ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-СЕРВІСУ ДЛЯ ПІДТРИМКИ ОРГАНІЗАЦІЙНИХ ПРОЦЕСІВ ДИПЛОМУВАННЯ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ REACT.JS**»

Виконав: студент 4 курсу, групи ПД – 42

спеціальності:

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Шовтенко В. Ю.

(прізвище та ініціали)

Керівник Залива В.В.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Київ-2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідува кафедри

Інженерії програмного забезпечення

Негоденко О.В.

“____” _____ 2023 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ РОБОТУ СТУДЕНТА

ШОВТЕНКА ВАЛЕНТИНА ЮРІЙОВИЧА

(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка web-сервісу для підтримки організаційних процесів дипломування з використанням бібліотеки React.JS»

Керівник роботи: Залива В. В., асистент кафедри ІПЗ

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26

2. Строк подання студентом роботи «1» червня 2023 року
3. Вхідні дані до роботи:
   3.1. Науково-технічна література, пов’язана з веб розробкою;
   3.2. Документація React.JS та Node.JS;
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно зробити):

4.1. Аналіз загальних положеннь побудови веб-сервісів;

4.2. Аналіз та вибір технологій і методів для розробки;

4.3. Архітектура: опис архітектури веб-сервісу та взаємодії між його складовими частинами.

4.4. Тестування та перевірка;

5. Перелік демонстраційного матеріалу:

5.1. Мета, об’єкт та предмет дослідження.

5.2. Задачі дипломної роботи.

5.3. Аналіз аналогів.

5.4. Вимоги до програмного забезпечення.

5.5. Програмні засоби реалізації.

5.6. Діаграма варіантів використання.

5.7. Діаграма діяльності.

5.8. Діаграма класів.

5.9. Екранні форми.

5.10. Апробація результатів дослідження.

5.11. Висновки.

6. Дата видачі завдання «17» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	23.03 – 25.03	Виконано
2	Визначення вимог	26.03 – 27.03	Виконано
3	Розробка архітектури серверної частини	28.03 – 04.04	Виконано
4	Розробка дизайну клієнтської частини	05.04 – 09.04	Виконано
5	Програмна реалізація веб-сервісу	10.04 – 28.04	Виконано

6	Вступ, висновки, реферат	29.04 – 08.05	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів.	09.05 – 16.05	Виконано
8	Попередній захист роботи	24.05	Виконано
9	Здача роботи	14.06	

Студент _____ Шовтенко В.Ю.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Залива В.В.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 61с., 18 рис., 2 табл., 11 джерел

Ключові слова: JavaScript, TypeScript, REST API, React.js, Node.js, MongoDB

Об'єкт дослідження – організаційні процеси дипломування. *Предмет дослідження* – веб-технології клієнтської та серверної частин при розробці сайтів.

Мета – підвищення ефективності та зручності процесу підтримки організаційних процесів дипломування. *Методи дослідження* – Аналіз науково-технічної літератури, аналіз вимог по веб-сервісу, проектування системи, розробка, тестування кінцевого продукту. Для реалізації поставленої задачі вирішено такі завдання:

1. Огляд та аналіз літературних джерел для створення веб-сервісів.
2. Аналіз програмного забезпечення, яке може бути використано для розробки.
3. Розробка архітектури додатку.
4. Створення додатку для підтримки організаційних процесів дипломування.
5. Тестування системи.

Галузь використання – освітний сектор, зокрема з університетами або іншими навчальними установами, які надають академічні ступені та дипломи. Практичне значення отриманих результатів: Розроблено веб-додаток для підвищення ефективності роботи процесів дипломування. Для розробки застосунку використовувалися інструменти Visual Studio Code, мови програмування JavaScript та TypeScript, з використанням фреймворків React.js та Express.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми та її актуальність: Дипломування є важливим етапом для студентів, який вимагає виконання багатьох організаційних процесів. Ці процеси включають в себе подачу заявок на дипломну роботу, вибір наукового керівника, здачу екзаменів та захист дипломної роботи. Розробка веб-сервісу, який підтримуватиме ці процеси, може значно полегшити життя студентів та організацію процесу дипломування в університетах.

Веб-сервіси стають все більш популярними, оскільки вони забезпечують легкий та швидкий доступ до інформації та послуг. Розробка веб-сервісу для дипломування може забезпечити студентам та викладачам легкий та зручний доступ до інформації, пов'язаної з процесом дипломування. Зростання популярності онлайн-навчання та дистанційного навчання викликає потребу віддаленої підтримки організаційних процесів.

Існуючі підходи до організації цього процесу часто мають низьку ефективність та не відповідають вимогам сучасності. Таким чином, розробка для підтримки організаційних процесів дипломування має великий потенціал для полегшення життя студентів та викладачів, покращення організації процесу дипломування.

Об’єктом дослідження є методи розробки систем для підтримки організаційних процесів дипломування. **Предметом дослідження** є веб-технології клієнтської та серверної частин при розробці сайтів. **Метою** є підвищення ефективності та зручності процесу підтримки організаційних процесів дипломування. Для реалізації поставленої мети потрібно вирішити наступні **завдання**:

1. Огляд та аналіз літературних джерел для створення веб-сервісів.
2. Огляд та аналіз програмного забезпечення, яке може бути використано для розробки.
3. Розробка архітектури додатку.

4. Створення додатку для підтримки організаційних процесів дипломування.
5. Тестування та вдосконалення;

Методика дослідження: Для розробки веб-сервісу використовувались інструменти Visual Studio Code, Postman, JavaScript, TypeScript в основі використовуючи принцип DRY. Також був проведений аналіз літературних джерел з веб розробки, проектування та розробки програмного забезпечення, проведенні тестування та аналізі результатів.

Наукова новизна полягає в наступному: створення великого, надійного веб-сервісу використовуючи бібліотеку React.JS. **Практична значущість результатів** складається з розроблених серверної та клієнтської частин веб-сервісу, який оптимізує процес організаційних процесів дипломування, для студентів та викладачів.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Аналіз загальних положенньь побудови веб-сервісів

1.1.1 Загальні відомості

Веб-програмування включає в себе процес створення сайтів і програм, які функціонують через мережу. Для цього розробляються спеціальні комп'ютерні програми, відомі як скрипти, які можна розділити на два типи: серверні і клієнтські. Сервер є комп'ютером, на якому розміщуються файли сайту, тоді як клієнт, або точніше кажучи, браузер, встановлений на персональному комп'ютері, виступає в ролі користувача, який взаємодіє зі стороною сервера, надсилаючи запити.

Одним із ключових елементів у веб-розробці є мова програмування, яка використовується для створення веб-додатків. Існує багато мов, які можуть бути використані як для розробки серверних, так і для клієнтських скриптів. Однак, деякі мови спеціалізуються лише на розробці серверної або клієнтської частини додатків.

Інтернет стрімко розвивається протягом останніх років, що призводить до збільшення кількості нових онлайн-сервісів. Починаючи з простих поштових сервісів і веб-сайтів, Інтернет перетворився на глобальну мережу з різноманітними сервісами, технологічними рішеннями, які стали місцем зустрічі практично всіх бізнес-організацій світу.

Також спостерігається зростання кількості компаній, які переходять до електронного бізнесу. Вони реєструють корпоративні домени, створюють власні веб-сайти, рекламують свої товари та послуги в Інтернеті, запускають онлайн-продажі.

1.1.2 Веб-сервіс (Вебслужба)

Веб-сервіс - це програмна система, яка надає функціональність для інтернет-застосунків за допомогою відкритих стандартів і протоколів передачі даних через мережу Інтернет. Веб-сервіси забезпечують зручний спосіб спілкування між різними програмними системами, що працюють на різних платформах та мовах програмування.

Веб-сервіси зазвичай забезпечують функції, які можна використовувати з будь-якої інтернет-прикладної програми, що має доступ до Інтернету. Наприклад, веб-сервіси можуть забезпечувати можливість отримання даних з бази даних, обробки даних, передачі електронних повідомлень та інші.

Веб-сервіси можуть бути розроблені на різних мовах програмування, таких як Java, PHP, Python, а також можуть використовувати різні протоколи передачі даних, такі як SOAP, REST, XML та інші.

1.1.3 Основні характеристики та функціональність програмного забезпечення

Програмне забезпечення для підтримки організаційних процесів дипломування має ряд характеристик та функціональності, що допомагають управляти процесом дипломування. Основні характеристики та функціональність такого програмного забезпечення можуть включати наступне:

- Збір та зберігання даних: програмне забезпечення повинне дозволяти збирати та зберігати різні дані, такі як особисті дані студента, дані про теми дипломних робіт та інше.
- Управління процесом дипломування: програмне забезпечення повинне забезпечувати можливість управління процесом дипломування, включаючи реєстрацію тем дипломних робіт, розподіл керівників.
- Керування календарем: програмне забезпечення повинне забезпечувати можливість керування календарем, щоб забезпечити планування термінів

процесів дипломування, включаючи терміни захисту дипломних робіт та екзаменаційну сесію.

- Онлайн-запис на захист дипломів: програмне забезпечення повинне забезпечувати можливість онлайн-запису студентів на захист дипломів, щоб зменшити навантаження на адміністраторів та забезпечити більш зручний та ефективний процес.
- Забезпечення безпеки та конфіденційності: програмне забезпечення повинне забезпечувати безпеку та конфіденційність даних, що містяться в системі, забезпечуючи контроль доступу та захист від несанкціонованого доступу.
- Інтеграція з іншими системами: програмне забезпечення повинне забезпечувати можливість інтеграції з іншими системами, такими як системи електронного навчання, електронної бібліотеки та іншими системами, щоб забезпечити єдиний інформаційний простір та зручний доступ до всієї необхідної інформації.

1.1.4 Рішення для заданої задачі

Локальні рішення

Локальні рішення, тобто програмне забезпечення, встановлене на комп'ютерах або сервері в межах організації, можуть підійти для підтримки організаційних процесів дипломування.

Такі рішення мають свої переваги, зокрема, повний контроль над даними та функціональністю, відсутність залежності від зовнішніх сервісів та можливість розширення функціональності за потребою.

Однак, вони також мають свої недоліки, такі як високі витрати на придбання та підтримку програмного забезпечення та обладнання, відповідальність за збереження та захист даних, а також складнощі з доступом до інформації з-за меж організації.

Отже, перед вибором локального рішення для підтримки організаційних процесів дипломування, необхідно обміркувати всі його переваги та недоліки і

порівняти їх з альтернативними рішеннями, такими як хмарні рішення або сервіси SaaS (Software as a Service).

SaaS (Software as a Service)

SaaS (Software as a Service) - це модель доставки програмного забезпечення, в якій користувачі отримують доступ до програм через Інтернет. У контексті підтримки організаційних процесів дипломування, SaaS може допомогти в таких аспектах:

- Швидкість і простота використання: SaaS може забезпечити швидкий та легкий доступ до програм, що дозволяє користувачам зосередитись на своїх задачах, а не на налаштуванні та підтримці інфраструктури.
- Масштабованість: SaaS дозволяє легко масштабувати ресурси в залежності від зростаючих потреб, що дозволяє організаціям швидко реагувати на змінні вимоги.
- Автоматизація: SaaS може допомогти автоматизувати багато рутинних завдань, що дозволяє звільнити час та ресурси для інших задач.
- Вартість: SaaS може бути менш витратним, ніж локальне програмне забезпечення, оскільки користувачі платять тільки за користування програмним забезпеченням, а не за обладнання та підтримку.
- Безпека: SaaS може забезпечити високий рівень захисту даних, оскільки постачальники SaaS зазвичай забезпечують більш ефективні механізми захисту даних, ніж ті, які зазвичай знаходяться в організаціях.

Хоча SaaS має декілька переваг, є деякі потенційні мінуси, які варто врахувати при використанні цієї моделі доставки програмного забезпечення для підтримки організаційних процесів дипломування:

- Залежність від постачальника: Користувачі SaaS повністю залежать від постачальника програмного забезпечення, і якщо постачальник зазнає проблем зі своїми службами, то користувачі можуть стати без доступу до своїх даних і програм.

- Обмеження функціональності: Оскільки користувачі SaaS не мають прямого доступу до програмного коду, вони не можуть внести власні зміни у програмне забезпечення, що може призвести до обмеження функціональності та можливостей користувачів.
- Відсутність контролю над даними: Користувачі SaaS мають меншу контроль над своїми даними, оскільки вони зберігаються на серверах постачальника, що може викликати проблеми з безпекою даних та конфіденційністю.
- Вартість: Хоча SaaS може бути менш витратним, ніж локальне програмне забезпечення, вартість може збільшуватися з часом, особливо при збільшенні обсягу користувачів або розширенні функціональності.
- Інтеграція з іншими системами: Інтеграція SaaS з іншими системами може бути складнішою, ніж при використанні локального програмного забезпечення, оскільки SaaS зазвичай має обмежені можливості взаємодії з іншими програмами і системами.

Отже, підсумовуючи, SaaS може бути ефективним рішенням для підтримки організаційних процесів дипломування, оскільки він може забезпечити швидкий та легкий доступ до програмного забезпечення, а також зменшити витрати на IT-інфраструктуру та обслуговування. Однак, перед вибором SaaS необхідно ретельно проаналізувати його переваги та мінуси, враховуючи специфіку потреб користувачів та їхніх організацій. У кінці кінців, вибір між SaaS та локальним програмним забезпеченням повинен бути здійснений на основі конкретних потреб та можливостей організації.

CRM

CRM (Customer Relationship Management) - це програмне забезпечення для управління взаємодією з клієнтами, що зазвичай використовується в маркетингових та продажних департаментах. Хоча він не був розроблений спеціально для підтримки організаційних процесів дипломування, але може бути використаний для деяких аспектів такого процесу.

CRM може бути і хмарним рішенням, і локальним програмним забезпеченням. Хмарні CRM-системи, такі як Salesforce, Microsoft Dynamics 365, HubSpot CRM та інші, зазвичай надаються як послуги (SaaS), тобто користувачі можуть отримати доступ до них через Інтернет. Це означає, що всі дані зберігаються на серверах провайдера, а користувачі можуть отримати доступ до системи з будь-якого місця за допомогою Інтернету.

У той же час, локальні CRM-системи встановлюються безпосередньо на комп'ютери або сервери організації, що означає, що всі дані зберігаються в локальних базах даних, і користувачі можуть отримати доступ до системи тільки з пристроїв, підключених до мережі організації.

CRM може допомогти зберігати дані про студентів та їх контакти, такі як адреса, номер телефону та електронна пошта. Він також може забезпечити функціональність збору даних про те, які дипломні роботи обрав студент, де він знаходиться у процесі написання своєї роботи, та де йому потрібна допомога.

Однак, для повного покриття організаційних процесів дипломування необхідне спеціалізоване програмне забезпечення, яке має функціональність для управління реєстрацією, плануванням, контролю термінів та етапів виконання дипломної роботи, а також забезпечення обміну інформацією між всіма учасниками процесу.

REST API

REST (Representational State Transfer) - це архітектурний стиль для створення веб-сервісів, який передбачає використання стандартних HTTP-методів для доступу до ресурсів за допомогою URL-адрес.

За допомогою REST API можна розробити серверну частину, яка буде надавати доступ до різних функцій та можливостей, таких як збереження даних про студентів, надання інформації про викладачів та інші. Саме цей варіант було обрано при розробці.

2 ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ ПРОГАМНОЇ РЕАЛІЗАЦІЇ

2.1 Мова програмування

2.2.1 Клієнт

Для розробки клієнтських додатків використовуючи бібліотеку React.js є 2 основних варіанти для мов програмування:

- JavaScript є найпопулярнішою мовою програмування для веб-розробки. Вона є стандартом для взаємодії з користувачем у веб-браузерах та використовується для розробки веб-додатків та мобільних додатків.
- TypeScript є розширенням JavaScript, яке надає можливість використовувати статичний типізм в розробці клієнтських додатків, зокрема з React. Використання TypeScript разом з React допомагає виявляти більше помилок на етапі компіляції, полегшує роботу зі складними даними та забезпечує більшу безпеку в процесі розробки.

Було прийнято рішення використовувати TypeScript, тому що він більш надійний ніж JavaScript. Розробка буде тривати довше, але кінцевий результат буде легче підтримувати з часом.

2.2.2 Сервер

При виконанні цієї роботи були розглянуті наступні мови програмування для розробки серверу:

- Java є однією з найпопулярніших мов програмування для серверів. Вона має велику екосистему, багато бібліотек та фреймворків, що дозволяє легко створювати високопродуктивні серверні додатки.
- JavaScript — універсальна мова програмування, яка використовується не лише для розробки фронт-енду, але також може бути використана на серверному боці для створення бекенд-додатків. Традиційно, JavaScript був пов'язаний з

розробкою фронт-енду, де він працює в веб-браузерах, забезпечуючи динамічну взаємодію та користувацький досвід на веб-сайтах та веб-додатках. Однак, з введенням платформ JavaScript на серверному боці, JavaScript тепер може бути використаний також для написання коду на серверній стороні.

- Ruby — дуже простою та зручною мовою програмування. Її фреймворки, такі як Ruby on Rails, роблять її дуже ефективною для швидкої розробки веб-додатків.
- PHP є однією з найпопулярніших мов програмування для серверів. Вона використовується для створення веб-додатків та динамічних сторінок в Інтернеті. У PHP також є велика кількість фреймворків, таких як Laravel та Symfony.
- Go — мова програмування, розробленою Google, для створення високопродуктивних серверних додатків. Вона має простий та зрозумілий синтаксис та має підтримку багатьох фреймворків, таких як Gin та Echo.

JavaScript дозволяє прискорити розробку порівняно з іншими мовами, не зважаючи на відсутність строгої типізації. Типізація важлива при роботі з базою даних, але використовуючи бібліотеку mongoose все одно сворюється чітка модель для всіх елементів.Тому було прийнято рішення використовувати JavaScript та Node.js.

2.2 Node.js

Node.js є серверним середовищем JavaScript. Вона базується на рушії V8, який також використовується в браузерах Chrome та Opera. Node.js є дуже ефективним для створення високопродуктивних додатків (тому що він написаний на C++) з багатьма з'єднаннями, таких як чат-боти та мережеві додатки.

V8 відомий своїм швидким і ефективним виконанням JavaScript, що робить його одним із найпоширеніших двигунів JavaScript. Він використовує компіляцію Just-In-Time (JIT) для оптимізації коду JavaScript для виконання, що забезпечує високу продуктивність і низьку затримку. V8 також підтримує сучасні функції мови JavaScript і надає багатий набір інструментів для налагодження та профілювання для розробників.

V8 зосереджений на продуктивності та оптимізації, з постійними розробками та вдосконаленнями для підвищення швидкості виконання JavaScript і зменшення використання пам’яті. Він постійно оновлюється для підтримки останніх специфікацій мови JavaScript і забезпечує міцну основу для створення високопродуктивних програм JavaScript.

Окрім використання у веб-браузерах і Node.js, V8 також використовується в інших програмах і вбудованих системах, де потрібне виконання JavaScript. Його природа з відкритим вихідним кодом дозволяє робити внески спільноти та співпрацювати, що робить його популярним вибором для багатьох розробників і програм JavaScript.

2.3 Express.js

Express.js — це мінімалістичний, швидкий і схожий на Sinatra серверний фреймворк Node.js, який надає надійні функції та інструменти для розробки масштабованих серверних програм. Він дає вам систему маршрутизації та спрощені функції для розширення інфраструктури шляхом розробки більш потужних компонентів і частин залежно від випадків використання вашої програми.

Фреймворк надає набір інструментів, які допомагають у створенні та розгортанні великомасштабних корпоративних веб-додатків. Цей набір включає інструменти для обробки HTTP-запитів і відповідей, маршрутизації та проміжного програмного забезпечення.

Крім того, фреймворк надає інструмент командного рядка (CLI) під назвою Node Package Manager (NPM), який дозволяє розробникам отримувати готові пакети. Це допомагає уникнути повторення коду і сприяє принципу "Не повторюйся" (DRY).

Принцип DRY спрямований на зменшення дублювання коду шляхом застосування абстракцій або нормалізації даних. Це допомагає уникнути надмірності і полегшує розробку програмного забезпечення.

Express.js використовується для багатьох речей в екосистемі JavaScript/Node.js — ви можете розробляти програми, кінцеві точки API, системи маршрутизації та фреймворки.

Нижче наведено список лише кількох типів програм, які можна створити за допомогою Express.js:

Односторінкові програми

Односторінкові програми (SPA) — це сучасний підхід до розробки програм, за якого вся програма направляється на одну сторінку індексу. Express.js — це чудова структура для створення API, який об'єднує ці програми SPA та послідовно обслуговує дані. Деякі приклади односторінкових програм: Gmail, Google Maps, Airbnb, Netflix, Pinterest, Paypal та багато інших. Компанії використовують SPA для створення гнучкого, масштабованого досвіду.

- **Інструменти для співпраці в реальному часі**

Інструменти для співпраці створені для того, щоб спростити бізнес і щоденну співпрацю, а за допомогою Express.js ви можете з легкістю розробляти програми для спільної роботи та мережеві програми в режимі реального часу.

Крім того, фреймворк використовується для розробки додатків у реальному часі, таких як додатки для чату та інформаційної панелі, де стає простою інтеграція WebSocket у фреймворк.

Express.js керує частиною процесу маршрутизації та проміжного програмного забезпечення, дозволяючи розробникам зосередитися на життєво важливій бізнес-логіці цих функцій реального часу під час розробки живих інструментів для спільної роботи.

Потокові програми

Програми потокового передавання в реальному часі, такі як Netflix, є складними та мають багато рівнів потоків даних. Щоб розробити таку програму, вам потрібна надійна структура для ефективної обробки асинхронних потоків даних. Це ідеальний фреймворк для створення та розгортання корпоративних і масштабованих потокових програм.

Програми Fintech

Fintech — це комп'ютерна програма та інша технології, які використовується для підтримки або надання банківських і фінансових послуг. Створення фінтех-додатків зараз є галузевою тенденцією, і Express.js є основою вибору для створення високомасштабованих фінтех-додатків.

Система маршрутизації

Express.js має вбудовану та потужну систему маршрутизації, яка є надійною і ефективною. Ця система дозволяє вашому додатку відповідати на запити клієнта через специфічні ендпоінти.

У Express.js ви можете організувати свою систему маршрутизації шляхом використання екземпляра маршрутизатора фреймворку. Це дозволяє вам логічно розділити ваші маршрути на керовані файли. Використання системи маршрутизації Express.js сприяє кращій структурі програми, дозволяючи групувати різні маршрути в одному місці, наприклад, в окремій папці або каталозі.

Проміжне програмне забезпечення

Express.js — це фреймворк, що містить серію проміжного програмного забезпечення для створення безперебійного процесу розробки. Проміжне програмне забезпечення — це коди, які виконуються до того, як запит HTTP досягне обробника маршруту або до того, як клієнт отримає відповідь, що дає структурі можливість запускати типовий сценарій до або після запиту клієнта.

За допомогою проміжного програмного забезпечення розробники можуть підключати сценарії, щоб перехопити потік програми, наприклад, розробники можуть використовувати проміжне програмне забезпечення, щоб перевірити, чи вдалось користувачу авторизуватися чи вийти.

2.4 База даних

При виборі бази даних є 2 варіанти, SQL та NoSQL. Між ними є п'ять практичних відмінностей:

Мова

SQL існує вже понад 40 років, тому він впізнаваний, задокументований і широко використовується. Безпечний і універсальний, він особливо добре підходить для складних запитів. Однак SQL обмежує користувача працювати в рамках попередньо визначеної табличної схеми, тому потрібно приділяти більше уваги організації та розумінню даних перед їх використанням.

Динамічні схеми баз даних NoSQL дозволяють представляти альтернативні структури, часто поруч одна з одною, заохочуючи більшу гнучкість. Менше уваги приділяється плануванню, більше свободи при додаванні нових атрибутів або полів, а також можливість різноманітного синтаксису в базах даних. Однак мовам NoSQL як групі не вистачає стандартного інтерфейсу, який надає SQL, тому складніші запити може бути важко виконати.

Хоча існує багато діалектів SQL, усі мають загальний синтаксис і майже ідентичну граматику. Під час запитів до реляційних баз даних вільне володіння однією мовою означає знання більшості інших. З іншого боку, між мовами NoSQL дуже мало узгодженості, оскільки вони стосуються різноманітного набору непов’язаних технологій. Багато баз даних NoSQL мають унікальну мову обробки даних, обмежену певними структурами та можливостями.

Масштабованість

Більшість баз даних SQL можна масштабувати вертикально за рахунок збільшення обчислювальної потужності наявного обладнання. Бази даних NoSQL використовують архітектуру головний-підлеглий, яка краще масштабується горизонтально, з додатковими серверами або вузлами. Це корисні узагальнення, але важливо зазначити:

Бази даних SQL також можна масштабувати горизонтально, хоча логіка шардингу або розділення часто залежить від користувача і не підтримується належним чином.

Технології NoSQL різноманітні, і хоча багато з них покладаються на архітектуру «головний-підлеглий», також існують варіанти вертикального масштабування.

Економія, отримана за допомогою більш ефективних структур даних, може перевищити відмінності в масштабованості; найважливіше — зрозуміти варіант використання та відповідно спланувати.

Структура

Схеми бази даних SQL завжди представляють реляційні табличні дані з правилами узгодженості та цілісності. Вони містять таблиці зі стовпцями (атрибути) і рядками (записи), а ключі мають обмежені логічні зв'язки.

Бази даних NoSQL не повинні дотримуватися цього формату, але зазвичай підпадають під одну з чотирьох широких категорій:

- Бази даних, орієнтовані на стовпці, транспонують РСУБД, орієнтовані на рядки, що дозволяє ефективно зберігати дані великого розміру та окремі записи з різними атрибутами.
- Сховища ключ-значення — це словники, які отримують доступ до різноманітних об'єктів із унікальним для кожного ключем.
- Сховища документів зберігають напівструктуровані дані: об'єкти, які містять всю власну релевантну інформацію та можуть повністю відрізнятися один від одного.
- Графічні бази даних додають концепцію зв'язків (прямих зв'язків між об'єктами) до документів, що дозволяє швидко обходити сильно пов'язані набори даних.

Властивості

На високому рівні SQL і NoSQL відповідають окремим правилам вирішення транзакцій. РСУБД повинні демонструвати чотири властивості «ACID»:

- **A**tomicity. Це означає, що всі транзакції повинні бути успішними або повністю невдалими. Вони не можуть бути частково повними, навіть у разі збою системи.
- **C**onsistency означає, що на кожному кроці база даних дотримується інваріантів: правил, які підтверджують і запобігають пошкодженню.

- **I**solation запобігає впливу одночасних транзакцій одна на одну. Трансакції повинні призводити до такого ж кінцевого стану, як і якщо вони виконуються послідовно, навіть якщо вони виконуються паралельно.
- **D**urability робить транзакції остаточними. Навіть збій системи не може відкотити наслідки успішної транзакції.

Технології NoSQL дотримуються теореми «CAP», яка говорить, що в будь-якій розподіленій базі даних можуть бути гарантовані тільки дві з наступних властивостей одночасно:

- **C**onsistency: кожен запит отримує останній результат або помилку. (Зверніть увагу, що це відрізняється від ACID)
- **A**vailability: кожен запит має результат без помилок, незалежно від того, наскільки останній цей результат.
- **P**artition tolerance: будь-які затримки або втрати між вузлами не призведуть до переривання роботи системи.

Підтримка та спільноти

Бази даних SQL представляють величезні спільноти, стабільні кодові бази та перевірені стандарти. Безліч прикладів опубліковано в Інтернеті, і експерти доступні для підтримки тих, хто новачок у програмуванні реляційних даних.

Технології NoSQL швидко впроваджуються, але спільноти залишаються меншими та більш роз'єднаними. Однак багато мов SQL є пропрієтарними або пов'язані з великими окремими постачальниками, тоді як спільноти NoSQL отримують переваги від відкритих систем і узгоджених зобов'язань щодо адаптації користувачів.

SQL доступний для більшості основних платформ, від операційних систем до архітектур і мов програмування. Сумісність для NoSQL значно відрізняється, і залежності потрібно досліджувати ретельніше.

Чому саме MongoDB?

MongoDB заснована на документах, оскільки вона є базою даних типу NoSQL. Замість використання реляційної структури даних, MongoDB зберігає дані

у вигляді документів. Це надає MongoDB велику гнучкість і здатність адаптуватися до різних вимог сучасного бізнесу.

MongoDB підтримує спеціальні запити, такі як пошук за полями, запити діапазону і пошук за регулярними виразами. Ви можете створювати запити для вибіркового виводу певних полів з документів.

Індексування в MongoDB дозволяє покращити продуктивність пошуку шляхом створення індексів для будь-якого поля в документі.

MongoDB підтримує реплікацію для забезпечення високої доступності. Набір реплік складається з кількох серверів MongoDB, де кожен сервер може виступати як основний або вторинний. Основний сервер обробляє всі операції читання/запису, а вторинні сервери зберігають копії даних. У разі відмови основного сервера, автоматично вибирається новий основний сервер з вторинних реплік.

MongoDB використовує шардування для горизонтального масштабування і балансування навантаження. Це дозволяє розподіляти дані між різними серверами MongoDB, що дозволяє підтримувати високу продуктивність та доступність системи, навіть у разі збоїв обладнання.

2.5 Mongoose

Mongoose - це бібліотека для моделювання об'єктів (ODM) для Node.js, яка надає зручний та гнучкий спосіб взаємодії з MongoDB, популярною NoSQL базою даних. Вона діє як рівень абстракції між Node.js та MongoDB, дозволяючи розробникам працювати з MongoDB у більш знайомому та об'єктно-орієнтованому стилі.

Однією з основних функцій Mongoose є підтримка схем даних. Схеми визначають структуру даних, які будуть зберігатися в MongoDB, включаючи поля, типи даних та правила перевірки. Розробники можуть визначати схеми, використовуючи API визначення схем Mongoose, яке надає велику кількість типів

даних, таких як рядки, числа, дати та масиви, а також опції для визначення власної логіки перевірки.

Mongoose також підтримує Middleware, які є функціями, які можуть виконуватися перед або після певних операцій, таких як збереження, оновлення або видалення документів. Це дозволяє розробникам додавати власну логіку на різних етапах життєвого циклу даних, таку як маніпулювання даними, аутентифікація або логування. Middleware може бути визначено на рівні схеми або на глобальному рівні, що надає гнучкість в обробці різних операцій та сценаріїв.

Крім того, Mongoose надає багатий набір методів запитів, які дозволяють розробникам легко взаємодіяти з даними MongoDB. Це включає підтримку розширених параметрів запитів, таких як фільтрація, сортування та розбиття на сторінки. Mongoose також підтримує заповнення, що дозволяє розробникам посилатися на документи в інших колекціях і автоматично отримувати посилання на дані, полегшуючи роботу з пов’язаними даними в MongoDB.

Mongoose також містить такі функції, як підтримка плагінів, які є багаторазовими частинами функціональності, які можна легко додати до схем, забезпечуючи додаткові функції, такі як часові мітки, версії або геопросторове індексування. Mongoose має велику й активну спільноту розробників, і він широко використовується в багатьох програмах Node.js. Це означає, що доступні численні ресурси, зокрема документація, навчальні посібники та форуми, що полегшує пошук підтримки та вирішення типових проблем.

Загалом Mongoose спрощує роботу з MongoDB у програмах Node.js, надаючи вищий рівень абстракції, дозволяючи розробникам працювати з даними більш звичним та об’єктно-орієнтованим способом. Його підтримка схем даних, перевірки, проміжного програмного забезпечення, запитів і плагінів робить його потужним і популярним вибором серед розробників для створення додатків із підтримкою MongoDB у Node.js.

2.6 Socket.io

Socket.IO - це бібліотека JavaScript, яка забезпечує комунікацію у реальному часі між браузерами та серверами. Вона дозволяє бідирекційну комунікацію, забезпечуючи передачу даних в реальному часі та подійно-орієнтовану взаємодію між клієнтами та серверами.

Socket.IO використовує протокол WebSockets, який дозволяє постійну, повнодуплексну комунікацію через одне з'єднання між клієнтом (зазвичай веб-браузером) та сервером. Це дозволяє тримати зв’язок без необхідності в частих HTTP-запитах та відповідях, що може бути неефективним для деяких випадків використання, таких як чат-додатки, ігри та спільне редагування.

Socket.IO підтримує кілька платформ, включаючи веб-брузери, сервери Node.js та інші платформи, що робить його універсальним для різних веб- та мобільних додатків. Він надає простий API для відправки та отримання подій, що дозволяє розробникам створювати додатки в реальному часі з легкістю.

Однією з ключових особливостей Socket.IO є його здатність автоматично обробляти резервні варіанти та адаптуватися до різних мережевих середовищ. Він може використовувати різні види сигналів, такі як WebSockets, довге опитування (long polling) та опитування (polling), залежно від доступності та можливостей клієнта та сервера. Це робить його стійким до різних умов мережі та забезпечує безперебійний зв’язок у складних умовах.

Socket.IO також надає додаткові функції, такі як спілкування на основі кімнати, що дозволяє групувати клієнтів у віртуальні кімнати для цільового обміну повідомленнями, а також підтримку спеціальних подій і серіалізації даних.

Загалом, Socket.IO — це популярна та широко використовувана бібліотека для створення програм реального часу, які вимагають ефективного та масштабованого зв’язку між клієнтами та серверами. Його гнучкість, простота використання та підтримка багатьох платформ роблять його потужним інструментом для розробників для створення програм у режимі реального часу для різних випадків використання.

2.7 React.js

Реакт.js, часто називається просто Реакт, - це відкрита бібліотека JavaScript для побудови користувацьких інтерфейсів (UI) для веб-додатків. Вона була розроблена та підтримується Facebook та співтовариством розробників. Реакт.js використовує компонентну архітектуру, що дозволяє розробникам створювати повторно використовувані компоненти користувацького інтерфейсу, які можна складати для побудови складних UI.

Реакт.js надає декларативний підхід до побудови користувацьких інтерфейсів, де розробники описують бажаний стан UI, і Реакт відповідає за оновлення фактичного стану UI на основі змін у цьому стані. Це дозволяє здійснювати ефективні та оптимізовані оновлення UI, що призводить до високопродуктивних веб-додатків. Деякі основні особливості React.js включають:

- Архітектура на основі компонентів: React дозволяє розробникам створювати повторно використовувані компоненти користувацького інтерфейсу, які можна комбінувати разом для побудови складних UI.
- Віртуальний DOM: React використовує віртуальне представлення фактичного DOM (Document Object Model), що дозволяє здійснювати ефективні оновлення UI без прямої маніпуляції фактичним DOM, що призводить до покращення продуктивності.
- JSX: React використовує JSX (JavaScript XML), розширення синтаксису, яке дозволяє розробникам писати код, схожий на HTML, в JavaScript, що спрощує визначення компонентів UI у декларативний спосіб.
- Односторонній потік даних: React слідує односторонньому потоку даних, де дані рухаються від батьківських компонентів до дочірніх компонентів, що спрощує розуміння того, як дані змінюються в додатку.
- Однонаправлені оновлення: React забезпечує однонаправлений потік даних, де зміни даних розповсюджуються від батьківських компонентів до дочірніх

компонентів, а оновлення передаються вниз по дереву компонентів. Це спрощує розуміння та налагодження змін даних в додатку.

- Багатогранний екосистема: React має велику та жваву екосистему, з великою кількістю бібліотек та інструментів для побудови UI, таких як Redux для керування станом, React Router для роботи з маршрутизацією та Axios для здійснення викликів API, серед багатьох інших.

React.js широко використовується в спільноті веб-розробників через його продуктивність, гнучкість та можливості повторного використання. Він часто використовується для створення сучасних веб-додатків, включаючи односторінкові додатки (SPA), прогресивні веб-додатки (PWA) та мобільні додатки з використанням технологій, таких як React Native.

в React.js 16.8 була введена нова концепція побудови компонентів за допомогою функцій, які називаються функціональними компонентами (function components). За допомогою них можна створювати компоненти без використання класів.

Функціональні компоненти в React.js не мають методів життєвого циклу, які мають класові компоненти. Проте, з введенням React Hooks, з'явилася можливість використовувати функціональні компоненти зі спеціальними хуками, які надають функціональним компонентам схожу функціональність з класовими компонентами.

- useState: Хук useState дозволяє компонентам зберігати внутрішні стани. Він приймає початкове значення стану і повертає масив з двома елементами: поточне значення стану і функцію для його оновлення. При зміні стану компонент автоматично оновлюється і заново рендериться.
- useEffect: Хук useEffect дозволяє виконувати певні дії під час рендерингу компонентів. Він приймає два аргументи: функцію, яка буде виконуватися, і залежності (масив значень), від яких залежить ця функція. useEffect виконується після кожного рендерингу компонента або при зміні залежностей.

- useContext: Хук useContext дозволяє отримати доступ до значення контексту, встановленого вище в дереві компонентів.
- useReducer: Хук useReducer дозволяє використовувати редуктори для управління станом компонента. Він подібний до useState, але надає більш потужні можливості для складних станів, що потребують багатьох залежностей.
- useMemo: Хук useMemo дозволяє кешувати обчислені значення, залежні від вхідних даних. Він приймає функцію і масив залежностей, і повертає кешоване значення, яке автоматично оновлюється, коли одна зі залежностей змінюється.
- useCallback: Хук useCallback дозволяє кешувати мемоізовані версії обробників подій, що передаються вниз по компонентам. Він приймає функцію і масив залежностей, і повертає кешовану версію функції, яка автоматично оновлюється, коли одна зі залежностей змінюється.

Це лише кілька основних хуків, які надають функціональним компонентам можливість виконувати певні дії під час рендерингу і зберігати стани. Вони дозволяють створювати потужні компоненти, не використовуючи класи і методи життєвого циклу, які були характерні для старих версій React.js.

2.8 Redux

Redux - це відкрита бібліотека JavaScript, яка часто використовується разом з React, популярною бібліотекою користувацького інтерфейсу (UI), для керування станом додатків. Вона слідує принципам Flux, патерну проектування для побудови масштабованих і підтримуваних додатків, і надає передбачуваний механізм керування станом для організованого керування станом додатка в централізованому вигляді.

На своєму корінні, Redux працює з єдиним незмінним деревом стану, яке представляє весь стан додатка. Це дерево стану зберігається в одному магазині (store), який відповідає за керування станом та відправку дій (actions) для оновлення стану. Дії - це прості об'єкти JavaScript, які представляють зміни в стані і

відправляються компонентами або іншими частинами додатка, щоб викликати оновлення. Редуктори (reducers) - це чисті функції, які приймають поточний стан і дію (action) в якості вводу та повертають новий стан на основі дії. Вони відповідають за обробку дій та оновлення стану за незмінним принципом.

Одним з ключових концепцій Redux є однонапрямковий потік даних, де компоненти в інтерфейсі користувача відправляють дії, які потім обробляються редукторами для оновлення стану в магазині, і оновлений стан повертається до компонентів для повторного відтворення інтерфейсу користувача. Це робить керування станом в Redux передбачуваним і допомагає в розладці та розумінні змін стану з часом.

Redux також підтримує проміжний шар (middleware), який є функціями, що можуть перехоплювати дії перед тим, як вони досягнуть редукторів, дозволяючи здійснювати додаткову обробку, таку як асинхронні запити, ведення журналу та інше. Це робить Redux дуже розширюваним та гнучким для обробки складної логіки додатка.

Redux став популярним вибором для керування станом у великомасштабних і складних додатках React, оскільки він забезпечує єдиний та надійний стан, допомагає в розділенні відповідальностей та дозволяє легке відлагодження та тестування. Однак, імплементація потребує більше часу та досвіду. Через це, інші бібліотеки керування станом, можуть бути кращим рішенням для менших додатків або простих випадків використання.

2.9 Axios

Axios - популярна бібліотека JavaScript, яка використовується для здійснення HTTP-запитів з веб-переглядача або середовища Node.js. Вона надає простий і зручний інтерфейс для асинхронних HTTP-запитів, обробки відповідей та виконання загальних задач, таких як налаштування заголовків, робота з тайм-аутами, та відміна запитів. Axios широко використовується в клієнтських і серверних додатках JavaScript для отримання даних через виклик API.

2.10 Webpack

Webpack — популярний модульний збірник із відкритим кодом для програм JavaScript. Він широко використовується в сучасній веб-розробці для об’єднання та оптимізації зовнішніх ресурсів, таких як JavaScript, CSS і зображення.

Webpack дотримується модульного підходу, це дозволяє розробникам писати код використовуючи різні модулі та залежності, а потім об’єднувати їх в один або кілька файлів, залежно від конфігурації. Такий підхід допомагає зменшити кількість мережевих запитів і підвищити продуктивність веб-додатків.

2.11 Babel

Babel - це популярний відкритий компілятор JavaScript, який дозволяє розробникам писати сучасний код JavaScript, який може бути трансформований та використаний в старіших браузерах або середовищах, які не підтримують останній синтаксис та функції JavaScript.

Babel можна налаштувати для перетворення останніх версій JavaScript (ES2015, ES2016, ES2017 тощо) в більш широко підтримувану версію JavaScript, яка може працювати на більшому діапазоні браузерів та платформ.

Крім того, Babel також може трансформувати код, написаний в JSX (розширенні синтаксису для JavaScript, який використовується React), в стандартний JavaScript, який може бути зрозумілий всіма браузерами.

Babel став невід'ємним інструментом для розробників, які працюють над веб-застосунками та бібліотеками, оскільки це дозволяє їм використовувати останні можливості JavaScript, забезпечуючи при цьому безперебійну роботу їх коду на широкому спектрі пристроїв та платформ.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА ТЕСТУВАННЯ

1.1 Діаграма використання

Спочатку була розроблена діграма використання застосутку по функціональним вимогам. Діаграма використання (Use case diagram) - це один із типів UML-діаграм, який використовується для моделювання функціональної поведінки системи або програмного продукту з точки зору її користувачів або зовнішніх систем.

Діаграма використання допомагає описати, які дії можуть виконувати користувачі системи та які результати вони можуть очікувати. Вона містить в собі взаємодію між користувачем та системою в рамках певних сценаріїв використання.

У діаграмі використання можуть бути зображені такі елементи як актори, сценарії використання (use cases), системні компоненти, залежності між сценаріями та компонентами.

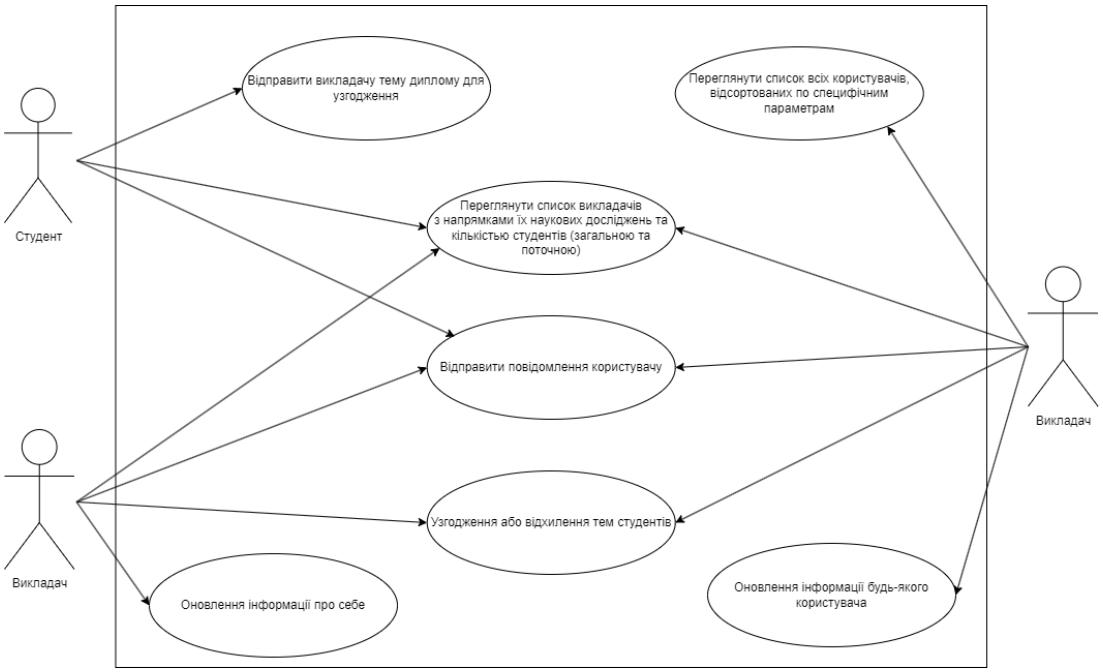


Рисунок 3.1 – Діаграма використання

Діаграма використання є важливим інструментом для розуміння та спілкування між розробниками програмного забезпечення, замовниками та іншими учасниками проекту. Вона може допомогти забезпечити взаєморозуміння між усіма зацікавленими сторонами та допомогти визначити необхідні функціональні вимоги до системи.

3.2 Діаграма класів

Наступним кроком була розробка діаграми класів, яка використовується для моделювання структури об'єктно-орієнтованої частини застосунку.

У діаграмі класів зображуються класи, що є основними будівельними блоками ООП, їх атрибути та методи, взаємозв'язки між класами та інші взаємодії між ними. Класи зображаються у вигляді прямокутників зі списком атрибутів та методів, а зв'язки між класами зображаються за допомогою ліній зі стрілками.

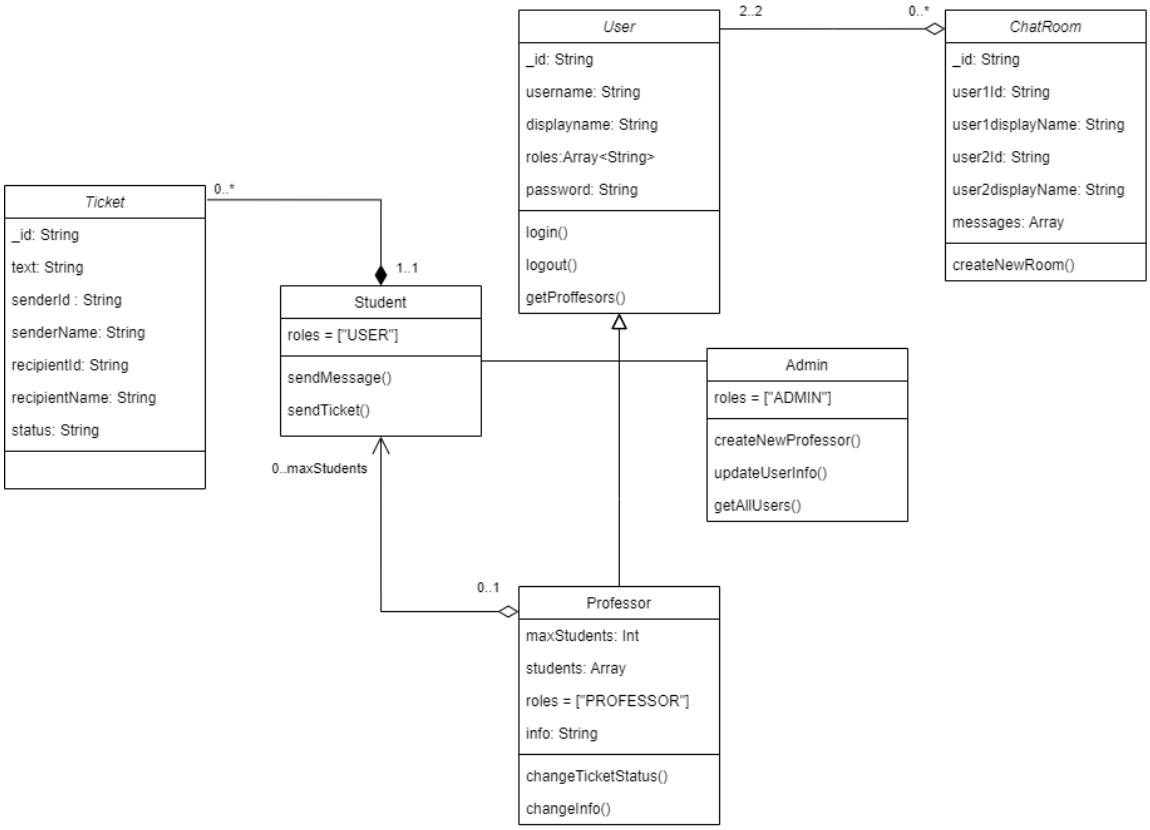


Рисунок 3.2 – Діаграма класів

Діаграма класів є важливим інструментом для аналізу та проектування програмних систем. Вона дозволяє описати структуру системи та відношення між її компонентами, що може допомогти визначити основні компоненти та їх функціональні залежності.

3.3 Необхідні програми

- Node.js

Для встановлення Node.js потрібно перейти на офіційний сайт (nodejs.org), завантажити останню стабільну версію для вашої операційної системи. Після завантаження запустіть інсталятор Node.js та слідуйте інструкціям на екрані. Після успішної установки Node.js ви можете перевірити, чи встановлено все правильно, запустивши термінал та ввівши «node -v».

- Будь який редактор коду. Персональний вибір: Visual Studio Code.

3.4 База даних

Була підключена віддалена база даних MongoDB, через «MongoDB Atlas». MongoDB Atlas - це повністю керований хмарний сервіс баз даних, який пропонується компанією MongoDB Inc. Він дозволяє користувачам легко розгортати, керувати та масштабувати бази даних MongoDB в хмарі.

За допомогою MongoDB Atlas користувачі можуть легко налаштовувати та керувати кластерами MongoDB, автоматизувати рутинні завдання та моніторити продуктивність своїх баз даних за допомогою зручної інтерфейсної панелі. Сервіс надає такі можливості, як автоматичне масштабування, резервне копіювання та відновлення даних, а також моніторинг продуктивності в режимі реального часу.

Один з головних переваг використання MongoDB Atlas полягає в тому, що користувачі можуть розгорнути свої бази даних в ультра-надійному та глобально

розподіленому середовищі. MongoDB Atlas пропонує різноманітні варіанти розгортання в хмарних сервісах провайдерів, таких як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure.

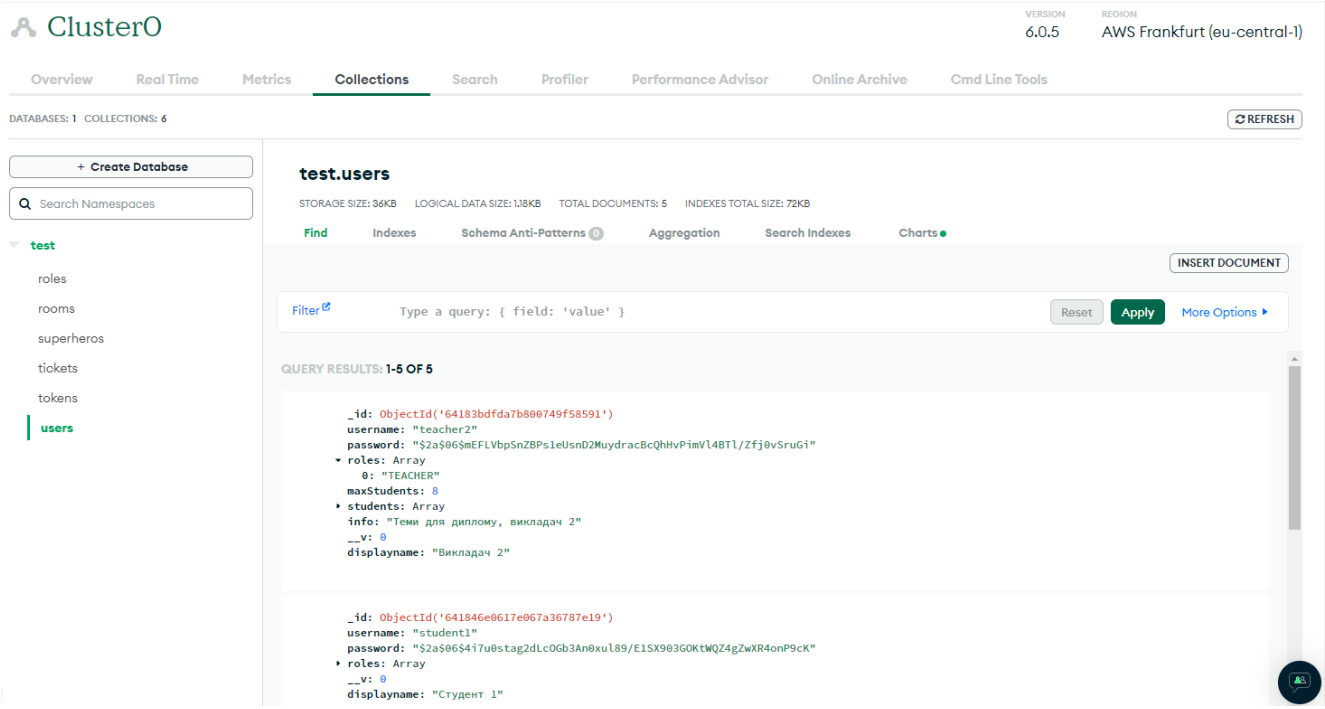


Рисунок 3.3 – Інтерфейс бази даних у MongoDB Atlas

3.5 Створення дизайну

Для створення дизайну було використано MaterializeCSS та Material UI.

Material Design – розроблена та спроєктована Google, вона поєднує традиційні принципи дизайну разом із сучасними інноваціями та технологіями. Метою Google є створення комплексної системи дизайну, яка забезпечує узгоджену взаємодію з користувачем у всіх їхніх продуктах, незалежно від використовуваної платформи.

Завдяки цьому зменшені витрати часу на створення зручного та адаптивного інтерфейсу.

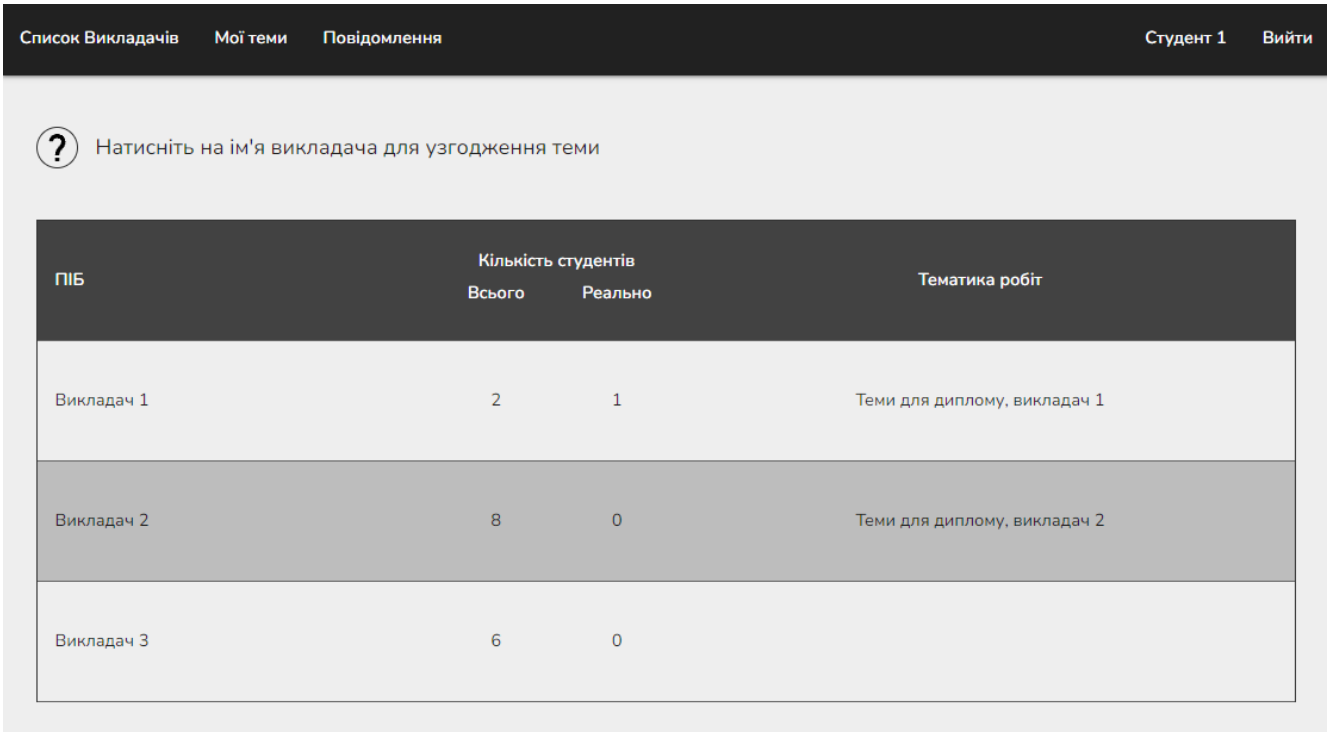


Рисунок 3.4 – Головна сторінка додатку, якщо користувач авторизований

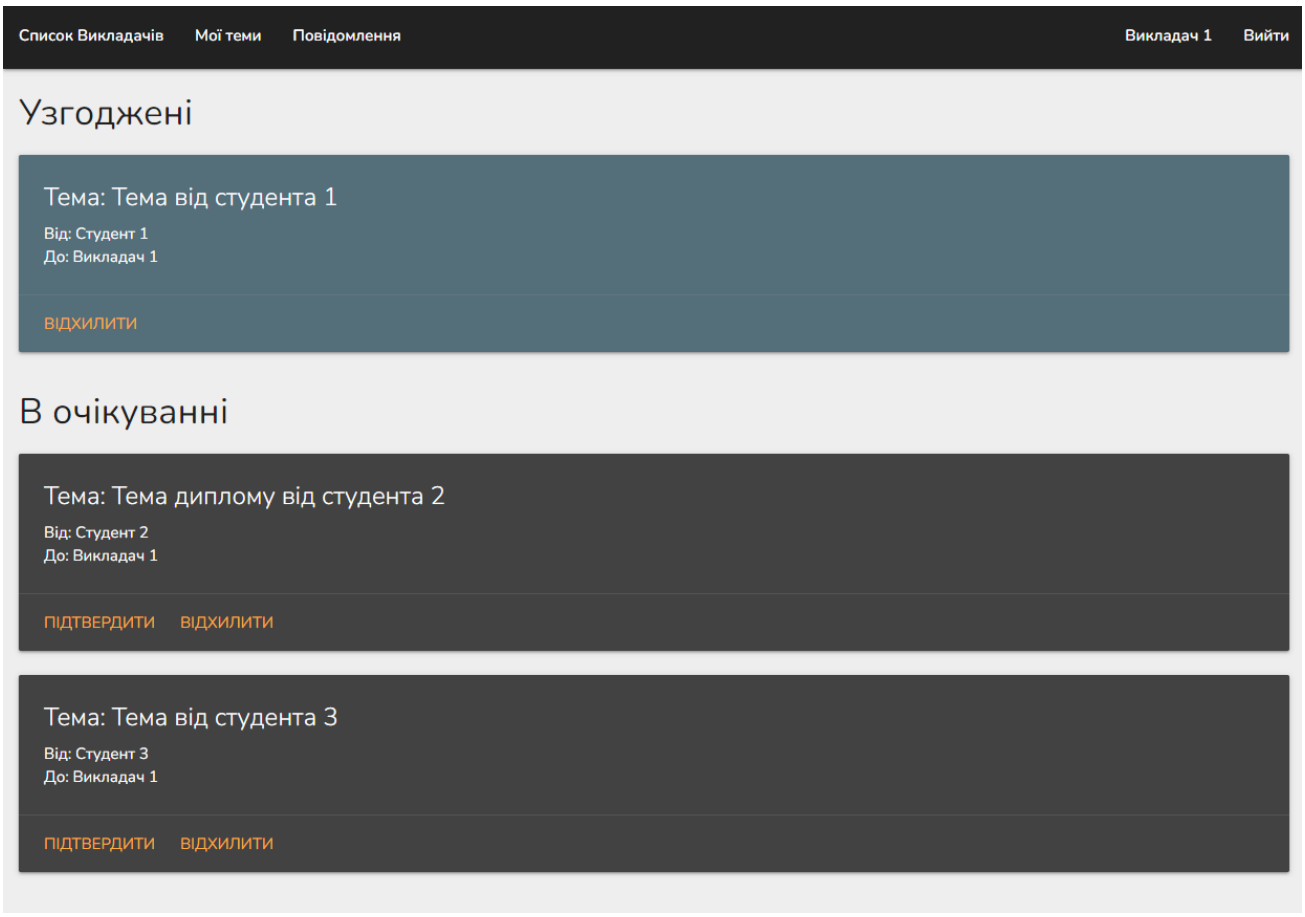


Рисунок 3.5 – Сторінка прийняття тем (Викладач)

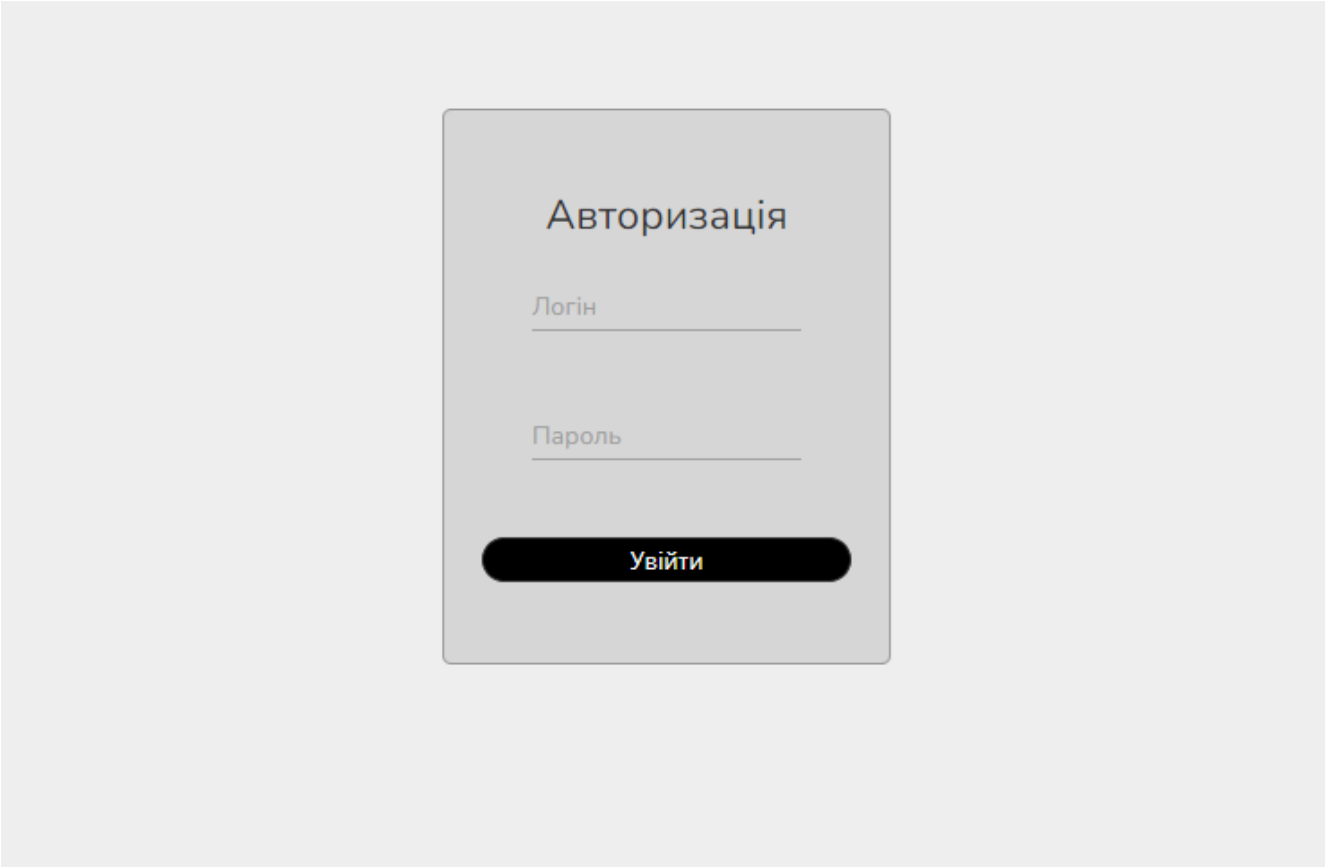


Рисунок 3.6 – Форма авторизації

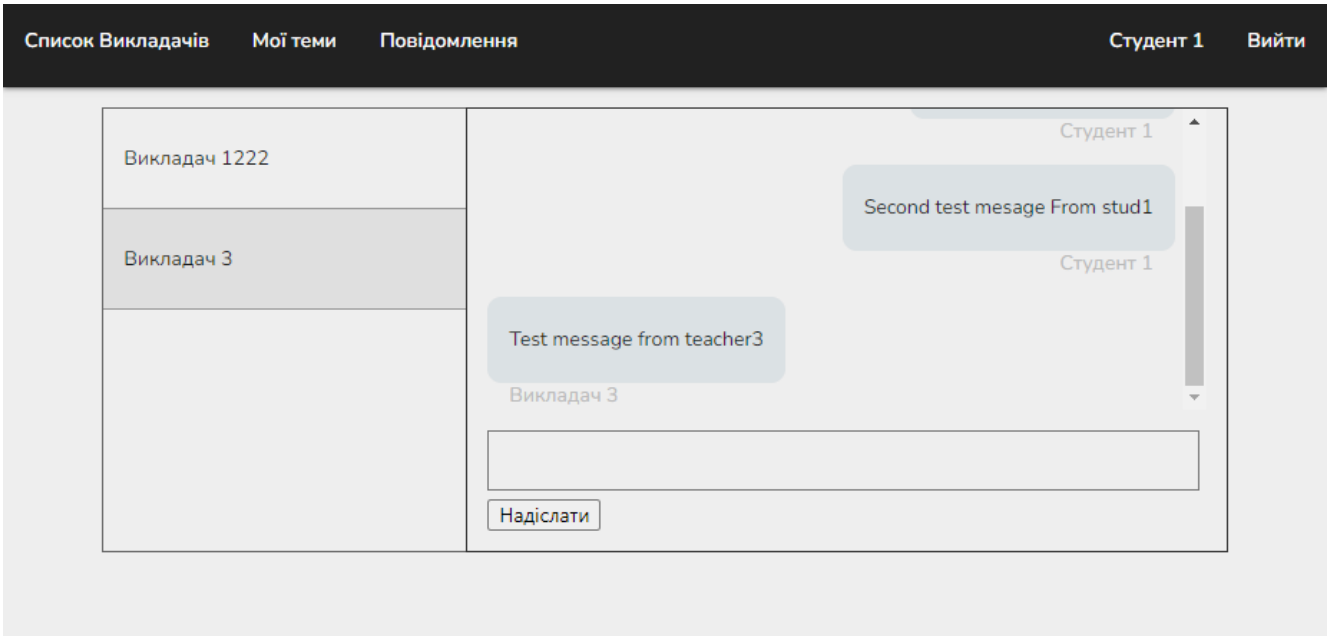


Рисунок 3.7 – Сторінка чатів користувача

3.6 Підключення серверу до бази даних

Для підключення бази даних на сервері потрібно встановити бібліотеку mongoose, яка містить в собі основні функції для взаємодії із базою даних. Після чого підключитися завдяки функції «connect».

```
await connect("mongodb+srv://diploma:root@cluster0.9c4gdf4.mongodb.net/?retryWrites=true&w=majority", () => {
  console.log("connected to DB");
});
```

Рисунок 3.8 – Підключення бази даних на сервері

Але залишати посилання у такому вигляді небезпечно, тому рядок з підключенням було перенесено до іншого файлу. Завдяки бібліотеці «dotenv» можна приховати конфіденційні файли , які потрібні додатку для правильної роботи, таких як облікові дані бази даних, ключі API та інша конфіденційна інформація. Для цього потрібно сворити файл .env з даними у форматі "ключ=значення".

3.7 Реалізація основних функцій

Для реалізації REST API був використаний фреймворк Express. В основі патерн MVC (Model-View-Controller).

MVC (Model-View-Controller) - це патерн проектування програмного забезпечення, що розділяє додаток на три основних компоненти: модель (Model), представлення (View) та контролер (Controller). Кожен з цих компонентів виконує певні функції та має свою відповідальність у додатку.

Модель (Model) відповідає за логіку додатку та управління даними. Вона зберігає дані та виконує операції з ними, такі як додавання, видалення, оновлення тощо. Модель може взаємодіяти з базою даних або з іншими джерелами даних.

```
import { Schema, model } from "mongoose";

const User = new Schema({
  username: { type: String, unique: true, required: true },
  displayname: { type: String, required: true },
  password: { type: String, required: true },
  roles: [{ type: String, ref: "Role" }],
  maxStudents: { type: Number, default: undefined },
  students: { type: Array, default: undefined },
  info: { type: String, default: undefined },
});

export default model("User", User);
```

Рисунок 3.9 – Реалізація моделі користувача

Представлення (View) відповідає за відображення даних користувачеві та взаємодію з ним. Вона відображає дані, збережені в моделі, у вигляді, зрозумілому користувачеві. Представлення може бути у вигляді сторінок веб-сайту, інтерфейсу користувача додатку тощо.

Контролер (Controller) відповідає за обробку запитів користувача та взаємодію між моделлю та представленням. Він приймає запити користувача, виконує необхідні дії з моделлю, отримує результат та передає його в представлення. Контролер може мати різні методи для обробки різних запитів користувача.

```
class authConrtoller {
  async registration(req, res) {···
  }
  async login(req, res, next) {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        return res.status(400).json({ message: "Error during login", errors });
      }
      const { username, password } = req.body;
      const user = await User.findOne({ username });
      if (!user) {
        throw new ApiError.BadRequest(`User ${username} not found`);
      }
      const validPassword = bcrypt.compareSync(password, user.password);
      if (!validPassword) {
        throw new ApiError.BadRequest(`Wrong password`);
      }
      const tokens = generateTokens({ ...user });
      return res.json({ tokens, user: { _id: user._id, displayname: user.displayname, roles: user.roles } });
    } catch (error) {
      next(error);
    }
  }
}

export default new authConrtoller();
```

Рисунок 3.10 – Контролер відповідальний за вхід та реєстрацію

Використання патерна MVC дозволяє розділити логіку додатку на окремі компоненти, що полегшує розробку, тестування та підтримку додатку. Також він дозволяє змінювати один компонент без впливу на інші компоненти, що збільшує гнучкість та масштабованість додатку.

Клієнт для отримання,додавання чи оновлення інформації у базі даних, натсилає запити на сервер. REST-сервер повинен слухати на певному URL-адресі та підтримувати HTTP-методи, такі як GET, POST, PUT, DELETE тощо.

Наприклад, якщо REST-сервер слухає на адресі "http://example.com/api/users", клієнт може зробити запит на список користувачів за допомогою GET-запиту до цієї адреси. Клієнт може також додати нового користувача до системи за допомогою POST-запиту до тієї ж адреси "http://example.com/api/users".

При виконанні запиту клієнт може включати додаткову інформацію у заголовки (headers) запиту, таку як тип контенту (content-type), токени авторизації тощо, щоб отримати доступ до захищених ресурсів на сервері.

```
app.use(cors());
app.use(json());
app.use("/chat", chatRouter);
app.use("/users", userRouter);
app.use("/auth", authRouter);
app.use("/tickets", ticketRouter);
app.use("/teacher", teacherRouter);
app.use(errorMiddleware);
```

Рисунок 3.11 – Підключення різних роутів (шляхів) доступу

Також були використані middleware. Middleware - це функції, які виконуються послідовно при обробці запиту на веб-сервері у фреймворку Express. Вони можуть виконувати будь-які дії, такі як перевірка авторизації, обробка запиту, модифікація об'єкту запиту та відповіді, або передача управління на наступний middleware у ланцюжку.

Middleware у Express можуть бути глобальними, тобто вони застосовуються до всіх запитів, що приходять до сервера, або локальними, тобто застосовуються лише до запитів, що відповідають конкретному маршруту. Для додавання middleware у Express використовується функція use().

3.8 Реалізація WebSocket чату

У додатку було реалізовано спілкування між студентом та викладачами через чати. Для реалізації чату був використаний протокол WebSocket.

Чому WebSocket, а не HTTP протокол?

- HTTP - це протокол передачі гіпертексту, який використовується для передачі запитів та відповідей між веб-сервером та веб-клієнтом. В HTTP запити клієнта та відповіді сервера є окремими, і кожен запит клієнта передається окремо на сервер, після чого сервер повертає відповідь клієнту.
- WebSocket - це протокол повного дуплексного зв'язку, який дозволяє клієнтові та серверу взаємодіяти одночасно, без потреби в окремих запитах

та відповідях. WebSocket забезпечує стабільне з'єднання між клієнтом та сервером, через яке можна передавати дані у режимі реального часу. У порівнянні з HTTP, WebSocket використовує більше ресурсів сервера та мережі, але забезпечує вищу швидкість передачі даних та кращу продуктивність.

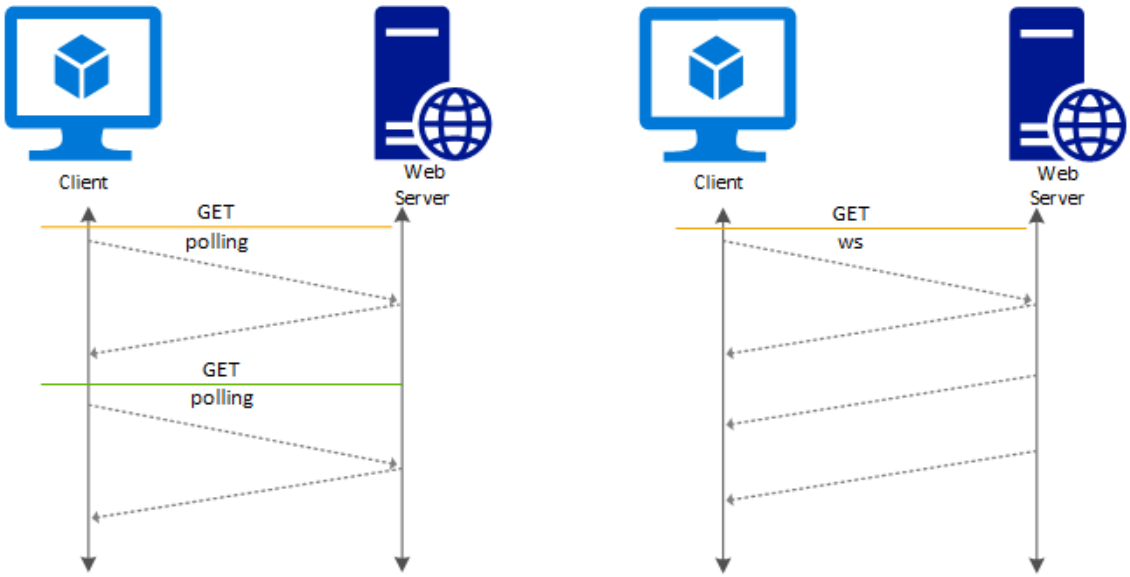


Рисунок 3.12 – Різниця між частотою запитів у WebSocket та HTTP протоколів

Отже, основна різниця між HTTP та WebSocket полягає у способі передачі даних. HTTP передає дані за запитами та відповідями, тоді як WebSocket забезпечує повний дуплексний зв'язок, який дозволяє передавати дані в режимі реального часу.

3.9 Зв’язок серверу та клієнту

Для доступу до бази даних потрібно надсилати http запити на сервер. Для цього були використані React Hooks та Redux toolkit.

Хуки - це функції в React, які дозволяють вам використовувати стан та інші можливості React у функціональних компонентах. Хуки були введені в React у версії 16.8 як спосіб спростити та поліпшити роботу зі станом компонентів.

Хуки працюють шляхом додавання "вбудованого стану" в функціональні компоненти React. За допомогою хуків, ви можете додавати стан, методи життєвого циклу та інші функції в ваші компоненти. Хуки дозволяють розділяти логіку між різними хуками та зберігати стан між рендерами.

Наприклад, хук useState дозволяє вам додавати локальний стан в ваш компонент. Він приймає початковий стан як аргумент та повертає масив, що містить поточний стан та функцію, яку можна використовувати для оновлення стану.

Але використовувати useState не завжди краща ідея, є таке поняття як prop drilling. Prop drilling - це шаблон, що використовується в React для передачі властивостей (props) вниз по ієрархії компонентів. Це означає, що властивості передаються з компоненту в компонент, що може призвести до того, що ви повинні передати одні й ті ж властивості кілька разів по ієрархії компонентів.

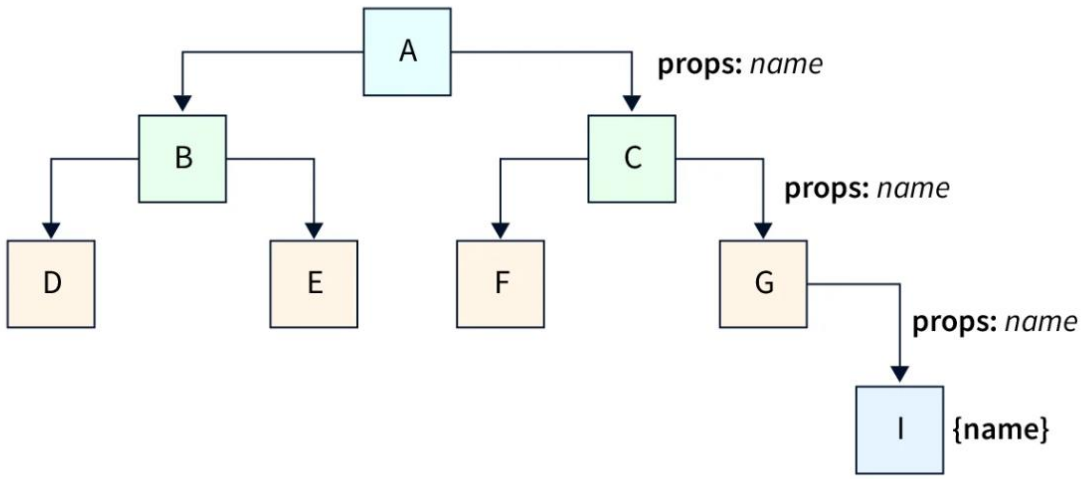


Рисунок 3.13 – Приклад передання даних у інші компоненти за шаблоном prop drilling

На рисунку 3.12 зображено ієрархію додатку, компоненту “I” потрібна змінна “name”, яка знаходиться у компоненті “A”. Щоб її отримати потрібно передати ту саму змінну через додаткові компоненти, які у свою чергу через зміну стану вони

будуть відображені повторно. Це призводить до зменшення продуктивності додатку в цілому. Тому було використано глобальне сховище Redux.

У Redux використовується архітектура Flux, яка є шаблоном для розробки веб-додатків з одностороннім потоком даних. Це означає, що дані можуть бути змінені лише через взаємодію зі спеціальними об'єктами, які називаються "store" (сховище). У Redux інформація зберігається в "store", а зміни до даних вносяться за допомогою "actions" (дій).

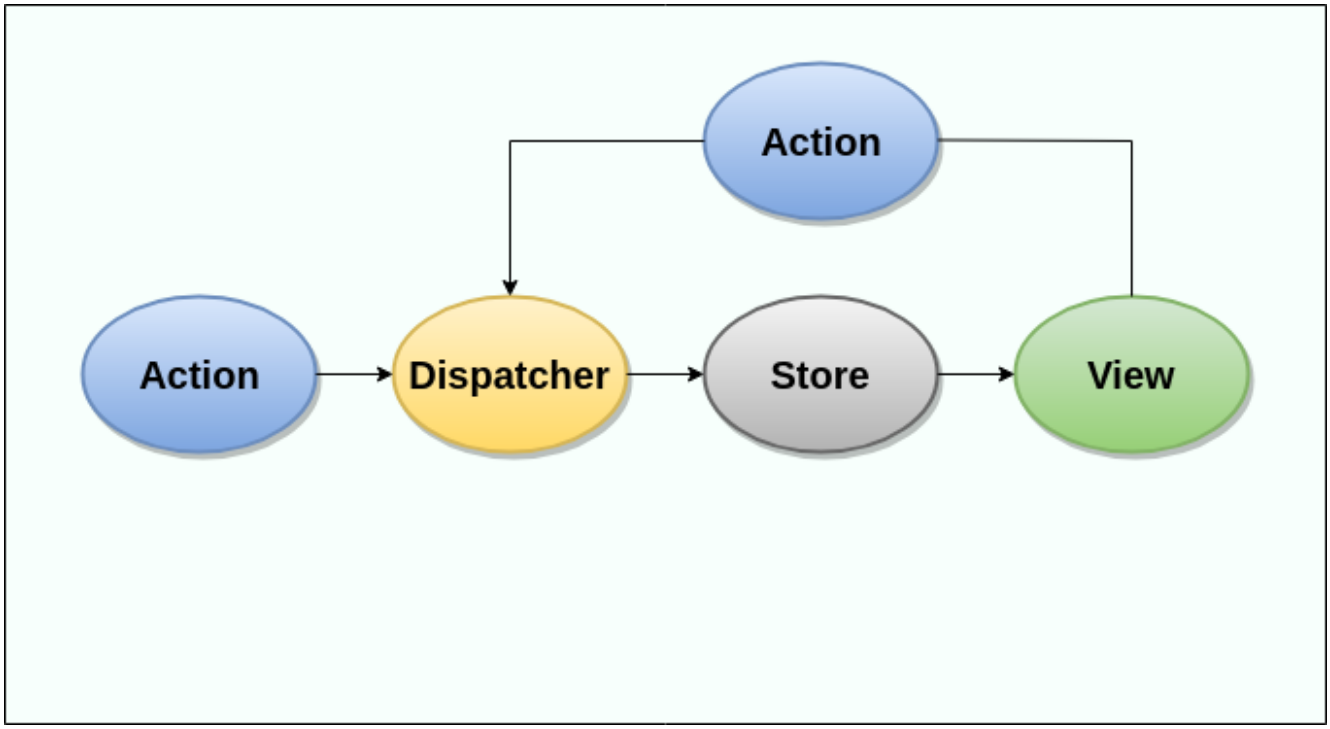


Рисунок 3.14 – Flux архітектура

Ця архітектура забезпечує декларативний підхід до зміни даних в додатку, що спрощує управління станом додатку та робить його більш передбачуваним. У Flux також використовуються поняття "dispatcher" (розсилка) та "view" (вид), які відповідають за передачу даних між компонентами додатку та за оновлення інтерфейсу користувача на основі змін у даних. У Redux такі поняття також присутні, хоча їх використовують інакше.

У Redux "store" містить увесь стан додатку, "actions" описують, що саме відбувається з даними, а "reducers" (редуктори) забезпечують зміни до даних у

"store" на основі "actions". Така структура дозволяє забезпечити чистоту та передбачуваність змін у стані додатку, що полегшує його тестування та підтримку.

```
export const store = configureStore({
  reducer: {
    user: userReducer,
    teacherList: teacherReducer,
    tickets: ticketReducer,
    chat: chatReducer,
    error: errorReducer,
  },
});
```

Рисунок 3.15 – Сховище Redux toolkit у додатку

```
export const userSlice = createSlice({
  name: "user",
  initialState,
  reducers: {
    setUser: (state, action) => {
      state.id = action.payload._id;
      state.displayName = action.payload.displayname;
      state.roles = action.payload.roles;
    },
  },
  extraReducers(builder) {
    builder
      .addCase(fetchUser.pending, (state) => {
        state.status = "loading";
      })
      .addCase(fetchUser.fulfilled, (state, action) => {
        state.id = action.payload._id;
        state.displayName = action.payload.displayname;
        state.roles = action.payload.roles;
        state.maxStudents = action.payload.maxStudents;
        state.students = action.payload.students;
        state.info = action.payload.info;
        state.status = "fullfiled";
      })
      .addCase(fetchUser.rejected, (state) => {
        state.error = "Can't load user";
        state.status = "fullfiled";
      });
  },
});
```

Рисунок 3.16 – Reducer відповідальний за поточного користувача (отримання та збереження інформації)

Бібліотека Redux Toolkit є набором інструментів, які спрощують розробку додатків з використанням Redux. Вона надає зручний та ефективний спосіб створення "store" та "reducers", що дозволяє зосередитися на бізнес-логіці додатку, а не на деталях роботи з Redux.

Використовуючи методи життєвого циклу компонентів, при створенні викликається фунція, яка надсилає запит до серверу та зберігає відповіть у глобальне сховище Redux.

3.10 Роутінг на клієнті

Роутінг на клієнті (client-side routing) - це процес, коли управління переходами між сторінками здійснюється на клієнті (в браузері), а не на сервері. Раніше, коли користувач клікав на посилання на веб-сторінці, браузер відправляв запит на сервер, сервер оброблював запит, генерував HTML-сторінку та відправляв її назад до браузера. Кожен перехід на нову сторінку призводив до повторного завантаження всієї сторінки з сервера, що займало багато часу та ресурсів.

З використанням роутінгу на клієнті, зміст сторінки завантажується асинхронно (зазвичай за допомогою AJAX-запитів) і відображається на одній сторінці, без необхідності повторного завантаження всієї сторінки. Роутінг на клієнті може бути реалізований за допомогою JavaScript-бібліотек та фреймворків, таких як React Router, Vue Router, Angular Router тощо.

Один з головних принципів роутінгу на клієнті - це спрощення користувальницького досвіду та зменшення часу очікування користувача на завантаження сторінок. Роутінг на клієнті також дозволяє зберігати стан сторінки між переходами, що дозволяє забезпечити плавну та зручну навігацію для користувача.

3.11 Тестування

При розробці окремих компонентів клієнту, їх «фінальні» версії було протестовано вручну. Переваги ручного тестування включають гнучкість і можливість виявлення непередбачених проблем, а також здатність виконувати тестування в умовах, коли автоматизовані засоби недоступні або недоцільні. Ручне тестування також може бути ефективним на початкових етапах розробки, коли функціональність програми ще не повністю визначена.

Таблиця 3.1 – Тест кейс входу у систему

Заголовок	Крок	Очікуваний результат
Вхід у систему	Відкрити сайт	З’являється форма авторизації
	Ввести некоректні дані у поле «Логін»	Програється анімація і слово «Логін» піднімється та змінює колір
	Ввести коректні дані у поле «Пароль»	Програється анімація і слово «Пароль» піднімється та змінює колір
	Натиснути кнопку «Увійти»	Поле «Логін» змінить колір на червоний, з’являється повідомлення про помилку
	Ввести коректні дані у поле «Логін»	Поле змінить колір на початковий
	Натиснути кнопку «Увійти»	З’являється головна сторінка

Таблиця 3.2 – Тест кейс реєстрації теми студентом

Заголовок	Крок	Очікуваний результат
Реєстрації теми студентом	Натиснути на ім’я викладача	З’являється чат та форма для теми
	Натиснути «Надіслати»	З’являється повідомлення «Тема не може бути пустою»
	Ввести у текстове поле форми «Тема 1»	У полі буде написано «Тема 1»
	Натиснути «Надіслати»	Перехід на сторінку «Мої теми». На сторінці під текстом «В очікуванні» є «Тікет» з темою, ім’ям стунта, ім’ям викладача та кнопкою «Редагувати»

Для тестування серверу використовувалася програма Postman.

Postman - це інструмент для тестування та розробки API (Application Programming Interface). Він дозволяє розробникам легко створювати, тестувати та документувати різні типи запитів API.

Postman може бути використаний для створення та відправки HTTP-запитів на різні API-маршрути. За допомогою Postman можна виконувати різні типи запитів, такі як GET, POST, PUT, DELETE тощо, та перевіряти відповідь сервера. Він також дозволяє виконувати автоматичні тести, складати запити з різними параметрами, додавати хедери та інші налаштування запитів.

Postman також має функціонал для автоматичного створення документації API. За допомогою Postman можна створити документацію API в різних форматах, таких як HTML-сторінки, PDF-документи, Markdown-файли тощо.

Postman є дуже корисним інструментом для розробників, які працюють з API. Він дозволяє ефективно тестувати та відлагоджувати API, документувати його та спрощувати процес розробки. За допомогою Postman можна значно зекономити час та зусилля, які зазвичай пов'язані з тестуванням та розробкою API.

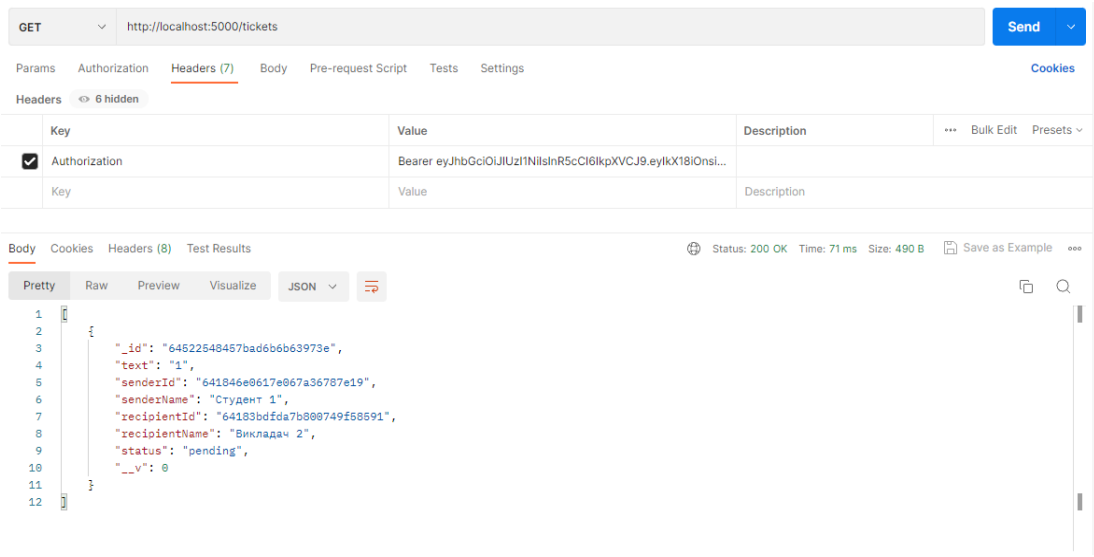


Рисунок 3.17 – Приклад тестування серверу через Postman (отримання тем користучава)

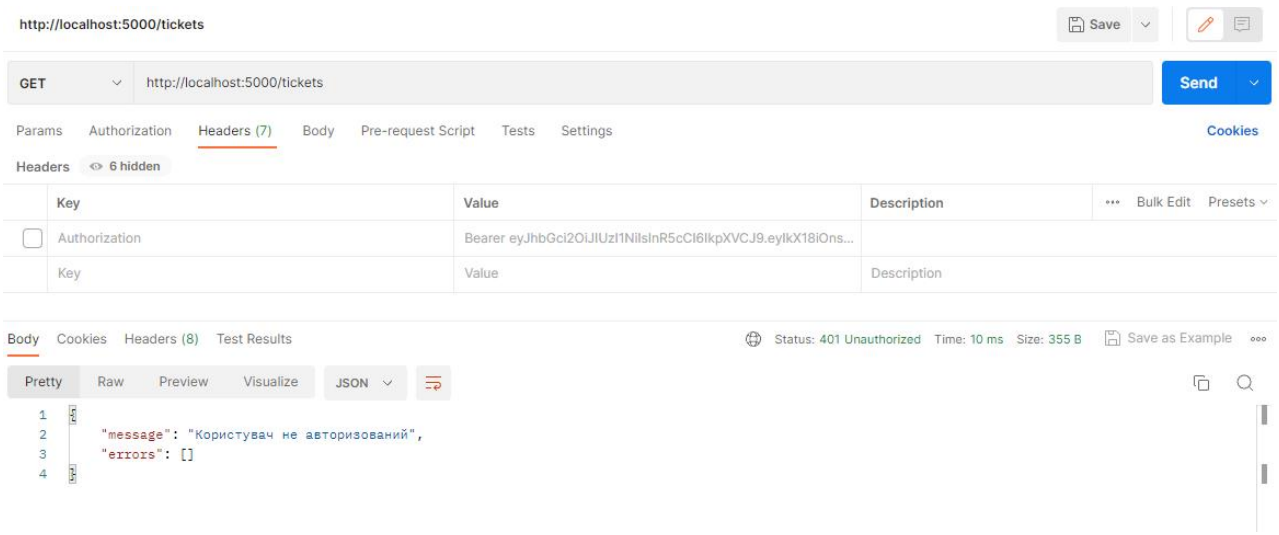


Рисунок 3.18 – Приклад тестування серверу через Postman (відсутній токен авторизації)

ВИСНОВКИ

У даній дипломній роботі був розроблений веб-сервіс для підтримки організаційних процесів дипломування з використанням бібліотеки React.js. Проект має на меті полегшити процес дипломування студентів та управління цим процесом з боку адміністраторів.

У результаті розробки були вирішені основні завдання, поставлені перед розробником. Було створено інтерфейс користувача, який дозволяє студентам обрати тему своєї роботи. Також адміністраторам було надано можливість керувати процесом дипломування, перевіряти роботи студентів та визначати результати захисту.

При розробці проекту були використані сучасні технології та інструменти, зокрема бібліотека React.js, яка дозволила створити динамічний та ефективний інтерфейс користувача, Node.js та Express.

Загалом, розробка веб-сервісу для підтримки організаційних процесів дипломування з використанням бібліотеки React.js є актуальною та перспективною темою. Результати дослідження можуть бути використані у подальшій розробці сучасних технологій для покращення процесу дипломування та управління ним.

ПЕРЕЛІК ПОСИЛАНЬ

1. Taylor D. What is MongoDB? Introduction, Architecture, Features & Example [Електронний ресурс] / David Taylor // guru99. – 2023. – Режим доступу до ресурсу: https://www.guru99.com/what-is-mongodb.html.
2. Express 4.x Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/.
3. Perera M. Simplifying Redux with Redux Toolkit [Електронний ресурс] / Madushika Perera // medium. – 2021. – Режим доступу до ресурсу: https://blog.bitsrc.io/simplifying-redux-with-redux-toolkit-6236c28cdfcb.
4. SQL vs NoSQL: Differences, Databases, and Decisions [Електронний ресурс] – Режим доступу до ресурсу: https://ua.talend.com/resources/sql-vs-nosql/.
5. Karnik N. Introduction to Mongoose for MongoDB [Електронний ресурс] / Nick Karnik // medium. – 2018. – Режим доступу до ресурсу: https://medium.com/free-code-camp/introduction-to-mongoose-for-mongodb-d2a7aa593c57.
6. WebSocket protocol [Електронний ресурс] // wallarm. – 2021. – Режим доступу до ресурсу: https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is.
7. socket.io Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://socket.io/docs/v4/.
8. Salihefendic A. Flux vs. MVC [Електронний ресурс] / Amir Salihefendic // medium. – 2015. – Режим доступу до ресурсу: https://medium.com/hacking-and-gonzo/flux-vs-mvc-design-patterns-57b28c0f71b7.
9. What is Babel Transpiler? [Електронний ресурс] – Режим доступу до ресурсу: https://www.educative.io/answers/what-is-babel-transpiler.

10. The DRY Principle: Benefits and Costs with Examples [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://thevaluable.dev/dry-principle-cost-benefit-example/.

11. Prop Drilling in React [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.scaler.com/topics/react/prop-drilling-in-react/.

ДОДАТОК А

Графічна частина

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«Розробка web-сервісу для підтримки організаційних процесів дипломування з використанням бібліотеки React.JS»

Виконав студент 4 курс
Групи ПД-42
Шовтенко Валентин Юрійович
Керівник роботи
асистент кафедри ІПЗ, Залива Віталій Вікторович

Київ – 2023

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета** - підвищення ефективності та зручності організаційних процесів дипломування.
- **Об’єкт дослідження** - організаційні процеси дипломування.
- **Предмет дослідження** - веб-технології клієнтської та серверної частин при розробці веб додатків.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Огляд та аналіз літературних джерел для створення веб-сервісів.
2. Огляд та аналіз програмного забезпечення, яке може бути використано для розробки.
3. Розробка архітектури додатку.
4. Створення додатку для підтримки організаційних процесів дипломування.
5. Тестування системи.

3

АНАЛІЗ АНАЛОГІВ

	Microsoft Excel	Google Sheets	Smartsheet	**Розроблений додаток**
Платформи	Android, iOs, Windows, Mac OS	Web	Android, iOs, Web	Web
Особисті повідомлення	-	-	-	+
Можливість редагування	+	+	+	+
Спільне редагування	-	+	+	+
Спеціалізовано для підтримки організаційних процесів дипломування	-	-	-	+
Наявність сповіщень	-	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

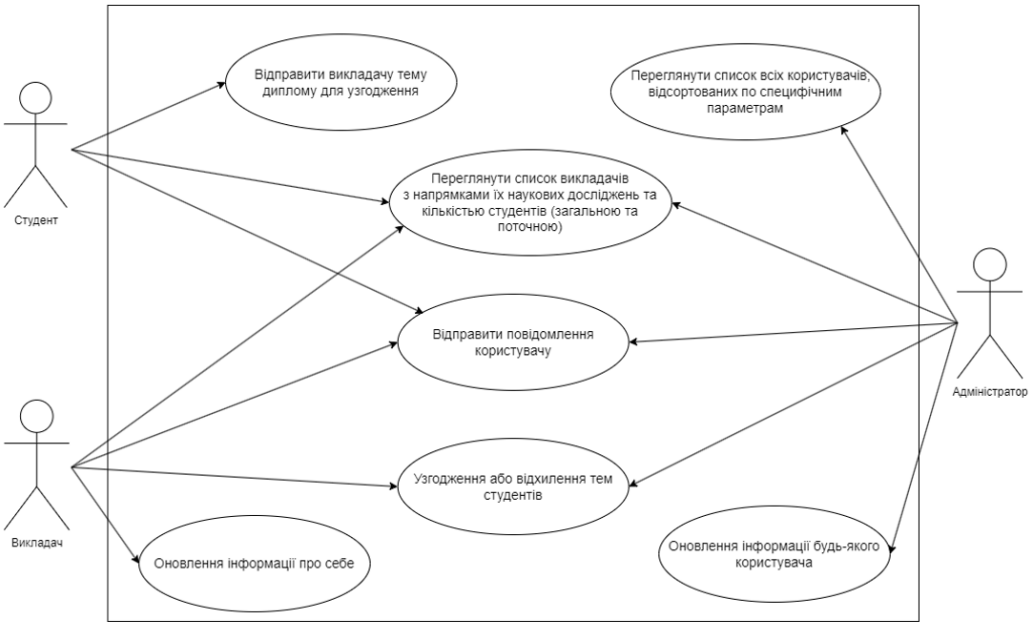
1. Реєстрація, авторизація та аутентифікація користувачів.
2. Перегляд викладачів та напрямків їх наукових досліджень.
3. Редагування інформації про себе для викладачів.
4. Зворотній зв’язок студентів та викладачів – студенти та викладачі можуть відправляти повідомлення один одному щодо їх дипломної роботи.
5. Рецензування тем – викладачі можуть перевірити отриману тему після чого прийняти або відхилити її.
6. Мати користувача адміністратора, який має можливість закріпити будь якого студента за керівником та виправити тему.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

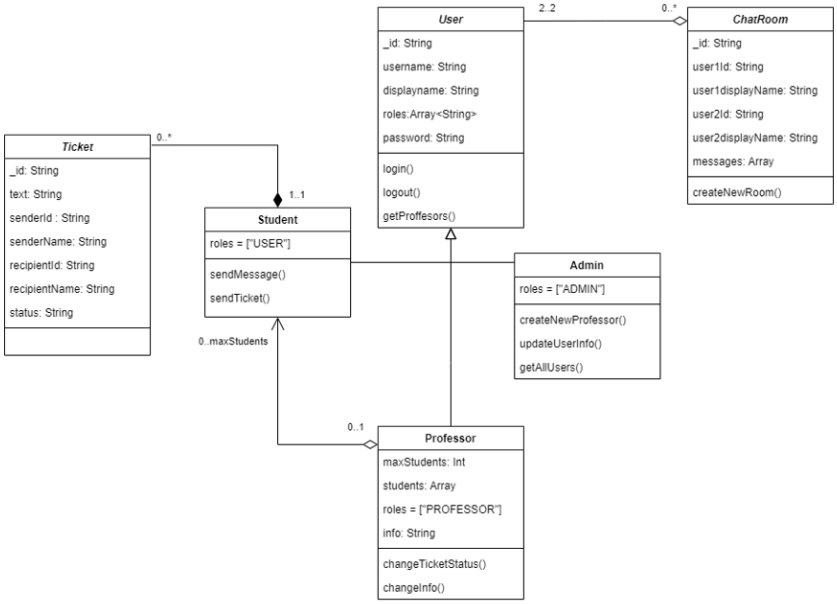


7

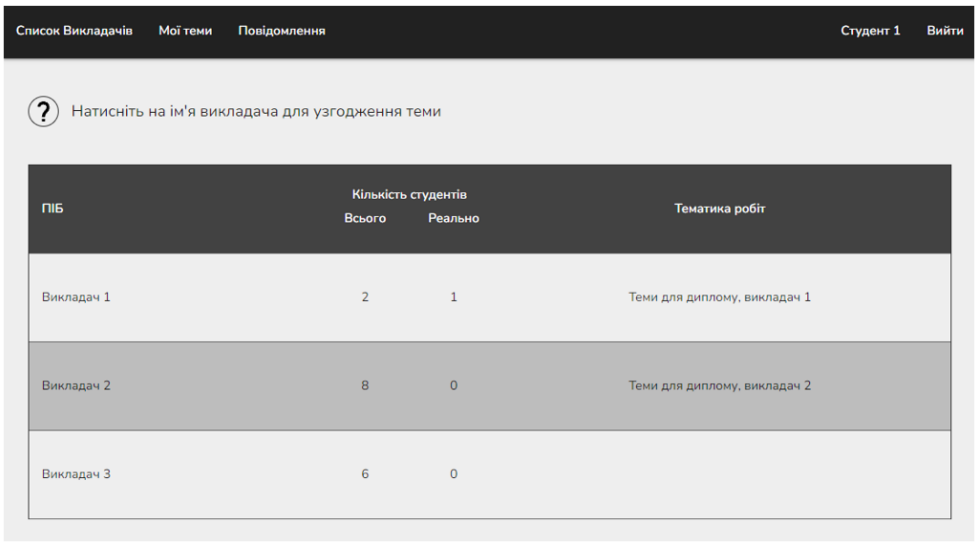
ДІАГРАМА ДІЯЛЬНОСТІ

8

ДІАГРАМА КЛАСІВ

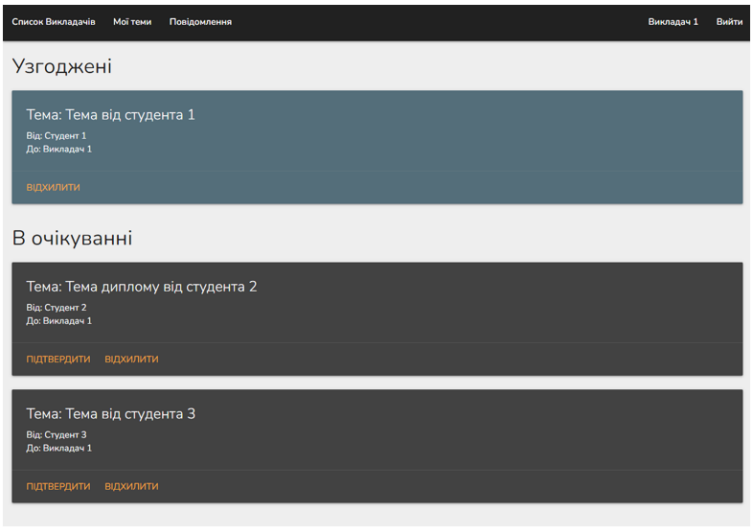


ЕКРАННІ ФОРМИ

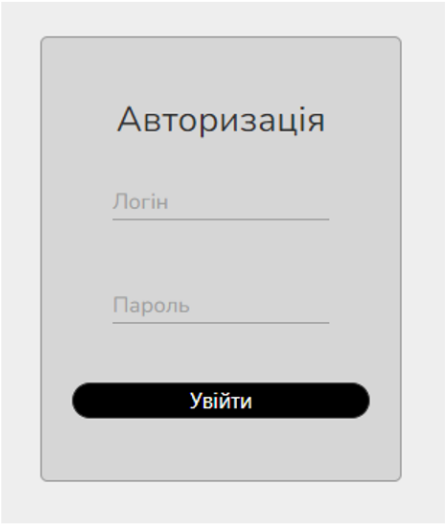


Головна сторінка додатку, якщо користувач авторизований

ЕКРАННІ ФОРМИ



Сторінка прийняття тем (викладач)



Форма авторизації

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Шовтенко В.Ю. Використання Websocket для клієнт-серверної комунікації / Шовтенко В.Ю. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 93.
2. Шовтенко В.Ю. Redux toolkit як спосіб зберігання стану у веб-додатках / Шовтенко В.Ю. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 94.

13

ВИСНОВКИ

1. Проведений аналіз предметної галузі, визначена її специфіка та актуальність.
2. Після аналізу основних методів розробки веб-сервісів було обрано REST архітектуру.
3. Визначені переваги та недоліки інших рішеннь для поставленого завдання.
4. Розроблений веб-сервіс для підтримки організаційних процесів дипломування.
5. Проведено тестування додатку, результати яких підтвердили, що програма відповідає поставленим вимогам.