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

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 121 «Інженерія програмного забезпечення»

освітня програма «Інженерія програмного забезпечення розподілених систем»

на тему: «Підсистема організації спільної роботи учасників освітнього процесу»

Виконав (-ла):
студент (-ка) IV курсу, групи ТВ-з71
Колотуша Владислав Віталійович _____

Керівник:
Професор Гаврилко Євген Володимирович _____

Рецензент:
Посада, науковий ступінь, вчене звання,
Прізвище, ім’я, по батькові _____

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

Київ – 2021 року

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

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 121 «Інженерія програмного забезпечення»

освітня програма «Інженерія програмного забезпечення розподілених систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2021р.

ЗАВДАННЯ

на дипломну роботу студенту

Колотуша Владислав Віталійович
(прізвище, ім’я, по батькові)

1. Тема роботи Підсистема організації спільної роботи учасників освітнього процесу керівник роботи професор Гаврилко Євген Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи мови програмування JavaScript та TypeScript, платформа веб-браузер

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Дослідити предметну область та визначити існуючі аналоги.

Визначити необхідну функціональність, яка буде приносити найбільшу користь.

Спроектувати гнучку та надійну архітектуру системи.

Розробити дизайн застосунку.

Створити повноцінний додаток.

Пройти усі етапи випробування та усунути знайдені недоліки.

5. Перелік ілюстративного матеріалу

Актуальність, Задачі, Аналоги, Схема системи, Вибір платформи, Вибір технологій, Діаграма класів, Структура бази даних, UML-діаграма діяльності, Вхідні дані,

Вихідні дані, Висновок.

6. Консультанти розділів роботи

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

7. Дата видачі завдання ”10” жовтня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	09.10.2020	
2.	Вивчення та аналіз задачі	10.04.2021	
3.	Розробка архітектури та загальної структури системи	21.04.2021	
4.	Розробка структур окремих підсистем	27.04.2020	
5.	Програмна реалізація системи	01.05.2021	
6.	Оформлення пояснювальної записки	10.05.2021	
7.	Захист програмного продукту	11.05.2021	
8.	Передзахист	25.05.2021	
9.	Захист	16.06.2021	

Студент _____ Колотуша В. В.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Гаврилко Е.В.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту обсягом 61 сторінка складається з п’ятьох розділів і містить 11 рисунків, 15 таблиць, 3 додатки, 15 джерел.

У дипломному проекті реалізовано підсистему узгодження текстових документів. Метою підсистеми є поліпшення освітнього процесу у віддаленому режимі. Задачами розробки є:

1. Дослідити предметну область та визначити існуючі аналоги.
2. Визначити необхідну функціональність, яка буде приносити найбільшу користь.
3. Спроектувати гнучку та надійну архітектуру системи.
4. Розробити дизайн застосунку.
5. Створити повноцінний додаток.
6. Пройти усі етапи випробування та усунути знайдені недоліки.

Як основну платформу було обрано веб-браузер через наступні причини:

– високий рівень доступу;
– не має необхідності встановлювати щось на свій пристрій;
– для запуску не потрібно проходити перевірки.

Під час написання системи використано функціональна та об’єктно-орієнтована парадигми програмування.

Об’єктно-орієнтована парадигма програмування використовується для написання класів, які реалізують моделі об’єктів з бази даних, а також для класів, які працюють з локальним сховищем або HTTP запитами.

Функціональна парадигма програмування використовується для написання функцій, які реалізують REST API, реалізації бізнес логіки програмного продукту,

Все вищенаведене допомагає створювати надійні та гнучкі системи, в яких не складно розібратися, а внесення правок не забирає купу часу.

Основні результати, яких було досягнуто в процесі виконання роботи:

1.Функціонал додатку допоможе в організації освітнього процесу у віддаленому режимі.

2.Отримано систему, готову до змін, розширення або інтеграції до комплексного рішення.

3. Додаток має зручний та зрозумілий інтерфейс.

4. Застосунок має весь необхідний функціонал.

5. Система працює надійно, знайдені недоліки були усунуті.

ДОКУМЕНТООБІГ, УЗГОДЖЕННЯ ДОКУМЕНТІВ, ОСВІТНІЙ ПРОЦЕС, АВТОМАТИЗАЦІЯ.

ABSTRACT

Structure and scope of work. The 60-page explanatory note of the diploma project consists of five sections and contains 11 figures, 15 tables, 3 appendices, 20 sources.

In the diploma project the subsystem of coordination of text documents is realized. The purpose of the subsystem is to improve the educational process remotely. The development tasks are:

1. Investigate the subject area and identify existing analogues.
2. Identify the required functionality that will be most useful.
3. Design a flexible and reliable system architecture.
4. Develop an application design.
5. Create a full-fledged application.
6. Go through all the stages of the test and eliminate the shortcomings found.

The web browser was chosen as the main platform for the following reasons:

- high level of access;
- no need to install anything on your device;
- tests are not necessary to make the app available for users.

Functional and object-oriented programming paradigms were used in writing the system.

The object-oriented programming paradigm is used to write classes that implement object models from a database, as well as classes that work with local storage or HTTP requests.

Functional programming paradigm is used to write functions that implement the REST API, the implementation of business logic software product,

All of the above helps to create reliable and flexible systems that are not difficult to understand, and making changes does not take a lot of time.

The main results that were achieved in the process of work:

1. The functionality of the application will help in organizing the educational process remotely.

2. A system is ready for change, expansion or integration into a comprehensive solution.

3. The application has a user-friendly interface.

4. The application has all the necessary functionality.

5. The system works reliably, the identified shortcomings have been eliminated.

DOCUMENT FLOW, DOCUMENT COORDINATION, EDUCATIONAL PROCESS, AUTOMATION.

ЗМІСТ

ВСТУП

Вже не для кого не секрет, що з кожним роком все більше і більше сфер починає використовувати цифрові можливості задля збільшення ефективності своєї роботи: починаючи від банківської справи і закінчуючи медициною. Ледь не кожен день оновлюється перелік послуг, які кожен з нас може отримати не контактуючи з іншими людьми.

Проте не всі сфери нашого життя поспішали за всім світом. І не було ніякого секрету в пояснені чому ж так відбувалося. Відповідь на таке питання доволі проста: не було необхідності, не було попиту. Саме тому студенти рік за роком ходили на заняття, які проходили офлайн, роздруковували тисячі сторінок рефератів, доповідей, курсових робіт і постійно фізично контактували з іншими студентами та викладачами.

Проте в один момент все змінилося. Зовнішні фактори змусили нас швидко переосмислити принципи взаємодії між різними учасниками освітнього процесу та навчитися жити у новій реальності. Там, де не має місця офлайн заняттям та фізичним контактам між людьми. Навіть, якщо цей фактор має тимчасовий характер, ми не можемо точно сказати наскільки довго він буде впливати на нас та чи не з’явиться у майбутньому схожа проблема.

Отже, перед нами постала проблема перевести всіх учасників освітнього процесу на віддалений режим роботи. Для цього нам необхідна велика кількість програмного забезпечення, яка дозволить нам взаємодіяти між собою. Тому мною було вирішено розробити застосунок для узгодження текстових документів, який може бути частиною великої системи. Користувачами моєї програми можуть бути студенти та викладачі. Вони можуть обмінюватися документами, коментувати їх та в кінці кінців узгоджувати.

На даний момент існує декілька сервісів які надають схожі можливості: Google Documents та Deals. Проте у них є недоліки:

- вони не спеціалізуються на вирішені проблеми узгодження документів
- не усі можливості безкоштовні

- їх розробка ніяк не контролюється освітньою спільнотою, а тому є ризик втратити необхідні можливості у майбутньому

Під час обдумування проблеми було вирішено, що застосунок буде працювати на платформі веб-браузера, так як вона найбільш доступна для звичайного користувача. Отже, список технологій, які можна було обрати суттєво зменшився. В результаті було вирішено слідувати наступним пунктам:

- програмне забезпечення ділиться на 2 модуля: клієнтська та серверна частина
- для розробки першої буде застосовуватися мова програмування Typescript та бібліотека для створення інтерфейсів користувача React
- серверна частина буде зроблена за допомогою Node.js та фреймворка Express
- для зберігання даних використаємо нереляційну базу даних MongoDB
- клієнтську частину було вирішено поділити на 4 частини: авторизація, список документів, створення документів та перегляд документу

Усе вищенаведене допоможе швидко розробляти застосунок та вносити будь-які зміни, уникати великої кількості помилок, які могли б бути виявленими на пізніх стадіях розробки та тестування програмного забезпечення. Також з легкістю буде забезпечено зручний та зрозумілий інтерфейс користувача, що збільшить довіру та лояльність користувачів.

Отже, дана робота присвячена підсистемі організації спільної роботи учасників освітнього процесу, а саме веб-застосунку для узгодження текстових документів. Вона дозволить усім учасникам освітнього процесу можуть завантажувати документи, давати доступ до них іншим користувачам для коментування та узгодження. Як приклад, записка даної дипломної роботи може бути завантажена у застосунок для подальшого узгодження з дипломним керівником.

Метою даного проекту є пришвидшення та полегшення процесу узгодження будь-яких текстових документів між учасниками освітньої спільноти.

1. ПОСТАНОВКА ЗАДАЧІ

1.1 Призначення розробки

Призначенням підсистеми є обмін доступом до своїх документів задля узгодження їх тексту (студент – студент, студент – викладач).

1.2 Цілі та задачі розробки

Цілями розробки застосунку є покращити процес навчання у віддаленому режимі за рахунок:

- надання інструменту для обміну документами;
- можливості зберігати як і узгоджені документи, так і ті, що тільки очікують узгодження;
- можливості коментувати свої документи та ті, до яких вам надали доступ;
- регулювання рівня доступу до свого документу;
- завантаження оновлених версій документів;

1.3 Задачі для клієнтської частини

Загальні вимоги до веб-застосунку:

- система повинена відповідати всім рекомендаціям дизайну;
- інтерфейс повинен бути зручним та зрозумілим;
- частота, з якою пристрій формування зображення відображає послідовні зображення не повинна бути менша за 60;
- відгуки на дії користувача мають бути чіткими та швидкими;
- у порталі не перевизначені звичні функції системних значків;
- додаток не має визначати або некоректно використовувати стандартні шаблони інтерфейсу користувача.

Головні задачі:

- веб-портал розрахований на 1 категорію користувачів;
- застосунок має складатися 4 модулів:
- авторизація та реєстрація;
- список документів;
- створення документу (завантаження файлу, введення основної інформації та надання доступу);
- перегляд документу (з можливістю коментування та узгодження).

1.4 Задачі для серверної частини

Загальні вимоги:

- клієнт та сервер взаємодіють за допомогою HTTP-запитів. У випадках, коли дані для передачі є специфічними об’єктами, вони конвертуються у рядковий формат JSON. Його використання обумовлено легкістю серіалізації та зворотної операції складних структур даних;
- у випадку відправлення запитів GET відбувається декодування даних у відповіді;
- швидке реагування на запити клієнта;
- надійна робота;
- відсутність серйозних проблем у випадку майбутнього розширення системи;
- клієнти та сервер створені за допомогою різних технологій, але це не впливає на основні принципи їх взаємодії.

Головні задачі:

- сервер надає усі необхідні дані клієнтській частині;
- він відповідальний за авторизацію та реєстрацію користувача в системі;
- сервер надає доступ до бази даних, що містять персональні дані користувачів та їх документи.

Висновок до розділу

У рамках цього розділу було визначено функціональні вимоги, сформульовано призначення, мету. А також було розглянуті основні вимоги до програмного забезпечення та задачі, які необхідно розв’язати.

2. ЗАГАЛЬНІ ПОЛОЖЕННЯ

2.1 Опис предметного середовища

Для опису прикладного середовища візьмемо типову ситуацію, коли студент працює над будь-якою доповіддю. І йому необхідно надати викладачу свій документ та узгодити його.

По-перше, обоє учаснику процесу повинні бути зареєстровані в системі. Якщо ж це не так, то потрібно пройти цей етап. Далі студент завантажує файл та надає доступ до нього викладачу. Після цього вони обоє мають змогу продивитись документ, залишити коментарі або ж узгодити документ.

Даний дипломний проект надає можливість навчальним закладам легко впровадити в свою інфраструктуру підсистему узгодження текстів документів, ставати більш технологічними, а студентам та викладачам бути більш ефективними за рахунок зменшення кількості особистих зустрічей та витрачання часу на друкування документів, які з великою ймовірністю будуть відхилені через не відповідність вимогам.

2.2 Опис процесу діяльності

Проведемо розгляд дій, які повинні бути виконані для того, щоб використовувати систему та послідовність дій при її використанні.

Спершу користувачу необхідно зареєструватися в системі. Далі у нього буде можливість додати новий документ. Цей процес складається з наступних етапів:

- завантаження файлу;
- введення основної інформації;
- надання доступу до документу іншим користувачам.

Після цього з’являється можливість переглянути документ, додати до нього коментарі, якщо це необхідно. Якщо усі згодні що документ відповідає вимогам то треба позначити документ як узгоджений. В протилежному випадку, додаються

коментарі, завантажується нова версія документу и ці кроки будуть повторюватись поки усі не сторони не дійдуть згоди.

Також опишемо ці дії за допомогою UML-діаграми діяльності, яку наведено на рисунку 2.1.

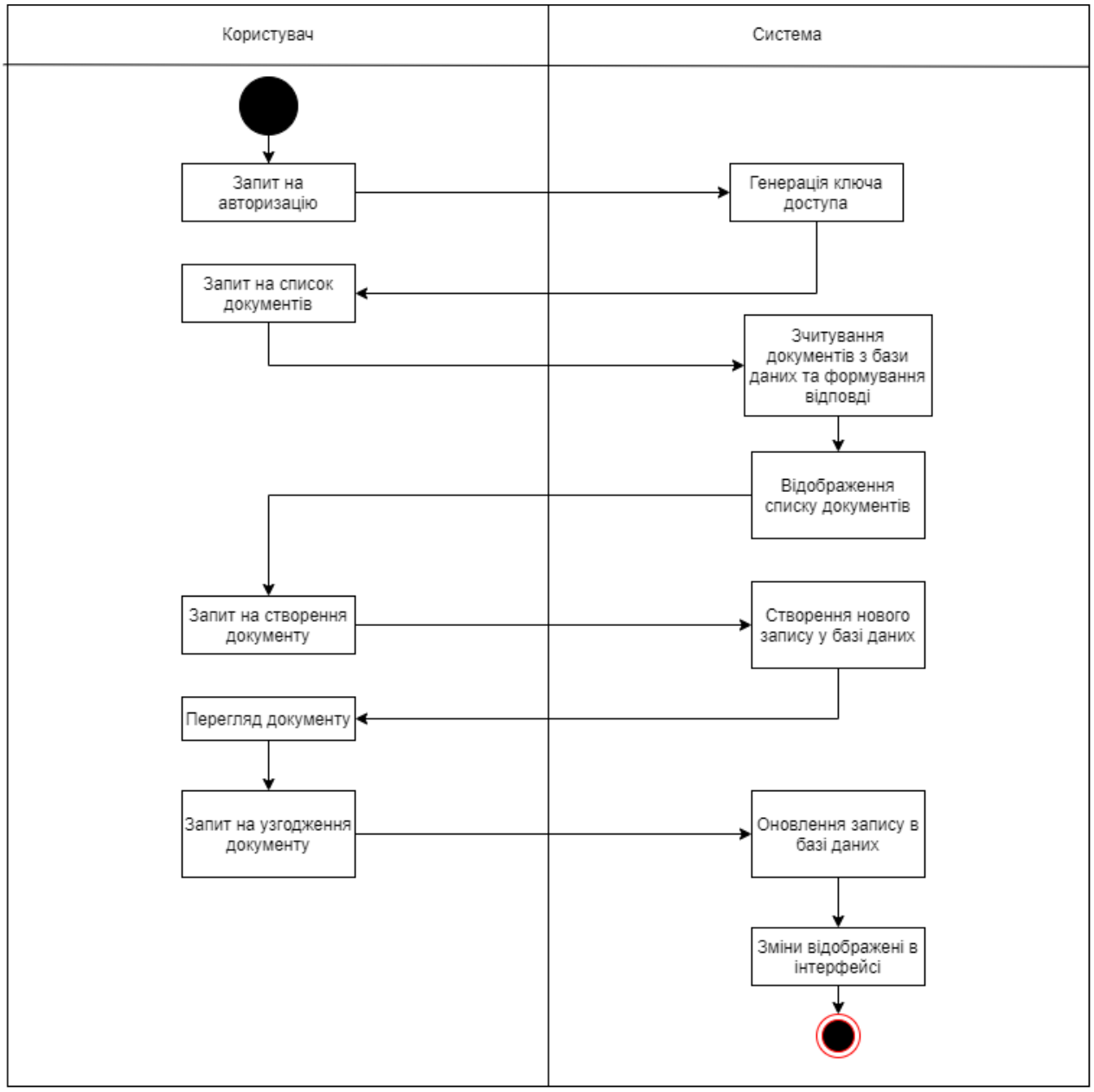

Рисунок 2.1 - Діаграма діяльності

2.3 Опис функціональної моделі

Для опису функціональної моделі використаємо UML-діаграму варіантів використання (Use Case Diagram). Щоб побудувати діаграму спочатку потрібно

визначити акторів, тобто дійових осіб системи. Після чого необхідно описати доступні в межах додатку дії для кожного актору.

Визначимо наступний набір акторів:

- творець;
- рецензент.

Наведемо опис кожного актору.

Творець – це користувач який створює документ та надає доступ до нього іншим користувачам. Частіше всього це будуть студенти.

Рецензент – це користувач якому надають доступ до документу. Частіше всього це будуть викладачі.

В залежності від ситуації один і той же користувач може бути творцем чи рецензентом. Тобто система не регулює жорстко ролі, а надає більше гнучкі правила роботи з нею. Тому набір акторів буде мати однаковий набір функцій:

- авторизація;
- реєстрація;
- перегляд списку документів (як і узгоджені документи, так і ті, що тільки очікують узгодження);
- створення нового документу;
- перегляд документу;
- додавання коментарів;
- завантаження нових версій документу;
- кінцеве узгодження.

Тепер, коли визначено всіх акторів та функції, які їм доступні, можемо визначити дії, які актори можуть виконувати в межах системи та наведемо їх в таблиці 2.1.

Таблиця 2.1 – Дії акторів у системі

Актор	**Варіант використання**	**Опис дії варіанта використання**
Творець	Створення документу	Актор завантажує файл, вводить його назву та опис. Далі йому необхідно надати доступу іншим користувачам та визначити їх рівень доступу.
	Завантаження нової версії документу	Якщо рецензент(и) вказали на те, що документ не відповідає вимогам, то творець має змогу виправити документ у сторонньому програмному забезпечені та завантажити нову версію у цей же документ. При чому стара версія також залишається доступною.
	Додавання коментарів	Актор може залишати коментарі в документі. Ця можливість потрібна, якщо є необхідність відповісти на коментар рецензента.
	Кінцеве узгодження документу	Актор має змогу позначити зі своєї сторони, що сторони досягли згоди щодо тексту документу. Проте без такої ж дії від рецензента документ не буде вважатися узгодженим.

Продовження табл. 2.1

Актор	**Варіант використання**	**Опис дії варіанта використання**
Рецензент	Перегляд документу	Актор має можливість переглянути документ, до якого йому надали доступ.
	Додавання коментарів	Актору надається можливість додати коментар в разі такої необхідності, щоб донести до творця документу, що саме не відповідає вимогам.
	Позначити коментар як виправлений	Актор може позначити коментар як виправлений, якщо після попереднього перегляду документу, творець завантажив нову версію і в ній виправлено недолік.
	Кінцеве узгодження документу	Актор має змогу позначити зі своєї сторони, що сторони досягли згоди щодо тексту документу. Проте без такої ж дії від творця документ не буде вважатися узгодженим.
Творець та рецензент	Авторизація	Актор має змогу авторизуватися використовуючи свою пошту та пароль.
	Реєстрація	Актор має змогу створити новий аккаунт використовуючи свою пошту та пароль.
	Перегляд списку документів	Актор має можливість переглянути увесь список документів, які він створював.
	Видалення документу	Актор має можливість видалити свій документ.

2.4 Аналогічні системи

На даний момент існує декілька сервісів які надають схожі можливості: Google Documents та Deals. Проте у них є недоліки:

- вони не спеціалізуються на вирішені проблеми узгодження документів
- не усі можливості безкоштовні
- їх розробка ніяк не контролюється освітньою спільнотою, а тому є ризик втратити необхідні можливості у майбутньому

2.4.1 Застосунок Google Documents.

Google Documents це текстовий процесор, що дозволяє редагувати текстові документи OpenDocument, Microsoft Word, а також електронні таблиці. Створений за допомогою технології AJAX.

Сервіс працює в рамках браузера, без встановлення на комп'ютер користувача. Документи і таблиці, що створюються користувачем, зберігаються на серверах Google або можуть бути збережені у файл на комп'ютер користувача. Це одна з ключових переваг програми, оскільки доступ до введених даних може здійснюватися з будь-якого комп'ютера, під'єднаного до інтернету. Доступ до особистих документів захищений паролем.

Доступна велика кількість засобів форматування: зміна розміру і стилю шрифту, вибір кольору та оздоблення, створення списків і таблиць, вставка зображень, посилань і спеціальних знаків. Можна робити закладки, коментарі.

Зберігаються документи автоматично, по ходу внесення змін, але кожна правка записується, і можна користуватися функцією скасування і повернення змін так само, як і у звичайному текстовому редакторі. Існує можливість завантажувати файли на сервер і скачувати з нього у різних форматах.

Підтримуються формати: простий текст, HTML, DOC, RTF, OpenDocument, PDF, і декілька графічних форматів. Можна отримати добірка текстів у вигляді файлів HTML в архіві ZIP. Заявлена (але поки не реалізована) підтримка Word Perfect.

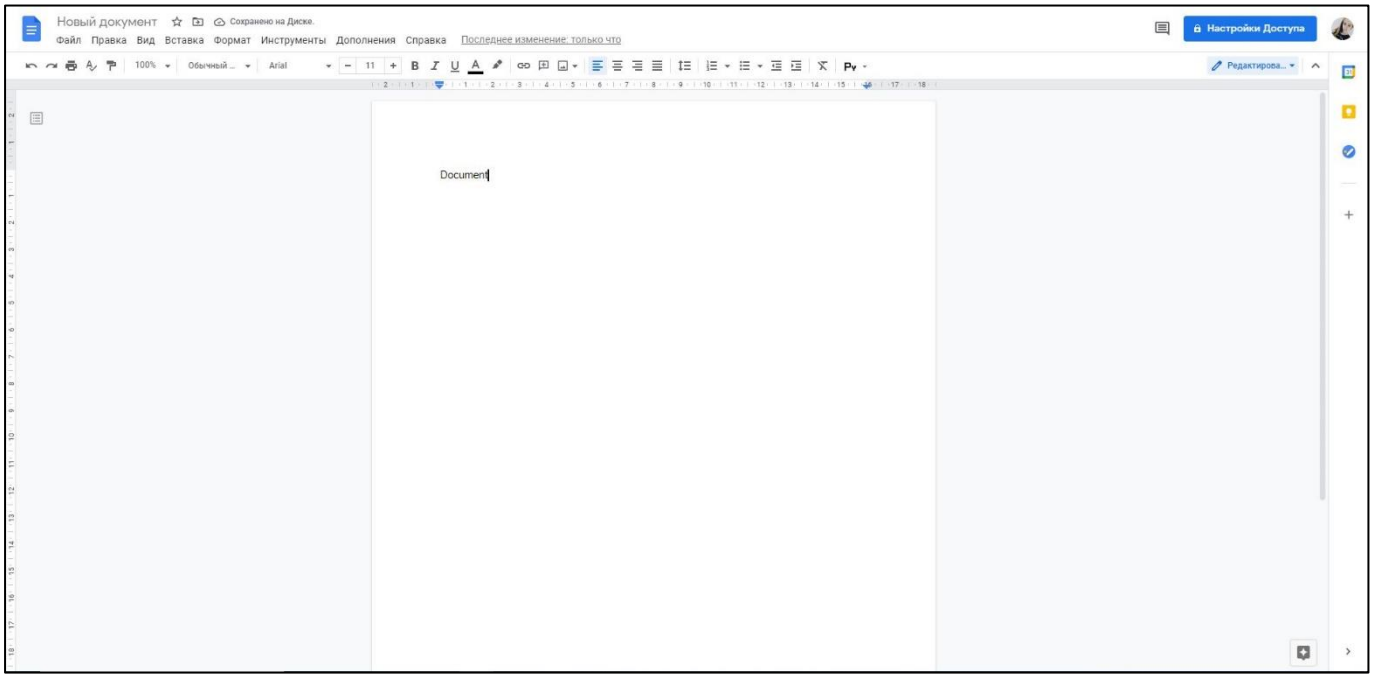

Рисунок 2.1 - Google Documents

Документи Google дозволяють користувачам створювати та редагувати документи в Інтернеті, співпрацюючи з іншими користувачами в режимі реального часу. Оновлення представили функції, що використовують машинне навчання, включаючи "Огляд", пропонуючи результати пошуку на основі вмісту документа та "Елементи дії", що дозволяє користувачам призначати завдання іншим користувачам

Функція чату на бічній панелі дозволяє співавторам обговорювати редагування. Історія версій дозволяє користувачам бачити доповнення, внесені до документа, причому кожен автор відрізняється кольором. Можна порівняти лише сусідні версії, і користувачі не можуть контролювати, як часто редакції зберігаються.

Нижче наведено обмеження розмірів файлів, що вставляються, загальної довжини та розміру документа

– до 1,02 мільйона символів, незалежно від кількості сторінок або розміру шрифту. Файли документів, перетворені у формат .gdoc (Docs)
– не можуть перевищувати 50 МБ

– вставлені зображення не можуть перевищувати 50 МБ і мають бути у форматах .jpg, .png або .gif.

2.4.2 Застосунок Deals.

Deals — онлайн-рішення для погодження та підписання в електронному форматі будь-яких юридично значущих документів

В системі Deals можна використовувати як КЕП, так і електронну печатку. В першу чергу необхідно підписати документ за допомогою КЕП. Це обов'язкова умова для того, щоб документ став легітимним. На вашому носії можуть зберігатися два файли — підписи та печатки. У момент підписання документа у розкривному списку в першу чергу необхідно вибрати саме файл з КЕП. Накладення електронної печатки необов'язкове, тому підписати документ з її допомогою можна тільки після накладення КЕП.

Система надає можливість запросити організацію-партнера, що зареєстрована в системі, а також організацію, яка ще не зареєстрована.

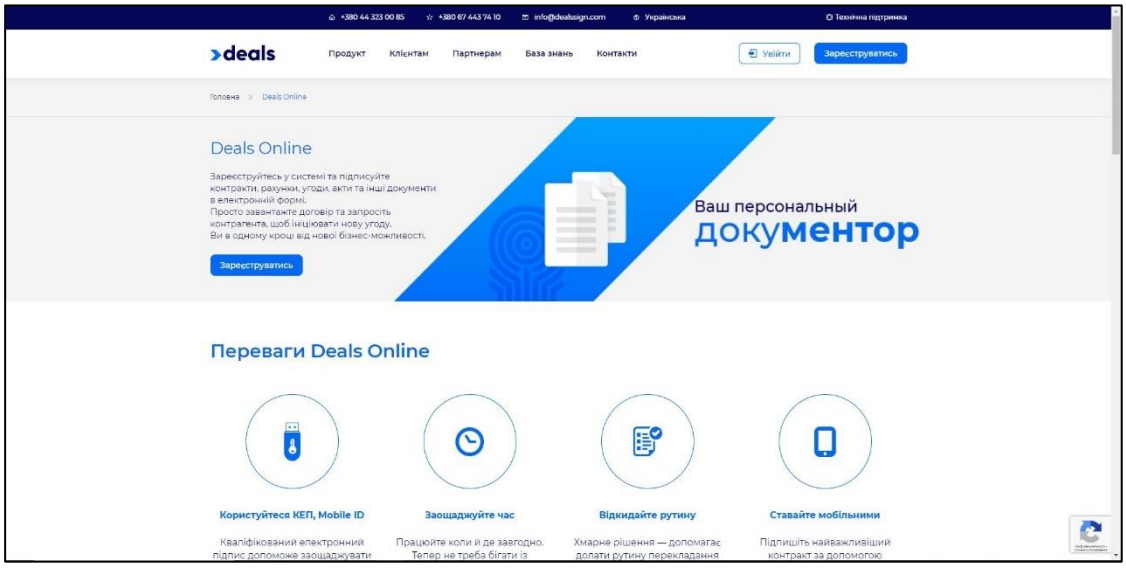

Рисунок 2.2 - Deals

Висновок до розділу

У рамках цього розділу проведено аналіз предметної області та описано процес використання системи, яка розроблена в даному дипломному проекті. Наведено опис

процесу діяльності, розроблено функціональну модель, виділено групи користувачів та функції, які може виконувати кожна група.

Виконано пошук та дослідження аналогів, в ході якого було визначено, що ідентичних аналогів немає, але існую близькі по функціоналу системи. Здійснено порівняння розробленого додатку з цими системами.

3. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Вхідні дані

Вхідні дані можуть надходити в систему тільки від одного джерела, а саме від користувача. Розглянемо детальніше такі дані.

Для входу в систему користувач повинен пройти авторизацію. Для цього потрібно ввести логін та пароль.

Для того, щоб створити новий документ користувач повинен зробити наступні кроки:

- завантажити файл;
- ввести назву документу;
- ввести опис документу (за необхідністю);
- ввести пошту користувача, якому необхідно надати доступ до документу (за необхідністю можна додати декілька користувачів);
- вибрати рівень доступу до документу для кожного користувача.

Користувач системи має можливість:

- створювати документи;
- редагувати документ (завантаживши оновлену версію документу);
- додавати коментарі до документу;
- позначати коментар як виправлений;
- позначати документ як узгоджений.

Робота з системою відбувається за допомогою HTTP-запитів [1]. Розглянемо які саме дані потрібно передавати в запитах для вищеописаних дій.

Для виконання будь-яких дій з системою, користувач повинен завжди передавати в запиті наступні дані:

- свою пошту;
- унікальний ключ доступу до системи.

Вищезгадані дані користувач отримує після авторизації.

Для реєстрації та авторизації профілів користувачів у системі потрібно додатково надсилати в запиті пароль.

3.2 Вихідні дані

Після того, як користувач авторизувався, серверна частина повертає наступні дані унікальний ключ доступу до системи.

Кожен користувач має змогу отримати наступні вихідні дані:

- список документів;
- назва документу;
- опис документу;
- список користувачів, які мають доступ до документу;
- список коментарів для кожного документу;
- позначка про те, що коментар виправлений;
- текст файлу;
- позначка про те, що документ узгоджений.

3.3 Опис структури бази даних

Для розробки системи було використано нереляційну базу даних MongoDB для зберігання даних про користувачів та документи разом з даними про них.

Так як MongoDB документо-орієнтована база даних [3], то замість таблиць і записів з атрибутами будуть розглядатись колекції та документи з полями. У розробленій моделі бази даних створено 2 колекції. Наведемо перелік колекцій та сутностей, що їм відповідають у таблиці 3.1.

Таблиця 3.1 – Перелік колекції та сутностей

№	Назва колекції	Сутність
1	docs	Документ
2	users	Профіль користувача

Колекція docs містить список усіх завантажених документів, users – список користувачів системи.

У таблиці 3.2. детальний опис кожної колекції.

Таблиця 3.2 – Детальний опис таблиць бази даних

Назва колекції	**Назва поля документу**	**Тип даних**	**Опис поля**
docs	_id	ObjectId	Первинний ключ
	name	String	Назва документу
	userId	String	Ідентифікатор користувача
	id	String	Ідентифікатор
	description	String	Інформація про організацію
	permissions	Array<Object>	Список користувачів, що мають доступ до документу та їх рівень доступу
	comments	Array<Object>	Список коментарів
	files	Array<Object>	Список файлів
	approved	Boolean	Позначка про те, чи узгоджений документ
users	_id	ObjectId	Первинний ключ
	id	String	Ідентифікатор користувача в системі
	email	String	Пошта користувача
	password	String	Пароль

Колекція docs містить список коментарів, користувачів з доступом та файлів у полях comments, permissions та files відповідно. Ці дані мають тип Array<Object>, поля яких описано в таблиці 3.3.

Таблиця 3.3 – Опис даних про фото клієнта

Назва документу	**Назва поля**	**Тип даних**	**Опис поля**
comments	text	String	Текст коментаря
	userEmail	String	Пошта користувача, який залишив коментар
	fixed	String	Позначка про те, чи коментар виправлений
permissions	userEmail	String	Пошта користувача, якому надано доступ до документу
	accessType	String	Рівень доступу
files	data	String	Конвертований у формат base64 текст файлу
	version	Number	Версія файлу

На основі описаної вище структури бази даних побудовано ER-діаграму, що зображена на рисунку 3.1

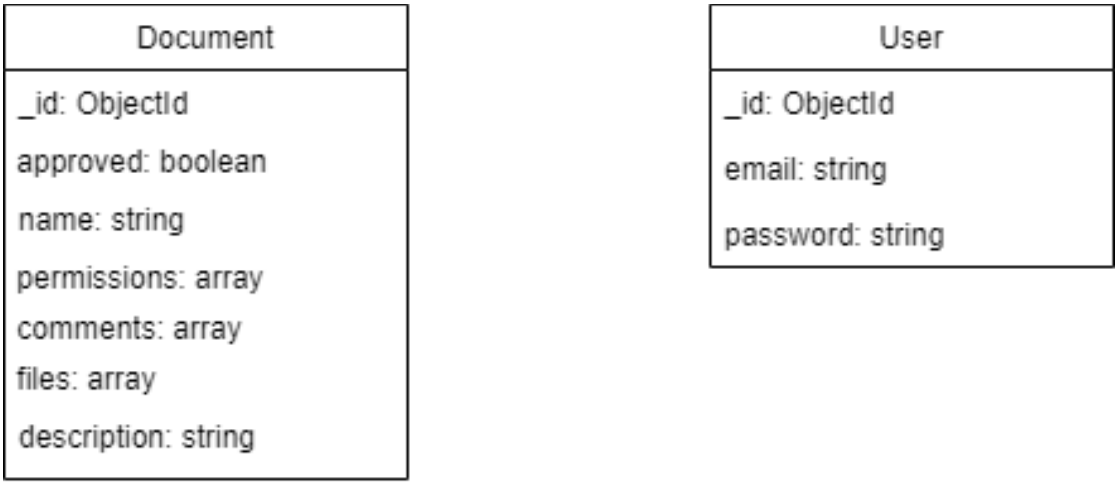

Рисунок 3.1- Структура бази даних

Висновок до розділу

У цьому розділі було сформовано список всіх вхідних та вихідних даних для користувача системи. Також було наведено інформацію щодо бази даних, яка використовувалась при розробці: опис колекцій та документів MongoDB.

4. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для того, щоб розробити програмний продукт було використано наступні технології та засоби розробки:

- асинхронне програмування;
- мова програмування TypeScript;
- середовище розробки WebStrorm;
- мова програмування JavaScript;
- фреймворк для побудови серверної частини Express;
- бібліотека для створення користувацьких інтерфейсів React;
- база даних MongoDB;
- технологія керування версіями git.

Розглянемо ці технології та засоби нижче та обґрунтуємо вибір.

4.1.1 Асинхронне програмування

Як правило, програмний код виконується послідовно, тільки одна конкретна операція відбувається в даний момент часу. Якщо функція залежить від результату виконання іншої функції, то вона повинна дочекатися поки потрібна їй функція не закінчить свою роботу і не поверне результат і до тих пір поки це не відбудеться, виконання програми, по суті, буде зупинено з точки зору користувача.

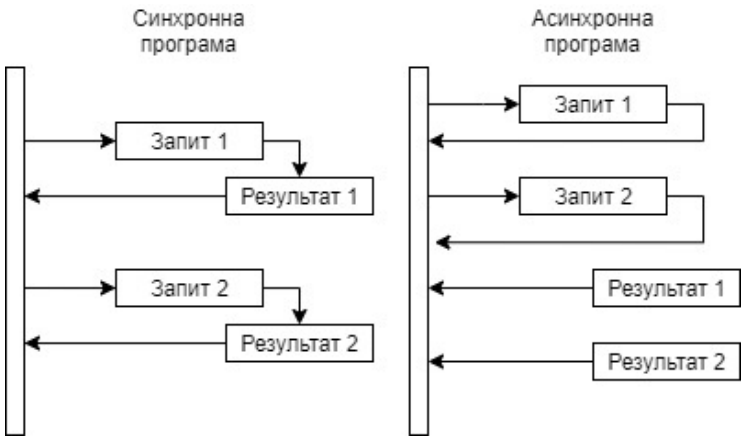

Рисунок 4.1 – Схема роботи синхронної програми та асинхронної

Користувач сучасного ПК, напевно, спостерігав, як курсор змінює свій вигляд і стає "різнобарвним спінером" (у користувачів MacOS). Таким чином операційна система повідомляє - "поточна програма, очікує завершення якогось тривалого процесу в системі і я вирішила повідомити тобі, що б ти не хвилювався"

При проектуванні сучасних програм все більше використовується асинхронне програмування, щоб програма мала можливість виконувати кілька операцій в конкретний момент часу. Як тільки ви почнете використовувати нові, більш потужні можливості API, ви виявите безліч ситуацій, де вирішити потрібну задачу можна тільки асинхронно. Раніше було складно писати асинхронний код. До сих пір, потрібен час, щоб звикнути до такого підходу, але процес став набагато легше. Далі, в цьому розділі, ми будемо глибше дослідити питання, коли ж асинхронний код необхідний і як спроектувати програму, щоб уникнути проблем, описаних вище.

Безліч Web API особливостей тепер використовують асинхронний код, особливо ті, що мають доступ до зовнішніх пристроїв або отримують від них деякі ресурси, такі як отримання файлу з мережі, запит до бази даних і отримання даних з бази, доступ до потокового відео на веб-камері , перегляд дисплея на гарнітурі віртуальної реальності.

Є два типи стилю асинхронного коду, з якими ви зіткнетеся в коді JavaScript, старий метод - колбеки (callbacks) і новіший - проміс (promises). У наступних розділах ми познайомимося з кожним з них.

Асинхронні колбеки - це функції, які визначаються як аргументи при виконанні функції, яка почне виконання коду на задньому тлі. Коли код на задньому фоні завершує свою роботу, він викликає колбек-функцію, сповіщає, що робота зроблена, або сповіщає про труднощі в завершенні роботи. Зворотні виклики - трохи застаріла практика, але вони все ще вживаються в деяких старомодних, але часто використовуваних API.

Проміс - новий стиль написання асинхронного коду, який використовується в сучасних Web API. Хорошим прикладом є fetch API, який сучасніше і ефективніше ніж XMLHttpRequest

Після fetch може слідувати:

– then блок, який повертає новий promise, це означає що ви можете об'єднувати в ланцюжки блоки .then(), таким чином можна виконати кілька асинхронних операцій по порядку, одну за одною;
– catch, який буде запущений якщо який-небудь .then () блок завершиться з помилкою - це аналогічно синхронного try ... catch, помилка стає доступною всередині catch(), що може бути використано для повідомлення користувачу про тип виникла помилки.

У своїй основній формі JavaScript є синхронною, блокуючою, одно-поточною мовою, в якій одночасно може виконуватися тільки одна операція. Але веб-браузери визначають функції і API, які дозволяють нам реєструвати функції, які не повинні виконуватися синхронно, а повинні викликатися асинхронно, коли відбувається будь-яка подія (час, взаємодія користувача з мишею або отримання даних по мережі, наприклад). Це означає, що ви можете дозволити своїм кодом робити кілька речей одночасно, не зупиняючи і не блокуючи основний потік.

Чи будемо ми запускати код синхронно або асинхронно, буде залежати від того, що ми намагаємося зробити.

Є моменти, коли ми хочемо, щоб всі завантажувалося і відбувалося прямо зараз. Наприклад, при застосуванні деяких користувальницьких стилів до веб-сторінці ви хочете, щоб стилі застосовувалися як можна швидше.

Якщо ми виконуємо операцію, яка вимагає часу, наприклад, запит до бази даних і використання отриманих результатів для заповнення шаблонів, краще виштовхнути це з основного потоку і виконати завдання асинхронно. Згодом ви дізнаєтеся, коли має сенс вибирати асинхронний техніку замість синхронної.

4.1.2 Фреймворк Express

Express [6] - це фреймворк, написаний на Node.js.

Як і інші фреймворки, він поширюється безкоштовно і має відкритий вихідний код, який можна знайти на GitHub.

Express - програмний каркас розробки серверної частини веб-застосунків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT.

Він спроектований для створення веб-застосунків і API. Де-факто є стандартним каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на увазі, що він мінімалістичний, але має велику кількість плагінів, що підключаються.

Express є мінімалістичним та гнучким фреймворком для веб-застосунків, побудованих на Node.js, що надає широкий набір функціональності. Маючи в своєму розпорядженні безліч допоміжних HTTP-методів та проміжних обробників, створювати надійні API можна легко і швидко. Express забезпечує тонкий прошарок базової функціональності для веб-застосунків, що не спотворює звичну та зручну функціональність Node.js.

Маршрутизація в Express стосується того, як кінцеві точки програми (URI) реагують на запити клієнтів.

Ви визначаєте маршрутизацію, використовуючи методи об’єкта програми Express, які відповідають методам HTTP. Ці методи маршрутизації визначають функцію зворотного виклику (яку іноді називають “функцією обробника”), що викликається, коли програма отримує запит на вказаний маршрут (кінцеву точку) та метод HTTP. Іншими словами, програма "прослуховує" запити, що відповідають зазначеному маршруту (маршрутам) та методу (методам), і коли виявляє відповідність, вона викликає вказану функцію зворотного виклику.

Насправді методи маршрутизації можуть мати в якості аргументів більше однієї функції зворотного виклику. З кількома функціями зворотного виклику важливо надати next як аргумент функції зворотного виклику, а потім викликати next() у тілі функції, щоб передати контроль наступному зворотному виклику.

4.1.3 База даних MongoDB

MongoDB [9] - документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій.

На відміну від реляційних баз даних, де оперується поняття таблиць та записів, MongoDB використовує концепцію колекцій та документів.

Колекція – це група документів, яка є еквівалентом для таблиць у реляційних базах. Всі документи в колекції можуть зберігати складну структуру даних. Документ можна показати як сховище ключів та значень.

4.1.4 Мова програмування Typescript

TypeScript [11] - це мова з відкритим кодом, яка базується на JavaScript, одному з найбільш використовуваних у світі інструментів, додаючи визначення статичного типу.

Типи дають спосіб описати форму об’єкта, забезпечуючи кращу документацію та дозволяючи TypeScript перевірити, чи правильно працює ваш код. Типи написання можуть бути необов’язковими в TypeScript, оскільки заключення типу дозволяє отримати велику потужність без написання додаткового коду.

Весь дійсний код JavaScript - це також код TypeScript. Існує ймовірність отримати помилки перевірки типу, але це не завадить запустити отриманий JavaScript. Проте є можливість підключити більш жорстку перевірку, що збільшить надійність коду, але й також час на його написання. Також в такому разі програміст буде мати більше контролю.

Код TypeScript перетворюється в код JavaScript за допомогою компілятора TypeScript або Babel. Заключення типу TypeScript означає, що вам не доведеться коментувати свій код, поки ви не захочете отримати більше безпеки. TypeScript є

зворотньо-сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js.

Прийняття TypeScript не є бінарним вибором, ви можете почати з анотації існуючого JavaScript за допомогою JSDoc, потім переключити кілька файлів, що перевіряються TypeScript, і з часом підготувати вашу кодову базу до повного перетворення.

Переваги над JavaScript:

- можливість явного визначення типів (статична типізація),
- підтримка використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах),
- підтримка підключення модулів.

За задумом ці нововведення мають підвищити швидкість розробки, читаність, полегшити рефакторинг і повторне використання коду, здійснювати пошук помилок на етапі розробки та компіляції, а також швидкодію програм.

Планується, що в силу повної зворотної сумісності адаптація наявних застосунків на нову мову програмування може відбуватися поетапно, шляхом поступового визначення типів. Підтримка динамічної типізації зберігається — компілятор TypeScript успішно обробить і не модифікований код на JavaScript.

Основний принцип мови — будь-який код на JavaScript сумісний з TypeScript, тобто в програмах на TypeScript можна використовувати стандартні JavaScript-бібліотеки і раніше створені напрацювання. Більш того, можна залишити наявні JavaScript-проекти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли, які не заважатимуть розробці і прямому використанню проекту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

На момент релізу представлені файли для сприйняття розширеного синтаксису TypeScript для Vim і Emacs, а також плагін для Microsoft Visual Studio.

Одночасно з виходом специфікації розробники підготували файли з деклараціями статичних типів для деяких популярних JavaScript-бібліотек, серед яких jQuery.

4.1.5 Мова програмування Javascript

JavaScript (JS) [8] — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

JavaScript дозволяє вам застосовувати складні речі на веб-сторінці - тобто, коли на веб-сторінці відбувається щось більше, ніж просто її статичне відображення. Наприклад, відображення контенту, який періодично оновлюється, або інтерактивних карт, або анімація 2D / 3D графіки, або прокрутка відео в програвачі, і так далі.

Ядро мови JavaScript складається з певної кількості звичайних можливостей, які дозволяють робити наступне:

- зберігати дані всередині змінних;
- операції над фрагментами текстів (відомими в програмуванні як "рядки");
- запускати код відповідно до визначених подіями що відбуваються на веб-сторінці.

API-інтерфейси браузера вбудовані в ваш веб-браузер і можуть відображати дані з навколишнього комп'ютерного оточення або робити корисні складні речі. наприклад:

- API-інтерфейс DOM (Document Object Model) дозволяє вам маніпулювати HTML і CSS, створювати, видаляти і змінювати HTML, динамічно застосовувати нові стилі до вашої сторінці і так далі;
- API геолокації видає географічну інформацію;
- API Canvas і WebGL дозволяють створювати анімовані 2D і 3D-графіки;

- аудіо та відео API, такі як HTMLMediaElement і WebRTC, дозволяють робити дійсно цікаві речі з мультимедіа, такі як програвання аудіо і відео прямо на веб-сторінці, або захоплювати відео з веб-камери і відображати його на Чужий комп'ютер.

Отже, мова JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- створення одно-сторінкових та прогресивних веб-застосунків (React, AngularJS, Vue.js);
- програмування на боці сервера (Node.js, Express.js);
- стаціонарних застосунків (Electron, NW.js);
- мобільних застосунків (React Native, Cordova);
- сценаріїв в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів тощо.

4.1.6 Бібліотека React

Окрім вказаних вище технологій, під час розробки програмного забезпечення було використано додаткові бібліотеки та технології, створені на їх основі.

Основним фреймворком для створення користувацьких інтерфейсів був React.

React [10] — це декларативна, ефективна і гнучка JavaScript-бібліотека, призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду — “компонентів”.

React спрощує створення інтерактивних інтерфейсів. Потрібно лише описати, як різні частини інтерфейсу виглядають у кожному стані вашого додатку і React ефективно оновить та відрендерить лише потрібні компоненти, коли ваші дані зміняться. Декларативні інтерфейси роблять код більш передбачуваним і його набагато легше налагоджувати.

React дозволяє створювати інкапсульовані компоненти, які керують власним станом, а з них будувати складні інтерфейси. Оскільки логіка компонентів написана

на JavaScript, замість шаблонів, з легкістю можна передавати складні дані у додатку і зберігати стан окремо від DOM.

React також може рендеритись на сервері, використовуючи Node, і приводити в дію мобільні додатки, які використовують React Native.

Властивості передаються в рендерер компоненту, як властивості html тегу. Компонент не може напряму змінювати властивості, що йому передані, але може їх змінювати через callback функції. Такий механізм називають «властивості донизу, події нагору».

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно (diff) зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP.

React використовують не лише для рендерингу HTML в браузері. Наприклад, Facebook має динамічні графіки які рендеряться в теги <canvas>, Netflix та PayPal використовують ізоморфне завантаження для рендерингу ідентичного HTML на сервері та клієнті.

Методи життєвого циклу — це різні методи, які вбудовуються за допомогою ReactJS. Вони дозволяють розробнику обробляти дані в різних точках життєвого циклу програми React. Наприклад:

- shouldComponentUpdate — це метод життєвого циклу, який каже Javascript оновити компонент, використовуючи логічні змінні.
- componentDidMount — це метод життєвого циклу, подібний до компонента WillMount, за винятком того, що він працює після методу render, і може використовуватися для додавання JSON-даних, а також для визначення

властивостей та станів.

Найважливішим методом життєвого циклу є *render*, необхідним у будь-якому компоненті. Метод render повертає опис того, що ви хочете бачити на екрані. React приймає цей опис і відтворює результат. Зокрема, render повертає React-елемент — полегшену версію того, що треба відрендерити. Більшість React-розробників користується спеціальним синтаксисом під назвою “JSX”, який спрощує написання цих конструкцій

4.2 Вимоги до технічного забезпечення

Для запуску та функціонування розробленої системи потрібно забезпечити наявність апаратного забезпечення з наступними мінімальними вимогами:

- оперативна пам’ять – 1 ГБ;
- жорсткий диск – 25 ГБ;
- процесор – двох-ядерний процесор з частотою 2.1 ГЦ;
- будь-яка операційна система.

Для можливості працювати з системою (авторизуватися, створювати нові акаунти, створювати, переглядати та редагувати документи) потрібно мати доступ до мережі інтернет.

4.3 Архітектура програмного забезпечення

Розроблений додаток має клієнт-серверну архітектуру. Клієнт написаний за допомогою мови програмування TypeScript та бібліотеки для створення інтерфейсів користувача React. Серверна частина працює у середовищі Node.js, а також для її створення було задіяно фреймворк Express. Сервер має зовнішні з'єднання з базою даних MongoDB. На стороні серверу реалізовано REST API, що необхідно для взаємодії клієнтської та серверної частин.

Клієнт-серверна архітектура це обчислювальна модель, в якій сервер розміщує та керує більшістю ресурсів та послуг, які використовує клієнт [4]. У цьому типі

архітектури розглядається один або декілька клієнтських комп’ютерів, які підключені до серверу через мережеве або інтернет з’єднання.

Перевагами цієї архітектури є:

- відсутність дублювання коду з серверу на клієнті;
- знижені вимоги до клієнтських комп’ютерів, так як всі обрахунки проводяться сервером;
- збереження даних на сервері, який у більшості випадків більш захищений.

Одним з недоліків обраної архітектури є те, що коли на сервері відбувається якийсь збій, який приводить його в неробочий стан, то вся обчислювальна мережа стає недоступною.

Для того щоб користувач міг взаємодіяти з розробленим додатком було розроблено REST(Representational State Transfer) API [5]. REST API – це архітектурний стиль для розподілених систем, який слідує наступним принципам:

- клієнт-сервер – відокремлення користувацького інтерфейсу від обчислень та збереження даних на сервері;
- не збереження стану – кожен запит не повинен використовувати будь-який стан, збережений на сервері, а повинен містити всю потрібну інформацію в самому запиті;
- кешованість – сервер має можливість зберігати дані в кеш та використовувати їх для нових запитів;
- уніфікований інтерфейс – застосування принципу загальності розробки програмного забезпечення;
- багаторівнева система – архітектура складається з багатьох шарів.

4.3.1 Діаграма класів

Під час написання системи використовувались функціональна та об’єктно-орієнтована парадигми програмування.

Об’єктно-орієнтована парадигма програмування використовується для:

- написання класів, які реалізують моделі об’єктів з бази даних;

– класів, які реалізують валідацію даних, отриманих з запитів API;

– класів, які працюють з HTTP запитами.

Функціональна парадигма програмування використовується для:

– написання функцій, які реалізують REST API;

– реалізації бізнес логіки програмного продукту;

– створення компонентів користувацького інтерфейсу.

Опис специфікації функцій наведено у пункті 4.3.4 пояснювальної записки.

Перелік створених класів та їх опис наведено у таблиці 4.1.

Таблиця 4.1 – Опис класів

Клас	**Опис**
User	Клас, який описує модель даних користувача й реалізує функціонал роботи з об’єктом бази даних.
Document	Клас, який описує модель даних документу й реалізує функціонал роботи з об’єктом бази даних.
Клас	**Опис**
Comment	Клас, який описує модель даних коментаря й реалізує функціонал роботи з об’єктом бази даних.
File	Клас, який описує модель даних файлу й реалізує функціонал роботи з об’єктом бази даних.
Permission	Клас, який описує модель даних користувача з доступом до документу й реалізує функціонал роботи з об’єктом бази даних.

DBService	Клас, який реалізує роботу з базою даних на сервері.
AuthApiService	Робота с HTTP запитами на авторизацію.
EncryptService	Клас, який відповідає за шифрування даних.
AuthStorageService	Клас, який відповідає за збереження даних авторизації на клієнті.
ValidatorService	Клас, який реалізує перевірку даних для авторизації.
CreateDocApiService	Робота с HTTP запитами на створення документу.
DocListApiService	Робота с HTTP запитами на отримання списку документів.
DocViewerApiService	Робота с HTTP запитами на перегляд та редагування документу.
LocalStorageService	Робота з localStorage.
FetchHttpService	Робота с HTTP запитами
Expect	Перевірка даних, які надходять від серверної частини

У таблиці 4.2 наведено опис атрибутів класів.

Таблиця 4.2 – Опис атрибутів класів

Клас	**Назва поля**	**Тип даних**	**Опис поля**
Document	name	String	Назва документу
	userId	String	Ідентифікатор користувача
	id	String	Ідентифікатор
	description	String	Інформація про документ
	permissions	Array<Permission>	Список користувачів

	comments	Array<Comment>	Список коментарів
	files	Array<File>	Список файлів
	approved	Boolean	Позначка про те, чи узгоджений документ
User	id	String	Ідентифікатор
	email	String	Пошта користувача
	password	String	Пароль
Comment	text	String	Текст коментаря
	userEmail	String	Пошта користувача
	fixed	Boolean	Позначка про те, чи коментар виправлений
Permission	userEmail	String	Пошта користувача
	accessType	String	Рівень доступу
File	data	String	Конвертований у формат base64 текст файлу
Клас	**Назва поля**	**Тип даних**	**Опис поля**
File	version	Number	Версія файлу
AuthApiService	http	Http	Робота с HTTP запитами
CreateDocApiService	http	Http	Робота с HTTP запитами
DocListApiService	http	Http	Робота с HTTP запитами
DocViewerApiService	http	Http	Робота с HTTP запитами
AuthStorageService	storage	Storage	Робота з localStorage.

У таблиці 4.3 наведено опис методів класів.

Таблиця 4.3 – Опис методів класів

Клас	**Метод**	**Опис**

DBService	getDocs	Зчитування списку документів з бази даних
	getDoc	Зчитування документу з бази даних
	createDoc	Занесення документ до бази даних
	updateDoc	Оновлення документу в базі даних
	removeDoc	Видалення документу з бази даних
	getUser	Зчитування користувача з бази даних
	createUser	Занесення користувача до бази даних
	_generateId	Створення унікального ідентифікатора
AuthApiService	signIn	HTTP запит на авторизацію
	signUp	HTTP запит на реєстрацію
EncryptService	run	Шифрування даних
Клас	**Метод**	**Опис**
SharedStorageService	saveId	Збереження ідентифікатора користувача на клієнті
	getId	Отримання ідентифікатора користувача на клієнті
	saveUser	Збереження даних про користувача на клієнті
	getUser	Отримання даних про користувача на клієнті
ValidatorService	checkEmail	Перевірка пошти користувача
DocListApiService	getDocs	HTTP запит на отримання документів
	removeDoc	HTTP запит на видалення документу
CreateDocApiService	createDoc	HTTP запит на створення документу
DocViewerApiService	getDoc	HTTP запит на отримання документу

	updateDoc	HTTP запит на оновлення документу
LocalStorageService	save	Зберегти запис у сховищі
	get	Отримати запис зі сховища
	remove	Видалити запис у сховищі
	clear	Повністю очистити сховище даних
FetchHttpService	get	Виконання GET запиту
	post	Виконання POST запиту
	put	Виконання PUT запиту
	patch	Виконання PATCH запиту
	remove	Виконання DELETE запиту
	processResp	Обробка відповіді сервера
Expect	json	Очікування на те, що сервер поверне JSON
	string	Очікування на те, що сервер поверне строку
	whatever	Очікування на те, що сервер поверне неважливо що

Зв’язок між класами показано на UML-діаграмі, яку наведено на рисунку 4.2.

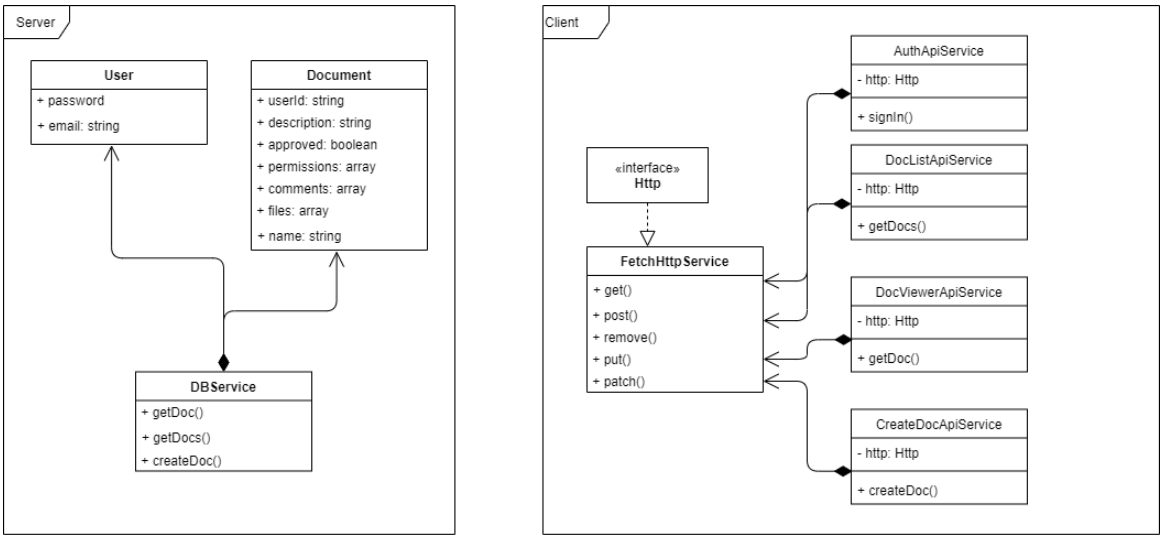

Рисунок 4.2 - Діаграма класів

4.3.3 Діаграма компонентів

Умовно розроблену систему можна поділити на два компоненти [14]:

– клієнтська частина;

– серверна частина;

Розглянемо детальніше кожен з цих компонентів та набори пакетів, які вони в собі містять.

Клієнтська частина

Цей компонент відповідає за клієнтську частину системи. Він дає користувача зручний інтерфейс для взаємодії з системою. Даний компонент містить в собі наступний набір пакетів:

– styles – загальні стилі;

– modules – містить модулі для створення інтерфейсу користувача;

– assets – містить усі необхідні файли, такі як картинки, наприклад;

– services – містить функціонал для роботи з серверною частиною;

– tools – містить набір допоміжних функцій.

Серверна частина

Ця частина відповідає за серверну частину системи. У ньому реалізовано функціонал для обробки запитів від клієнтської частини та REST API для того, щоб користувачі могли користуватись усіма функціями системи. Даний компонент містить в собі наступний набір пакетів:

– controllers – містить функціонал, що реалізує бізнес-логіку системи;

– database – містить класи, які відповідають за роботу з базою даних;

– routes – містить функції, що обробляють запити, які надходять до компоненту.

Також даний компонент має зовнішній зв’язок з базою даних, яка містить дані про всі створені документи та користувачів.

Структурну схему додатку наведено на рисунку 4.3.

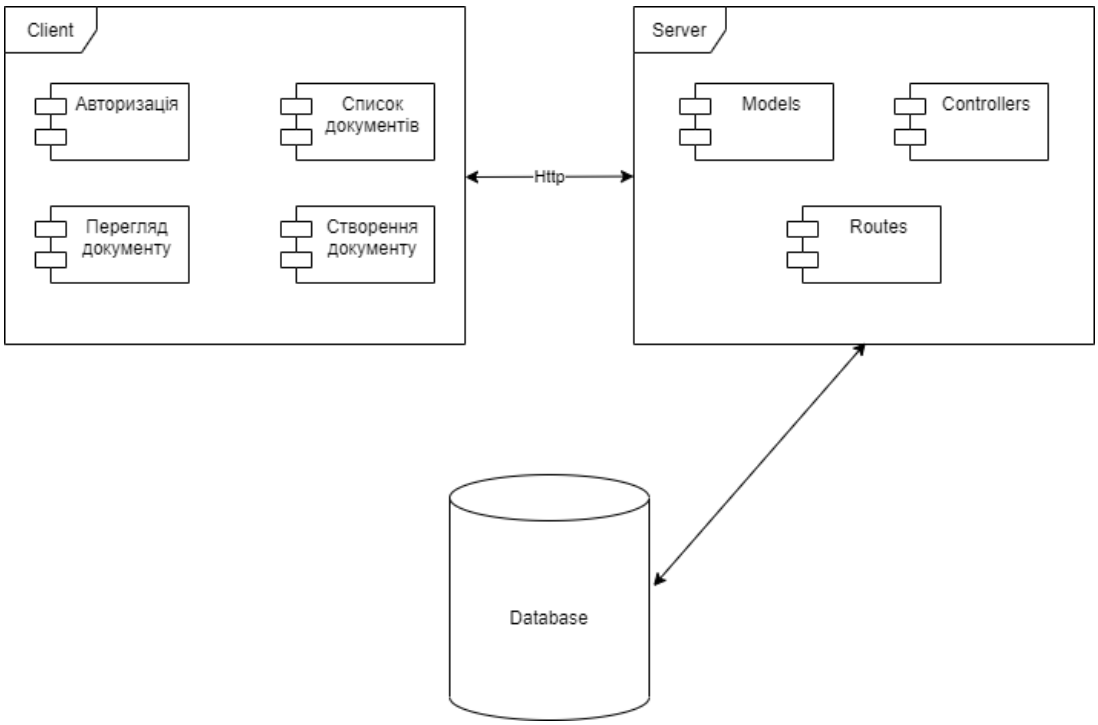

Рисунок 4.3 – Структурна схема

4.3.4 Специфікація функцій

Проведемо огляд доступних методів (далі – endpoint) цього API, функцій, які за них відповідають, та структуру HTTP-запиту до цього endpoint'у. Назву endpoint'у будемо представляти у форматі:

Endpoint <тип HTTP-запиту> <шлях до endpoint'у>

Endpoint GET /docs

Оброблюється функцією *getDocs(request)*, яка приймає на вхід запит та повертає у відповідь список усіх документів користувача. Нижче наведено структуру HTTP-запиту:

```
GET /docs HTTP/1.1

Content-Type: application/json

userEmail: <пошта користувача>
```

Endpoint GET /docs/{docId}

Оброблюється функцією *getDoc(request)*, яка приймає на вхід запит та повертає у відповідь дані про документі, ідентифікатор якого відповідає значенню *docId* у запиті. У випадку, якщо профілю з таким ідентифікатором не існує, то відповідь

серверу міститиме відповідне повідомлення. Нижче наведено структуру HTTP-запиту:

```
GET /docs/<ідентифікатор документу> HTTP/1.1

Content-Type: application/json

userEmail: <пошта користувача>
```

Endpoint POST /docs

Оброблюється функцією *createDoc(request)*, яка приймає на вхід запит з даними про документ та створює відповідний запис у базу даних. Нижче наведено структуру HTTP-запиту:

```
POST /docs HTTP/1.1

Content-Type: application/json

token: <ключ доступу організації>
```

Endpoint PUT /docs

Оброблюється функцією *updateDoc(request)*, яка приймає на вхід запит з даними про документ та оновлює відповідний запис у базу даних, для якого збігається ідентифікатор. Нижче наведено структуру HTTP-запиту:

```
PUT /docs HTTP/1.1

Content-Type: application/json

<дані про документу>
```

Endpoint DELETE /docs/{docId}

Оброблюється функцією *deleteDoc (request)*, яка приймає на вхід запит та повертає у відповідь повідомлення про видалення документу, ідентифікатор якого відповідає значенню *docId* у запиті. Нижче наведено структуру HTTP-запиту:

```
DELETE /docs/<ідентифікатор документу> HTTP/1.1

Content-Type: application/json
```

Endpoint POST /sign-in

Оброблюється функцією *getUser(request)*, яка приймає на вхід запит з ключом доступу та повертає у відповідь дані про користувача. Нижче наведено структуру HTTP-запиту:

```
POST /sign-in HTTP/1.1
```

```
Content-Type: application/json

token: <ключ доступу>
```

Endpoint POST /sign-up

Оброблюється функцією *createUser(request)*, яка приймає на вхід запит з даними про користувача та створює відповідний запис у базу даних. Нижче наведено структуру HTTP-запиту:

```
POST /sign-up HTTP/1.1

Content-Type: application/json

<дані про користувача>
```

Можливі варіанти відповідей та структура відповідей описані в керівництві користувача (пункт 5.1.2).

Висновок до розділу

У цьому розділі було описано технології та засоби розробки програмного продукту для дипломного проекту, обґрунтовано їх вибір та розглянуто переваги та недоліки.

Описано архітектуру системи та кожну архітектурну одиницю окремо. Наведено опис класів, компонентів системи та відображено на UML-діаграмах. Також побудовано UML-діаграму послідовності для процесу створення документу.

5. ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Система має тільки одну групу користувачів. Нижче наведено керівництво користувача.

Взаємодія користувача з серверною частиною відбується через веб-додаток, а далі REST API та HTTP-запити. Користувач має можливість:

- авторизуватися;
- реєструватися;
- переглянути список документів;
- створити документ;
- переглянути документ;
- додати коментар;
- завантажити нову версію файлу.

У відповіді на кожен запис повертається код відповіді та дані у форматі JSON.

Для початку роботи користувач повинен авторизуватись в систему. На рисунку 5.1 зображено форму входу.

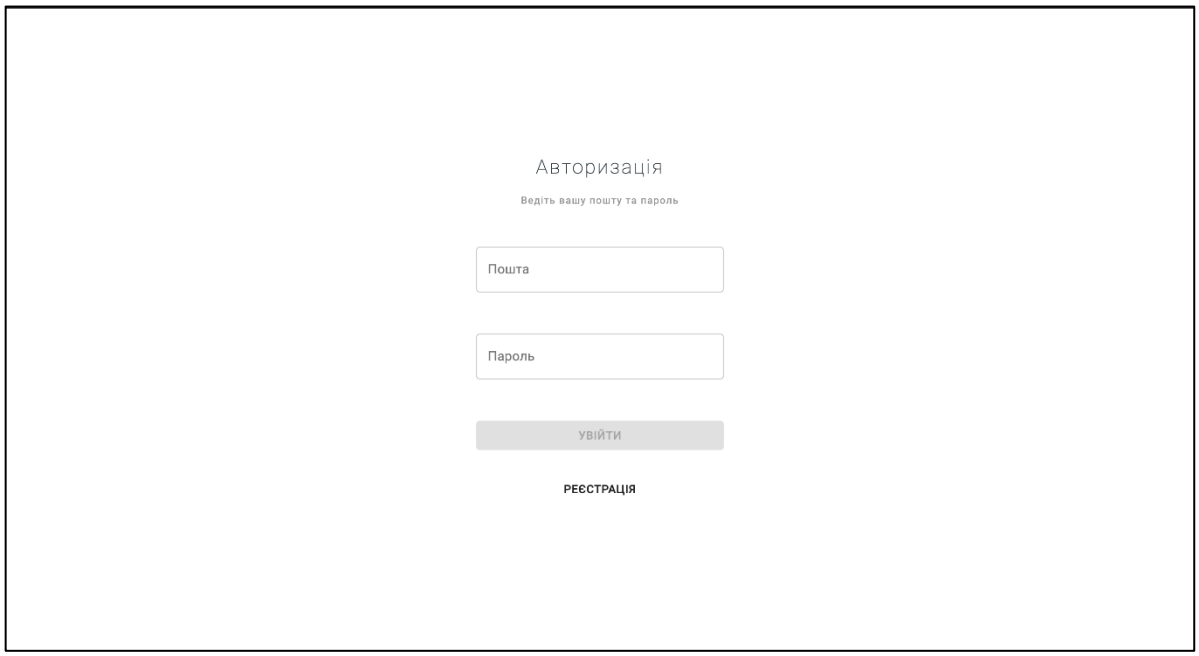

Рисунок 5.1 – Сторінка входу в систему

Якщо користувач не має свого облікового запису в системі, то його необхідно створити. На рисунку 5.2 зображено форму реєстрації.

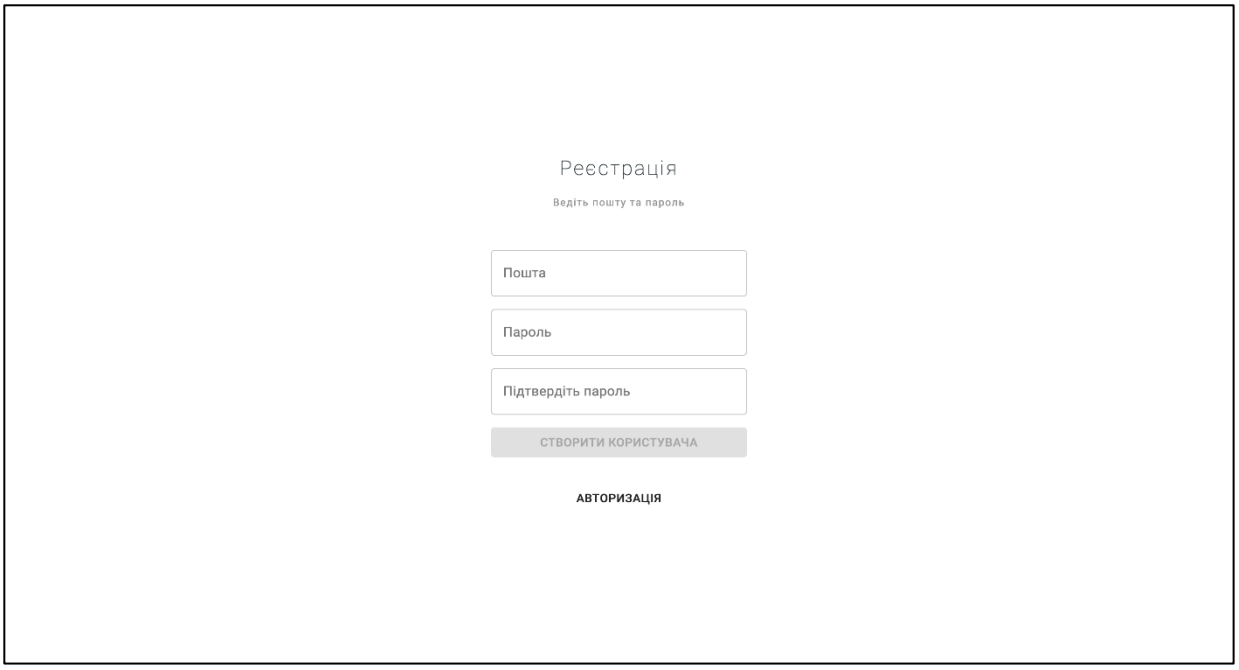

Рисунок 5.2 - Форма реєстрації

Після цього користувач отримує доступ до користування функціями системи. Безпосередньо після авторизації користувач потрапляє на сторінку з списком його документів, що зображена на рисунку 5.2.

Стартовою сторінкою застосунку є список документів: як тих що зараз у роботі, так і тих що все були узгоджені. Звідси користувач може перейти до наступних модулів програми: авторизація, створення документу та перегляд документу.

Також на цій сторонці користувач має змогу:

- дізнатися ім’я та опис документу;
- дізнатися про те, чи угоджений документ;
- видалити документу;

Важливо зазначити, що перед видаленням документу з’явиться вікно, до користувача попросять підтвердити дану дію.

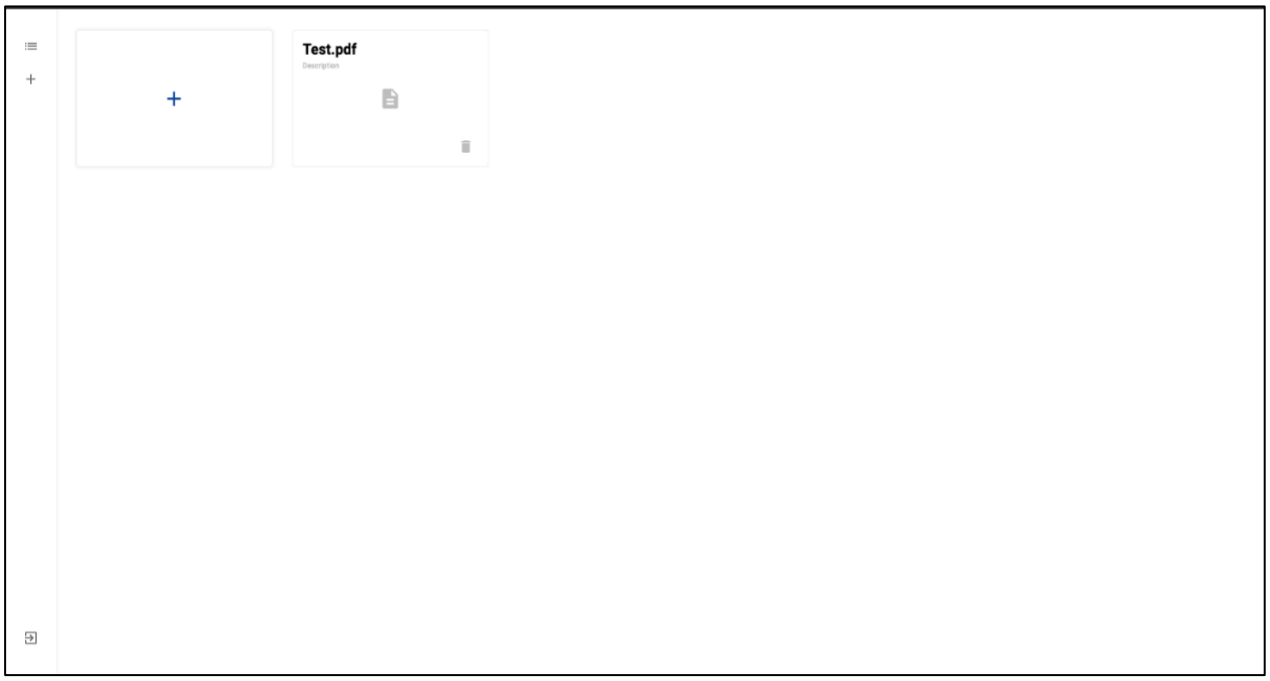

Рисунок 5.3 – Сторінка з списком документів

Тепер перейдемо до створення документу.

Цей процес складається з трьох кроків:

– завантаження файлу (Рисунок 5.4);
– введення основної інформації про документ, а саме назву та опис, якщо є така необхідність (Рисунок 5.5);
– надання доступу іншим користувачам, при цьому тут є можливість обрати наступні рівні доступу до вашого документу: повний доступ, редагування та тільки перегляд (Рисунок 5.6).

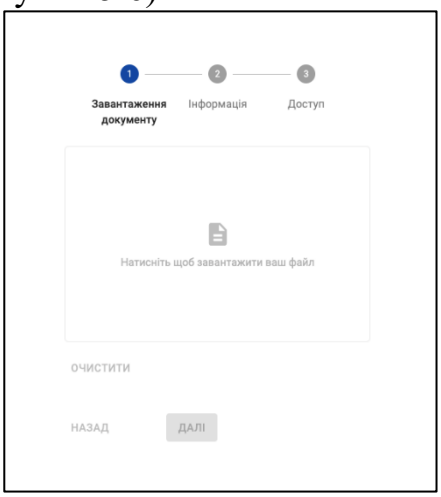

Рисунок 5.4 - Завантаження файлу

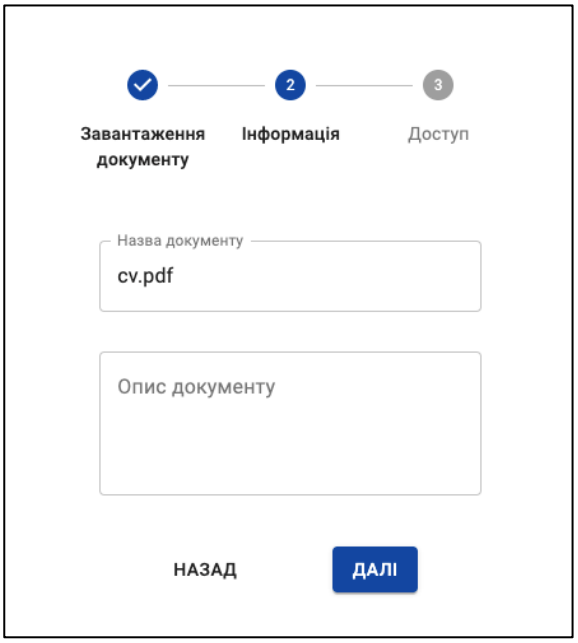

Рисунок 5.5 - Ведення інформації про документ

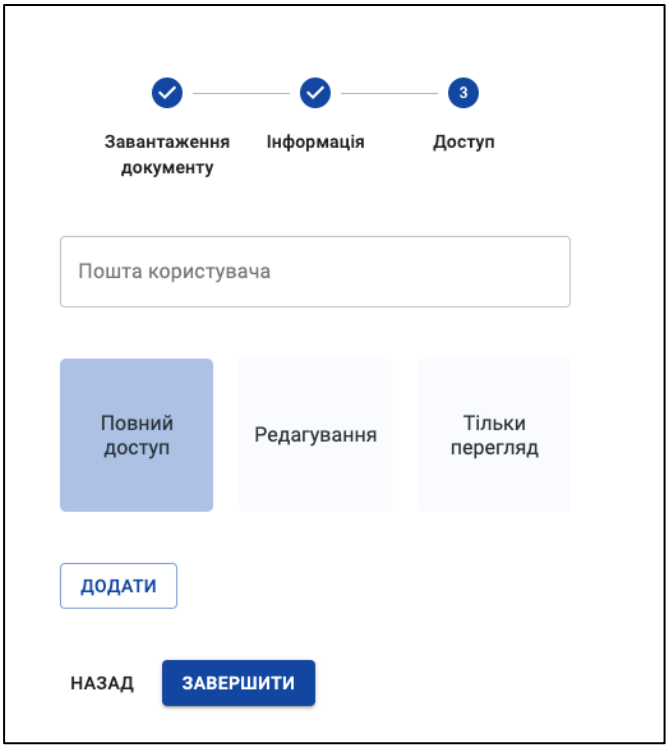

Рисунок 5.6 - Надання доступу іншим користувачам

Найважливішою частиною застосунку є перегляд документу. Тут користувач маж наступні можливості:

– перегляд різних версій документу;

– перегляд назви документу та його опису;
– перегляд та додавання коментарів;
– позначити коментар як виправлений;
– завантаження нових версій;
– перегляд списку користувачів, яким надано доступ;
– узгодження.

Рисунок 5.5 - Перегляд документу

На рисунку 5.5 можна побачити користувацький інтерфейс, який і надає змогу користувачу виконувати вищеперераховані операції.

5.2 Випробування програмного продукту

Метою випробувань розроблюваного програмного продукту являється перевірка функцій Системи на відповідність вимогам технічного завдання.

5.2.1 Результати випробувань

Під час тестування було перевірено функціональність користувача.

У таблицях 5.1-5.8 наведено перелік випробувань функціоналу.

Таблиця 5.1 – Тестування введення логіну та паролю

Мета тесту	**Перевірка функції «Авторизація»**
Початковий стан системи	Відкрита сторінка «Авторизація».
Вхідні дані	Логін та пароль користувача.
Схема проведення тесту	Ввести логін та пароль користувача у відповідні поля.
Очікуваний результат	Вхід в систему виконано, відкрито сторінку «Список документів».
Стан системи після проведення випробування	Вхід в систему виконано, відкрито сторінку «Список документів».

Таблиця 5.2 – Тестування додавання нового документу

Мета тесту	**Перевірка функції «Додавання документу»**
Початковий стан	Відкрита сторінка «Додати документу».
Вхідні дані	Файл, його назва, опис документу та список користувачів, яким надано доступ до документу разом з рівнем доступу.
Схема проведення тесту	Завантажити файл та ввести відповідні дані про документ у відповідні поля на сторінці
Очікуваний результат	Додано новий документ, згенеровано унікальний ідентифікатор.
Стан системи після проведення випробування	Додано новий документ, згенеровано унікальний ідентифікатор, відкрито сторінку «Список документів», де відображається щойно доданий документ.

Таблиця 5.3 – Тестування завантаження нової версії файлу

Мета тесту	**Перевірка функції «Завантаження нової версії файлу»**
Початковий стан системи	Відкрита сторінка «Перегляд документу», яка містить дані про документ та файл.
Вхідні дані	Новий файл.
Схема проведення тесту	Завантажити нову версію файлу.
Очікуваний результат	Новий файл завантажено, у списку версій документу відображено дві версії доступні для перегляду.
Стан системи після проведення випробування	Новий файл завантажено, у списку версій документу відображено дві версії доступні для перегляду.

Таблиця 5.4 – Тестування видалення документу

Мета тесту	**Перевірка функції «Видалення документу»**
Початковий стан	Відкрита сторінка «Список документів».
Вхідні дані	Відсутні.
Схема проведення тесту	Натиснути кнопку «Видалити» на карточці документу.
Очікуваний результат	Видалено дані про документ, відкрито сторінку «Список документів», де відсутні дані про видалений документ.
Стан системи після проведення випробування	Видалено дані про документ, відкрито сторінку «Список документів», де відсутні дані про видалений документ.

Таблиця 5.5 – Тестування узгодження документу

Мета тесту	**Перевірка функції «Узгодження документу»**
Початковий стан системи	Відкрита сторінка «Перегляд документу», яка містить дані про документ та файл, усі інші користувачі схвалили документу.
Вхідні дані	Відсутні.
Схема проведення тесту	Натиснути кнопку «Узгодити» у правій частині сторінки.
Очікуваний результат	Документ узгоджено, що відображено в інтерфейсі користувача.
Стан системи після проведення випробування	Документ узгоджено, що відображено в інтерфейсі користувача. Відкрита сторінка «Перегляд документу».

Таблиця 5.6 – Тестування додавання коментаря

Мета тесту	**Перевірка функції «Додавання коментаря»**
Початковий стан системи	Відкрита сторінка «Перегляд документу», яка містить дані про документ та файл.
Вхідні дані	Текст коментаря.
Схема проведення тесту	Ввести текст коментаря у відповідне поле та натиснути кнопку «Додати» у правій частині сторінки.
Очікуваний результат	Коментар додано. Відкрита сторінка «Перегляд документу», у списку коментарів відображено щойно доданий коментар.
Стан системи після проведення випробування	Коментар додано. Відкрита сторінка «Перегляд документу», у списку коментарів відображено щойно доданий коментар.

Таблиця 5.7 – Тестування позначення коментаря як виправленого

Мета тесту	**Перевірка функції «Позначення коментаря як виправленого»**
Початковий стан системи	Відкрита сторінка «Перегляд документу», яка містить дані про документ та файл.
Вхідні дані	Відсутні.
Схема проведення тесту	Натиснути кнопку «Позначити» у карточці коментаря.
Очікуваний результат	Коментар позначено як виправлений, що відображено в інтерфейсі користувача. Відкрита сторінка «Перегляд документу».
Стан системи після проведення випробування	Коментар позначено як виправлений, що відображено в інтерфейсі користувача. Відкрита сторінка «Перегляд документу».

Таблиця 5.8 – Тестування реєстрації

Мета тесту	**Перевірка функції «Реєстрація»**
Початковий стан системи	Відкрита сторінка «Реєстрація».
Вхідні дані	Логін та пароль користувача.
Схема проведення тесту	Ввести логін та пароль користувача у відповідні поля. Повторити введення пароля у відповідне поле.
Очікуваний результат	Вхід в систему виконано, створено новий аккаунт, відкрито сторінку «Список документів».
Стан системи після проведення випробування	Вхід в систему виконано, створено новий аккаунт, відкрито сторінку «Список документів».

Висновок до розділу

У цьому розділі надано інструкція користувача. В інструкції описані дії, які мають можливість виконувати користувачі.

Також були проведені функціональні випробування програмного продукту, які підтвердили його працездатність.

ВИСНОВКИ

На початку роботи було сформульовано проблему, яка з’явилася в освітньому процесі, а саме необхідності в обміні та узгоджені документів між учасниками цього процесі. Далі необхідно було спроектувати систему, як б допомогла в цьому питанні. На даному етапі були прийняті наступні рішення:

- використати клієнт-серверну архітектуру;
- поділити клієнтську частину на 4 модулі;
- обрати усі необхідні технології, які дадуть змогу створити систему;
- сформувати дві основні моделі даних: Користувач та Документ.

В результаті було досягнуто наступних результатів:

- клієнтська та серверна частини можуть розвиватися незалежно один від одного, що додасть системі гнучкості та надійності;
- завдяки вдало обраним технологіям швидкість розробки зросла, як і надійність системі, адже було використано інструменті, які вже перевірені тисячами інших розробників;
- клієнтська частина була поділена на менші модулі, заради того, щоб ця частина не була надто громіздкою, як в плані користувацького інтерфейсу, так і самого коду програми.

Наступним етапом було розробка самої системи, яка б відповідала поставленим вимогам. Отже, тут можна відзначити наступні досягнення:

- додаток має простий та зрозумілий інтерфейс;
- швидкість роботи програми ніяким чином не впливає на користувацький досвід;
- були реалізовані усі задумані функції, а саме: авторизація, реєстрація, перегляд списку документів, створення нового документу з можливістю додати необхідну інформацію та зробити усе налаштування, переглянути документ та усі дані про нього, залишити свій коментар або ж позначити його як виправлений, завантажити нову версію файлу у разі необхідності, позначити документ як узгоджений та, в кінці кінців, видалити документ;

- створена серверна частина, яка в майбутньому зможе з легкість змінюватися та розширюватися;
- створена база даних, яка не ускладнює усю систему, але й відповідає усім вимогам.

Також система була протестована на різних наборах даних для усіх можливих випадків дії користувача. На цьому етапі було усунуті знайдені недоліки та покращена робота системи.

В результаті було отримано систему, яка вирішує проблему дистанційного та швидкого узгодження текстів документів та допомагає учасникам освітнього процесу економити свій час та бути ще більш ефективними.

ПЕРЕЛІК ПОСИЛАНЬ

1. Aviani G. HTTP and everything you need to know about it [Електронний ресурс] / Goran Aviani. – 2018. – Режим доступу до ресурсу: https://medium.com/faun/http-and-everything-you-need-to-know-about-it-8273bc224491.
2. Raimi K. Illustrated: 10 CNN Architectures [Електронний ресурс] / Karim Raimi. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d.
3. Редмонд Э. Семь баз данных за семь недель / Э. Редмонд, Д. Р. Уилсон. – Москва: ДМК Пресс, 2013. – 384 с.
4. Змерзлий І. Клієнт-серверна архітектура та ролі серверів [Електронний ресурс] / Іван Змерзлий. – 2017. – Режим доступу до ресурсу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229.
5. Ivashchenko A. REST: простым языком [Електронний ресурс] / Andriy Ivashchenko. – 2019. – Режим доступу до ресурсу: https://medium.com/@andr.ivas12/rest-простым-языком-90a0bca0bc78.
6. Фреймворк Express [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/ru/
7. Мова розмітки HTML [Електронний ресурс] – Режим доступу до ресурсу: https://devdocs.io/html/
8. Мова програмування Javascript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/
9. База даних MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mongodb.com/
10. Бібліотека для створення користувацьких інтерфейсів React [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/docs/getting-started.html
11. Мова програмування Typescript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/

12. Head First. Патерни проектування / Ерік Фрімен, Елізабет Робсон, Кеті Сьерра і Берт Бейтс; пер. з англ. Г. Якубовська – Харків : ВД «Фабула», 2020. – 672 с.
13. Чистий код: створення і рефакторинг за допомогою Agile / Мартін Роберт; пер. з англ. І. Бондар-Терещенко. – Харків : ВД «Ранок» : Фабула, 2019. – 448 с.
14. Чиста архітектура: Мистецтво розроблення програмного забезпечення / Мартін Роберт; пер. з англ. І. Бондар-Терещенко. – Харків : ВД «Ранок» : Фабула, 2019. – 368 с.

ДОДАТОК А

Підсистема організації спільної роботи учасників освітнього процесу

Специфікація

Аркушів 1

Київ – 2021 року

Позначення	Найменування	Примітки
Документація		
	ТВз-71_Колотуша_В_В_Записка	Текстова частина дипломної роботи
Компоненти		
	server.js	Доступ до бази даних та авторизація користувача
	auth.container.ts	Користувацький інтерфейс для авторизації та реєстрації
	create-doc.container.ts	Користувацький інтерфейс для створення нового документу
	docs-list.container.ts	Користувацький інтерфейс для перегляду списку своїх документів
	doc-viewer.container.ts	Користувацький інтерфейс для перегляду та узгодження документу

ДОДАТОК Б

Підсистема організації спільної роботи учасників освітнього процесу

Лістинг програми

Аркушів 9

Київ – 2021 року

server/controller.js

```
const { DBService } = require('./database/db.service');

async function getDocs(req, res) {
    const { userId } = req.query;
    const docs = await DBService.getDocs(userId);
    res.json(docs);
}

async function getDoc(req, res) {
    const { docId } = req.params;
    const { userEmail } = req.query;
    const doc = await DBService.getDoc(docId);
    if (doc) {
        if (doc.permissions.find(item => item.userEmail === userEmail)) {
            res.json(doc);
        } else {
            res.sendStatus(405);
        }
    } else {
        res.sendStatus(404);
    }
}

async function createDoc(req, res) {
    let doc = req.body;
    doc = {
        ...doc,
        approved: false,
        comments: []
    }
    const id = await DBService.createDoc(doc);
    res.json({ id });
}

async function updateDoc(req, res) {
    const doc = req.body;
    await DBService.updateDoc(doc);
    res.sendStatus(200);
}

async function removeDoc(req, res) {
    const { docId } = req.params;
    await DBService.removeDoc(docId);
    res.sendStatus(200);
}
```

```
async function getUser(req, res) {
    const { body } = req;
    const user = await DBService.getUser(body.email);
    if (!user) return res.sendStatus(404);
    if (user.password !== body.password) return res.sendStatus(401);
    const data = { id: user.id, email: user.email };
    return res.json(data)
}

async function createUser(req, res) {
    let user = req.body;
    const id = await DBService.createUser(user);
    const data = { id: id, email: user.email };
    res.json(data);
}

module.exports = { getDocs, getDoc, createDoc, updateDoc, removeDoc, getUser, createUser };
```

server/db.service.js

```
const MongoClient = require('mongodb').MongoClient;
const { config } = require('../constants');

let db;

MongoClient.connect(config.uri, function(err, client) {
    db = client.db('diplom');
});

class DBService {

    static getDocs(userId) {
        return new Promise((resolve) => {
            const cursor = db.collection('docs').find({ userId });
            cursor.toArray((error, docs) => resolve(docs));
        })
    }

    static getDoc(docId) {
        return db.collection('docs').findOne({ id: docId });
    }

    static async createDoc(data) {
        try {
            const id = this._generateId();
            await db.collection('docs').insertOne({...data, id});
```

```
            return id;
        } catch (e) {
            console.log(e)
        }
    }

    static updateDoc(data) {
        return db.collection('docs').replaceOne({ id: data.id }, data);
    }

    static removeDoc(docId) {
        return db.collection('docs').deleteOne( { id: docId } );
    }

    static getUser(email) {
        return db.collection('users').findOne({ email });
    }

    static async createUser(data) {
        try {
            const id = this._generateId();
            await db.collection('users').insertOne({...data, id});
            return id;
        } catch (e) {
            console.log(e)
        }
    }

    static _generateId() {
        return Math.random().toString(36).substring(2, 15) +
Math.random().toString(36).substring(2, 15);
    }

}

module.exports = { DBService };
```

app.components.ts

```
import React, { ReactElement, FunctionComponent } from 'react';
import { BrowserRouter as Router, Switch, Route } from 'react-router-dom';
import { ThemeProvider } from '@material-ui/styles';
import { createMuiTheme, Theme } from '@material-ui/core';
import { palette } from 'tools/reusable-ui';

import { Header } from './components/header/header.component';
import { SignIn } from '../modules/auth/components/sign-in/sign-in.component';
```

```
import { SignUp } from '../modules/auth/components/sign-up/sign-up.component';
import { DocListContainer } from '../modules/doc-list/doc-list.container';
import { CreateDocContainer } from '../modules/create-doc/create-doc.container';
import { DocViewerPage } from '../modules/doc-viewer/doc-viewer.component';
import { useApp } from './app.hook';
import './app.component.scss';

export const theme: Theme = createMuiTheme({ palette });


export const App: FunctionComponent = (): ReactElement => {
    const { state } = useApp();

    return (
        <ThemeProvider theme={theme}>
            <div className="app-container">
                <Router>
                    <Switch>
                        <Route path="/sign-in" component={SignIn} />
                        <Route path="/sign-up" component={SignUp} />
                        {
                            state.user ?
                                <>
                                   <Header />
                                   <Route path="/doc-list" component={DocListContainer} />
                                   <Route path="/doc-viewer/:id" component={DocViewerPage} />
                                   <Route path="/create-doc" component={CreateDocContainer} />
                                </> :
                                null
                        }
                    </Switch>
                </Router>
            </div>
        </ThemeProvider>
    );
};
```

doc-viewer.component.ts

```
import { Document, Page, pdfjs } from 'react-pdf';
import IconButton from '@material-ui/core/IconButton';
import ArrowBackIcon from '@material-ui/icons/ArrowBackIos';
import ArrowForwardIcon from '@material-ui/icons/ArrowForwardIos';
import 'react-pdf/dist/esm/Page/AnnotationLayer.css';

import { useDocViewer } from './doc-viewer.hook';
import { Sidebar } from './components/sidebar/sidebar.component';
```

```
import { DocFiles } from './components/files/files.component';
import './doc-viewer.component.scss';

pdfjs.GlobalWorkerOptions.workerSrc =
`//cdnjs.cloudflare.com/ajax/libs/pdf.js/${pdfjs.version}/pdf.worker.js`;

export const DocViewerPage = () => {
    const { state, handlers } = useDocViewer();

    if (state.doc) {
        return (
            <div className="doc-viewer-page">
                <div className="doc-viewer-page-main">
                    <DocFiles />
                    <h2 className="doc-viewer-page-main-title">{state.doc.name}</h2>
                    <div className="doc-viewer-page-main-doc-container">
                        <div className="doc-viewer-page-main-doc">
                            <Document
                                file={state.file?.data}
                                onLoadSuccess={handlers.onDocumentLoadSuccess}
                                options={{
                                    cMapUrl: `//cdn.jsdelivr.net/npm/pdfjs-
dist@${pdfjs.version}/cmaps/`,
                                    cMapPacked: true
                                }}
                            >
                                <Page
                                    key={`page_${state.selectedPage + 1}`}
                                    pageNumber={state.selectedPage}
                                />
                            </Document>
                        </div>
                    </div>
                    <div className="doc-viewer-page-main-selector">
                        <IconButton onClick={handlers.prevPage}><ArrowBackIcon /></IconButton>
                        <p className="doc-viewer-page-main-selector-
text">{`${state.selectedPage} / ${state.numPages}`}</p>
                        <IconButton onClick={handlers.nextPage}><ArrowForwardIcon
/></IconButton>
                    </div>
                </div>
                <Sidebar />
            </div>
        );
    }

    return null;
```

```
};
```

doc-viewer-api.service.ts

```
import { expect, Http } from 'tools/reusable-http';
import { Doc } from 'entities/doc/type';
import { docDecoder } from 'entities/doc/decoder';

export class DocViewerApiService {

    private http: Http;

    constructor(http: Http) {
        this.http = http;
    }

    public getDoc(docId: string, userEmail: string) {
        return this.http.get(`/docs/${docId}?userEmail=${userEmail}`,
expect.json(docDecoder));
    }

    public updateDoc(doc: Doc) {
        return this.http.put('/docs', expect.whatever(), doc);
    }

}
```

fetch-http.service.ts

```
import { ResultMethods } from '../result';
import { TransformResponse, ErrorResponse, HttpConfig, Result } from '../helpers/types';
import { Expect } from '../helpers/expect';
import { Http } from '../http.interface';

export class FetchHttpService implements Http {

    private readonly baseUrl: string
    private readonly expect: Expect
    private readonly result: ResultMethods

    constructor({ baseUrl }: HttpConfig) {
        this.baseUrl = baseUrl;
        this.expect = Expect.getInstance();
        this.result = ResultMethods.getInstance();
    }

    public async get <ResponseType>(
        url: string,
```

```
    expect: TransformResponse<ResponseType>
): Promise<Result<ResponseType, ErrorResponse>> {
    const response = await fetch(`${this.baseUrl}${url}`, {
        method: 'GET',
        headers: {
            Accept: 'application/json'
        }
    });

    return this.processResponse(response, expect);
}

public async post <RequestType extends Record<string, unknown>, ResponseType>(
    url: string,
    expect: TransformResponse<ResponseType>,
    body?: RequestType
): Promise<Result<ResponseType, ErrorResponse>> {
    const response = await fetch(`${this.baseUrl}${url}`, {
        method: 'POST',
        body: body ? JSON.stringify(body) : undefined,
        headers: {
            'Content-Type': 'application/json',
            Accept: 'application/json'
        }
    });

    return this.processResponse(response, expect);
}

public async put <RequestType extends Record<string, unknown>, ResponseType>(
    url: string,
    expect: TransformResponse<ResponseType>,
    body?: RequestType
): Promise<Result<ResponseType, ErrorResponse>> {
    const response = await fetch(`${this.baseUrl}${url}`, {
        method: 'PUT',
        body: body ? JSON.stringify(body) : undefined,
        headers: {
            'Content-Type': 'application/json',
            Accept: 'application/json'
        }
    });

    return this.processResponse(response, expect);
}

public async patch <RequestType extends Record<string, unknown>, ResponseType>(
```

```
    url: string,
    expect: TransformResponse<ResponseType>,
    body?: RequestType
): Promise<Result<ResponseType, ErrorResponse>> {
    const response = await fetch(`${this.baseUrl}${url}`, {
        method: 'PATCH',
        body: body ? JSON.stringify(body) : undefined,
        headers: {
            'Content-Type': 'application/json',
            Accept: 'application/json'
        }
    });

    return this.processResponse(response, expect);
}

public async remove (url: string): Promise<Result<void, ErrorResponse>> {
    const response = await fetch(`${this.baseUrl}${url}`, {
        method: 'DELETE',
        headers: {
            'Content-Type': 'application/json',
            Accept: 'application/json'
        }
    });

    return this.processResponse(response, this.expect.whatever());
}
```

ДОДАТОК В

Підсистема організації спільної роботи учасників освітнього процесу

Опис програмного модулю

Аркушів 10

Київ – 2021 року

АНОТАЦІЯ

У якості програмного модуля реалізовано підсистему узгодження текстових документів.

Як основну платформу було обрано веб-браузер через наступні причини:

– високий рівень доступу;

– не має необхідності встановлювати щось на свій пристрій;

– для запуску не потрібно проходити перевірки.

Під час написання системи використано функціональна та об’єктно-орієнтована парадигми програмування.

Об’єктно-орієнтована парадигма програмування використовується для написання класів, які реалізують моделі об’єктів з бази даних, а також для класів, які працюють з локальним сховищем або HTTP запитами.

Функціональна парадигма програмування використовується для написання функцій, які реалізують REST API, реалізації бізнес логіки програмного продукту,

Все вищенаведене допомогло створити надійний та гнучкий програмний модуль, в якому не складно розібратися, а внесення правок не забирає купу часу.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Для функціонування програмного модуля необхідне наступне програмне забезпечення:

- база даних MongoDB;
- NPM CLI - інтерфейс командної строки;
- Node.js - JavaScript оточення для виконання програм (версії 14.17.0).

Мови програмування використані для створення системи:

- JavaScript;
- TypeScript.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмний модуль розв’язує наступні класи задач:

- надання користувацького інтерфейсу для роботи з системою;
- збереження даних;
- надання доступу до збережених даних;

Програма призначена для узгодження текстових документів між учасниками освітнього процесу за рахунок:

- можливості зареєструватися або авторизуватися;
- переглянути список своїх документів;
- створити новий документ;
- переглянути документ.

Система має наступні функціональні обмеження: можливе завантаження тільки PDF-файлів, доступ до документу надається тільки по пошті користувача, але на саму пошту повідомлення не приходить, реєстрація можлива тільки за допомогою пошти користувача.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Під час написання системи використовувались функціональна та об’єктно-орієнтована парадигми програмування (на цьому слайді ви можете побачити діаграму класів).

Об’єктно-орієнтована парадигма програмування використовується для написання класів, які реалізують моделі об’єктів з бази даних, а також для класів, які працюють з локальним сховищем або HTTP запитами.

Функціональна парадигма програмування використовується для:

- написання функцій, які реалізують REST API;
- реалізації бізнес логіки програмного продукту;
- створення компонентів користувацького інтерфейсу.

В результаті було отримано систему, яка має багаторівневу структуру, а самі рівні поділені на менші компоненти, кожен з яких має свою зону відповідальності.

Умовно розроблену систему можна поділити на два компоненти:

- клієнтська частина;
- серверна частина;

Розглянемо детальніше кожен з цих компонентів та набори пакетів, які вони в собі містять.

Клієнтська частина

Цей компонент відповідає за клієнтську частину системи. Він містить увесь потрібний функціонал для користувачів. Даний компонент містить в собі наступний набір пакетів:

- styles – загальні стилі;
- modules – містить модулі для створення інтерфейсу користувача;
- assets – містить усі необхідні файли, такі як картинки, наприклад;
- services – містить функціонал для роботи з серверною частиною;
- tools – містить набір допоміжних функцій.

Серверна частина

Ця частина відповідає за серверну частину системи. У ньому реалізовано функціонал для обробки запитів від клієнтської частини та REST API для того, щоб організації-користувачі могли користуватись системою. Даний компонент містить в собі наступний набір пакетів:

- controllers – містить функціонал, що реалізує бізнес-логіку системи;
- models – містить класи, які відповідають за роботу з базою даних;
- routes – містить функції, що обробляють запити, які надходять до компоненту.

Також даний компонент має зовнішній зв’язок з базою даних, яка містить дані про всі створені документи, їх користувачів та про дії в системі.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для надання можливості запуску та функціонування системи необхідно забезпечити наявність серверу з наступними вимогами:

- оперативна пам’ять – не менше 1 ГБ;
- жорсткий диск – 25 ГБ;
- процесор – двох-ядерний процесор з частотою 2.1 ГЦ;
- будь-яка операційна система.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Запуск програми можна поділити на дві частини, тому розглянемо їх окремо.

Почнемо з серверної частини:

1. Запуск бази даних в терміналі за допомогою команди *mongo*;
2. Завантаження усіх необхідних пакетів командою *npm install*;
3. Запуск сервера в терміналі в корінній директорії проекту командою *node/index.js.*

Тепер перейдемо до клієнтської частини. Для неї необхідно дві команди. Кожна виконується в терміналі в корінній директорії проекту. Далі команди:

1. Завантаження усіх необхідних пакетів командою *npm install*;
2. Запуск додатку командою *npm start.*

ВХІДНІ ДАНІ

У системі існують дві основні сутності: Користувач та Документ. Опишемо вхідні дані для кожної з них окремо.

Вхідні дані користувача:

- пошта у правильному форматі (*test@mail.com*) , тип *string*;
- пароль, закодований алгоритмом sha512, тип *string*.

Вхідні дані документу;

- конвертований у формат base64 файл з розширенням .pdf, тип *string*;
- назва документу, тип *string*;
- опис документу (за необхідністю) , тип *string*;
- пошта користувача, тип *string*;
- текст коментаря, тип *string;*
- рівень доступу до документу, тип *boolean*.

ВИХІДНІ ДАНІ

На вихід система віддає наступні дані:

- список сутностей Документ, в інтерфейсі у списку відображено тільки коротку інформацію про документ;
- сутність Документ для його повного перегляду разом з файлом та коментарями;
- сутність Користувач, яка відповідає тому користувачу, який її отримав.