МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Сайт для загальноосвітньої школи № 1 м. Конотоп»

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

Керівник роботи — **Бабій М.С.**

Студента групи ІН – 64 — **Дяченко Д.О.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-64 спеціальності "Інформатика" денної форми навчання Дяченка Дениса Олеговича.

Тема:" **Сайт для загальноосвітньої школи № 1 м. Конотоп** "

Затверджена наказом по СумДУ

№_____ от _____2020 г.

Зміст пояснювальної записки: 1) аналітичний огляд відомих інформаційних сайтів, які використовують навчальні заклади; 2) постановка задачі та формування плану дослідження; 3) опис основних інструментів та технологій для створення веб сайту; 5) розробка веб сайту; 6) аналіз результатів роботи.

Дата видачі завдання "_____"_____2020 г.

Керівник випускної роботи _____ Бабій М.С.

Завдання приняв до виконання _____ Дяченко Д.О.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	02.03.20 – 05.03.20	
3	Визначення вимог	06.03.20 – 11.03.20	
4	Визначення інструментарію	12.03.20 – 13.03.20	
6	Складання календарного плану	14.03.20 – 15.03.20	
8	Розробка інтерфейсу користувача	16.03.20 – 03.04.20	
9	Проектування бази даних	04.04.20 – 06.04.20	
	Написання серверної частини	07.04.20 – 10.04.20	
12	Тестування розробником	11.05.20 – 15.05.20	
14	Оформлення документації	16.03.20 – 30.05.20	
16	Введення в експлуатацію	31.05.20 – 31.05.20	

Студент _____ Дяченко Д.О.
(підпис)

Керівник роботи _____ Бабій М.С.
(підпис)

РЕФЕРАТ

Записка: 67 стор., 12 рис., 6 додатків, 14 джерел.

Об’єкт дослідження — електронні інформаційні ресурси навчальних закладів.

Мета роботи — розробка зручного веб-сайту школи для Конотопської гімназії.

Методи дослідження — аналіз сучасного стеку технологій для створення швидкого односторінкового сайту.

Результати — розроблено інформаційну систему, що здатна забезпечити зручне ведення журналу з оцінками. Також веб-додаток не обмежує у форматах файлів, тож публікувати можна буде не тільки таблиці журналу, а й текстові документи, відео, презентації, архіви та навіть вкладені каталоги. Розроблено веб-сайт для Конотопської гімназії. На сайті є можливість додавати та переглядати новинами, можливість зареєструватися та переглянути розклад занять. Готовий додаток було завантажено на хостинг.

ІНФОРМАЦІЙНИЙ РЕСУРС, MERN, REACT, REDUX, NODE.JS, MONGODB, EXPRESS

Зміст

Вступ

З поширенням розвитку інформаційних технологій все більше навчальних закладів створює свої особисті веб сайти для того щоб ділитися новою інформацією, новинами та різного роду матеріалами. Сайт навчального закладу як і будь який інший веб сайт має бути зручним для користувача і містити тільки необхідні функції.

На сьогоднішній день майже всі школи мають свій веб сайт на якому вони розміщують найнеобхіднішу інформацію для своїх учнів та їх батьків. Це дуже зручно отримувами всю необхідну інформацію за допомогою сайту. Завдяки сайтам люди можуть отримувати всю саму необхідну інформацію.

Головною ідеєю проекту є створення веб сайту Конотопської гімназії на якому учні зможуть отримувати актуальну інформацію щодо навчального процесу та різні інші оголошення. Також є ідея ведення розкладу занять на особистому сайті. Ще однією причиною створення оновленого шкільного сайту було вдосконалення UI та UX, оскільки в попередньому було занадто багато непотрібних віджетів та мобільна версія не була адаптивною, а оскільки більша кількість користувачів користуються переважно смартфонами, адаптація сайту під мобільні пристрої є просто необхідною.

1 Інформаційний огляд

1.1 Огляд існуючих рішень

Головне призначення сайту школи – це можливість зручно переглянути наступні речі:

- Останні новини школи;
- Загальні відомості школи
- Розклад занять.

Тому, дослідження проводилось саме серед таких сайтів, які мають вище перечислені можливості.

Школа №8 м.Київ

Один із великої кількості сайтів шкіл. У нього є настіпні можливості:

- Регістрація/Авторизація;
- Перегляд новин;
- Перегляд історії школи;
- Перегляд розкладу занять.

Недоліки сайту:

- Немає адаптивності;
- Сайт виконаний у старому стилі, на якому дуже складно знайти необхідну інформацію.

Недоліки є достатньо суттєвими, оскільки це заважає користувачу шукати необхідну інформацію, та мати можливість користуватися сайтом з мобільних пристроїв.

Школа №1 м.Конотоп

У даного сайту є наступні переваги:

- Можливість авторизації;
- Перегляд новин;

- Корисні посилання.

Недоліки сайту:

- Немає можливості переглядати розклад занять;
- Важко знайти необхідну інформацію;
- Немає адаптивності.

Проаналізовані сайти мають великий перелік переваг, але також і дуже значні недоліки які потрібно виправити у новому сайті для Конотопської гімназії. За приклад будуть взяті саме ці сайти.

1.2 Постіновка задачі

Метою роботи є створення зручного сайту для перегляду новин, різної інформації та розкладу занять.

Основні вимоги до проекту:

- Можливість авторизації для перегляду більш детальної інформації;
- Перегляд останніх новин;
- Адаптивність;
- У користувача повинна бути одна з ролей, а саме public, user, moderator, admin.

Адміністратор та модератор повинні мати змогу створювати статті, редагувати та видаляти їх, незареєстровані та зареєстровані користувачі мають змогу лише переглядати статті.

В особистому кабінеті користувач може переглядати особисті дані, дані його класного керівника та розклад його занять.

Можливості ролі адміністратора:

- Створення та перегляд списків классів;
- Управління користувачами;
- Оновлення, створення, перегляд та видалення статей;

- Створення та перегляд розкладу занять.

Можливості ролі модератора:

- Оновлення, створення, перегляд та видалення статей;
- Створення та перегляд розкладу занять.

Можливості ролі зареєстрованого користувача:

- Перегляд статей;
- Перегляд особистої інформації;
- Перегляд особистого розкладу занять;
- Перегляд інформації класного керівника.

Користувачі повинні мати змогу зареєструватися без очікування підтвердження реєстрації адміністратором або модератором.

Сайт повинен бути адаптивним та кроссбраузерним для того щоб користувачі могли легко користуватися ним з будь яких пристроїв, та з будь якого браузера.

2 Вибір методу рішення

2.1 Огляд методів розробки

Можна виділити 5 методів розробки веб-сайту:

1. Ручний за допомогою чистих HTML, CSS, JS;
2. За допомогою програмних засобів розробки сайтів;
3. За допомогою CMS;
4. За допомогою фреймворків;
5. На SaaS платформах.

Ручний метод – Раніше це був дуже найпоширеніший метод. Але у цього методу є дуже великий недолік, цей спосіб дуже трудомісткий.

Програмні засоби – це готові програмні засоби котрі надають можливість генерувати html код та розробляти сайт у візуальному режимі. Серед найвідоміших можна виділити такі як:

• Adobe DreamWeaver – це мабуть один з найкращих візуальних редакторів, котрі генерують HTML код. Користуватися ним можна в декількох режимах, у візуальному, в якому не потрібно мати навичок роботи з кодом, та напряму з HTML розміткою. Не дивлячись на це все у програми є один достатньо серйозний недолік, це те що програма генерую дуже великий за об’ємом код, в якому дуже багато зайвого. Маючи знання HTML та CSS, можна відредагувати створений програмою сайт. Ця програма дуже добре підходить для людей які не мали великого досвіду написання розмітки веб сайтів.

• Microsoft FrontPage - це достатньо простий в опануванні і зручний Web-редактор для проектування, редагування, підготовки та публікації Web-сайтів. Програма інтегрована з пакетом продуктів MS Office, має дуже простий інтерфейс та дуже велику кількість шаблонів які дозволяють початківцю дуже швидко розібратися в цій програмі та почати створювати свої власні веб додатки. Але ця програма створена не лише для початківців, оскільки вона надає великі

можливості в роботі та дозволяє зручно розробляти додатки колективно. Вона надає можливість створювати веб додатки майже будь якої складності.

CMS (англ. Content management system, CMS) – це система для керування контентом, тобто додаток або система, що надає можливість публікувати та редагувати інформацію на сайті та керувати його функціоналом, також CMS часто називають двигуном сайту. Структура CMS зазвичай модульна, є базовий движок на який встановлюються різні модулі, теми та плагіни які дозволяють наповнити сайт його функціоналом. Внутрішня система CMS складається з двух частин, а саме з внутрішньої, яка відповідає за функціонал та збереження інформації та зовнішньої, яка відповідає за взаємодію з користувачем.
Найбільш розповсюджені CMS системами:

• WordPress – надає можливість користувачу створювати веб додатки будь якого типу, та має дуже гарну розширюваність. Він має дуже великі переваги, а саме те що він дуже легко встановлюється, дуже легко змінювати візуальні теми та шаблони та має гарні SEO показники. На даний час WordPress є найпопулярнішою CMS. Приблизно 35% сайтів у світі користується саме цією системою менеджменту контентом. Загалом, система управління контентом це додаток, що надає можливість власникам сайтів, авторам та редакторам керувати своїми сайтами власноруч та публікувати новий контент без аби яких знань програмування. WordPress використовує саме PHP і MySQL, оскільки вони підтримуються більшою частиною хостинг провайдерами. Використовуючи спеціальні тарифні плани для хостингу сайту написаного на WordPress можна забезпечити кращу швидкість, продуктивність та надійність. Зазвичай цю CMS використовують для створення блогу, але дуже легко перетворити його на лендінг, інтернет магазин чи портфоліо. Однією з головних особливостей WordPress є дуже зрозумілий та інтуїтивний інтерфейс. WordPress дуже легко освоїти.

• Drupal - безкоштовна CMS, з хорошою SEO-адекватністю, безпекою та розширюваністю. Движок широко використовується при побудові сайтів різного

призначення. Його використано при розробці сайтів сенату США та потужного сайту рейтингування Університетів світу Webometrics. Drupal це гнучка CMS заснована на LAMP стеку, з модульною структурою, що дозволяє додавати і видаляти функціонал через установку і видалення модулів і дозволяє змінити зовнішній вигляд сайту через установку і видалення тем оформлення. Основа Drupal, відома як ядро Drupal, містить PHP скрипти необхідні для запуску основного функціоналу CMS, декількох додаткових модулів і тим, і безлічі JavaScript, CSS і файлів зображень. Безліч додаткових модулів і тем оформлення ви можете завантажити з сайту Drupal.org.

Drupal також може бути запущений на інших технологічних стеках:

–Веб сервер може бути Nginx або IIS замість Apache.

–Операційна система може бути Windows або Mac OS замість Linux.

–Операційні системи, веб сервери і бази даних можуть також бути іншими, але скрипти, які використовує Drupal, написані на PHP, будуть одні і ті ж.

–База даних може бути PostgreSQL або SQLite замість MySQL або інша MySQL-сумісна заміна, така як MariaDB або Percona.

• Joomla. Переваги Joomla:

–Великий функціоналом менеджера матеріалів, який надає можливість публікувати необмежену кількість матеріалів, причому з поділом за категоріями.

–Зручна система модулів, завдяки якій можна відображати в різних позиція необхідні дані.

–Інтуїтивно-зрозумілий інтерфейс панелі адміністратора, завдяки чому навіть новачок з легкістю зможе створити сайт на даній CMS.

–Універсальність і простота настройки. Кожен елемент системи - легко налаштовується під кожного користувача.

–Простота поновлення.

Недолік є високе навантаження на сервер і невисоку швидкість роботи. Проблеми з SEO-адекватністю.

Framework (web фреймворк) – це можна назвати каркасом, який створений для побудови динамічних веб сторінок, сервісів, ресурсів чи додатків. Він створений для полегшення розробки і позбавляє від необхідності написання рутинного коду. За допомогою фреймворку можна зпростити написаання клієнтського інтерфейсу(у фронтенді), простіше підключатися до БД та зменьшують повторювання коду. Для створення фронтенд частини зараз використовують:

–React;

–Angular 2.X;

–Vue.js.

SaaS-платформи для створення веб додатків - це можливість запустити досить простий веб-проект дуже швидко і дешево (ще й на умовах оренди і без необхідності орендувати хостинг хостингу). Щось більш подібне до Гугл сайтів, але зі значно більш широкими можливостями. Рішення підходить для звичайних сайтів, тимчасових проектів і для швидкої перевірки бізнес-ідей. SaaS-платформи, як і CMS, бувають достатньо специфічними (наприклад, лише для інтернет-магазинів) і універсальними (для всіх розповсюджених видів сайтів). Цей метод наразі активно розвивається та використовується.

Я обрав варіант створити сайт за допомогою фреймворків. Таким чином у мене буде достатньо контролю над проектом, та швидкість створення буде набагато більша ніж при створенні сайту вручну.

2.2 Вибір мови програмування

Для створення клієнтської частини сайту я буду використовувати такі мови:

- HTML5;
- CSS3;
- JavaScript.

Навідміну від клієнту, сервер можна написати на дуже великій кількості мов програмування. Оскільки для написання клієнту я використовую JS, то було б

дуже гарно використовувати ту саму мову для написання і серверної частини, і така можливість в мене є. Я можу написати сервер на Node.js.

Однією з найважливіших переваг платформи є асинхронність в поєднанні з подієвим підходом. Подієво-орієнтоване програмування засноване на якихось зовнішніх діях, на відміну від потокового. Таким чином виконання програми напряму залежить від дій користувача або надійшов в програму мережевого пакету і т.д.

Таке рішення спрощує програмування інтерактивних програм, які заснованих на роботі з використанням введення-виведення. Це можуть бути чати чи ігрові, інтерактивні веб-додатки або будь-які калькулятори, голосування, рейтинги і так далі.

2.3 Вибір фреймворку

Фреймворк диктує правила побудови архітектури додатку, задаючи на початковому етапі розробки поведінка за умовчанням, формуючи каркас, який потрібно буде розширювати і змінювати відповідно до зазначених вимог. Фреймворк може включати допоміжні програми, бібліотеки коду, мова сценаріїв і інше програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

Оскільки я вирішив робити веб-сайт за допомогою фреймворку, то для створення я вирішив взяти React, оскільки він водночас і достатньо легкий для вивчення і гнучкий навідміну від Angular. А ще у react community більше ніж у Vue.js, саме через це писати клієнтську частину на react буде набагато простіше.

Для створення серверної частини я буду використовувати Node.js. Node.js – Виконавча середа коду JavaScript поза браузера. Ця платформа дозволяє писати серверний код для динамічних веб-сторінок і веб-додатків, а також для програм командного рядка. За допомогою Node.js реалізується парадигма «JavaScript для всього». Вона передбачає використання однієї мови програмування для розробки веб-додатків замість застосування різних мов для

роботи над фронтенда і бекенд. Node.js - не окремий мову програмування, а платформа для використання JavaScript на стороні сервера. Якщо говорити про мову, то як для фронденда, так і для бекенд використовується один і той же JavaScript. Різниця тільки в наборі API, які використовують фронтендери і бекендери. Браузерні JavaScript використовує Web API, які забезпечують доступ до DOM і призначеного для користувача інтерфейсу сторінок і веб-додатків. Серверний JavaScript використовує API, що забезпечують доступ до файлової системи додатків, http-запитів, потокам. Тобто Node.js - це технологія для використання JS на бекенд.

Для створення серверної частини я обрав фреймворк Express. Наразі він найпопулярніший для створення серверу на Node.js.

2.4 Вибір середовища розробки

Термін IDE зустрічається досить часто і зовсім необов'язково означає інтерфейс підключення жорстких дисків. Якщо ця абревіатура вживається в контексті програмування або ж просто розмови про програмне забезпечення, то, швидше за все, вона розшифровується як Integrated Development Environment - інтегроване середовище розробки додатків.

Під інтегрованим середовищем розробки зазвичай розуміють комплексне засіб, що включає все необхідне програмісту для створення програмного забезпечення. Чіткої дефініції для цього терміна не існує: ніхто не скаже вам про той чи інший програмний продукт, що це середовище розробки менш інтегрована, а та - більш. Проте, існує певний "джентльменський набір" компонентів, які повинні бути присутніми в інтегрованих середовищах розробки. По-перше, це компілятор або інтерпретатор, по-друге - редактор вихідного коду програм (обов'язково хоча б з підтримкою підсвічування синтаксису того мови програмування, для якого призначена середа), ну а по-третє - відладчик.

Отладчик - це, мабуть, навіть більше істотна частина інтегрованого середовища розробки, ніж компілятор або інтерпретатор, оскільки нерідко саме налагодження програми стає самим складним і дорогим етапом її створення.

Звичайно, сучасні інтегровані середовища розробки пропонують програмістам набагато більше можливостей, ніж входять в описаний вище необхідний мінімум. Наприклад, багато сучасних IDE є візуальними - вони дозволяють створювати інтерфейс програми за допомогою мишки, точно в такому вигляді, в якому він постане потім користувачеві. IDE, які не є візуальними, вимагають від програміста писати спеціальний код, відповідальний за створення призначеного для користувача інтерфейсу програми.

Залежно від того, для яких платформ можна писати програми і на яких платформах працює сама IDE, середовища розробки поділяються на крос-платформні (підтримують роботу з різними платформами) або переносних залежні (ті, які працюють тільки з однією платформою). Класичний приклад крос-платформної середовища розробки - Eclipse, переносних залежною - Delphi.

Залежно від кількості підтримуваних мов програмування, середовища можуть бути багатомовними або одномовними.

Список популярних середовищ розробки великий, і всі значущі продукти цього класу динамічно розвиваються в сторону все більшої зручності для розробників.

Для розробки була обрана IDE Visual Studio Code від компанії Microsoft. Ліцензія є безплатною. Visual Studio Code є дуже зручним середовищем для розробки веб-сайтів.

2.5 Вибір бази даних

Для створення сайту школи я обрав нереляційну СУБД, а саме MongoDB. Вона має такі переваги:

- Динамічна схема – Як згадувалося вище, ця СУБД дозволяє гнучко працювати зі схемою даних без необхідності змінювати самі дані;
- Масштабованість – MongoDB горизонтально масштабируема, що дозволяє легко зменшити навантаження на сервера при великих обсягах даних;
- Зручність в управлінні – СУБД не потребує окремому адміністратора бази даних. Завдяки достатній зручності у використанні, їй легко можуть користуватися як розробники, так і системні адміністратори;
- Швидкість – Висока продуктивність при виконанні простих запитів;
- Гнучкість – В MongoDB можна без шкоди для існуючих даних, їх структури і продуктивності СУБД додавати поля або колонки.

2.6 Вибір середовища розміщення

Хостинг – це послуга яку надають різні сервіси для розміщення різних веб сайтів, додатків на сервері, який має постійний доступ до мережі. Хостинг потрібен для розміщення веб сайту в мережі с цілодобовим доступом до нього та документів які на ньому зберігаються. Існує декілька типів хостингу:

–VPS;

–Загальний;

–Хостинг виділених серверів;

–Хмарний.

Для постійної працездатності сайту, потрібно розміщення його на одному із веб хостингів та придбати доменне ім’я. Для даного проекту, було вирішено розмістити сайт на сервері ресурсу www.heroku.com. А оскільки сервіс heroku надає можливість обрати безкоштовне доменне ім’я я скористаюсь цією можливістю.

2.7 Вибір системи контролю версій

Системою контролю версій (СКВ) називається система, котра контролює та реєструє зміни в одному чи кількох файлах так, щоб пізніше можна було повернутися до необхідних версій цього файлу або файлів.

В наш час файли є кінцевим результатом для більшості професій (письменництво, архітектура, навчальні та наукові роботи і звісно розробка різноманітних програм). На розробку витрачається дуже велика частина часу і нікому не хотілося б витрачати стільки ж часу на відновлення втрачених даних після збою, тощо.

Наприклад, програміст розробляє скрипт чи додаток який складається з одного не дуже великого файлу. Після випуску альфа версії додатку, йому потрібно :

–Виправляти певні проблеми про які йому повідомляють користувачі першої версії;

–Працювати над новою версією.

Навіть якщо треба просто виправляти виникаючі проблеми, то велика ймовірність, що після будь-яких змін проект перестає працювати, і треба визначити, що було змінено, щоб було простіше локалізувати проблему.

Також гарним варіантом є вести якийсь журнал виконаних змін і виправлень, щоб не робити одну й ту ж роботу декулька разів.

В простішому випадку вищезгадану проблему можна вирішити зберіганням кількох копій файлів, наприклад, один для виправлення допущених помилок в першій версії проекту та другий для внесення нових змін. Так як зазвичай зміни не є дуже великими в порівнянні з розміром файлу, то можливо зберігати лише змінені рядки користуючись утилітою diff і потім об'єднувати їх використовуючи утиліту patch. Якщо проект складається з декількох тисяч файлів і над ним працює сотня осіб, то в цьому випадку доцыльныше використовувати метод зі зберіганням окремих копій файлів (чи тільки змін).

Якщо виконувати роботу графічного або веб-дизайнера то можна зберігати кожну версію зображення або макета, то використовувати систему контролю версій буде дуже доцільним рішенням. СКВ надає нам можливість повертати окремі документи до колишнього вигляду, чи повертати до попереднього стану весь проект, переглядати створені з часом зміни, та дізнаватись, хто останнім

вносив зміни у проект і чому раптово перестав працювати модуль, хто і коли вніс до коду якусь помилку, і таке інше. Взагалі, якщо, користуючись СКВ, зіпсувати або втратити файли, все можна буде легко відновити. До того ж, накладні витрати за все, що ви отримуєте, будуть дуже незначними.

Системи контролю версій:

– Локальні системи котролю версій – є один дуже часто зустрічаємий приклад, наприклад коли користувач копіює файли до іншого каталогу та додає наприкінці дату останніх змін. Цей приклад СКВ є дуже поширеним, але він дуже часто дає збої, оскільки людина може дуже легко помилитися та замінити помилково не той файл, неправильно назвати або скопіювати не в той каталог. Для вирішення цієї проблеми, вже давно створили локальні СКВ з базою даних, в якій зберігаються всі зміни необхідних файлів. Однією з найбільш популярних СКВ такого типу є RCS (Revision Control System), котра досі встановлюється на велику кількість комп'ютерів;

– Централізовані – це такі системи, наприклад як CVS, Perforce і Subversion, у яких є центральний сервер, на якому зберігаються всі файли під контролем версій, і ряд клієнтів, які отримують копії файлів з нього. Саме це було стандартом впродовж останніх років. Цей підхід в порівнянні з локальними СКВ має великий ряд переваг. Кожен може знати хто і чим саме займається в проекті. Адміністратори мають детальний контроль над тим, що і хто може робити. Також адмініструвати ЦСКВ простіше ніж локальні бази кожного клієнта. Попри все в даному варіанті є також і декілька мінусів. Звичайно тепер коли все зберігається на центральному сервері він є вразливим місцем цілої системи. У разі вимкнення сервера весь час доки він не працюєніхто не може зберегти оновлену версію своєї праці. А в разі пошкодження диску с центральною БД, то втрачається вся історія проекту;

– CVS – наразі є дуже широко використованою системою контролю версій, хоча вона має серйозні недоліки через які вона втрачає свою популярність. CVS використовує клієнт-сервер архітектуру в якій вся інформація про версії

зберігається на сервері. Використання клієнт-сервер архітектури дозволяє використовувати CVS навіть географічно розподіленим командами користувачів де кожен користувач має свій робочий директорій з копією проекту;

– Розподілені системи контролю версій - це такі системи як Git та Mercurial. Користувачі не лише завантажують останню версію файлу, вони повністю копіюють репозиторій. Через це, коли «падає» сервер, будь який клієнтський репозиторій може бути скопійований на сервер. Кожен раз, коли клієнт забирає свіжу версію файлів, він створює собі повну копію всіх даних. Крім того, в більшій частині цих систем можна працювати з декількома віддаленими репозиторіями, таким чином, можна одночасно працювати по-різному з різними групами людей в рамках одного проекту.

GitHub один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git.

3 Програмна реалізація

3.1 Структура проекту

Структура проекту зображена на рисунку 3.1.

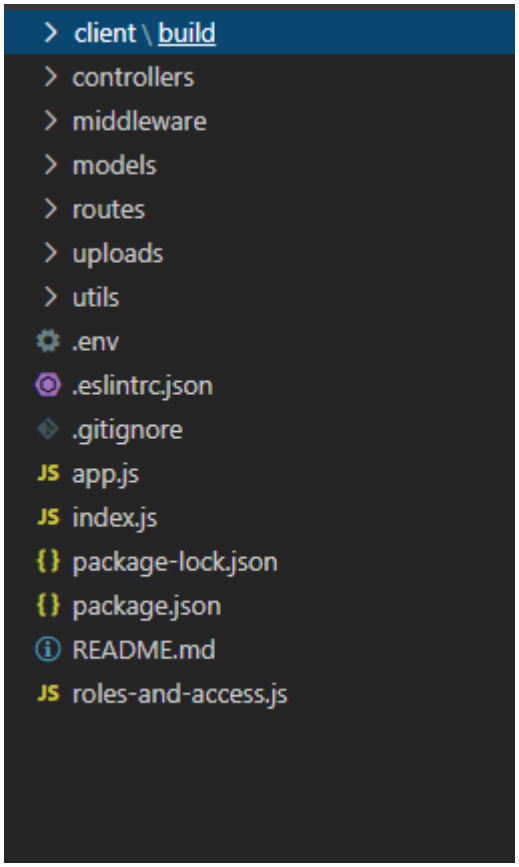

Рисунок 3.1 – Структура проекту

Опис файлів та каталогів:

- client – папка в якій зберігається клієнтська частина веб сайту;
- controllers – папка в якій зберігаються контроллери;
- middleware – папка для middlewares;
- models – папка моделей БД;
- routes – папка всіх роутів створеної API;
- uploads – папка завантажених фото;
- utils – папка утиліт;
- .env – конфігурація;
- .eslintrc.json – правила для js лінтера;

- .gitignore – файл в якому описуються каталоги та файли які не будуть завантажуватись до репозиторію;
- app.js – конфігурація сервера, підключення бібліотек та ін.;
- index.js – точка входу до програми;
- roles-and-access.js – файл для перевірки ролей та доступу до ресурсів;
- package.json – опис проекту, версія, список залежностей для роботи.

3.2 Реалізація авторизації на сервері

Для аутентифікації буде використаний звичайний middleware Passport.js. Passport – це middleware для перевірки істинності при доступі до приватних роутів. У passport.js є велика кількість стратегій перевірки авторизації, але я обрав перевірку по JWT (JSON Web Token). При авторизації користувач отримує токен який записується до localStorage та перевіряється при кожному запиті на приватний роут. Нижче я наведу приклад написаного middleware для перевірки токена.

```
const JwtStrategy = require('passport-jwt').Strategy;
const { ExtractJwt } = require('passport-jwt');
const User = require('../models/User');

const options = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: process.env.JWT,
};

module.exports = (passport) => {
    passport.use(new JwtStrategy(options, async (payload, done) => {
        try {
        const user = await User.findById(payload.userId).select('ema
il role id');
```

```
            if (user) {
                done(null, user);
            } else {
                done(null, false);
            }
        } catch (e) {
            console.log(e);
        }
    }));
};
```

У випадку якщо час дії токена вийшов то клієнта перекидує на сторінку авторизації, щоб він повторно міг зайти в систему.

3.3 Збереження стану додатку на клієнті

Оскільки сайт школи достатньо великий за об’ємом, доцільніше буде зберігати стан кожного компоненту окремо від нього в одному глобальному стейті. Так буде простіше зробити обмін інформацією між компонентами. Якщо зберігати всю інформацію про роботу додатку в одному головному компоненті, то щоб передати інформацію до третього дочірнього компоненту в ланцюгу вкладеності, нам потрібно буде перекидувати інформацію через всі батьківські компоненти цієї компоненти. Саме через це буде простіше зберігати всю інформацію додатку в одному місці і якщо компоненту потрібні будуть якісь дані то він зможе звернутися до цього головного сховища та отримати необхідну інформацію. Для цього я буду використовувати Redux.

Сховище буде зберігати всю інформацію про проект і кожен компонент зможе отримати інформацію з нього напряму. Для зміни свого стану, компонент повинен відправити action і якщо все виконано правильно то компонент отримає нові дані та перемалює свій UI.

Код action-ів авторизації виглядає приблизно наступним чином:

```
const loginRequest = () => ({
    type: "LOGIN_REQUEST",
});
```

```
const loginSuccess = (currentUser) => ({
    type: "LOGIN_SUCCESS",
    payload: currentUser,
});

const loginError = (message) => ({
    type: "LOGIN_ERROR",
    payload: {
        message,
    },
});
```

Ми маємо дії для всих трьох станів, а саме запит, успіх та помилка. Нижче наведений код Reducer-а

```
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;
    }
};
```

3.4 Результат роботи

У результаті виконання практичної роботи було реалізовано частину проекту, а саме - основний функціонал: головну сторінку, механізм авторизації, особистий кабінет та управління статтями, классами та користувачами.

На рисунку 3.2 зображено головну сотрінку. На головній сторінці знаходиться :

- Шапка сайту;
- Галерея фотографій;
- Секція статей.

В шапці сайту є кнопка авторизації/регістрації. Форма авторизації зображена на рисунку 3.3. Для входу зареєстрованому користувачеві необхідно ввести його Email та пароль, а для реєстрації необхідно ввести пошту користувача, його ім’я та пароль для входу в систему.

На рисунку 3.4 зображена панель управління користувачами. Адміністратор може бачити певну інформацію про користувача:

- Ім’я;
- Пошту;
- Роль;
- Статус (студент або вчитель);
- Активність (чи є запис активним або заблокованим).

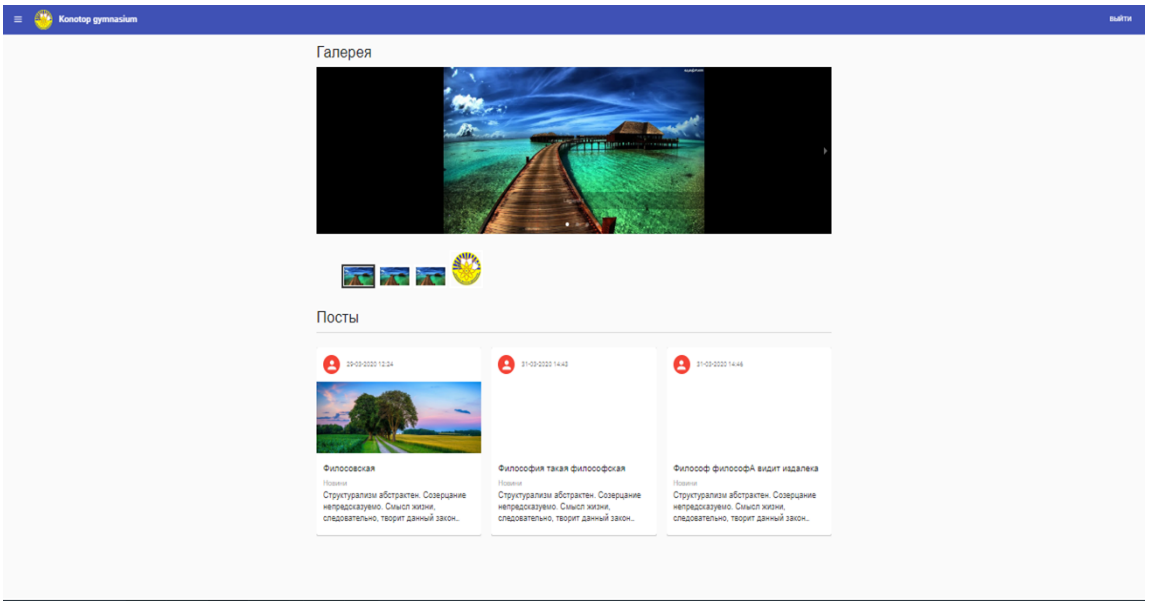

Рисунок 3.2 – Скріншот головної сторінки з ПК

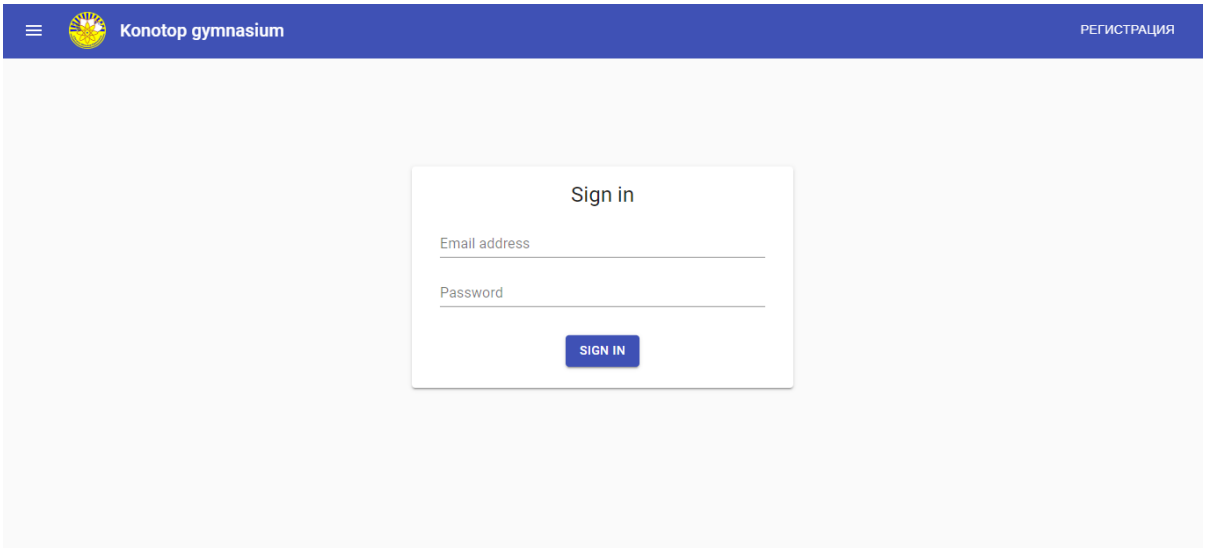

Рисунок 3.3 – Сторінка авторизації

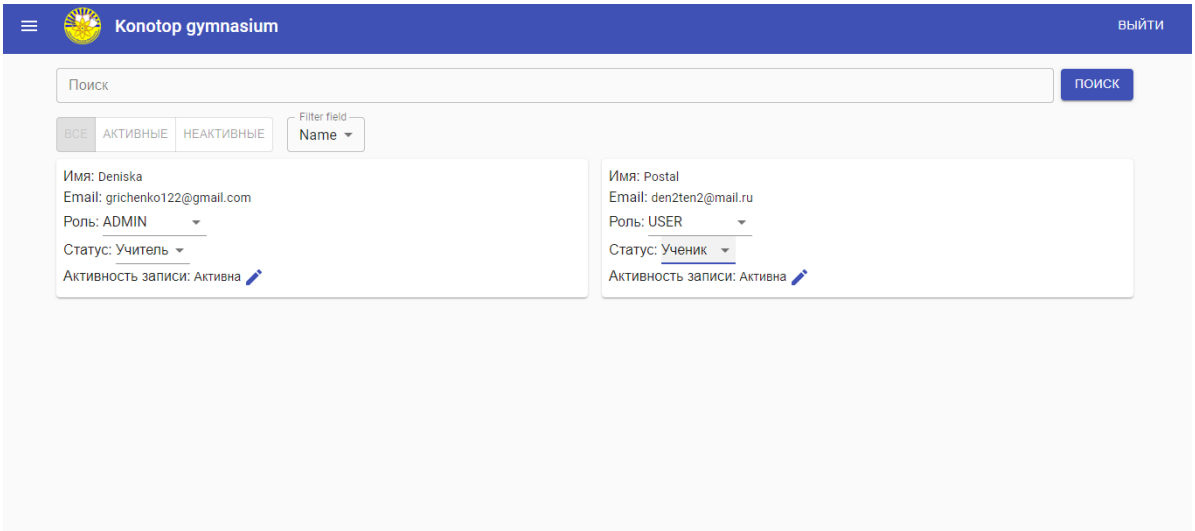

Рисунок 3.4 – Управління користувачами

На рисунку 3.6 зображений особистий й кабінет користувача. На боковій панелі видно інформацію про поточного користувача:

– Ім’я;
– Клас;
– Пошта;
– Ім’я класного керівника;
– Пошта класного керівника.

Також користувач може змінювати особисту інформацію в налаштуваннях профілю. Форма налаштувань зображена на рисунку 3.5.

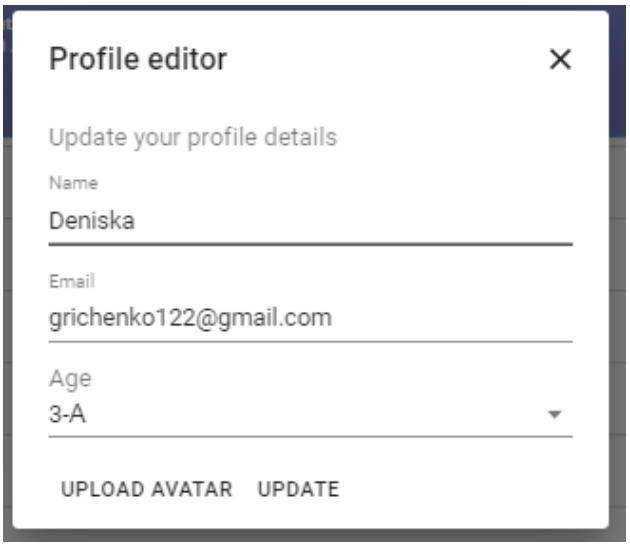

Рисунок 3.5 – Налаштування профілю

З іншої сторони зображений розклад уроків на день. Користувач може обрати день на який він хоче переглянути розклад.

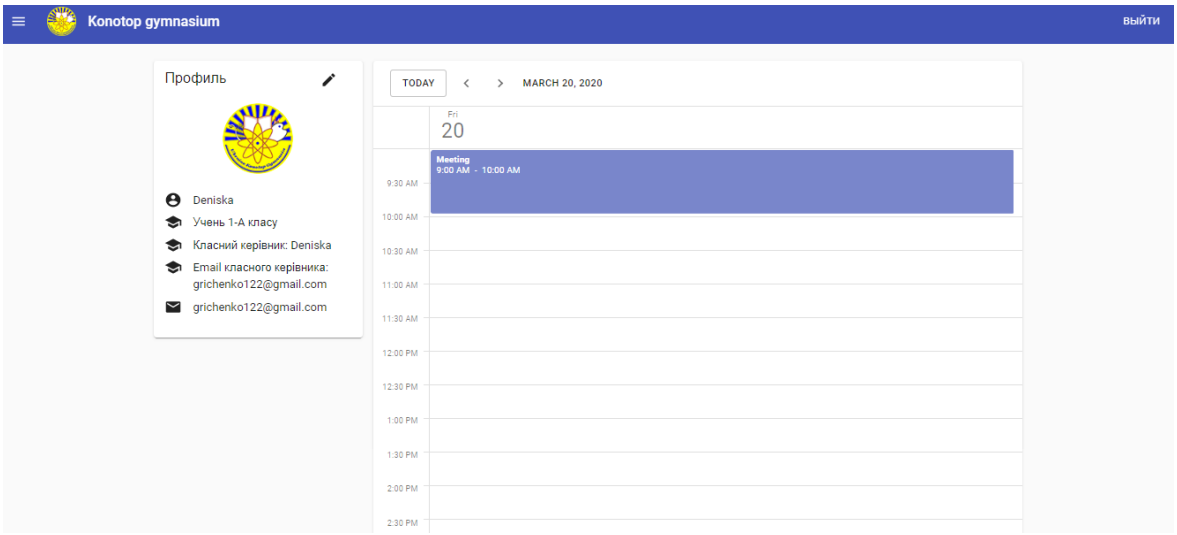

Рисунок 3.6 – Особистий кабінет

На рисунку 3.7 зображена панель управління класами. Адміністратор може додавати та видаляти класи, а також закріпити класного керівника за певним класом.

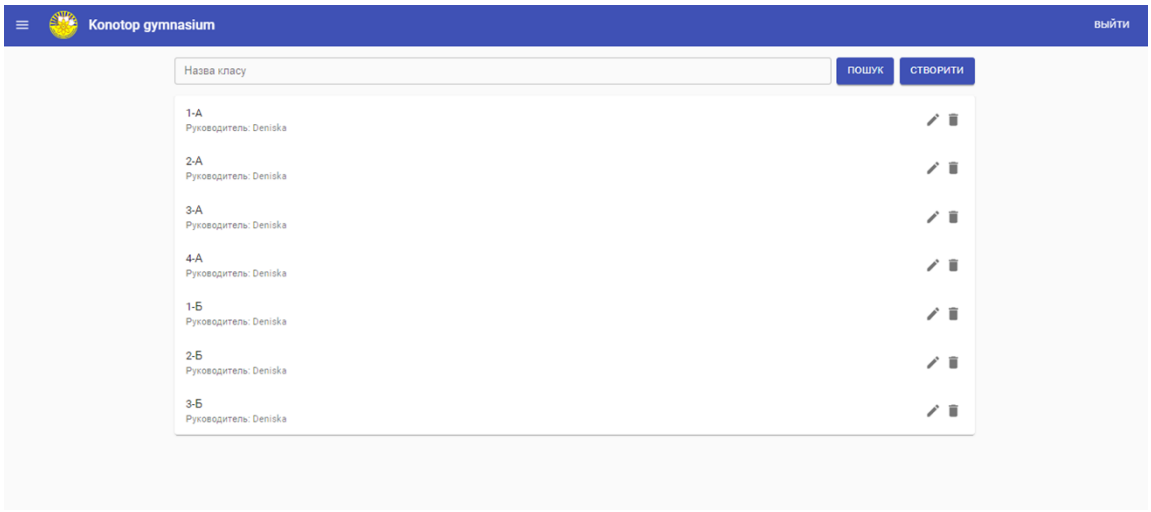

Рисунок 3.7 – Управління класами

На рисунку 3.8 зображена панель управління статтями. Адміністратор та модератор мають право створювати, редагувати та видаляти статті, також можна шукати статті по фрагменту назви.

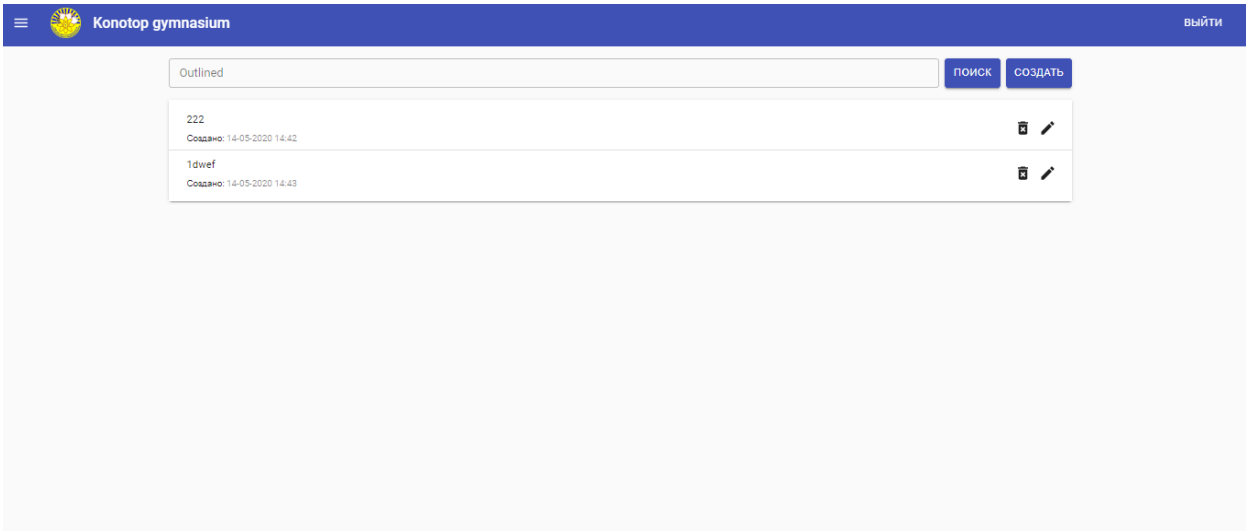

Рисунок 3.8 – Управління статтями

На рисунку 3.9 зображений редактор статті. Користувач повинен заповнити наступні поля:

– Назва статті;
– Текст статті;
– Додати картинку (опціонально).

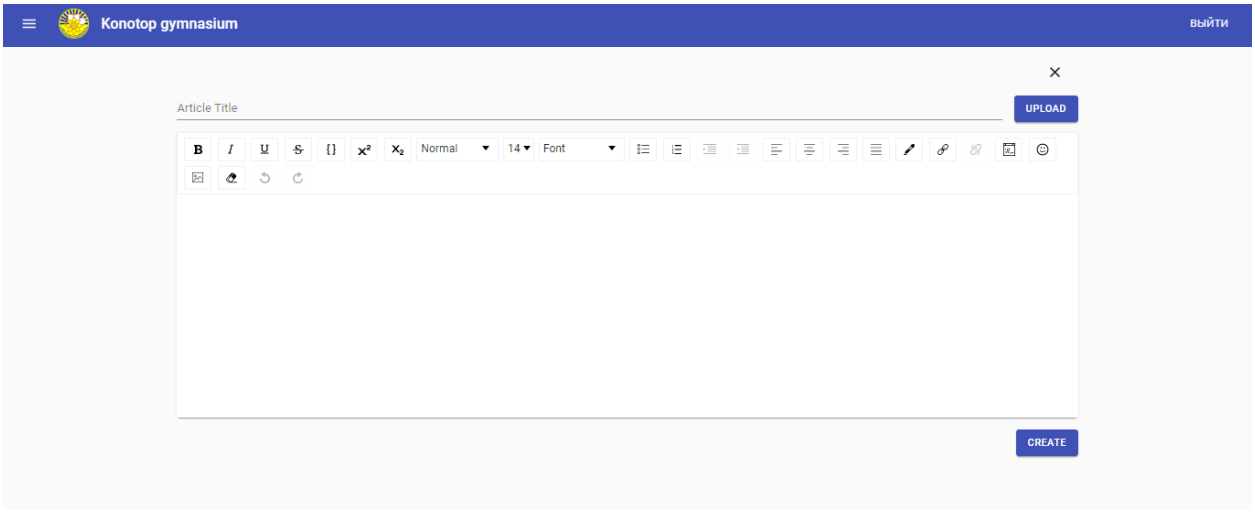

Рисунок 3.9 – Редактор статті

Виконаний сайт э адаптивним під мобільні пристрої. На рисунку 3.10 та 3.11 зображений сайт на мобільному пристрою.

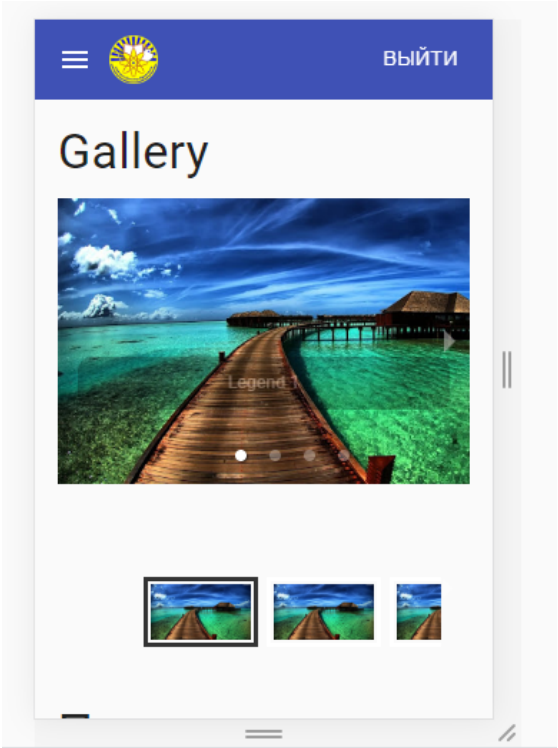

Рисунок 3.10 – Мобільний вид

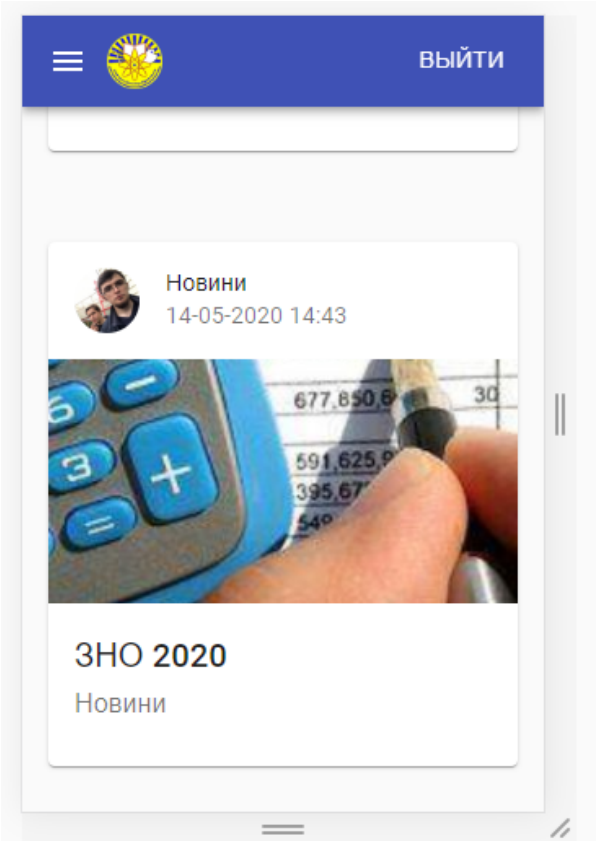

Рисунок 3.11 – Мобільний вид

На рисунку 3.12 зображена створена стаття. Стаття складається з заголовку, картинки, тексту та дати створення.

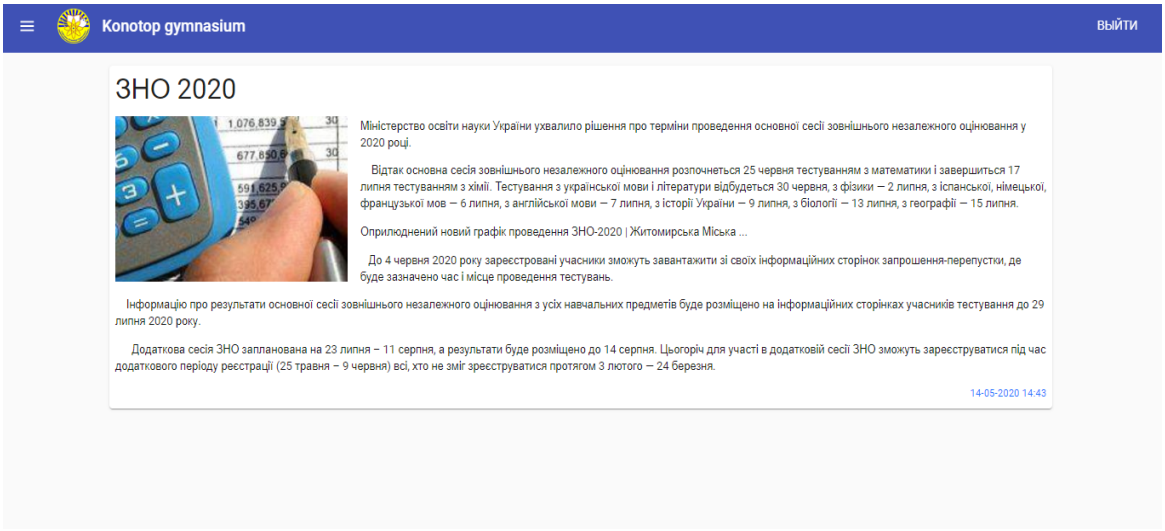

Рисунок 3.12 – Вигляд створеної статті

Висновок

З кожним новим днем інформаційні технології все більше і більше розвиваються. Завдяки стрімкому розвитку інформаційних технологій, майже всі заклади вже частково працюють в режимі онлайн. В наш час всі повинні мати доступ до будь-якої корисної інформації не виходячи з дому. Все більше підвищуються вимоги до навчальних закладів, і вже майже неможливо навчатися чомусь новому не використовуючи при цьому інтернет.

Саме завдяки інтернету ми можемо продовжувати отримувати знання під час дистанційного навчання. І взагалі воно було б не можливе якби не існувало інформаційних технологій.

Виконаний, в ході практичної роботи, проект можна буде покращувати та доводити до ідеалу. Завдяки ньому учням буде набагато легше слідкувати за новинами свого навчального закладу, що дуже вагомо полегшить їм навчання.

Список літератури

1. Методи розробки веб сайтів [Електронний ресурс]. – Режим доступу: https://sites.google.com/site/tz5103voinovakateryna/metodi-rozrobki-web-sajtiv

2. SQL против NoSQL на примере MySQL и MongoDB [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/sql-vs-nosql/

3. Інформаційні технології [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D1%96_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97

4. Чем хорош Node.js и зачем он нужен [Електронний ресурс]. – Режим доступу: https://support.google.com/accounts/answer/112802?co=GENIE.Platform%3DDesktop&hl=ru

5. Сайт конотопської гімназії [Електронний ресурс]. – Режим доступу: https://gymnasia.osvita-konotop.gov.ua/

6. Що таке Joomla [Електронний ресурс] – Режим доступу: https://joomla.ru/docs/administrator/joomla3-start/1742-chto-takoe-joomla

7. Що таке Drupal [Електронний ресурс] – Режим доступу: https://www.drupal.org/ru/docs/user_guide/ru/understanding-drupal.html

8. Що таке Wordpress [Електронний ресурс] – Режим доступу: https://www.hostinger.com.ua/rukovodstva/chto-takoe-wordpress-obzor-populjarnoj-cms/

9. Веб фреймворк Express [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs

10. Варіанти створення сайту [Електронний ресурс] – Режим доступу: https://alexsend.com.ua/blog/sposobi-sozdaniya-saita/

11. Вибір IDE для створення веб сайту [Електронний ресурс] – Режим доступу: https://blog.education-ecosystem.com/ide-%D0%B4%D0%BB%D1%8F-wordpress/

12. Выбор базы данных [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/348220/

13. Создание сайта с нуля [Електронний ресурс] – Режим доступу: https://alpha-byte.ru/sozdanie-sajta

14. Створення сайту на React [Електронний ресурс] – Режим доступу: https://coursehunter.net/course/react-redux-professionalnaya-razrabotka

Додаток А

Авторизація за допомогою Passport.js

```
const JwtStrategy = require('passport-jwt').Strategy;
const { ExtractJwt } = require('passport-jwt');
const User = require('../models/User');

const options = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: process.env.JWT,
};

module.exports = (passport) => {
    passport.use(new JwtStrategy(options, async (payload, done) => {
        try {
            const user = await User.findById(payload.userId).select(
'email role id');

            if (user) {
                done(null, user);
            } else {
                done(null, false);
            }
        } catch (e) {
            console.log(e);
        }
    }));
};
```

Додаток В

Збереження зображень на сервері

```
const multer = require('multer');
const moment = require('moment');

const storage = multer.diskStorage({
    destination(req, file, callback) {
        callback(null, 'uploads/');
    },
    filename(req, file, callback) {
        const date = moment().format('DDMMYYYY-HHmmss_SSS');
        callback(null, `${date}-${file.originalname}`);
    },
});

const fileFilter = (req, file, callback) => {
    if (file.mimetype === 'image/png'
        || file.mimetype === 'image/jpeg'
        || file.mimetype === 'image/jpg') {
        callback(null, true);
    } else {
        callback(null, false);
    }
};

const limits = {
    fileSize: 1024 * 1024 * 5,
};

module.exports = multer({
    storage,
    fileFilter,
    limits,
});
```

Додаток С

Авторизація на клієнті

```
//ACTIONS
import history from "../history";

const loginRequest = () => ({
    type: "LOGIN_REQUEST",
});

const loginSuccess = (currentUser) => ({
    type: "LOGIN_SUCCESS",
    payload: currentUser,
});

const loginError = (message) => ({
    type: "LOGIN_ERROR",
    payload: {
        message,
    },
});

const login = (
    schoolService,
    userInfo,
) => () => async (dispatch) => {
    dispatch(loginRequest());
    try {
        const user = await schoolService.login(userInfo);
        localStorage.setItem("school-user-with-
jwt", JSON.stringify(user));
        dispatch(loginSuccess(user));
        history.push("/");
    } catch (e) {
        dispatch(loginError(e.message));
    }
};
```

```
const checkAuthorization = () => (dispatch) => {
    const LSItem = localStorage.getItem("school-user-with-jwt");
    if (LSItem) dispatch(loginSuccess(JSON.parse(LSItem)));
};

const logoutRequest = () => ({
    type: "LOGOUT",
});

const logout = () => (dispatch) => {
    localStorage.removeItem("school-user-with-jwt");
    dispatch(logoutRequest());
    history.push("/login");
};

const registrationRequest = () => ({
    type: "REGISTRATION_REQUEST",
});

const registrationSuccess = () => ({
    type: "REGISTRATION_SUCCESS",
});

const registrationError = (message) => ({
    type: "REGISTRATION_ERROR",
    payload: {
        message,
    },
});

const registration = (
    schoolService,
    userInfo,
) => () => async (dispatch) => {
    dispatch(registrationRequest());
    try {
        await schoolService.registration(userInfo);
```

```
            dispatch(registrationSuccess());
            history.push("/login");
        } catch (e) {
            dispatch(registrationError(e.message));
        }
};

const clearAuthError = () => ({
    type: "CLEAR_AUTH_ERROR",
});

const authActions = {
    login,
    logout,
    registration,
    clearAuthError,
    checkAuthorization,
    loginSuccess,
};

export default authActions;

// REDUCER
const loginRequest = (state) => ({
    ...state,
    authentication: true,
    authError: false,
    isAuth: false,
});

const loginSuccess = (state, action) => ({
    ...state,
    authentication: false,
    authError: false,
    isAuth: true,
    currentUser: action.payload,
});
```

```
const loginError = (state, action) => ({
    ...state,
    authentication: false,
    isAuth: false,
    authError: true,
    errorMessage: action.payload.message,
});

const logout = (state) => ({
    ...state,
    isAuth: false,
    currentUser: {},
});

const registrationRequest = (state) => ({
    ...state,
    authentication: true,
    authError: false,
    isAuth: false,
});

const registrationSuccess = (state) => ({
    ...state,
    authentication: false,
    authError: false,
    isAuth: false,
});

const registrationError = (state, action) => ({
    ...state,
    authentication: false,
    isAuth: false,
    authError: true,
    errorMessage: action.payload.message,
});
```

```
const clearAuthError = (state) => ({
    ...state,
    authError: false,
    errorMessage: "",
});

const initialState = {
    isAuth: false,
    authentication: false,
    authSuccess: false,
    authError: false,
    currentUser: {},
    errorMessage: "",
};

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;
    }
};
```

```
    export default authReducer;

    //LOGIN
    const LogIn = ({ login, authentication }) => {
        const classes = useStyles();

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

        const onSubmit = (data) => {
            login(data);
        };

        return (
            <form className={classes.form} onSubmit={handleSubmit(onSubm
it)}>
                <Typography variant="h5" align="center" gutterBottom>
                    Sign in
                </Typography>
                <FormControl className={classes.email}>
                    <InputLabel htmlFor="email">Email address</InputLabe
l>
                    <Input
                        name="email"
                        type="email"
                        inputRef={register({ required: true, pattern: re
gExp.email })}
                    />
                    {errors.email
                        && (
                            <FormHelperText className={classes.helper} i
d="email-helper-text">
                                { emailErrors[errors.email.type] }
                            </FormHelperText>
                        )}
```

```
                </FormControl>
                <FormControl className={classes.input}>
                    <InputLabel htmlFor="password">Password</InputLabel>
                    <Input
                        name="password"
                        type="password"
                        inputRef={register({ required: true, minLength:
6, pattern: regExp.password })}
                    />
                    {errors.password
                        && (
                            <FormHelperText className={classes.helper} i
d="email-helper-text">
                                { passwordErrors[errors.password.type] }
                            </FormHelperText>
                        )}
                </FormControl>
                <Button
                    disabled={authentication}
                    type="submit"
                    variant="contained"
                    color="primary"
                    className={classes.button}
                >
                    Sign in
                </Button>
            </form>
        );
    };

    const mapStateToProps = ({ authReducer: { authentication } }) => ({
        authentication,
    });

    const mapDispatchToProps = (dispatch, ownProps) => bindActionCreator
s({
        login: (userInfo) => authActions.login(
```

```
            ownProps.schoolService,
            userInfo,
        )(),
    }, dispatch);

    //REGISTRATION
    const Registration = ({
        registration, authentication,
    }) => {
        const classes = useStyles();
        const {
            register, handleSubmit, watch, errors,
        } = useForm();

        const onSubmit = (data) => {
            const registrationData = {
                email: data.email,
                name: data.name,
                password: data.password,
            };
            registration(registrationData);
        };

        return (
            <form className={classes.form} onSubmit={handleSubmit(onSubm
it)}>
                <Typography variant="h5" align="center" gutterBottom>
                    Sign up
                </Typography>
                <FormControl className={classes.email}>
                    <InputLabel htmlFor="email">Email address</InputLabe
l>
                    <Input
                        name="email"
                        type="email"
                        inputRef={register({ required: true, pattern: re
gExp.email })}
```

```
                    />
                    {errors.email
                    && (
                        <FormHelperText className={classes.helper} id="e
mail-helper-text">
                            { emailErrors[errors.email.type] }
                        </FormHelperText>
                    )}
                </FormControl>
                <FormControl className={classes.input}>
                    <InputLabel htmlFor="name">Name</InputLabel>
                    <Input
                        name="name"
                        type="text"
                        inputRef={register({ required: true, minLength:
2, pattern: regExp.name })}
                    />
                    {errors.name
                    && (
                        <FormHelperText className={classes.helper} id="e
mail-helper-text">
                            { nameErrors[errors.name.type] }
                        </FormHelperText>
                    )}
                </FormControl>
                <FormControl className={classes.input}>
                    <InputLabel htmlFor="password">Password</InputLabel>
                    <Input
                        name="password"
                        type="password"
                        inputRef={register({ required: true, minLength:
6, pattern: regExp.password })}
                    />
                    {errors.password
                    && (
                        <FormHelperText className={classes.helper} id="e
mail-helper-text">
```

```
                                { passwordErrors[errors.password.type] }
                            </FormHelperText>
                        )}
                    </FormControl>
                    <FormControl className={classes.input}>
                        <InputLabel htmlFor="password-
confirm">Confirm password</InputLabel>
                        <Input
                            name="passwordConfirm"
                            type="password"
                            inputRef={register({ validate: (value) => value
=== watch("password") })}
                        />
                        {errors.passwordConfirm
                        && (<FormHelperText className={classes.helper} id="e
mail-helper-text">
                        { passwordConfirmErrors[errors.passwordConfirm.type]
 }
                            </FormHelperText>
                        )}
                    </FormControl>
                    <Button
                        type="submit"
                        variant="contained"
                        color="primary"
                        disabled={authentication}
                        className={classes.button}
                    >
                        Sign up
                    </Button>
                </form>
            );
        };
        const mapStateToProps = ({ authReducer: { authentication } }) => ({
            authentication,
        });
```

```
    const mapDispatchToProps = (dispatch, ownProps) => bindActionCreator
s({
        registration: (userInfo) => authActions.registration(
            ownProps.schoolService,
            userInfo,
        )(),
    }, dispatch);

    export default withSchoolService()(connect(mapStateToProps, mapDispa
tchToProps)(Registration));
```

Додаток D

Створення High Order Components (HOC)

```
import React from "react";
import { SchoolServiceConsumer } from "../school-service-context";

const withSchoolService = () => (Wrapped) => (props) => (
    <SchoolServiceConsumer>
        {
            (schoolService) => (
                <Wrapped {...props} schoolService={schoolService} />
            )
        }
    </SchoolServiceConsumer>
);

export default withSchoolService;
```

Додаток Е

Особистий кабінет користувача

```
import React from "react";
import { connect } from "react-redux";
import {
    Avatar, Paper, Typography, Button,
} from "@material-ui/core";
import AccountCircleIcon from "@material-ui/icons/AccountCircle";
import SchoolIcon from "@material-ui/icons/School";
import MailIcon from "@material-ui/icons/Mail";
import CreateIcon from "@material-ui/icons/Create";
import useStyles from "./styles";
import EditProfile from "../edit-profile";
import editProfileActions from "../../actions/edit-profile.actions";

import { makeStyles } from "@material-ui/core/styles";

const useStyles = makeStyles((theme) => ({
    avatar: {
        width: "100px",
        height: "100px",
        margin: "16px auto",
    },
    detailsContainer: {
        padding: theme.spacing(1, 2),
    },
    detailsTextContainer: {
        margin: theme.spacing(3, 0),
    },
    detailsText: {
        display: "flex",
        marginBottom: theme.spacing(1),
        wordWrap: "break-word",
    },
    detailsIcon: {
        marginRight: theme.spacing(2),
    },
```

```
    profileHeader: {
        display: "flex",
        justifyContent: "space-between",
    },
    warningText: {
        color: "red",
    },
}));

export default useStyles;

const ProfileDetails = ({
    currentUser,
    openEditor,
}) => {
    const classes = useStyles();
    let imagePath;
    if (currentUser.avatar) {
        imagePath = currentUser.avatar.replace("\\", "/");
    }

    console.log(`${window.origin}/${imagePath}`);

    return (
        <>
            <Paper className={classes.detailsContainer}>
                <div className={classes.profileHeader}>
                    <Typography variant="h6">
                        Профиль
                    </Typography>
                    <Button onClick={openEditor}>
                        <CreateIcon />
                    </Button>
                </div>
                <Avatar
                    alt={currentUser.name}
                    src={`${window.origin}/${imagePath}`}
```

```
                    className={classes.avatar}
                  />
              <div className={classes.detailsTextContainer}>
               <div className={classes.detailsText}>
                <AccountCircleIcon className={classes.detailsIcon} />
                          <Typography variant="body1">
                            {currentUser.name}
                          </Typography>
                        </div>
                        <div className={classes.detailsText}>
                          <SchoolIcon className={classes.detailsIcon} />
                            <Typography variant="body1">
                              {
                               currentUser.grade
                               ? `Учень ${currentUser.grade.name} класу`
                               : (
                                <span className={classes.warningText}>
                                     Вкажіть свій клас
                                </span>
                                          )
                                   }
                              </Typography>
                       </div>
                       <div className={classes.detailsText}>
                        <SchoolIcon className={classes.detailsIcon} />
                             <Typography variant="body1">
                               {
                                     currentUser.grade
                                     ? `Класний керівник: ${currentUse
r.grade.classroomTeacher.name}`
                                     : (
                                  <span className={classes.warningText}>
                                                Вкажіть свій клас
                                           </span>
                                     )
                                  }
                             </Typography>
```

```
                        </div>
                        <div className={classes.detailsText}>
          <SchoolIcon className={classes.detailsIcon} />
                <Typography variant="body1">
                    {currentUser.grade
                     ? `Email класного керівника: ${currentUser.grade.cl
assroomTeacher.email}`
                     : (
                         <span className={classes.warningText}>
                             Вкажіть свій клас
                          </span>}
                            </Typography>
                        </div>
                        <div className={classes.detailsText}>
                         <MailIcon className={classes.detailsIcon} />
                            <Typography variant="body1">
                                {currentUser.email}
                            </Typography>
                        </div>
                    </div>
                </Paper>
                <EditProfile />
            </>
        );
    };
    const mapStateToProps = ({
        authReducer: {currentUser},
    }) => ({
        currentUser,
    })
    const mapDispatchToProps = {
        openEditor: () => editProfileActions.openProfileEditor(),
    };

    export default connect(mapStateToProps, mapDispatchToProps)(ProfileD
etails);
```

Додаток F

Управління статтями

```
import { authActions } from "./index";

const getAllArticlesRequest = () => ({
    type: "GET_ARTICLES_REQUEST",
});

const getAllArticlesSuccess = (articles) => ({
    type: "GET_ARTICLES_SUCCESS",
    payload: {
        articles,
    },
});

const getAllArticlesError = (message) => ({
    type: "GET_ARTICLES_ERROR",
    payload: {
        message,
    },
});

const getAllArticles = (
    schoolService,
    skip,
    limit,
    term,
) => () => async (dispatch) => {
    dispatch(getAllArticlesRequest());
    try {
        const responseData = await schoolService.getAllArticles(skip
, limit, term);
        dispatch(getAllArticlesSuccess(responseData));
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(getAllArticlesError(e.message));
    }
```

```
};

const getArticleByIdSuccess = (article) => ({
    type: "GET_ARTICLE_BY_ID_SUCCESS",
    payload: {
        article,
    },
});

const getArticleByIdError = (message) => ({
    type: "GET_ARTICLE_BY_ID_ERROR",
    payload: {
        message,
    },
});

const getArticleById = (
    schoolService,
    id,
) => () => async (dispatch) => {
    try {
        const responseData = await schoolService.getArticleById(id);
        dispatch(getArticleByIdSuccess(responseData));
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(getArticleByIdError(e.message));
    }
};

const updateOrCreateArticleSuccess = () => ({
    type: "UPDATE_OR_CREATE_ARTICLE_SUCCESS",
});

const updateOrCreateArticleError = (updateOrCreateMessage) => ({
    type: "UPDATE_OR_CREATE_ARTICLE_ERROR",
    payload: {
        updateOrCreateMessage,
```

```
    },
});

const clearUpdateAndCreateError = () => ({
    type: "CLEAR_UPDATE_AND_CREATE_ERROR",
});

const createArticle = (
    schoolService,
    data,
) => () => async (dispatch) => {
    try {
        await schoolService.createArticle(data);
        dispatch(updateOrCreateArticleSuccess());
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(updateOrCreateArticleError(e.message));
    }
};

const updateArticle = (
    schoolService,
    id,
    data,
) => () => async (dispatch) => {
    try {
        await schoolService.updateArticle(id, data);
        dispatch(updateOrCreateArticleSuccess());
    } catch (e) {
        if (e.status === 401) authActions.logout()(dispatch);
        dispatch(updateOrCreateArticleError(e.message));
    }
};

const deleteArticleError = (message) => ({
    type: "DELETE_MESSAGE_ERROR",
    payload: {
```

```
            message,
        },
    });

    const deleteArticle = (
        schoolService,
        id,
    ) => () => async (dispatch) => {
        try {
            await schoolService.deleteArticle(id);
            const responseData = await schoolService.getAllArticles(0, 0
, "");
            dispatch(getAllArticlesSuccess(responseData));
        } catch (e) {
            if (e.status === 401) authActions.logout()(dispatch);
            deleteArticleError(e.message);
        }
    };

    const setArticlesTerm = (term) => ({
        type: "SET_ARTICLES_TERM",
        payload: {
            term,
        },
    });

    const setArticleEditing = (editing, id) => ({
        type: "SET_ARTICLE_EDITING",
        payload: {
            editing,
            id,
        },
    });

    const setUpdate = () => ({
        type: "SET_UPDATE",
    });
```

```
const setCreate = () => ({
    type: "SET_CREATE",
});

const clearCurrentArticle = () => ({
    type: "CLEAR_CURRENT_ARTICLE",
});

const articlesActions = {
    getAllArticles,
    createArticle,
    setArticlesTerm,
    deleteArticle,
    setArticleEditing,
    getArticleById,
    updateArticle,
    setUpdate,
    setCreate,
    clearCurrentArticle,
    clearUpdateAndCreateError,
};

export default articlesActions;

import React, { useEffect } from "react";
import { connect } from "react-redux";
import {
    List,
    ListItem,
    Divider,
    Button,
    Paper,
} from "@material-ui/core";
import DeleteForeverIcon from "@material-ui/icons/DeleteForever";
import CreateIcon from "@material-ui/icons/Create";
import { bindActionCreators } from "redux";
```

```
import withSchoolService from "../hoc/with-school-service";
import articlesActions from "../../actions/articles.actions";
import ArticlesSearchPanel from "../articles-search-panel";
import useStyles from "./styles";
import alertActions from "../../actions/alert.actions";

const ArticlesList = ({
    articles,
    getAllArticles,
    deleteArticle,
    setArticleEditing,
    setUpdate,
    openAlert,
}) => {
    const classes = useStyles();

    useEffect(() => {
        getAllArticles(0);
    }, [getAllArticles]);

    return (
        <>
            <ArticlesSearchPanel />
            <Paper>
                <List className={classes.root}>
                    {articles.map(({
                        _id, title, date,
                    }, index) => (
                        <React.Fragment key={_id}>
                            <ListItem alignItems="flex-start">
                             <div className={classes.contentWrapper}
>
                             <div className={classes.titleAndButtons
}>
                                    <div className="primaryText">
<div>
    {title}
```

```
</div>
<div className={classes.date}>
    <span className={classes.dateText}>
          {`Создано:  `}
    </span>
    {date}
</div>
</div>
     <Button
        onClick={() => {
        openAlert(
        "Confirm",
        "Do you want to delete this article?",
        () => deleteArticle(_id),
        );
        }}
        className={classes.button}
        size="small"
     >
     <DeleteForeverIcon />
     </Button>
     <Button
          onClick={() => {
                setArticleEditing(true, _id);
                setUpdate();
          }}
          className={classes.button}
          size="small"
     >
     <CreateIcon />
         </Button>
            </div>
           </div>
      </ListItem>
           {(index !== articles.length -
 1) && <Divider component="li" />}
                            </React.Fragment>
```

```
                        ))}
                    </List>
                </Paper>
            </>
        );
    };

    const mapStateToProps = ({
        articlesReducer: { articles, editing },
    }) => ({
        articles,
        editing,
    });

    const mapDispatchToProps = (dispatch, { schoolService }) => bindActi
onCreators({
        getAllArticles: (
            skip, limit,
        ) => articlesActions.getAllArticles(schoolService, skip, limit)(
),
        deleteArticle: (id) => articlesActions.deleteArticle(schoolServi
ce, id)(),
        setArticleEditing: (
            editing,
            id,
        ) => articlesActions.setArticleEditing(editing, id),
        setUpdate: () => articlesActions.setUpdate(),
        openAlert: (
            title,
            content,
            cb,
        ) => alertActions.openAlert(title, content, cb),
    }, dispatch);


    import React, { useEffect } from "react";
    import { connect } from "react-redux";
```

```
    import Button from "@material-ui/core/Button";
    import { bindActionCreators } from "redux";
    import CloseIcon from "@material-ui/icons/Close";
    import TextEditor from "../text-editor";
    import ArticleTitleInput from "../article-title-input";
    import ArticlePictureInsert from "../article-picture-insert";
    import articleCreatorActions from "../../actions/article-
creator.actions";
    import articlesActions from "../../actions/articles.actions";
    import withSchoolService from "../hoc/with-school-service";
    import Spinner from "../spinner/spinner";
    import useStyles from "./styles";
    import alertActions from "../../actions/alert.actions";

    const ArticleCreator = ({
        title,
        setTitle,
        image,
        setImage,
        content,
        setContent,
        createArticle,
        updateArticle,
        getArticleById,
        editingArticleId,
        currentArticle,
        clearCurrentArticle,
        setArticleEditing,
        create,
        update,
        openAlert,
        updateOrCreateError,
        clearUpdateAndCreateError,
        updateOrCreateMessage,
    }) => {
        const classes = useStyles();
        const onClose = () => {
```

```
            setArticleEditing(false);
            setTitle("");
            setImage(null);
            setContent("");
            clearCurrentArticle();
            clearUpdateAndCreateError();
        };

        useEffect(() => {
            if (editingArticleId && currentArticle === null) {
                getArticleById(editingArticleId);
            }
            if (editingArticleId && currentArticle) {
                setTitle(currentArticle.title);
                setImage(currentArticle.image);
                setContent(currentArticle.content);
            }
        }, [currentArticle, editingArticleId, setTitle, setImage, setCon
tent, getArticleById]);

        useEffect(() => {
            if (updateOrCreateError) {
                openAlert(
                    "Error",
                    `${updateOrCreateMessage}`,
                    () => clearUpdateAndCreateError(),
                    true,
                );
            }
            if (updateOrCreateError === false) {
                onClose();
            }
        });

        const onSubmit = (e) => {
            e.preventDefault();
            const data = new FormData();
```

```
    data.append("title", title);
    data.append("content", content);
    if (image) data.append("image", image);
    if (update) updateArticle(currentArticle._id, data);
    if (create) createArticle(data);
};

if (update && !content) return <Spinner />;

return (
    <>
        <div className={classes.closeButton}>
            <Button
                onClick={() => {
                    openAlert(
                        "Confirm closing form",
                        "Do you want to close?",
                        onClose,
                    );
                }}
            >
                <CloseIcon />
            </Button>
        </div>
        <form id="create-article-form" onSubmit={onSubmit}>
            <div className={classes.flexContainer}>
                <ArticleTitleInput
                    title={title}
                    setTitle={setTitle}
                />
                <ArticlePictureInsert
                    setImage={setImage}
                />
            </div>

            <TextEditor
                content={content}
```

```
                    setContent={setContent}
                />

                <Button
                    type="submit"
                    color="primary"
                    variant="contained"
                    className={classes.submit}
                >
                    {update ? "Update" : "Create"}
                </Button>
            </form>
        </>
    );
};

const mapStateToProps = ({
    articleCreatorReducer: { title, image, content },
    articlesReducer: {
        currentArticle,
        editingArticleId,
        editing,
        create,
        update,
        updateOrCreateError,
        updateOrCreateMessage,
    },
}) => ({
    title,
    image,
    content,
    currentArticle,
    editingArticleId,
    editing,
    create,
    update,
    updateOrCreateError,
```

```
        updateOrCreateMessage,
    });

    const mapDispatchToProps = (dispatch, { schoolService }) => bindActi
onCreators({
        createArticle: (data) => articlesActions.createArticle(schoolSer
vice, data)(),
        updateArticle: (
            id,
            data,
        ) => articlesActions.updateArticle(schoolService, id, data)(),
        setTitle: (title) => articleCreatorActions.setArticleTitle(title
),
        setImage: (image) => articleCreatorActions.setArticleImage(image
),
        setContent: (content) => articleCreatorActions.setArticleContent
(content),
        getArticleById: (id) => articlesActions.getArticleById(schoolSer
vice, id)(),
        setArticleEditing: (editing) => articlesActions.setArticleEditin
g(editing),
        clearCurrentArticle: () => articlesActions.clearCurrentArticle()
,
        openAlert: (
            title,
            content,
            cb,
            confirm,
        ) => alertActions.openAlert(title, content, cb, confirm),
        clearUpdateAndCreateError: () => articlesActions.clearUpdateAndC
reateError(),
    }, dispatch);
    import { makeStyles } from "@material-ui/core/styles";

    const useStyles = makeStyles((theme) => ({
        flexContainer: {
            display: "flex",
```

```
        alignItems: "center",
    },
    submit: {
        marginTop: theme.spacing(2),
        marginLeft: "auto",
        display: "block",
    },
    closeButton: {
        display: "flex",
        justifyContent: "flex-end",
    },
}));
export default useStyles;
```