НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки ”

на тему «Модуль викладача в системі управління дипломним проектуванням кафедри»

Виконав (-ла): студент (-ка) 4 курсу, групи ТМ-52

Лещенко Владислав Юрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ст. викладач Карпенко Є. Ю. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший бакалаврський

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту

Лещенко Владиславу Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Модуль викладача в системі управління дипломним проектуванням кафедри»
керівник роботи Карпенко Євгеній Юрійович старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”22” 05 2019р. № 1325С

2. Строк подання студентом роботи .06.2019
3. Вихідні дані до роботи програмний застосунок розроблено у середовищі PhpStorm 2018 з використанням мови JavaScript для клієнтської та серверної сторони.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дослідити предметну область. Розробити загальну архітектуру системи. Розробити модуль викладача в системі управління дипломним проектуванням кафедри. Надати зручний інтерфейс для користування системою.

5. Перелік ілюстративного матеріалу
«Мета та завдання роботи», «Проблеми які вирішує система для викладача», «Діаграма використання», «Існуючі рішення», «Використані технології», «Загальна архітектура системи», «Архітектура клієнтської сторони», «Діаграма компонентів», «Приклади роботи програми», «Висновки».

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”14” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	*Затвердження теми роботи*	09.10.2019	
2.	Вивчення та аналіз задачі	14.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	02.02.2019-03.03.2019	
4.	Розробка структур окремих підсистем	04.03.2019-14.04.2019	
5.	Програмна реалізація системи	15.04.2019-19.05.2019	
6.	Оформлення пояснювальної записки	20.05.2019-05.06.2019	
7.	*Захист програмного продукту*	14.05.2019	
8.	*Передзахист*	28.05.2019	
9.	Захист	17.06.2019-21.06.2019	

Студент _____ Лещенко В. Ю.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Карпенко Є. Ю.
(підпис) (прізвище та ініціали,)

Анотація

Метою роботи було створення веб-додатку по управлінню дипломним проектуванням кафедри. Користувацький інтерфейс дозволяє користуватися системою з будь-якого електронного пристрою. Користувачі поділяються на три групи: студенти, викладачі та додатковий персонал. Система забезпечує ефективну взаємодію між ними в навчальному процесі.

Записка містить 50 сторінок, 29 рисунків та 6 посилань.

Abstract

The purpose of the work was to create a web application for the management of graduate design department. The user interface allows you to use the system from any electronic device. Users are divided into three groups: students, teachers and additional staff. The system provides an effective interaction between them in the educational process.

The note contains 50 pages, 29 images and 6 links.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- IDE (англ. Integrated Development Environment) – інтегроване середовище розробки
- СКБД – Система керування базами даних
- SPA – Single Page Application
- npm – node package manager
- СКВ – Система контролю версій

ВСТУП

Управління дипломними роботами – складний процес. Це є однією з актуальних проблем навчального процесу в університеті.

Організація і контроль за процесом підготовки й захисту дипломних робіт покладається на викладачів-керівників дипломних робіт. Студентами обираються теми дипломних робіт, вони повинні подати заявки для того щоб закріпитися за дипломною робо, які можуть бути по якійсь причині відхилені. На підставі завдання на виконання дипломної роботи та календарного плану роботи, студентом складається план дипломної роботи, узгоджується з науковим керівником.

На всіх етапах взаємодії між студентами, викладачами та додатковим персоналом під час виконання дипломних робіт можуть виникати труднощі в комунікаціях, інформуванні та звітуванні.

Таким чином, було прийнято рішення розробити систему по управління дипломними роботами для забезпечення конструктивної взаємодії студентів з викладачами та додатковим персоналом.

1. ЗАДАЧА РОЗРОБКИ МОДУЛЮ ВИКЛАДАЧА В СИСТЕМІ УПРАВЛІННЯ ДИПЛОМНИМ ПРОЕКТУВАННЯМ КАФЕДРИ

Дипломна робота є кваліфікаційною роботою випускника. Зміст дипломної роботи показує рівень загальнотеоретичної та професійної підготовки студента. За рівнем її виконання та результатами захисту Державна екзаменаційна комісія визначає можливість присвоєння випускнику відповідної кваліфікації та видачі диплома.

Оскільки управління дипломними роботами є складною задачею - воно потребує максимально ефективної взаємодії між учасниками. Завдяки тому, що ми живемо в епоху цифрових технологій ми маємо можливість вдосконалити цей процес за допомогою веб-додатку: “Системи управління дипломними роботами”.

Головні проблеми які виникають в процесі управління, підготовці та захисті дипломних робіт:

- Складнощі для викладачів та додаткового персоналу в управлінні темами дипломних робіт.
- Надмірне турбування викладачів для отримання даних щодо статусу тем дипломних робіт.
- Студенти не завжди володіють актуальними даними необхідними для прийняття рішення у виборі дипломної роботи.
- Студенти змушені тратити час для отримання інформації щодо: статусу тем дипломних робіт, зайнятості викладачів, термінів виконання, правил захисту.
- Проблема знаходження контактної інформації.

Як результат дослідження, потрібно створити програмний застосунок, основними функціями якого є:

- Забезпечення користувача персональним кабінетом.
- Розділення користувачів системи на: студентів, викладачів та додаткового персоналу.

- Зручний графічний інтерфейс, який може працювати як на комп’ютері, так і на мобільному телефоні.
- Можливості для викладачів:
    - Створення, редагування та видалення тем для дипломних робіт.
    - Перегляд інформації щодо термінів виконання дипломних робіт.
    - Перегляд інформації про навантаження.
    - Перегляд заявок на теми дипломних робіт від студентів.
    - Закріплення студента за дипломною роботою.
    - Можливість створювати, видаляти та редагувати проекти для дипломних робіт.
    - Контроль за кожним етапом виконання дипломної роботи. Можливість залишати нотатки.
    - Перегляд інформації щодо лабораторій (напрямки, склад, загальна інформація).

2. АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ МОДУЛЮ ВИКЛАДАЧА В СИСТЕМІ УПРАВЛІННЯ ДИПЛОМНИМ ПРОЕКТУВАННЯМ КАФЕДРИ

Технології набувають все більшого розвитку. За останнє десятиліття побудовано багато різноманітного програмного забезпечення, яке виконує дуже велику кількість завдань. Серед них можливо знайти і задачу управління проектами. Проте їх використання може бути незручним враховуючи специфіку наших потреб.

2.1 Аналіз існуючих програмних засобів

У процесі пошуку інформації, та аналізу існуючих рішень, було виявлено, що на даний момент такі системи реалізують поставлені нами задачі не в повному обсязі:

1. “Atlassian Jira” — це засіб для управління проектами та завданнями. JIRA може застосовуватися у всіх випадках, коли необхідно організувати роботу співробітників, ефективно призначати їм завдання, мати миттєві засоби контролю виконання.
2. “Redmine” — відкритий серверний веб-додаток для управління проектами та завданнями. Redmine написаний на Ruby і являє собою додаток на основі широко відомого веб-фреймворку Ruby on Rails. Поширюється згідно GNU General Public.
3. “Wrike” — це система управління проектами, в тому числі для створення завдань, збору команди, підключення до вже знайомим сервісів спільної роботи, інтеграції пошти. Сервіс забезпечує онлайн-середовище для робочого взаємодії як в локальній, так і розподіленої команді, дозволяє планувати проекти, пріоритезувати завдання і відстежувати графік їх виконання. Крім веб-сервісу, Wrike також доступний у вигляді додатків для iOS і Android.

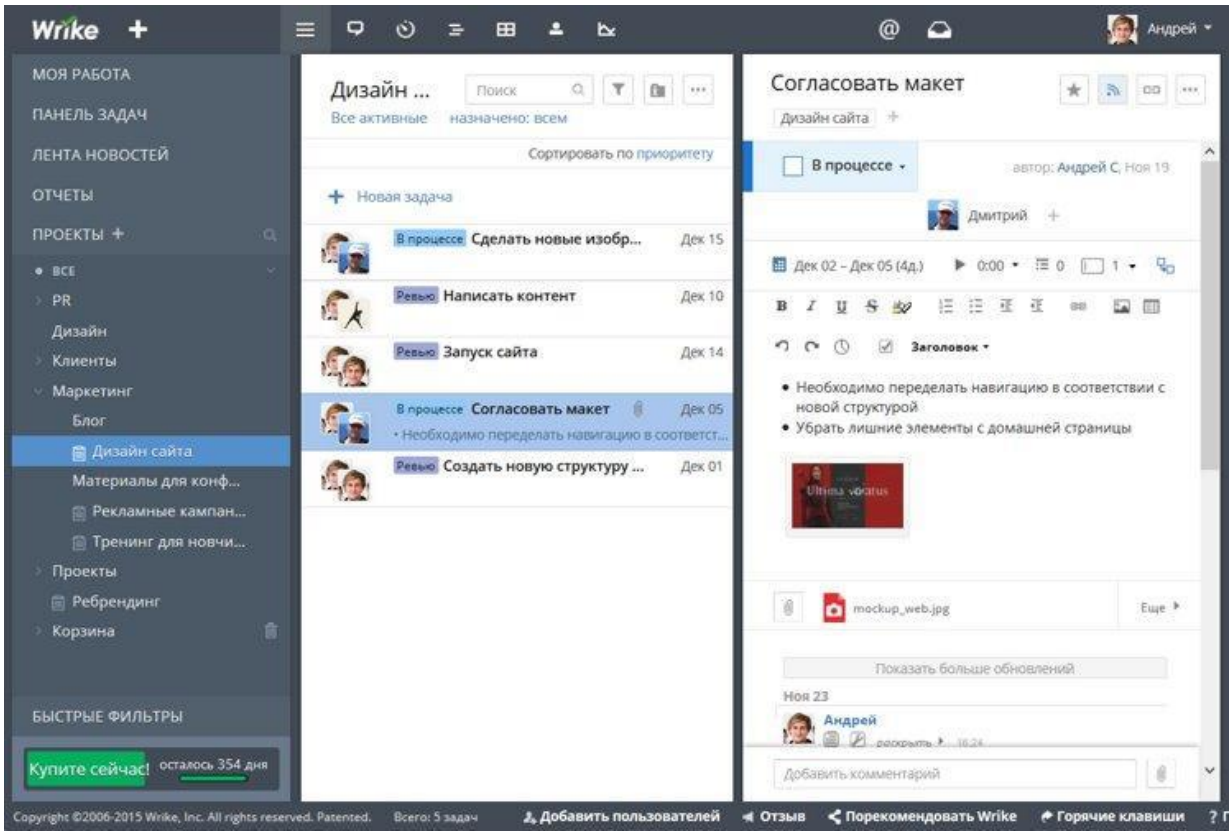

Рисунок 2.1 — веб-додаток “Wrike”

4. “Kaiten” — це веб-сервіс для візуального управління виробничими процесами. Розробникам вдалося поєднати конструктор процесів і управління завданнями в одному інструменті. Інструмент дозволяє залучати до спільної роботи бізнес і IT, з легкістю взаємодіяти між відділами і зі сторонніми підрядниками. Kaiten заснований на гнучких підходах і повністю підтримує Scrum і Kanban. Унікальна функціональність продукту — зведені дошки, вони дозволяють візуалізувати всі процеси компанії, починаючи з стратегічних цілей до завдань конкретних виконавців.

Висновки до розділу

В ході дослідження було розглянуто чотири програмних продукти, які найбільш підходять нашим вимогам:

1. Wrike
2. Atlassian Jira

3. Redmine
4. Kaiten

Були розглянуті основні можливості цих програмних продуктів та їх застосування.

Недоліком цих систем є їх громіздкість. Для початківців ці програми можуть здатися переповненими функціоналом та дуже специфічними в плані налаштування. Також у них є певні обмеження, наприклад в кількості користувачів, для доступних тарифів. Тому користування цими системами буде економічно не вигідно.

Таким чином, виходячи з усього вищезазначеного можна зробити висновок, що найкращим варіантом який зможе працювати на будь-яких сучасних пристроях, є розробка власної програмного продукту, який заснований на сучасних веб-технологіях.

3. ЗАСОБИ РОЗРОБКИ

Аналізуючи поставлену задачу та методи її вирішення, було вирішено розроблювати програмний продукт на основі веб-технологій. Головною перевагою веб-застосунку перед іншими варіантами є його універсальність і можливість використання на будь-яких пристроях в яких є браузер.

Програмний продукт було написано мовою JavaScript (як серверної, так і клієнтської частини) з використанням фреймворків та бібліотек. Все дані зберігаються в базі даних за допомогою СУБД MySQL.

3.1 Середовище розробки PhpStorm

Середовище розробки PhpStorm - це інноваційне інтегроване середовище розробки (IDE), розроблене JetBrains. Забезпечує запобігання помилок, краще автозаповнення та рефакторинг коду, налагодження з нульовою конфігурацією і розширений редактор HTML, CSS і JavaScript. IDE забезпечує інтелектуальне завершення коду, підсвічування синтаксису, розширену конфігурацію форматування коду, перевірку на льоту, згортання коду, підтримку мовних сумішей та багато іншого. Він надає засоби для роботи з базами даних та SQL у проекті. Має інтеграцію з системами управління версіями. Користувачі можуть з легкістю розширити функціонал середовища розробки за допомогою встановлення плагінів з онлайн каталогу або власноруч написаних.

Середовище розробки PhpStorm дозволяє ефективно розробляти програми на Node.js і підтримує повноцінне налагодження Node.js додатків. Новий додаток можна створити, використовуючи шаблон Node.js Express, а необхідні модулі легко встановити через менеджер пакетів npm.

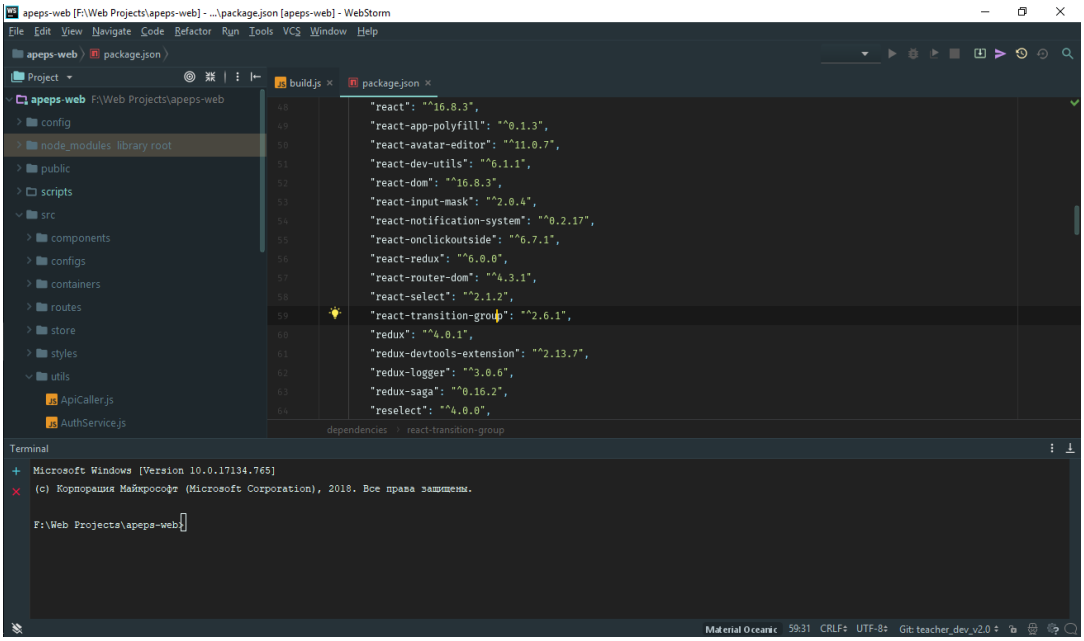

Рисунок 3.1 — Середовище розробки PhpStorm 2019.3.3

3.2 Система керування базами даних MySQL

База даних — це сукупність даних або записів. Система управління базами даних (СУБД) — це програмна система, яка використовує стандартний метод для зберігання та організації даних. Дані можна додавати, оновлювати, видаляти або проходити за допомогою різних стандартних алгоритмів і запитів. СУБД широко використовуються, оскільки вони пропонують ряд корисних функціональних можливостей.

Функції СУБД:

- Визначення моделей даних (таблиці, стовпці, відносини тощо).
- Управління доступом.
- Обробка операцій CRUD (створення, читання, оновлення, видалення).
- Забезпечення узгодженості даних.
- Запит доступу з інтерфейсом (наприклад, SQL).
- Обробка одночасних користувачів.
- Забезпечення та підтримка словника даних.

- Управління транзакціями - переконавшись, що всі операції в транзакції або завершені, або взагалі невиконані.

Існує кілька типів систем управління базами даних. Нижче наведено список семи найпоширеніших типів систем управління базами даних:

- Ієрархічні
- Мережеві
- Реляційні
- Об'єктно-орієнтовані
- Бази даних графіків
- Бази даних моделі ER
- Бази даних документів

У системах управління реляційними базами даних взаємозв'язок між даними є реляційним і дані зберігаються в табличній формі стовпців та рядків. Кожен стовпець в таблиці є атрибутом, а кожен рядок записом. Кожне поле таблиці відображає значення даних.

Мова структурованих запитів (SQL) — це мова, яка використовується для запиту СУБД, наприклад для вставки, оновлення, видалення та пошук записів.

Робота реляційних баз даних на кожній таблиці має ключове поле, яке однозначно вказує на кожен рядок, також ці ключові поля можна використовувати для підключення однієї таблиці даних до іншої.

Реляційні бази даних є найбільш популярними та широко використовуваними базами даних. Деякі з популярних СУБД: SQL Server, MySQL, SQLite від IBM DB2. Як і будь-які інші бази даних, реляційні бази даних мають як переваги, так і слабкі місця. Можна легко програмувати таблиці для взаємодії один з одним, а потім використовувати запит для пошуку баз даних для потрібної інформації, без необхідності виконувати таблицю за таблицею запитів. Інформація може передаватися користувачам та іншим інструментам третьої сторони без особливих зусиль. Однак мови реляційних баз даних не настільки широко використовуються як об'єктно-орієнтована мова програмування, як Java, що робить прогалини у навичках великою проблемою. Розробники повинні мати можливість відтворити логіку і

виявити будь-які невідповідності між таблицями, що вимагає дуже специфічного набору навичок у порівнянні з традиційними базами даних.

3.3 Система контролю версій

Система контролю версій Git — це тип системи керування версіями (VCS), що полегшує відстеження змін у файлах. Наприклад, коли ви редагуєте файл, Git може допомогти вам визначити, що саме змінилося, хто його змінив і чому. Це корисно для координації роботи між кількома людьми в проекті, а також для відстеження прогресу з часом, зберігаючи "контрольні точки". Ви можете використовувати його також для відстеження змін в ілюстраціях. Git не єдина система контролю версій, але вона є найпопулярнішою. Багато розробників програмного забезпечення використовують Git щодня.

Система контролю версій Git призначена для відстеження змін у комп'ютерних файлах і координації роботи над цими файлами серед декількох людей. Тому Git не обов'язково покладається на центральний сервер для зберігання всіх версій файлів проекту. Замість цього кожен користувач "клонує" копію репозиторію (колекції файлів) і має повну історію проекту на своєму жорсткому диску. Цей клон має всі метадані оригіналу, в той час як оригінал сам зберігається на сервері, що розміщується самостійно, або на сервері хостингу третьої сторони, наприклад на GitHub, GitLab або BitBucket .

Особливості Git:

- Безкоштовний і відкритий код. Git випускається під ліцензією GPL (General Public License) з відкритим кодом. Вам не потрібно купувати Git. Це абсолютно безкоштовно. А оскільки це відкритий код, ви можете змінити вихідний код відповідно до ваших вимог.
- Швидкість. Оскільки вам не потрібно підключатися до будь-якої мережі для виконання всіх операцій, вона виконує всі завдання дуже швидко. Тести продуктивності, проведені Mozilla, показали, що Git на порядок швидший, ніж інші системи контролю версій. Вибірка історії версій з локально збереженого сховища може бути в сто разів швидше, ніж вилучення її з віддаленого сервера.

Основна частина Git написана на мові C, що дозволяє уникнути накладних витрат на виконання, пов'язаних з іншими мовами високого рівня.

- Масштабованість. Система контролю версій Git - дуже масштабована. Отже, якщо в майбутньому кількість співробітників збільшиться, то Git може легко впоратися з цією зміною. Хоча Git являє собою ціле сховище, дані, що зберігаються на стороні клієнта, дуже малі, оскільки Git стискає всі величезні дані за допомогою техніки стиснення без втрат.
- Надійність. Оскільки кожен учасник має власне локальне сховище то будь-які дані можуть бути відновлені з будь-яких локальних сховищ. Ви завжди будете мати резервну копію всіх ваших файлів.
- Безпечність. Git використовує SHA1 (Secure Hash Function) для назви та ідентифікації об'єктів у своєму сховищі. Кожен файл і фіксація перевіряються підсумовуються і витягуються його контрольною сумою під час перевірки. Історія Git зберігається таким чином, що ідентифікатор конкретної версії (термін фіксації в термінах Git) залежить від повної історії розробки, що передує цьому фіксації. Після його опублікування неможливо змінити старі версії без того, щоб це було помічено.

Деякі компанії, які використовують Git для керування версіями: Facebook, Yahoo, Zynga, Quora, Twitter, eBay, Salesforce, Microsoft і багато інших.

В системі управління дипломним проектування кафедри в якості веб-сервісу хостингу репозиторіїв я використовував Bitbucket. Він є масштабованим рішеннями для управління сховищами Git, призначеним для команд розробників програмного забезпечення.

Середовище розробки PhpStorm пропонує зручний інтерфейс для роботи з системами контролю версій, що значно полегшує процес розробки програмного забезпечення в команді.

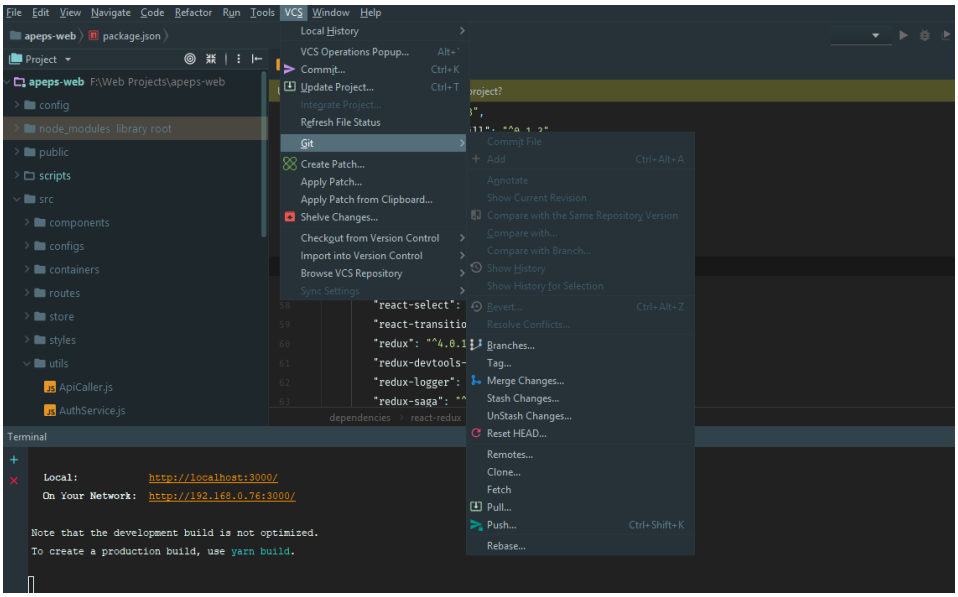

Рисунок 3.2 – Візуальний інтерфейс для роботи з системами контролю версій

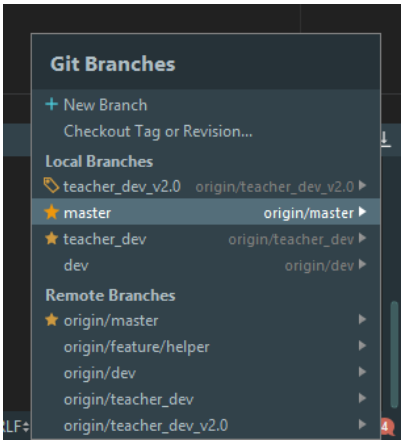

Рисунок 3.3 – Візуальний інтерфейс для роботи з гілками в системі контролю версій Git

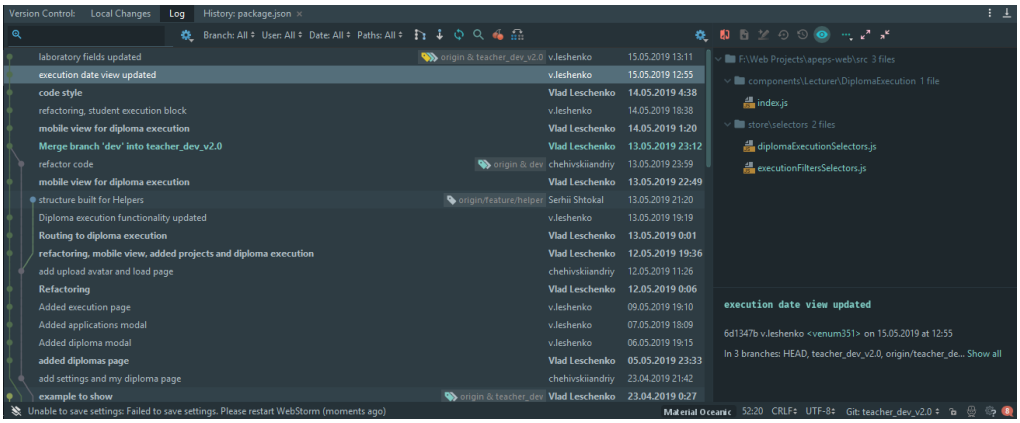

Рисунок 3.4 – Інтерфейс перегляду історії змін з використанням системи контролю версій Git

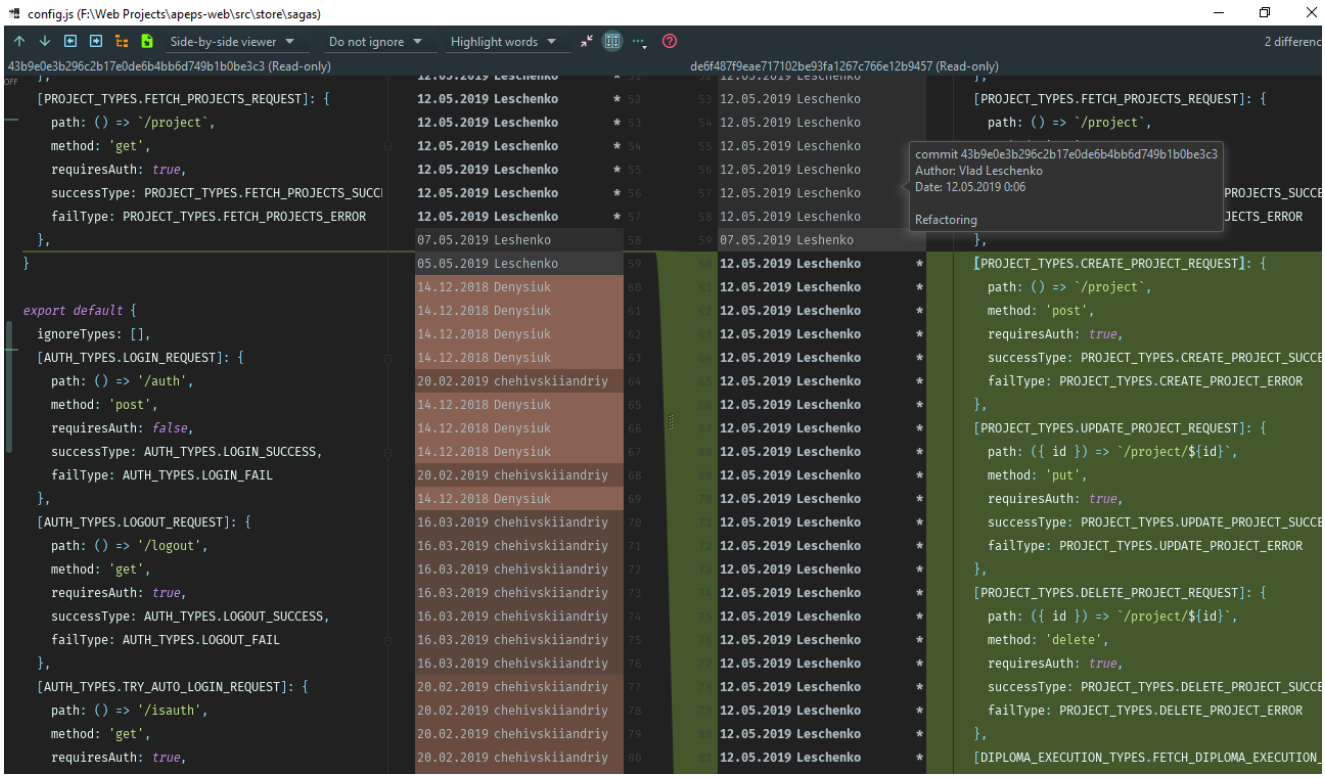

Рисунок 3.5 – Інтерфейс перегляду змін коду з використанням системи контролю версій Git

Приклади роботи з системою Git в середовищі розробки PhpStorm наведені на рисунках 3.2, 3.3, 3.4 та 3.5. Зокрема на рисунку 3.5 показано інтерфейс перегляду зміни коду (автор, дата та час). З цього ж інтерфейсу натиснувши правою кнопкою мишки можна відновити код до змін.

3.4 Розробка користувацького інтерфейсу

В основі клієнтської сторони лежить мова JavaScript, вона є обєктно–орієнтованою мовою скриптів, яка використовується для розробки вбудованих додатків, які виконуються як на стороні клієнта, так і на стороні сервера. Частіше за все вони виконуються в формі JavaScript в клієнтській частині і реалізуються в додатку у вигляді інтегруємого веб–браузером компонента, що дозволяє розробляти покращені інтерфейси і динамічні веб–сайти. JavaScript використовує можливості середовища, в якому виконуються написані на ньому сценарії. JavaScript являється діалектом стандарту ECMAScript і охарактеризований як динамічна мова скриптів.

Для реалізації поставленої задачі було вирішено використовувати наступні бібліотеки:

- Бібліотека React
- Бібліотека Redux
- Бібліотека Styled Components
- Бібліотека Moment
- Пакет модулів Webpack
- Бібліотека Reselect

3.5 Бібліотека React

Бібліотека React (іноді React.js або ReactJS) — JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів. React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій. React може використовуватися для розробки односторінкових і мобільних додатків. Його мета - надати високу швидкість, простоту і масштабованість.

Особливості React:

- Односпрямована передача даних. Властивості передаються від батьківських компонентів до дочірнім. Компоненти отримують властивості як безліч незмінних (англ. Immutable) значень, тому компонент не може безпосередньо змінювати властивості, але може викликати зміни через callback функції. Такий механізм називають “властивості вниз, події наверх”.
- Віртуальний DOM. React використовує віртуальний DOM (англ. Virtual DOM). React створює кеш структуру в пам'яті, що дозволяє обчислювати різницю між попереднім і поточним станами інтерфейсу для оптимального поновлення DOM браузера. Таким чином програміст може працювати зі сторінкою, вважаючи, що вона оновлюється вся, але бібліотека самостійно вирішує, які компоненти сторінки необхідно оновити.

- JavaScript XML (JSX) — це розширення синтаксису JavaScript, яке дозволяє використовувати схожий на HTML синтаксис для опису структури інтерфейсу. Як правило, компоненти написані з використанням JSX, але також є можливість використання звичайного JavaScript. JSX нагадує іншу мову, створений в компанії Фейсбук для розширення PHP, XHP.

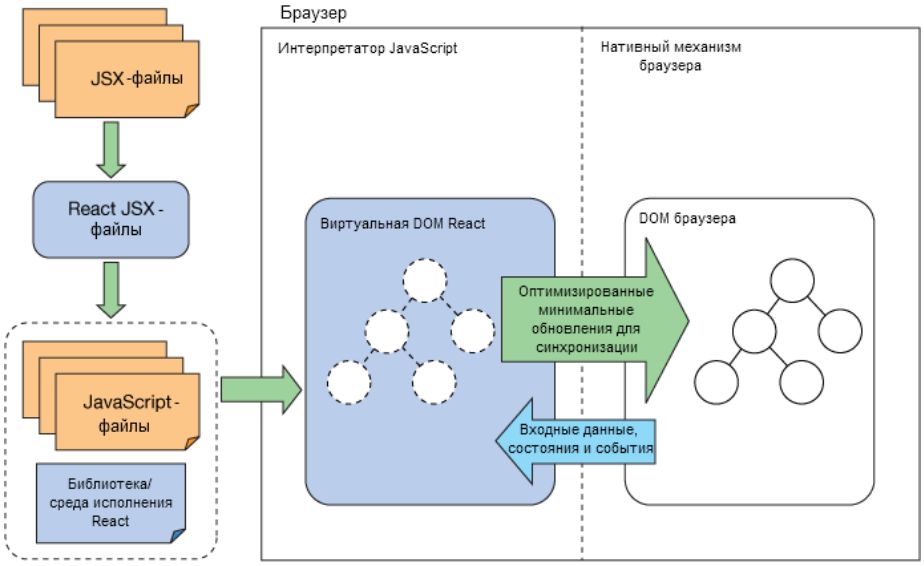

Рисунок 3.6 – Використання бібліотеки React

Бібліотека React — одна з найпопулярніших бібліотек JavaScript для створення веб-інтерфейсів користувача (рисунок 3.6). Його успіх зумовлений кількома факторами, але, можливо, одним з них є чистий і ефективний підхід до програмування.

У середовищі React кожна частина інтерфейсу користувача є компонентом. Компоненти можуть бути складені разом для створення інших компонентів. Сам додаток є компонентом: набором компонентів.

Розробник, вивчаючи React, природно, починає думати про це з точки зору об'єктно-орієнтованого програмування. Сам синтаксис для визначення компонента сприяє цій ідеї:

```
class HelloReact extends Component {
   render() {
        return (<div>Hello React!</div>);
   }
}
```

Однак під об'єктно-орієнтованою обгорткою React приховує функціональну природу. Перевагами дотримання функціонального програмування при розробці React компонентів є:

- Невеликі модулі можна швидко і легко кодувати.
- Модулі загального призначення можуть бути багаторазовими, що призводить до більш швидкої розробки наступних програм.
- Модулі програми можуть бути перевірені самостійно, допомагаючи скоротити час, витрачений на налагодження.
- Функціональні програми є референційно прозорими, тобто якщо змінній присвоюється певне значення в програмі, то це значення знову не може бути змінено.
- Проблема, яку намагається вирішити ваш код, вирішується меншим кодом, але не жертвуючи його функціоналом.
- Програма виглядає більш абстрактно, майже символічно, як математичне рівняння.
- Автоматичне зменшення розповсюджених помилок при розробці (завдяки тому що побічні ефекти зводяться до мінімуму).

Головні принципи які потрібно розуміти для в функціональному програмуванні:

- Чисті функції замість спільного стану та побічних ефектів
- Незмінність над мінливими даними
- Композиція функцій
- Багато загальних, багаторазових утиліт, які використовують функції вищого порядку для роботи з багатьма типами даних замість методів, які працюють тільки для конкретних даних
- Декларативний, а не імперативний код (що робити, а не як це робити)

- Контейнери та функції вищого порядку замість спеціального поліморфізму

3.6 Бібліотека Redux

Бібліотека Redux — це менеджер станів. Найчастіше його використовують з React, але його можливості не обмежуються однією цією бібліотекою. Хоча в React є власний метод управління станами, він погано маштабується. Переміщення стану вгору по дереву працює для простих додатків, але в більш складних архітектур зміна стану проводиться через властивості (props). Ще краще робити це через зовнішнє глобальне сховище. Redux — це спосіб управління станом додатку. Він заснований на декількох концепціях, вивчивши які, можна з легкістю вирішувати проблеми зі станом.

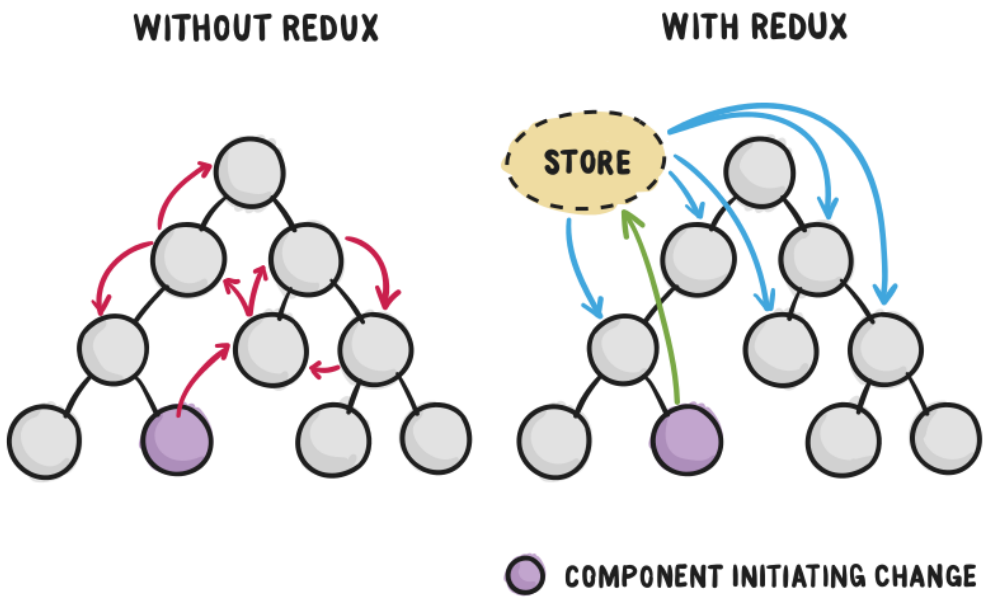

Рисунок 3.7 – Використання бібліотеки Redux

З Redux, очевидно, що всі компоненти отримують свої стани зі сховища (рисунок 3.7). Також очевидно, що всі компоненти відправляють зміни свого стану в сховище. Компонент при запуску змін думає тільки про відправку змін в сховище і не турбується про всі компоненти, яким потрібні ці зміни. Так Redux робить потік даних простіше і зрозуміліше.

Основна концепція у використанні сховища для координації стану додатка це шаблон відомий як Flux. Це шаблон проектування, який відмінно поєднується з архітектурою односпрямованого потоку даних як в React.

Ключові поняття Redux:

- Сховище (store) – зберігає стан додатка.
- Дії (actions) – деякий набір інформації, який виходить від додатка до сховища і який вказує, що саме потрібно зробити. Для передачі цієї інформації у сховища викликається метод dispatch().
- Творці дій (action creators) – функції, які створюють дії.
- Reducer – функція (або кілька функцій), яка отримує дію і відповідно до цієї дії змінює стан сховища.

Загальну схему взаємодії елементів архітектури Redux можна виразити наступним чином (рисунок 3.8):

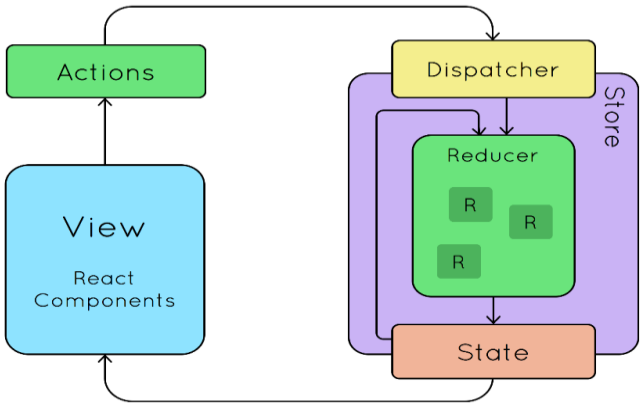

Рисунок 3.8 – Використання бібліотеки Redux

З View (тобто з компонентів React) ми посилаємо дію, цю дію отримує функція reducer, яка відповідно до дії оновлює стан сховища. Потім компоненти React застосовують оновлений стан зі сховища.

Таким чином, програми написані з використанням бібліотект Redux ведуть себе передбачувано, можуть знаходитись в різних середовищах (клієнт, сервер) та легко перевіряються.

3.7 Бібліотека Styled Components

За допомогою бібліотеки Styled Components ми можемо писати простий CSS в нашому JavaScript файлі. Це означає те, що ви можете використовувати весь функціонал CSS, навіть такий як медіа запити, псевдо-селектори, вкладення в JavaScript. Styled-components використовує шаблонні рядки для стилізації компонентів. Це означає те, що коли ви визначаєте ваші стилі, ви насправді створюєте звичайний React компонент, який має ваші стилі, які вже прикріплені до нього. Використовуючи стилізовані компоненти, ми можемо створити легко перевикористовуваний компоненти зі стилями.

Особливо цікава концепція – використання CSS в JS для абстрактного CSS до самого компонентного рівня, використовуючи JavaScript для опису стилів в декларативному і підтримуваному способі. Макс Стойнбер розробив бібліотеку styled-components, яка зробила цю концепцію найбільш поширеною, ніж будь-коли.

Отже, яка різниця між стилями CSS-in-JS та вбудованими стилями? Ось просте пояснення, яке робить все зрозумілішим на найнижчому рівні: CSS-in-JS додає тег <style> поверх DOM, тоді як вбудовані стилі просто додають властивості до вузла DOM.

У React, CSS-in-JS дозволяє мислити і розробляти стилі абстрактно на рівні компонентів, використовуючи принципи модульності і ізоляції, модульного тестування, принципу DRY і так далі.

Проте, офіційні документація React стверджує: “React не має думки про те, як визначаються стилі; Якщо ви сумніваєтеся, розпочніть з визначенні стилів у окремому файлі * .css як зазвичай та посилайтеся на них за допомогою className."

Оскільки компоненти React стають будівельними елементами інтерфейсу нашої програми, їх формування стає важливою частиною нашого робочого процесу розробки.

При масштабуванні нашої кодової бази наш метод стилізації є ключем до балансування модульності і уникнення загальних проблем масштабування, які можуть поставити під загрозу нашу швидкість розвитку.

3.8 Збирач модулів Webpack

Інструмент Webpack — це збирач модулів для сучасних JavaScript-додатків. Коли Webpack обробляє додаток, він рекурсивно створює граф залежностей, що включає всі модулі, необхідні додатком, потім упаковує всі ці модулі в задане число файлів — частіше за все в один — для запуску вашого застосування.

Збирач модулів Webpack також здатний виконувати безліч інших операцій:

- збирати воєдино ваші ресурси;
- стежити за змінами для повторного виконування завдання (живий перегляд змін)
- може виконати транспіляцію JavaScript новітнього покоління до старішого стандарту JavaScript (ES5) за допомогою Babel, що дозволить використовувати новітні функції JavaScript, не турбуючись про те, чи підтримує їх браузер чи ні
- може виконати транспіляцію CoffeeScript в JavaScript
- може конвертувати вбудовані зображення в data: URI
- може запустити webpack-dev-server (в ньому вбудований локальний серве)
- може працювати з Hot Module Replacement (заміна гарячого модуля)
- може розділити вихідний файл (output file) на кілька файлів, щоб уникнути повільного завантаження сторінки через великого розміру JS-файлу
- може виконати Tree Shaking

Webpack не обмежується одним лиш фронтендом, його також успішно застосовують в бекенд розробці на Node.js.

У Webpack є попередники, у яких він перейняв багато ідей. Основна відмінність полягає в тому, що ті інструменти відомі як task runners (такс-раннери), в той час як Webpack ніщо інше, як збирач модулів.

Webpack – це більш цілеспрямований інструмент. Вам достатньо вказати точку входу в ваш додаток (це може бути навіть HTML-файл з тегами <script>), а webpack проаналізує файли і об'єднає їх в один вихідний JavaScript-файл, який містить все необхідне для запуску програми.

3.9 Розробка серверної частини

Серверна частина побудована на Node.js з використанням фреймворку Express.js.

Node.js — це серверна платформа для роботи з JavaScript через двигун V8. JavaScript виконує дію на стороні клієнта, а Node.js — на сервері. За допомогою Node.js можна писати повноцінні програми. Node.js вміє працювати з зовнішніми бібліотеками, викликати команди з коду на JavaScript і виконувати роль веб-сервера.

З Node.js простіше масштабуватися. При одночасному підключенні до сервера тисяч користувачів Node.js працює асинхронно, тобто ставить пріоритети і розподіляє ресурси грамотніше. Java ж, наприклад, виділяє на кожне підключення окремий потік.

Обгрунтування вибору Node.js як серверної мови програмування:

- Одна мова програмування – JavaScript. Будемо використовувати його і на клієнті і на сервері.
- Загальний код на клієнті і на сервері. Значить ми можемо використовувати однаковий код, ту ж бібліотеку і на клієнті і на сервері. Втім на практиці це повторне використання вельми обмежена і воно зазвичай стосується бібліотек загального вигляду. Припустимо є об'єкт «User». Цей об'єкт «User» в браузері робить одне, використовує можливості браузера, а на сервері він зазвичай робить інше, в тому числі працює з базами даних, тут вже взагалі інший код. Тобто прямо перевикористованого коду не так багато, але тим не менше він є.
- Вирішує основні завдання для Web-розробки. Node.js призначений для вирішення основних завдань, які перед нами ставить Web-розробка. Хочемо працювати з базою даних, легко — Node.JS відмінно працює з найпоширенішими сучасними базами даних. Хочемо отримати низькорівневий доступ до мережі http, https, tcp, udp, в Node.JS міститься відмінно пророблена колекція модулів. Node.JS Це інструмент який спочатку проектувався для вирішення завдань під Web і в ньому дуже багато зроблено. З іншого боку Node.JS напевно не найкращий якщо ви хочете створити віконний додаток під Windows або порахувати число Пі з точністю до якогось n-го знака. В цьому

випадку напевно будуть більш кращі інші інструменти. Node.JS це засіб яке вирішує основні завдання під Web і робить це добре.

- Багато з'єднань і завдань одночасно. Node.JS найкраще поводиться там, де потрібно підтримувати велику кількість з'єднань. Наприклад це 10 000 клієнтів в чаті одночасно або це онлайн гра, де теж дуже багато користувачів підключені і виконують певні дії. Велика кількість з'єднань, завдань це те з чим Node.JS справляється добре. При цьому ці завдання не повинні бути обчислювальними. Хоча якщо завдання обчислювальні то в Node.JS є кошти для паралелізації, але вони не так добре опрацьовані як на деяких інших платформах.
- Легко зробити робочий прототип. На Node.JS дуже легко створити щось і змусити це працювати, причому працювати в тому числі під навантаженням. Коли ми розробляємо, то сучасні методики розробки ітеративні. Тобто ми робимо щось, воно повинно працювати, потім ми додаємо до цього особливості, воно знову працює, потім ми додаємо ще щось. І на Node.JS перший етап проходить дуже швидко, а якщо все в порядку з архітектурою, то і далі воно відмінно розширюється.
- Зручний менеджер пакетів, де багато всього. У Node.JS є дуже зручний менеджер пакетів, NPM, ми його детально розглянемо і є дуже багато готових пакетів, які можна поставити і працювати з ними.
- Велике і активна спільнота навколо. Ну і нарешті, співтовариство. Є багато, багато людей які люблять Node.JS, які пишуть під Node.JS, які підтримують те, що вони зробили під Node.JS. Переважно це індивідуальні розробники або невеликі компанії плюс співтовариство. Виходячи з цього можемо зробити висновок, що є дуже багато готових пакетів, модулів які потрібно поставити і вони разом нормально працюють. Для додатків на Node.JS, абсолютно нормально якщо вони використовують двадцять, тридцять, п'ятдесят, вісімдесят готових модулів. Така ось матрична структура виходить, коли є багато, багато маленьких модулів які разом інтегруються і відмінно працюють.

- В якості менеджеру пакетів для Node.js використовується NPM (node package manager). В ньому є понад півмільйона пакетів, що робить його найбільшим в світі репозиторієм коду, написаного на одній мові. Це дозволяє говорити про те, що в npm можна знайти пакети, призначені для вирішення практично будь-яких завдань. Спочатку NPM створювався як система управління пакетами для Node.js, але в наші дні він використовується і при розробці фронтенд-проектів на JavaScript. Для взаємодії з реєстром NPM використовується однойменна команда, яка дає розробнику величезна кількість можливостей.

Фреймворк Express — мінімалістичний і гнучкий фреймворк веб-додатків Node.js, який надає набір функцій для розробки веб і мобільних додатків. Він суттєво спрощує розробку веб-додатків на базі Node.js. Нижче наведені деякі з основних функцій фреймворка Express:

- Дозволяє налаштовувати посередників для відповіді на HTTP-запити.
- Визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу HTTP та URL-адреси.
- Дозволяє динамічно відтворювати HTML-сторінки на основі передачі аргументів шаблонам.

3.10 Висновки до розділу

В цьому розділі були розглянуті та обгрунтовані головні засоби розробки.

На клієнтській стороні були використані наступні бібліотеки: React, Redux, Styled Components, Webpack, Moment, Reselect. Були в деталях роглянуті головні з них, переваги та особливості кожної.

На серверній стороні використовувалася мова Node.js, особливості якої повністю задовільняють поставленим вимогам. В якості СУБД було використано MySQL.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмна реалізація будується на основі клієнт-серверної архітектури. Клієнтська частина реалізована як односторінковий додаток з компонентним підходом до розробки, розділенням рівнів відображення та бізнес-логіки.

4.1 Архітектура системи

Програмна реалізація будується на основі клієнт-серверної архітектури. Клієнт-серверна архітектура — це архітектурний стиль розгортання, який описує поділ функціональності на шари, причому кожен сегмент є рівнем, який може бути розташований на фізично окремому комп'ютері. Вони розвивалися за допомогою компонентно-орієнтованого підходу, зазвичай використовуючи специфічні для платформи методи для комунікації замість підходу, основаного на повідомленнях.

Ця архітектура має різні застосування з різними додатками. Він може використовуватися в веб-додатках і розподілених програмах.

Використовуючи цю архітектуру, програмне забезпечення розділене на 3 різних яруси: рівень презентації, логічний рівень і рівень даних. Кожен ярус розробляється і підтримується як незалежний рівень.

1-й рівень презентації. Це найвищий рівень програми. Рівень презентації забезпечує користувальницький інтерфейс програми (UI). Як правило, це передбачає використання графічного інтерфейсу користувача для інтелектуального взаємодії з клієнтами та веб-технологій для взаємодії на основі браузера. Рядок презентації відображає інформацію, пов'язану з такими послугами, як перегляд змісту товарів, купівлі та кошика. Він обмінюється даними з іншими рівнями шляхом виведення результатів у рівень браузера / клієнта та всіх інших рівнів у мережі.

2-логічний рівень (називається також бізнес-логікою, рівнем доступу до даних або середнім рівнем). Логічний рівень витягується з рівня презентації і, як власний шар; він контролює функціональність програми, виконуючи детальну обробку. Логічний рівень — це вирішення критично важливих бізнес-завдань. Компоненти,

які складають цей шар, можуть існувати на серверній машині, щоб допомогти у спільному використанні ресурсів. Ці компоненти можуть використовуватися для забезпечення бізнес правил, таких як бізнес-алгоритми, юридичні чи урядові правила, а також правила даних, які призначені для збереження послідовності структур даних у межах певної або декількох баз даних. Оскільки ці компоненти середнього рівня не прив'язані до конкретного клієнта, вони можуть бути використані всіма програмами і можуть бути переміщені в різні місця. Наприклад, прості редагування можуть бути розміщені на стороні клієнта, щоб мінімізувати мережеві двосторонні подорожі, або правила даних можуть бути розміщені в збережених процедурах.

3-й рівень. Цей рівень складається з серверів баз даних, є фактичним рівнем доступу СУБД. Доступ до нього можна отримати через рівень бізнес-послуг і, іноді, користувацьким рівнем послуг. Тут інформація зберігається і витягується. Цей рівень зберігає дані нейтральними та незалежними від серверів додатків або бізнес-логіки. Надання даних власного рівня також покращує масштабованість і продуктивність. Цей шар складається з компонентів доступу до даних для надання допомоги у спільному використанні ресурсів і для того, щоб дозволити клієнтам налаштовуватися без встановлення бібліотек СУБД і драйверів ODBC на кожному клієнті. Прикладом може служити комп'ютер з системою управління базами даних (СУБД), такий як база даних MySQL Server.

На рисунку 4.1 представлена реалізація клієнт-серверної архітектури розробленої системи.

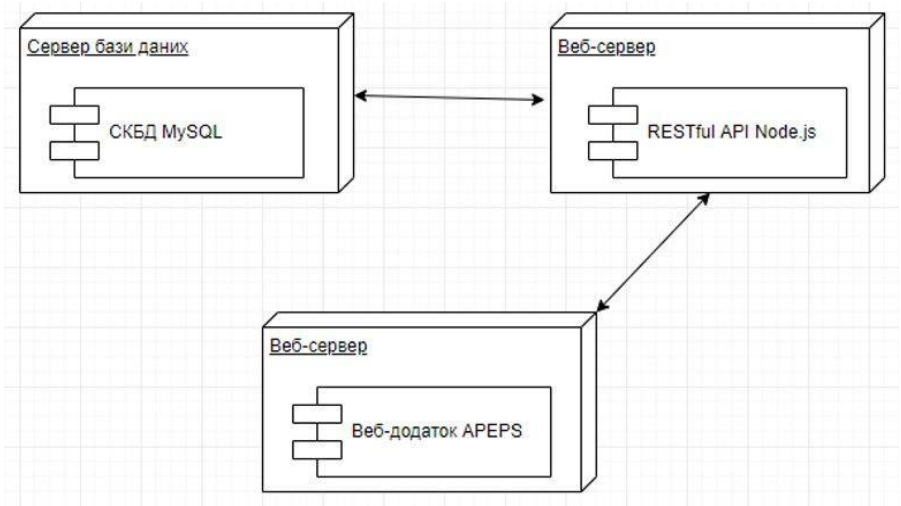

Рисунок 4.1 — Архітектура системи

Плюсами даної архітектури є:

- масштабованість;
- гнучкість. Оскільки кожен рівень може управлятися або масштабуватися незалежно, гнучкість збільшується;
- конфігурованість — ізольованість рівнів один від одного дозволяє швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів;
- висока безпека;
- висока надійність;
- низькі вимоги до швидкості каналу (мережі) між терміналами і сервером додатків;
- низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їх вартості.

Мінусами даної архітектури є:

- росте складність серверної частини і, як наслідок, витрати на адміністрування і обслуговування;
- більш висока складність створення додатків;
- складніше в розгортанні і адмініструванні;
- високі вимоги до продуктивності серверів додатків і сервера бази даних, а, значить, і висока вартість серверного обладнання;

Система управління дипломними роботами складається з трьох модулів, а саме: викладача, студента та додатковий персоналу. Система працює як односторінковий додаток.

Односторінковий додаток або SPA (Single Page Application) — односторінковий JavaScript додаток, що запускається і працює в браузері. На відміну від “традиційного” сайту, архітектура на SPA-сайтах побудована так, що рендеринг сторінки повністю відбувається на стороні клієнта, а не на стороні сервера. У браузері користувача запускається JavaScript-додаток, а весь необхідний вміст сторінок динамічно завантажується за допомогою AJAX. Навігація по сайту відбувається без

перезавантаження сторінок. За рахунок такої архітектури, SPA-сайти працюють швидше, ніж “традиційні” сайти (рисунок 4.2).

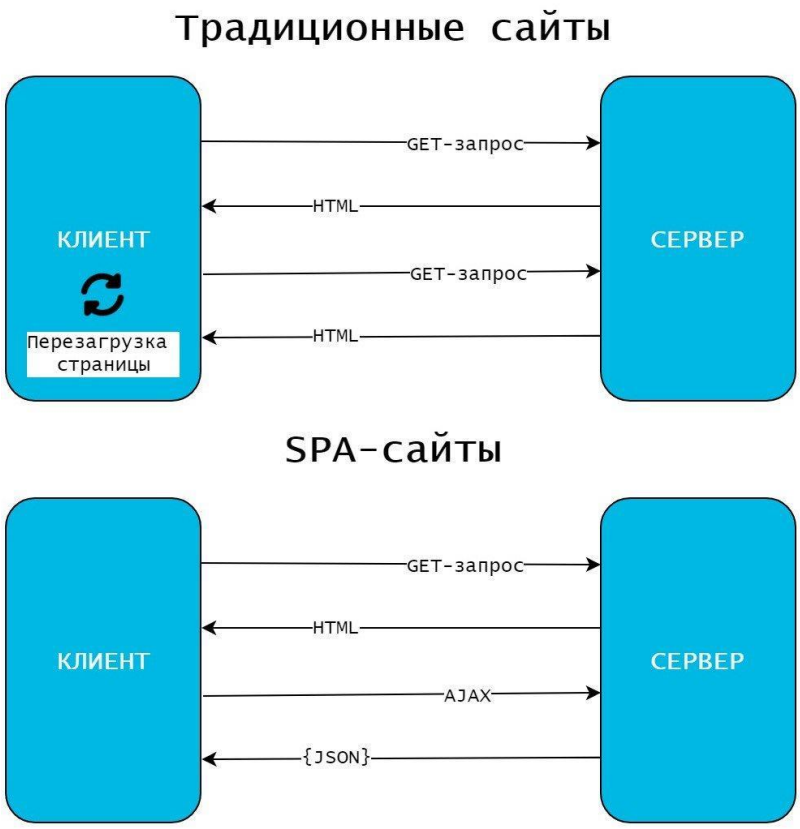

Рисунок 4.2 – Схема роботи SPA

Розглянемо детальніше, як відбувається завантаження і рендеринг вмісту на SPA-сайтах:

- Користувач запитує HTML вміст сайту.
- У відповідь приходить JavaScript-додаток.
- Додаток визначає, на якій сторінці знаходиться користувач, і який вміст йому потрібно відобразити.
- За допомогою AJAX користувач отримує необхідний контент: CSS, JS, HTML і текстовий контент.
- JavaScript-додаток обробляє отримані дані і відображає їх в браузері.
- При навігації по сайту оновлюється не вся сторінка, а тільки необхідний вміст.

Завдяки SPA відкриваються певні можливості, а саме: висока швидкість роботи, швидка розробка, створення версій для різних платформ на основі готового коду. Але

існують і мінуси такого підходу: JavaScript не обробляється більшістю пошукових систем; SPA-сайти не працюють без включеного JS в браузері.

В розробці React компоненти розділялися на наступні категорії:

- Контейнер (container або розумний компонент) — це компонент успадкований від Component і містить логіку і / або змінює свій стан. Використовується для породження і ініціалізації інших компонентів. Його так само можна назвати statefull.
- Презентаційний (presentational або пустий компонент) — це компонент успадкований від Component не має логіки і не змінює свій стан. Може мати власні стилі. Так само його називають ще stateless. Застосовується для відображення даних, як правило отриманих через props. Зараз його використовувати немає сенсу, так як для його заміни був створений новий вид компонентів описуваний нижче.
- Функціональні (або пустий компоненти так само є презентаційними) — це компоненти-функції які не мають логіки і не зберігають стан і мають власні стилі. Так само називаються stateless. В їх обов'язок входить відображати дані отримані через props.
- Чистий (pure) — компонент успадкований від PureComponent, може відноситься як до першої категорії так і другої і третьої. В його завдання входить виробляти неглибоке порівняння стану компонента щоб уникнути зайвого рендеру і позбавити програміста від написання зайвої перевірки.

Переваги такого підходу:

- Кращий розподіл відповідальності.
- Краще перевикористання. Ви можете використовувати один і той же пустий компонент з абсолютно різними джерелами стану.

В моїй системі компоненти контейнери містяться в директорії з назвою “containers”, а презентаційні в: “components” (рисунок 4.2).

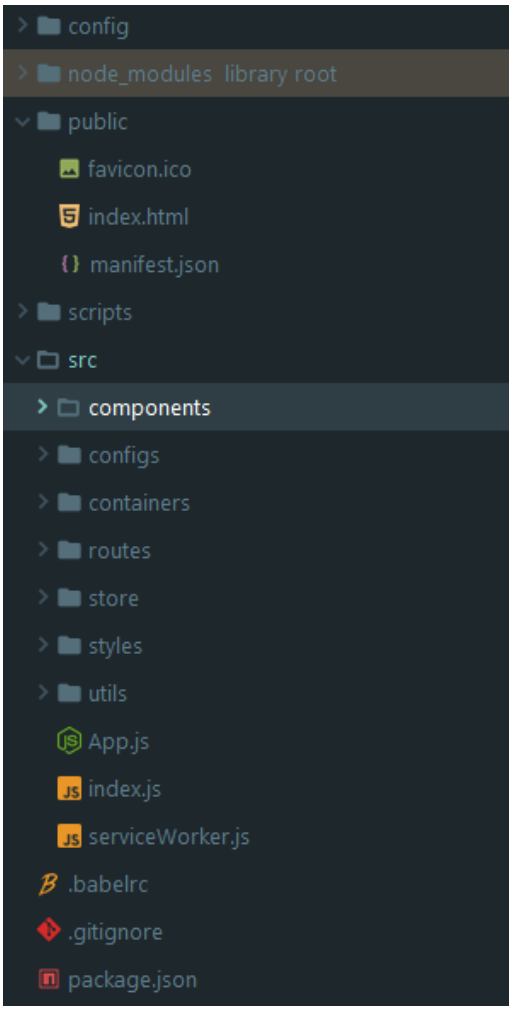

Рисунок 4.3 — Структура клієнтської частини системи

Пояснення найбільш важливих директорій та файлів (рисунок 4.3):

- config — містить файли конфігурацій для зборки проекту;
- node_modules — встановлені залежності проекту;
- public — статичні файлі;
- scripts — містить необхідний для запуску проекту функціонал;
- routes — налаштування адресних шляхів додатку;
- store — сховище Redux;
- styles — стилізовані компоненти;
- utils — загальний функціонал додатку;
- файл package.json — містить необхідні залежності проекту;
- файл babelrc — це локальна конфігурація коду в проекті;

- файл .gitignore — допоміжний файл який вказує системі контролю версій Git на файли та директорії за якими не потрібно наглядати;

Проблеми масштабування сторінки при зміні розміру вікна браузера або при використанні мобільних пристроїв для відображення web-порталу було розв’язано за допомогою використання Styled Components.

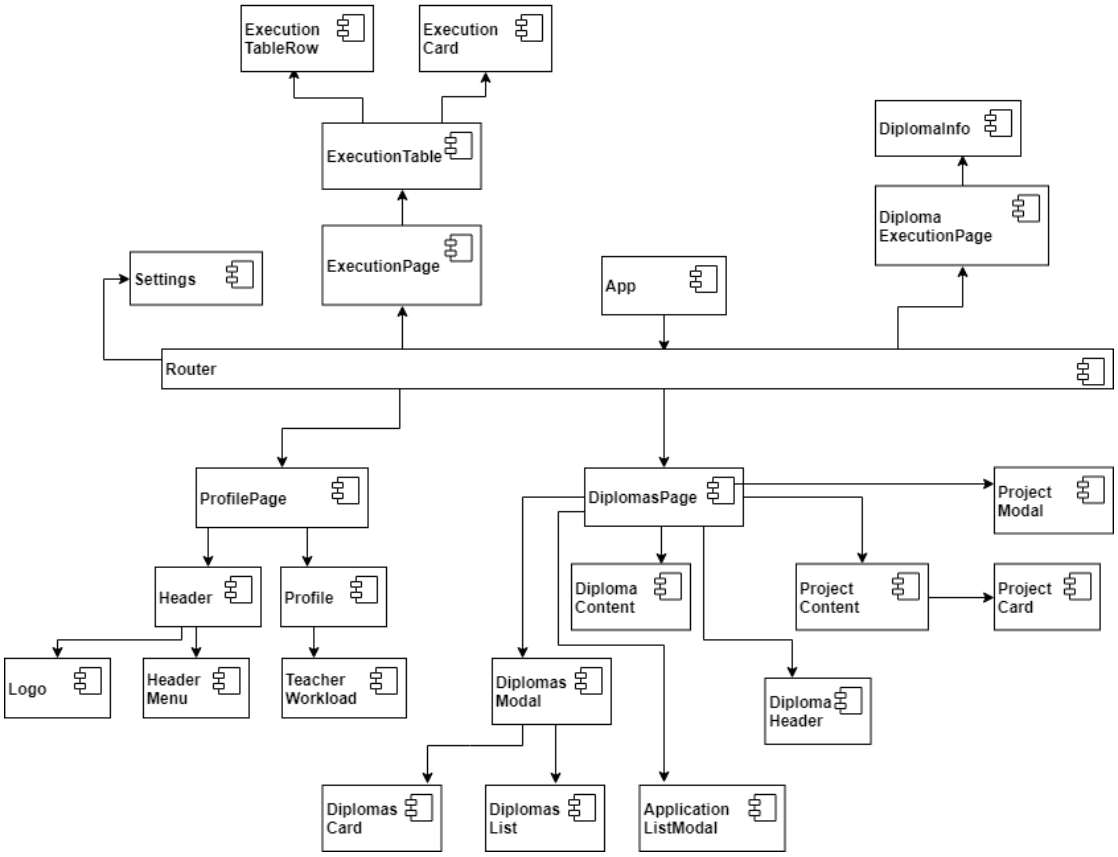

Рисунок 4.4 — Діаграма компонентів розробленого модулю викладача

За допомогою мови моделювання UML на рисунку 4.4 представлена діаграма React компонентів які були створені в процесі розробки програмного продукту.

4.2 Проектування бази даних

База даних (БД) — це добре організована і структурована колекція даних. База даних фактично побудована і підтримується за допомогою мови програмування бази

даних. Основною метою БД є боротьба з великою кількістю даних шляхом зберігання, керування та вилучення даних.

На сьогодні баз даних управляється за допомогою системи управління базами даних (СУБД). База даних насправді являє собою таблицю, що складається з декількох стовпців і рядків, де кожен стовпець має певний атрибут і кожен рядок вказує певне значення для відповідного атрибута.

Кількість стовпців залежить від кількості категорій / типів інформації, яка має зберігатися в базі даних, тоді як кількість рядків ґрунтується на кількості об'єкта. За допомогою цієї простої організації комп'ютерна програма може легко вибирати та обробляти необхідну інформацію.

Сервер MySQL надає систему управління базами даних з можливостями запитів і підключення, а також здатністю мати відмінну структуру даних і інтеграцію з багатьма різними платформами. Вона може обробляти великі бази даних надійно і швидко в умовах високих вимог. Сервер MySQL також надає багатофункціональні можливості, такі як підключення, швидкість і безпека, що робить його придатним для доступу до баз даних.

Сервер MySQL працює в системі клієнта і сервера. Ця система включає в себе багатопотоковий сервер SQL, який підтримує різноманітні бекенди, різні клієнтські програми та бібліотеки, засоби адміністрування та багато інтерфейсів прикладного програмування (API).

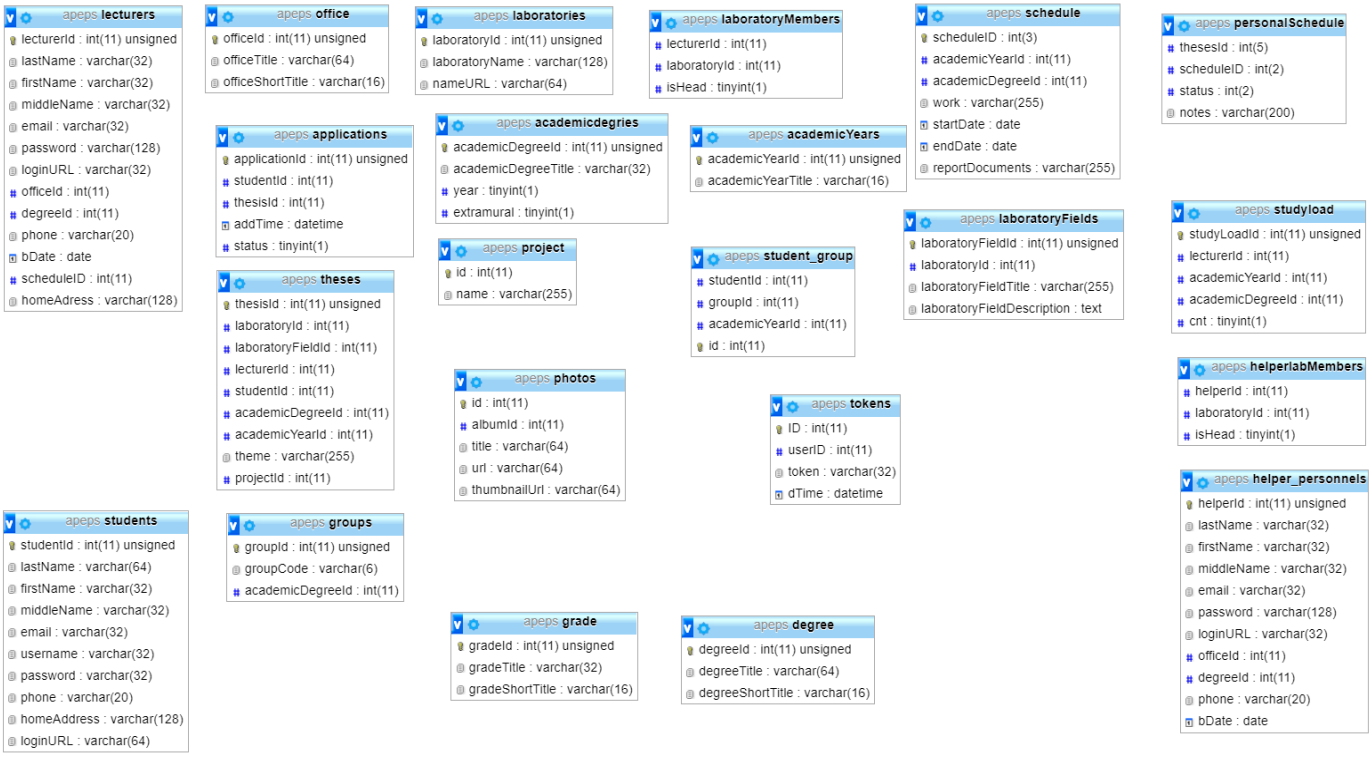

Рисунок 4.5 — Модель бази даних створеної системи

На рисунку 4.5 показана модель бази даних системи управління дипломним проектуванням кафедри. Кожен блок представляє собою таблицю, яка має назви та поля певного типу. Головними таблицями є:

- Lecturers — зберігається інформація про викладачів
- Students — зберігається інформація про студентів
- Applications — список заявок від студентів на певні теми дипломних робіт.
- Theses — список тем для дипломних робіт.
- PersonalSchedule — графік виконаня дипломної роботи.
- Groups — існуючі групи.
- Laboratories — лабораторії.
- LaboratoryFields — напрями лабораторій.
- Project — проекти дипломних робіт.
- Tokens — інформація авторизування користувачів.

4.3 Розробка в команді

Розробка програмного забезпечення в певній мірі є творчим процесом. Оскільки існує багато стилів написання коду, іменування змінних, шаблонів проектування, технік. Завжди існує багато варіантів реалізації певних задач. Тому на начальних етапах розробка в команді потребує оговорення певних правил, яких повинні дотримуватись усі. Також повинен бути певний механізм контролю за дотриманням цих правил.

Для дотримання стилю написання коду ми в команді використовували наступні розширення для IDE PhpStorm:

- ESLint
- Prettier

ESLint виконує автоматичне сканування JavaScript-файлів для поширених помилок синтаксису та стилю. Ймовірно, найкращий з існуючих. Серед його достоїнств варто виділити:

1. Велика кількість перевірок. Їх настільки багато, що прочитання всіх описів і обрання потрібного набору правил може зайняти декілька днів.
2. Автоматичне виправлення коду. ESLint бере на себе всю рутину по виправленню простих помилок, таких як стильові недоліки, забутий debugger, не зазначений "use strict" і багато інших.
3. Легка розширюваність. Існують готові сторонні плагіни з наборами перевірок. Наприклад, є готові плагіни для React або Node.js. Знайти інші плагіни можна в npm по префіксу "eslint-plugin-". Якщо цього все ще мало, можна написати свої власні перевірки. Це досить просто.
4. Добре розвинена інтеграція з редакторами коду, системами збирання і не тільки.
5. Активний розвиток. Новини про оновлення з'являються із завидною регулярністю.
6. Підтримка новітніх стандартів

7. Відмінна документація. Наприклад, можна легко використовувати готові конфігурації від Google або Airbnb.

Prettier сканує файли для знаходження порушень правил стилю написання кода та автоматично переформатує код, щоб забезпечити дотримання правильних правил для відступу, інтервалу, крапки з комою, одинарних лапок або подвійних лапок тощо. Головні переваги використання цих розширень:

- Єдиний стиль написання коду.
- Висока якість розробленого програмного продукту.
- Систему легко підтримувати.
- Заощадження часу на написанні коду, тому що ми можемо безпечно ігнорувати всі стильові проблеми і зосередитися на речах, які насправді мають значення, таких як структура та семантика коду.
- Знаходження помилок. ESLint фактично ловить багато синтаксичних помилок і простих форм помилок типу, таких як невизначені змінні.
- Встановлення цих речей є одноразовою задачею, яка заощадить час для всього подальшого процесу розробки.

В якості системи контролю версія обрана система Git.

Системи контролю версій (СКВ) дають можливість декільком розробникам працювати над одним проектом і зберігати внесені зміни. СКВ були створені для вирішення проблеми взаємодії з іншими розробниками. Часто розробники працюють в команді над одним проектом, а значить відразу кілька людей можуть змінювати один файл одночасно. Щоб уникнути плутанини в таких випадках використовують систему контролю версій, яка дозволяє зберігати історію змін у проекті та при необхідності допомагає повернутися до попередньої версії.

Git варто виокремити від інших СКВ через підхід до роботи з даними. Більшість інших систем зберігають інформацію у вигляді списку змін в файлах. Замість цього, підхід Git до зберігання даних більше схожий на набір знімків мініатюрної файлової системи. Кожен раз, коли ви зберігаєте стан свого проекту в Git, система запам'ятовує, як виглядає кожен файл в цей момент, і зберігає посилання на цей знімок.

Розробка функціоналу відбувається паралельно на різних гілках. Завжди є як мінімум одна головна гілка, наприклад з назвою “master”. Вона вміщає в собі весь коректно працюючий, тобто відтестований, функціонал. По завершені роботи над якимось функціоналом відправляється запит на злиття з головною гілкою (рисунок 4.6). Запит переглядають інші учасники команди, і якщо все добре — запит приймається, якщо ні — то вносяться зміни на робочу гілку з подальшим повторним запитом.

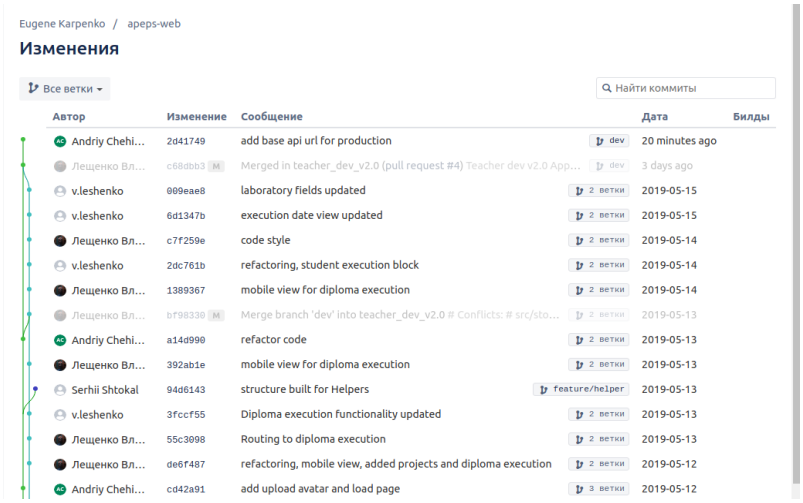

Рисунок 4.6 — Перегляд історії активності в системі Atlassian BitBucket.

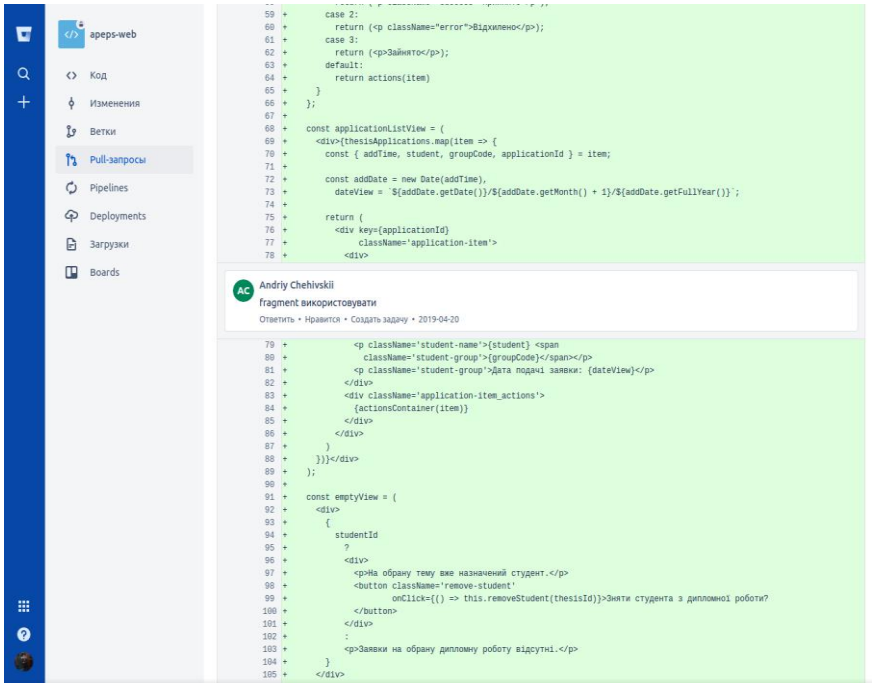

Рисунок 4.7 — Перегляд запиту на злиття двох гілок розробки в системі Atlassian BitBucket.

На рисунку 4.7 відображається активність команди розробників в системі контролю версій Git.

4.4 Опис функціональності системи

За допомогою програмного продукту викладач має такі можливості:

- Заходити в систему в будь-який час з будь-якого пристрію який має браузер та вихід до інтернету.
- Переглядати список, напрями та склад лабораторій.
- Створювати / редагувати / видаляти дипломні роботи.
- Створювати / редагувати / видаляти проекти.
- Переглядати список заявок від студентів на певну дипломну роботу.
- Переглядати загальний графік виконання дипломних робіт.
- Назначати студента на дипломну роботу.
- Контролювати кожен етап виконання дипломної роботи.
- Переглядати навантаження.
- Редагувати свої контактні дані.
- Змінювати тему оформлення.
- Переглядати список дипломних робіт

4.5 Висновки до розділу

Розробка програмного продукту відбувалася в команді розробників за допомогою системи контролю версій Git та розширень ESLint, Prettier для середовища розробки PhpStorm. Було розроблено продукт який характеризується:

- Високою якістю коду.
- Маштабованістю.
- Легкістю в підтримці.
- Швидкістю в роботі.
- Новітніх підходів до розробки.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Система розроблена на основі новітніх веб-технологій та найкращих практик у розробці. Працювати з нею можливо майже з будь-якого електронного пристрію який має браузер.

5.1 Встановлення та системні вимоги

Веб-додаток не потребує встановлення на пристрій користувача. Проте, для використання необхідний веб-браузер, який підтримує актуальні веб-стандарти. Також необхідно мати стабільний доступ до інтернету.

Таким чином, для того щоб розпочати роботу з системою необхідно мати будь-який електронний пристрій який задовольняє наступним критеріям:

- Мінімальна версія браузеру:
  - Chrome 6.0
  - Firefox 4.0
  - Internet Explorer 9.0
  - Opera 9.6
- Розширення екрану пристрою в ширину не менше ніж 320px.
- Увімкнена обробка JavaScript в налаштуваннях браузеру.

5.2 Інструкція з використання програмного продукту

При вході на клієнтський додаток, користувачеві необхідно авторизуватися в системі щоб почати працювати в системі. На рисунку 5.1 зображена форма авторизації. Для того щоб увійти необхідно ввести логін, пароль та тип користувачач.

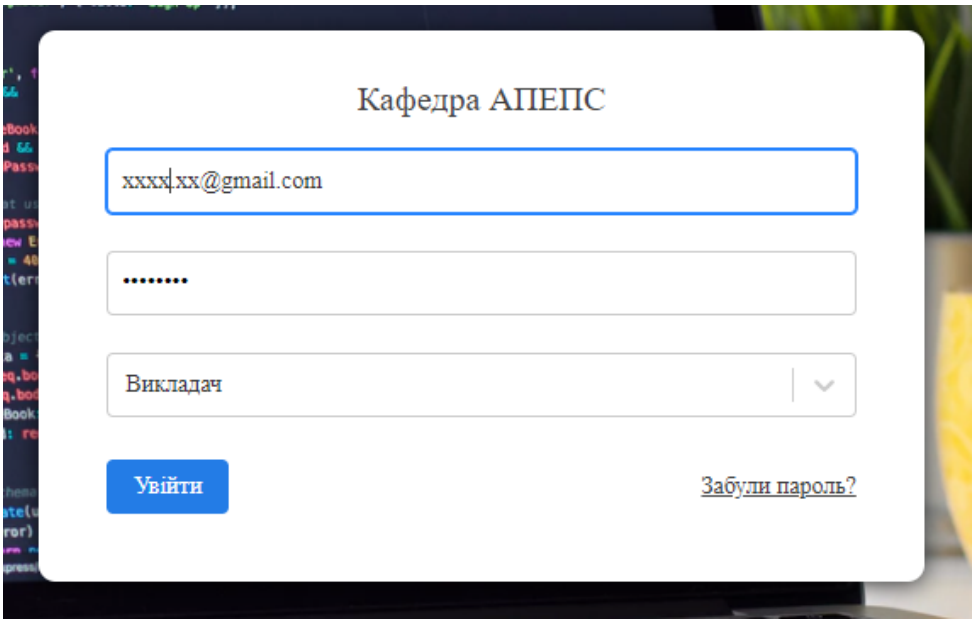

Рисунок 5.1 — Форма авторизації

На рисунку 5.2 зображена форма відновлення паролю.

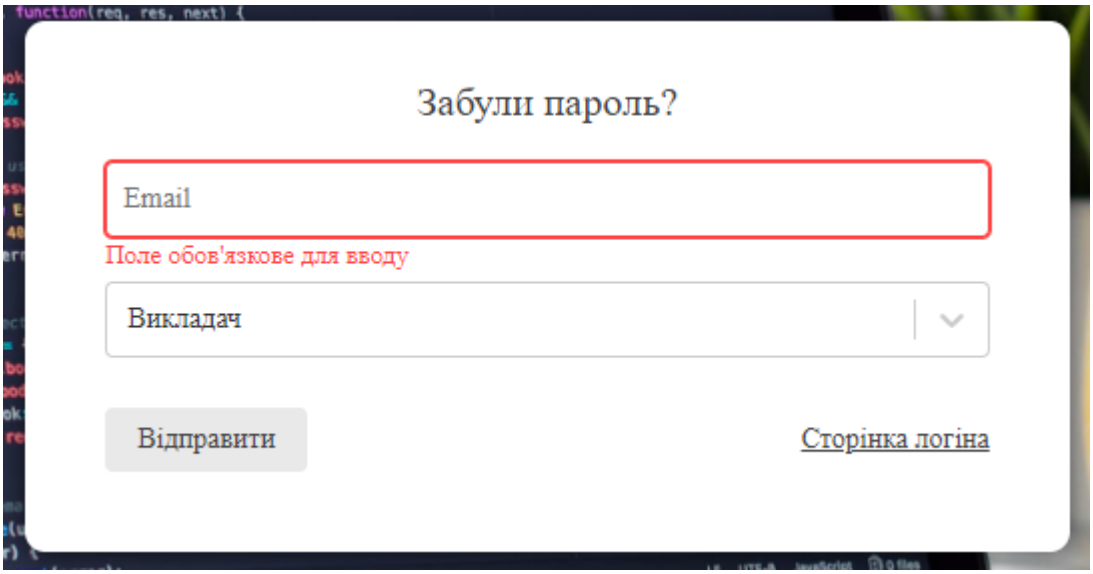

Рисунок 5.2 — Форма відновлення паролю

Після того, як користувач авторизувався в системі, він отримує доступ до головного меню системи. За допомогою цього меню користувач має доступ до усіх сторінок свого профілю, або має можливість вийти з свого профілю. Також в нього автоматично відкрився власний робочий простір – кабінет користувача. На рисунку 5.3 зображена сторінка особистого кабінету викладача.

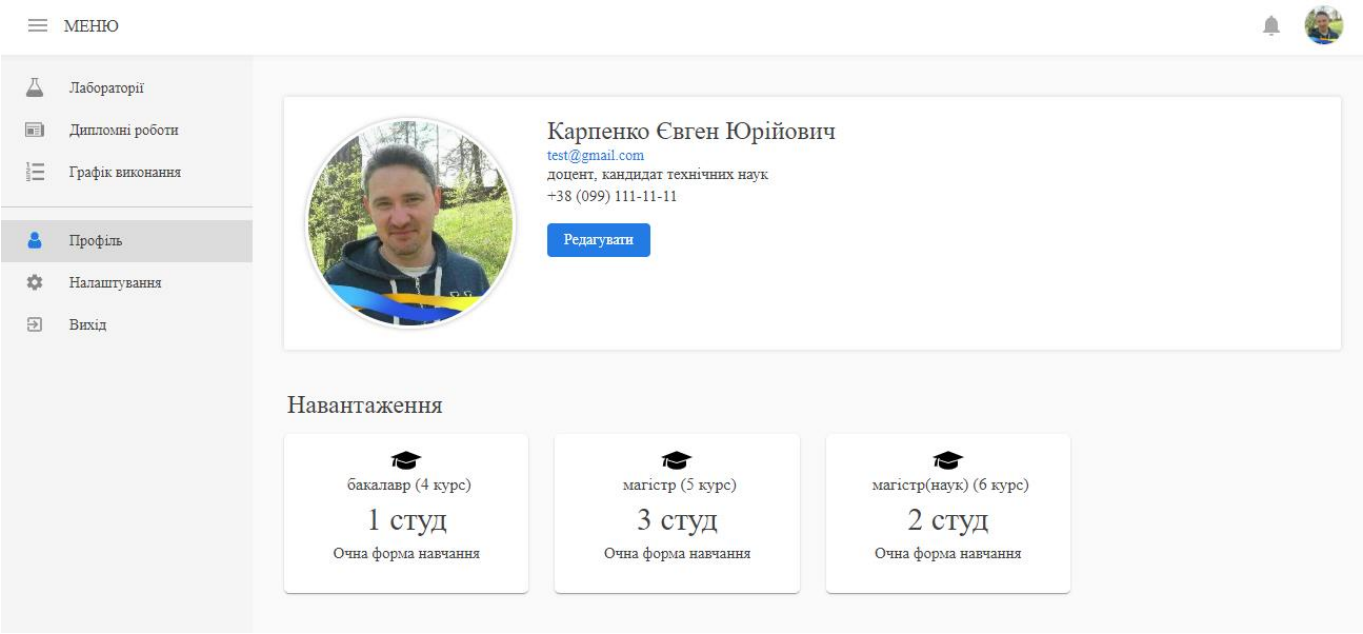

Рисунок 5.3 — Сторінка профілю викладача

На рисунку 5.3 викладач може побачити своє навантаження та контактні дані. Також з цього інтерфейсу є можливість змінити зображення профілю натиснувши на нього лівою кнопкою мишки.

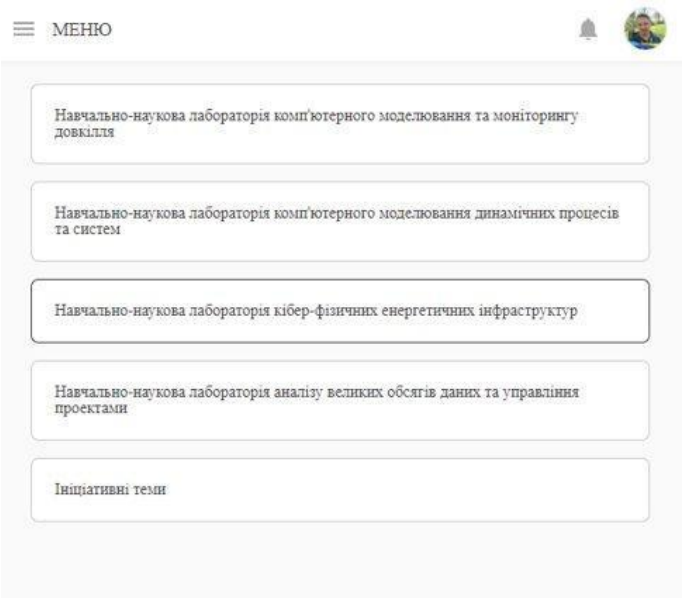

Рисунок 5.4 — Сторінка списку лабораторій

На рисунку 5.5 зображено список лабораторій, для перегляду детальної інформації потрібно натиснути на необхідну лабораторію.

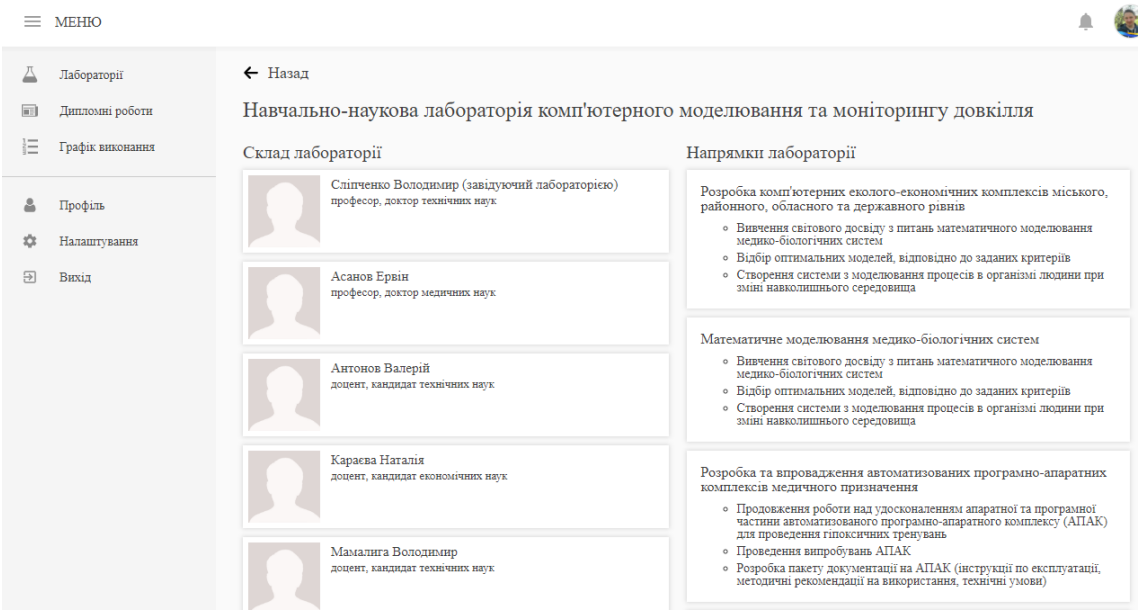

Рисунок 5.5 — Сторінка детальної інформації лабораторії

На рисунках 5.4 та 5.5 показані сторінки на яких можна побачити детальну інформацію про лабораторії, їх склад та напрямки.

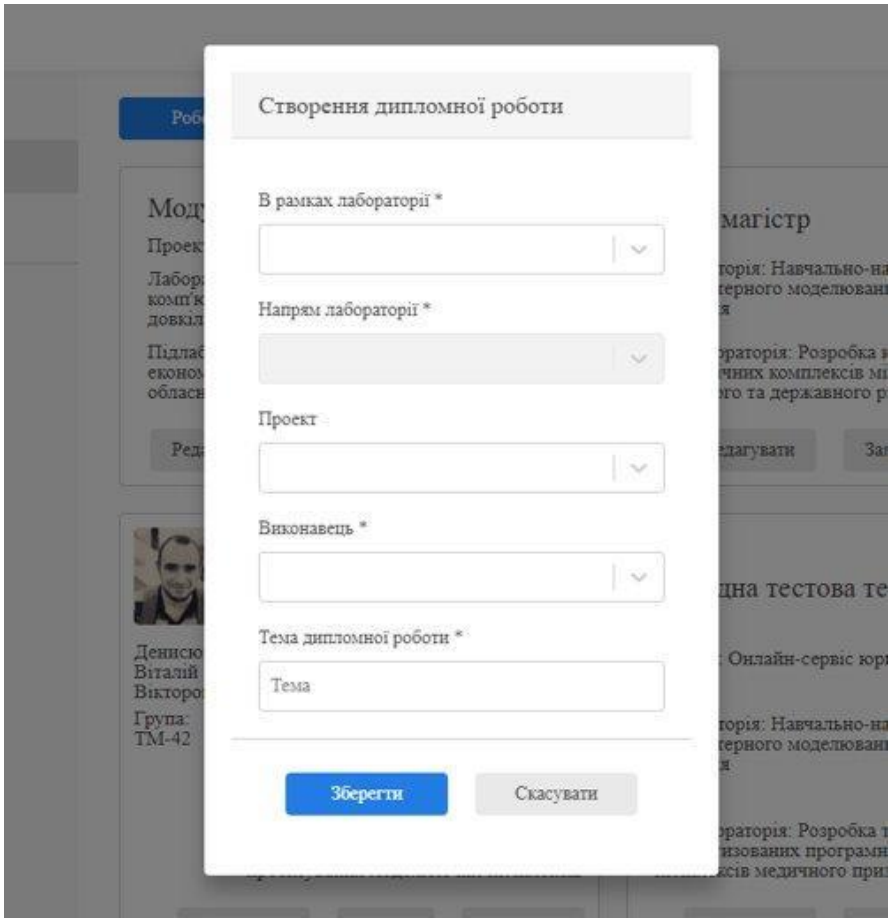

Рисунок 5.6 — Створення дипломної роботи

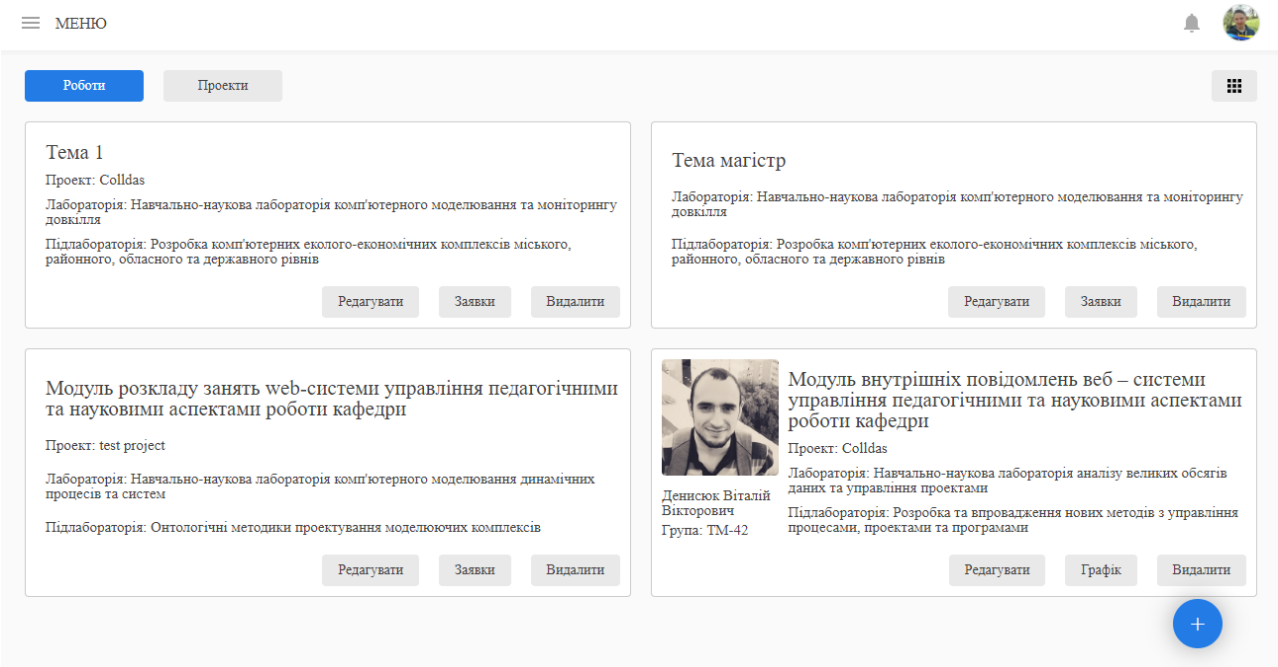

Рисунок 5.6 — Сторінка списку дипломних робіт

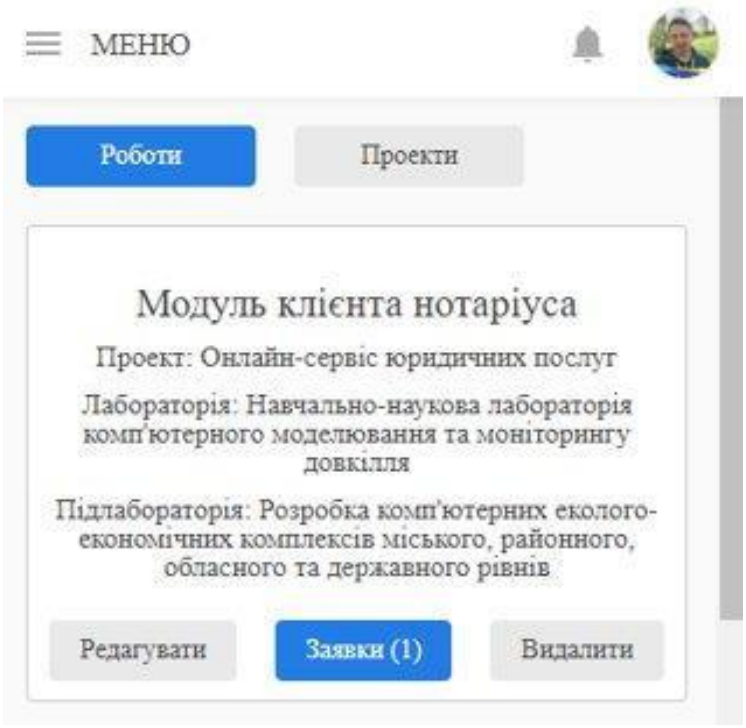

Рисунок 5.7 — Мобільна версія сторінки списку дипломних робіт

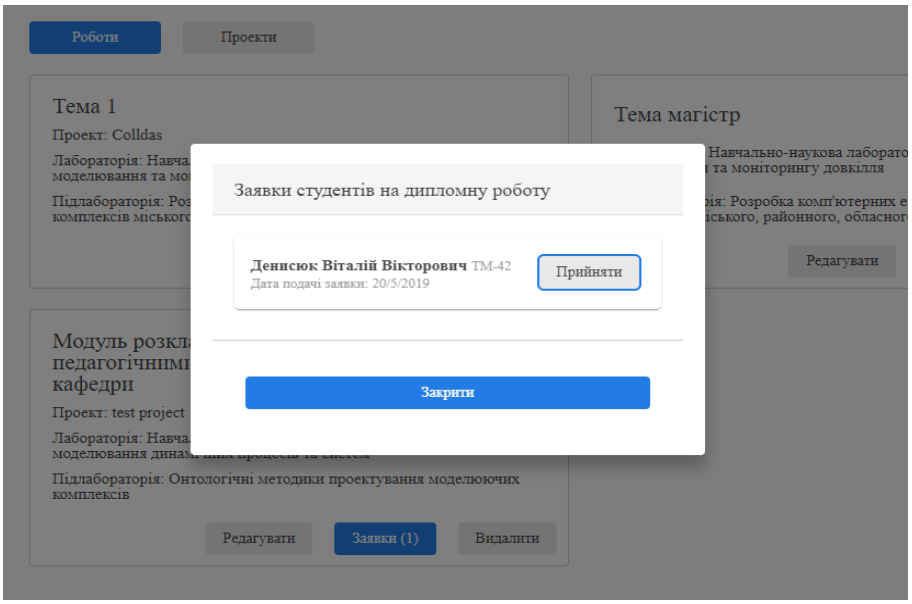

Рисунок 5.8 — Список заявок на дипломну роботу

На рисунках 5.6 та 5.7 показаний головний інтерфейс по управлінню дипломними роботами. Тут виводиться список дипломних робіт, з можливістю їх створення, редагування та видалення. Також натиснувши на кнопку “Заявки” ми маємо можливість переглянути заявки студентів на конкретну дипломну роботу (рисунок 5.8). Натиснувши на кнопку “Графік” ми переходимо до інтерфейсу контролю виконання цією дипломної роботи (рисунок 5.10). Натиснувши на вкладку “Проекти” ми відкриємо список проектів дипломних робіт (рисунок 5.9).

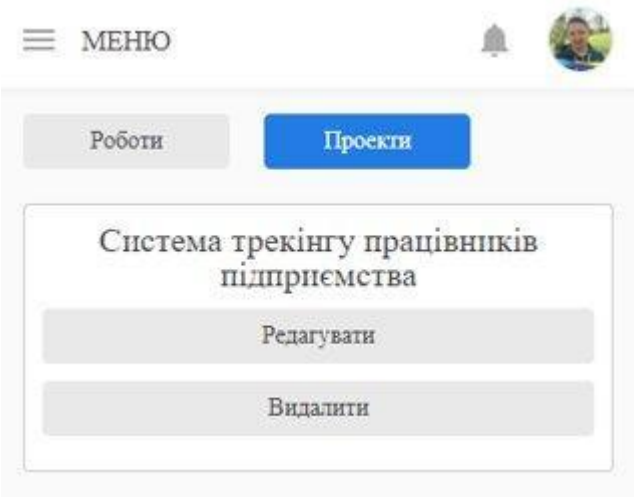

Рисунок 5.9 — Мобільна версія списку проектів дипломних робіт

На рисунку 5.9 показано список проектів з базовими операціями над ними: створення, редагування, видалення.

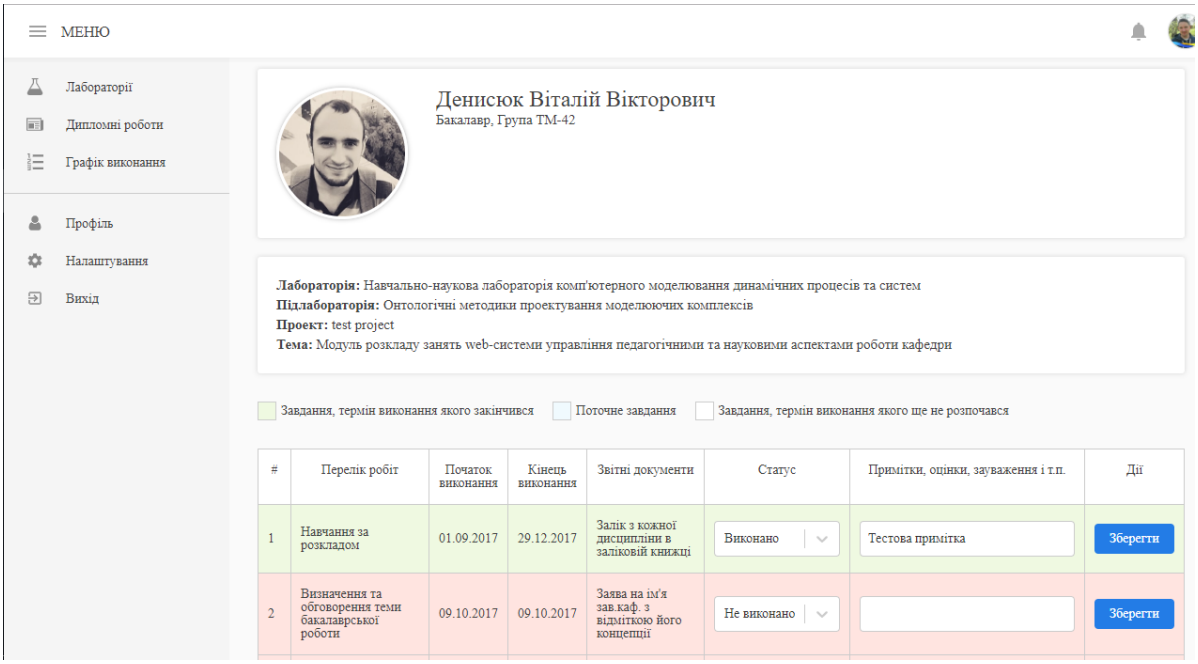

Рисунок 5.10 — Сторінка контролю виконання дипломної роботи

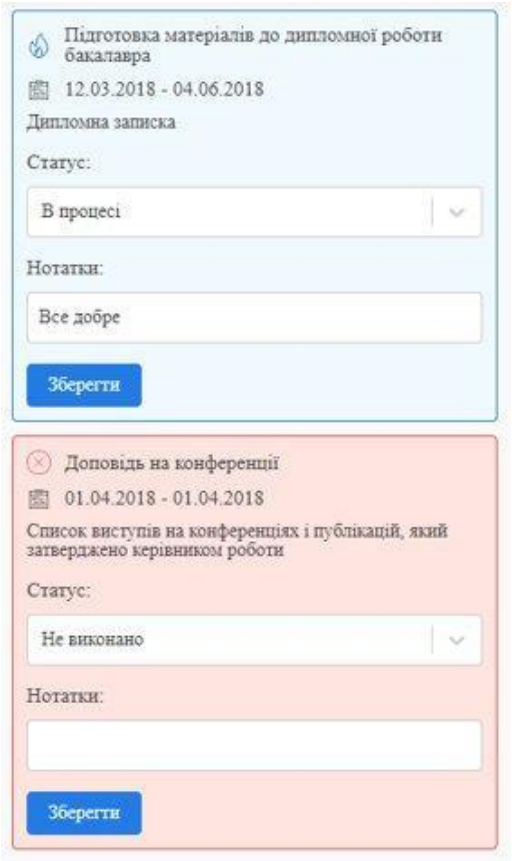

Рисунок 5.11 — Мобільний вигляд виконання дипломної роботи

На сторінці контролю виконання окремої дипломної роботи, яка показана на рисунках 5.10 та 5.11, виводиться інформація про дипломну роботу, студента, який її виконує, та надається можливість встановити актуальний статус виконання дипломної роботи, або зробити примітки на майбутнє. Після внесення необхідних даних потрібно натиснути на кнопку “Зберегти”.

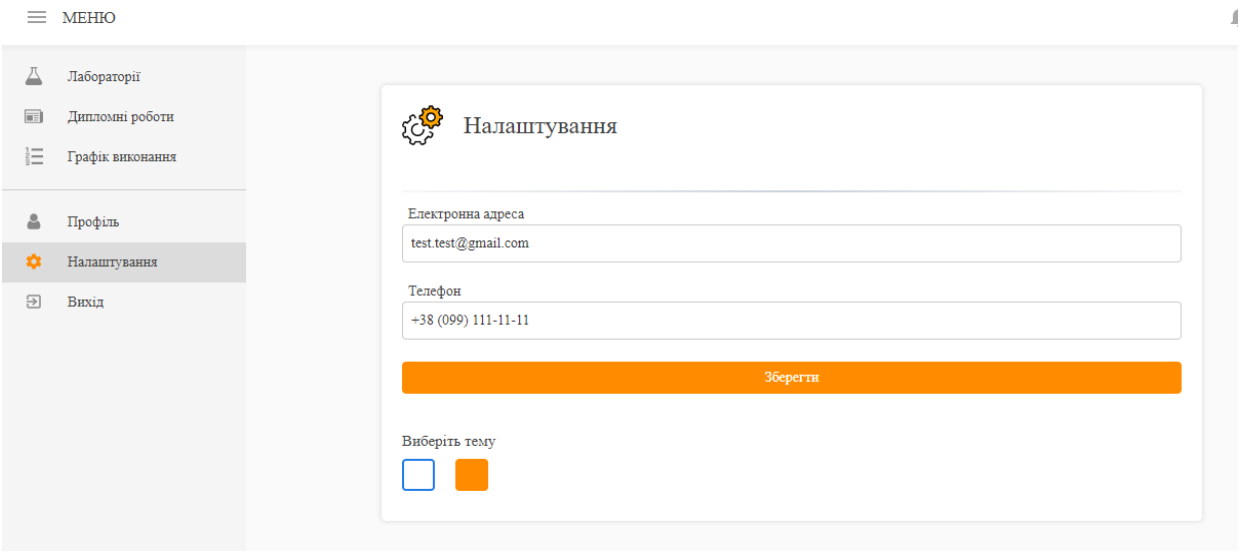

Рисунок 5.11 — Сторінка налаштувань

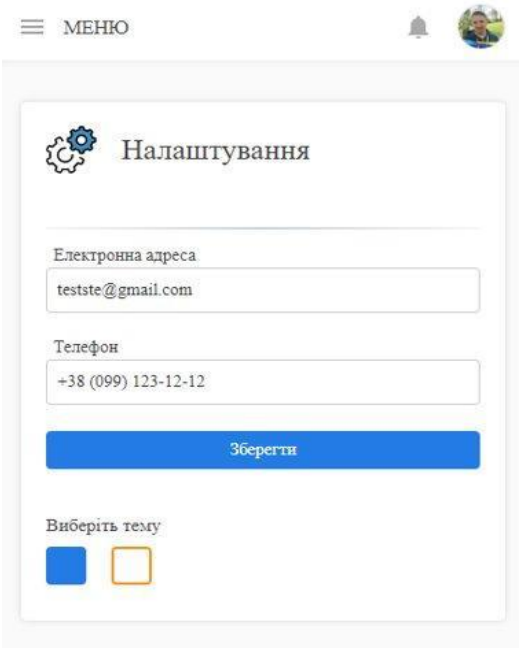

Рисунок 5.12 — Сторінка налаштувань в мобільному вигляді

На сторінці “Налаштування”, яка зображена на рисунку 5.9 викладачу надається можливість змінити свої контактні дані або тему оформлення.

Висновки

У ході аналізу існуючого програмного забезпечення управління дипломними роботами було досліджено системи, які слугують для вирішення поставлених задач. Аналіз показав, що існуючі системи вирішують задачу не у повному обсязі, вони є громіздкими та мають високі апаратні вимоги до пристроїв користувачів.

Розроблений програмний продукт дозволяє полегшити процес управління дипломними роботами.

Проведено огляд методів і засобів розробки програмної системи. Було повністю аргументовано вибір створення програмної системи, в якості веб-додатку, створеного з використанням новітніх технологій та підходів до розробки. Це зробило систему маштабованою, гнучкою, легкою в підтримці та розробці в цілому.

Користувачами системи є викладачі, студенти та додатковий персонал. Програмне забезпечення може бути використано майже на будь-якому електронному пристрої на якому встановлено браузер.

Отже, в ході розробки системи я покращив свої знання в різноманітних технологіях, які використовуються під час розробки програмного забезпечення. Отримав важливий досвід роботи в команді. Ознайомився з різноманітними техніками та алгоритмами реалізації програмного продукту, які лягли в основу розробленого програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Azat Mardan — Practical Node.js: Building Real-World Scalable Web Apps. — 2015.

2. Alex Banks, Eve Porcello — Learning React: Functional Web Development with React and Redux. — 2017.

3. Yuriy Dyimov — React.js: собираем с нуля изоморфное / универсальное приложение [Електронний ресурс]. — 2016. – Режим доступу: https://habr.com/ru/post/309958/.

4. Martin Fowler — GUI Architectures. Часть 1 [Електронний ресурс]. — 2009. — Режим доступу: http://www.rusdoc.ru/articles/18358/.

5. Martin Fowler — GUI Architectures. Часть 2 [Електронний ресурс]. — 2009 — Режим доступу: https://habr.com/post/53536/.

6. Болье А. — Learning SQL [Електронний ресурс]. — 2005. — Режим доступу: http://shop.oreilly.com/product/9780596007270.do.

ДОДАТОК 1

Модуль викладача в системі управління дипломним проектуванням кафедри

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52 19Б 81-1	Лещенко_В.Ю_ТМ52.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52 19Б 12-1	app.js	Модулі серверної частини
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52 19Б 12-2	App.js	Модуль клієнтської частини
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52 19Б 13-2	Опис.docx	Опис модуля інтерфейсу клієнтської частини програми

ДОДАТОК 2

Модуль викладача в системі управління дипломним проектуванням кафедри

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52_19Б 12-1

Аркушів 10

Київ – 2019

// підключення пакетів

```
{
    "name": "apeps-web",
    "version": "0.1.0",
    "private": true,
    "dependencies": {
      "@babel/core": "7.1.0",
      "@svgr/webpack": "2.4.1",
      "axios": "^0.18.0",
      "babel-core": "7.0.0-bridge.0",
      "babel-eslint": "^10.0.1",
      "babel-jest": "23.6.0",
      "babel-loader": "8.0.4",
      "babel-plugin-named-asset-import": "^0.2.3",
      "babel-polyfill": "^6.26.0",
      "babel-preset-react-app": "^6.1.0",
      "bfj": "6.1.1",
      "case-sensitive-paths-webpack-plugin": "2.1.2",
      "chalk": "2.4.1",
      "css-loader": "1.0.0",
      "dotenv": "6.0.0",
      "dotenv-expand": "4.2.0",
      "eslint": "^5.10.0",
      "eslint-config-react-app": "^3.0.5",
      "eslint-loader": "2.1.1",
      "eslint-plugin-flowtype": "2.50.1",
      "eslint-plugin-import": "2.14.0",
      "eslint-plugin-jsx-a11y": "6.1.2",
      "eslint-plugin-react": "7.11.1",
      "file-loader": "2.0.0",
      "fork-ts-checker-webpack-plugin-alt": "0.4.14",
      "formik": "^1.4.2",
      "fs-extra": "7.0.0",
      "html-webpack-plugin": "4.0.0-alpha.2",
      "identity-obj-proxy": "3.0.0",
      "jest": "23.6.0",
      "jest-pnp-resolver": "1.0.1",
      "jest-resolve": "23.6.0",
      "js-cookie": "^2.2.0",
      "lodash": "^4.17.11",
      "mini-css-extract-plugin": "0.4.3",
      "moment": "^2.23.0",
      "optimize-css-assets-webpack-plugin": "5.0.1",
      "pnp-webpack-plugin": "1.1.0",
```

```
    "postcss-flexbugs-fixes": "4.1.0",
    "postcss-loader": "3.0.0",
    "postcss-preset-env": "6.0.6",
    "postcss-safe-parser": "4.0.1",
    "react": "^16.8.3",
    "react-app-polyfill": "^0.1.3",
    "react-avatar-editor": "^11.0.7",
    "react-dev-utils": "^6.1.1",
    "react-dom": "^16.8.3",
    "react-input-mask": "^2.0.4",
    "react-notification-system": "^0.2.17",
    "react-onclickoutside": "^6.7.1",
    "react-redux": "^6.0.0",
    "react-router-dom": "^4.3.1",
    "react-select": "^2.1.2",
    "react-transition-group": "^2.6.1",
    "redux": "^4.0.1",
    "redux-devtools-extension": "^2.13.7",
    "redux-logger": "^3.0.6",
    "redux-saga": "^0.16.2",
    "reselect": "^4.0.0",
    "resolve": "1.8.1",
    "sass-loader": "7.1.0",
    "style-loader": "0.23.0",
    "styled-components": "^4.1.2",
    "terser-webpack-plugin": "1.1.0",
    "url-loader": "1.1.1",
    "webpack": "4.19.1",
    "webpack-dev-server": "3.1.9",
    "webpack-manifest-plugin": "2.0.4",
    "workbox-webpack-plugin": "3.6.3",
    "yup": "^0.26.7"
  },
  "scripts": {
    "start": "cross-env REACT_APP_HOST_ENV=development node scripts/start.js",
    "build:production": "cross-env REACT_APP_HOST_ENV=production node
scripts/build.js",
    "build:development": "cross-env REACT_APP_HOST_ENV=development node
scripts/build.js",
    "build:stage": "cross-env REACT_APP_HOST_ENV=stage node scripts/build.js",
    "test": "node scripts/test.js",
    "lint": "standard --fix"
  },
  "standard": {
```

```
    "parser": "babel-eslint",
    "ignore": [
      "/config/**/*.js"
    ]
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": [
    ">0.2%",
    "not dead",
    "not ie <= 11",
    "not op_mini all"
  ],
  "jest": {
    "collectCoverageFrom": [
      "src/**/*.{js,jsx,ts,tsx}",
      "!src/**/*.d.ts"
    ],
    "resolver": "jest-pnp-resolver",
    "setupFiles": [
      "react-app-polyfill/jsdom"
    ],
    "testMatch": [
      "<rootDir>/src/**/__tests__/**/*.{js,jsx,ts,tsx}",
      "<rootDir>/src/**/?(*.)(spec|test).{js,jsx,ts,tsx}"
    ],
    "testEnvironment": "jsdom",
    "testURL": "http://localhost",
    "transform": {
      "^.+\\.(js|jsx|ts|tsx)$": "<rootDir>/node_modules/babel-jest",
      "^.+\\.css$": "<rootDir>/config/jest/cssTransform.js",
      "^(?!.*\\.(js|jsx|ts|tsx|css|json)$)": "<rootDir>/config/jest/fileTransform.js"
    },
    "transformIgnorePatterns": [
      "[/\\\\]node_modules[/\\\\].+\\.(js|jsx|ts|tsx)$",
      "^.+\\.module\\.(css|sass|scss)$"
    ],
    "moduleNameMapper": {
      "^react-native$": "react-native-web",
      "^.+\\.module\\.(css|sass|scss)$": "identity-obj-proxy"
    },
    "moduleFileExtensions": [
      "web.js",
```

```
            "js",
            "web.ts",
            "ts",
            "web.tsx",
            "tsx",
            "json",
            "web.jsx",
            "jsx",
            "node"
        ]
    },
    "devDependencies": {
        "@babel/plugin-proposal-decorators": "^7.2.0",
        "babel-plugin-module-resolver": "^3.1.1",
        "babel-plugin-styled-components": "^1.9.4",
        "babel-plugin-transform-decorators-legacy": "^1.3.5",
        "cross-env": "^5.2.0",
        "eslint-plugin-module-resolver": "^0.8.0",
        "standard": "^12.0.1"
    }
}

// Главный компонент

/* REACT */
import React from 'react'

/* MODULES */
import { Switch, Router, Route, Redirect } from 'react-router-dom'
import { connect } from 'react-redux'
import { ThemeProvider } from 'styled-components'

/* CUSTOM MODULES */
import RouteWithSubRoutes from '/src/routes/routeWithSubRoutes'
import { Layout } from '/src/components/HOC'
import history from './utils/history'
import { AuthService } from '/src/utils'
import { Loader } from '/src/components/UI'
import GlobalLoader from '/src/components/GlobalLoader'
import GlobalStyle from '/src/styles/global'
import { NotificationProvider } from './components/HOC'
import AuthPage from './containers/Auth'
import ForgotPasswordPage from '/src/containers/Auth/ForgotPassword'
import { routesSelector } from '/src/store/selectors/authSelectors'
```

```
/* ACTIONS */
import { loginAction, tryAutoLogin, loadedApp } from './store/actions/authActions'
import { referenceAction } from './store/actions/referenceActions'

/* STYLES */
import { getTheme } from './styles/theme'

@connect(
  state => ({
    auth: state.auth,
    settings: state.settings,
    ROUTES: routesSelector(state)
  }),
  { loginAction, tryAutoLogin, loadedApp, referenceAction }
)
class App extends React.Component {
  componentDidMount () {
    const { tryAutoLogin, referenceAction, loadedApp } = this.props

    if (AuthService.getCredentials()) {
      tryAutoLogin(() => { referenceAction() })
    } else {
      loadedApp()
    }
  }

  render () {
    const { auth, settings, ROUTES } = this.props
    const theme = getTheme(settings.theme)
    // Can use different switches or private route and login route for protect routes
    let routes = (
      <Switch>
        {(
          ROUTES && ROUTES.map(route => (
            <RouteWithSubRoutes
              key={route.path}
              {...route}
            />
          ))
        )}
        <Redirect to='/' />
      </Switch>)
```

```
    if (!auth.token) {
      routes = (<Switch>
        <Route path='/login' component={AuthPage} exact />
        <Route path='/forgot-password' component={ForgotPasswordPage} exact />
        <Redirect to='/login' />
      </Switch>)
    }

    let content = (
      <Layout>
        {routes}
      </Layout>
    )
    if (!auth.isLoaded) content = (<Loader size='3rem'>Авторизація...</Loader>)

    return (
      <Router history={history}>
        <ThemeProvider theme={theme}>
          <NotificationProvider>
            <GlobalStyle />
            {content}
            {auth.isLoaded && settings.isGlobalLoading &&
<GlobalLoader>Завантаження...</GlobalLoader>}
          </NotificationProvider>
        </ThemeProvider>
      </Router>
    )
  }
}

export default App


// Компонент дипломної роботи
import React, { Component } from 'react'
import { connect } from 'react-redux'
import {
  changeLecturerDiplomasTab,
  fetchApplication,
  fetchDiplomasAction, openDiplomaApplicationsAction,
  openDiplomaModalAction, removeDiplomaAction
} from '../../../store/actions/lecturerDiplomasActions'
import { FloatingButton } from '../../UI/FloatingButton'
import { fetchProjects, openProjectModal } from '../../../store/actions/projectActions'
```

```
import { isMobileSelector } from '../../../store/selectors/settingsSelectors'
import { NotificationContext } from '../../HOC/NotificationProvider'
import Header from './Header'
import DiplomaContent from './DiplomaContent'
import DiplomasModal from './Modals/DiplomasModal'
import ApplicationListModal from './Modals/ApplicationListModal'
import ProjectModal from './Modals/ProjectModal'
import ProjectContent from './ProjectContent'
import { changeViewModeAction } from '../../../store/actions/settingsActions'
import { lecturerDiplomasSelector } from '../../../store/selectors/lecturerDiplomasSelectors'
import { Title } from '../../UI/Title'
import { Button } from '../../UI'
import Modal from '../../Modal'
import { ModalBody, ModalFooter } from '../../Modal/style'

export const DiplomasContext = React.createContext()

const mapStateToProps = (state) => ({
  diplomas: lecturerDiplomasSelector(state),
  activeTab: state.lecturerDiplomas.activeTab,
  projects: state.projects.projects,
  projectListIsLoading: state.projects.loading,
  singleViewMode: state.settings.singleViewMode,
  isMobile: isMobileSelector(state)
})

@connect(
  mapStateToProps,
  {
    fetchDiplomasAction,
    removeDiplomaAction,
    openDiplomaModalAction,
    openDiplomaApplicationsAction,
    fetchApplication,
    fetchProjects,
    changeLecturerDiplomasTab,
    openProjectModal,
    changeViewModeAction
  }
)
class LecturerDiplomas extends Component {
  state = {
    removeConfirmModalIsOpen: false,
    diploma: null
```

```
}
static contextType = NotificationContext

componentDidMount () {
  this.props.fetchDiplomasAction()
  this.props.fetchApplication()
  this.props.fetchProjects()
}

changeView = () => {
  const { changeViewModeAction, singleViewMode } = this.props

  window.localStorage.setItem('single-view-mode', `${!singleViewMode}`)
  changeViewModeAction(!singleViewMode)
}

removeDiplomaAction = (diploma) => {
  this.setState({
    removeConfirmModalIsOpen: true,
    diploma,
  })
}

submitHandler = () => {
  this.props.removeDiplomaAction(this.state.diploma, () => {
    this.context.addNotification({
      title: 'Success',
      message: 'Дипломну роботу було успішно видалено',
      level: 'success'
    })
    this.closeModal()
    this.props.fetchDiplomasAction()
  })
}

create() {
  if (this.props.activeTab === 'diplomas') {
    this.props.openDiplomaModalAction()
  } else if (this.props.activeTab === 'projects') {
    this.props.openProjectModal()
  }
}

closeModal () {
```

```
    this.setState({
      removeConfirmModalIsOpen: false,
      diploma: null
    })
  }

  render () {
    const { diplomas, projects, activeTab, isMobile, singleViewMode } = this.props

    const container = (activeTab === 'diplomas') ? <DiplomaContent /> : <ProjectContent
/>
    const removeConfirmationModal = (
      <Modal isOpen={this.state.removeConfirmModalIsOpen} closeHandler={() =>
this.closeModal()} center centerButton>
        <ModalBody style={{ margin: '1rem' }}>
          <Title
            size='2rem'
            align='center'
            title={`Ви впевнені, що хочете видалити дипломну роботу?`}/>
        </ModalBody>
        <ModalFooter>
          <Button width='12rem' onClick={() => this.closeModal()}>Відмінити</Button>
          <Button primary width='12rem' onClick={() =>
this.submitHandler()}>Видалити</Button>
        </ModalFooter>
      </Modal>
    )

    return (
      <DiplomasContext.Provider value={{
        singleViewMode,
        isMobile,
        changeView: this.changeView,
        activeTab,
        diplomas,
        projects,
        changeTab: this.props.changeLecturerDiplomasTab,
        openDiplomaModalAction: this.props.openDiplomaModalAction,
        openDiplomaApplicationsAction: this.props.openDiplomaApplicationsAction,
        openProjectModal: this.props.openProjectModal,
        removeDiplomaAction: this.removeDiplomaAction,
        showLoading: this.props.projectListIsLoading,
      }}>
        <>
```

```
        <Header />
        {container}
        <FloatingButton onClick={() => this.create()}>+</FloatingButton>
        <DiplomasModal />
        <ApplicationListModal />
        <ProjectModal />
        {removeConfirmationModal}
      </>
    </DiplomasContext.Provider>
  )
 }
}

export default LecturerDiplomas
```

ДОДАТОК 3

Модуль викладача в системі управління дипломним проектуванням кафедри

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52_19Б 13-2

Аркушів 8

Київ – 2019

АНОТАЦІЯ

Інтерфейс створеного веб-додатку по управлінню дипломним проектуванням кафедри дозволяє користуватися системою з будь-якого електронного пристрою. Користувачі поділяються на три групи: студенти, викладачі та додатковий персонал. Система забезпечує ефективну взаємодію між ними в навчальному процесі.

Додаток було розроблено за допомогою платформи JetBrains PhpStorm 2018.3.3 з використанням мови Node.js для серверної сторони та мови Javascript з бібліотеками React та Redux для клієнтської.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до теми дипломної роботи, програма має назву «Система дипломним проектуванням кафедри».

Програма працює через браузер з будь-якого електронного пристрою та потребує доступу до мережі інтернет. Вона забезпечує практичність та швидкість в роботі.

Система була написана мовою Javascript з використанням бібліотек React та Redux для клієнтської сторони та програмної платформи Node.js для серверної.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний засіб покликаний покращити навчальний процес на кафедрі, а саме - забезпечити зручне та ефективне дипломне проектування. Це було реалізовано за допомогою наступного функціоналу:

– можливість входу в систему для кожного викладача;

– створення / редагування / видалення / перегляд тем та проектів для дипломних робіт;

– перегляд заявок від студентів на виконання дипломної роботи;

– можливість закріплення студента за дипломною роботою;

– контроль над статусом виконання дипломної роботи;

– перегляд загального графіку виконання дипломної роботи;

– перегляд загальної інформації про лабораторії, її напрямки та склад;

– можливість перегляду інформації в зручному вигляді з будь-якого електронного пристрою з доступом до мережі інтернет;

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Програмний продукт складається з клієнтської та серверної частини.

Для того щоб увійти в систему викладач спочатку повинен отримати дані для входу від додаткового персоналу.

Загальний принцип роботи додатку на прикладі модуля «Вчитель» такий:

1) вчитель входить в систему обираючи тип користувача «Вчитель»;
2) для перегляду інформації про лабораторії переходить на сторінку «Лабораторії»;
3) для керування дипломними роботами потрібно перейти на сторінку «Дипломні роботи».
4) створення теми дипломної роботи відбувається при натисненні на кнопку «+» на сторінці «Дипломні роботи»;
5) для вибору студента на дипломну роботу потрібно натиснути на «Заявки» та серед студентів які подавали заявки на дипломну роботу обрати потрібного натиснувши на «Прийняти»;
6) для керування процесом виконання потрібно натиснути на кнопку «Графік виконання»

При відкритті сторінок дані для відображення завантажуються одноразово: посилається запит на сервер, сервер посилає запит на базу даних повертаючи результат на клієнтську сторону, де в залежності від результату оновлюється користувацький інтерфейс.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для організації доступу до програмного продукту потрібно мати будь-який електронний пристрій.

Кінцевим користувачам для роботи з програмою потрібен браузер та доступ до мережі інтернет.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є:

— дані для входу перших користувачів, список існуючих лабораторій, груп;

Вихідними даними є:

— представлення інтерфейсу логічно згрупованого вхідними даними для управління списками студентів, груп, викладачів, лабораторій, дипломних робіт та проектів;

— графік виконання в табличній формі по кожній дипломній роботі.