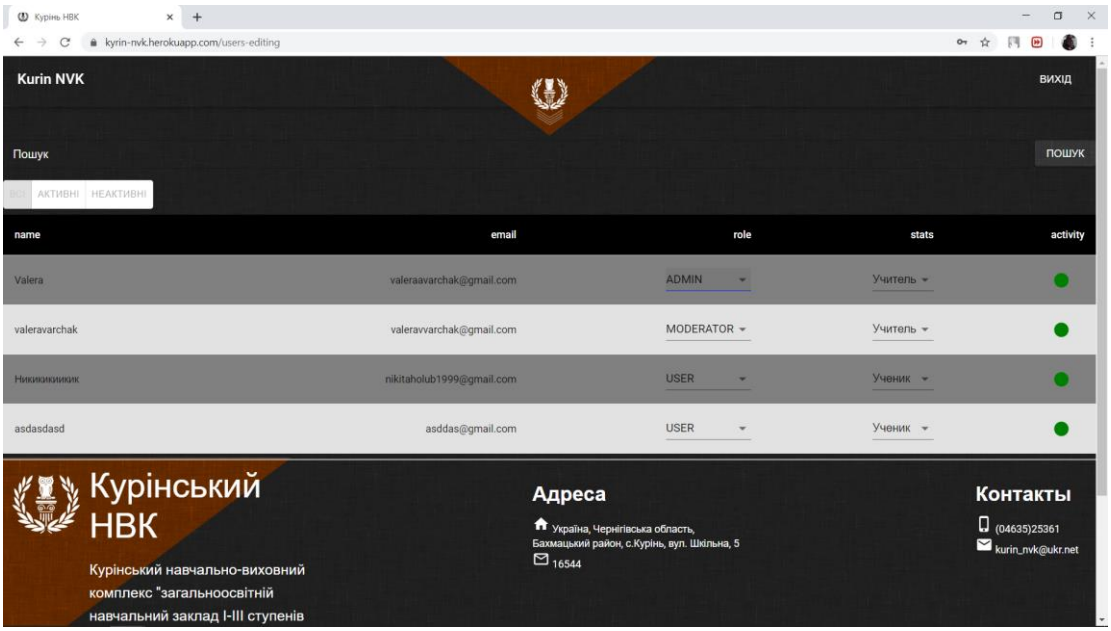

Рисунок 3.23 — Сторінка управління користувачами

Адміністратор зможе найти користувача завдяки полю пошуку, відфільтрувати користувачів, назначити роль любому користувачу як показано на рисунку 3.24, та відключити користувача при натисненні на зелену кнопку, або відновити при на тисненні на зелену як показано на рисунку 3.25.

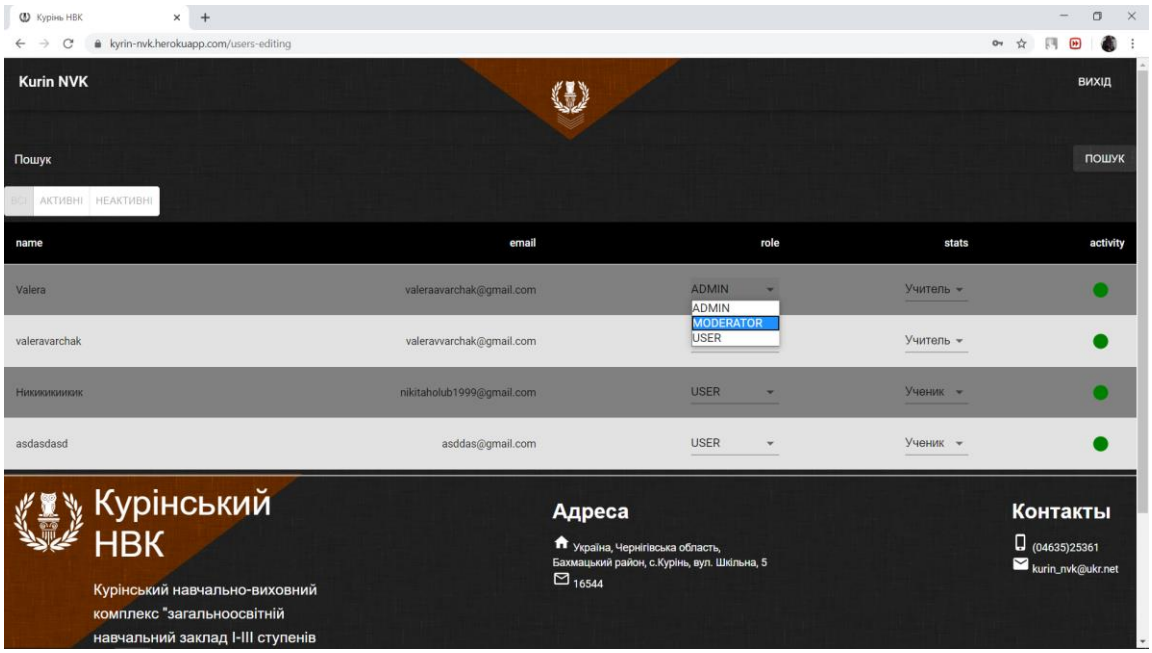

Рисунок 3.24 — Список для назначення ролі

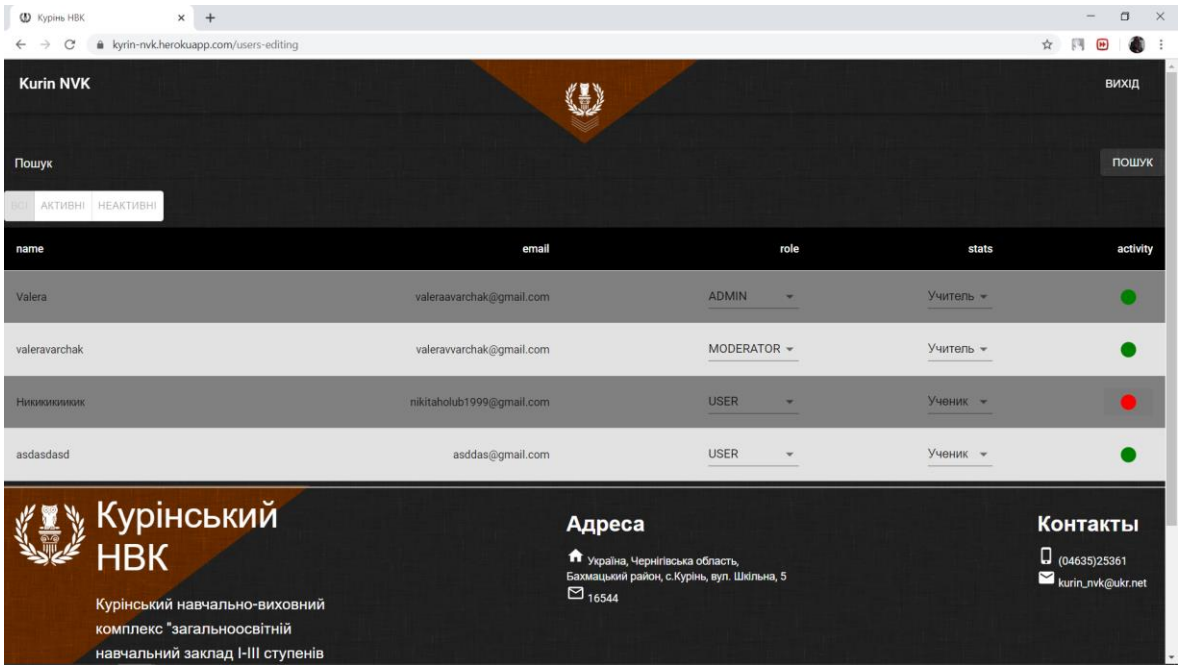

Рисунок 3.25 — Кнопки управління активності користувача

ВИСНОВОК

В ході виконання роботи були розглянуті та проаналізовані сучасні технології створення веб-ресурсів, проведено аналіз тематичної літератури, вибір програмних засобів для реалізації проекту.

Результатом виконання даної роботи є створення веб сайту для школи мережа із використанням стеку технологій MERN, а саме MongoDB, Express, Reacr, Node.js.

В результаті роботи було досягнуто поставлені завдання, а саме: проведено дослідження сучасних WEB-технології та бібліотек, розроблений веб-сайт «Курінський НВК».

Програмний продукт включає в себе:

1. Базу даних, що містить всю необхідну інформацію.
2. Ролі для управління вею-сайтом та окремих компонентів сайту
3. WEB-сервер, що надає WEB-клієнту API для управління даними.
4. Адаптивний дизайн який допомагає користувачеві використовувати сайт на різних платформах.

Шкільний сайт, як і будь-який інший, - це складна інформаційна система. При його створенні необхідно врахувати і проаналізувати цілий комплекс чинників: хто буде споживачем інформації вашого сайту, що зі шкільного життя дійсно доцільно розміщувати на сайті, яка буде його структура. Розроблений ресурс має зручний функціонал та задовольняє всім вимогам, які ставилися на етапі постановки завдання.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript / И. Браун // Санкт-Петербург: Питер, 2017. - 214 с.
2. Гринберг С. UX-дизайн. Идея - эскиз - воплощение/ Санкт-Петербург:Питер, 2014. - 250 с.
3. Хопкинс К. PHP Быстрый старт. / К. Хопкинс // Москва: Эксмо, 2014. – 110с.
4. A Javascript library for building user interfaces – React. [Електронний ресурс]. – Режим доступа: https://facebook.github.io/react/.
5. 5 практических примеров для изучения фреймворка React. [Електронний ресурс]. – Режим доступа: https://habrahabr.ru/post/229629/
6. Node.js [Электронный ресурс]. – Режим доступа: https://nodejs.org/en/.
7. MongoDB for GIANT Ideas | MongoDB [Електронний ресурс]. – Режим доступа: https://www.mongodb.com/.
8. Introduction to MongoDB – MongoDB Manual 3.4. [Електронний ресурс]. – Режим доступа: https://docs.mongodb.com/manual/introduction/

ДОДАТОК

Файл school-service.js асинхронні функції для отримання данних з API.

```
const Exception = (message, status) => ({
   message,    status,
});
class SchoolService {
    _baseUrl = "https://kyrin-nvk.herokuapp.com/api";
    // _baseUrl = "http://localhost:5000/api";
    getResource = async (url, method, body, isFormData) => {
        let response;
        const headers = new Headers();
        headers.append("Accept", "application/json");
        if (isFormData) {
            console.log(method);
            console.log(isFormData);
            if (localStorage.getItem("school-user-with-jwt")) {
                const token = JSON.parse(localStorage["school-user-
with-jwt"]).token.toString();
                headers.append("Authorization", token);
            }
            response = await fetch(`${this._baseUrl}${url}`, {
                method,
                headers,
                body,
            });
        } else {
            headers.append("Content-Type", "application/json");
            if (localStorage.getItem("school-user-with-jwt")) {
                const token = JSON.parse(localStorage["school-user-
with-jwt"]).token.toString();
                headers.append("Authorization", token);
            }

            response = await fetch(`${this._baseUrl}${url}`, {
                method,
                headers,
```

```
                body: JSON.stringify(body),
            });
        }

        if (!response.ok) {
            if (response.status === 401) throw
Exception("Unauthorized", response.status);
            const resJSON = await response.json();
            throw Exception(resJSON.message, response.status);
        }

        return await response.json();
    };
    login = async (userInfo) => (
        await this.getResource("/auth/login", "POST", userInfo)
    );
    registration = async (userInfo) => (
        await this.getResource("/auth/registration", "POST", userInfo)
    );
    getAllUsers = async (showMode = "all", filterField, term, teachers
= false) => (
        await
this.getResource(`/user?showMode=${showMode}&filterField=${filterField
|| ""}&term=${term || ""}&teachers=${teachers}`, "GET")
    );
    updateUser = async (userId, body, isForm = true) => (
        await this.getResource(`/user/${userId}`, "PATCH", body,
isForm)
    );
    getAllArticles = async (skip = 0, limit = 0, term = "") => (
        await
this.getResource(`/article?limit=${limit}&skip=${skip}&term=${term}`,
"GET")
    );
    getArticleById = async (articleId) => (
        await this.getResource(`/article/${articleId}`, "GET")
    );
    createArticle = async (body) => (
```

```
        await this.getResource(`/article`, "POST", body, true)
    );
    updateArticle = async (articleId, body) => (
        await this.getResource(`/article/${articleId}`, "PATCH", body,
true)
    );
    deleteArticle = async (articleId) => (
        await this.getResource(`/article/${articleId}`, "DELETE")
    );
    getAllClasses = async () => (
        await this.getResource(`/grade`, "GET")
    );
    getClassById = async (classId) => (
        await this.getResource(`/grade/${classId}`, "GET")
    );
    createClass = async (body) => (
        await this.getResource(`/grade`, "POST", body)
    );
    updateClass = async (classId, body) => (
        await this.getResource(`/grade/${classId}`, "PATCH", body)
    );
    deleteClass = async (classId) => (
        await this.getResource(`/grade/${classId}`, "DELETE")
    )
}
export default SchoolService;
```

Данні про користувачів на стороні клієнта файл users.action.js

```
import { authActions } from "./index";
const usersRequest = () => ({
    type: "USERS_REQUEST",
});
const usersSuccess = (users) => ({
    type: "USERS_SUCCESS",
```

```
    payload: {
        users,
    },
});
const usersError = (message) => ({
    type: "USERS_ERROR",
    payload: {
        message,
    },
});
const getAllUsers = (
    schoolService,
    active = "all",
) => () => async (dispatch) => {
    dispatch(usersRequest());
    try {
        const responseData = await schoolService.getAllUsers(active);
        dispatch(usersSuccess(responseData));
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(usersError(e.message));
    }
};
const teachersSuccess = (teachers) => ({
    type: "TEACHERS_SUCCESS",
    payload: {
        teachers,
    },
});
const getAllTeachers = (
    schoolService,
    active = "all",
) => () => async (dispatch) => {
    dispatch(usersRequest());
    try {
```

```
        const responseData = await schoolService.getAllUsers(active,
"", "", true);
        dispatch(teachersSuccess(responseData));
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(usersError(e.message));
    }
};
const updateUser = (
    schoolService,
    userId,
    data,
    isForm = true,
) => () => async (dispatch) => {
    try {
        await schoolService.updateUser(userId, data, isForm);
        dispatch(usersRequest());
        const responseData = await schoolService.getAllUsers("all");
        dispatch(usersSuccess(responseData));
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
    }
};

const findUsers = (
    schoolService,
    active = "all",
    filter,
    term,
) => () => async (dispatch) => {
    dispatch(usersRequest());
    try {
        const responseData = await schoolService.getAllUsers(active,
filter, term);
        dispatch(usersSuccess(responseData));
    } catch (e) {
```

```
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(usersError(e.message));
    }
};
const setShowMode = (showMode) => ({
    type: "SET_SHOW_MODE",
    payload: {
        showMode,
    },
});
const setUsersTerm = (term) => ({
    type: "SET_USERS_TERM",
    payload: {
        term,
    },
});
const setFilterField = (filterField) => ({
    type: "SET_FILTER_FIELD",
    payload: {
        filterField,
    },
});

const usersActions = {
    getAllUsers,
    updateUser,
    findUsers,
    setShowMode,
    setUsersTerm,
    setFilterField,
    getAllTeachers,
};
export default usersActions;
```

Файл users.reducer.js

```
const initialState = {
    users: [],
    teachers: [],
    usersError: false,
    errorMessage: "",
    filterField: "name",
    showMode: "all",
    term: null,
    loading: false,
};

const usersReducer = (state = initialState, action) => {
    switch (action.type) {
    case "USERS_REQUEST":
        return {
            ...state,
            users: [],
            usersError: false,
            errorMessage: null,
            loading: true,
        };
    case "USERS_SUCCESS":
        return {
            ...state,
            users: action.payload.users,
            usersError: false,
            errorMessage: null,
            loading: false,
        };
    case "USERS_ERROR":
        return {
            ...state,
            users: [],
            usersError: true,
            errorMessage: action.payload.message,
            loading: false,
```

```
        };
    case "TEACHERS_SUCCESS":
        return {
            ...state,
            teachers: action.payload.teachers,
        };
    case "SET_SHOW_MODE":
        return {
            ...state,
            showMode: action.payload.showMode,
        };
    case "SET_TERM":
        return {
            ...state,
            term: action.payload.term,
        };
    case "SET_FILTER_FIELD":
        return {
            ...state,
            filterField: action.payload.filterField,
        };
    case "SET_USERS_TERM":
        return {
            ...state,
            term: action.payload.term,
        };
    default:
        return state;
    }
};

export default usersReducer;
```

Файл users-lists.jsx

```
const StyledTableCell = withStyles((theme) => ({
```

```
    head: {
      backgroundColor: theme.palette.common.black,
      color: theme.palette.common.white,
    },
    body: {
      fontSize: 14,
    },
  }))(TableCell);




const UsersList = ({
    users, currentUser, updateUser, getAllUsers,
}) => {
    const handleChangeRole = (event, id) => {
        if (currentUser._id === id) return;
        const role = event.target.value;
        updateUser(id, { role }, false);
    };

    const handleChangeActivity = (id, active) => {
        console.log(active);
        if (currentUser._id === id) return;
        updateUser(id, { active }, false);
    };

    const handleChangeStatus = (event, id) => {
        let isStudent;
        if (event.target.value === "true") isStudent = true;
        else isStudent = false;
        updateUser(id, { isStudent }, false);
    };

    useEffect(() => {
```

```
        getAllUsers("all");
    }, [getAllUsers]);

    return (
        <TableContainer>
        <Table aria-label="simple table">
        <TableHead>
            <TableRow>
                <StyledTableCell>name</StyledTableCell>
                <StyledTableCell align="right">email</StyledTableCell>
                <StyledTableCell align="right">role</StyledTableCell>
                <StyledTableCell align="right">stats</StyledTableCell>
                <StyledTableCell
align="right">activity</StyledTableCell>
            </TableRow>
        </TableHead>
        <TableBody>
            {users.map((user) => (
                <UsersTableItem
                    key={user._id}
                    handleChangeRole={handleChangeRole}
                    handleChangeActivity={handleChangeActivity}
                    handleChangeStatus={handleChangeStatus}
                    {...user}
                />
            ))}
        </TableBody>
        </Table>
        </TableContainer>
    );
};

const mapStateToProps = ({
    usersReducer: { users },
    authReducer: { currentUser },
}) => ({
```

```
    users,
    currentUser,
});


const mapDispatchToProps = (dispatch, { schoolService }) =>
bindActionCreators({
    getAllUsers: (active) => usersActions.getAllUsers(schoolService,
active)(),
    updateUser: (
        userId,
        data,
        isForm,
    ) => usersActions.updateUser(schoolService, userId, data,
isForm)(),
}, dispatch);


export default withSchoolService()(connect(mapStateToProps,
mapDispatchToProps)(UsersList));
```

Дані про користувача на стороні сервера файл controllers > user.js

```
const User = require('../models/User');
const errorHandler = require('../utils/error-handler');


module.exports.getAll = async (req, res) => {
    try {
        const { showMode } = req.query;
        const { filterField } = req.query;
        const { term } = req.query;
        const { teachers } = req.query;
        let users;


        if (showMode === 'all') {
            users = await User.find()
                .populate({
                    path: 'grade',
```

```
                populate: {
                    path: 'classroomTeacher',
                    select: 'name _id email',
                },
            });
        }
        if (showMode === 'active') {
            users = await User.find({ active: true })
                .populate({
                    path: 'grade',
                    populate: {
                        path: 'classroomTeacher',
                        select: 'name _id email',
                    },
                });
        }
        if (showMode === 'inactive') {
            users = await User.find({ active: false })
                .populate({
                    path: 'grade',
                    populate: {
                        path: 'classroomTeacher',
                        select: 'name _id email',
                    },
                });
        }

        if (filterField && term) {
            users = users.filter((user) => (
user[filterField].toLowerCase().indexOf(term.toLowerCase()) > -1));
        }

        if (teachers === 'true') {
            users = users.filter((user) => (
                user.isStudent === false
```

```
            ));
        }


        res.status(200).json(users);
    } catch (e) {
        errorHandler(res, e);
    }
};

module.exports.getById = async (req, res) => {
    try {
        const user = await User.findById(req.params.id)
            .populate({
                path: 'grade',
                populate: {
                    path: 'classroomTeacher',
                    select: 'name _id email',
                },
            });
        res.status(200).json(user);
    } catch (e) {
        errorHandler(res, e);
    }
};

module.exports.delete = async (req, res) => {
    try {
        await User.remove({ _id: req.params.id });
        res.status(200).json({
            message: 'User removed',
        });
    } catch (e) {
        errorHandler(res, e);
    }
```

```
};

module.exports.update = async (req, res) => {
    try {
        let updated = {};

        if (req.user.role === 'USER') {
            if (req.body.email) updated.email = req.body.email;
            if (req.body.name) updated.name = req.body.name;
            if (req.body.grade) updated.grade = req.body.grade;
        } else {
            updated = {
                ...req.body,
            };
        }

        if (req.file) {
            updated.avatar = req.file.path;
        }

        if (req.body.isStudent && req.body.isStudent === false) {
            updated.grade = null;
        }

        const user = await User.findOneAndUpdate(
            { _id: req.params.id },
            { $set: updated },
            { new: true },
        ).populate({
            path: 'grade',
            populate: {
                path: 'classroomTeacher',
                select: 'name _id email',
            },
        });
```

```
        res.status(200).json(user);
    } catch (e) {
        errorHandler(res, e);
    }
};
```

Файл models > user.js

```
const mongoose = require('mongoose');

const { Schema } = mongoose;

const userSchema = new Schema({
    email: {
        type: String,
        required: true,
        unique: true,
    },
    name: {
        type: String,
        required: true,
    },
    avatar: {
        type: String,
        required: true,
        default: 'uploads/user-avatar.jpg',
    },
    password: {
        type: String,
        required: true,
    },
    role: {
        type: String,
        required: true,
        default: 'USER',
    },
```

```
    active: {
        type: Boolean,
        required: true,
        default: true,
    },
    isStudent: {
        type: Boolean,
        required: true,
        default: true,
    },
    grade: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'grade',
    },
});

module.exports = mongoose.model('users', userSchema);
```

Файл routers > user.js

```
const express = require('express');
const passport = require('passport');
const controller = require('../controllers/user');
const hasAccess = require('../middleware/access');
const { ROLES } = require('../roles-and-access');
const upload = require('../middleware/upload');

const router = express.Router();

router.get(
    '/',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.ADMIN),
    controller.getAll,
);
router.get(
```

```
    '/:id',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.USER),
    controller.getById,
);
router.delete(
    '/:id',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.ADMIN),
    controller.delete,
);
router.patch('/:id',
    passport.authenticate('jwt', { session: false }),
    hasAccess(ROLES.USER),
    upload.single('avatar'),
    controller.update);

module.exports = router;
```

Редагування профілю файл edit-profile.jsx

```
import React, { useEffect, useState } from "react";
import { connect } from "react-redux";
import { useForm } from "react-hook-form";
import Dialog from "@material-ui/core/Dialog";
import DialogContent from "@material-ui/core/DialogContent";
import DialogTitle from "@material-ui/core/DialogTitle";
import {
    TextField, InputLabel, Select,
} from "@material-ui/core";
import CloseIcon from "@material-ui/icons/Close";
import Button from "@material-ui/core/Button";
import { bindActionCreators } from "redux";
import useStyles from "./styles";
import editProfileActions from "../../actions/edit-profile.actions";
```

```
import withSchoolService from "../hoc/with-school-service";
import { emailErrors, nameErrors, regExp } from "../../validation";
import classActions from "../../actions/class.actions";
import "./style.scss";

const EditProfile = ({
    isProfileEditorOpen,
    closeEditor,
    currentUser,
    loading,
    updateUser,
    classesList,
    getAllClasses,
}) => {
    const classes = useStyles();
    const {
        register, handleSubmit, errors,
    } = useForm();
    const [name, setName] = useState("");
    const [email, setEmail] = useState("");
    const [grade, setGrade] = useState("");
    const [avatar, setAvatar] = useState("");

    useEffect(() => {
        if (currentUser) {
            setName(currentUser.name);
            setEmail(currentUser.email);
            setAvatar(currentUser.avatar);
            if (currentUser.grade) {
                setGrade(currentUser.grade._id);
            }
        }
    }, [currentUser]);

    useEffect(() => {
```

```
    getAllClasses();
}, [getAllClasses]);

const onSubmit = (data) => {
    const formData = new FormData();
    formData.append("name", data.name);
    formData.append("email", data.email);
    formData.append("grade", data.grade);
    if (avatar) formData.append("avatar", avatar);
    updateUser(currentUser._id, formData, true);
};

const onClose = () => closeEditor();

return (
    <Dialog
        open={isProfileEditorOpen}
        onClose={onClose}
        maxWidth="lg"
    >
        <div className="edit_profile">
            <div className={classes.flexContainer}>
                <DialogTitle>Редагування профілю</DialogTitle>
                <Button onClick={onClose}>
                    <CloseIcon />
                </Button>
            </div>
            <DialogContent>
                <div>
                    Обновіть деталі свого профілю
                </div>
                <form onSubmit={handleSubmit(onSubmit)}>
                    <TextField
                        defaultValue={name}
                        error={!!errors.name}
```

```
                            name="name"
                            label="Ім'я"
                            className={classes.input}
                            helperText={errors.name &&
nameErrors[errors.name.type]}
                            inputRef={register({ required: true,
pattern: regExp.name })}
                        />
                        <TextField
                            defaultValue={email}
                            error={!!errors.email}
                            name="email"
                            label="Email"
                            className={classes.input}
                            helperText={errors.email &&
emailErrors[errors.email.type]}
                            inputRef={register({ required: true,
pattern: regExp.email })}
                        />
                        <InputLabel htmlFor="age-native-
simple">Класс</InputLabel>
                        <Select
                            native
                            name="grade"
                            defaultValue={grade}
                            className={classes.input}
                            inputRef={register}
                        >
                            <option aria-label="None" value="" />
                            {classesList.map((item) => (
                                <option key={item._id}
value={item._id}>{item.name}</option>
                            ))}
                        </Select>
                        <input
                            accept="image/*"
                            style={{ display: "none" }}
                            id="upload-avatar"
```

```
                            multiple
                            type="file"
                            onChange={(e) => {
                                if (e.target.files !== null)
setAvatar(e.target.files[0]);
                            }}
                        />
                        <label htmlFor="upload-avatar">
                            <Button className="button_avatar"
disabled={loading} component="span">Оновити Аватар</Button>
                        </label>
                        <Button disabled={loading} type="submit">
                            Оновити
                        </Button>
                    </form>
                </DialogContent>
            </div>


        </Dialog>
    );
};


const mapStateToProps = ({
    editProfileReducer: {
        editProfileError, editProfileMessage, isProfileEditorOpen,
loading,
    },
    authReducer: { currentUser },
    classReducer: { classesList },
}) => ({
    isProfileEditorOpen,
    currentUser,
    editProfileError,
    editProfileMessage,
    loading,
    classesList,
```

```
});


const mapDispatchToProps = (dispatch, { schoolService }) =>
bindActionCreators({
    closeEditor: () => editProfileActions.closeProfileEditor(),
    getAllClasses: () => classActions.getAllClasses(schoolService)(),
    updateUser: (
        id,
        data,
        isForm,
    ) => editProfileActions.updateProfileData(schoolService, id, data,
isForm)(),
}, dispatch);


export default withSchoolService()(connect(mapStateToProps,
mapDispatchToProps)(EditProfile));
```

Файл edit-profile.action.js

```
const User = require('../models/User');
const errorHandler = require('../utils/error-handler');


module.exports.getAll = async (req, res) => {
    try {
        const { showMode } = req.query;
        const { filterField } = req.query;
        const { term } = req.query;
        const { teachers } = req.query;
        let users;

        if (showMode === 'all') {
            users = await User.find()
                .populate({
                    path: 'grade',
                    populate: {
                        path: 'classroomTeacher',
```

```
                    select: 'name _id email',
                },
            });
        }
        if (showMode === 'active') {
            users = await User.find({ active: true })
                .populate({
                    path: 'grade',
                    populate: {
                        path: 'classroomTeacher',
                        select: 'name _id email',
                    },
                });
        }
        if (showMode === 'inactive') {
            users = await User.find({ active: false })
                .populate({
                    path: 'grade',
                    populate: {
                        path: 'classroomTeacher',
                        select: 'name _id email',
                    },
                });
        }

        if (filterField && term) {
            users = users.filter((user) => (
user[filterField].toLowerCase().indexOf(term.toLowerCase()) > -1));
        }

        if (teachers === 'true') {
            users = users.filter((user) => (
                user.isStudent === false
            ));
        }
```

```
        res.status(200).json(users);
    } catch (e) {
        errorHandler(res, e);
    }
};

module.exports.getById = async (req, res) => {
    try {
        const user = await User.findById(req.params.id)
            .populate({
                path: 'grade',
                populate: {
                    path: 'classroomTeacher',
                    select: 'name _id email',
                },
            });
        res.status(200).json(user);
    } catch (e) {
        errorHandler(res, e);
    }
};

module.exports.delete = async (req, res) => {
    try {
        await User.remove({ _id: req.params.id });
        res.status(200).json({
            message: 'User removed',
        });
    } catch (e) {
        errorHandler(res, e);
    }
};
```

```
module.exports.update = async (req, res) => {
    try {
        let updated = {};

        if (req.user.role === 'USER') {
            if (req.body.email) updated.email = req.body.email;
            if (req.body.name) updated.name = req.body.name;
            if (req.body.grade) updated.grade = req.body.grade;
        } else {
            updated = {
                ...req.body,
            };
        }

        if (req.file) {
            updated.avatar = req.file.path;
        }

        if (req.body.isStudent && req.body.isStudent === false) {
            updated.grade = null;
        }

        const user = await User.findOneAndUpdate(
            { _id: req.params.id },
            { $set: updated },
            { new: true },
        ).populate({
            path: 'grade',
            populate: {
                path: 'classroomTeacher',
                select: 'name _id email',
            },
        });
        res.status(200).json(user);
    } catch (e) {
```

```
        errorHandler(res, e);
    }
};
```

Файл edit-profile.reducer.js

```
const initialState = {
    loading: false,
    email: null,
    name: null,
    grade: null,
    editProfileError: false,
    editProfileMessage: "",
    isProfileEditorOpen: false,
};

const editProfileReducer = (state = initialState, action) => {
    switch (action.type) {
    case "SET_PROFILE_NAME":
        return {
            ...state,
            name: action.payload.name,
        };
    case "SET_PROFILE_EMAIL":
        return {
            ...state,
            email: action.payload.email,
        };
    case "SET_PROFILE_GRADE":
        return {
            ...state,
            grade: action.payload.grade,
        };
    case "UPDATE_PROFILE_DATA_REQUEST":
        return {
            ...state,
```

```
            editProfileError: false,
            editProfileMessage: "",
            loading: true,
        };
    case "UPDATE_PROFILE_DATA_SUCCESS":
        return {
            ...state,
            editProfileError: false,
            editProfileMessage: "",
            loading: false,
        };
    case "UPDATE_PROFILE_DATA_ERROR":
        return {
            ...state,
            editProfileError: true,
            editProfileMessage: action.payload.message,
            loading: false,
        };
    case "OPEN_PROFILE_EDITOR":
        return {
            ...state,
            isProfileEditorOpen: true,
        };
    case "CLOSE_PROFILE_EDITOR":
        return {
            ...state,
            isProfileEditorOpen: false,
        };
    default:
        return state;
    }
};

export default editProfileReducer;
```