НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Веб-застосунок для спільного програмування

Виконав: студент 4 курсу, групи ІО-04
(шифр групи)

Водзінський Роман Вікторович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Валько В. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Іваніщев Б. В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент ст. викл. кафедри ІСТ, к.т.н., Орленко С. П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)
Освітньо-професійна програма
“Комп’ютерні системи та мережі”
спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Водзінського Романа Вікторовича

1. Тема проєкту Веб-застосунок для спільного програмування

керівник проєкту асистент Валько В. В.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 27 травня 2024 року № 2112-с

2. Термін здачі студентом закінченого проєкту *4 червня 2024 р.*

3. Вихідні дані до проєкту *технічна документація. теоретичні дані.*

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

Розділ 1. Огляд існуючих рішень

Розділ 2. Огляд та аналіз інструментів для створення веб-застосунку

Розділ 3 Розробка веб-застосунку

Розділ 4. Огляд та тестування роботи веб-застосунку

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) алгоритм дії розробленого застосунку (принципова схема), діаграма бази даних (функціональна схема), структура компонентів застосунку (структурна схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Іваніщев Б. В.		

7. Дата видачі завдання «8» грудня 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	01.12.2023-08.12.2023	
2.	*Вивчення та аналіз завдання*	08.12.2023-15.03.2024	
3.	*Розробка архітектури та загальної структури системи*	15.03.2024-29.03.2024	
4.	*Розробка структур окремих підсистем*	29.03.2024-12.04.2024	
5.	*Програмна реалізація системи*	12.04.2024-29.04.2024	
6.	*Оформлення пояснювальної записки*	03.05.2024-03.06.2024	
7.	*Захист програмного продукту*	03.06.2024	
8.	*Передзахист*	04.06.2024	
9.	*Захист*	18.06.2024	

Студент-дипломник _____ Роман ВОДЗІНСЬКИЙ
(підпис)

Керівник проєкту _____ Володимир ВАЛЬКО
(підпис)

АНОТАЦІЯ

У бакалаврському дипломному проєкті реалізовано веб-застосунок для спільного програмування.

Застосунок призначений для спрощення взаємодії між програмістами в процесі написання коду. Він надає функціонал користувачам для створення нових проєктів та приєднання до вже існуючих. В процесі користувачі можуть редагувати, створювати нові та зберігати відредаговані файли. Весь процес взаємодії відбувається в режимі реального часу, і всі, хто працює над проєктом, миттєво бачать зміни, зроблені іншими користувачами.

Веб-застосунок було створено за допомогою фреймворку Spring для серверної частини, фреймворку Angular для клієнтської частини та PostgreSQL як базу даних.

ANNOTATION

In the bachelor's thesis project, a web application for collaborative programming was implemented.

The application is designed to simplify the interaction between programmers in the process of writing code. It provides functionality for users to create new projects and join existing ones. In the process, users can edit, create new files, and save edited files. The entire interaction process takes place in real time, and everyone working on the project can instantly see the changes made by other users.

The web application was created using the Spring framework for the server side, the Angular framework for the client side, and PostgreSQL as a database.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екзем-пляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	**ІАЛЦ.467200.002 ТЗ**	Веб-застосунок для спільного	4		
			програмування			
			Технічне завдання			
	А4	**ІАЛЦ.467200.003 ПЗ**	Веб-застосунок для спільного	59		
			програмування			
			Пояснювальна записка			
	А4	**ІАЛЦ.467200.004 Д1**	Веб-застосунок для спільного	1		
			програмування			
			Алгоритм дії розробленого			
			застосунку			
			(принципова схема)			
	А4	**ІАЛЦ.467200.005 Д2**	Веб-застосунок для спільного	1		
			програмування			
			Діаграма бази даних			
			(функціональна схема)			
	А4	**ІАЛЦ.467200.006 Д3**	Веб-застосунок для спільного	1		
			програмування			
			Структура компонентів			
			застосунку			
			(структурна схема)			
	А4	**ІАЛЦ.467200.007 Д4**	Веб-застосунок для спільного	53		
			програмування			
			Текст програмного коду			

Зм.	Арк.	№ докум.	Підпис	Дата	**ІАЛЦ.467200.001 ОА**		
Розроб.		Водзінський Р. В.			*Веб-застосунок для спільного програмування* Опис альбому	Літ.	Арк. 1 / Аркушів 1
Перевір.		Валько В. В.					
Н. Контр.		Іваніщев Б. В.				***КПІ ім. Ігоря Сікорського, ФІОТ, ІО-04***	
Затверд.							

ТЕХНІЧНЕ ЗАВДАННЯ

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Веб-застосунок для спільного програмування»

Київ – 2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на дипломний проєкт бакалавра за темою: «Веб-застосунок для спільного програмування». Застосунок може бути практично використаний користувачами для спільного редагування та написання коду.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту за освітньо-професійною програмою “Комп’ютерні системи та мережі” спеціальності 123 “Комп’ютерна інженерія”, затверджене кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної роботи є розробка веб-застосунку для спільного програмування в командах. Призначенням веб-застосунку є надання інструментів для спільного редагування коду в режимі реального часу.

4. ДЖЕРЕЛА РОЗРОБКИ

Основними джерелами розробки є публікації на дану тему в мережі Інтернет, загальнодоступна документація існуючих програм, наукові статті та науково-технічна література з теорії і практики програмування.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Простий і інтуїтивно-зрозумілий інтерфейс.
- Надати можливість користувачам реєструватися в застосунку.
- Надати можливість користувачам створювати проєкти та приєднуватись до існуючих проєктів.
- Надати можливість користувачам створювати нові та редагувати існуючі файли.
- Надати можливість користувачам миттєво отримувати зміни, зроблені іншим користувачем.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Chrome або інший браузер.

5.3. Вимоги до апаратного забезпечення

- ЦП не менше ніж Intel® Core i3 і вище.
- RAM не менше ніж 4 ГБ.
- Доступ до мережі Інтернет.

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	01.12.2023-08.12.2023
Вивчення та аналіз завдання	08.12.2023-15.03.2024
Розробка архітектури та загальної структури системи	15.03.2024-29.03.2024
Розробка структур окремих частин системи	29.03.2024-12.04.2024
Програмна реалізація системи	12.04.2024-29.04.2024
Виправлення помилок	29.04.2024-03.05.2024
Оформлення пояснювальної записки	03.05.2024-03.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Веб-застосунок для спільного програмування»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	(Application Programming Interface) прикладний програмний інтерфейс
CLI	(Command Line Interface) інтерфейс командного рядка
XSS	(Cross-Site Scripting) міжсайтовий скриптинг
CSRF	(Cross-Site Request Forgery) міжсайтова підробка запиту
SQL	(Structured Query Language) мова структурованих запитів
CRUD	(Create, Read, Update, Delete) створення, читання, оновлення, видалення
REST	(Representational State Transfer) передача репрезентативного стану
ORM	(Object-Relational Mapping) об'єктно-реляційне відображення
NoSQL	(Not Only SQL) не тільки SQL
HTTP	(Hypertext Transfer Protocol) протокол передачі гіпертексту
DOM	(Document Object Model) об'єктна модель документа
ACID	(Atomicity, Consistency, Isolation, Durability) атомарність, послідовність, ізоляція, довговічність
СКБД	Система керування базами даних
БД	База даних

ВСТУП

Розробка масштабних програмних застосунків є справжнім викликом для одного розробника. Складність сучасних проєктів вимагає злагодженої співпраці цілої команди програмістів, адже саме командна робота є ключем до створення якісного та надійного коду. Більшість проблем, що виникають під час взаємодії між програмістами, можна вирішити за допомогою систем контролю версій та систем управління проєктами, які дозволяють ефективно координувати роботу команди та відстежувати прогрес розробки.

Однак, незважаючи на наявність цих інструментів, часто виникають ситуації, коли найкращим варіантом є спільне написання коду в режимі реального часу. Такий підхід дозволяє кожному учаснику команди внести своє бачення та ідеї, а також оперативно виправляти помилки та недоліки в коді колег. Спільне програмування значно пришвидшує роботу над проєктом, адже дозволяє швидко обмінюватися думками та знаходити оптимальні рішення.

Веб-застосунок для спільного програмування може бути корисним не лише для співпраці над великими проєктами, але й для навчання програмуванню. Викладачі можуть використовувати веб-застосунок для демонстрації коду, пояснення концепцій та взаємодії зі студентами в процесі навчання, виправляючи помилки та надаючи поради безпосередньо в коді. Також застосунок може бути корисним для проведення технічних співбесід та полегшити ітерв'юеру та кандидату процес оцінки навичок.

Метою цієї роботи є розробка конкурентного веб-застосунку для спільного програмування, який поєднуватиме в собі зручний інтуїтивно зрозумілий інтерфейс, що дозволить новим користувачам швидко освоїти його та набір інструментів, необхідних для роботи над кодом.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Опис завдання

Завданням дипломної роботи є створення веб-застосунку для спільного програмування в командах. В загальному можна виділити два типи таких застосунків. Перший – розширення для вже існуючого середовища розробки, такого як VS Code чи IntelliJ IDEA. Другий полягає в створенні повністю нового середовища розробки з необхідним функціоналом. Створення нового веб-застосунку може мати декілька переваг в порівнянні з першим типом, а саме:

- Доступність з будь-якого пристрою через веб-браузер: користувачам не потрібно завантажувати та встановлювати додаткового програмного забезпечення.
- Повний контроль над функціональністю та дизайном.
- Незалежність від сторонніх середовищ розробки.

До основних вимог застосунку такого типу для ефективної та комфортної роботи можна віднести:

- Можливість реєстрації користувача.
- Можливість створення кімнат або проєктів та приєднання до вже існуючих
- Можливість створювати нові або редагувати вже існуючі файли.
- Можливість користувачам бачити зміни, зроблені іншими користувачами в режимі реального часу.

Одним з головних завдань даної роботи є вирішення проблеми оптимізації передачі змін від користувача іншим учасникам проєкту. Дані, що передаються, мають мати мінімально необхідний розмір та весь процес має бути максимально оптимізований для комфортної роботи без затримок.

1.2 Огляд існуючих рішень

Code With Me – популярний інструмент для спільного програмування, розроблений компанією JetBrains та інтегрований в більшість їхніх середовищ розробки, а саме: IntelliJ IDEA Ultimate, PyCharm Professional, WebStorm та інші [1]. Через те, що це по суті частина середовища розробки, а не зовнішнє розширення, має багатий функціонал та зручний в користуванні.

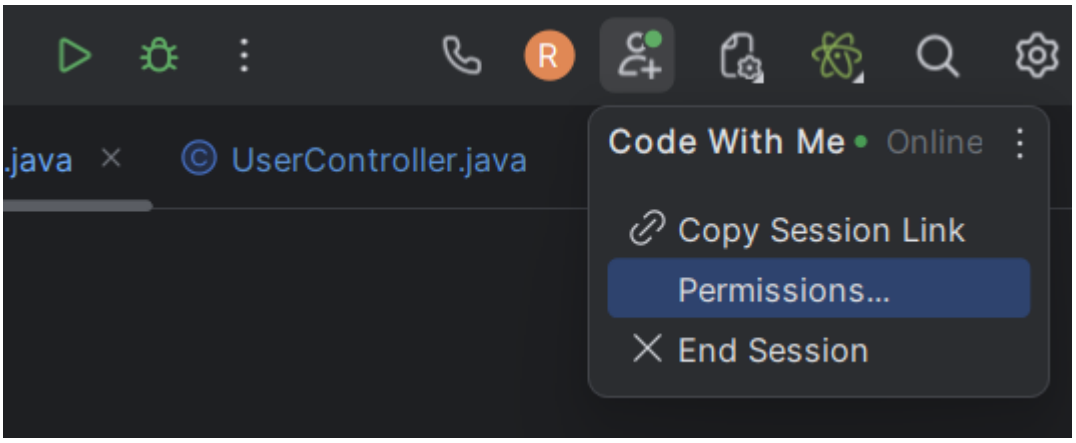


Рисунок 1.1 – Інтерфейс інструменту Code With Me

Окрім необхідного функціоналу, має додаткові можливості, такі як:

- Відеодзвінок між учасниками сесії з можливістю демонстрації екрану.
- Можливість налаштовувати дозволи для користувачі, наприклад, чи можуть вони працювати в терміналі, запускати виконання програми, створювати нові файли та інше.
- Можливість закріплюватись за вибраним користувачем та бачити всі зміни, які він робить, автоматично переміщуючись за його курсором в файлі та між файлами.

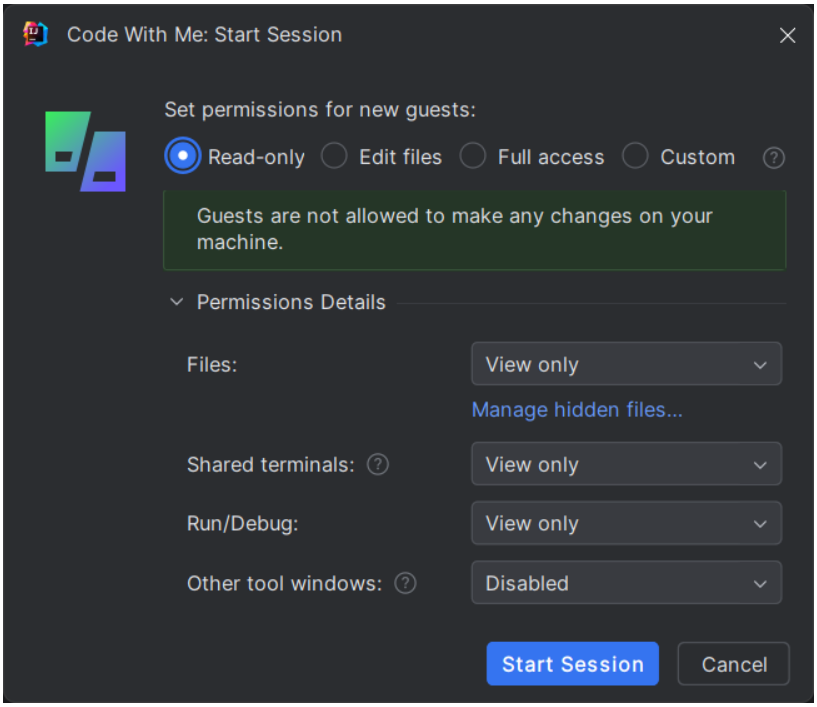


Рисунок 1.2 – Вікно створення нової сесії в Code With Me

Серед недоліків даного інструменту можна виділити:

- Платний доступ - більшість продуктів JetBrains розповсюджуються за підпискою, а безкоштовні версії не мають цього інструменту.
- Потребує значну кількість ресурсів комп'ютера - дана проблема виникає через власне середовище розробки, яке, хоч і має дуже великий функціонал, але при цьому потребує чимало ресурсів.
- Інтеграція лише з продуктами компанії.

CodeTogether – популярне розширення для спільного програмування, доступне для більшості популярних середовищ розробки, зокрема: VS code, Visual Studio, Eclipse, PyCharm, Intellij, WebStorm та інші [2]. Має багатий функціонал багато в чому аналогічний до Code With Me з хорошою інтеграцією в середовище.

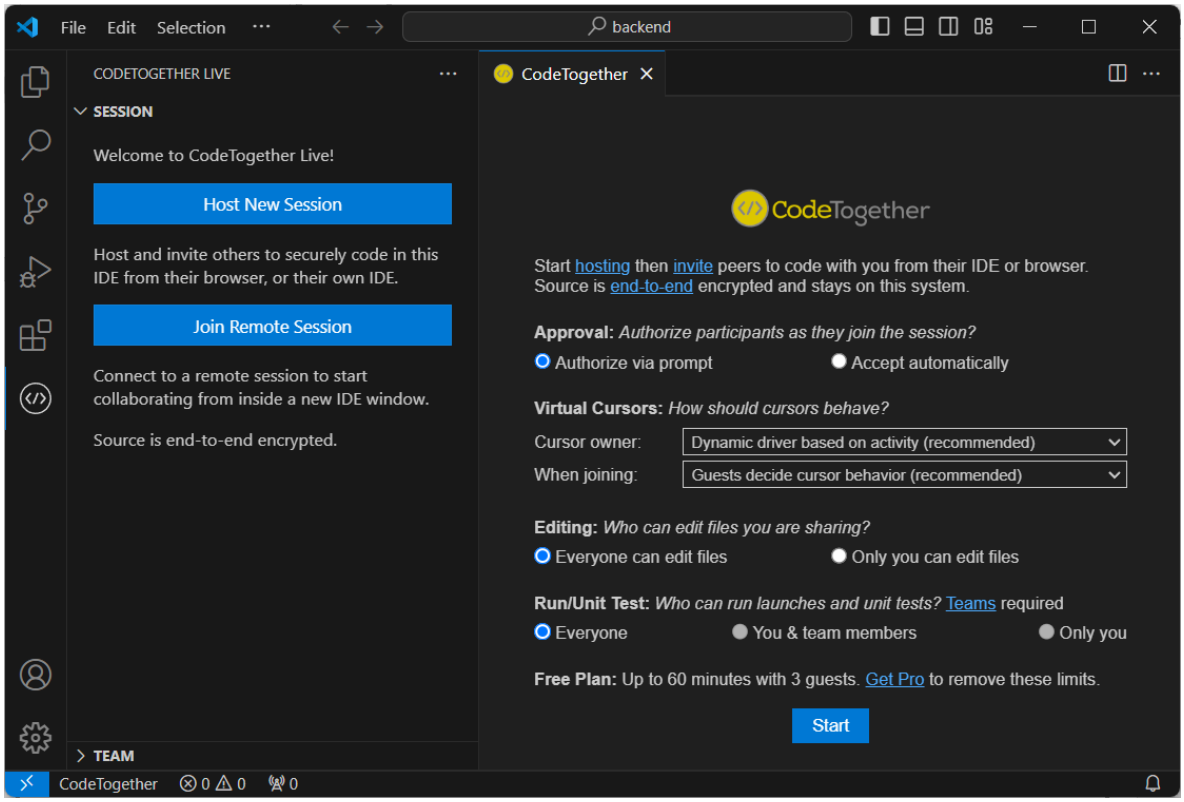


Рисунок 1.3 – Вікно створення нової сесії в CodeTogether

Перевагами цього розширення є:

- Гнучке налаштування сесії та взаємодії користувачів.
- Можливість анонімного входу для користувачів, які приєднуються до сесії, ввівши лише своє ім’я.
- Для користувачів, які приєднуються, є можливість відкривати проєкт в браузері в спеціальному веб-редакторі.

Серед недоліків даного розширення можна виділити лише 60 хвилинний період роботи з 3 учасниками в сесії в безкоштовній версії, що в багатьох випадках може бути недостатньо.

Codeshare - веб-застосунок для спільної розробки коду в режимі реального часу. Дозволяє декільком розробникам віддалено одночасно редагувати, переглядати та налагоджувати код. Для приєднання до вже існуючої сесії достатньо перейти за посиланням, яке повинен відправити власник сесії.

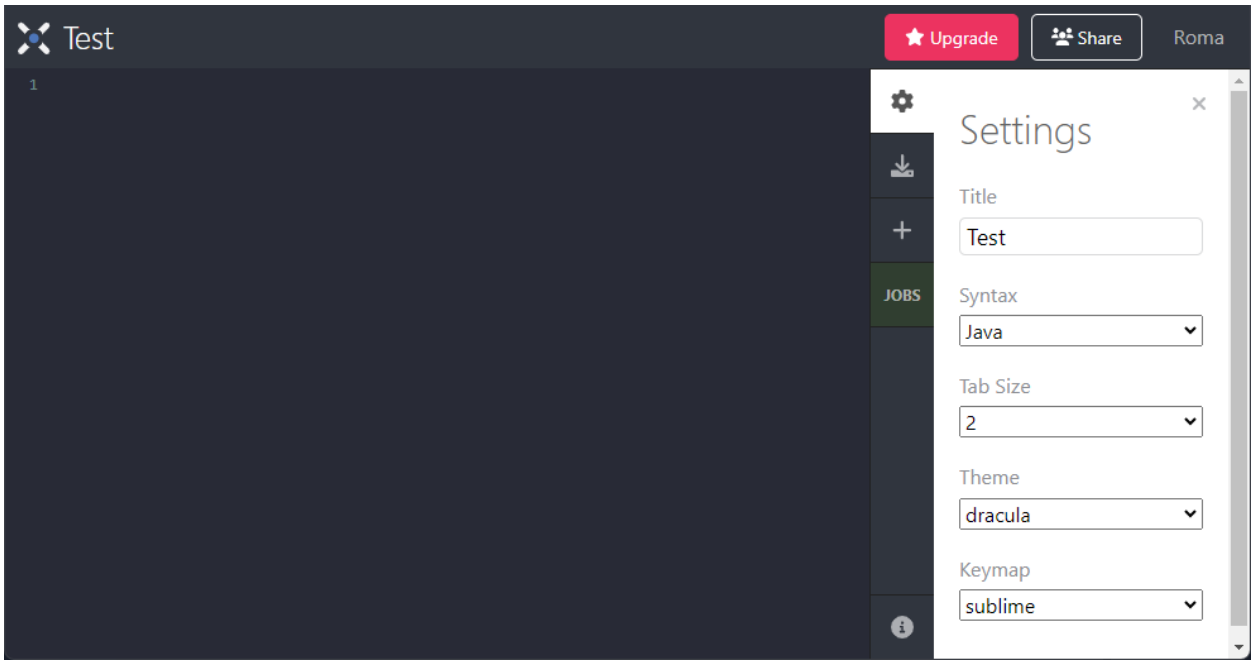


Рисунок 1.3 – Інтерфейс codeshare

Основними перевагами даного застосунку є:

- Мінімалістичний та інтуїтивно зрозумілий інтерфейс з можливістю вибору теми редактора із запропонованого переліку.
- Підтримка переважної більшості мов програмування, вибрати які можна з запропонованого переліку вручну.
- Необмежена кількість учасників в сесії навіть в безкоштовній версії.

До мінусів цього застосунку можна віднести:

- Доволі мінімальне налаштування сесії та всього середовища розробки.

- Відсутня можливість створення декількох файлів в одній сесії. Всі дії відбуваються в одному просторі, який і є по суті єдиним файлом, який як назву використовує назву сесії.
- Наявність платної версії, в якій присутні деякі корисні функції, такі як можливість створення сесії, де користувачі можуть лише переглядати код, а не редагувати його. Наявність в безкоштовній версії реклами.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було описано специфіку проєкту та головні вимоги до розроблюваного веб-застосунку. Серед ключових вимог виділено: реєстрацію користувачів, створення та управління проєктами, редагування коду, надійну синхронізацію змін між учасниками в режимі реального часу та інші. Окремо наголошено на важливості оптимізації передачі даних про зміни для забезпечення стабільної роботи.

В результаті аналізу існуючих рішень, таких як Code With Me, CodeTogether та Codeshare, було виявлено їх основні переваги та недоліки. Code With Me та CodeTogether мають багатий функціонал та зручну інтеграцію з популярними середовищами розробки, але потребують платної підписки для повноцінного використання. Codeshare, в свою чергу, має мінімалістичний інтерфейс та підтримку багатьох мов програмування, але має обмежені можливості налаштування сесії та відсутність створення декількох файлів.

Розроблений застосунок має на меті вирішити вищезгадані проблеми, надаючи користувачам безкоштовний доступ та достатній функціонал для комфортної та ефективної спільної роботи над проєктами в командах.

РОЗДІЛ 2. ОГЛЯД ТА АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ

2.1 Аналіз та вибір backend частини застосунку

Вибір технології для реалізації серверної частини веб-застосунку є важливим рішенням. Від нього залежить ефективність, продуктивність та масштабованість всього застосунку. Серверна частина відповідає за обробку запитів користувача, взаємодію з базою даних, бізнес-логіку, автентифікацію, авторизацію та багато інших важливих аспектів.

Існує широкий вибір мов програмування та фреймворків, які можна використовувати для розробки серверної частини. Серед мов програмування, які зазвичай використовуються для розробки серверної частини застосунку, можна виділити Java, Python, JavaScript (Node.js), PHP та Go. Кожна з цих мов має власні фреймворки та бібліотеки, які роблять процес розробки ефективнішим та менш часозатратним. Під час вибору фреймворку необхідно звертати увагу на такі аспекти, як: широкий набір вбудованих функцій, зручність розробки, продуктивність застосунку та його безпека.

2.1.1 Spring

Spring - це потужний та популярний фреймворк для розробки веб-застосунків мовою Java [3]. Він надає повноцінний набір засобів для створення надійних, масштабованих та безпечних застосунків. Spring використовує концепції інверсії контролю та ін'єкції залежностей, що забезпечує низьку зв'язність між компонентами системи та підвищують їх модульність. Такий підхід робить архітектуру гнучкою та зменшує залежність одних частин програми від інших.

Spring Framework - це модульна платформа, яка складається з різних компонентів, призначених для вирішення широкого спектру завдань при розробці веб-застосунків:

- Spring Core - це основа фреймворку, яка реалізує головні концепції Spring, такі як інверсія контролю (IoC) та ін'єкція залежностей (DI).
- Spring Boot - це інструмент, який полегшує процес розробки та розгортання автономних застосунків на базі Spring. Він автоматично налаштовує всі необхідні компоненти, що дозволяє розробникам зосередитися на написанні бізнес-логіки. Spring Boot також надає вбудований сервер та засоби моніторингу.
- Spring MVC - це частина фреймворку Spring, що реалізує шаблон Model-View-Controller. Spring MVC спрощує розробку веб-застосунків, надаючи потужні інструменти для обробки запитів, управління даними та генерації відповідей.
- Spring Data - це інструмент для роботи з різними базами даних та сховищами. Він пропонує єдиний інтерфейс для взаємодії з популярними ORM-фреймворками, такими як Hibernate, та NoSQL-рішеннями.
- Spring Security - це потужний інструмент для забезпечення безпеки веб-застосунків. Він надає гнучкі механізми автентифікації користувачів, управління ролями та дозволами.

Окрім цих основних модулів, екосистема Spring містить багато інших проєктів: Spring Cloud для створення розподілених систем та мікросервісів, Spring Batch для пакетної обробки даних, Spring Integration для побудови систем обміну повідомленнями та інтеграції додатків в екосистемі Spring тощо. Завдяки такій багатій функціональності Spring є одним з найпопулярніших фреймворків для розробки веб та корпоративних застосунків на Java.

Переваги використання Spring Framework для створення веб-застосунків:

- Широкий набір функцій: Spring надає багато готових інструментів для вирішення типових завдань: робота з базами даних, безпека, веб-сервіси, обробка транзакцій тощо.
- Інверсія контролю (IoC) та ін'єкція залежностей (DI): Spring забезпечує гнучку і потужну реалізацію IoC та DI, що спрощує управління залежностями і підвищує модульність.
- Сумісність з іншими технологіями: Spring легко інтегрується з багатьма популярними інструментами та фреймворками (Hibernate, JPA, REST тощо).
- Продуктивність розробки: Spring допомагає пришвидшити розробку завдяки готовим шаблонам проєктування, абстракціям та автоматизації типових задач.
- Безпека: Spring Security - це потужний та гнучкий інструмент для забезпечення автентифікації та авторизації у веб-застосунках.
- Підтримка: Spring має велику спільноту розробників, багато навчальних ресурсів, інструментів та розширень, що полегшує освоєння та використання фреймворку.

Недоліки використання Spring Framework:

- Складність: Spring - великий та комплексний фреймворк. Вивчення всіх його можливостей та ефективне використання потребує часу та зусиль.
- Розмір застосунку: Програми на Spring можуть мати більший розмір та споживати більше ресурсів у порівнянні з застосунками на більш мінімалістичних фреймворках.

2.1.2 Django

Django - це потужний веб-фреймворк, створений на мові програмування Python. Він був розроблений для швидкої та ефективної розробки веб-застосунків та дотримується принципу "Batteries included"[4]. Це означає, що Django містить багато вбудованих компонентів та функцій, необхідними для повноцінної веб-розробки. Однією з помітних особливостей фреймворку є наявність автоматично створеної панелі адміністратора Django — це дозволяє легко виконувати операції CRUD без ручного написання коду.

Переваги використання Django:

- Простота використання: Завдяки високому рівню абстракції та автоматизації багатьох задач, Django значно прискорює процес розробки.
- Безпека: Django включає в себе вбудовані функції безпеки, такі як аутентифікація користувачів, система дозволів та захист від поширених веб-вразливостей. Це допомагає забезпечити надійність та захищеність веб-застосунків.
- Велика спільнота: Django має велику та активну спільноту розробників, що забезпечує підтримку та хорошу документацію.

Недоліки використання Django:

- Складність: Може бути занадто громіздким для простих проєктів. Django має багато вбудованих компонентів, які можуть бути непотрібними для базових веб-застосунків.
- Крута крива навчання: Django має багато вбудованих функцій, що може ускладнити його освоєння для новачків.
- Проблеми з продуктивністю: Django може бути менш продуктивним у порівнянні з іншими фреймворками, особливо при високих навантаженнях.

2.1.3 Express.js

Node.js - це платформа, яка дозволяє виконувати код, написаний мовою JavaScript, на сервері. Node.js побудований на основі рушія V8 JavaScript від Google.

Express.js - це мінімалістичний і гнучкий фреймворк для Node.js, призначений для створення веб-застосунків. Він надає простий спосіб визначення ендпоінтів, обробки запитів та відповідей, а також інтеграції з іншими бібліотеками та інструментами [5].

Переваги використання Express.js:

- Простота: Це мінімалістичний та легкий фреймворк, завдяки чому його можна швидко освоїти.
- Продуктивність: Express.js базується на Node.js, що забезпечує високу швидкодію та масштабованість для обробки великої кількості запитів.
- Популярність: Express.js має велику спільноту розробників та багатий вибір бібліотек, які можна легко додати до проєкту

Недоліки використання Express.js:

- Відсутність вбудованої підтримки для деяких функцій: Express.js не має вбудованої підтримки для деяких функцій , зокрема, обробка сесій, авторизація та кешування, що вимагає встановлення додаткових модулів.
- Відсутність вбудованої підтримки для асинхронних функцій: Express.js не має вбудованої підтримки для асинхронних функцій, що може ускладнити роботу з асинхронним кодом.
- Відсутність вбудованих механізмів безпеки: Express.js не має вбудованих засобів захисту від поширених веб-вразливостей, серед яких XSS, CSRF або SQL Injection.

2.1.4 Laravel

Laravel - це популярний PHP-фреймворк з відкритим кодом для розробки веб-застосунків [6]. Створений на основі ще одного популярного PHP-фреймворку – Symfony, Laravel значно розширює та доповнює його функціональність. Має багату бібліотеку пакетів, що дозволяє легко інтегрувати додаткову функціональність в проєкт.

Переваги використання Laravel:

- Простий у використанні: Laravel має простий та інтуїтивно зрозумілий синтаксис, що полегшує написання коду як початківцями, так і досвідченими розробниками.
- Artisan CLI: Laravel має вбудований потужний інтерфейс командного рядка. Він надає безліч корисних команд для генерації коду, міграцій бази даних, тестування тощо. Це дозволяє пришвидшити процес розробки та зосередитися на написанні якісного коду.
- Вбудовані функції: Laravel має широкий вибір вбудованих функцій, а саме: автентифікація, авторизація, кешування та інші. Це значно полегшує процес розробки.
- Міграція баз даних: Laravel забезпечує простий та надійний спосіб міграції схем баз даних між різними середовищами, що запобігає втраті або пошкодженню даних.

Недоліки використання Laravel:

- Продуктивність: Через високий рівень абстракції та велику кількість вбудованих функцій, Laravel може бути дещо повільнішим порівняно з іншими, легшими, фреймворками.
- Складність оновлення версій: Перехід на нову версію Laravel часто вимагає внесення істотних змін в існуючий код проєкту. Оскільки фреймворк активно розвивається, між версіями можуть бути значні відмінності в роботі тих чи інших компонентів.

2.1.5 Gin

Gin - це легкий і мінімалістичний веб-фреймворк, розроблений для мови програмування Go [7]. Gin базується на стандартному пакеті net/http мови Go, що забезпечує його надійність та сумісність. Водночас, він увібрав найкращі ідеї та практики інших популярних веб-фреймворків, адаптувавши їх до специфіки мови Go. Це робить Gin інтуїтивно зрозумілим для розробників, знайомих з мовою.

Переваги використання Gin:

- Висока продуктивність: Gin відомий своєю відмінною продуктивністю. Він відрізняється високою швидкістю обробки запитів у порівнянні з іншими веб-фреймворками Go. Це робить його придатним для створення високопродуктивних API та веб-застосунків.
- Обробка помилок: Gin дозволяє легко відстежувати та обробляти помилки, що виникають під час HTTP-запиту.
- Широка підтримка проміжного програмного забезпечення: Gin пропонує багато вбудованих middleware-компонентів для вирішення типових задач, серед яких автентифікація, логування та парсинг запитів. Також можна створити власне проміжне програмне забезпечення, яке буде відповідати специфічним вимогам конкретної програми.

Недоліки використання Gin:

- Відсутність функцій: Gin відомий своїм мінімалістичним підходом. У порівнянні з іншими фреймворками, він може не мати деяких корисних можливостей та інструментів.
- Обмежена спільнота розробників: В порівнянні з такими популярними фреймворками як Django чи Spring, спільнота Gin хоча й активно розвивається, проте є значно меншою.

2.1.6 Вибір backend частини застосунку

Для реалізації backend частини мого веб-застосунку я обрав фреймворк Spring. Він є потужним, надійним фреймворком з багатим функціоналом та можливостями, яких, наприклад, немає в Gin та Express.js, що значно пришвидшує та спрощує розробку застосунку. Spring забезпечує високий рівень безпеки та надає потужні вбудовані інструменти для аутентифікації та захисту даних. Java, на якій базується Spring, має статичну типізацію, що полегшує роботу зі моделями даних.

2.2 Аналіз та вибір frontend частини застосунку

Вибір фреймворку для розробки фронтенд частини веб-застосунку є не менш важливим рішенням, ніж вибір технологій для бекенду. Фронтенд відповідає за користувацький інтерфейс, інтерактивність та загальне враження користувача від застосунку. Правильний вибір фреймворку може значно полегшити розробку, покращити продуктивність та забезпечити хорошу підтримку і розвиток проєкту в майбутньому.

Серед популярних фреймворків та бібліотек для розробки клієнтської частини застосунку можна виділити Angular, React та Vue. Кожен з них має свої особливості, переваги та недоліки.

2.2.1 Фреймворк Angular

Angular - це популярний фреймворк для створення клієнтської частини веб-застосунку. Розроблений командою Google, він базується на мові програмування TypeScript і використовує компонентний підхід та MVC-архітектуру [8]. Компонентний підхід означає, що компонент є самодостатньою частиною застосунку, що включає в себе логіку, шаблон та стилі. Використання фреймворком мови TypeScript надає статичну типізацію,

що допомагає уникнути помилок на етапі розробки та спрощує підтримку коду. Також Angular використовує концепцію ін’єкції залежностей, що сприяє більшій гнучкості та тестованості коду.

Angular має вбудовану систему маршрутизації, що дозволяє легко створювати односторінкові застосунки з навігацією між різними компонентами. Фреймворк надає потужні інструменти для роботи з формами, включаючи валідацію, контроль стану та обробку подій. Він добре підходить для створення великих та складних веб-застосунків завдяки своїй архітектурі та можливостям масштабування.

Переваги використання Angular:

- Модульність: Angular заохочує розробку модульних і добре структурованих застосунків. Компонентна архітектура дозволяє розбивати застосунок на незалежні та легко тестовані компоненти, які можна використовувати повторно, що полегшує підтримку та масштабування проєкту.
- Angular CLI: Це потужний інструмент, який автоматизує та пришвидшує процес розробки Angular-застосунків. Він надає набір команд та шаблонів, які допомагають розробникам швидко та легко створювати нові проєкти, генерувати необхідні файли та структури коду, а також керувати залежностями.
- Двостороння архітектура зв'язування даних: Даний підхід полегшує автоматичну синхронізацію між моделлю та відображенням. Зміни в моделі миттєво відображаються у інтерфейсі і навпаки.
- Багатий функціонал: Angular пропонує великий набір вбудованих функцій і інструментів для вирішення поширених завдань веб-розробки. Наприклад, Angular має вбудовані інструменти для роботи з формами, валідацією, обробкою подій, директиви для роботи з DOM, анімації та багато іншого. Додатково, Angular має офіційну бібліотеку компонентів користувацького інтерфейсу під назвою Angular Material. Ця бібліотека надає набір готових, добре

оформлених та налаштовуваних компонентів, наприклад, кнопки, форми, меню, таблиці тощо [9].

Недоліки використання Angular:

- Крута крива навчання: Angular може бути складним для вивчення, особливо для початківців. Фреймворк має багато концепцій, а саме: dependency injection, decorators, directives, pipes тощо.
- Надмірна складність: Для невеликих і простих проєктів Angular може бути занадто громіздким і надлишковим.

2.2.2 React

React - широко використовувана бібліотека JavaScript з відкритим вихідним кодом, для створення користувацьких інтерфейсів [10]. Вона дозволяє розробникам створювати веб-застосунки використовуючи компонентний підхід. Кожен компонент містить власну функціональність і може повторно використовуватися в різних частинах програми.

React використовує технологію віртуального DOM для оптимізації оновлення інтерфейсу користувача. Це забезпечує високу продуктивність навіть у складних застосунках.

Переваги використання React:

- Багатий інструментарій: React має багату колекцію бібліотек, інструментів та ресурсів, які полегшують розробку та розширюють можливості фреймворку.
- Висока продуктивність: Завдяки віртуальній DOM та ефективному оновленню компонентів, React забезпечує високу продуктивність та швидкість рендерингу.
- Спільнота розробників: React має велику та активну спільноту розробників, що забезпечує доступ до ресурсів, включаючи навчальні посібники і статті, форуми та сторонні бібліотеки.

Недоліки використання React:

- Необхідність додаткових бібліотек: Для створення повноцінного застосунку часто потрібні додаткові бібліотеки, наприклад, для управління станом, навігації тощо.
- Висока крива навчання для новачків: React може бути складним для освоєння, особливо для новачків, через концепції, такі як JSX, props, стан, життєвий цикл компонентів, хуки тощо.

2.2.3 Vue

Vue — це прогресивний JavaScript-фреймворк для розробки інтерактивних користувацьких інтерфейсів, відомий своєю гнучкістю, адаптивністю та простотою [11]. Він має низький поріг входження, що робить його привабливим для початківців, але при цьому пропонує достатньо потужності та гнучкості для створення повноцінних застосунків. Vue підтримує реактивність, що дозволяє автоматичне оновлення користувацького інтерфейсу відповідно до змін даних, спрощуючи розробку та зменшуючи ймовірність помилок.

Переваги використання Vue:

- Простота вивчення: Vue має низький поріг входження і простий у вивченні, особливо для розробників, знайомих з HTML, CSS та JavaScript.
- Продуктивність і легкість: Vue є швидким та ефективним фреймворком з віртуальним DOM і підтримкою реактивності. Він також має невеликий розмір, близько 18 КБ після стиснення.
- Гнучкість: Vue може бути використаний у різних випадках, від розширення статичного HTML до створення повноцінних інтерактивних застосунків.

Недоліки використання Vue:

- Менша популярність: Vue менш популярний порівняно з React та Angular, що означає меншу кількість навчальних матеріалів, статей, сторонніх бібліотек.
- Продуктивність у великих застосунках: Незважаючи на те, що Vue загалом демонструє хорошу продуктивність, у деяких випадках він може мати проблеми з нею у великих і складних застосунках.

2.2.4 Вибір frontend частини застосунку

Для реалізації frontend частини мого веб-застосунку я обрав фреймворк Angular. Це потужний та універсальний інструмент з багатим функціоналом, який дозволяє швидко та ефективно розробляти складні інтерактивні інтерфейси. Він має чітку та структуровану архітектуру, що спрощує розробку та підтримку великих проєктів. На відміну від React та Vue, Angular надає багато вбудованих можливостей та готових рішень, що може пришвидшити розробку.

2.3 Аналіз та вибір бази даних

Бази даних є важливою складовою веб-застосунку. Вони відповідають за зберігання, організацію та доступ до даних, які використовуються в застосунку. Правильний вибір бази даних може значно вплинути на продуктивність, масштабованість та розвиток проєкту в майбутньому. Загалом, бази даних поділяють на два типи: реляційні (SQL) та нереляційні (noSQL).

Реляційні бази даних, такі як PostgreSQL, Oracle DB та MySQL, використовують структуровану мову запитів (SQL) для визначення та маніпулювання даними. Данні зберігаються в таблицях, в яких стовпці

використовуються для визначення інформації, що зберігається, а рядки містять самі значення. Перевагами реляційних баз даних є відповідність стандарту ACID, а також можливість створення складних запитів за допомогою SQL. Вони добре підходять для структурованих даних з чіткими зв'язками між сутностями та можуть бути менш гнучкими при роботі з неструктурованими або частково структурованими даними.

З іншого боку, нереляційні бази даних не використовують традиційну табличну структуру, натомість пропонують більш гнучкий підхід до зберігання даних. Нереляційні бази даних підтримують різні моделі даних, такі як документо-орієнтовані (MongoDB), ключ-значення (Redis), графові (Neo4j) та стовпчикові (Cassandra). Кожна модель оптимізована під певні сценарії використання. NoSQL бази даних зазвичай забезпечують високу продуктивність та масштабованість, що робить їх придатними для великих обсягів даних та розподілених систем. Однак варто пам'ятати, що нереляційні БД не є заміною реляційних на всі випадки. Вибір типу БД залежить від конкретних вимог проєкту та структури даних. Часто в застосунках використовують комбінацію реляційних та нереляційних баз даних, застосовуючи переваги кожного підходу.

Розглянемо деякі популярні реляційні та нереляційні бази даних, які необхідні для реалізації веб-застосунку:

- PostgreSQL – це високопродуктивна, надійна об'єктно-реляційна база даних з багатим набором функцій. Підтримує складні запити, індексування, транзакції та процедури, забезпечуючи високу цілісність та безпеку даних. Вона має чудову підтримку спільноти та активний розвиток [12].
- MySQL є ще однією популярною реляційною базою даних. Відома своєю простотою, продуктивністю та широкою сумісністю з різними платформами та мовами програмування. Але має обмежену підтримку складних запитів та транзакцій порівняно з Postgres [13].

- MongoDB - документо-орієнтована нереляційна база даних, що дозволяє зберігати та швидко отримувати доступ до JSON-подібних документів, забезпечуючи гнучкість та горизонтальне масштабування [14].
- Redis - нереляційна база даних типу "ключ-значення", яка зберігає дані в оперативній пам'яті для забезпечення високої продуктивності [15]. Зазвичай використовується для кешування, обміну повідомленнями в реальному часі та управління сеансами, що є важливими аспектами багатьох проєктів.

Для веб-застосунку я обрав PostgreSQL в якості основної бази даних та Redis для зберігання сесій користувача при автентифікації. PostgreSQL було обрано завдяки її потужності, надійності та багатому набору функцій, що дозволяє ефективно зберігати та керувати даними. Redis було обрано через його високу швидкість та зручну модель ключ-значення, що ідеально підходить для зберігання сесій.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі проаналізовано та обрано технології й інструменти для розробки клієнтської та серверної частин, а також для створення бази даних веб-застосунку. Було досліджено переваги та недоліки кожного рішення, враховуючи такі критерії, як продуктивність, безпека, масштабованість, наявність додаткового функціоналу, активність спільноти розробників та зручність розробки застосунків.

Під час вибору технології для реалізації серверної частини веб-застосунку були розглянуті такі фреймворки як Spring, Django, Express.js, Laravel та Gin. Після детального аналізу переваг і недоліків кожного рішення, був обраний фреймворк Spring на мові програмування Java завдяки його багатому функціоналу, вбудованим інструментам безпеки, роботи з базами даних та іншими важливими аспектами веб-розробки. Використання Spring дозволить забезпечити надійну та масштабовану архітектуру серверної частини застосунку.

Що стосується фронтенд-розробки, після порівняльного аналізу провідних фреймворків та бібліотек таких як Angular, React та Vue, я дійшов висновку, що оптимальним вибором для мого проєкту буде Angular. Цей фреймворк забезпечує структуровану компонентну архітектуру, підтримку TypeScript та ін'єкції залежностей, що дозволить створювати складні та гнучкі користувацькі інтерфейси.

Як база даних в моєму застосунку було обрано PostgreSQL для зберігання необхідних сутностей. Для зберігання сесій користувача при автентифікації буде використовуватись нереляційна база даних Redis, яка завдяки своїй високій швидкодії та моделі ключ-значення є ідеальним рішенням для цієї задачі.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Розробка серверної частини застосунку

Серверна частина веб-застосунку розроблятиметься з застосуванням фреймворку Spring. Для збереження даних використовуватиметься реляційна база даних PostgreSQL, а для збереження сесій користувачів - Redis. Поєднання цих технологій дозволить створити потужну та ефективну серверну частину веб-застосунку.

3.1.1 Ініціалізація проєкту та встановлення залежностей

Spring проєкт можна реалізувати вручну, створивши необхідні файли та каталоги, або за допомогою інструменту spring initializr, який надає зручний та інтуїтивно зрозумілий графічний інтерфейс зображений на рис. 3.1.

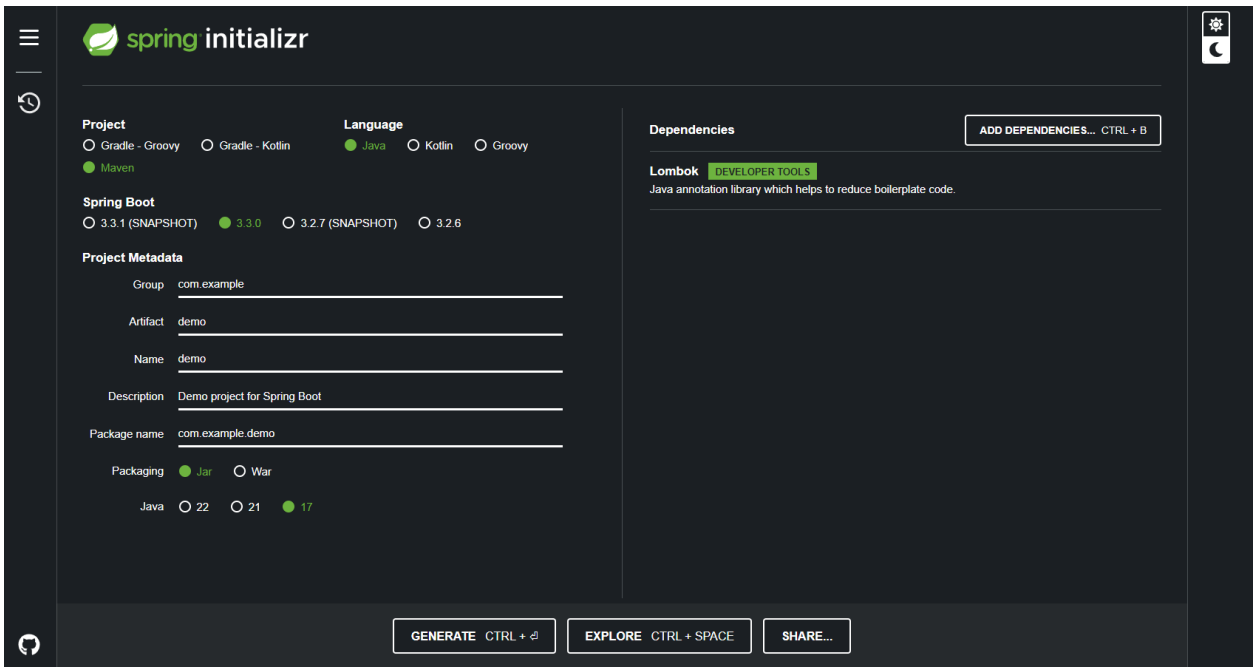


Рисунок 3.1 – Інтерфейс spring initializr

В ньому можна вказати всі необхідні налаштування. Наприклад, систему збірки проєктів (Gradle або Maven), мову програмування, назви проєкту та пакетів, а також додати всі необхідні залежності, вибравши їх із запропонованого каталогу. Після введення всіх налаштувань можна згенерувати та завантажити готовий проєкт зі всіма необхідними файлами та залежностями, що значно спрощує та прискорює процес створення Spring проєкту.

У моєму випадку я реалізовую застосунок, використовуючи систему збірки проєктів Maven. Ось кілька головних залежностей, які можна виділити в моєму проєкті:

- spring-boot-starter-web: Ця залежність містить все необхідне для створення веб-застосунків на базі Spring MVC. Вона включає компоненти для розробки RESTful веб-сервісів, обробки HTTP-запитів, маршрутизації та інше.
- spring-boot-starter-security: Залежність, яка додає засоби безпеки до Spring застосунку. Вона включає модулі Spring Security, що дозволяють реалізувати автентифікацію, авторизацію, захист від різних атак та інші функції безпеки.
- spring-boot-starter-data-jpa: Ця залежність включає необхідні компоненти для роботи з JPA в Spring Boot застосунках. Вона надає абстракцію над конкретною реалізацією бази даних, спрощуючи роботу з даними.
- lombok: Це бібліотека, яка за допомогою анотацій автоматично генерує поширений "шаблонний" код, наприклад, методи getter та setter, конструктори, метод toString() та інші. Це дозволяє зробити код більш лаконічним і зручним для читання [16].
- postgresql: Бібліотека, що додає драйвер для роботи з базою даних PostgreSQL. Вона забезпечує можливість підключення до бази даних, виконання SQL-запитів, обробки результатів.

3.1.2 Створення та інтеграція бази даних в проєкт

Як вже було зазначено раніше, як база даних в моєму проєкті використовується Postgres. Для роботи з даною СКБД можна використовувати графічний інтерфейс, наприклад, застосунок pgAdmin, або інтерфейс командного рядка, в випадку Postgres - це psql.

В моєму застосунку існує дві таблиці, які необхідно створити в БД. Це таблиця "users", яка зберігає всю необхідну інформацію про користувачів в застосунку та таблиця "projects", що зберігає створені користувачами проєкти та всю їхню інформацію. Розглянемо їх детальніше:

1. Таблиця "users" містить поля id типу UUID та рядкові поля email, username, provider_id та password. Всі поля, крім id та username, є не обов’язковими, а поля email та provider_id мають бути унікальними.
2. Таблиця "projects" містить поля id та owner_id типу UUID, де owner_id, по суті, реалізовує зв’язок "багато до одного" з таблицею "users". Це означає, що кожен проєкт має одного власника, проте користувач може бути власником декількох проєктів. Також містить рядкове поле name.

Для реалізації зберігання користувачів в проєкті необхідно створити зв’язок "багато до багатьох" між таблицями "projects" та "users". Це можна здійснити за допомогою додаткової таблиці "project_members", яка містить два поля: project_id та user_id, які відповідно містять первині ключі проєкта та користувача.

Діаграма створеної бази даних із позначенням зв’язків між таблицями зображена на рис. 3.2. Ця діаграма надає візуальне представлення архітектури бази даних, демонструючи, як різні таблиці пов'язані між собою за допомогою первинних та зовнішніх ключів.

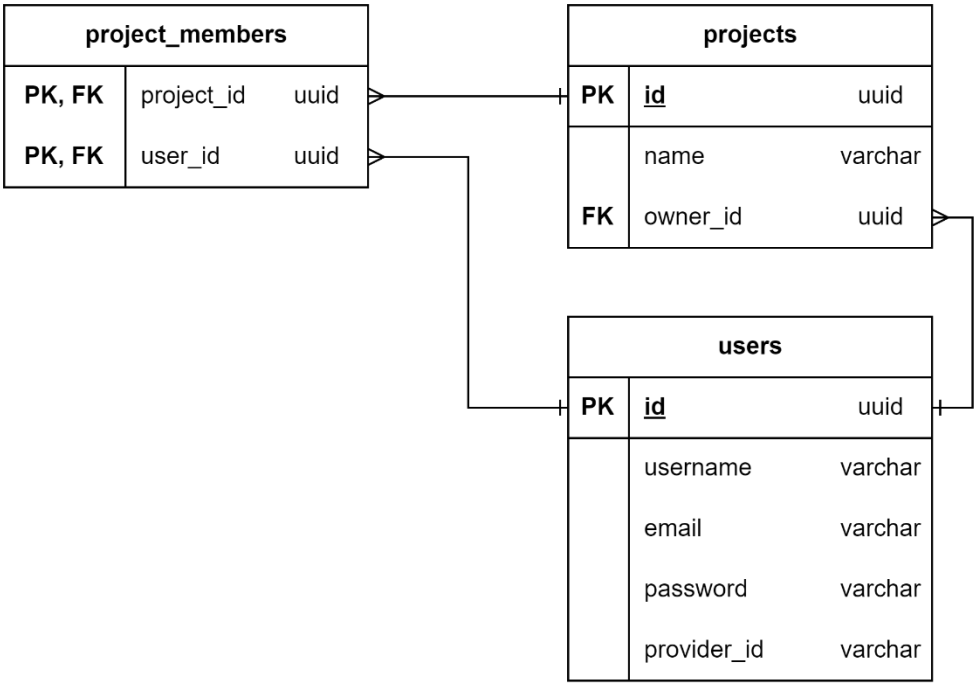


Рисунок 3.2 – Діаграма бази даних

Для підключення бази даних до серверної частини застосунку, необхідно вказати деякі налаштування в файлі application.properties, зображені на рис 3.3.

```
3  spring.datasource.url=${POSTGRES_URL}
4  spring.datasource.username=${POSTGRES_USER}
5  spring.datasource.password=${POSTGRES_PASSWORD}
6  spring.jpa.hibernate.ddl-auto=update
7
```

Рисунок 3.3 – Налаштування application.properties

Де url – це посилання на створену базу даних, яке має вигляд jdbc:postgresql://localhost:5432/test, де localhost:5432 – адреса хоста, а test – власне ім'я БД, до якої ми підключаємось. Username та password – це дані користувача для входу в БД. Вони необхідні для реалізації безпеки та запобігання несанкціонованим діям з БД.

Параметр spring.jpa.hibernate.ddl-auto=update вказує на те, що структура бази даних буде автоматично оновлюватися відповідно до змін у моделях даних, визначених у застосунку. Це спрощує розробку та підтримку бази даних, оскільки не потрібно вручну вносити зміни до схеми БД при оновленні моделей.

Після підключення до бази даних потрібно створити класи-моделі для реалізації сутностей цієї бази даних. Приклад реалізації моделі "User" зображено на рисунку 3.4.

```
9   @Entity  Vodzinskiy *
10  @Table(name = "users")
11  @Getter
12  @Setter
13  @NoArgsConstructor
14  @AllArgsConstructor
15  public class User {
16
17      @Id
18      @GeneratedValue(strategy = GenerationType.UUID)
19      private UUID id;
20
21      @Column(nullable = false)
22      private String username;
23
24      @Column(unique = true)
25      private String email;
26
27      @Column(unique = true)
28      private String providerId;
29
30      private String password;
31
32      @ManyToMany(mappedBy = "members")
33      private Set<Project> projects;
```

Рисунок 3.4 – Модель User

Поля класу відповідають стовпцям в таблиці "users" бази даних. З основних анотацій можна виділити: @Entity, визначає що цей клас є сутністю і його потрібно зберігати в базі даних; @Table(name = "users") - задає назву таблиці в базі даних для цієї сутності; @GeneratedValue - вказує, що значення id буде автоматично генеруватися як UUID та @Column, яка використовується для налаштування відповідних стовпців в таблиці.

Анотація @ManyToMany(mappedBy = "members") встановлює двонаправлений зв'язок "багато до багатьох" між сутностями User та Project. Параметр mappedBy вказує на поле members в класі Project, яке є відповідальним за зв'язок з боку Project.

Для взаємодії з базою даних всередині застосунку необхідно створити репозиторій. У Spring Data JPA репозиторії представлені інтерфейсами, які розширюють інтерфейс JpaRepository. Приклад реалізації інтерфейсу "UserRepository" зображено на рисунку 3.5.

```
10    @Repository  10 usages  Vodzinskiy
11    public interface UserRepository extends JpaRepository<User, UUID> {
12        Optional<User> findByEmail(String email);  2 usages  Vodzinskiy
13        Optional<User> findByProviderId(String username);  2 usages  Vodzinskiy
14        boolean existsByEmailOrUsername(String email, String username);  1 usage
15    }
```

Рисунок 3.5 – Інтерфейс UserRepository

Анотація @Repository вказує, що цей інтерфейс є репозиторієм і має бути автоматично створений як бін Spring. Розширюючи інтерфейс JpaRepository, UserRepository отримує базові методи для роботи з даними, а саме: збереження, оновлення, видалення та пошук. Крім того, UserRepository містить додаткові методи, наприклад, findByEmail(), для отримання користувача за полем "email".

3.1.3 Розробка контролерів та сервісів

Контролери є ключовою частиною серверної частини веб-застосунку і відповідають за обробку запитів від клієнтської частини та повернення відповідей. В моєму застосунку контролери побудовані на базі архітектурного стилю REST [17]. Це означає, що вони використовують стандартні HTTP-методи, такі як GET, POST, PUT і DELETE для взаємодії з ресурсами.

Приклад реалізації REST контролера для проєкту зображена на рис. 3.6.

```
14  @RestController  Vodzinskiy *
15  @RequestMapping("/api/project")
16  @RequiredArgsConstructor
17  public class ProjectController {
18      private final ProjectService projectService;
19      private final UserService userService;
20
21      @PostMapping  Vodzinskiy
22      public ResponseEntity<ProjectResponse> createProject(@RequestBody String name, HttpSession session) {
23          return ResponseEntity.status(HttpStatus.CREATED).body(
24                  projectService.createProject(userService.getIdFromSession(session), name));
25      }
26
27      @PostMapping("/join/{id}")  Vodzinskiy
28      public ResponseEntity<ProjectResponse> joinProject(@PathVariable UUID id, HttpSession session) {
29          return ResponseEntity.ok(projectService.joinProject(id, userService.getIdFromSession(session)));
30      }
31
32      @GetMapping("/{id}")  new *
33      public ResponseEntity<ProjectResponse> getProject(@PathVariable UUID id, HttpSession session) {
34          return ResponseEntity.ok(projectService.getProjectById(id, userService.getIdFromSession(session)));
35      }
36
```

Рисунок 3.6 – ProjectController

Цей контролер обробляє HTTP-запити, пов'язані з операціями над проєктами, використовуючи відповідні анотації та методи. Для створення контролера необхідно додати до класу анотацію @RestController. Вона вказує, що цей клас є контролером і буде обробляти HTTP-запити та повертати відповіді у форматі JSON або XML. За потреби, як в прикладі, можна використати анотацію @RequestMapping до класу, щоб вказати базовий шлях для всіх методів у цьому контролері.

На прикладі методу joinProject розберемо загальну будову методу контролера. Анотація @PostMapping("/join/{id}") вказує, що метод обробляє POST-запити на шлях "/join/{id}". Частина {id} є змінною частиною шляху. Отримуємо її за допомогою анотації @PathVariable UUID id і далі можемо використовувати як звичайну змінну в методі. Аналогічно можна отримувати і тіло запиту за допомогою анотації @RequestBody. В тілі методу викликається метод joinProject з сервісу ProjectService, який робить всю необхідну логіку та повертає об’єкт ProjectResponse, який містить всю необхідну для користувача

інформацію. Сам же метод контролера "обгортає" ProjectResponse в об'єкт ResponseEntity, встановлює відповідний HTTP-статус (в даному випадку 200 OK) та повертає його як тіло HTTP-відповіді.

Реалізація сервісу для роботи з проєктами (ProjectServiceImpl) зображена на рис. 3.7. У ній я використовую інтерфейс ProjectService, який визначає всі необхідні методи для роботи з проєктами. Цей інтерфейс імплементується класом ProjectServiceImpl, що містить конкретну реалізацію цих методів. Такий підхід має кілька переваг. По-перше, він дозволяє абстрагувати деталі реалізації сервісу за інтерфейсом, що спрощує взаємодію з сервісом для інших компонентів системи. По-друге, використання інтерфейсу забезпечує гнучкість та можливість заміни реалізації сервісу в майбутньому без впливу на інші частини програми. Крім того, такий підхід полегшує тестування сервісу, оскільки дозволяє створювати mock-об'єкти на основі інтерфейсу для модульних тестів.

```
20    @Service  Vodzinskiy *
21    @RequiredArgsConstructor
22    public class ProjectServiceImpl implements ProjectService {
23        private final ProjectRepository projectRepository;
24        private final UserService userService;
25        private final SocketIOServer server;
26
27        @Override  1 usage  Vodzinskiy
28        public ProjectResponse createProject(UUID ownerId, String name) {
29            User owner = userService.getUser(ownerId);
30            Project project = new Project(name, owner);
31            projectRepository.save(project);
32            return new ProjectResponse(project.getId(), project.getName(), owner.getUsername(), new HashSet<>(), Role.OWNER);
33        }
34
35        @Override  1 usage  Vodzinskiy *
36        public ProjectResponse joinProject(UUID projectId, UUID userId) {
37            Project project = getProject(projectId);
38            User user = userService.getUser(userId);
39            if (!(project.getMembers().contains(user) && project.getOwner().getId().equals(userId))) {
40                project.addMember(user);
41                projectRepository.save(project);
42            }
43            updateMembersList(project);
44            return projectToProjectResponse(project, userId);
45        }
```

Рисунок 3.7 – ProjectServiceImpl

Щоб створити сервіс в Spring, потрібно додати анотацію @Service до класу, який реалізує логіку цього сервісу. Анотація @Service є спеціальним маркером, який вказує Spring, що даний клас є сервісом і повинен бути зареєстрований в контейнері IoC для подальшого використання.

Розглянемо як приклад роботу методу joinProject. Метод дозволяє користувачеві приєднатися до проєкту за допомогою ідентифікаторів проєкту (projectId) та користувача (userId). Спочатку метод отримує об'єкти Project та User, використовуючи відповідні методи. Далі відбувається перевірка, чи користувач вже є учасником проєкту або його власником. Якщо користувач ще не приєднаний до проєкту, він додається до списку учасників проєкту. Після чого оновлений проєкт зберігається в базу даних. Метод updateMembersList оновлює список учасників проєкту та сповіщає всіх користувачів про приєднання нового учасника за допомогою механізму сокетів. Нарешті, метод joinProject повертає об'єкт ProjectResponse, який містить необхідну інформацію про проєкт.

3.1.4 Впровадження механізмів безпеки

Безпека в веб-застосунках є надзвичайно важливою. Spring Framework надає потужні та гнучкі механізми безпеки, що дозволяють впроваджувати надійні заходи захисту в своїх застосунках [18].

Важливою складовою безпеки є впровадження механізму автентифікації користувачів. Одним із основних підходів до автентифікації в Spring є використання сесій. Сесія - це спосіб збереження інформації про автентифікованого користувача протягом певного періоду часу. Коли користувач успішно проходить автентифікацію, створюється сесія, яка зберігає дані про користувача на сервері. Кожній сесії присвоюється унікальний ідентифікатор, який передається клієнту у вигляді файлу cookie [19]. Цей ідентифікатор також зберігається на сервері, в моєму випадку для збереження використовується база даних типу "ключ-значення" – Redis.

При наступних запитах клієнт надсилає отриманий ідентифікатор сесії разом із запитом на сервер. Сервер перевіряє наявність сесії з таким ідентифікатором у базі даних Redis. Якщо сесія знайдена, то сервер може отримати збережені дані про користувача з сесії та використовувати їх для перевірки прав доступу користувача та персоналізації відповіді. Однією з ключових переваг сесій, в порівнянні з іншим популярним підходом, а саме використання токенів доступу, таких як JWT, є можливість миттєвого відкликання доступу. Якщо адміністратор або система безпеки виявляє підозрілу активність або потребує негайного відкликання доступу, сесія може бути знищена на сервері, що миттєво припинить доступ користувача. З JWT відкликання доступу є більш складним, оскільки токени зберігаються на стороні клієнта і мають встановлений термін дії [20].

Ще одним важливим аспектом безпеки застосунку є хешування паролів перед збереженням в базі даних. Хешування - це процес перетворення вхідних даних будь-якого розміру в унікальний вихідний рядок фіксованої довжини, який називається хеш-значенням. Хешування є односторонньою операцією, тобто з хеш-значення неможливо відновити оригінальні дані [21]. В моєму випадку для хешування паролю використовується вбудований в Spring Security клас BCryptPasswordEncoder. Він використовує алгоритм хешування BCrypt для безпечного збереження паролів.

Під час автентифікації користувача введений пароль передається в BCryptPasswordEncoder для хешування з використанням тієї ж солі, яка була збережена разом із хеш-значенням пароля в базі даних. Потім хеш-значення порівнюються, щоб визначити, чи є введений пароль правильним.

Також в застосунку реалізована можливість входу користувачів за допомогою їх облікових записів Google або GitHub. Для забезпечення цієї функціональності використовується протокол авторизації OAuth 2.0.

OAuth 2.0 - це відкритий стандарт авторизації, який дозволяє користувачам надавати доступ до своїх ресурсів на одному сервісі іншому

сервісу без необхідності передавати свої облікові дані. Це забезпечує безпечний спосіб делегування доступу до ресурсів користувача [22].

Коли користувач обирає вхід через Google або GitHub, застосунок перенаправляє його на сторінку автентифікації відповідного сервісу. На цій сторінці користувач повинен ввести свої облікові дані (або вибрати запропонований акаунт, якщо він вже авторизований) та підтвердити, що він дозволяє застосунку доступ до певних даних або ресурсів його аккаунту. Після чого сервіс автентифікації генерує спеціальний код авторизації та передає його назад до застосунку. Застосунок використовує цей код, щоб отримати токен доступу від сервісу автентифікації. Маючи дійсний токен доступу, застосунок може робити запити до API відповідного сервісу, щоб отримати необхідні дані про користувача.

3.1.5 Реалізація веб-сокет сервера

Веб-сокет - це протокол зв'язку, що забезпечує повний дуплексний канал зв'язку поверх одного TCP-з'єднання. Він дозволяє серверам ініціювати передачу даних до клієнтів, а не тільки відповідати на запити від клієнтів, як це відбувається у традиційній моделі HTTP [23]. Ця технологія ідеально підходить для реалізації миттєвого відображення змін в файлі для інших користувачів, що необхідно в моєму застосунку.

Для реалізації веб-сокет сервера було обрано бібліотеку SocketIO. Це популярна бібліотека для створення веб-сокет з'єднань, яка надає додаткові функціональні можливості та спрощує роботу з веб-сокетами [24]. Перевагами використання саме цієї бібліотеки є можливість автоматичного відновлення з'єднання у разі його втрати та підтримка кімнат та просторів імен, що спрощує організацію та управління з'єднаннями.

Для створення веб-сокет серверу на основі бібліотеки SocketIO необхідно спочатку встановити залежність, використовуючи Maven, а саме com.corundumstudio.socketio:netty-socketio. Після чого потрібно створити конфігурацію сервера, де необхідно вказати хост, порт та інші налаштування за необхідності. Далі створити клас, який буде ініціалізувати SocketIO сервер після запуску застосунку. Приклад реалізації класу зображено на рис. 3.8.

```
10  public class SocketServer implements CommandLineRunner {
11
12      private final SocketIOServer server;
13
14      @Override  Vodzinskiy
15      public void run(String... args) { server.start(); }
18  }
```

Рисунок 3.8 – SocketServer

Після цих дій, можна створювати обробники подій. Наприклад, за допомогою анотації @OnConnect можна визначити логіку при підключенні клієнта до веб-сокет сервісу, а за допомогою @OnDisconnect – при відключенні.

Розглянемо як приклад метод, що обробляє подію оновлення файлу, зображеного на рис. 3.9.

```
@OnEvent("updateFile")  Vodzinskiy *
public void onUpdateFile(SocketIOClient client, List<Map<String, Object>> change, UUID fileId) {
    String projectId = client.getHandshakeData().getSingleUrlParam( name: "projectId");
    if (projectId != null) {
        server.getRoomOperations(projectId).getClients().stream()
                .filter(c -> !c.getSessionId().equals(client.getSessionId()))
                .forEach(c -> c.sendEvent( s: "fileUpdates", change, fileId));
    }
}
```

Рисунок 3.9 – Реалізація методу onUpdateFile

Цей метод є обробником події "updateFile" для веб-сокет з'єднань. Він викликається при оновленні файлу клієнтом і відповідає за розсилку цих оновлень іншим клієнтам у тій самій кімнаті проєкту. Метод приймає зміни у файлі та ідентифікатор файлу. Спочатку він отримує ідентифікатор проєкту з даних клієнта. Якщо ідентифікатор наявний, метод отримує список підключених до кімнати клієнтів, фільтрує їх, виключаючи ініціатора оновлення і кожному іншому клієнту відправляє подію "fileUpdates" з переданими змінами та ідентифікатором файлу. Таким чином, інші клієнти у кімнаті проєкту отримують повідомлення про оновлення файлу і можуть синхронізувати зміни.

3.2 Розробка клієнтської частини застосунку

Клієнтська частина застосунку розроблятиметься з використанням фреймворку Angular. Для забезпечення привабливого та зручного інтерфейсу застосунку використовуватиметься бібліотека Angular Material.

3.2.1 Ініціалізація проєкту та встановлення залежностей

Для ініціалізації нового проєкту в Angular можна використати Angular CLI [8]. Для цього потрібно виконати команду "ng new my-app", де "my-app" – назва проєкту. Під час ініціалізації проєкту, Angular CLI запропонує кілька опцій конфігурації, а саме: вибір формату таблиці стилів, ввімкнення серверного рендерингу і статичної генерації сайтів. Також за допомогою додаткових ключів можна конфігурувати проєкт. Наприклад, "--routing" автоматично створює модуль маршрутизації для застосунку, а "--skipTests" – запобігає створенню файлів для модульного тестування під час ініціалізації нового проєкту Angular.

У своєму Angular застосунку я використовую низку допоміжних бібліотек для розширення функціональності та спрощення розробки. Серед них:

- ngx-monaco-editor-v2 - це потужний редактор коду із підсвічуванням синтаксису, автодоповненням та іншими можливостями, інтегрований за допомогою Angular обгортки.
- jszip - бібліотека для роботи із ZIP-архівами. Дозволяє стискати, розпаковувати файли та передавати дані в стисненому вигляді.
- socket.io-client - клієнтська бібліотека Socket.IO для організації двосторонньої комунікації між клієнтом і сервером у режимі реального часу.
- rxjs - бібліотека, що реалізує патерн Observer для роботи з асинхронними потоками даних та подіями, використовується в Angular для обробки HTTP-запитів, подій введення та інших асинхронних операцій [25].

3.2.2 Розробка сервісів

Сервіси відіграють важливу роль у структуруванні та організації коду застосунку. Вони забезпечують спосіб інкапсуляції логіки, яка може бути використана в різних частинах застосунку, сприяючи модульності та повторному використанню коду. Сервіси також використовуються для взаємодії з зовнішніми ресурсами, такими як API та веб-сокет сервер.

В Angular створити сервіс можна вручну, створивши typescript файл, в якому реалізувати повний обсяг необхідного коду, або використовуючи Angular CLI. Для створення сервісу, використовуючи командний рядок, необхідно в потрібній директорії виконати команду "ng generate service name", де "name" це назва сервісу. Після чого автоматично буде згенерований файл із назвою name.service.ts у відповідній директорії. Цей файл буде містити базовий клас із декоратором @Injectable(). Усередині класу можна визначати методи, властивості та залежності, необхідні для реалізації логіки сервісу.

Розглянемо приклад реалізації сервісу для взаємодії з проєктом, частково зображеного на рис 3.10.

```
@Injectable({  Show usages  Vodzinskiy *
  providedIn: 'root'
})
export class ProjectService {
  private apiUrl : string  = env.API_URL;
  private projectSubject : BehaviorSubject<Project | null... = new BehaviorSubject<Project | null>( _value: null);
  public project$ : Observable<Project | null>  = this.projectSubject.asObservable();

  constructor(private http: HttpClient, private router: Router,  no usages  Vodzinskiy
              private fileService: FileService, private socket: SocketService) {}

  public joinProject(projectId: string) : Observable<HttpResponse<Projec... {  Show usages  Vodzinskiy *
    if (this.projectSubject.value !== null) {
      this.leaveProject(this.projectSubject.value?.id)
    }
    return this.http.post<Project>( url: `${this.apiUrl}project/join/${projectId}`,  body: {},
      options: {observe: 'response', withCredentials: true})
  }

```

Рисунок 3.10 – Сервіс для взаємодії з проєктом

Необхідною складовою для створення сервісу є декоратор @Injectable, який розміщується безпосередньо перед оголошенням класу. Даний декоратор вказує, що клас є сервісом, який може бути ін'єктований в інші частини застосунку. Параметр "root" вказує на те, що сервіс буде доступний у всьому застосунку Angular.

На прикладі "joinProject" розглянемо реалізацію методу для взаємодії з сервером. Це публічний метод, який приймає параметр projectId типу string. Метод відповідає за приєднання користувача до проєкту за вказаним ідентифікатором. Спочатку відбувається перевірка, чи користувач приєднаний до іншого проєкту. Якщо так, він автоматично від'єднується від того проєкту. Потім метод відправляє POST-запит на URL-адресу, що складається з базової URL-адреси this.apiUrl, шляху "project/join/" та значення projectId. Тіло запиту є порожнім об'єктом {}. Додатково вказуються опції запиту: observe зі значенням 'response' для отримання повної відповіді HTTP та withCredentials зі значенням true для відправки файлів cookie разом із запитом. Метод повертає Observable<Project>, тому що HTTP-запити виконуються асинхронно.

Observable дозволяє обробляти потік даних, реагувати на успішне завершення запиту, обробляти помилки та виконувати додаткові дії з отриманими даними за допомогою операторів RxJS.

3.2.3 Створення компонентів

Компонент - це автономний блок коду, який інкапсулює пов'язану функціональність, шаблон і стилі. Він слугує будівельним блоком для створення складних інтерфейсів користувача шляхом композиції менших компонентів. Використання компонентів в Angular має ряд переваг, зокрема підвищення читабельності коду, полегшення обслуговування та можливість повторного використанню коду в усьому застосунку.

Як і сервіс, компонент можна реалізувати вручну, створивши typescript файл, який містить його код, та за потреби html та css файли або за допомогою Angular CLI. Щоб створити компонент за допомогою командного рядка, потрібно перейти в необхідну директорію і виконати команду "ng generate component name", замінивши "name" на бажану назву компонента. Після чого автоматично створиться нова директорія, яка міститиме в собі згенеровані typescript, html та css файли.

Після створення компонент є автономним, для його використання в застосунку необхідно зареєструвати цей компонент в модулі Angular. Реєстрація компонента в модулі дозволяє використовувати його в інших компонентах та сервісах, що належать до цього модуля.

Компоненти в Angular мають свій життєвий цикл, який складається з декількох етапів: ініціалізація, оновлення, знищення та інші. На кожному з цих етапів можна виконувати певні дії, використовуючи спеціальні методи життєвого циклу, такі як ngOnInit, ngOnChanges, ngOnDestroy тощо. Ці методи дозволяють реагувати на зміни в даних чи властивостях компонента, виконувати необхідні обчислення чи запити до сервера.

Загальна структура Angular компоненту включає: імпорти модулів і сервісів, декоратор @Component з метаданими (селектор, шаблон, стилі) та клас компонента, який містить необхідні поля та модулі. Приклад реалізації Angular компоненту зображено на рис. 3.11.

```
@Component({  Show usages  Vodzinskiy
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrl: './signup.component.scss'
})
export class SignupComponent {
  signupForm : FormGroup<{name: FormControl<s...  = new FormGroup( controls: {
    name: new FormControl( value: '', Validators.required),
    email: new FormControl( value: '',  validatorOrOpts: [Validators.required, Validators.email]),
    password: new FormControl( value: '', Validators.required)
  });

  constructor(private router: Router, private authService: AuthService, private userService: UserService) {}

  getControl(name: string): FormControl {  Show usages  Vodzinskiy
    return this.signupForm.get(name) as FormControl;
  }

  onSubmit() : void  {  Show usages  Vodzinskiy
    if (this.signupForm.valid) {
      const user: User = {
```

Рисунок 3.11 – Компонент сторінки реєстрації

Цей компонент відповідає за відображення сторінки реєстрації користувача. Він є частиною загальної структури застосунку і викликається через систему маршрутизації Angular. Також містить в собі додаткові компоненти, які викликаються в html коді. Наприклад, виклик за допомогою селектору "app-custom-input" компоненту CustomInput, який відповідає за поле вводу та вивід підказок чи помилок користувачу, зображено на рис.3.12.

```
<mat-card-content>
  <form class="d-flex flex-column align-items-center" [formGroup]="signupForm">
    <app-custom-input [control]="getControl( name: 'name')"
      formControlName="name"
      placeholder="Ім'я"
      requiredErrorMessage="Введіть своє ім'я">
    </app-custom-input>
```

Рисунок 3.12 – Виклик компоненту CustomInput

3.2.4 Реалізація веб-сокет клієнта та синхронізації змін

Раніше було описано реалізацію веб-сокет серверу. Для повноцінної роботи веб-сокетів, потрібно також створити веб-сокет клієнт, використовуючи бібліотеку Socket.IO [24]. Реалізацію якого і буде описано в цьому підрозділі.

Взаємодію з веб-сокет сервером найкраще реалізовувати в окремому сервісі. Загалом, структура такого сервісу не сильно відрізняється від сервісу для взаємодії з API, який описувався раніше, з тією відмінністю, що замість HTTP-протоколу використовується протокол WebSocket, а замість HTTP методів застосовуються методи "on" для відстеження подій та "emit" для відправлення подій на сервер.

Приклад реалізації сервісу для взаємодії з файлом зображено на рис. 3.13.

```
6   @Injectable({  Show usages  Vodzinskiy *
7     providedIn: 'root'
8   })
9   export class FileSocketService {
10
11    constructor(public socket: SocketService) {}  no usages  Vodzinskiy
12
13    updateFile(change: any, fileId: string): void {  Show usages  new *
14      this.socket.getSocket().emit('updateFile', change, fileId);
15    }
```

Рисунок 3.13 – Веб-сокет сервіс

Розберемо роботу методів для передачі даних через WebSocket на прикладі методу updateFile. Цей метод дозволяє клієнту надсилати серверу інформацію про зміни у файлі для його оновлення в реальному часі. Він приймає два параметри: change, який містить дані про зміни у файлі, та fileId - ідентифікатор файлу, що потребує оновлення. В тілі методу updateFile викликається метод emit об'єкта socket, який відповідає за передачу даних

через WebSocket. Метод emit відправляє на сервер подію "updateFile" разом із параметрами change та fileId, сигналізуючи про необхідність оновлення вказаного файлу зі змінами.

Сервер, отримавши подію "updateFile", розсилає зміни усім користувачам проєкту (окрім автора змін) через подію "fileUpdates", яка містить зміни та ідентифікатор файлу. Отримуючи цю подію клієнтська частина застосунку оновлює вміст файлу, додаючи та видаляючи лише ті символи, що були змінені, а не весь файл.

Виявлення та застосування змін відбувається за допомогою методів бібліотеки Monaco Editor. Вона забезпечує багатий функціонал, включаючи підсвічування синтаксису для різних мов програмування, навігацію по коду та інтелектуальні підказки. Ця бібліотека використовується для реалізації редактора коду. Код для відстеження змін у файлі зображено на рис.3.14.

```
this.editor.onDidChangeModelContent((event: any) : void  => {
  if (!this.isRemoteChange && !event.isFlush) {
    const operations = event.changes.map((change: any) : {range: any, text: any}  => ({
      range: change.range,
      text: change.text
    }));
    this.socket.updateFile(operations, this.file.id);
  }
  this.isRemoteChange = false;
});
```

Рисунок 3.14 – Реалізація виявлення змін

Коли користувач вносить зміни в редакторі, метод onDidChangeModelContent() відстежує ці зміни. Всередині цього методу перевіряється, чи зміни не прийшли від іншого користувача (змінна isRemoteChange), щоб уникнути нескінченного переходу змін між користувачами. Також перевіряється, чи подія не є результатом операції очищення буфера. Далі метод формує масив об'єктів, які представляють кожну зміну в редакторі. Кожен об'єкт містить інформацію про діапазон зміни та новий текст.

Цей масив змін передається на сервер за допомогою методу updateFile() сервісу FileSocketService разом з ідентифікатором файлу. При надходженні змін використовується метод applyEdits(), який застосовує отримані зміни у файл, як це зображено на рис.3.15.

```
this.socket.fileUpdated(this.file.id).subscribe( observerOrNext: (operations: any) : void => {
  if (this.editor && this.editor.getModel()) {
    this.isRemoteChange = true;
    this.editor.getModel().applyEdits(operations);
  }
});
```

Рисунок 3.15 – Застосування отриманих змін

Таким чином, зміни, отримані від сервера, відображаються в редакторі клієнта, забезпечуючи синхронізацію файлу між усіма учасниками проєкту.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було розглянуто процес розробки веб-застосунку для спільного програмування. Для розробки були використані сучасні та потужні рішення, зокрема фреймворк Spring для серверної частини та фреймворк Angular для клієнтської частини.

При розробці серверної частини особливу увагу було приділено таким важливим аспектам, як ініціалізація проєкту, встановлення залежностей, інтеграція з базами даних PostgreSQL та Redis, розробка контролерів і сервісів, впровадження механізмів безпеки, зокрема систему автентифікації з використанням сесій та хешування паролів. Також для зручності та спрощення входу в систему додано автентифікацію за допомогою облікових записів Google або GitHub з використанням протоколу авторизації OAuth 2.0. Використання фреймворку Spring забезпечило надійність, масштабованість та гнучкість серверної частини застосунку. Особливо важливим аспектом розробки було впровадження веб-сокет сервера для забезпечення миттєвого обміну даними між користувачами застосунку. Для реалізації цієї функціональності було використано популярну бібліотеку Socket.IO, яка спрощує процес створення веб-сокет з'єднань та обміну повідомленнями в реальному часі.

Клієнтська частина застосунку була розроблена на основі фреймворку Angular, що дозволило створити динамічний та інтуїтивно зрозумілий інтерфейс. Використання бібліотеки Angular Material допомогло забезпечити сучасний вигляд застосунку. Було розглянуто процес ініціалізації проєкту, встановлення залежностей, розробку сервісів та компонентів. Також особливу увагу було приділено реалізації веб-сокет клієнта, виявленню змін та їх синхронізації між користувачами в режимі реального часу.

РОЗДІЛ 4. ОГЛЯД ТА ТЕСТУВАННЯ РОБОТИ ВЕБ-ЗАСТОСУНКУ

4.1 Огляд процесу реєстрації та автентифікації

Для коректної роботи застосунку потрібно забезпечити користувачам можливість реєстрації в системі та подальшого входу в свій акаунт. Сторінка реєстрації користувача зображена на рис. 4.1.

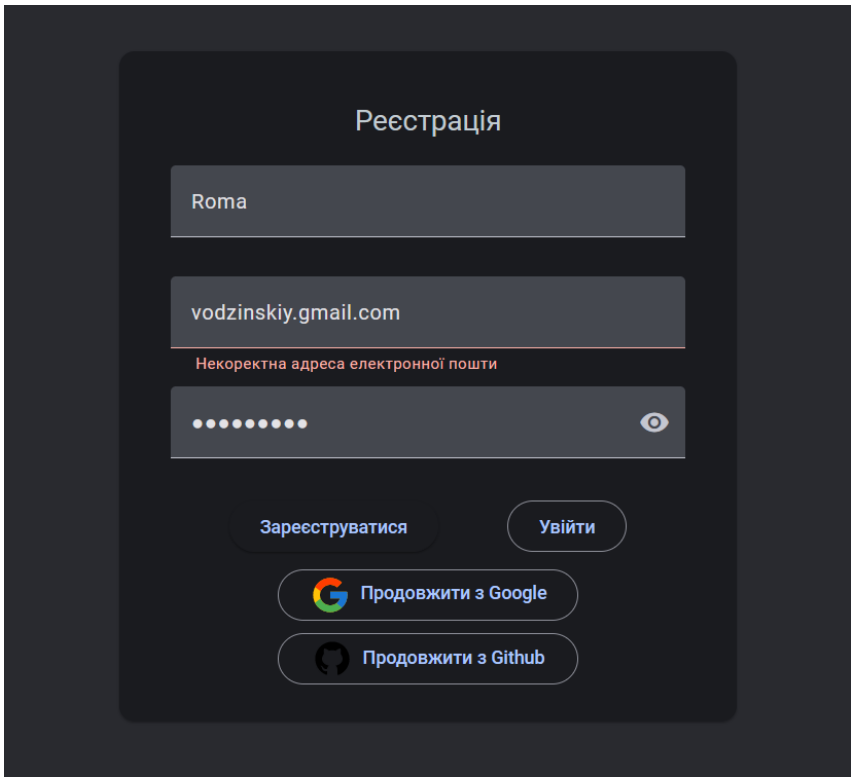


Рисунок 4.1 – Сторінка реєстрації

Під час реєстрації спеціально допущено помилку в написані адреси електронної пошти для демонстрації виведення помилки користувачу.

Якщо користувач вже має акаунт в системі, він може перейти на сторінку входу, яка зображена на рис.4.2. Перевіримо роботу системи автентифікації, спеціально вказавши неправильний пароль.

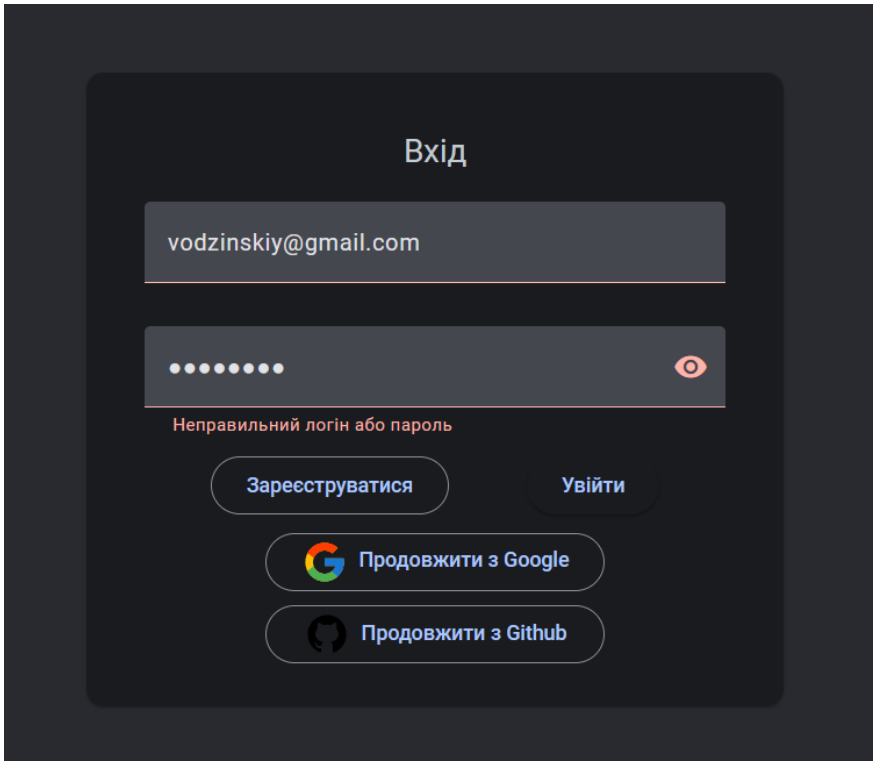


Рисунок 4.2 – Сторінка автентифікації

Як бачимо на рисунку, система інформує, що введені данні не є правильними. Після виправлення паролю на коректний та повторного входу користувача перенаправляє на головну сторінку. В застосунку також реалізована система входу за допомогою акаунтів Google та Github. Коли користувач успішно автентифікується через акаунт одного з цих сервісів, система отримує необхідні дані про користувача, такі як ім'я та електронна пошта. Ця інформація використовується для автоматичної реєстрації користувача в застосунку. Таким чином, користувач може швидко розпочати роботу з застосунком без необхідності проходити повну процедуру реєстрації. Це також підвищує безпеку, оскільки користувачам не потрібно створювати та запам'ятовувати окремі паролі для кожного застосунку.

4.2 Опис головної сторінки застосунку

Після успішного входу в застосунок користувач перенаправляється на головну сторінку. Головна сторінка застосунку складається з трьох основних компонентів: панелі навігації, файлового дерева та області редагування.

Панель навігації, зображена на рис.4.3, розташована у верхній частині сторінки і містить 4 елемента.

Рисунок 4.3 – Панель навігації

Перший елемент - це меню "Файл", яке надає користувачам доступ до операцій з файлами та директоріями, таких як створення нового файлу чи директорії, відкриття існуючого та завантаження всього проєкту локально на комп’ютер.

Другий елемент навігаційної панелі - меню "Проєкт", яке надає користувачам доступ до функцій управління проєктами. Це меню включає опції для створення нового проєкту, приєднання до існуючого та від'єднання від поточного проєкту. Крім того, якщо користувач є власником проєкту, меню "Проєкт" надає можливість видалення всього проєкту.

Третій елемент навігаційної панелі - меню з інформацією про поточний проєкт. Назва меню відображає ім'я проєкту. У верхній частині меню розміщена кнопка для копіювання ідентифікатора. Нижче знаходиться динамічний список учасників, який оновлюється в реальному часі.

Четвертий елемент навігаційної панелі - меню профілю користувача, яке містить два елементи: перший дозволяє перейти на сторінку редагування особистих даних, а другий - здійснити вихід з облікового запису.

Приклад файлового дерева зображено на рис.4.4. Воно відповідає за відображення ієрархічної структури файлів та папок проєкту. Файлове дерево дозволяє користувачам зручно переміщуватись по проєкту, переглядати його вміст та швидко отримувати доступ до потрібних файлів.

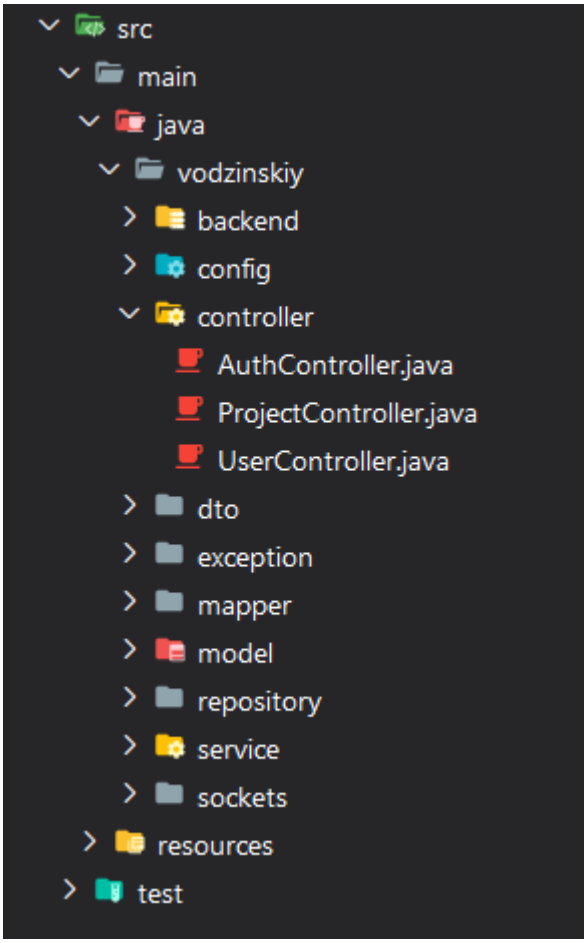


Рисунок 4.4 – Файлове дерево

Файлове дерево також надає можливість виконувати різні операції з файлами та папками: створення нових елементів, перейменування та видалення. Ці операції доступні через контекстне меню, яке з'являється при натисканні правою кнопкою миші на елемент дерева.

За потреби можна змінити ширину області з файловим деревом, перетягуючи її межі мишкою, щоб надати більше простору редактору коду або навпаки. Також можливо повністю приховати цю область, щоб редактор займав усю ширину екрана.

Повний інтерфейс головної сторінки застосунку зображений на рис. 4.5.

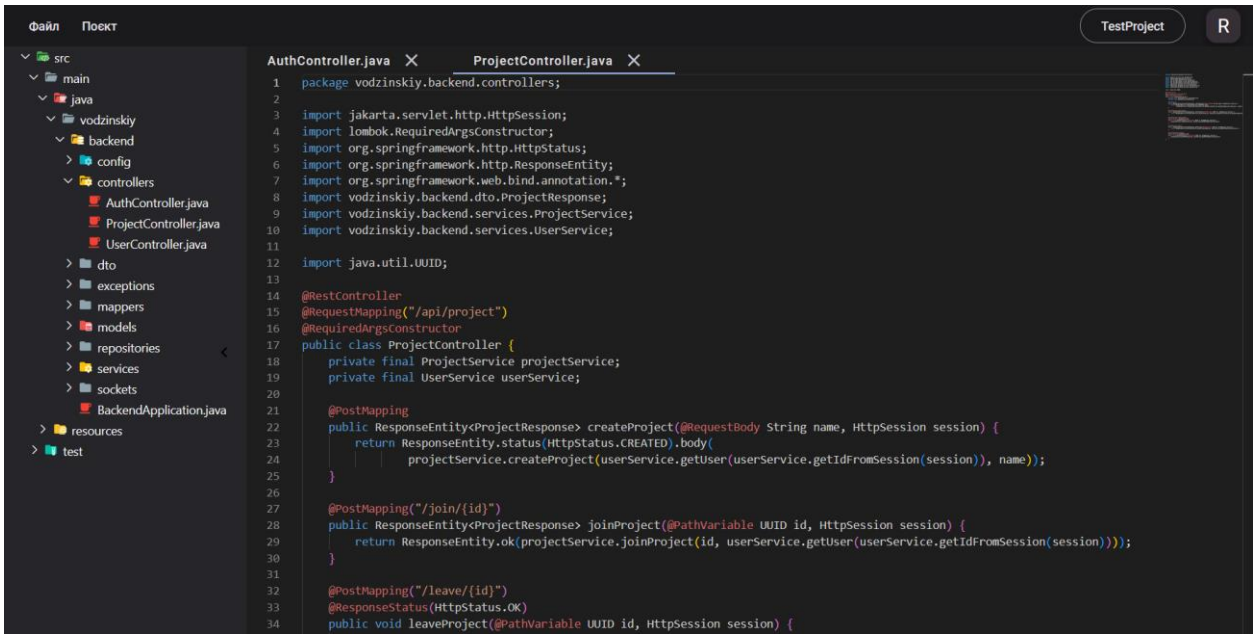


Рисунок 4.5 – Головна сторінка

Для зручності користування, окрім файлового дерева, присутні вкладки для швидкого перемикання між файлами. В самому редакторі присутні деякі корисні функції, які допомагають полегшувати розробку, наприклад, підсвічування синтаксису, пошук в файлі за допомогою сполучення клавіш Ctrl + F, можливість приховувати тіло методу чи класу.

Вікно створення нового файлу зображено на рис. 4.6. В ньому можна вказати назву файлу разом із розширенням, яке потім буде використовуватись для визначення мови програмування та коректного підсвічування синтаксису.

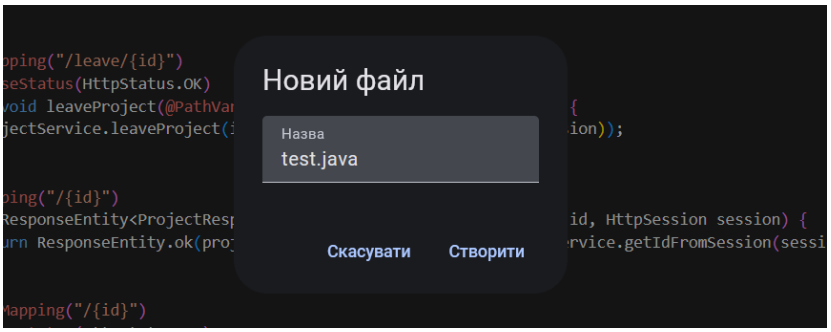


Рисунок 4.6 – Вікно створення файлу

4.3 Опис сторінки профілю

Сторінка для редагування профілю користувача зображена на рис. 4.7. Вона надає можливість користувачу змінювати свою особисту інформацію.

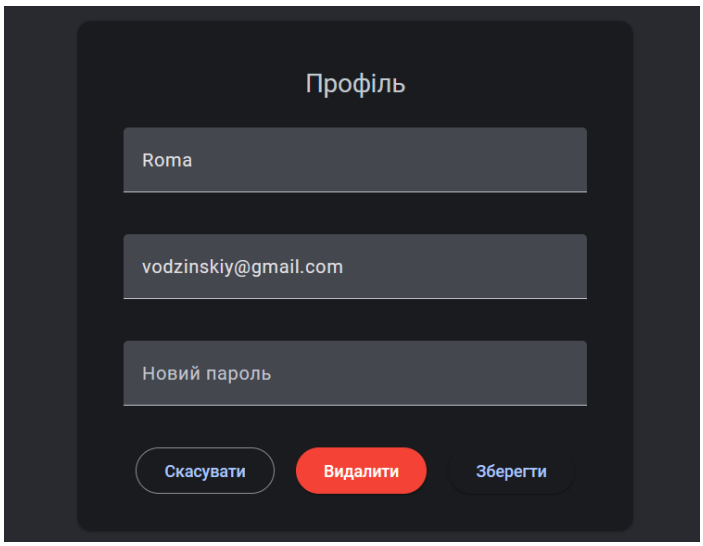


Рисунок 4.7 – Сторінка профілю

Як і на сторінці реєстрації, при редагуванні профілю присутня перевірка електронної адреси на валідність та перевірка, чи не використовує її інший користувач. У разі спроби змінити адресу на вже зареєстровану, відображатиметься помилка (рис. 3.8).

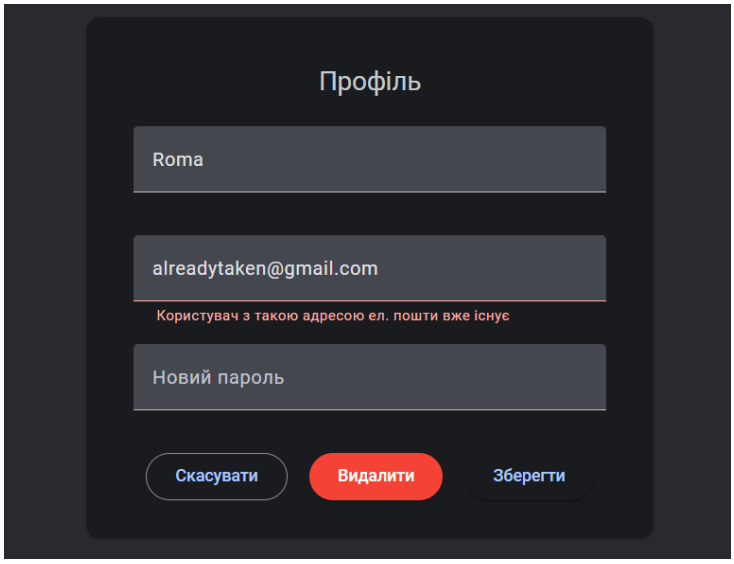


Рисунок 4.8 – Вивід помилки «Пошта вже використовується».

Також є можливість видалення своєї сторінки. В такому випадку користувач автоматично покине всі проєкти, в яких він є учасником. Ті ж проєкти, де він є власником, будуть видалені. Для запобігання випадкового видалення акаунту існує вікно застереження, яке зображено на рис. 4.9.

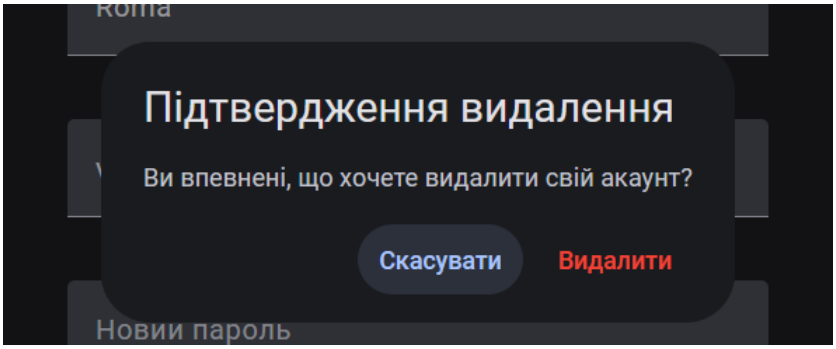


Рисунок 4.9 – Вікно застереження

ВИСНОВОК ДО РОЗДІЛУ 4

В цьому розділі було розглянуто основні функціональні можливості розробленого веб-застосунку. Застосунок надає користувачам зручні інструменти для реєстрації, автентифікації та подальшої роботи над проєктами.

На початку розділу був описаний процес реєстрації та автентифікації користувачів. Застосунок надає можливість створення нового акаунту, а також входу за допомогою існуючих облікових записів Google та Github, що значно спрощує процедуру реєстрації. Також продемонстровано можливість застосунку обробляти та виводити повідомлення про помилки, такі як невалідна адреса електронної пошти, неправильний логін чи пароль тощо.

Далі було детально розглянуто головну сторінку застосунку, яка складається з трьох основних компонентів: панелі навігації, файлового дерева та області редагування. Було описано функціональні можливості кожного компонента.

Також було описано сторінку профілю користувача для редагування особистих даних. Продемонстровано вивід повідомлення про помилку у разі спроби змінити адресу на вже зареєстровану. Описано процес підтвердження та видалення акаунту, а також дії, які після цього відбуваються.

ВИСНОВОКИ

У цій дипломній роботі було розроблено веб-застосунок для спільного програмування в команді. Застосунок надає можливість створення проєктів, приєднання до існуючих, відкриття та створення файлів і директорій в проєкті, редагування коду з миттєвою синхронізацією змін між усіма користувачами проєкту.

У першому розділі було описано специфіку проєкту, визначені основні вимоги до веб-застосунку, також проведено огляд існуючих рішень для спільного програмування. Розглянуто функціонал, переваги та недоліки таких сервісів, як Code With Me, CodeTogether та Codeshare. Визначено, що недоліками наявних рішень є або платна основа, або недостатній функціонал.

Другий розділ присвячено аналізу та вибору технологій для створення веб-застосунку. Для реалізації серверної частини було обрано фреймворк Spring на мові Java завдяки його потужності, надійності та багатому функціоналу. Для клієнтської частини застосунку серед популярних бібліотек та фреймворків було обрано фреймворк Angular через його структурованість та можливість створювати складні інтерактивні інтерфейси. Як основна база даних було обрано PostgreSQL завдяки її потужності, надійності та великому набору функцій. Для зберігання сесій користувачів було обрано Redis через її високу швидкодію та модель ключ-значення, ідеальну для такої задачі.

У третьому розділі описано процес розробки веб-застосунку. Розглянуто створення серверної частини: ініціалізацію проєкту, встановлення залежностей, інтеграцію з базами даних PostgreSQL та Redis, розробку контролерів і сервісів, впровадження механізмів безпеки та реалізацію веб-сокет сервера. Для клієнтської частини описано ініціалізацію Angular проєкту, розробку сервісів для взаємодії з API та веб-сокетами, компонентів, реалізацію веб-сокет клієнта, виявленню змін у редакторі коду для синхронізації змін між користувачами.

Четвертий розділ присвячений огляду та тестуванню функціональних можливостей застосунку розробленого веб-застосунку. Продемонстровано процес реєстрації, автентифікації користувачів та опис головної сторінки з її компонентами. Також описано сторінку профілю для редагування особистих даних користувача.

Таким чином, у результаті дипломної роботи було створено повнофункціональний веб-застосунок для спільного програмування з акцентом на зручність у використанні та ефективну синхронізацію змін між користувачами в режимі реального часу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Getting started with Code With Me. IntelliJ IDEA Help. URL: https://www.jetbrains.com/help/idea/code-with-me.html.
2. Live share ides and coding sessions. CodeTogether. URL: https://www.codetogether.com/live/.
3. Spring projects. Spring. URL: https://spring.io/projects.
4. Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.0/.
5. Express - Node.js web application framework. Express. URL: https://expressjs.com/.
6. Laravel - the PHP framework for web artisans. Laravel. URL: https://laravel.com/docs/11.x.
7. Documentation. Gin Web Framework. URL: https://gin-gonic.com/docs/.
8. Angular documentation. Angular. URL: https://angular.dev/overview.
9. Angular. Angular Material UI component library. Angular Material. URL: https://material.angular.io.
10. Quick start – react. React. URL: https://react.dev/learn.
11. Vue.js - the progressive JavaScript framework. Vue.js. URL: https://vuejs.org/guide/introduction.html.
12. PostgreSQL 16.3 documentation. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/.
13. MySQL 8.4 reference manual. Oracle. URL: https://docs.oracle.com/cd/E17952_01/mysql-8.4-en/index.html.
14. What is mongodb? - mongodb manual v7.0. MongoDB. URL: https://www.mongodb.com/docs/manual/.
15. Redis docs. Redis. URL: https://redis.io/docs/latest/.
16. A complete guide to lombok. Auth0 - Blog. URL: https://auth0.com/blog/a-complete-guide-to-lombok/.

17. Building REST services with Spring. Spring. URL: https://spring.io/guides/tutorials/rest.
18. Spring security. Spring. URL: https://docs.spring.io/spring-security/reference/.
19. Understanding Session-Based Authentication from Scratch. Medium. URL: https://medium.com/@884m884/understanding-session-based-authentication-from-scratch-64110bcfc00f.
20. JSON Web Tokens Introduction. jwt. URL: https://jwt.io/introduction.
21. What is hashing, and how does it work?. Codecademy Blog. URL: https://www.codecademy.com/resources/blog/what-is-hashing/.
22. OAuth 2.0. OAuth Community Site. URL: https://oauth.net/2/.
23. WebSocket. Wikipedia. URL: https://en.wikipedia.org/wiki/WebSocket.
24. Socket.IO documentation. Socket.IO. URL: https://socket.io/docs/v4/.
25. RxJS - introduction. RxJS. URL: https://rxjs.dev/guide/overview.

ДОДАТОК 1

Веб-застосунок для спільного програмування

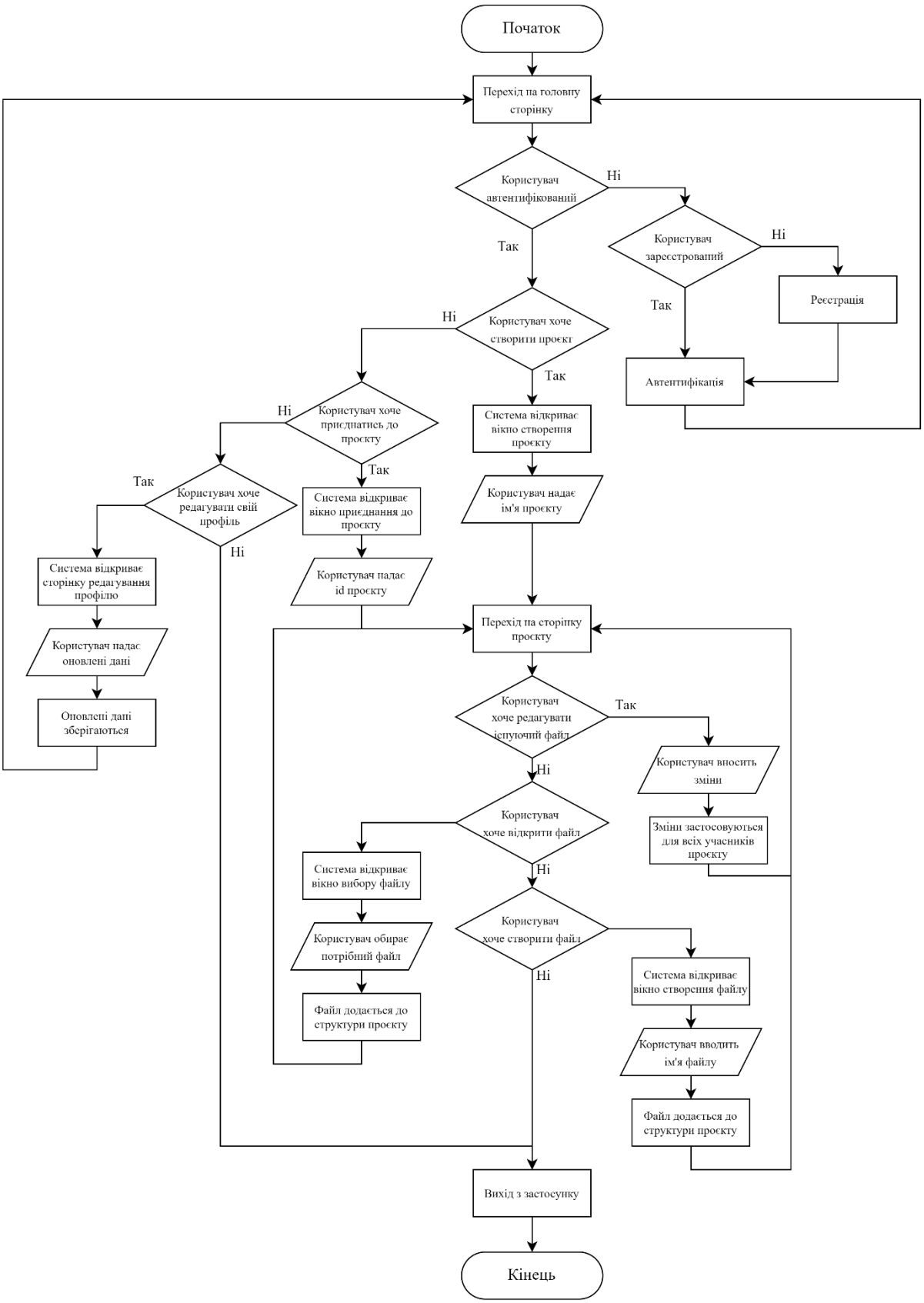
Алгоритм дії розробленого застосунку

(принципова схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Водзінський Р. В.			*Веб-застосунок для спільного програмування* Алгоритм дії розробленого застосунку (принципова схема)	Літ. / Арк. 1 / Аркушів 1
Перевір.		Валько В. В.				
Н. Контр.		Іваніщев Б. В.				***КПІ ім. Ігоря Сікорського, ФІОТ, ІО-04***
Затверд.						

ДОДАТОК 2

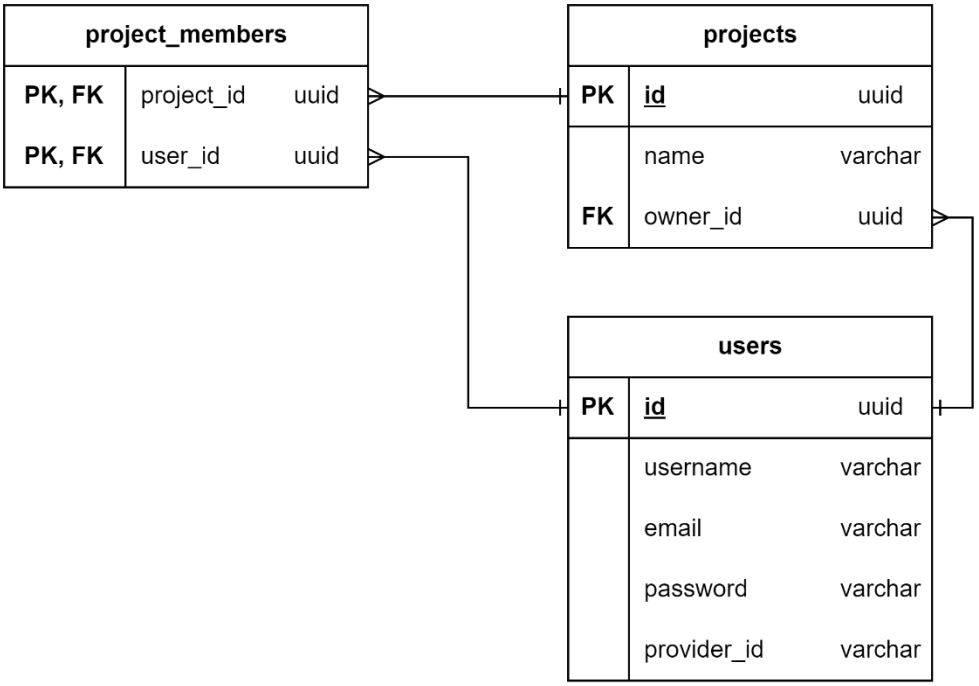
Веб-застосунок для спільного програмування

Діаграма бази даних (функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Водзінський Р. В.			*Веб-застосунок для спільного програмування* Діаграма бази даних (функціональна схема)	Літ. / Арк. 1 / Аркушів 1
Перевір.		Валько В. В.				
Н. Контр.		Іваніщев Б. В.				***КПІ ім. Ігоря Сікорського, ФІОТ, ІО-04***
Затверд.						

ДОДАТОК З

Веб-застосунок для спільного програмування

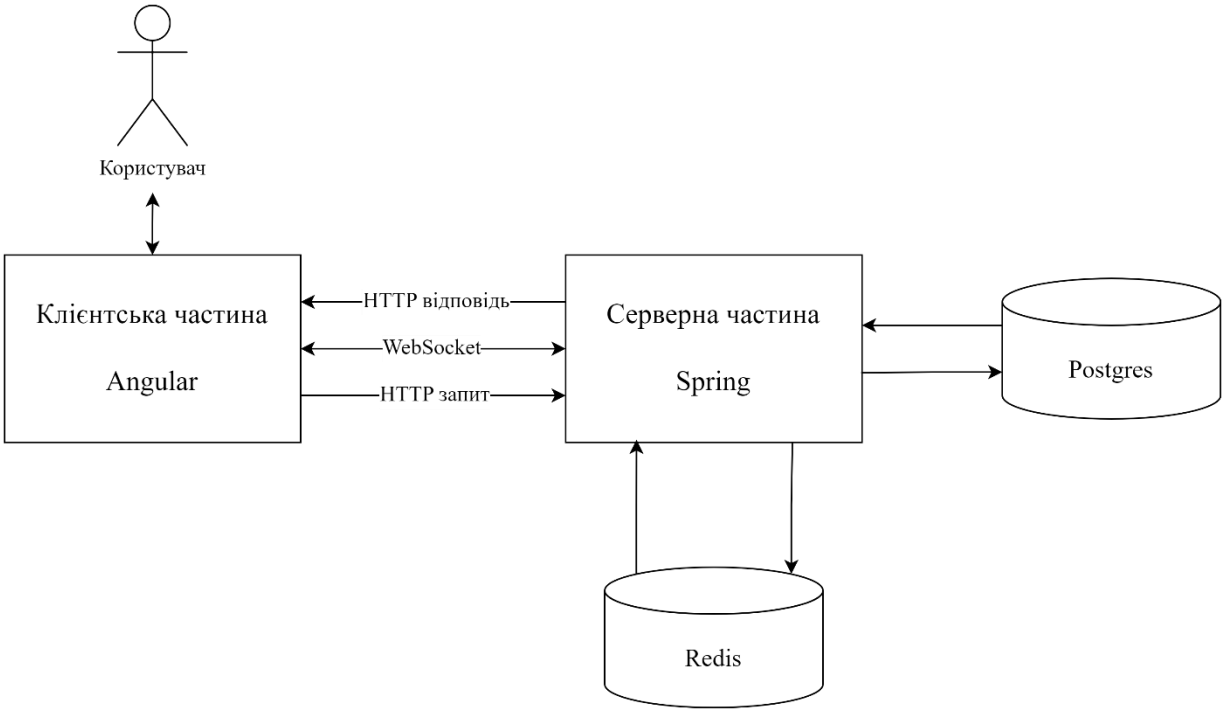
Структура компонентів застосунку

(структурна схема)

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.006 ДЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Водзінський Р. В.			*Веб-застосунок для спільного програмування* Структура компонентів застосунку (структурна схема)	Літ.	Арк. 1 / Аркушів 1
Перевір.		Валько В. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-04	
Н. Контр.		Іваніщев Б. В.					
Затверд.							

ДОДАТОК 4

Веб-застосунок для спільного програмування

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 53

Київ 2024 р

BackendApplication.java

```
package vodzinskiy.backend;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BackendApplication {
	public static void main(String[] args) {
		SpringApplication.run(BackendApplication.class, args);
	}
}
```

AuthController.java

```
package vodzinskiy.backend.controllers;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import vodzinskiy.backend.dto.LoginRequest;
import vodzinskiy.backend.dto.UserResponse;
import vodzinskiy.backend.services.AuthenticationService;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

   private final AuthenticationService authenticationService;

   @PostMapping("/signin")
   public ResponseEntity<UserResponse> signIn(@RequestBody LoginRequest loginRequest,
                                              HttpServletRequest request,
                                              HttpServletResponse response) {;
      return ResponseEntity.ok(authenticationService.signin(loginRequest, request, response));
   }
```

```
    @GetMapping("/session")
    public ResponseEntity<Void> checkSession(Authentication authentication) {
        if (authentication != null && authentication.isAuthenticated()) {
            return ResponseEntity.ok().build();
        } else {
            return ResponseEntity.status(HttpStatus.NOT_MODIFIED).build();
        }
    }
}
```

ProjectController.java

```
package vodzinskiy.backend.controllers;

import jakarta.servlet.http.HttpSession;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import vodzinskiy.backend.dto.ProjectResponse;
import vodzinskiy.backend.services.ProjectService;
import vodzinskiy.backend.services.UserService;

import java.util.UUID;

@RestController
@RequestMapping("/api/project")
@RequiredArgsConstructor
public class ProjectController {
    private final ProjectService projectService;
    private final UserService userService;

    @PostMapping
    public ResponseEntity<ProjectResponse> createProject(@RequestBody String name, HttpSession
session) {
        return ResponseEntity.status(HttpStatus.CREATED).body(
                projectService.createProject(userService.getUser(userService.getIdFromSession(session)),
name));
    }

    @PostMapping("/join/{id}")
    public ResponseEntity<ProjectResponse> joinProject(@PathVariable UUID id, HttpSession session) {
        return ResponseEntity.ok(projectService.joinProject(id,
userService.getUser(userService.getIdFromSession(session))));
    }

    @PostMapping("/leave/{id}")
    @ResponseStatus(HttpStatus.OK)
    public void leaveProject(@PathVariable UUID id, HttpSession session) {
        projectService.leaveProject(id, userService.getIdFromSession(session));
```

```
    }

    @GetMapping("/{id}")
    public ResponseEntity<ProjectResponse> getProject(@PathVariable UUID id, HttpSession session) {
        return ResponseEntity.ok(projectService.getProjectById(id,
userService.getIdFromSession(session)));
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public void deleteProject(@PathVariable UUID id, HttpSession session) {
        projectService.deleteProject(id, userService.getIdFromSession(session));
    }
}
```

UserController.java

```
package vodzinskiy.backend.controllers;

import jakarta.servlet.http.HttpSession;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import vodzinskiy.backend.dto.UserRequest;
import vodzinskiy.backend.dto.UserResponse;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.services.UserService;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    @PostMapping("/auth/signup")
    public ResponseEntity<UserResponse> registerUser(@RequestBody UserRequest request) {
        UserResponse user = userService.createUser(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(user);
    }

    @GetMapping("/user")
    public ResponseEntity<UserResponse> getUser(HttpSession session) {
        User user = userService.getUser(userService.getIdFromSession(session));
        return ResponseEntity.ok(new UserResponse(user.getId(), user.getUsername(), user.getEmail()));
    }

    @DeleteMapping("/user")
    public ResponseEntity<Void> deleteUser(HttpSession session) {
        userService.deleteUser(userService.getIdFromSession(session));
```

```
        return ResponseEntity.ok().build();
    }

    @PatchMapping("/user")
    public ResponseEntity<UserResponse> updateUser(HttpSession session,
                                                   @RequestBody UserRequest userRequest) {
        UserResponse userResponse = userService.editUser(userService.getIdFromSession(session),
userRequest);
        return ResponseEntity.ok(userResponse);
    }
}
```

SecurityConfig.java

```
package vodzinskiy.backend.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration
;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.annotation.web.configurers.SessionManagementConfigurer
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.logout.HttpStatusReturningLogoutSuccessHandler;
import org.springframework.security.web.context.HttpSessionSecurityContextRepository;
import org.springframework.security.web.context.SecurityContextRepository;
import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.repositories.UserRepository;
import vodzinskiy.backend.services.impl.OAuth2UserService;

import java.util.List;
import java.util.Optional;

import static org.springframework.security.config.Customizer.withDefaults;
```

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableRedisHttpSession
public class SecurityConfig {
    private final UserDetailsService userDetailsService;
    private final OAuth2UserService oAuth2UserService;
    private final UserRepository userRepository;

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Bean
    public SecurityContextRepository securityContextRepository() {
        return new HttpSessionSecurityContextRepository();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOriginPatterns(List.of("*"));
        configuration.setAllowedMethods(List.of("*"));
        configuration.setAllowedHeaders(List.of("*"));
        configuration.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
```

```
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/user/**").authenticated()
                .anyRequest().permitAll()
            )
            .logout(logout -> logout.logoutUrl("/api/auth/signout")
                .logoutSuccessHandler(new HttpStatusReturningLogoutSuccessHandler(HttpStatus.OK))
                .logoutSuccessUrl(System.getenv("FRONTEND_URL")))
            .authenticationProvider(authenticationProvider())
            .httpBasic(withDefaults())
            .sessionManagement(sessionManagement -> sessionManagement
                .sessionFixation(SessionManagementConfigurer.SessionFixationConfigurer::newSession)
            )
            .oauth2Login(oauth2 -> oauth2
                .successHandler((request, response, authentication) -> {
                   Optional<User> user = userRepository.findByProviderId(authentication.getName());
                   if (user.isPresent()) {
                      request.getSession().setAttribute("userID", user.get().getId());
                      response.sendRedirect(System.getenv("FRONTEND_URL"));
                   }
                })
                .userInfoEndpoint(userInfo ->
                    userInfo.userService(oAuth2UserService)
                )
            );
        return http.build();
    }
}
```

SocketServer.java

```
package vodzinskiy.backend.config;

import lombok.RequiredArgsConstructor;
import org.springframework.boot.CommandLineRunner;
import com.corundumstudio.socketio.SocketIOServer;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class SocketServer implements CommandLineRunner {
    private final SocketIOServer server;

    @Override
    public void run(String... args) {
        server.start();
    }
}
```

WebSocketConfig.java

```
package vodzinskiy.backend.config;

import com.corundumstudio.socketio.SocketIOServer;
import com.corundumstudio.socketio.annotation.SpringAnnotationScanner;
import com.corundumstudio.socketio.store.RedissonStoreFactory;
import org.redisson.Redisson;
import org.redisson.config.Config;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class WebSocketConfig  {

    @Value("${socket-server.host}")
    private String host;

    @Value("${socket-server.port}")
    private Integer port;

    @Value("${REDIS_URL}")
    private String redisUrl;

    @Bean
    public SocketIOServer socketIOServer() {
        com.corundumstudio.socketio.Configuration conf = new
com.corundumstudio.socketio.Configuration();
        conf.setHostname(host);
        conf.setPort(port);

        Config redissonConfig = new Config();
        redissonConfig.useSingleServer().setAddress(redisUrl);
        Redisson redisson = (Redisson) Redisson.create(redissonConfig);
        RedissonStoreFactory redisStoreFactory = new RedissonStoreFactory(redisson);

        conf.setStoreFactory(redisStoreFactory);
        return new SocketIOServer(conf);
    }

    @Bean
    public SpringAnnotationScanner springAnnotationScanner() {
        return new SpringAnnotationScanner(socketIOServer());
    }
}
```

ErrorResponse.java

```
package vodzinskiy.backend.dto;

import java.time.LocalDateTime;

public record ErrorResponse(LocalDateTime timestamp, int status, String message) {
   public ErrorResponse(String message, int status) {
      this(LocalDateTime.now(), status, message);
   }
}
```

LoginRequest.java

```
package vodzinskiy.backend.dto;

public record LoginRequest(String email, String password) {}
```

ProjectObject.java

```
package vodzinskiy.backend.dto;

import java.util.UUID;

public record ProjectObject(UUID id, String type, String path, String name, String data, String fPath) {}
```

ProjectResponse.java

```
package vodzinskiy.backend.dto;

import java.util.Set;
import java.util.UUID;

public record ProjectResponse(UUID id, String name, String owner, Set<String> members, Role role) {}
```

Role.java

```
package vodzinskiy.backend.dto;

public enum Role {
   OWNER,
   MEMBER
}
```

UserRequest.java

```
package vodzinskiy.backend.dto;

public record UserRequest(String username, String email, String password) {}
```

UserResponse.java

```
package vodzinskiy.backend.dto;

import java.util.UUID;

public record UserResponse(UUID id, String username, String email) {}
```

UserMapper.java

```
package vodzinskiy.backend.mappers;

import org.mapstruct.*;
import vodzinskiy.backend.dto.UserRequest;
import vodzinskiy.backend.models.User;

@Mapper(componentModel = "spring")
public interface UserMapper {
    @BeanMapping(nullValuePropertyMappingStrategy = NullValuePropertyMappingStrategy.IGNORE)
    @Mapping(target = "password", ignore = true)
    void updateUserFromUserRequest(UserRequest userRequest, @MappingTarget User user);
}
```

File.java

```
package vodzinskiy.backend.models;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;
import java.util.UUID;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class File {
    private UUID id;
    private String name;
    private String data;
    private List<File> content;
    private String path;
}
```

Project.java

```
package vodzinskiy.backend.models;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;
import java.util.UUID;

@Entity
@Table(name = "projects")
```

```
@Getter
@Setter
@NoArgsConstructor
public class Project {

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    private String name;

    @ManyToOne
    @JoinColumn(name = "owner_id")
    private User owner;

    @ManyToMany
    @JoinTable(
            name = "project_members",
            joinColumns = @JoinColumn(name = "project_id"),
            inverseJoinColumns = @JoinColumn(name = "user_id")
    )
    private Set<User> members;

    public Project(String name, User owner) {
        this.name = name;
        this.owner = owner;
        this.members = new HashSet<>();
    }

    public void addMember(User user) {
        this.members.add(user);
    }

    public void removeMember(UUID userId) {
        members.removeIf(user -> user.getId().equals(userId));
    }
}
```

User.java

```
package vodzinskiy.backend.models;

import jakarta.persistence.*;
import lombok.*;

import java.util.Set;
import java.util.UUID;

@Entity
@Table(name = "users")
```

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(nullable = false)
    private String username;

    @Column(unique = true)
    private String email;

    @Column(unique = true)
    private String providerId;

    private String password;

    @ManyToMany(mappedBy = "members")
    private Set<Project> projects;

    public User(String username, String email, String password, String providerId) {
        this.username = username;
        this.email = email;
        this.password = password;
        this.providerId = providerId;
    }
}
```

ProjectRepository.java

```
package vodzinskiy.backend.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import vodzinskiy.backend.models.Project;

import java.util.List;
import java.util.UUID;

public interface ProjectRepository extends JpaRepository<Project, UUID> {
    List<Project> findByOwnerId(UUID id);
    List<Project> findByMembersId(UUID id);
}
```

UserRepository.java

```
package vodzinskiy.backend.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import vodzinskiy.backend.models.User;

import java.util.Optional;
import java.util.UUID;

@Repository
public interface UserRepository extends JpaRepository<User, UUID> {
   Optional<User> findByEmail(String email);
   Optional<User> findByProviderId(String username);
   boolean existsByEmail(String email);
}
```

AuthenticationService.java

```
package vodzinskiy.backend.services;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import vodzinskiy.backend.dto.LoginRequest;
import vodzinskiy.backend.dto.UserResponse;

public interface AuthenticationService {
   UserResponse signin(LoginRequest loginRequest, HttpServletRequest request, HttpServletResponse
response);
}
```

ProjectService.java

```
package vodzinskiy.backend.services;

import vodzinskiy.backend.dto.ProjectResponse;
import vodzinskiy.backend.models.Project;
import vodzinskiy.backend.models.User;

import java.util.UUID;

public interface ProjectService {
   ProjectResponse createProject(User owner, String name);
   Project getProject(UUID id);
   ProjectResponse getProjectById(UUID projectId, UUID userId);
   ProjectResponse updateProject(ProjectResponse project);
   void deleteProject(UUID projectId, UUID userId);
   ProjectResponse joinProject(UUID projectId, User user);
   void leaveProject(UUID projectId, UUID userId);
}
```

UserService.java

```
package vodzinskiy.backend.services;

import jakarta.servlet.http.HttpSession;
import vodzinskiy.backend.dto.UserRequest;
import vodzinskiy.backend.dto.UserResponse;
import vodzinskiy.backend.models.User;

import java.util.UUID;

public interface UserService {
    UserResponse createUser(UserRequest request);
    UserResponse editUser(UUID id, UserRequest request);
    User getUser(UUID id);
    User getUserByEmail(String email);
    UUID getIdFromSession(HttpSession session);
    void deleteUser(UUID id);
}
```

AuthenticationServiceImpl.java

```
package vodzinskiy.backend.services.impl;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.context.SecurityContextRepository;
import org.springframework.stereotype.Service;
import vodzinskiy.backend.dto.LoginRequest;
import vodzinskiy.backend.dto.UserResponse;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.services.AuthenticationService;
import vodzinskiy.backend.services.UserService;

@Service
@RequiredArgsConstructor
public class AuthenticationServiceImpl implements AuthenticationService {

    private final AuthenticationManager authenticationManager;
    private final SecurityContextRepository securityContextRepository;
    private final UserService userService;


    @Override
```

```
    public UserResponse signin(LoginRequest loginRequest, HttpServletRequest request,
HttpServletResponse response) {
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(loginRequest.email(), loginRequest.password()));
        SecurityContext context = SecurityContextHolder.getContext();
        context.setAuthentication(authentication);
        SecurityContextHolder.setContext(context);
        securityContextRepository.saveContext(context, request, response);
        User user = userService.getUserByEmail(loginRequest.email());
        request.getSession().setAttribute("userID", user.getId());
        return new UserResponse(user.getId(), user.getUsername(), user.getEmail());
    }
}
```

OAuth2UserService.java

```
package vodzinskiy.backend.services.impl;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.oauth2.client.userinfo.DefaultOAuth2UserService;
import org.springframework.security.oauth2.client.userinfo.OAuth2UserRequest;
import org.springframework.security.oauth2.core.OAuth2AuthenticationException;
import org.springframework.security.oauth2.core.user.OAuth2User;
import org.springframework.stereotype.Service;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.repositories.UserRepository;

import java.util.Optional;

@Service
@RequiredArgsConstructor
@Slf4j
public class OAuth2UserService extends DefaultOAuth2UserService {

    private final UserRepository userRepository;

    @Override
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
OAuth2AuthenticationException {
        OAuth2User oAuth2User = super.loadUser(userRequest);
        processOAuth2User(oAuth2User);
        return oAuth2User;
    }

    private void processOAuth2User(OAuth2User oAuth2User) {
        String username = Optional.ofNullable(oAuth2User.getAttributes().get("name"))
                .map(Object::toString)
                .orElseGet(() -> oAuth2User.getAttributes().get("login").toString());
        String email = Optional.ofNullable(oAuth2User.getAttributes().get("email"))
```

```
                .map(Object::toString)
                .orElse(null);
        Optional<User> optionalUser = userRepository.findByProviderId(oAuth2User.getName());
        if (optionalUser.isEmpty()) {
            User user = new User(username, email, null, oAuth2User.getName());
            userRepository.save(user);
        }
    }
}
```

ProjectServiceImpl.java

```
package vodzinskiy.backend.services.impl;

import com.corundumstudio.socketio.SocketIOServer;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import vodzinskiy.backend.dto.ProjectResponse;
import vodzinskiy.backend.dto.Role;
import vodzinskiy.backend.exceptions.ForbiddenException;
import vodzinskiy.backend.exceptions.NotFoundException;
import vodzinskiy.backend.models.Project;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.repositories.ProjectRepository;
import vodzinskiy.backend.services.ProjectService;

import java.util.*;
import java.util.stream.Collectors;


@Service
@RequiredArgsConstructor
public class ProjectServiceImpl implements ProjectService {
    private final ProjectRepository projectRepository;
    private final SocketIOServer server;

    @Override
    public ProjectResponse createProject(User owner, String name) {
        Project project = new Project(name, owner);
        projectRepository.save(project);
        return new ProjectResponse(project.getId(), project.getName(), owner.getUsername(), new
HashSet<>(), Role.OWNER);
    }

    @Override
    public ProjectResponse joinProject(UUID projectId, User user) {
        Project project = getProject(projectId);
        UUID userId = user.getId();
        if (!(project.getMembers().contains(user) && project.getOwner().getId().equals(userId))) {
            project.addMember(user);
```

```
            projectRepository.save(project);
        }
        updateMembersList(project);
        return projectToProjectResponse(project, userId);
    }

    @Override
    public Project getProject(UUID id) {
        Optional<Project> project = projectRepository.findById(id);
        if (project.isEmpty()) {
            throw new NotFoundException(String.format("Project with id \"%s\" not found", id));
        }
        return project.get();
    }

    @Override
    public ProjectResponse getProjectById(UUID projectId, UUID userId) {
        Project project = getProject(projectId);
        return projectToProjectResponse(project, userId);
    }

    @Override
    public ProjectResponse updateProject(ProjectResponse project) {
        return null;
    }

    @Override
    public void deleteProject(UUID projectId, UUID userId) {
        if (!getProject(projectId).getOwner().getId().equals(userId)) {
            throw new ForbiddenException("Only the owner can delete a project");
        }
        server.getRoomOperations(projectId.toString()).sendEvent("projectUserListUpdated", new
ArrayList<>());
        projectRepository.deleteById(projectId);
    }

    @Override
    public void leaveProject(UUID projectId, UUID userId) {
        Project project = getProject(projectId);
        project.removeMember(userId);
        projectRepository.save(project);
        updateMembersList(project);
    }

    private void updateMembersList(Project project) {
        List<String> memberNames = project.getMembers().stream()
                .map(User::getUsername)
                .collect(Collectors.toCollection(ArrayList::new));
        memberNames.addFirst(project.getOwner().getUsername());
```

```
        server.getRoomOperations(project.getId().toString()).sendEvent("projectUserListUpdated",
memberNames);
    }

    private ProjectResponse projectToProjectResponse(Project project, UUID userId ) {
        Role role = Role.MEMBER;
        if (userId.equals(project.getOwner().getId())) {
            role = Role.OWNER;
        }
        Set<String> memberNames = project.getMembers().stream()
                .map(User::getUsername)
                .collect(Collectors.toSet());
        return new ProjectResponse(project.getId(),
                project.getName(),
                project.getOwner().getUsername(),
                memberNames,
                role
        );
    }
}
```

UserDetailsServiceImpl.java

```
package vodzinskiy.backend.services.impl;

import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.repositories.UserRepository;

import java.util.ArrayList;

@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException(String.format("User with email \"%s\"
not found", email)));

        return new org.springframework.security.core.userdetails.User(
                user.getEmail(),
                user.getPassword(),
```

```
            new ArrayList<>()
        );
    }
}
```

UserServiceImpl.java

```
package vodzinskiy.backend.services.impl;

import jakarta.servlet.http.HttpSession;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import vodzinskiy.backend.dto.UserRequest;
import vodzinskiy.backend.dto.UserResponse;
import vodzinskiy.backend.exceptions.AlreadyExistsException;
import vodzinskiy.backend.exceptions.ForbiddenException;
import vodzinskiy.backend.exceptions.NotFoundException;
import vodzinskiy.backend.mappers.UserMapper;
import vodzinskiy.backend.models.Project;
import vodzinskiy.backend.models.User;
import vodzinskiy.backend.repositories.ProjectRepository;
import vodzinskiy.backend.repositories.UserRepository;
import vodzinskiy.backend.services.ProjectService;
import vodzinskiy.backend.services.UserService;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final ProjectRepository projectRepository;
    private final ProjectService projectService;
    private final UserMapper userMapper;
    private final PasswordEncoder passwordEncoder;

    @Override
    public UserResponse createUser(UserRequest request) {
        checkEmailExists(request.email());
        User user = new User(request.username(), request.email(),
passwordEncoder.encode(request.password()), null);
        userRepository.save(user);
        return new UserResponse(user.getId(), user.getUsername(), user.getEmail());
    }

    @Override
    public User getUser(UUID id) {
```

```
        Optional<User> optionalUser = userRepository.findById(id);
        if (optionalUser.isEmpty()) {
            throw new NotFoundException(String.format("User with id \"%s\" not found", id));
        }
        return optionalUser.get();
    }

    @Override
    public User getUserByEmail(String email) {
        Optional<User> optionalUser = userRepository.findByEmail(email);
        if (optionalUser.isEmpty()) {
            throw new NotFoundException(String.format("User with email \"%s\" not found", email));
        }
        return optionalUser.get();
    }

    @Override
    public UserResponse editUser(UUID id, UserRequest request) {
        User user = getUser(id);
        if (!user.getEmail().equals(request.email())) {
            checkEmailExists(request.email());
        }
        userMapper.updateUserFromUserRequest(request, user);
        if (request.password() != null && !request.password().isEmpty()) {
            user.setPassword(passwordEncoder.encode(request.password()));
        }
        userRepository.save(user);
        return new UserResponse(user.getId(), user.getUsername(), user.getEmail());
    }

    public void checkEmailExists(String email) {
        if (userRepository.existsByEmail(email)) {
            throw new AlreadyExistsException(String.format("User with email \"%s\" already exists",
email));
        }
    }

    public UUID getIdFromSession(HttpSession session) {
        try {
            return UUID.fromString(session.getAttribute("userID").toString());
        } catch (Exception e) {
            throw new ForbiddenException("You must be authenticated for this action");
        }
    }

    @Override
    public void deleteUser(UUID id) {
        getUser(id);
        List<Project> ownedProjects = projectRepository.findByOwnerId(id);
```

```
        for (Project project : ownedProjects) {
            projectService.deleteProject(project.getId(), id);
        }
        List<Project> joinedProjects = projectRepository.findByMembersId(id);
        for (Project project : joinedProjects) {
            projectService.leaveProject(project.getId(), id);
        }
        userRepository.deleteById(id);
    }
}
```

SocketHandler.java

```
package vodzinskiy.backend.sockets;

import com.corundumstudio.socketio.SocketIOClient;
import com.corundumstudio.socketio.SocketIOServer;
import com.corundumstudio.socketio.annotation.OnConnect;
import com.corundumstudio.socketio.annotation.OnDisconnect;
import com.corundumstudio.socketio.annotation.OnEvent;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import vodzinskiy.backend.dto.ProjectObject;
import vodzinskiy.backend.models.File;

import java.util.*;

@Component
@RequiredArgsConstructor
@Slf4j
public class SocketHandler {
    private final SocketIOServer server;

    @OnConnect
    private void onConnect(SocketIOClient client) {
        String projectId = client.getHandshakeData().getSingleUrlParam("projectId");
        Collection<SocketIOClient> clients = server.getRoomOperations(projectId).getClients();
        Iterator<SocketIOClient> iterator = clients.iterator();
        if (clients.size() > 1) {
            SocketIOClient sourcedClient = iterator.next();
            if (sourcedClient.getSessionId().equals(client.getSessionId()) && iterator.hasNext()) {
                sourcedClient = iterator.next();
            }
            sourcedClient.sendEvent("requestFiles", client.getSessionId());
        }
        client.joinRoom(projectId);
    }

    @OnDisconnect
```

```
    private void onDisconnect(SocketIOClient client) {
        client.getAllRooms().stream().findFirst().ifPresent(client::leaveRoom);
    }

    @OnEvent("updateFile")
    public void onUpdateFile(SocketIOClient client, List<Map<String, Object>> change, UUID fileId) {
        String projectId = client.getHandshakeData().getSingleUrlParam("projectId");
        if (projectId != null) {
            server.getRoomOperations(projectId).getClients().stream()
                    .filter(c -> !c.getSessionId().equals(client.getSessionId()))
                    .forEach(c -> c.sendEvent("fileUpdates", change, fileId));
        }
    }

    @OnEvent("files")
    public void onFiles(SocketIOClient client, List<File> files, String recipientId) {
        server.getClient(UUID.fromString(recipientId)).sendEvent("files", files);
    }

    @OnEvent("addFile")
    public void onAddFile(SocketIOClient client, ProjectObject object) {
        String projectId = client.getHandshakeData().getSingleUrlParam("projectId");
        if (projectId != null) {
            server.getRoomOperations(projectId).getClients().stream()
                    .filter(c -> !c.getSessionId().equals(client.getSessionId()))
                    .forEach(c -> c.sendEvent("addFile", object));
        }
    }

    @OnEvent("renameFile")
    public void onRenameFile(SocketIOClient client, String path, String newName) {
        String projectId = client.getHandshakeData().getSingleUrlParam("projectId");
        if (projectId != null) {
            server.getRoomOperations(projectId).getClients().stream()
                    .filter(c -> !c.getSessionId().equals(client.getSessionId()))
                    .forEach(c -> c.sendEvent("renameFile", path, newName));
        }
    }

    @OnEvent("removeFile")
    public void onRemoveFile(SocketIOClient client, String path) {
        String projectId = client.getHandshakeData().getSingleUrlParam("projectId");
        if (projectId != null) {
            server.getRoomOperations(projectId).getClients().stream()
                    .filter(c -> !c.getSessionId().equals(client.getSessionId()))
                    .forEach(c -> c.sendEvent("removeFile", path));
        }
    }
}
```

main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

app-routing.module.ts

```
import {NgModule} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';
import {WorkspaceMainComponent} from "./modules/workspace/components/workspace-
main/workspace-main.component";
import {ProfileComponent} from "./modules/profile/profile.component";

const routes: Routes = [{
  path: '',
  component: WorkspaceMainComponent
  },
  {
    path: 'p/:id',
    component: WorkspaceMainComponent

  },
  {
    path: 'profile',
    component: ProfileComponent
  },
  {
    path: '**',
    redirectTo: ''
  }]

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
  providers: []
})
export class AppRoutingModule {
}
```

app.component.ts

```
import {Component, OnInit} from '@angular/core';
import {NavigationStart, Router} from "@angular/router";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
```

```
export class AppComponent implements OnInit {
  title = 'frontend';
  navBar: boolean = true;
  private hiddenRoutes = ['/login', '/signup', '/profile'];

  constructor(private router: Router) {}

  ngOnInit() {
    this.router.events.subscribe(event => {
      if (event instanceof NavigationStart) {
        this.navBar = !this.hiddenRoutes.includes(event.url);
      }
    });
  }
}
```

app.component.html

```
<app-navbar *ngIf="navBar"></app-navbar>
<router-outlet></router-outlet>
```

navbar.component.ts

```
import {Component, OnInit} from '@angular/core';
import {AuthService} from "../../../modules/auth/services/auth.service";

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrl: './navbar.component.scss'
})
export class NavbarComponent implements OnInit{
  isLoading = true;

  constructor(public authService: AuthService) {}

  ngOnInit(): void {
    this.authenticated()
  }

  authenticated() {
    this.authService.checkAuthStatus().subscribe({
        next: ()=> this.isLoading = false,
        error: ()=> this.isLoading = false
      }
    );
  }
}
```

navbar.component.html

```
<mat-toolbar class="d-flex align-content-center" [ngClass]="{'justify-content-between':
(authService.isAuthenticated$ | async), 'justify-content-end': !(authService.isAuthenticated$ | async)}">
  <div *ngIf="(authService.isAuthenticated$ | async)">
    <app-file-menu></app-file-menu>
    <app-project-menu></app-project-menu>
  </div>
  <div *ngIf="!isLoading">
    <div *ngIf="(authService.isAuthenticated$ | async); else notAuthenticated" class="d-flex align-items-
center">
      <app-project-details></app-project-details>
      <app-user-menu></app-user-menu>
    </div>
    <ng-template #notAuthenticated>
      <button routerLink="/signup" class="mx-2" mat-stroked-button>Зареєструватись</button>
      <button routerLink="/login" class="mx-2" mat-raised-button>Вхід</button>
    </ng-template>
  </div>
</mat-toolbar>
```

file-menu.component.ts

```
import {Component, ElementRef, OnInit, ViewChild} from '@angular/core';
import {FileService} from "../../../../modules/workspace/services/file.service";
import {ProjectService} from "../../../services/project.service";

@Component({
  selector: 'app-file-menu',
  templateUrl: './file-menu.component.html',
  styleUrl: './file-menu.component.scss'
})
export class FileMenuComponent implements OnInit {
  @ViewChild('fileInput') fileInput?: ElementRef;
  hideMenu: boolean = true;
  tree: any[] = [];

  constructor(public fileService: FileService, protected projectService: ProjectService) {}

  ngOnInit(): void {
    this.projectService.project$.subscribe({
        next: project => this.hideMenu = project === null})

    this.fileService.filesObservable$.subscribe({
      next: f => this.tree = f
    })
  }

  newFile() {
    this.fileService.createFile('file', '', this.tree, '')
  }
```

```
  newDirectory() {
    this.fileService.createFile('directory', '', this.tree, '')
  }

  openFileDialog() {
    if (this.fileInput) {
      this.fileInput.nativeElement.click();
    }
  }

  onFileSelected(event: any) {
    this.projectService.openProject(event.target.files)
  }

  saveFiles() {
    this.fileService.saveFiles()
  }
}
```

file-menu.component.html

```
<button mat-button [matTooltip]="hideMenu ? 'Спочатку створіть або доєднайтесь до проєкту' : ""
        [matMenuTriggerFor]="hideMenu ? null : file">Файл</button>
<mat-menu #file="matMenu" class="mt-1">
  <button mat-menu-item (click)="newFile()">Новий Файл</button>
  <button mat-menu-item (click)="newDirectory()">Нова Директорія</button>
  <button mat-menu-item (click)="openFileDialog()">Імпортувати Файли
    <input type="file" #fileInput (change)="onFileSelected($event)" style="display:none;" multiple/>
  </button>
  <button mat-menu-item (click)="saveFiles()">Експортувати проєкт</button>
</mat-menu>
```

project-details.component.ts

```
import {Component, OnInit} from '@angular/core';
import {ProjectService} from "../../../services/project.service";
import {ProjectSocketService} from "../../../services/project-socket.service";

@Component({
  selector: 'app-project-details',
  templateUrl: './project-details.component.html',
  styleUrl: './project-details.component.scss'
})
export class ProjectDetailsComponent implements OnInit {
  members: string[] = []
  projectName: string = "";
  protected id: string = '';

  constructor(protected projectService: ProjectService, private socket: ProjectSocketService) {}
```

```
  ngOnInit(): void {
    this.projectService.project$.subscribe({
      next: project => {
        this.id = project?.id ?? '';
        this.projectName = project?.name ?? '';
        this.members = project?.members ? [project.owner, ...project.members] : [];
      }
    }
    )
    this.socket.memberListUpdate().subscribe({
      next: m => {
        this.members = m;
        m.length === 0 && this.projectService.leaveProject(this.id);
      }
    })
  }

  copyId() {
    navigator.clipboard.writeText(this.id).then()
  }
}
```

project-details.component.html

```
<button *ngIf="id != '' mat-stroked-button class="mx-4" [matMenuTriggerFor]="projectMember">{{
projectName }}</button>
<mat-menu #projectMember="matMenu" class="mt-1">
  <div class="d-flex justify-content-center m-1">
    <button mat-stroked-button (click)="copyId()">Копіювати id</button>
  </div>
  <span mat-menu-item *ngFor="let name of members" [textContent]="name"></span>
</mat-menu>
```

project-menu.component.ts

```
import {Component, OnInit} from '@angular/core';
import {ProjectService} from "../../../services/project.service";
import {DialogComponent} from "../../../../shared/components/dialog/dialog.component";
import {MatDialog} from "@angular/material/dialog";
import {DialogData} from "../../../models/dialog-data.dto";
import {Role} from "../../../models/project.dto";
import {Router} from "@angular/router";
import {SocketService} from "../../../services/socket.service";
import {FileService} from "../../../../modules/workspace/services/file.service";

@Component({
  selector: 'app-project-menu',
  templateUrl: './project-menu.component.html',
  styleUrl: './project-menu.component.scss'
})
export class ProjectMenuComponent implements OnInit {
```

```
  protected owner: boolean = false;
  private id: string = '';

  constructor(private projectService: ProjectService, public dialog: MatDialog, private router: Router,
private socket: SocketService, public fileService: FileService) {}

  ngOnInit(): void {
    this.projectService.project$.subscribe({
      next: project => {
        if (project !== null) {
          this.owner = project.role === Role.OWNER
          this.id = project.id
          this.socket.connectToSocket(project.id)
        } else {
          this.id = ''
        }
      }
    }
    )
  }

  createProject() {
    const data: DialogData = {buttonTitle: "Створити", placeholder: "Назва", title: "Новий Проєкт"}
    const dialogRef = this.dialog.open(DialogComponent, {data});
    dialogRef.afterClosed().subscribe(result => {
      if (result) {
        this.projectService.createProject(result).subscribe({
          next: (project) => {
            void this.router.navigate(['/p', project.body?.id])
          }
        })
      }
    });
  }

  joinProject() {
    const data: DialogData = {buttonTitle: "Приєднатись", placeholder: "ID", title: "Приєднатись до
проєкту"}
    const dialogRef = this.dialog.open(DialogComponent, {data});
    dialogRef.afterClosed().subscribe(result => {
      if (result) {
        this.projectService.joinProject(result).subscribe({
          next: (project) => {
            void this.router.navigate(['/p', project.body?.id])
          }
        })
      }
    });
  }
```

```
  leaveProject() {
    this.projectService.leaveProject(this.id)
  }
  deleteProject() {
    if (this.owner) {
      this.projectService.deleteProject(this.id).subscribe({
        next: () => {
          this.socket.disconnectFromSocket()
          this.fileService.clearAllData()
          void this.router.navigate(['/'])
          this.owner = false
        }
      }
      )
    }
  }
}
```

project-menu.component.html

```
<button mat-button [matMenuTriggerFor]="project">Поєкт</button>
<mat-menu #project="matMenu" class="mt-1">
  <button mat-menu-item (click)="createProject()">Створити</button>
  <button mat-menu-item (click)="joinProject()">Приєднатись</button>
  <button *ngIf="'!owner" mat-menu-item (click)="leaveProject()">Від'єднатись</button>
  <button *ngIf="owner" mat-menu-item (click)="deleteProject()">Видалити</button>
</mat-menu>
```

user-menu.component.ts

```
import {Component, OnInit} from '@angular/core';
import {AuthService} from "../../../../modules/auth/services/auth.service";
import {UserService} from "../../../services/user.service";

@Component({
  selector: 'app-user-menu',
  templateUrl: './user-menu.component.html',
  styleUrl: './user-menu.component.scss'
})
export class UserMenuComponent implements OnInit {
  profileLetter: string = ''
  username: string = ''

  constructor(public authService: AuthService, public userService: UserService) {}

  ngOnInit(): void {
    this.userService.getUser().subscribe({
      next: user => {
        this.username = user.body?.username ?? ''
        this.profileLetter = this.username.length > 0 ? this.username.charAt(0).toUpperCase() : ''
      }
```

```
    })
  }

  signout() {
    this.authService.signout().subscribe()
    this.authService.setAuthStatus(false)
  }
}
```

user-menu.component.html

```
<button mat-mini-fab style="background: #362f36" [matMenuTriggerFor]="profile">{{ profileLetter
}}</button>
<mat-menu #profile="matMenu" class="mt-1 d-flex">
  <button mat-menu-item routerLink="/profile">Профіль</button>
  <button mat-menu-item (click)="signout()">Вийти</button>
</mat-menu>
```

dialog-data.dto.ts

```
export interface DialogData {
  title: string
  name?: string
  placeholder: string
  buttonTitle: string
}
```

file.model.ts

```
import {UUID} from "crypto";

export class FileModel {
  id: UUID;
  name: string;
  data: string;
  content = undefined
  fPath?: string

  constructor(id: UUID | undefined, name: string, data: string, fPath?: string) {
    this.id = id ? id : crypto.randomUUID();
    this.name = name;
    this.data = data
    this.fPath = fPath
  }
}
```

project.dto.ts

```
export interface Project {
  id: string;
  name: string;
  owner: string;
  members: string[];
```

```
  role: Role;
}
export enum Role {
  OWNER = 'OWNER',
}
```

project-object.dto.ts

```
import {UUID} from "crypto";

export interface ProjectObject {
  id?: UUID;
  type: 'directory' | 'file';
  path: string;
  name: string;
  data?: string;
  fPath: string;
}
```

user.dto.ts

```
export interface User {
  username?: string;
  email?: string;
  password?: string;
  id?: number;
}
```

file-socket.service.ts

```
import {Injectable} from '@angular/core';
import {Observable} from 'rxjs';
import {SocketService} from "./socket.service";
import {ProjectObject} from "../models/project-object.dto";

@Injectable({
  providedIn: 'root'
})
export class FileSocketService {

  constructor(public socket: SocketService) {}

  updateFile(change: any, fileId: string): void {
    this.socket.getSocket().emit('updateFile', change, fileId);
  }

  fileUpdated(fileId?: string): Observable<any> {
    return new Observable(observer => {
      this.socket.getSocket().on('fileUpdates', (operations: any, id: string) => {
        if (fileId === id) {
          observer.next(operations);
        }
        if (!fileId) {
```

```
        observer.next({operations, id});
      }
    });
  });
}

addFile(obj: ProjectObject) {
  this.socket.getSocket().emit('addFile', obj)
}

renameFile(path: string, newName: string) {
  this.socket.getSocket().emit('renameFile', path, newName);
}

removeFile(path: string) {
  this.socket.getSocket().emit('removeFile', path);
}

onAddFile(): Observable<ProjectObject>{
  return new Observable(observer => {
    this.socket.getSocket().on('addFile', (obj: ProjectObject) => {
      console.log("==" + obj.data)
      observer.next(obj);
    });
  });
}

onRenameFile(): Observable<{path: string, newName: string}> {
  return new Observable(observer => {
    this.socket.getSocket().on('renameFile', (path: string, newName: string) => {
      observer.next({path, newName});
    });
  });
}

onRemoveFile(): Observable<string> {
  return new Observable(observer => {
    this.socket.getSocket().on('removeFile', (path: string) => {
      observer.next(path);
    });
  });
}
}
```

project.service.ts

```
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {env} from "../../../environments/environment";
import {BehaviorSubject} from "rxjs";
```

```
import {Project} from "../models/project.dto";
import {Router} from "@angular/router";
import {FileService} from "../../modules/workspace/services/file.service";
import {SocketService} from "./socket.service";

@Injectable({
  providedIn: 'root'
})
export class ProjectService {
  private apiUrl = env.API_URL;
  public projectSubject = new BehaviorSubject<Project | null>(null);
  public project$ = this.projectSubject.asObservable();

  constructor(private http: HttpClient, private router: Router,
              private fileService: FileService, private socket: SocketService) {}

  public joinProject(projectId: string) {
    if (this.projectSubject.value !== null) {
      this.leaveProject(this.projectSubject.value?.id)
    }
    return this.http.post<Project>(`${this.apiUrl}project/join/${projectId}`, {},
      {observe: 'response', withCredentials: true})
  }

  public createProject(name: string) {
    if (this.projectSubject.value !== null) {
      this.leaveProject(this.projectSubject.value?.id)
    }
    return this.http.post<Project>(`${this.apiUrl}project`, name,
      {observe: 'response', withCredentials: true})
  }

  public leaveProject(projectId: string) {
    this.socket.disconnectFromSocket()
    this.fileService.clearAllData()
    void this.router.navigate(['/'])
    this.projectSubject.next(null);
    this.http.post(`${this.apiUrl}project/leave/${projectId}`, {}, {observe: 'response', withCredentials:
true}).subscribe()
  }

  openProject(files: FileList) {
    if (files) {
      Array.from(files).forEach((file) => {
        const reader = new FileReader();
        reader.onload = (e: any) => {
          this.fileService.addFile('file', '', this.fileService.filesSubject.value, "", file.name, true, undefined,
e.target.result)
        };
```

```
      reader.readAsText(<Blob>file);
    });
  }
 }

 public deleteProject(projectId: string) {
  this.socket.disconnectFromSocket()
  this.fileService.clearAllData()
  void this.router.navigate(['/'])
  this.projectSubject.next(null);
  return this.http.delete(`${this.apiUrl}project/${projectId}`, {observe: 'response', withCredentials:
true})
 }

 public getProject(projectId: string) {
  this.http.get<Project>(`${this.apiUrl}project/${projectId}`, {observe: 'response', withCredentials:
true})
    .subscribe({
      next: (response) => this.projectSubject.next(response.body),
      error: (e) => console.error(e)
     }
    )
 }
}
```

project-socket.service.ts

```
import {Injectable} from '@angular/core';
import {Observable} from 'rxjs';
import {FileService} from "../../modules/workspace/services/file.service";
import {FileModel} from "../models/file.model";
import {SocketService} from "./socket.service";

@Injectable({
 providedIn: 'root'
})
export class ProjectSocketService {
 constructor(public fileService: FileService, public socket: SocketService) {}

 public memberListUpdate() {
  return new Observable<string[]>(observer => {
   this.socket.getSocket().on('projectUserListUpdated', (members: string[]) => {
    observer.next(members);
   });
  });
 }

 public onRequestFiles() {
  this.socket.getSocket().on('requestFiles', (recipientId: string) => {
   this.socket.getSocket().emit('files', this.fileService.filesSubject.value, recipientId);
```

```
    })
  }

  public onFilesReceived() {
    this.socket.getSocket().on('files', (files: FileModel[]) => {
      this.fileService.filesSubject.next(files);
    })
  }
}
```

socket.service.ts

```
import {Injectable} from '@angular/core';
import io from 'socket.io-client';
import {env} from "../../../environments/environment";

@Injectable({
  providedIn: 'root'
})
export class SocketService {
  private readonly socket;

  constructor() {
    this.socket = io(env.SOCKET_URL, {autoConnect: false, transports: ["websocket"]});
  }

  connectToSocket(projectId: string | undefined) {
    if (this.socket) {
      this.socket.io.opts.query = {projectId: projectId};
      this.socket.connect();
    }
  }

  disconnectFromSocket() {
    if (this.socket) {
      this.socket.disconnect();
    }
  }

  public getSocket() {
    return this.socket;
  }
}
```

user.service.ts

```
import { Injectable } from '@angular/core';
import {env} from "../../../environments/environment";
import {User} from "../models/user.dto";
import {HttpClient} from "@angular/common/http";
```

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  private apiUrl = env.API_URL;

  constructor(private http: HttpClient) {}

  public getUser() {
    return this.http.get<User>(`${this.apiUrl}user`, { observe: 'response', withCredentials: true })
  }

  public updateUser(user: User) {
    return this.http.patch<User>(`${this.apiUrl}user`, user,{ observe: 'response', withCredentials: true })
  }

  public deleteUser() {
    return this.http.delete(`${this.apiUrl}user`, { observe: 'response', withCredentials: true })
  }
}
```

login.component.ts

```
import {Component} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";
import {AuthService} from "../../services/auth.service";
import {User} from "../../../../core/models/user.dto";
import {Router} from "@angular/router";

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrl: './login.component.scss'
})
export class LoginComponent {
  signinForm = new FormGroup({
    email: new FormControl('', [Validators.required, Validators.email]),
    password: new FormControl('', Validators.required)
  });

  constructor(private authService: AuthService, private router: Router) {
  }

  getControl(name: string): FormControl {
    return this.signinForm.get(name) as FormControl;
  }

  onSubmit() {
    const user: User = {
      email: this.signinForm.get('email')?.value || '',
      password: this.signinForm.get('password')?.value || ''
```

```
    };
    this.authService.signin(user).subscribe({
      next: () => {
        void this.router.navigate(['/'])
      },
      error: err => {
        if (err.status === 401) {
          this.signinForm.controls.email.setErrors({'incorrect': true})
          this.signinForm.controls.password.setErrors({'incorrect': true})
        }
      }
    });
  }
}
```

login.component.html

```
<div class="auth-window">
  <mat-card class="d-flex flex-column align-items-center">
    <mat-card-header class="mb-3">
      <mat-card-title>Вхід</mat-card-title>
    </mat-card-header>
    <mat-card-content>
      <form class="d-flex flex-column align-items-center" [formGroup]="signinForm">
        <app-custom-input [control]="getControl('email')"
                          placeholder="Електронна пошта"
                          requiredErrorMessage="Введіть електронну адресу">
        </app-custom-input>
        <app-custom-input [control]="getControl('password')"
                          placeholder="Пароль"
                          requiredErrorMessage="Введіть пароль"
                          isPassword="true">
        </app-custom-input>
      </form>
    </mat-card-content>
    <mat-card-actions class="d-flex justify-content-evenly w-100">
      <button routerLink="/signup" mat-stroked-button>Зареєструватися</button>
      <button (click)="onSubmit()" mat-raised-button>Увійти</button>
    </mat-card-actions>
    <app-social-login></app-social-login>
  </mat-card>
</div>
```

signup.component.ts

```
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import {AuthService} from "../../services/auth.service";
import {User} from "../../../../core/models/user.dto";
import {Router} from "@angular/router";

@Component({
```

```
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrl: './signup.component.scss'
})
export class SignupComponent {
  signupForm = new FormGroup({
    name: new FormControl('', Validators.required),
    email: new FormControl('', [Validators.required, Validators.email]),
    password: new FormControl('', Validators.required)
  });

  constructor(private router: Router, private authService: AuthService) {}

  getControl(name: string): FormControl {
    return this.signupForm.get(name) as FormControl;
  }

  onSubmit() {
    if (this.signupForm.valid) {
      const user: User = {
        username: this.signupForm.get('name')?.value || '',
        email: this.signupForm.get('email')?.value || '',
        password: this.signupForm.get('password')?.value || ''
      };
      this.authService.signup(user).subscribe({
        next: () => {
          this.authService.signin(user).subscribe({
            next: () => {
              void this.router.navigate(['/'])
            }
          });
        },
        error: () => this.signupForm.controls.email.setErrors({'exists': true})
        });
    }
  }
}
```

signup.component.html

```
<div class="auth-window">
  <mat-card class="d-flex flex-column align-items-center">
    <mat-card-header class="mb-3">
      <mat-card-title>Реєстрація</mat-card-title>
    </mat-card-header>
    <mat-card-content>
      <form class="d-flex flex-column align-items-center" [formGroup]="signupForm">
        <app-custom-input [control]="getControl('name')"
          formControlName="name"
          placeholder="Ім'я"
```

```
        requiredErrorMessage="Введіть своє ім'я">
       </app-custom-input>
       <app-custom-input [control]="getControl('email')"
        placeholder="Електронна пошта"
        requiredErrorMessage="Введіть електронну адресу">
       </app-custom-input>
       <app-custom-input [control]="getControl('password')"
        placeholder="Пароль"
        requiredErrorMessage="Введіть пароль"
        isPassword=true>
       </app-custom-input>
     </form>
   </mat-card-content>
   <mat-card-actions class="d-flex justify-content-evenly w-100">
    <button (click)="onSubmit()" mat-raised-button>Зареєструватися</button>
    <button routerLink="/login" mat-stroked-button>Увійти</button>
   </mat-card-actions>
   <app-social-login></app-social-login>
  </mat-card>
</div>
```

auth.service.ts

```
import {Injectable} from '@angular/core';
import {BehaviorSubject, Observable, tap} from "rxjs";
import {env} from "../../../../environments/environment";
import {HttpClient} from "@angular/common/http";
import {User} from "../../../core/models/user.dto";
import {ProjectService} from "../../../core/services/project.service";

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private apiUrl = env.API_URL;
  private isAuthenticatedSubject = new BehaviorSubject<boolean>(false);
  public isAuthenticated$: Observable<boolean> = this.isAuthenticatedSubject.asObservable();

  constructor(private http: HttpClient, private projectService: ProjectService) {}

  signup(userBody: User): Observable<any> {
    return this.http.post(`${this.apiUrl}auth/signup`, userBody, {observe: 'response', withCredentials:
true});
  }

  signin(loginBody: User) {
    return this.http.post<User>(`${this.apiUrl}auth/signin`, loginBody, {observe: 'response',
withCredentials: true})
  }
```

```
  signout(): Observable<any> {
    let id = this.projectService.projectSubject.value?.id
    if (id) {
      this.projectService.leaveProject(id)
    }
    return this.http.post(`${this.apiUrl}auth/signout`,{},{observe: 'response', withCredentials: true});
  }

  checkAuthStatus() {
    return this.http.get(`${this.apiUrl}auth/session`, {observe: 'response', withCredentials: true}).pipe(
      tap({
          next: () => this.isAuthenticatedSubject.next(true),
          error: () => this.isAuthenticatedSubject.next(false)
        }
      )
    );
  }

  setAuthStatus(isAuthenticated: boolean) {
    this.isAuthenticatedSubject.next(isAuthenticated);
  }
}
```

auth.routing.ts

```
import {NgModule} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';
import {LoginComponent} from "./components/login/login.component";
import {SignupComponent} from "./components/signup/signup.component";

const routes: Routes = [{
  path: 'login',
  component: LoginComponent
},
  {
    path: 'signup',
    component: SignupComponent
  }]

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AuthRoutingModule {}
```

profile.component.ts

```
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";
import {UserService} from "../../core/services/user.service";
import {Router} from "@angular/router";
import {User} from "../../core/models/user.dto";
```

```
import {AuthService} from "../auth/services/auth.service";
import {MatDialog} from "@angular/material/dialog";
import {DeleteDialogComponent} from "./delete-dialog.component";
import {ProjectService} from "../../core/services/project.service";

@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrl: './profile.component.scss'
})
export class ProfileComponent implements OnInit {
  profileForm = new FormGroup({
    username: new FormControl(''),
    email: new FormControl('', Validators.email),
    password: new FormControl('')
  });

  constructor(public userService: UserService, private router: Router, private authService: AuthService,
private dialog: MatDialog, private projectService: ProjectService) {
  }

  ngOnInit(): void {
    this.userService.getUser().subscribe({
      next: user => {
        this.profileForm.patchValue({
          username: user.body?.username ?? '',
          email: user.body?.email ?? ''
        });
      }
    });
  }

  delete() {
    const dialogRef = this.dialog.open(DeleteDialogComponent);
    dialogRef.afterClosed().subscribe(result => {
      if (result) {
        this.userService.deleteUser().subscribe();
        let id = this.projectService.projectSubject.value?.id
        if (id) {
          this.projectService.leaveProject(id)
        }
        this.authService.setAuthStatus(false)
        void this.router.navigate(['/']);
      }
    });
  }

  save() {
    const updatedUser: User = {
```

```
    ...(this.profileForm.value.username?.trim() && {username: this.profileForm.value.username.trim()}),
    ...(this.profileForm.value.email?.trim() && {email: this.profileForm.value.email.trim()}),
    ...(this.profileForm.value.password?.trim() && {password: this.profileForm.value.password.trim()})
   };
   if (Object.values(updatedUser).some(value => value !== undefined)) {
    this.userService.updateUser(updatedUser).subscribe({
     next: () => void this.router.navigate(['/']),
     error: () => {
      this.profileForm.controls.email.setErrors({'exists': true});
     }
    });
   }
  }
}
```

profile.component.html

```
<div class="auth-window">
 <mat-card class="d-flex flex-column align-items-center">
  <mat-card-header class="mb-3">
   <mat-card-title>Профіль</mat-card-title>
  </mat-card-header>
  <form [formGroup]="profileForm" class="d-flex flex-column align-items-center">
   <mat-form-field class="profile-input-field">
    <input matInput placeholder="Ім'я" formControlName="username">
   </mat-form-field>
   <mat-form-field class="profile-input-field">
    <input matInput placeholder="Електронна пошта" formControlName="email">
    <mat-error *ngIf="profileForm.controls.email.errors?.['exists']">
     Користувач з такою адресою ел. пошти вже існує
    </mat-error>
    <mat-error *ngIf="profileForm.controls.email.errors?.['email']">
     Некоректна адреса електронної пошти
    </mat-error>
   </mat-form-field>
   <mat-form-field class="profile-input-field">
    <input matInput placeholder="Новий пароль" formControlName="password">
   </mat-form-field>
   <mat-card-actions class="d-flex justify-content-evenly w-100">
    <button routerLink="" mat-stroked-button>Скасувати</button>
    <button (click)="delete()" color="warn" mat-raised-button>Видалити</button>
    <button (click)="save()" mat-raised-button>Зберегти</button>
   </mat-card-actions>
  </form>
 </mat-card>
</div>
```

delete-dialog.component.ts

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-confirm-delete-dialog',
  template: `
    <h2 mat-dialog-title>Підтвердження видалення</h2>
    <div mat-dialog-content>
      <p>Ви впевнені, що хочете видалити свій акаунт?</p>
    </div>
    <div mat-dialog-actions>
      <button mat-button [mat-dialog-close]="false">Скасувати</button>
      <button mat-button [mat-dialog-close]="true" color="warn">Видалити</button>
    </div>
  `
})
export class DeleteDialogComponent {}
```

file.service.ts

```
import {Injectable} from '@angular/core';
import {FileModel} from "../../../core/models/file.model";
import {BehaviorSubject, Observable} from "rxjs";
import {DialogData} from "../../../core/models/dialog-data.dto";
import {DialogComponent} from "../../../shared/components/dialog/dialog.component";
import {MonacoTreeElement} from "ngx-monaco-tree";
import {MatDialog} from "@angular/material/dialog";
import {FileSocketService} from "../../../core/services/file-socket.service";
import {UUID} from "crypto";
import {ProjectObject} from "../../../core/models/project-object.dto";
import JSZip from 'jszip';
import {saveAs} from 'file-saver';


@Injectable({
  providedIn: 'root'
})
export class FileService {
  public files: any[] = []
  public openFiles: FileModel[] = []

  private tabIndexSubject = new BehaviorSubject<number | null>(null);
  public tabIndexObservable$ = this.tabIndexSubject.asObservable();

  filesSubject = new BehaviorSubject<FileModel[]>(this.files);
  filesObservable$: Observable<FileModel[]> = this.filesSubject.asObservable();

  openFilesSubject = new BehaviorSubject<FileModel[]>(this.openFiles);
  openFilesObservable$: Observable<FileModel[]> = this.openFilesSubject.asObservable();

  constructor(public socket: FileSocketService, public dialog: MatDialog) {
    this.socket.fileUpdated().subscribe(({operations, id}) => {
      if (!this.openFilesSubject.value.some(file => file.id === id)) {
```

```
        this.applyChanges(id, operations);
      }
    });
  }

  createFile(type: 'directory' | 'file', path: string, localTree: MonacoTreeElement[], fullPath: string) {
    const title = type === 'directory' ? "Нова директорія" : "Новий файл";
    const data: DialogData = {buttonTitle: "Створити", placeholder: "Назва", title: title}
    const dialogRef = this.dialog.open(DialogComponent, {data});
    dialogRef.afterClosed().subscribe(name => {
      if (name) {
        this.addFile(type, path, localTree, fullPath, name, true)
      }
    });
  }

  public addFile(type: 'directory' | 'file', path: string, localTree: MonacoTreeElement[], fullPath: string,
name: string, author: boolean = false, id: UUID | undefined = undefined, data: string = '') {
    const spited = path.split('/');
    if (spited.length === 1) {
      if (path === '') {
        const newFullPath = `${fullPath}/${name}`;
        localTree.push(this.file(name, type, newFullPath, id, author, data));
      } else {
        const file = localTree.find((el) => el.name == path);
        if (!file) return;
        else if (file.content === undefined) {
          const newFullPath = `${fullPath}/${name}`;
          localTree.push(this.file(name, type, newFullPath, id, author, data));
        } else {
          const newFullPath = `${fullPath}/${path}/${name}`;
          file.content.push(this.file(name, type, newFullPath, id, author, data));
        }
      }
    } else {
      const file = localTree.find((el) => el.name == spited[0]);
      if (!file || !file.content) return;
      const newFullPath = `${fullPath}/${spited[0]}`;
      this.addFile(type, spited.slice(1).join('/'), file?.content, newFullPath, name, author, id);
    }
  }

  private file(name: string, type: 'directory' | 'file', fPath: string, id: UUID | undefined, author: boolean,
data: string = "") {
    const path = fPath.replace(/\/[^/]*$/, '').slice(1);
    if (type === 'file') {
      let file = new FileModel(id, name, data, fPath.slice(1))
      if (author) {
        const obj: ProjectObject = {data: data, fPath: "", id: file.id, name: name, path: path, type: type};
```

```
        this.socket.addFile(obj)
      }
      this.openFile(file)
      return file
    } else {
      if (author) {
        const obj: ProjectObject = {fPath: "", name: name, path: path, type: type};
        this.socket.addFile(obj)
      }
      return {name: name, content: [], type: type}
    }
  }

  updateFile(path: string, name: string) {
    const file = this.findFileRecursive(this.filesSubject.value, path);
    if (file !== null) {
      file.name = name;
      if (file.fPath) {
        file.fPath = file.fPath.replace(/[^/]*$/, name);
      }
      this.filesSubject.next(this.filesSubject.value);
    }
  }

  applyChanges(id: UUID, changes: any[]): void {
    const file = this.findFileRecursive(this.filesSubject.value, "", id);
    changes.sort((a, b) => a.rangeOffset - b.rangeOffset);
    let offset = 0;
    for (const change of changes) {
      const startIndex = this.getIndexFromPosition(file.data, change.range.startLineNumber,
change.range.startColumn) + offset;
      const endIndex = this.getIndexFromPosition(file.data, change.range.endLineNumber,
change.range.endColumn) + offset;
      const removedText = file.data.slice(startIndex, endIndex);
      file.data = file.data.slice(0, startIndex) + change.text + file.data.slice(endIndex);
      offset += change.text.length - removedText.length;
    }
  }

  private getIndexFromPosition(fileData: string, lineNumber: number, column: number): number {
    const lines = fileData.split('\n');
    let index = 0;
    for (let i = 0; i < lineNumber - 1; i++) {
      index += lines[i].length + 1;
    }
    index += column - 1;
    return index;
  }
```

```
  openFile(file: FileModel) {
    if (file) {
      this.openFiles.push(file);
      this.openFilesSubject.next(this.openFiles);
    }
  }

  closeFile(index: number) {
    if (index !== -1) {
      this.openFiles.splice(index, 1);
      this.openFilesSubject.next(this.openFiles);
    }
  }

  public switchToTab(name: string) {
    const tabIndex = this.openFiles.findIndex(file => file.fPath === name);
    if (tabIndex !== -1) {
      this.tabIndexSubject.next(tabIndex);
    } else {
      const fileToOpen = this.findFileRecursive(this.filesSubject.value, name);
      if (fileToOpen) {
        const fileModel = new FileModel(fileToOpen.id, fileToOpen.name, fileToOpen.data,
fileToOpen.fPath);
        this.openFile(fileModel);
      }
    }
  }

  private findFileRecursive(items: any[], name: string, id?: UUID): any {
    for (const item of items) {
      if (!item.content && (item.fPath === name || item.id === id)) {
        return item;
      }
      if (item.content) {
        const foundFile = this.findFileRecursive(item.content, name);
        if (foundFile) {
          return foundFile;
        }
      }
    }
    return null;
  }

  public saveFiles() {
    const zip = new JSZip;
    const processFiles = (items: any[], currentPath: string) => {
      for (const item of items) {
        if (!item.content) {
          zip.file(item.fPath, item.data);
```

```
      } else {
        processFiles(item.content, `${currentPath}/${item.name}`);
      }
     }
    };
    processFiles(this.filesSubject.value, '');
    zip.generateAsync({type: 'blob'}).then(function (content) {
     saveAs(content, 'files.zip');
    });
  }

  public clearAllData() {
    this.files = [];
    this.openFiles = [];
    this.tabIndexSubject.next(null);
    this.filesSubject.next([]);
    this.openFilesSubject.next([]);
  }
}
```

resize.service.ts

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ResizeService {
  private leftWidthSource = new BehaviorSubject<number>(Math.ceil(window.innerWidth * 0.2));
  currentLeftWidth = this.leftWidthSource.asObservable();

  constructor() { }

  updateLeftWidth(value: number) {
    this.leftWidthSource.next(value);
  }
}
```

document-content.component.ts

```
import {Component, HostListener, Input, OnInit} from '@angular/core';
import {ResizeService} from "../../services/resize.service";
import {FileModel} from "../../../../core/models/file.model";
import {FileSocketService} from "../../../../core/services/file-socket.service";
import { languageExtensions } from '../../../../../assets/language-extensions';

@Component({
  selector: 'app-document-content',
  templateUrl: './document-content.component.html',
  styleUrl: './document-content.component.scss'
```

```
})
export class DocumentContentComponent implements OnInit {
  @Input() file!: FileModel
  width: number = 0;
  isRemoteChange = false;
  editor: any;
  editorOptions = {theme: 'vs-dark', language:"java", automaticLayout: true};
  constructor(private socket: FileSocketService, protected resizeService: ResizeService) {}

  @HostListener('window:resize', ['$event'])
  onResize() {
    this.resize()
  }

  ngOnInit(): void {
    this.socket.fileUpdated(this.file.id).subscribe((operations: any) => {
      if (this.editor && this.editor.getModel()) {
        this.isRemoteChange = true;
        this.editor.getModel().applyEdits(operations);
      }
    });
    this.resize();
  }

  getLanguageFromExtension(fileName: string): string {
    const fileExtension = fileName.split('.').pop()?.toLowerCase() || '';
    return languageExtensions[fileExtension] || 'plaintext';
  }

  resize() {
    this.resizeService.currentLeftWidth.subscribe(value => {
      this.width = window.innerWidth - value;
    });
  }

  onEditorInit(editor: any) {
    this.editor = editor;
    this.editor.language = this.getLanguageFromExtension(this.file.name)
    this.editor.onDidChangeModelContent((event: any) => {
      if (!this.isRemoteChange && !event.isFlush) {
        const operations = event.changes.map((change: any) => ({
          range: change.range,
          text: change.text
        }));
        this.socket.updateFile(operations, this.file.id);
      }
      this.isRemoteChange = false;
    });
  }
```

```
}
```

document-content.component.html

```
<ngx-monaco-editor class="d-flex flex-grow-1" [options]="editorOptions" [(ngModel)]="file.data"
        [style.width.px]="width" (onInit)="onEditorInit($event)"></ngx-monaco-editor>
```

file-tree.component.ts

```
import {Component, OnDestroy, OnInit} from '@angular/core';
import {ContextMenuAction, MonacoTreeElement} from "ngx-monaco-tree";
import {DialogData} from "../../../../core/models/dialog-data.dto";
import {DialogComponent} from "../../../../shared/components/dialog/dialog.component";
import {MatDialog} from "@angular/material/dialog";
import {FileService} from "../../services/file.service";
import {FileSocketService} from "../../../../core/services/file-socket.service";
import {ProjectSocketService} from "../../../../core/services/project-socket.service";
import {ProjectObject} from "../../../../core/models/project-object.dto";
import {Subscription} from "rxjs";

@Component({
  selector: 'app-file-tree',
  templateUrl: './file-tree.component.html',
  styleUrl: './file-tree.component.scss'
})
export class FileTreeComponent implements OnInit, OnDestroy {
  dark: boolean = true;
  author: boolean = false;
  tree: any[] = [];
  private subscriptions: Subscription[] = [];

  constructor(public dialog: MatDialog, public fileService: FileService,
          public fileSocket: FileSocketService, public projectSocket: ProjectSocketService) {}

  ngOnInit(): void {
    this.projectSocket.onRequestFiles();
    this.projectSocket.onFilesReceived();

    this.subscriptions.push(
      this.fileService.filesObservable$.subscribe({
        next: f => this.tree = f
      }),
      this.fileSocket.onAddFile().subscribe((o: ProjectObject) => {
        console.log(o.data)
        this.fileService.addFile(o.type, o.path, this.tree, o.fPath, o.name, false, o.id, o.data)
      }),
      this.fileSocket.onRenameFile().subscribe(({path, newName}) => {
        this.rename(newName, path, this.tree);
      }),
```

```
      this.fileSocket.onRemoveFile().subscribe((path) => {
        this.remove(path, this.tree);
      })
    );
  }

  ngOnDestroy(): void {
    this.subscriptions.forEach(subscription => subscription.unsubscribe());
  }

  handleContextMenu(action: ContextMenuAction) {
    switch (action[0]) {
      case 'new_directory':
        return this.fileService.createFile('directory', action[1], this.tree, "");
      case 'new_file':
        return this.fileService.createFile('file', action[1], this.tree, "");
      case 'delete_file':
        this.author = true;
        return this.remove(action[1], this.tree);
      case 'rename_file':
        this.author = true;
        return this.renameWindow(action[1], this.tree);
    }
  }

  private renameWindow(path: string, localTree: MonacoTreeElement[]) {
    const data: DialogData = {buttonTitle: "Перейменувати", placeholder: "Назва", title:
"Перейменувати"}
    const dialogRef = this.dialog.open(DialogComponent, {data});
    dialogRef.afterClosed().subscribe(result => {
      if (result) {
        this.rename(result, path, localTree)
      }
    });
  }

  rename(name: string, path: string, localTree: MonacoTreeElement[], index = 0) {
    const parts = path.split('/');
    const part = parts[index];
    const file = localTree.find(el => el.name === part);
    if (!file) return;
    if (this.author) {
      this.fileSocket.renameFile(path, name)
      this.author = false;
    }
    if (index === parts.length - 1) {
      file.name = name
      this.fileService.updateFile(path, name);
    } else if (file.content) {
```

```
      this.rename(name, path, file.content, index + 1);
    }
  }

  remove(path: string, localTree: MonacoTreeElement[], closeAllInDir = true) {
    const closeFileByPath = (filePath: string) => {
      const index = this.fileService.openFiles.findIndex(file => file.fPath === filePath);
      if (index !== -1) this.fileService.closeFile(index);
    };
    const processDirectory = (dirPath: string, tree: MonacoTreeElement[]) => {
      tree.forEach(el => {
        const fullPath = `${dirPath}/${el.name}`;
        el.content ? processDirectory(fullPath, el.content) : closeFileByPath(fullPath);
      });
    };
    if (closeAllInDir) {
      const [name] = path.split('/');
      const dir = localTree.find(el => el.name === name);
      if (dir?.content) processDirectory(name, dir.content);
    }
    closeFileByPath(path);
    const [name, ...rest] = path.split('/');
    const index = localTree.findIndex(el => el.name === name);
    if (this.author) {
      this.fileSocket.removeFile(path);
      this.author = false;
    }
    if (index !== -1) {
      if (rest.length === 0) {
        localTree.splice(index, 1);
      } else {
        const content = localTree[index].content;
        if (content) this.remove(rest.join('/'), content, false);
      }
    }
  }
}
```

file-tree.component.html

```
<monaco-tree class="d-flex flex-grow-1"
  [theme]="dark ? 'vs-dark' : 'vs-light'"
  [tree]="tree"
  (clickFile)="fileService.switchToTab($event)"
  (clickContextMenu)="handleContextMenu($event)"
></monaco-tree>
```

tabs.component.ts

```
import {Component, HostListener, OnInit, ViewChild} from '@angular/core';
import {ResizeService} from "../../services/resize.service";
```

```
import {MatTabGroup} from "@angular/material/tabs";
import {FileService} from "../../services/file.service";
import {FileModel} from "../../../../core/models/file.model";

@Component({
  selector: 'app-tabs',
  templateUrl: './tabs.component.html',
  styleUrl: './tabs.component.scss'
})
export class TabsComponent implements OnInit {
  @ViewChild('tabGroup') tabGroup!: MatTabGroup;
  tabs: FileModel[] = []
  width: number = 0;

  constructor(protected resizeService: ResizeService, protected fileService: FileService) {}

  @HostListener('window:resize', ['$event'])
  onResize() {
    this.resize()
  }

  closeTab(index: number) {
    this.fileService.closeFile(index)
  }

  resize()  {
    this.resizeService.currentLeftWidth.subscribe(value => {
      this.width = window.innerWidth - value;
      if (this.tabGroup) {
        this.tabGroup.updatePagination()
      }
    });
  }

  ngOnInit(): void {
    this.resize()
    this.fileService.openFilesObservable$.subscribe({
      next: t => this.tabs = t
    });
    this.fileService.tabIndexObservable$.subscribe(tabIndex => {
      if (tabIndex !== null) {
        this.tabGroup.selectedIndex = tabIndex;
      }
    });
  }
}
```

tabs.component.html

```
<mat-tab-group mat-stretch-tabs="false" class="d-flex" [style.width.px]="width" #tabGroup>
  <mat-tab *ngFor="let file of tabs; let index = index">
    <ng-template mat-tab-label>
      <span class="user-select-none">{{file.name}}</span>
      <button style="pointer-events: auto;" (click)="closeTab(index);$event.stopPropagation()" mat-icon-
button><mat-icon>close</mat-icon></button>
    </ng-template>
    <app-document-content class="d-flex flex-grow-1 flex-column" [file]="file"></app-document-
content>
  </mat-tab>
</mat-tab-group>
```

workspace-main.component.ts

```
import {Component, HostListener, OnInit} from '@angular/core';
import {ResizeService} from "../../services/resize.service";
import {ActivatedRoute} from "@angular/router";
import {ProjectService} from "../../../../core/services/project.service";

@Component({
  selector: 'app-workspace-main',
  templateUrl: './workspace-main.component.html',
  styleUrl: './workspace-main.component.scss'
})
export class WorkspaceMainComponent implements OnInit {
  leftWidth = 384;
  leftVisible = true;
  isResizing = false;
  minWidth = 0;
  maxWidth = 0;

  constructor(private resizeService: ResizeService, private route: ActivatedRoute, private projectService:
ProjectService) {
  }

  ngOnInit(): void {
    this.minWidth = window.innerWidth * 0.05;
    this.maxWidth = window.innerWidth * 0.95;
    this.leftWidth = Math.ceil(window.innerWidth * 0.2);

    let id: string = this.route.snapshot.params['id']
    if (id) {
      this.projectService.getProject(id)
    }
  }

  @HostListener('document:mousemove', ['$event'])
  onMouseMove(event: MouseEvent) {
    if (this.isResizing) {
```

```
      const newValue = Math.max(this.minWidth, Math.min(this.maxWidth, event.clientX));
      this.resizeService.updateLeftWidth(newValue);
      this.leftWidth = newValue;
    }
  }

  @HostListener('document:mouseup')
  onMouseUp() {
    this.isResizing = false;
  }

  startResizing() {
    this.isResizing = true;
  }

  toggleLeft(): void {
    this.leftVisible = !this.leftVisible;
    if (this.leftVisible) {
      this.resizeService.updateLeftWidth(this.leftWidth);
    } else {
      this.resizeService.updateLeftWidth(0);
    }
  }
}
```

workspace-main.component.html

```
<div class="d-flex flex-row" style="height: calc(100vh - 48px)">
  <div *ngIf="leftVisible" class="d-flex overflow-x-hidden position-relative"
[style.width.px]="leftWidth">
    <app-file-tree class="d-flex flex-grow-1"></app-file-tree>
    <button class="button-close" (click)="toggleLeft()">
      <mat-icon>keyboard_arrow_left</mat-icon>
    </button>
  </div>
  <div *ngIf="leftVisible" class="resizer flex-shrink-0" (mousedown)="startResizing()"></div>
  <div class="d-flex flex-grow-1 position-relative" style="background: #1E1E1E">
    <button *ngIf="!leftVisible" class="button-open" (click)="toggleLeft()">
      <mat-icon>keyboard_arrow_right</mat-icon>
    </button>
    <app-tabs class="d-flex"></app-tabs>
  </div>
</div>
```