НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)
Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Архітектурне рішення для файлового сховища

Виконав студент IV курсу, групи ІП-01
(шифр групи)
Заранік Богдан Юрійович (підпис)
(прізвище, ім'я, по батькові)

Керівник професор, д.т.н., проф., засл.діяч, Павлов О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент професор кафедри ІСТ, д.т.н., проф. Корнага Я. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Зараніку Богдану Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Архітектурне рішення для файлового сховища
2. керівник проєкту Павлов Олександр Анатолійович, д.т.н., проф., засл.діяч
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. № 2112-с

3. Термін подання студентом проєкту « 17 » червня 2024 року

4. Вихідні дані до проєкту: технічне завдання

5. Зміст пояснювальної записки

1) Передпроєктне обстеження предметної області.
2) Розроблення вимог до програмного забезпечення.
3) Конструювання та розроблення програмного забезпечення.
4) Аналіз якості та тестування програмного забезпечення.
5) Розгортання та супровід програмного забезпечення.

6. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

7. Консультанти розділів проєкту

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

8. Дата видачі завдання «11» березня 2024 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.03.2024	
3	Постановка та формалізація задачі	31.03.2024	
4	Розробка інформаційного забезпечення	14.04.2024	
5	Алгоритмізація задачі	19.04.2024	
6	Обґрунтування вибору використаних технічних засобів	10.05.2024	
7	Розробка програмного забезпечення	19.05.2024	
8	Налагодження програми	22.05.2024	
9	Виконання графічних документів	01.06.2024	
10	Оформлення пояснювальної записки	01.06.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Богдан ЗАРАНІК

(підпис) (ініціали, прізвище)

Керівник _____ Олександр ПАВЛОВ

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 26 таблиць, 27 рисунків та 50 джерел – загалом 60 сторінок.

Дипломний проект призначений для забезпечення файлового сховище засобами онлайн-перегляду раніше недоступних форматів файлів користувача із можливістю імпорту файлів із сховища Google Drive на основі запропонованого оригінального архітектурного рішення.

Мета: спрощення процесу додавання нових підтримуваних форматів файлів, перегляд яких раніше був недоступний онлайн.

У розділі “Передпроектне обстеження предметної області” було розглянуто предметну область, загальні терміни й процеси, що відбуваються у сфері розробки файлових сховищ, а також визначено поширені методи, засоби розробки та наявні на зараз проблеми, що стосуються розробки подібних систем.

Розділ “Розроблення вимог до програмного забезпечення” присвячений розробленню та висуванню вимог до програмного забезпечення. В розділі наведено варіанти використання, функціональні і нефункціональні вимоги.

У розділі “Конструювання та розроблення програмного забезпечення” було проведено дослідження та огляд використаної у розробці архітектури, засобів розробки і конструювання програмного забезпечення, опис структури бази даних, утиліт, наведено аналіз безпеки даних.

У розділі “Аналіз якості та тестування програмного забезпечення” було наведено оцінку якості отриманого коду за вказаними метриками, описано процеси тестування та наведено контрольний приклад для перевірки відповідності програмного забезпечення висунутим до нього вимогам.

У розділі “Розгортання та супровід програмного забезпечення” було описано процеси і особливості розгортання та супроводу розробленого програмного забезпечення.

КЛЮЧОВІ СЛОВА: СХОВИЩЕ, АРХІТЕКТУРА, API, SPRING, JAVASCRIPT, АВТОМАТИЗАЦІЯ, ВІДКРИТИЙ КОД, DOCKER, SVELTE

ABSTRACT

The explanatory note of the diploma project consists of five chapters, contains 26 tables, 27 figures and 50 sources - a total of 60 pages.

The diploma project is designed to provide the file storage with means of online viewing of previously unavailable user file formats with the possibility of importing files from the Google Drive storage based on the proposed original architectural solution.

The goal: to simplify the process of adding new supported file formats that were previously not available to view online.

In the section "Pre-design survey of the subject area" the subject area, general terms and processes occurring in the field of file storage systems development were considered, as well as common methods, means of development and existing problems related to the development of such systems were identified.

The section "Development of software requirements" is devoted to the development and presentation of software requirements. The section provides options for use, functional and non-functional requirements.

In the "Software design and development" section, a study and review of the architecture used in the development, software development and design tools, a description of the database structure, utilities, and a data security analysis were conducted.

In the section "Quality analysis and testing of the software" the quality assessment of the received code according to the specified metrics was given, the testing processes were described and a control example was given for checking the compliance of the software with the requirements put forward to it.

In the section "Deployment and maintenance of software" the processes and features of deployment and maintenance of the developed software were described.

KEYWORDS: STORAGE, ARCHITECTURE, API, SPRING, JAVASCRIPT, AUTOMATION, OPEN SOURCE, DOCKER, SVELTE

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-0110.045440.01.91	Технічне завдання	19	
3	А4	КПІ.ІП-0110.045440.02.81	Пояснювальна записка	60	
4	А4	КПІ.ІП-0110.045440.03.12	Текст програми	35	
5	А4	КПІ.ІП-0110.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-0110.045440.05.34	Керівництво користувача	9	
7	А3	КПІ.ІП-0110.045440.06.99	Графічний матеріал	4	

КПІ.ІП-0110.045440.00.90

Змін	Арк.	№ докум.	Підп.	Дата
Розроб.		Заранік Б. Ю.		
Перевір.		Павлов О.А.		
Н.контр.		Головченко М.М.		
Затв.		Жариков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІП-01

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Архітектурне рішення для файлового сховища

Технічне завдання

КПІ.ІП-0110.045440.01.91

“ПОГОДЖЕНО”
Керівник проекту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Богдан ЗАРАНІК

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «Архітектурне рішення для файлового сховища».

Галузь застосування:

Галузь застосування: наведене технічне завдання поширюється на розробку веб-сервісу під назвою "Kexit Drive", який функціонує як хмарне файлове сховище. Основні функціональні вимоги включають в себе можливість входу користувачів через Google, управління файлами та каталогами, імпорт файлів з Google Drive, можливість переглядати файли будь-яких типів та зручний інтерфейс користувача.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Kexit Drive» є завдання на дипломне проектування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для забезпечення файлового сховище засобами онлайн-перегляду раніше недоступних форматів файлів користувача із можливістю імпорту файлів із сховища Google Drive на основі запропонованого оригінального архітектурного рішення.

Метою розробки є спрощення процесу додавання нових підтримуваних форматів файлів, перегляд яких раніше був недоступний онлайн.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу ПЗ

4.1.1.1 Сторінка входу (рисунок 4.1);

4.1.1.1.1 Напис «Sign up with Google»;

4.1.1.1.1.1 знаходиться над кнопкою «Google»;

4.1.1.1.2 кнопка «Google»;

4.1.1.1.2.1 при натисканні відправляє запит на логін та відкриває форму згоди на предоставлення інформації профіля застосунку (рисунок 4.2);

4.1.1.1.2.2 у випадку успішного підтвердження надання дозволу відбувається перехід на сторінку кореневого каталогу користувача (рисунок 4.4);

4.1.1.1.2.3 у випадку неуспішного запиту на вхід відкриває сторінку логіну із повідомленням про не успішну спробу логіну (рисунок 4.3);

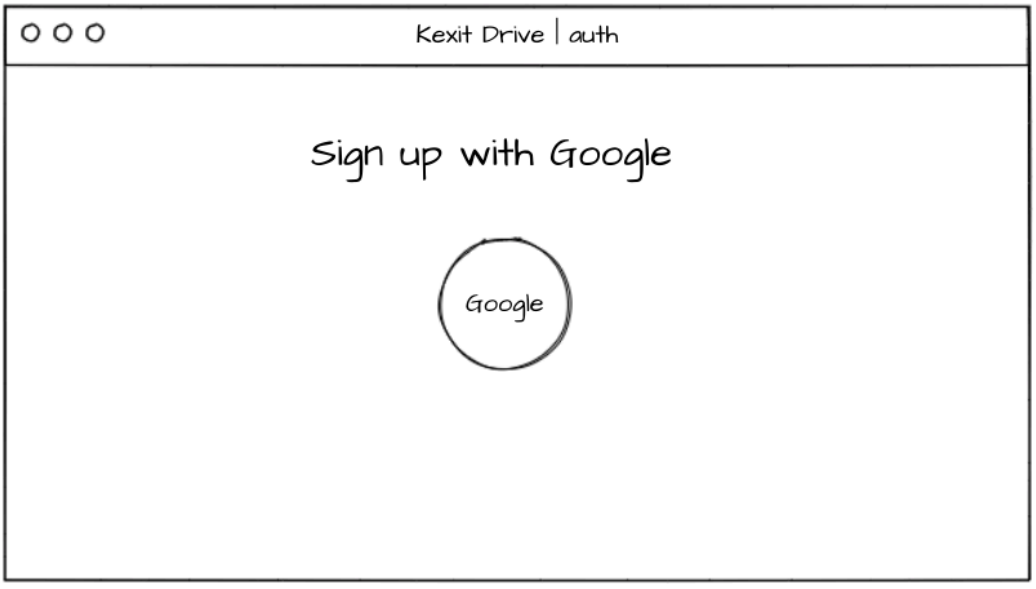


Рисунок 4.1 – Сторінка входу

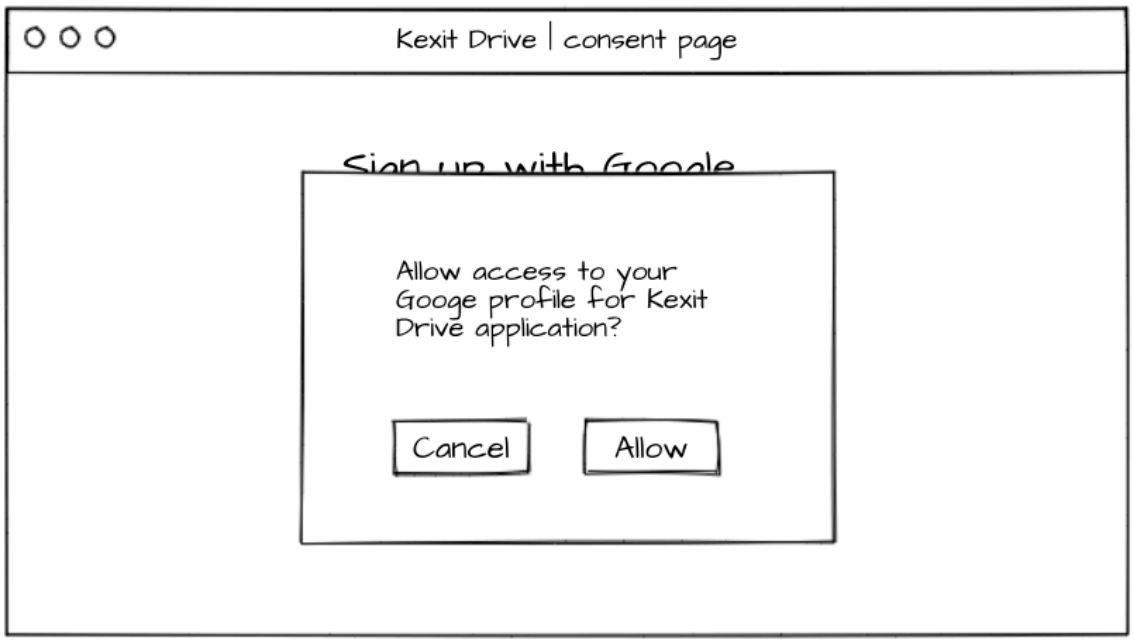


Рисунок 4.2 – Вікно згоди на надання інформації профілю

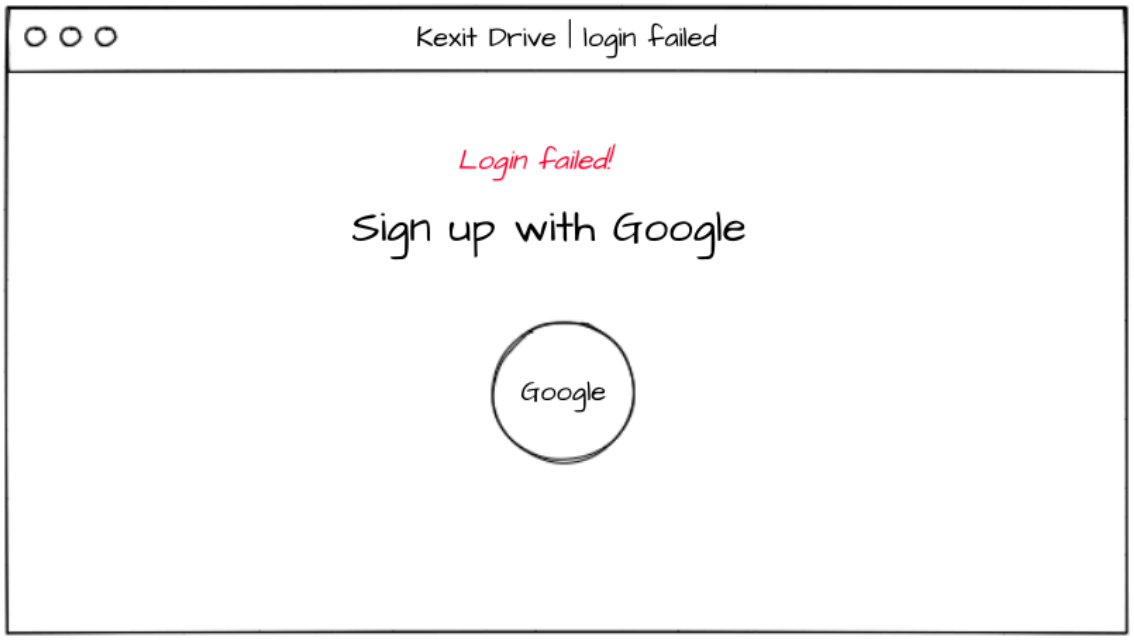


Рисунок 4.3 – Сторінка логіну із повідомленням про не успішну спробу логіну

4.1.1.2 вікно згоди на надання інформації профілю (рисунок 4.2);

4.1.1.2.1 модальне вікно запиту на надання дозволу використання інформації профілю;

4.1.1.2.2 текстове повідомлення з питанням про згоду надання дозволу використання інформації профілю;

4.1.1.2.3 кнопка «Cancel» - відміна входу;

4.1.1.2.3.1 при натисканні відкриває сторінку логіну із повідомленням про не успішну спробу логіну (рисунок 4.3);

4.1.1.2.4 кнопка «Allow» - надання дозволу;

4.1.1.2.4.1 при натисканні відбувається перехід на сторінку кореневого каталогу користувача (рисунок 4.4);

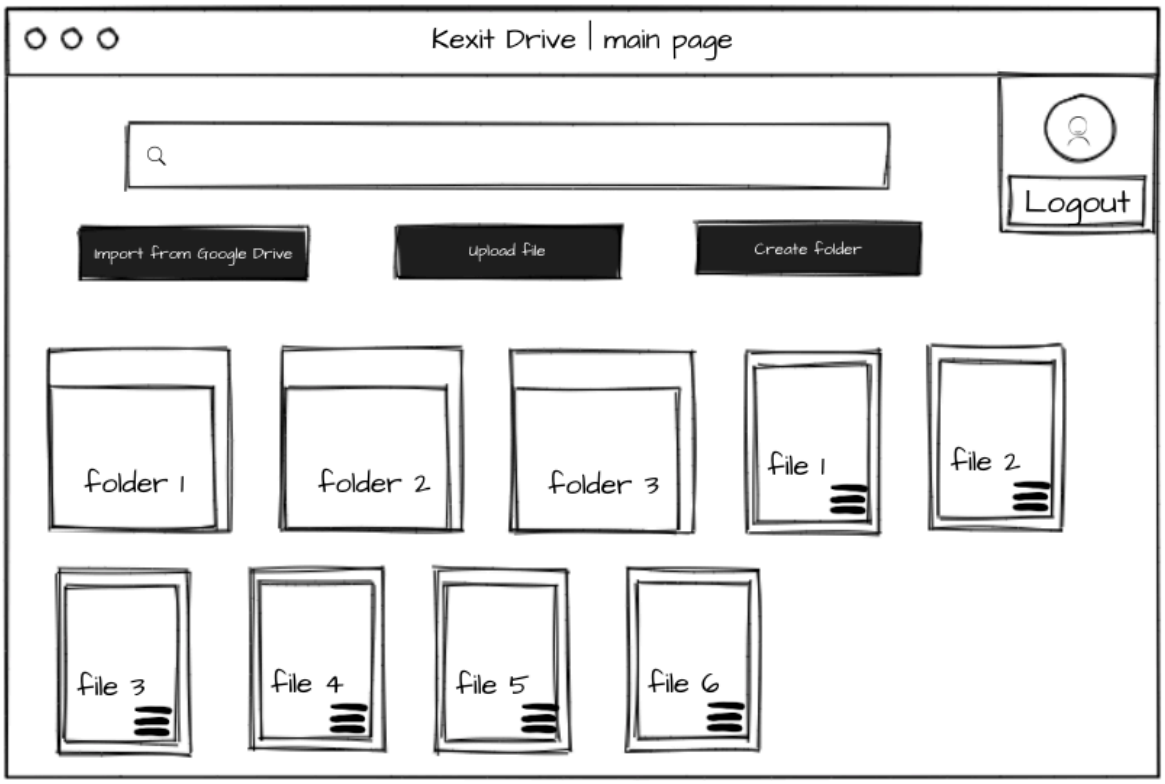


Рисунок 4.4 – Сторінка певного каталогу користувача

4.1.1.3 сторінка кореневого (та будь-якого) каталогу користувача (рисунок 4.4);

4.1.1.3.1 список директорій та файлів (рисунок 4.5);

4.1.1.3.1.1 при одинарному кліку на файл відкривається детальна інформація про даний файл (рисунок 4.6). На ній міститься кнопка для завантаження даного файла та детальний опис даного файлу: розмір, тип, назва і т.д.;

4.1.1.3.1.2 при подвійному кліку на файл він відкривається на програвання;

4.1.1.3.1.3 при одинарному кліку на папку вона відкривається і зображується її внутрішня структура - сторінка даного каталогу (рисунок 4.4);

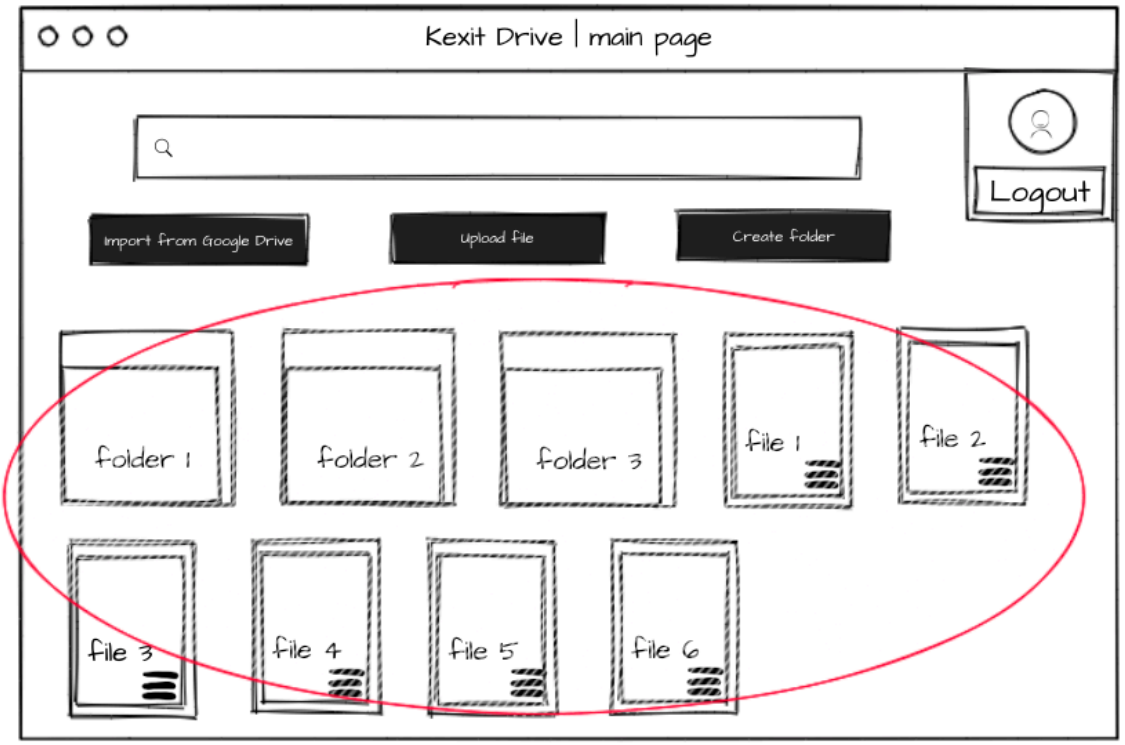


Рисунок 4.5 – Список папок і файлів певного каталогу

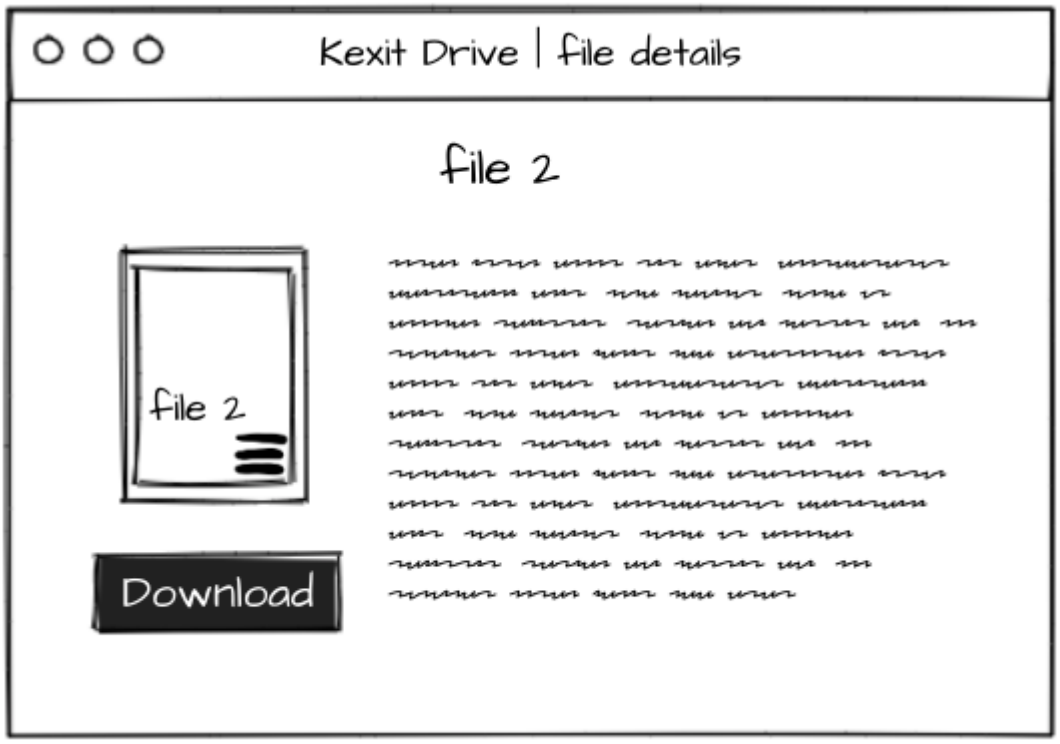


Рисунок 4.6 – Сторінка детальної інформації про файл

4.1.1.3.2 кнопка «Upload file» (рисунок 4.7);

4.1.1.3.2.1 при натисканні даної кнопки відкривається стандартний файловий менеджер браузера для завантаження файлів у дану директорію;

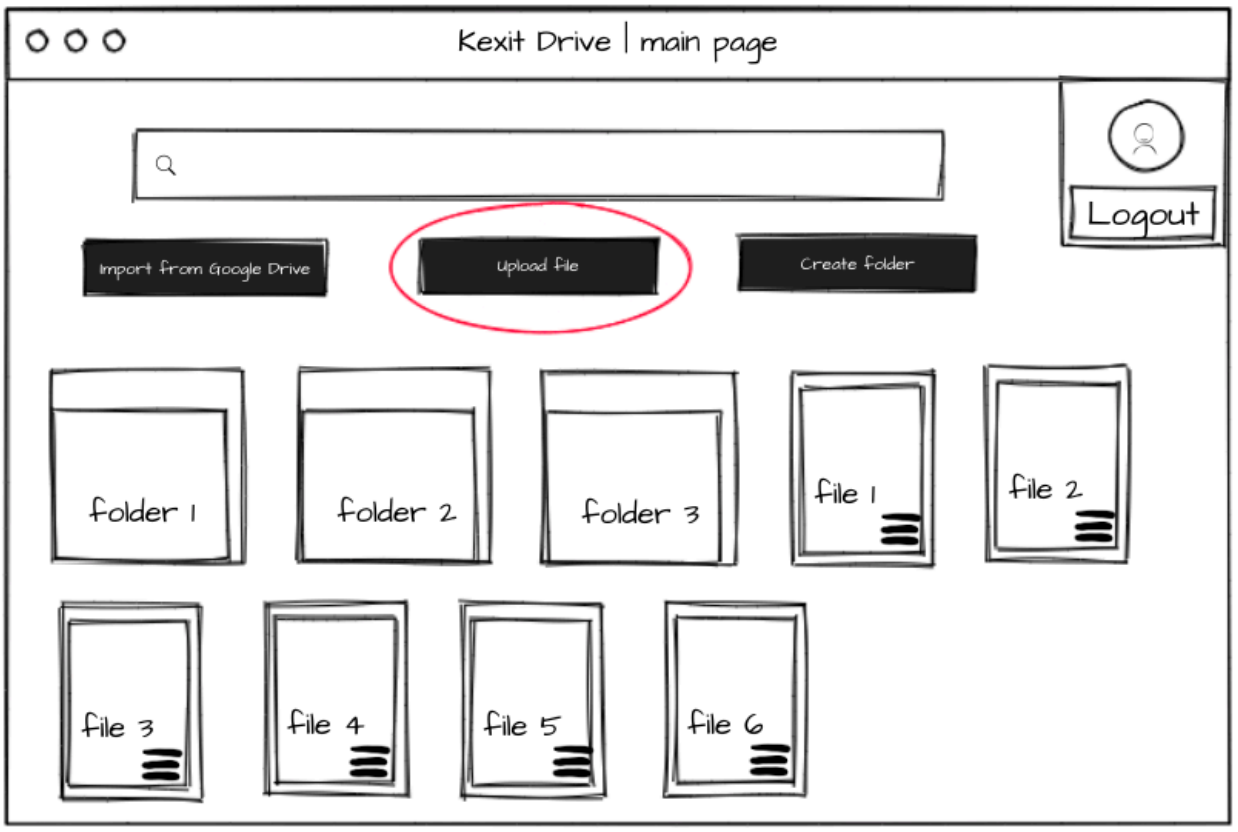


Рисунок 4.7 – Кнопка «Upload file»

4.1.1.3.3 кнопка «Import from Google Drive» (рисунок 4.8);

4.1.1.3.3.1 при натисканні відкривається модальне вікно вибору файлів на Google Drive даного користувача, яке містить список файлів та кнопку «Select» (рисунок 4.9). При виборі деяких файлів файли імпортуються до поточного каталогу;

4.1.1.3.4 кнопка «Create folder» (рисунок 4.10);

4.1.1.3.4.1 при натисканні на кнопку з'являється модальне вікно вводу назви директорії

4.1.1.3.4.2 поле має обмеження на довжину 50 символів

4.1.1.3.4.3 після введення назви директорії і натиснення клавіші «Enter» вона створюється

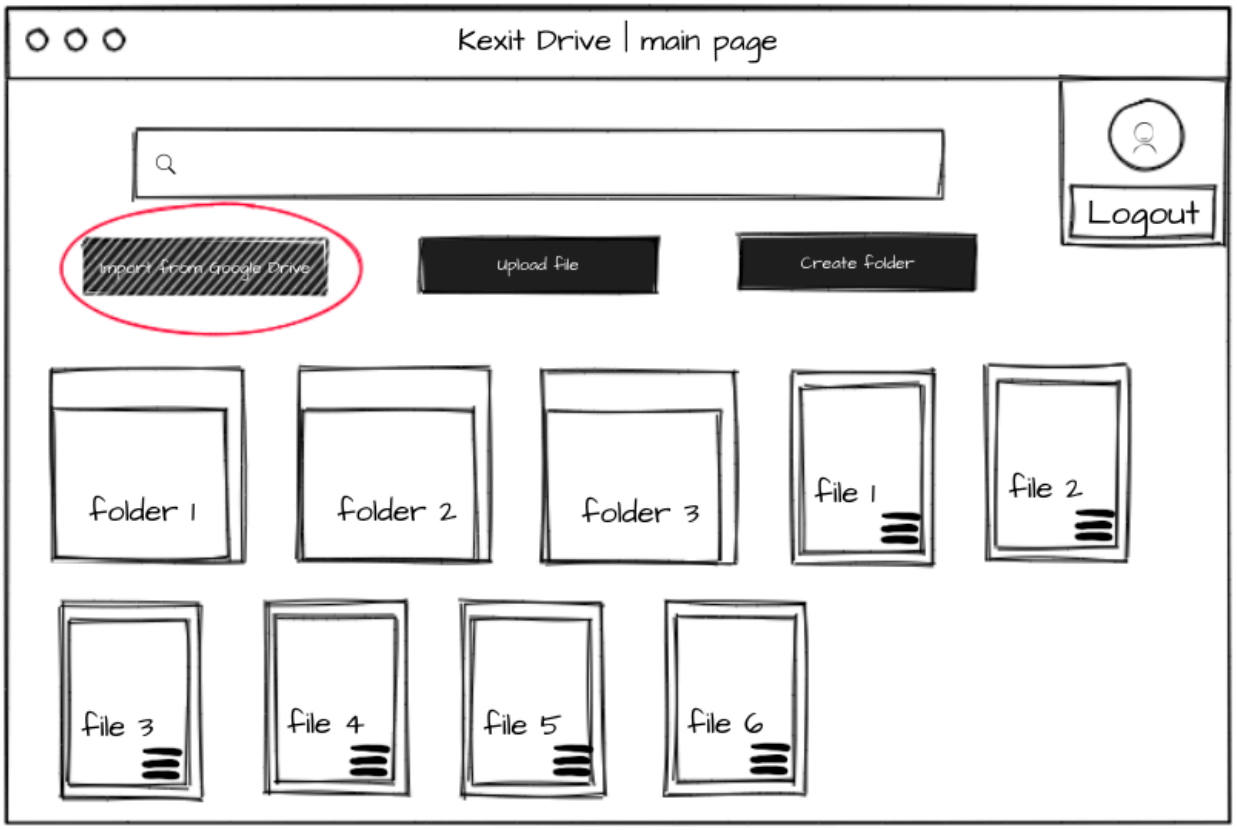


Рисунок 4.8 – Кнопка «Import from Google Drive»

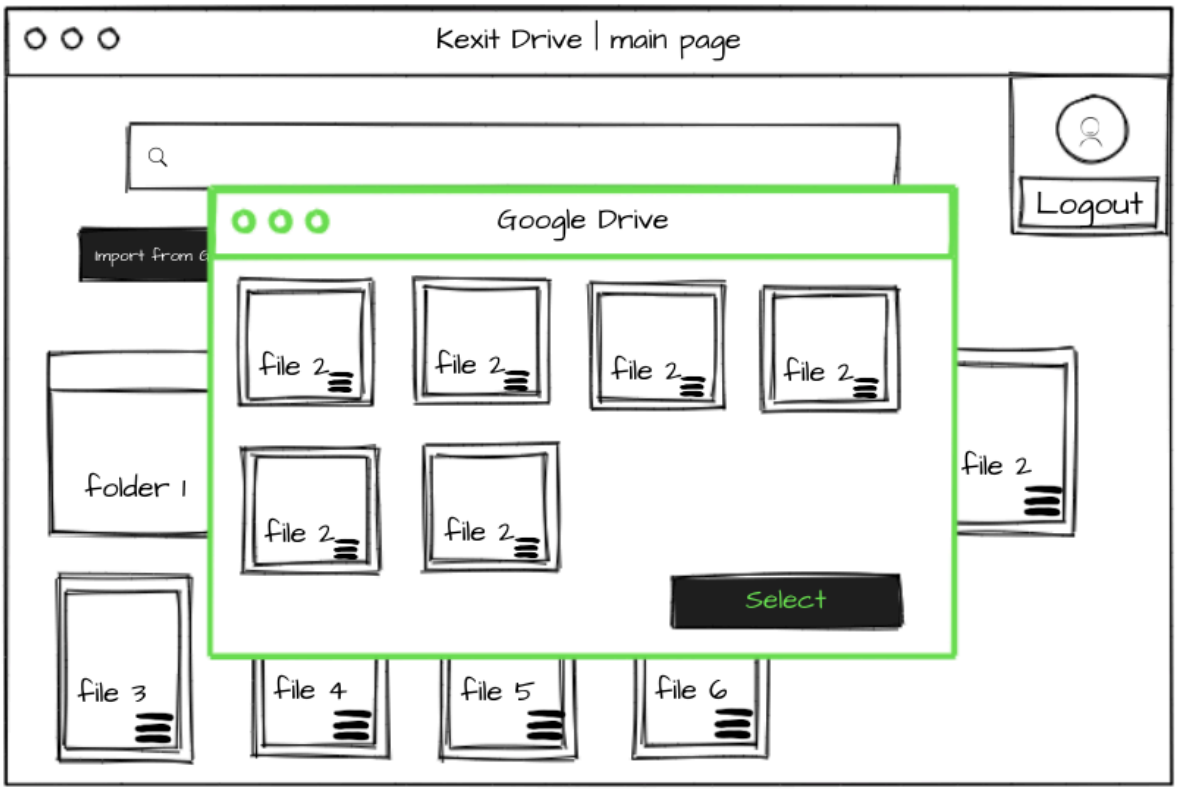


Рисунок 4.9 – Модальне вікно вибору файлів із Google Drive

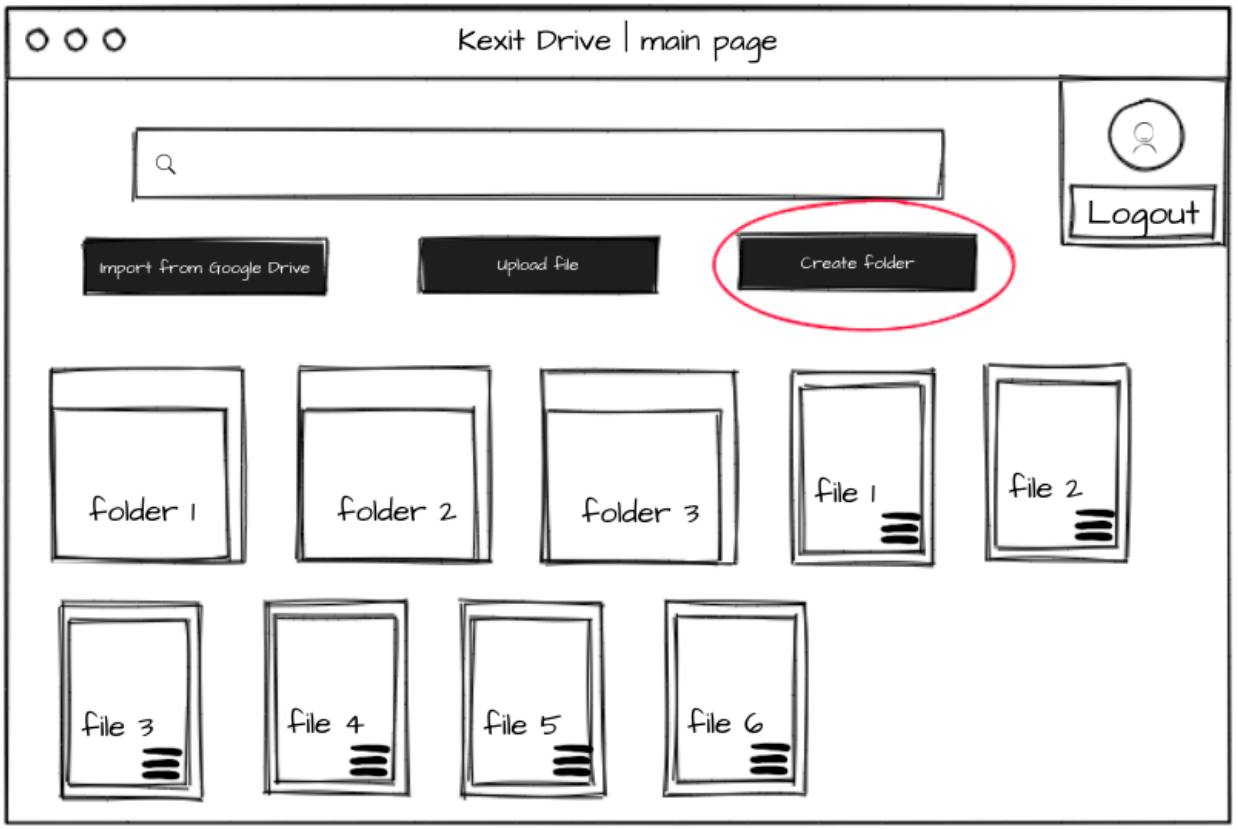


Рисунок 4.10 – Кнопка «Create folder»

4.1.1.3.5 поле пошуку (рисунок 4.11)

4.1.1.3.5.1 приймає будь-які текстові символи та фільтрує файли та папки за назвою;

4.1.1.3.5.2 довжина поля для пошуку має бути мінімум 3 символи;

4.1.1.3.6 іконка користувача і кнопка «Logout» (рисунок 4.12);

4.1.1.3.6.1 зображено фото користувача з Google аккаунта;

4.1.1.3.6.2 при натисканні на кнопку «Logout» користувач виходить із системи;

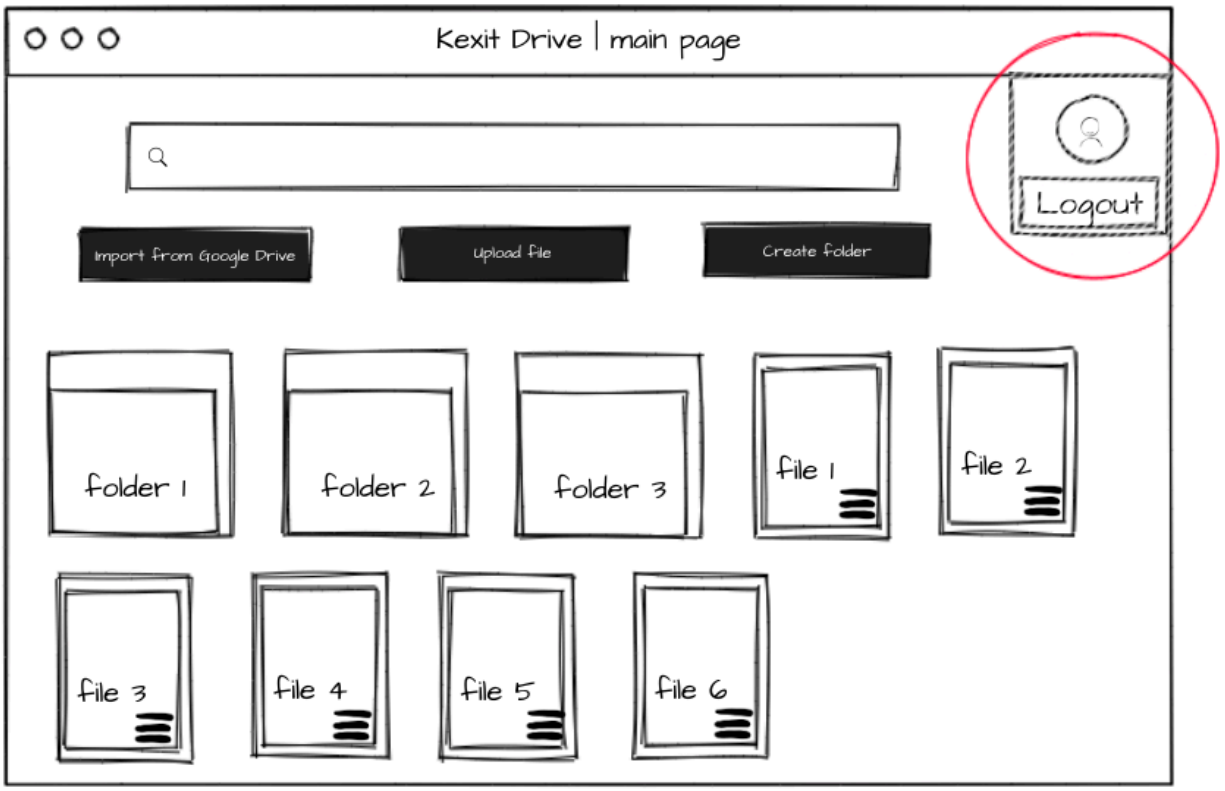


Рисунок 4.12 – Іконка користувача та кнопка «Logout»

4.1.2 Додаткові вимоги

4.1.2.1 застосування фреймворку Spring Boot;

4.1.2.2 вимоги до програмного інтерфейсу зображені у таблиці 4.1.

Таблиця 4.1 – Вимоги до програмного інтерфейсу

Контролер	Ендпоінт	Метод	Опис
auth	/api/oauth2/authorization/google	GET	Вхід користувача
	/api/logout	DELETE	Вихід користувача
user	/api/user/current	GET	Отримання власного профілю користувача
	/api/user/getAccessToken	GET	Токен доступу для фронтенду для роботи з Google Drive Picker

Продовження таблиці 4.1

file	/api/file/upload	POST	Завантаження нового файлу з файлової системи комп'ютера
	/api/file/:id	GET	Отримання файла за id
	/api/file/:id/download	GET	Завантаження файлу
	/api/file/:id	DELETE	Видалення файлу
	/move?fileId=&directoryId=	POST	Переміщення файла до директорії
	/import	POST	Імпорт файлів із Google Drive

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних. Для авторизації використовувати протокол OAuth 2, для аутентифікації - OIDC. Використовувати сервер авторизації, що надає Google.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Розгортання та підтримка програмного забезпечення, інтегрування робочих процесів.

4.3.2 Обслуговуючий персонал

Обслуговуючий персонал має мати необхідні навички та знання для розгортання, підтримки програмного забезпечення та інтегрування робочих процесів.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесора: Intel Core i3;
– об‘єм ОЗП: 8 Гб;
– об’єм постійної пам’яті: 512 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип процесора: Intel Core i7;
– об‘єм ОЗП: 32 Гб;
– об’єм постійної пам’яті: 2 Тб;
– підключення до мережі Інтернет зі швидкістю від 300 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Unix з встановленою платформою Docker.

4.5.1 Вимоги до вхідних даних

Файли будь-якої структури і форматів.

4.5.2 Вимоги до вихідних даних

Спеціальним чином оформлені вихідні дані відсутні.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Java 21 та JavaScript ES6

4.5.4 Вимоги до середовища розробки

Розробку виконати у середовищі JetBrains IntelliJ IDEA Ultimate Edition

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді файлів з розширеннями .java, .html, .css, .js, .mjs, .cjs, .json, .

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальною вимогою є створення докер-образів для подальшого розгортання програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема бази даних;
- креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”

Пояснювальна записка
до дипломного проекту

на тему: Архітектурне рішення для файлового сховища

КПІ.ІП-0110.045440.01.91

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки
API	–	Application programming interface, прикладний програмний Інтерфейс
SPA	–	Single page application – односторінковий додаток
IT	–	Інформаційні технології
ER	–	Entity-Relation diagram
JWT	–	JSON Web Token
БД	–	База даних
OIDC		Протокол аутентифікації OpenID Connect
OAuth2		Протокол авторизації OAuth2
SOA		Service Oriented Architecture (Сервіс-орієнтована архітектура)

ВСТУП

Сучасний розвиток інформаційних технологій вимагає створення ефективних і надійних рішень для зберігання та управління даними. З кожним роком обсяги інформації, які потребують зберігання та обробки, зростають в геометричній прогресії. Це зумовлює необхідність розробки інноваційних підходів до організації файлових сховищ, що здатні забезпечити високу продуктивність, масштабованість та безпеку даних. Особливо важливим є забезпечення доступності даних у будь-який час і з будь-якого місця, що підвищує вимоги до надійності та швидкості роботи таких систем.

Сучасні веб-сервіси для зберігання даних, такі як хмарні файлові сховища, повинні надавати користувачам можливість швидкого доступу до інформації, зручного управління файлами, а також інтеграції з іншими популярними сервісами для обміну та обробки даних.

Важливою функцією файлового сховища є можливість перегляду файлів у режимі онлайн. На жаль, більшість вже існуючих рішень не надає можливості перегляду файлів усіх типів, а лише з певного вузького списку форматів.

Тому постає питання в розробці оригінального архітектурного рішення, яке дозволить ефективно управляти файлами користувачів, забезпечити їх безпеку та доступність, інтеграцію з існуючими сервісами для максимальної зручності користувачів та можливість перегляду файлів онлайн у браузері.

Підсумовуючи, можна стверджувати, що розробка файлового сховища із можливістю перегляду файлів будь-яких типів шляхом реалізації певної архітектури дозволить створити конкурентоспроможне рішення для хмарного зберігання даних, яке відповідатиме сучасним вимогам до зручності, швидкості та безпеки.

1 ПЕРЕДПРОЕКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Предметна область цієї теми визначається реалізацією хмарного файлового сховища із можливістю онлайн-перегляду файлів будь-якого типу.

Файлове сховище є важливим компонентом для зберігання, управління та доступу до великих обсягів даних. Метою даного проекту є розробка сучасного та ефективного рішення для файлового сховища, яке задовольнятиме потреби українських споживачів та буде доступним на різних платформах.

На сьогодні ринок систем для управління файлами ще має вільне місце. Більшість відомих рішень є застарілими або дорогими для локальних користувачів, що створює потребу в новому, більш адаптованому програмному забезпеченні.

Мета - створити програмне забезпечення для файлового сховища, яке буде доступним для користувачів різних платформ. Це ПЗ повинно бути легко масштабованим та інтегрованим з іншими системами, такими як телеграм боти, які можуть автоматично надсилати звіти.

Розробка нового файлового сховища дозволить задовольнити потреби українських користувачів, забезпечивши зручний та ефективний інструмент для управління файлами. Можливість перегляду будь-яких типів файлів онлайн дасть суттєву бізнес-перевагу у порівнянні із конкурентами.

1.2 Аналіз існуючих рішень

Проаналізуємо сучасні алгоритмічні рішення та технічні засоби, які сприятимуть реалізації веб застосунку для управління мережею кінотеатрів. Розглянемо готові програмні рішення, допоміжні програмні інструменти та засоби розробки.

1.2.1 Аналіз відомих програмних продуктів

Головним гравцем у даному сегменті послуг є компанія Google [37] та її продукт - хмарне файлове сховище Google Drive [38].

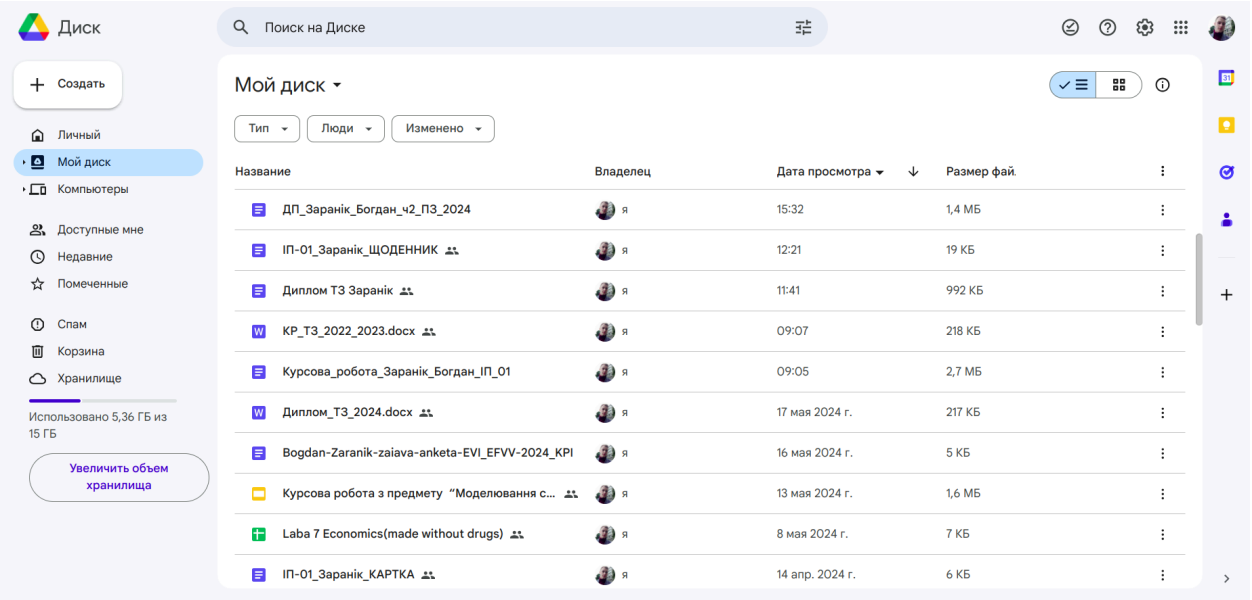


Рисунок 1.1 – Сторінка головної директорії хмарного сховища Google Drive

Для порівняння проекту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Kexit Drive	Google Drive [18]	Dropbox [33]	Пояснення
Ціни на додаткове сховище	безкоштовно	від $1.99/міс за 100 ГБ	$9.99/міс за 2 ТБ	-
Синхронізація і доступ	інтеграція з Google Drive	доступ через браузер, мобільні додатки для Android та iOS	доступ через браузер, мобільні додатки для Android та iOS	-
Онлайн-перегляд файлів довільних типів	+	-	-	Головная функціональність даної роботи

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

В ході розробки ПЗ не було використано алгоритмів, що потребують пояснень в даному підрозділі.

Стосовно технічних рішень, головним було визначити тип архітектури для веб застосунку. Зважаючи на необхідність забезпечення високої продуктивності, швидкості відгуку системи та гнучкості у масштабуванні, вибір стояв між клієнт-серверною та монолітною архітектурою.

Сервер [39] - це комп’ютер або програма, яка приймає та обробляє запити від клієнтів, виконує необхідну бізнес-логіку, керує базами даних та повертає результати. Сервери забезпечують централізоване управління даними та ресурсами.

Клієнт [40] - це програма або пристрій, який надсилає запити до сервера для виконання певних операцій. Клієнтами можуть бути веб-браузери, мобільні додатки або навіть інші сервери. Вони слугують точкою взаємодії для користувачів, дозволяючи їм отримувати доступ та керувати ресурсами сервера.

Клієнт-серверна архітектура [40] представляє модель взаємодії, в якій клієнт надсилає запити до сервера, а сервер обробляє ці запити та повертає відповіді. Ця модель дозволяє централізувати логіку та дані, що сприяє ефективному управлінню ресурсами та підвищенню безпеки.

До переваг можна віднести:

– гнучкість. Клієнти можуть бути різноманітними (веб, мобільні додатки) та працювати на різних платформах;

– ефективність: сервери можуть бути оптимізовані для виконання специфічних задач;

– масштабованість: можливість додавання або оновлення серверів без зміни клієнтської частини;

До недоліків можна віднести:

– залежність від сервера: Якщо сервер виходить з ладу, це може призвести до зупинки всієї системи;

– складність: може вимагати більше ресурсів для управління та підтримки інфраструктури.

Монолітна архітектура [41] - це традиційна модель програмного забезпечення, яка являє собою єдиний модуль, що працює автономно та незалежно від інших додатків. Моноліт часто асоціюється з чимось великим і громіздким, що точно відображає його сутність. Це окрема велика обчислювальна структура з єдиною кодовою базою, де об'єднана вся бізнес-логіка. Веб-застосунок [43] у такій архітектурі складається з тих самих компонентів, що і в клієнт-серверній, але всі вони інтегровані в один додаток.

Переваги монолітної архітектури:

– простота розробки: усі компоненти знаходяться в одному місці, що полегшує розуміння та розробку;

– простота розгортання: один великий блок легше розгорнути, ніж безліч дрібних сервісів.

Недоліки монолітної архітектури:

– слабка масштабованість: з часом кодова база стає громіздкою, що ускладнює впровадження нових технологій;

– обмежена гнучкість: будь-яке невелике оновлення потребує повного повторного розгортання.

У рамках даної дипломної роботи було обрано клієнт-серверну архітектуру як основний архітектурний шаблон через її масштабованість, гнучкість та ефективність. У сучасному світі, де все змінюється дуже швидко, ці показники є критично важливими.

Для даної дипломної роботи в якості основного архітектурного рішення була обрана саме клієнт-серверна архітектура через масштабованість, гнучкість та ефективність, оскільки в сучасному світі, де все змінюється не по днях а по годинах ці показники є критично важливими.

Також важливим рішенням є вибір архітектурного рішення для серверної частини застосунку. Основний вибір постав між n-рівневою та чистою архітектурами.

Монолітна N-рівнева архітектура [41] передбачає поділ системи на декілька рівнів, кожен з яких відповідає за певний аспект функціонування програми. Такий підхід дозволяє розподілити відповідальність між різними шарами, що може сприяти кращій організації коду та спрощенню розробки. Наприклад, може бути окремий рівень для інтерфейсу користувача, бізнес-логіки, доступу до даних тощо.

До переваг можна віднести:

– модульність: легкість внесення змін у певні рівні без впливу на інші.

До недоліків можна віднести:

– складність: управління залежностями між рівнями може бути складним;

– затримки: взаємодія між рівнями може призводити до затримок у відгуку.

У цій роботі була обрана чиста архітектура [42] як основний архітектурний шаблон для розробки серверної частини програмного забезпечення. Цей вибір був зроблений завдяки високій гнучкості та простоті тестування, які важливі для забезпечення стабільності та якості програми. Чиста архітектура полегшує адаптацію до мінливих вимог і технологій. Це особливо важливо в сучасному динамічному технологічному світі. Крім того, це сприяє написанню чистого коду, який зручно підтримувати, що полегшує розробку та впровадження нових функцій.

1.3 Опис бізнес-процесів

Для опису бізнес процесу використовується BPMN [19] модель основні процеси зображені на рисунках 1.2 - 1.3.

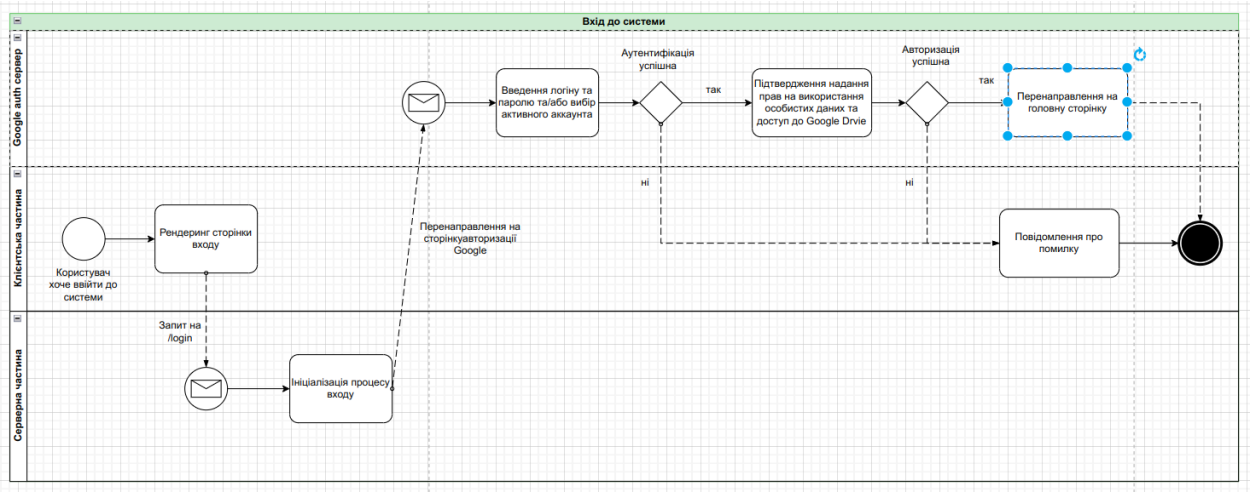


Рисунок 1.2 – Схема входу до системи

Опис послідовності входу до системи:

- користувач переходить на сторінку входу;
- користувач натискає на кнопку логіну і переходить на сторінку вводу логіну та паролю;
- користувач переходить на сторінку логіну в Google акаунт;
- у раз успішної аутентифікації користувач переходить на сторінку згоди на надання застосунку прав на доступ до ресурсів Google;
- у разі помилки, користувач переходить на сторінку із описом помилки;
- на сторінці згоди на надання застосунку прав на доступ до ресурсів Google користувач натискає “Погоджуюсь” або “Відмінити”;
- у випадку відмови користувач переходить на сторінку із помилкою;
- у випадку згоди користувача на використання особистих даних у застосунку користувач переходить головну сторінку застосунку;

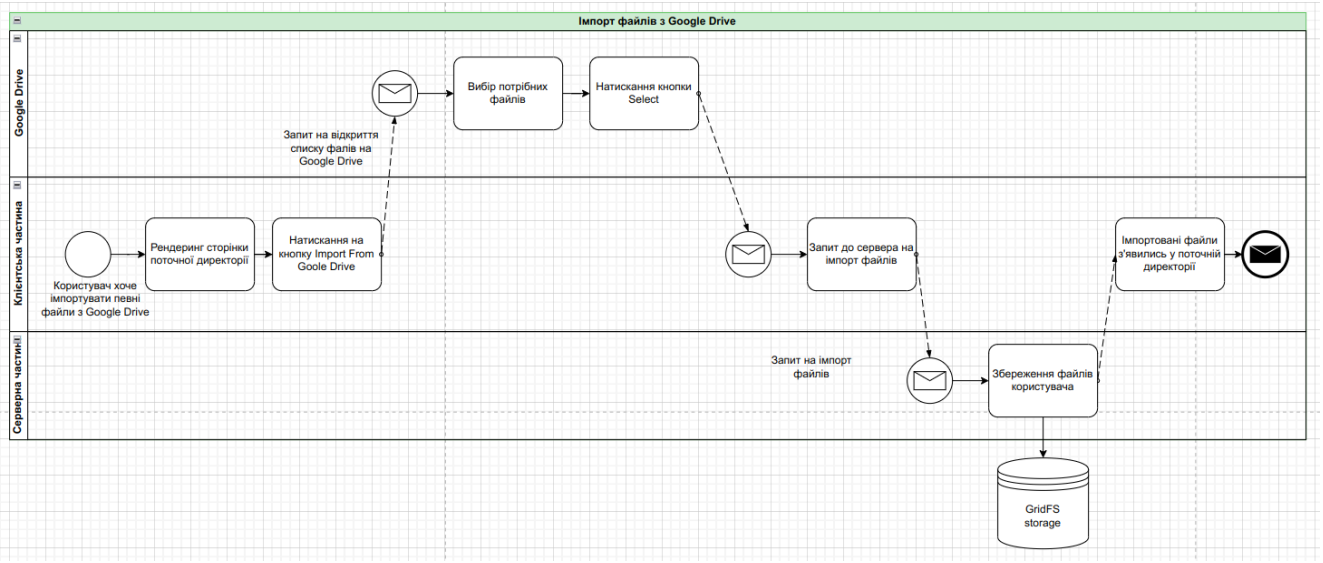


Рисунок 1.3 – Схема Імпорту файлу з Google Drive [18]

Опис послідовності входу до системи:

- відкривається сторінка поточної директорії;
- авторизований користувач натискає на кнопку “Import from Google Drive”;
- користувачеві відкривається вікно з вибором файлів його власного сховища Google Drive [18];
- користувач обирає файли зі списку у даному вікну та натискає кнопку “Select”;
- відбувається запит на сервер на імпорт файлів до сховища;
- файли імпортуються до поточної директорії користувача;
- файли відображаються у поточній директорії користувача.

1.4 Постановка задачі

Розробка призначена для забезпечення файлового сховище засобами онлайн-перегляду раніше недоступних форматів файлів користувача із можливістю імпорту файлів із сховища Google Drive[18] на основі запропонованого оригінального архітектурного рішення.

Метою розробки є спрощення процесу додавання нових підтримуваних форматів файлів, перегляд яких раніше був недоступний онлайн

Основні завдання, що підлягають розв’язанню в рамках розробки, включають:

- можливість входу через Google;
- можливість перегляду власного профіля користувача;
- можливість імпорту файлів із Google Drive [18];
- можливість завантаження файлів із компьютера;
- можливість створювати нові папки;
- можливість керувати файлами та папками;
- можливість переглядати будь-які типи файлів онлайн;
- можливість швидко додавати нові типи файли до списку тих, що можна переглядати.

Висновки до розділу

У першому розділі було проведено ретельний аналіз поточних ринкових рішень, щоб точно визначити їхні плюси та мінуси - це допомогло визначити сфери, на які має звернути увагу новий веб-додаток. Зокрема, було наголошено, що додаток має бути сумісним на всіх платформах, а також необхідно запровадити архітектурне рішення. Це рішення дозволить користувачам переглядати свої онлайн-файли незалежно від їх типу.

Під час огляду існуючих рішень було виявлено, що більшість доступного програмного забезпечення є застарілим і не відповідає сучасним стандартам, що спонукає до необхідності створення нової альтернативи з високими можливостями адаптації.

Також були визначені чіткі цілі для розробки веб-додатку, призначеного як всеохоплюючу систему для нагляду за різними аспектами функціонування файлового сховища: проектування, опис вимог та адаптації до програвання будь-яких типів файлів онлайн.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Програмне забезпечення для файлового сховища розроблено для задоволення різноманітних потреб користувачів у зберіганні та управлінні файлами. Нижче наведено основні сценарії використання цього програмного забезпечення на рисунку 2.1.

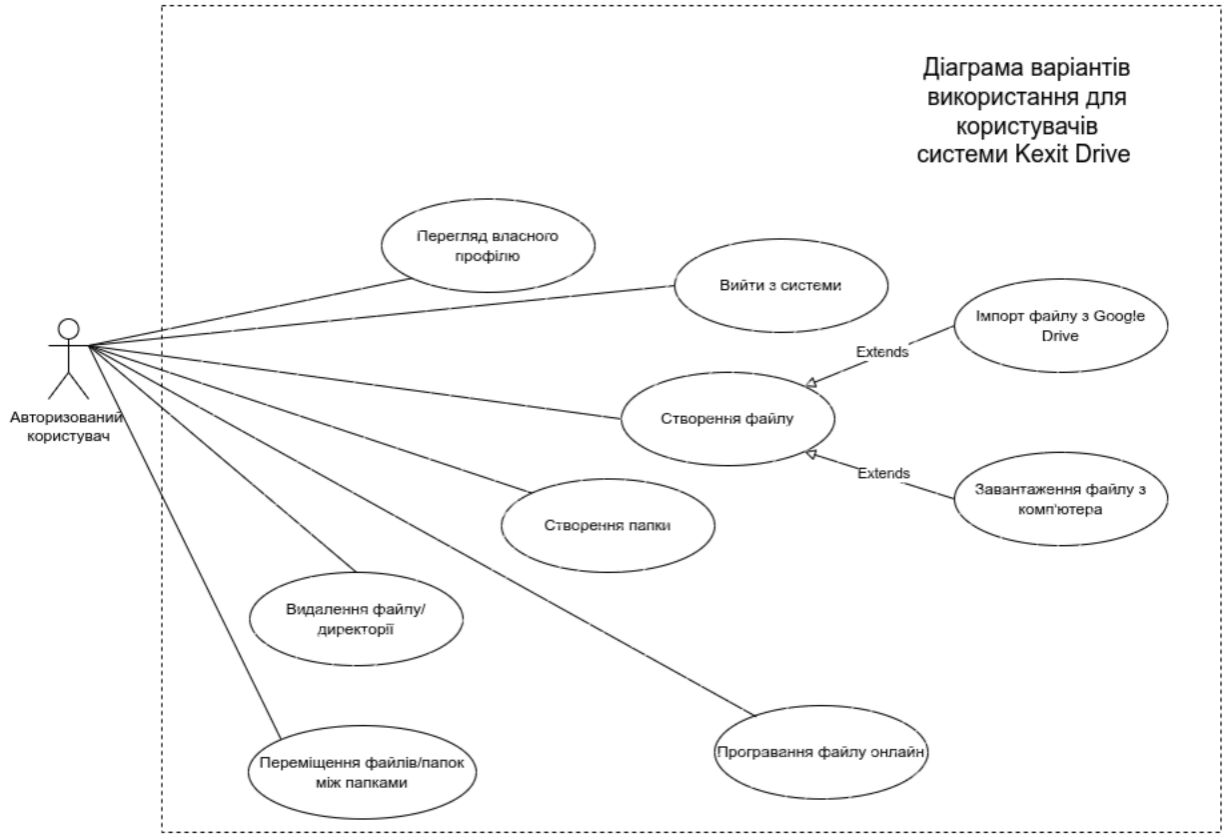


Рисунок 2.1 – Діаграма варіантів використання

Для більш зручного читання загальну діаграму варіантів використання було розбито на менші діаграми, які можна побачити у кресленні 1.

В таблицях 2.1 - 2.9 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Увійти до системи
Use case ID	UC-01
Goals	Вхід нового користувача в системі
Actors	Гість (неавторизований користувач)
Trigger	Користувач бажає увійти до системи
Pre-conditions	-
Flow of Events	Користувач переходить на авторизаційну сторінку та натискає на кнопку «Login with Google». Далф переходить на сторінку вибору Google акаунта, обравши його вводить логін і пароль, погоджується з використанням особистих даних та переходить на головну сторінку.
Extension	У випадку введення неправильного пароля або не надання дозволу на використання особистих даних, користувач переходить на сторінку помилки.
Post-Condition	Користувач переходить на головну сторінку.

Таблиця 2.2 - Варіант використання UC-2

Use case name	Перегляд власного профіля користувача
Use case ID	UC-02
Goals	Перегляд власного профіля користувача
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути власний профіль користувача
Pre-conditions	Користувач авторизований
Flow of Events	Користувач натискає на іконку із власною картинкою з Google акаунта та бачить усі дані свого профіля користувача

Продовження таблиці 2.2

Extension	-
Post-Condition	-

Таблиця 2.3 - Варіант використання UC-3

Use case name	Вийти з системи
Use case ID	UC-03
Goals	Вийти з системи
Actors	Авторизований користувач
Trigger	Користувач бажає вийти з системи
Pre-conditions	Користувач авторизований
Flow of Events	Користувач натискає на кнопку “Logout”.
Extension	-
Post-Condition	Перехід на сторінку логіну.

Таблиця 2.4 - Варіант використання UC-4

Use case name	Імпорт файлу з Google Drive
Use case ID	UC-04
Goals	Імпортувати файл із Google Drive у поточну директорію
Actors	Авторизований користувач
Trigger	Користувач бажає імпортувати файл із Google Drive у поточну директорію
Pre-conditions	Користувач авторизований та переглядає сторінку певної директорії

Продовження таблиці 2.4

Flow of Events	Користувач натискає кнопку «Import». В модальному вікні, що з’явилося, відображається вміст його Google Drive сховища, де він може обрати потрібні файли та натиснувши “Select”імпортувати їх до поточної директорії.
Extension	У випадку натискання кнопки “Cancel” модальне вікно закривається, файли не імпортуються.
Post-Condition	Обрані файли імпортуються до поточної директорії

Таблиця 2.5 - Варіант використання UC-5

Use case name	Завантаження файла з комп’ютера
Use case ID	UC-05
Goals	Завантажити файл з комп’ютера
Actors	Авторизований користувач
Trigger	Користувач бажає завантажити файл у поточну директорію
Pre-conditions	-
Flow of Events	Користувач натискає кнопку “Upload”, обрає файл, який хоче завантажити та натиає “Select” у модальному вікні, що відкривається.
Extension	-
Post-Condition	До поточної директорії завантажується обраний файл

Таблиця 2.6 - Варіант використання UC-6

Use case name	Створення папки
Use case ID	UC-06
Goals	Створити папку

Продовження таблиці 2.6

Actors	Авторизований користувач
Trigger	Користувач бажає створити директорію у поточній директорії
Pre-conditions	-
Flow of Events	Користувач натискає кнопку “Create folder”. У відкритому модальному вікні користувач вводить назву директорії та натискає кнопку “Ok”
Extension	-
Post-Condition	Створено нову папку

Таблиця 2.7 - Варіант використання UC-7

Use case name	Видалення папки/файлу
Use case ID	UC-07
Goals	Видалити папку/файл
Actors	Авторизований користувач
Trigger	Користувач бажає видалити папку/файл
Pre-conditions	Авторизований користувач переглядає сторінку певної папки
Flow of Events	Користувач натискає кнопку «Delete» на директорії/файлі. У відкритому модальному вікні він настикає “Delete”.
Extension	У випадку натискання “Cancel” у модальному вікні, вікно закривається, папка/файл не видаляється.
Post-Condition	Папка/файл видаляється.

Таблиця 2.8 - Варіант використання UC-8

Use case name	Програвання файлу онлайн
Use case ID	UC-08
Goals	Програти обраний файл плеєром онлайн
Actors	Авторизований користувач
Trigger	Користувач бажає програти обраний файл плеєром онлайн
Pre-conditions	-
Flow of Events	Користувач двічі натискає на обраний файл. Відкривається плеєр та програє файл онлайн.
Extension	У випадку, якщо на даний момент формат файлу не підтримується,
Post-Condition	-

2.2 Аналіз системних вимог

Аналіз системних вимог є ключовим етапом у розробці програмного забезпечення, оскільки дозволяє визначити необхідні апаратні та програмні ресурси для коректного функціонування системи. У даному проекті було проведено ретельний аналіз системних вимог для забезпечення стабільної та ефективної роботи хмарного файлового сховища "Kexit Drive".

Рекомендовані вимоги:

Процесор: Intel Core i3. Мінімальні вимоги до процесора обрано з метою забезпечення базової продуктивності для роботи серверної частини системи. Intel Core i3 забезпечує достатню обчислювальну потужність для обробки основних запитів і виконання базових операцій.

Оперативна пам'ять (ОЗП): 8 ГБ. Цей обсяг оперативної пам'яті забезпечує необхідний простір для одночасного виконання декількох процесів та обробки даних, що важливо для стабільної роботи системи.

Постійна пам'ять: 512 ГБ. Мінімальний обсяг постійної пам'яті обрано для забезпечення достатнього місця для зберігання файлів користувачів та системних даних.

Підключення до мережі Інтернет: швидкість від 20 мегабіт. Мінімальна швидкість Інтернет-з'єднання обрана для забезпечення базової швидкості доступу до файлів та онлайн-перегляду.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. У таблицях 2.9 та наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.3.

Таблиця 2.9 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Можливість увійти до системи	FR-1	Високий	Високий
2	Можливість створити папку	FR-2	Високий	Високий
3	Можливість імпортувати файл	FR-3	Високий	Високий
4	Можливість завантажити файл	FR-4	Високий	Високий
5	Можливість програвати файл онлайн	FR-5	Високий	Високий
6	Можливість видалення файлу/папки	FR-6	Високий	Високий
7	Можливість перегляду власного профілю	FR-7	Середній	Середній
8	Можливість вийти з системи	FR-8	Середній	Середній
9	Можливість перегляду інформації про файл	FR-9	Середній	Середній
10	Можливість переміщення файлів/папок між папками	FR-10	Середній	Високий

Таблиця 2.10 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Можливість увійти до системи. Вхід до системи реалізовано за допомогою протоколу OAuth 2 та OIDC. Провайдер серверу авторизації - Google.
FR-2	Можливість створити папку. Папка створюється у поточній директорії та є порожньою.
FR-3	Можливість імпортувати файл. Користувач натискає кнопку «Import». В модальному вікні, що з’явилося, відображається вміст його Google Drive сховища, де він може обрати потрібні файли та натиснувши “Select” імпортувати їх до поточної директорії.
FR-4	Можливість завантажити файл. Користувач натискає кнопку “Upload”, обрає файл, який хоче завантажити та натиає “Select” у модальному вікні, що відкривається.
FR-5	Можливість програвати файл онлайн. Користувач двічі натискає на обраний файл. Відкривається плеєр та програє файл онлайн. У залежності від формату файлу відкривається потрібний плеєр.
FR-6	Можливість видалення файлу/папки. Користувач натискає кнопку «Delete» на директорії/файлі. У відкритому модальному вікні він настикає “Delete”. При видаленні папки видаляються всі файли та папки, які знаходяться у видаленій.
FR-7	Можливість перегляду власного профілю. Користувач натискає на іконку із власною картинкою з Google акаунта та бачить всі дані свого профіля користувача.
FR-8	Можливість вийти з системи. Користувач настикає на кнопку “Logout” і виходить з системи. Після цього він переходить на сторінку логіну.
FR-9	Можливість перегляду інформації про файл. При одинарному кліку на файлі відкривається модальне вікно із детальною інформацією про даний файл.
FR-10	Можливість переміщення файлів/папок між папками. При переміщенні папки до іншої папки усі файли також переносяться.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10
UC-1	+	+								
UC-2					+					
UC-3										
UC-4				+				+		
UC-5					+	+				
UC-6										
UC-7							+			
UC-8						+	+	+	+	+
UC-9										

Рисунок 2.2 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Нефункціональні вимоги відіграють фундаментальну роль у розробці та використанні програми, оскільки вони закладають основу для надійної та оптимізованої платформи. Ці вимоги забезпечують найвищий рівень якості та продуктивності. Нижче наведено основні нефункціональні вимоги до програми:

− безпека даних: додаток повинен мати ефективні механізми захисту персональних і конфіденційних даних користувачів, запобігання несанкціонованого доступу та вилучення інформації;

− інтуїтивний інтерфейс: інтерфейс має бути зручним і легким у навігації, щоб користувачі могли взаємодіяти з додатком і використовувати всі його функції без зайвих зусиль;

− масштабованість: додаток має бути розроблено таким чином, щоб легко масштабуватися для зростання обробки даних і користувачів, зберігаючи високу продуктивність;

− обсяг пам’яті: скільки застосунок є файловим сховищем, то кількість місця повинна бути достатньою для зберігання файлів усіх користувачів системи.

Висновки до розділу

У другому розділі було розроблено вимоги до програмного забезпечення для файлового сховища з урахуванням різноманітних потреб користувачів та особливостей ринку. Цей розділ охоплює варіанти використання програмного забезпечення, системні вимоги, функціональні та нефункціональні вимоги, що закладають основу для подальшої розробки та реалізації проекту.

Варіанти використання програмного забезпечення: було визначено основні сценарії використання, які включають зберігання файлів, онлайн-перегляд, синхронізацію з Google Drive, керування доступом, інтеграцію з іншими сервісами, захист даних, резервне копіювання, масштабованість, зручний інтерфейс користувача та підтримку мобільних пристроїв. Ці сценарії забезпечують універсальність та функціональність системи, задовольняючи потреби різних категорій користувачів.

Аналіз системних вимог: було проведено аналіз системних вимог, необхідних для стабільної та ефективної роботи програмного забезпечення. Визначено апаратні та програмні ресурси, які повинні бути забезпечені для коректного функціонування системи.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Високорівнева архітектура програмного забезпечення представлена у вигляді контекстної діаграми (рисунок 3.1), що є першим рівнем в архітектурній моделі C4 [20].

Діаграма C1 представляє контекст проекту, що розробляється. З рисунку видно, що основою всієї системи є застосунок Cimas, який в свою чергу за потреби може звертатися до хмарного сховища Google drive.

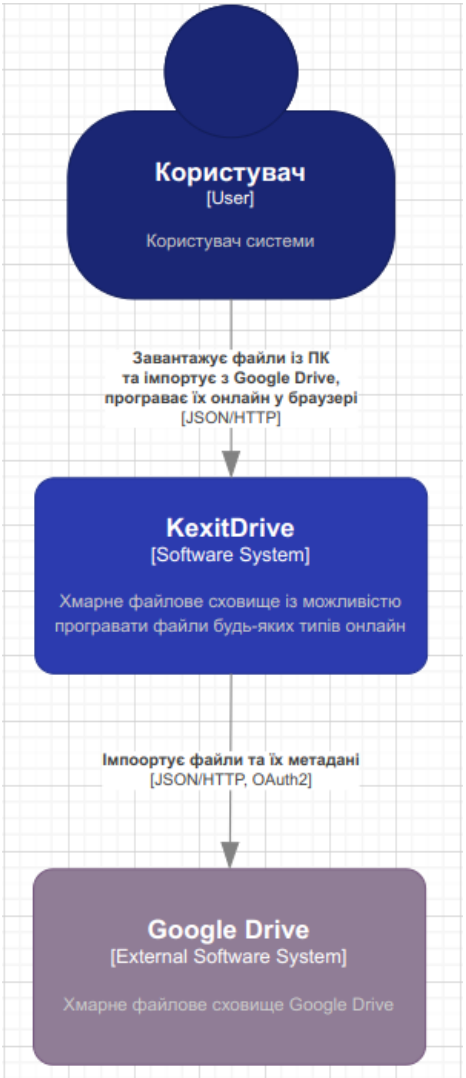


Рисунок 3.1 – Діаграма контексту

Діаграму контейнерів зображено на рисунку 3.2, що є другим рівнем представлення в моделі C4. Діаграму контейнерів знаходить у графічному матеріалі, креслення 3.

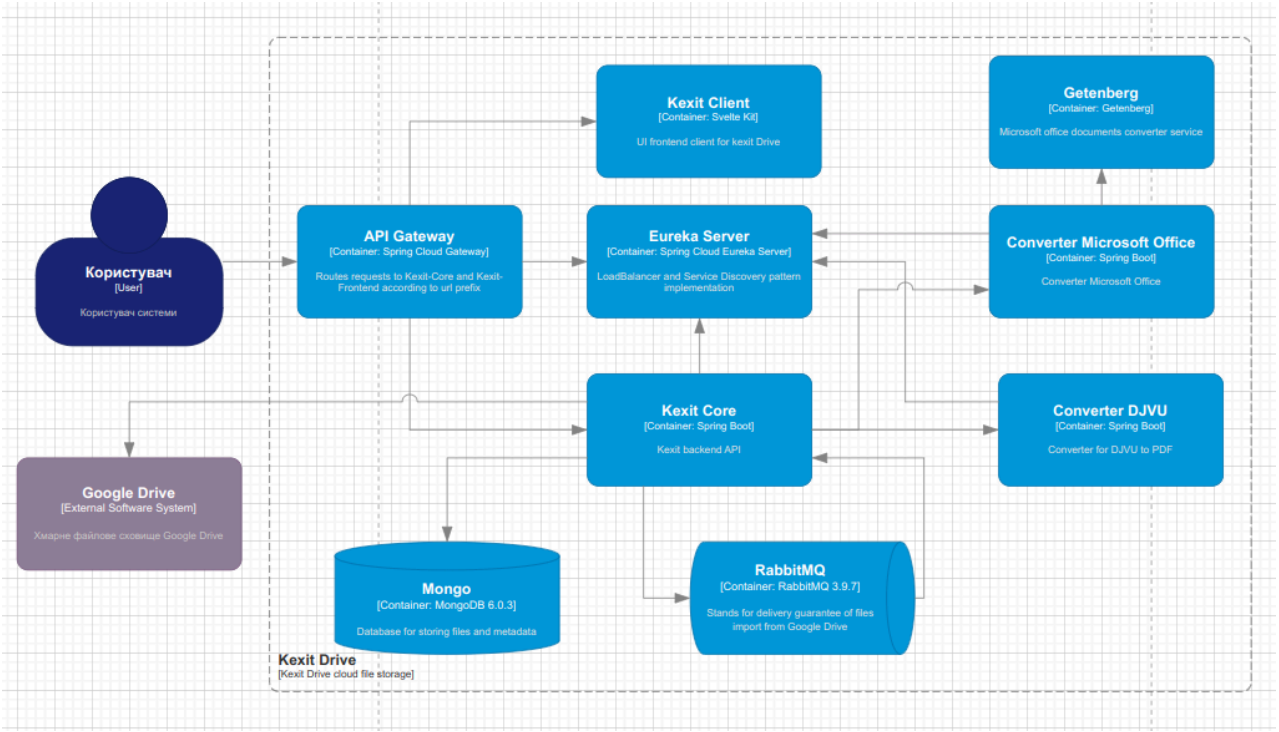


Рисунок 3.2 – Діаграма контейнерів

На моделі C2 зображена діаграма контейнерів. На діаграмі зображено усі контейнери системи Kexit Drive.

Api Gateway, що представлено у вигляді Spring Cloud Gateway [7] застосунку, який виконує роль роутера та єдиної точки входу.

Eureka Server - застосунок Spring Cloud Eureka Server, що виконує роль балансувальника навантаження та реалізацію паттерну “Service Discovery” [44].

Kexit Client - це клієнтський браузерний застосунок, що представлений у вигляді Svelte Kit застосунку. Він представляє візуальну частину системи та інтеграцію із Google Drive API.

Mongo [5] - контейнер, що є базою даних, у якій зберігаються завантажені та імпортовані файли користувача та їх метадані.

RabbitMQ [21] - контейнер, що є брокером повідомлень, які містять інформацію про файли, які потрібно імпортувати із сховища Google Drive [18] до поточної директорії. Сворювачем та споживачем даних повідомлень є сервіс Kexit Core. Даний механізм дозволяє забезпечити гарантію доставки повідомлень про імпорт файлів.

Як було зазначено у попередніх розділах, застосунок має сервіс-орієнтовану архітектуру (SOA) з метою декомпозиції функціоналу конверторів та надання можливості легко додавати нові сервіси-конвертори.

Сервіс-орієнтована архітектура (SOA) [22] – це архітектурний стиль, який підтримує інтеграцію різних сервісів через добре визначені інтерфейси. В основі SOA лежить концепція надання функціональності як окремих сервісів, які можуть взаємодіяти між собою через стандартизовані протоколи.

Основні переваги SOA:

− модульність: функціонал розділений на незалежні сервіси, що спрощує розробку, тестування та обслуговування.;

− можливість перезапуску: можливість оновлення або заміни окремих сервісів без впливу на інші компоненти системи;

− масштабованість: кожен сервіс можна масштабувати окремо в залежності від навантаження;

− повторне використання: сервіси можна повторно використовувати в різних додатках або процесах;

− гнучкість: легкість додавання нових сервісів-конверторів або модифікації існуючих без значного впливу на загальну архітектуру.

У даному застосунку SOA реалізована через набір окремих сервісів-конверторів, кожен з яких виконує певну роль у процесі конвертації даних. Ці сервіси взаємодіють через стандартизовані API, що забезпечує високий рівень інтеграції та взаємодії між ними. Наприклад, новий сервіс-конвертор може бути доданий до системи без потреби вносити зміни

до існуючих сервісів, завдяки чітко визначеним контрактам і протоколам взаємодії.

3.2 Використані патерни

3.2.1 Паттерн “Ланцюжок відповідальності” на прикладі Spring Security

У даній роботі використовується модуль фреймворку Spring Boot - Spring Security. Він надає змогу налаштовувати авторизацію у застосунку зручним чином, маючи багато готових компонентів. Головним патерном, який використовується у Spring Security - це “Ланцюжок відповідальності” або “Chain of responsibility” [45]. Він реєструє фільтри-обробники, які приймають рішення щодо наданні користувачеві доступ до певного ресурсу.

Переваги:

- гнучкість: легко додавати нові обробники або змінювати порядок існуючих;
- розширюваність: кожен обробник відповідає за свою частину логіки, що полегшує розширення системи новими функціями;
- зменшення зв'язності: кожен обробник виконує свою роботу незалежно від інших, що спрощує тестування та обслуговування.

Недоліки:

- продуктивність: якщо ланцюг обробників занадто довгий, це може вплинути на продуктивність, оскільки кожен запит проходить через всі обробники;
- важко відстежувати: у складних ланцюгах важко відстежувати, який обробник обробив запит або де виникла проблема;
- надлишковість: іноді обробники можуть виконувати надмірні перевірки або дублювати логіку.

3.2.2 “Service Discovery” на прикладі Spring Cloud Eureka Server

Переваги:

− динамічність: автоматичне виявлення нових сервісів і їхніх змін у реальному часі;

− масштабованість: підтримка великої кількості сервісів та їхньої взаємодії;

− зменшення ручного конфігурування: менше потреби у статичних конфігураціях сервісів, оскільки багато конфігурацій надається за замовчанням.

Недоліки:

− складність налаштування: потребує налаштування серверів реєстрації та клієнтів;

− точка відмови: сервер Eureka може стати єдиною точкою відмови, якщо не налаштовано кластеризацію;

− надлишкові запити: постійні запити для оновлення реєстру можуть створювати навантаження на мережу.

3.2.3 “API Gateway” [46] на прикладі Spring Cloud Gateway

Переваги:

− централізоване управління: єдина точка для керування трафіком, маршрутизації та політик безпеки;

− безпека: можливість централізовано впроваджувати політики безпеки, аутентифікації та авторизації;

− моніторинг та аналітика: централізоване логування та моніторинг запитів.

Недоліки:

− точка відмови: API Gateway може стати єдиною точкою відмови, якщо не налаштовано високу доступність;

− затримка: додатковий шар маршрутизації може додавати затримки в обробці запитів;

− складність: управління та конфігурація великої кількості маршрутів та політик може бути складним завданням.

3.3 Обґрунтування вибору засобів розробки

Для реалізації програмного забезпечення в цілях економії часу та полегшення процесу загалом було прийнято рішення використати вже готові фреймворки.

Для розробки клієнтської частини існує велика кількість готових фреймворків, кожен із яких має свої переваги та недоліки, сильні та слабкі сторони. На вибір було розглянуто такі фреймворки: Vue[23], React[24] та Svelte Kit[25].

Дані фреймворки є досить популярними на даний час і є лідерами у світовій індустрії розробки програмного забезпечення.

Vue - це фреймворк JavaScript[4], призначений для розробки веб-додатків. Він сприяє реактивним і компонентним методам проектування для створення інтерфейсів користувача.

Переваги:

– зручна та доступна документація: почати роботу легко, знайти відповідь на питання можна швидко;

– компактність: Vue має невелику кількість скомпільованого коду, що сприяє швидкому завантаженню на сторінку браузера та високій продуктивності;

– модульність: усі компоненти Vue ізольовані один від одного та можуть бути повторно використані, що спрощує розширення обсягу програм.

Недоліки:

– недостатня кількість кодової бази: Vue має меншу спільноту та менше інфраструктури порівняно з React;

– відносна складність написання програмного коду середнього та великого розміру у порівнянні із Svelte.

React – це бібліотека, розроблена та підтримувана компанією Facebook, і відкритим вихідним кодом. Надає широкі можливості для використання плагінів та модульність, що дозволяє ізолювати компоненти один від одного та використовувати їх повторно.

Переваги:

– проста: базовий синтаксис не потребує глибоких знань JavaScript;

– велике та активне співтовариство: має одну із найбільших спільнот у світі серед користувачів фреймворків;

– готові рішення: доступна велика кількість готових бібліотек, які інтегруються у застосунок та надають вже готові компоненти, які потрібно налаштувати згідно технічного завдання.

Недоліки:

– зворотна сумісність: відносно часті оновлення можуть вносити суттєві зміни у синтаксис фреймворку, що змушує підтримувати кодову базу в останній версії застосунку, що вимагає більших витрат коштів.

Svelte – це сучасний JavaScript-фреймворк, який виділяється тим, що перетворює компоненти у високоефективний імперативний код, який маніпулює DOM безпосередньо.

Переваги:

– висока продуктивність: завдяки відсутності залежності від бібліотеки під час виконання, Svelte забезпечує високу швидкість роботи;

– простота у вивченні: має простий та інтуїтивно зрозумілий синтаксис;

– мінімалістичний підхід: Svelte мінімізує обсяг коду, що потрібно написати для створення додатка.

Недоліки:

– менша популярність: спільнота та екосистема Svelte ще не такі великі, як у React чи Angular;

– мала кількість ресурсів: обмежена кількість навчальних матеріалів та бібліотек у порівнянні з більш популярними фреймворками.

У рамках даної дипломної роботи в якості основного засобу для розробки клієнтської частини було обрано фреймворк Svelte. Основною характеристикою, яка вплинула на прийняття даного рішення, стала простота у використанні даного фреймворку та малий розмір вихідного коду, що суттєво пришвидшує розробку клієнтської частини додатку.

Для розробки серверної частини веб застосунку було прийнято рішення, зважаючи на тенденції ринку,обрати фреймворк Spring Boot.

Spring Boot – це фреймворк для створення веб-застосунків і веб-сервісів, розроблений на основі Spring Framework [14]. Він значно спрощує налаштування та розгортання додатків на платформі Java [26]. Цей фреймворк також має різноманітні шаблони, включаючи API, який представляє собою набір методів і правил для взаємодії та обміну даними між різними програмами. Spring Boot з його підтримкою RESTful Web Services часто використовується для створення серверної частини сучасних веб-додатків.

Для забезпечення персистентності даних веб-застосунку також передбачається робота з базою даних. Спираючись на вище обраний фреймворк для полегшення розробки була обрана СКБД MongoDB [5].

MongoDB – це система управління базами даних, розроблена компанією MongoDB Inc. Ця програма виконує функції зберігання та надання даних у відповідь на запити інших застосунків, які можуть виконуватися як на тому ж самому сервері, так і в мережі. MongoDB є документо-орієнтованою базою даних NoSQL, що дозволяє зберігати дані у форматі JSON-подібних документів, забезпечуючи гнучкість і масштабованість для сучасних веб-додатків і служб. Також MongoDB надає можливість зберігати файли завдяки технології GridFS [27], яка використовується у даній роботі.

3.4 Конструювання програмного забезпечення

На рисунку 3.2 наведено EDR-діаграму сутностей. EDR-діаграма знаходиться у графічному матеріалі, креслення 1

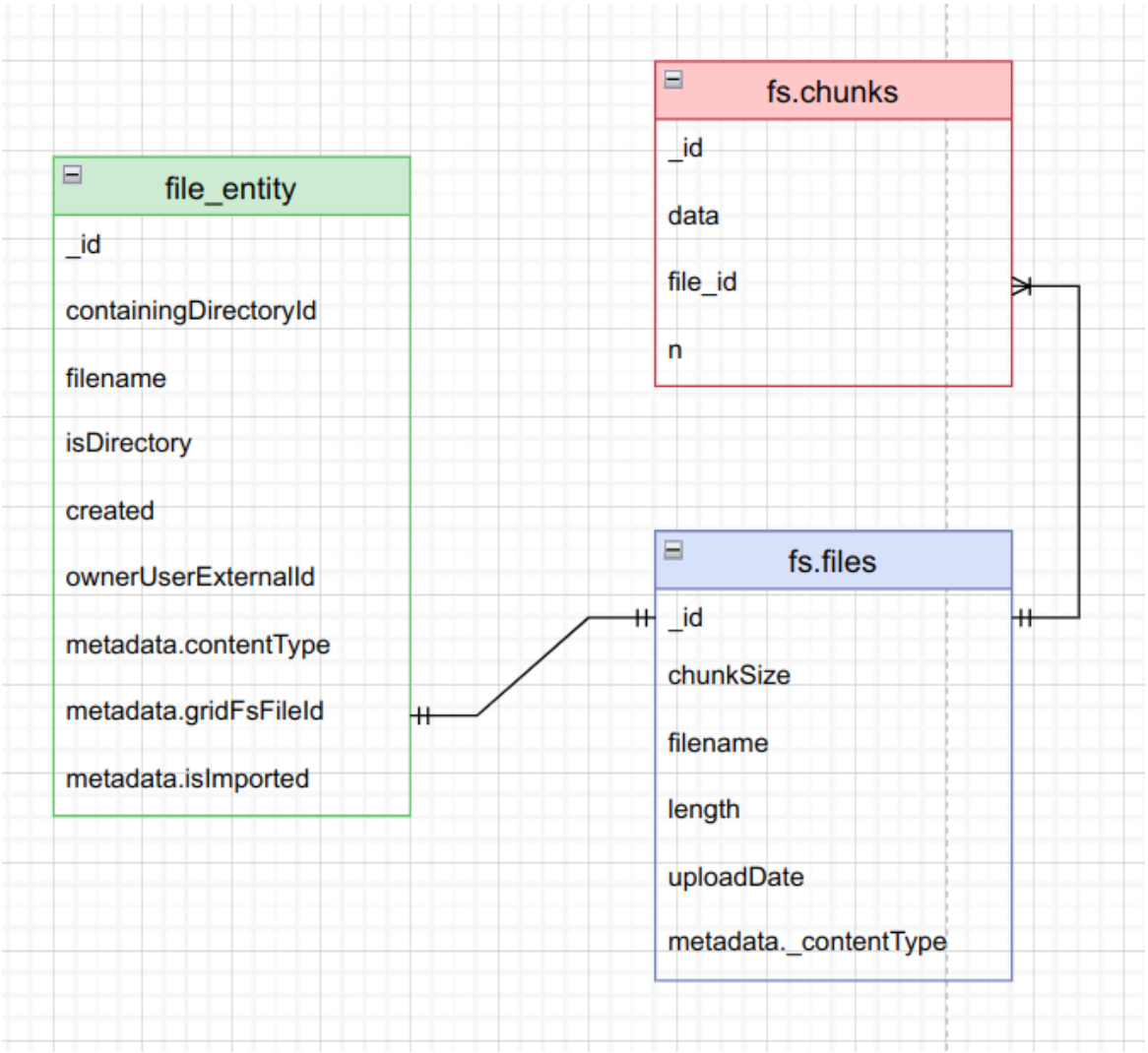


Рисунок 3.3 – ER діаграма сутностей file_entity, fs.files, fs.chunks

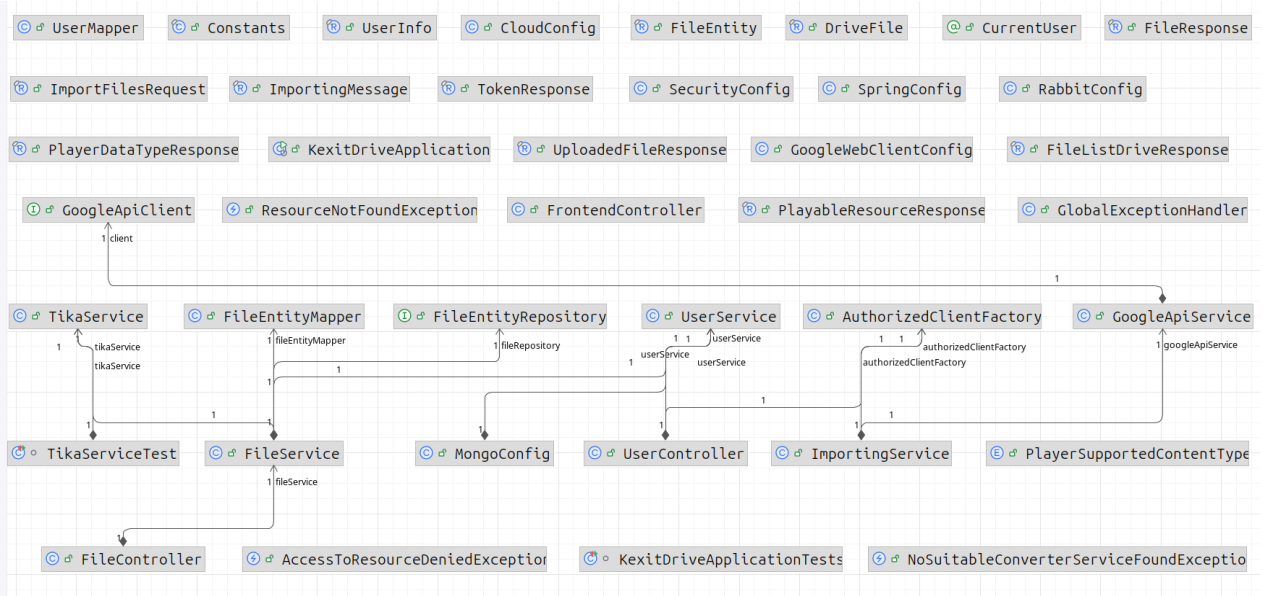


Рисунок 3.4 – Структурна схема класів

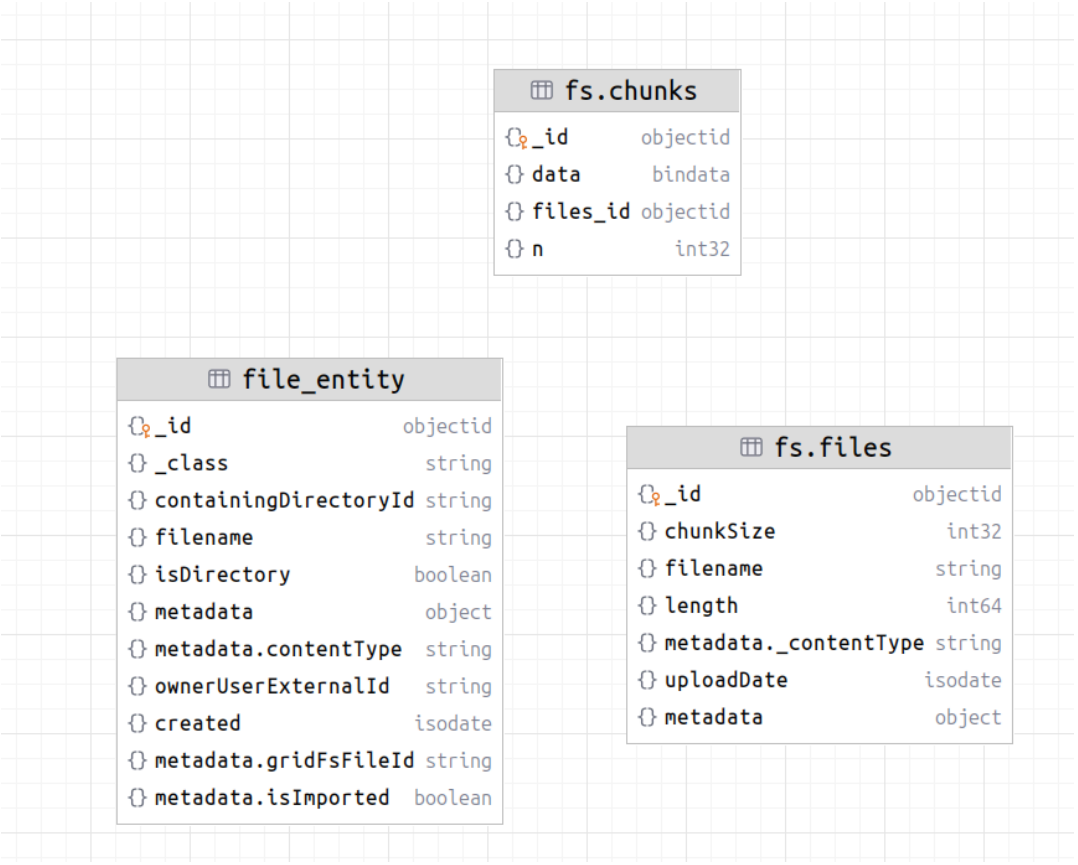


Рисунок 3.5 – Модель бази даних

На рисунку 3.4 зображено структурну схему класів, що є основними складовими програмного забезпечення. Також на рисунку 3.5 зображено модель базу даних, згенеровану засобами IDE.

Таблиця 3.1 – Опис таблиці FileEntity

Назва поля	Тип даних	Опис
_id	ObjectId	ідентифікаційний номер
_class	String	назва java класу, що описую дану сутність
containingDirectoryId	String	ідентифікатор директорії, що містить даний файл
filename	String	назва файлу
isDirectory	Boolean	прапорець, що ідентифікує дану сутність як файл чи папку
ownerUserExternalId	String	ідентифікатор користувача, який володіє даним файлом
created	ISODate	дата створення файлу
metadata.contentType	String	тип файлу
metadata.gridFsFileId	String	ідентифікатор файлу в сховищі GridFS
metadata.isImported	Boolean	прапорець, що показує, був даний файл завантажений чи імпортований з Google Drive

Таблиця 3.2 – Опис таблиці fs.chunks

Назва поля	Тип даних	Опис
_id	ObjectId	ідентифікаційний номер

Продовження таблиці 3.2

data	BinData	вміст частини файлу (чанку) у вигляді списку байтів
files_id	ObjectId	ідентифікатор файлу із таблиці fs.files
n	int32	порядковий номер чанку для даного файлу

Таблиця 3.3 – Опис таблиці fs.files

Назва поля	Тип даних	Опис
_id	ObjectId	ідентифікаційний номер
chunkSize	int32	розмір чанку
filename	String	назва файлу
length	int64	розмір файлу у байтах
uploadDate	ISODate	дата завантаження до сховища
metadata._contentType	String	тип файлу

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.22.

Таблиця 3.4 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	IntelliJ IDEA [47]	Головне середовище розробки програмного забезпечення серверної частини курсової роботи.
2	Visual Studio Code [48]	Головне середовище розробки програмного забезпечення клієнтської частини курсової роботи.

Продовження таблиці 3.4

3	Postman [49]	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
4	Mongo Compass [50]	Програмне забезпечення яке надає простий графічний інтерфейс для доступу до бази даних.

Таблиця 3.5 – Опис бібліотек для клієнтської частини

№ п/п	Назва утиліти	Опис бібліотеки
1	sveltejs/kit	Бібліотека, яка є основою фреймворка Svelte Kit
2	svelte	Бібліотека, яка є основою компілятора Svelte.
3	pdfjs-viewer-element	Допомагає відображати та маніпулювати PDF-файлами у веб-додатках.
4	flowbite-svelte	Надає компоненти та стилі для створення зручного та естетичного користувацького інтерфейсу у веб-додатках.

Таблиця 3.6 – Опис бібліотек для серверної частини

№ п/п	Назва утиліти	Опис бібліотеки
1	Spring Boot	Бібліотека, яка надає основний функціонал для роботи фреймворку Spring Boot.

Продовження таблиці 3.6

2	Spring Boot Starter Mongo	Модуль Spring Boot, який надає можливість роботи із СКБД MongoDB.
3	Spring Boot Starter AMQP	Модуль Spring Boot, який надає можливість роботи із брокером повідомлень RabbitMQ.
4	Spring Boot Starter Security	Модуль Spring Boot, який надає можливість налаштування авторизації у застосунку.
5	Spring Cloud	Модуль Spring Boot, який надає можливість інтеграції застосунків у кластері.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.5 Аналіз безпеки даних

У даному застосунку використовується OAuth2 [10] та OIDC (OpenID Connect [11]) для забезпечення безпечної аутентифікації та авторизації користувачів. OAuth 2 є стандартним протоколом для авторизації, який дозволяє програмам отримувати обмежений доступ до ресурсів користувача без необхідності розкриття їх пароля. OIDC доповнює OAuth 2, забезпечуючи можливість аутентифікації користувачів і отримання основної інформації про них через стандартизований протокол. Це дозволяє застосунку гарантувати безпечний і надійний спосіб управління доступом та ідентифікацією користувачів. Також використання протоколу OAuth2 [10] для авторизації надає можливість доступу до хмарного файлового сховища Google Drive.

Висновки до розділу

У даному розділі було детально розглянуто процес конструювання та розроблення програмного забезпечення для файлового сховища з можливістю онлайн-перегляду файлів будь-яких типів. Розробка включала визначення архітектури програмного забезпечення, обґрунтування вибору засобів розробки, безпосереднє конструювання та аналіз безпеки даних.

Архітектура програмного забезпечення: вибір архітектури ґрунтувався на сервіс-орієнтованій архітектурі (SOA) завдяки її масштабованості, гнучкості та ефективності. Було визначено, що ця архітектура дозволяє легко додавати нові сервіси-конвертори без потреби вносити зміни до існуючих сервісів, що забезпечує високу адаптивність та модульність системи. Основними компонентами системи стали Api Gateway, Eureka Server, Kexit Client, Mongo, RabbitMQ та Kexit Core, які забезпечують ефективне функціонування та взаємодію між різними частинами програмного забезпечення.

Вибір інструментів розробки був зумовлений потребою забезпечення високої продуктивності, масштабованості та безпеки даних. Було використано Spring Cloud для інтеграції застосунків у кластері, що дозволяє ефективно керувати навантаженням та забезпечувати безперервність сервісів.

Особлива увага приділяється забезпеченню можливості перегляду файлів онлайн у браузері, що є важливою функцією для користувачів.

Таким чином, у даному розділі було успішно розроблено архітектурне рішення для файлового сховища, що відповідає сучасним вимогам до зручності, швидкості та безпеки. Це рішення забезпечує можливість ефективного управління файлами та перегляду їх онлайн, що робить його конкурентоспроможним на ринку хмарних сховищ.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метриками для оцінки якості ПЗ обрано наступні:

– відсоток дуплікування коду.

– безпека коду;

– підтримуваність коду;

– надійність коду;

Підтримуваність, масштабованість та низький відсоток повторюваності коду є важливими чинниками, що спростять підтримку проекту та його подальшого розвитку.

Для перевірки відповідності коду цим метрикам було обрано статичний хмарний аналізатор коду SonarCloud [8]. Звіт щодо результатів аналізу вказаний на рисунку 4.1.

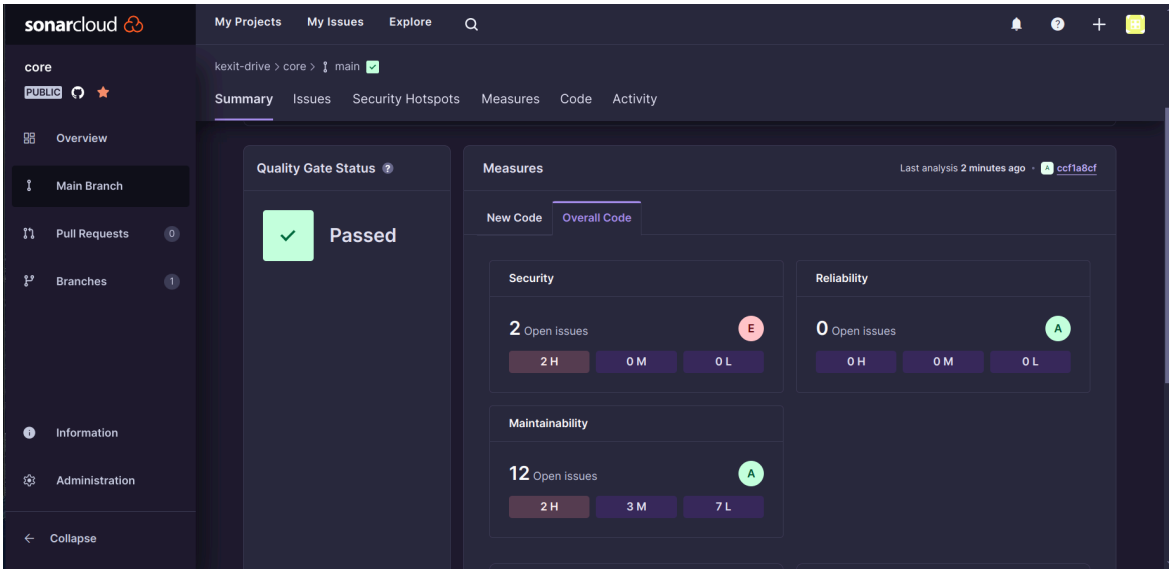


Рисунок 4.1 – Звіт щодо аналізу коду з SonarCloud [8]

Отже, код цього дипломного проекту відповідає необхідному рівню якості за вищевказаними метриками.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.9.

Таблиця 4.1 – Тест 1.1 Вхід до системи, позитивний тест

Початковий стан системи	-
Вхідні дані	-
Опис проведення тесту	Користувач переходить на авторизаційну сторінку та натискає на кнопку «Login with Google». Далі переходить на сторінку вибору Google акаунта, обравши його вводить логін і пароль (за потреби), погоджується з використанням особистих даних.
Очікуваний результат	Відбувається успішний вхід до системи та відкривається головна сторінка із кореневою директорією.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Вхід до системи, негативний тест

Початковий стан системи	-
Вхідні дані	-
Опис проведення тесту	Користувач переходить на авторизаційну сторінку та натискає на кнопку «Login with Google». Далі переходить на сторінку вибору Google акаунта, обравши його вводить логін і пароль, НЕ погоджується з використанням особистих даних.
Очікуваний результат	Відбувається перехід на сторінку логіну.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Перегляд власного профіля користувача

Початковий стан системи	Користувач авторизований
Вхідні дані	-
Опис проведення тесту	Користувач натискає вкладку Profile
Очікуваний результат	Відбувається перехід на сторінку профіля користувача і користувач бачить усі дані свого профіля Google.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Вихід із системи

Початковий стан системи	Користувач авторизований
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку Logout
Очікуваний результат	Відбувається перехід на сторінку логіна та користувач виходить із системи.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Імпорт файлу з Google Drive

Початковий стан системи	Користувач авторизований
Вхідні дані	-

Продовження таблиці 4.5

Опис проведення тесту	Користувач бажає імпортувати файл із Google Drive у поточну директорію. Користувач авторизований та переглядає сторінку певної директорії. Користувач натискає кнопку «Import». В модальному вікні, що з’явилося, відображається вміст його Google Drive сховища, де він може обрати потрібні файли та натиснувши “Select”імпортувати їх до поточної директорії
Очікуваний результат	Обрані файли імпортуються до поточної директорії.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Завантаження файла з комп’ютера

Початковий стан системи	Користувач авторизований
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку “Upload”, обрає файл, який хоче завантажити та натиає “Select” у модальному вікні, що відкривається
Очікуваний результат	До поточної директорії завантажується обраний файл.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Завантаження файла з комп’ютера

Початковий стан системи	Користувач авторизований
Вхідні дані	-

Продовження таблиці 4.7

Опис проведення тесту	Користувач натискає кнопку “Upload”, обрає файл, який хоче завантажити та натиає “Select” у модальному вікні, що відкривається
Очікуваний результат	До поточної директорії завантажується обраний файл.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Видалення файла з комп’ютера

Початковий стан системи	Користувач авторизований
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку «Delete» на директорії/файлі. У відкритому модальному вікні він настикає “Delete”.
Очікуваний результат	До поточної директорії завантажується обраний файл.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Програвання файла онлайн

Початковий стан системи	Користувач авторизований
Вхідні дані	-
Опис проведення тесту	Користувач двічі натискає на обраний файл.
Очікуваний результат	Відкривається плеєр та програє файл онлайн.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

У якості клієнту під час виконання контрольного прикладу був використаний Postman [28] і браузер Google. Для виконання успішної перевірки контрольного прикладу необхідно виконати такі кроки:

– крок 1: перейти на сторінку логіну;

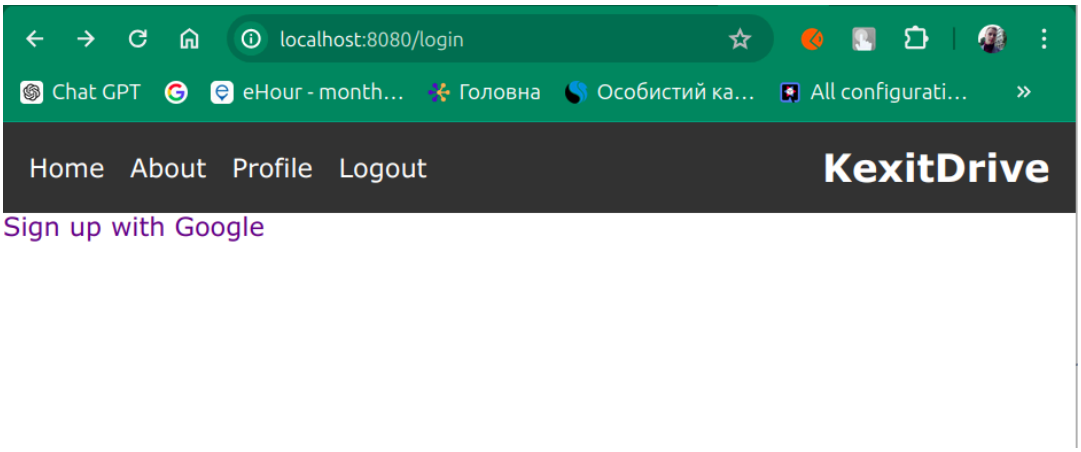


Рисунок 4.2 – Сторінка логіну

– крок 2: натиснути кнопку “Sign Up With Google”;

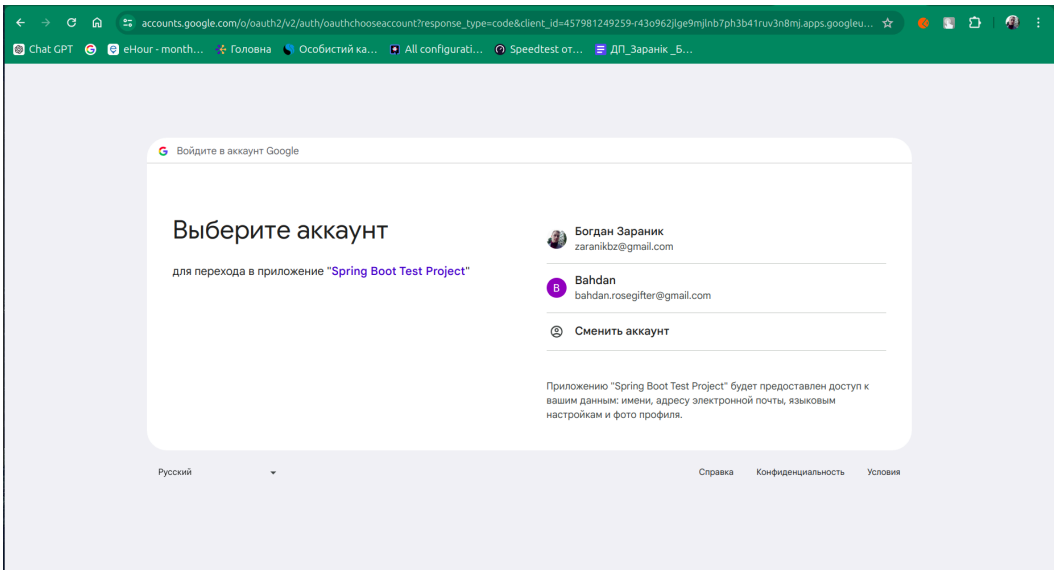


Рисунок 4.3 – Сторінка вибору акаунта

– крок 3: обрати акаунт та погодитись із наданням застосунку прав доступу до Google акаунта користувача;

– крок 4: автоматичний перехід на сторінку кореневого каталогу;

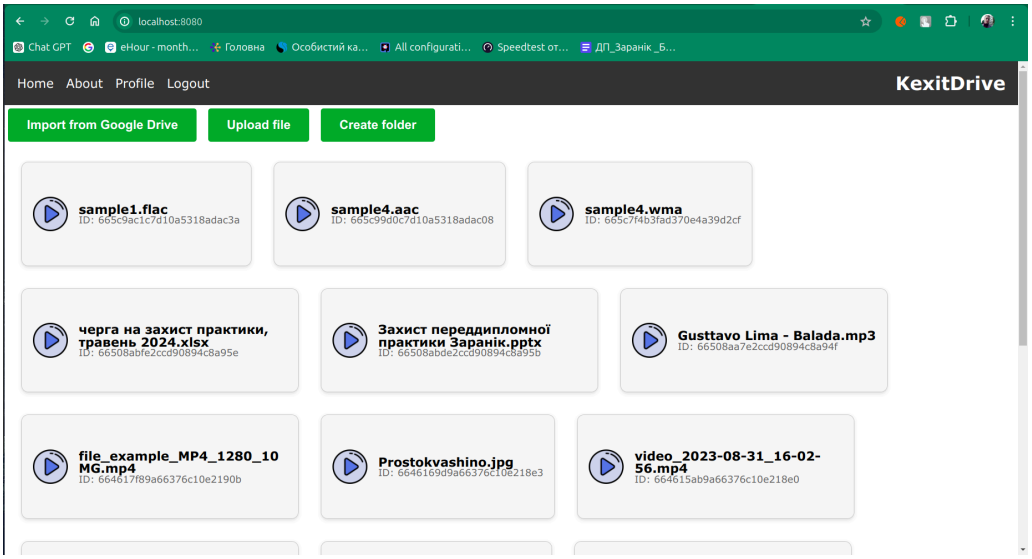


Рисунок 4.4 – Сторінка кореневого каталогу

– крок 5: натиснути кнопку “Import from Google Drive”;

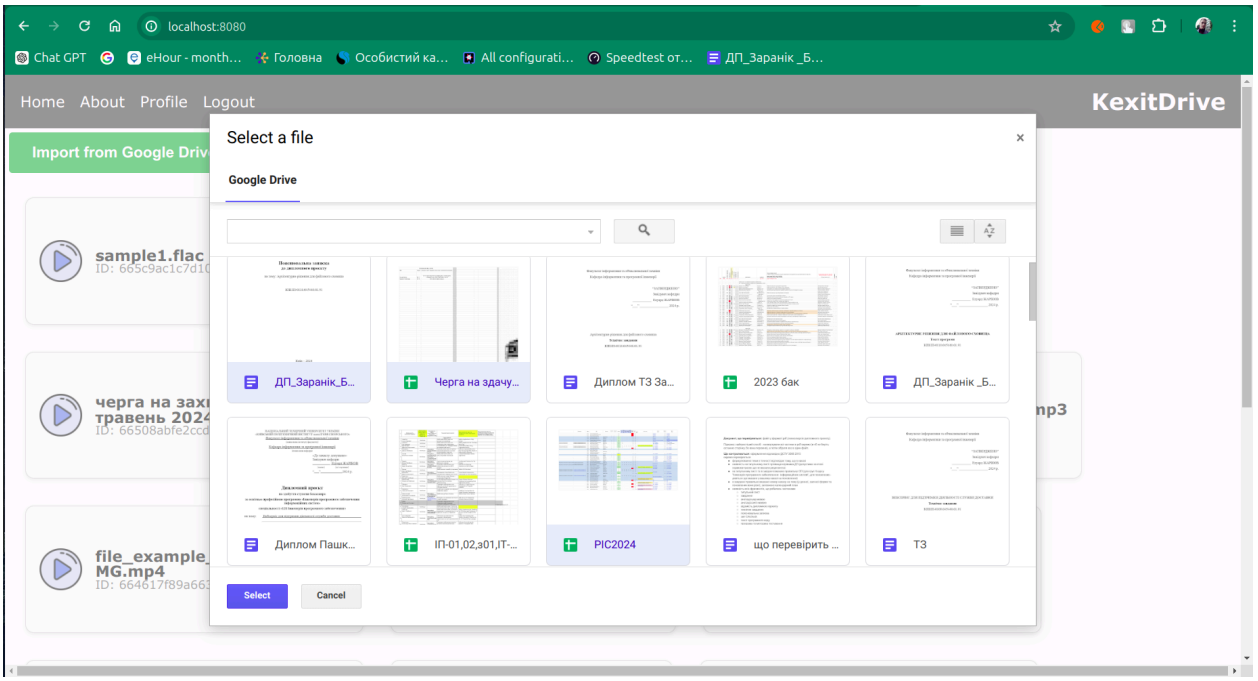


Рисунок 4.5 – Модальне вікно вибору файлів з Google Drive

– крок 6: у відкритому модальному вікні обрати будь-які файли та натиснути кнопку “Select”

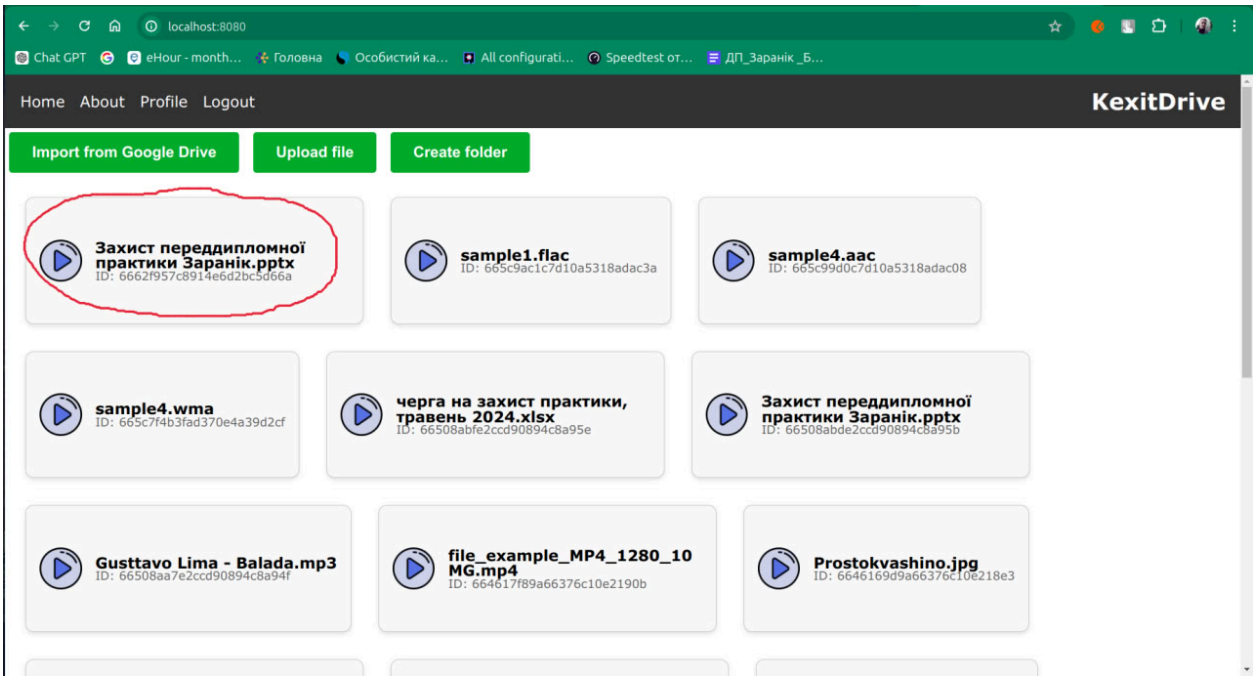


Рисунок 4.6 – Файл імпортовано з Google Drive до поточного каталогу

– крок 7: натиснути кнопку “Upload file”;

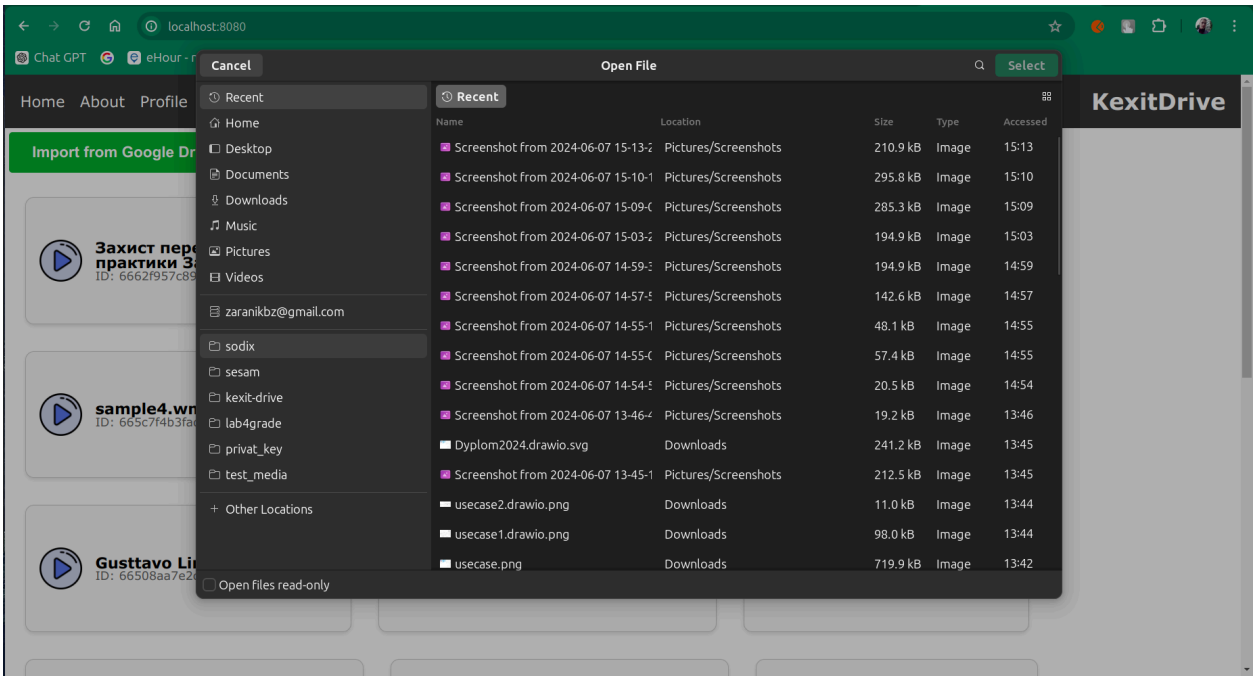


Рисунок 4.7 – Вікно файлового провідника

– крок 8: у відкритому вікні файлового провідника обрати файл та завантажити його до системи;

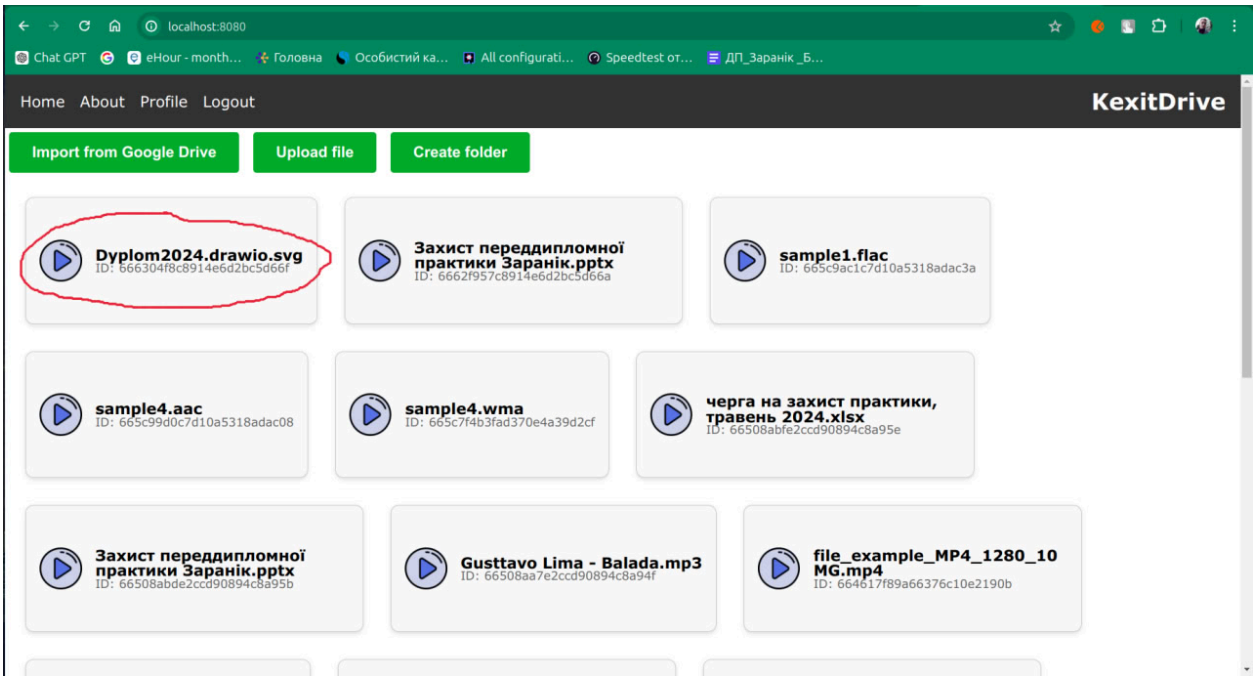


Рисунок 4.8 – Файл завантажено з локального сховища до поточного каталогу

– крок 9: натиснути на кнопку плеера на файлі (або двічі клацнути на файл), відкриється плеєр і файл можна програти онлайн;

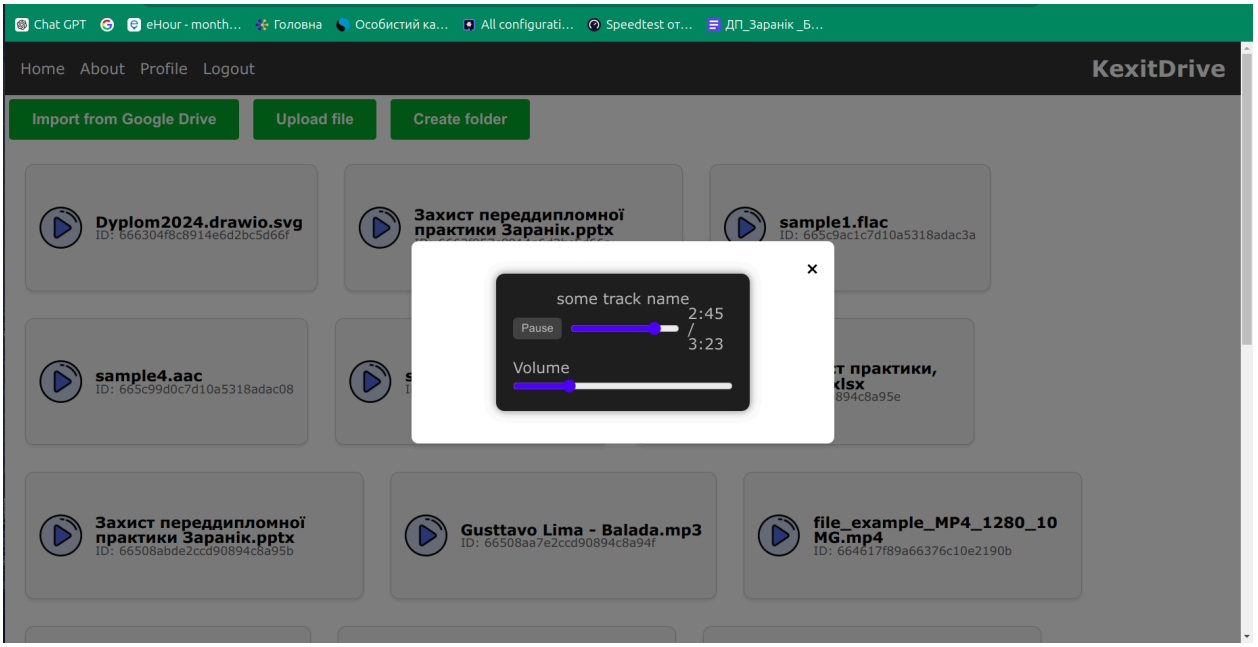


Рисунок 4.9 – Модальне вікно з плеєром, який відкриває файл даного типу

– крок 10: натиснути кнопку “Create folder” та у відкритому модальному вікні ввести назву нової папки;

– крок 11: натиснути вкладку “Profile” - відкривається сторінка із особистою інформацією користувача;

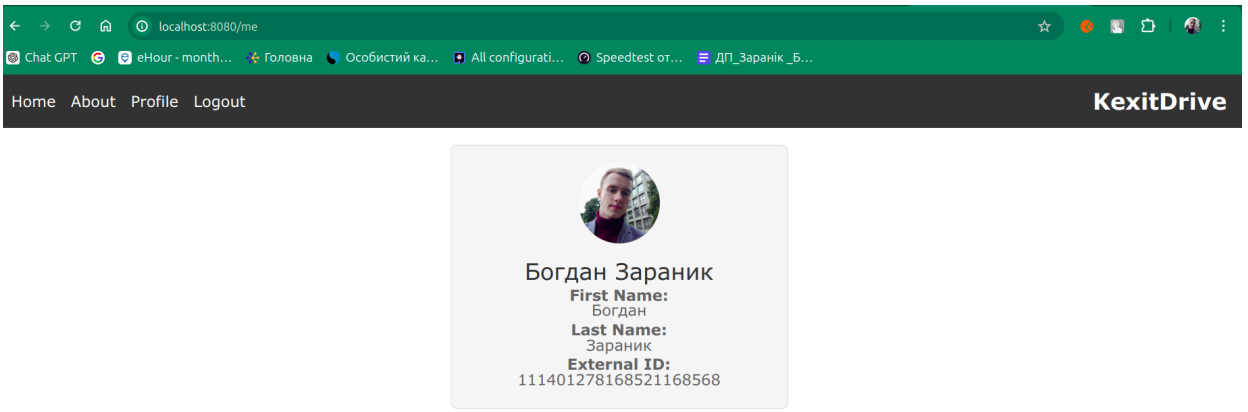


Рисунок 4.10 – Сторінка із особистою інформацією користувача

– крок 12: натиснути на кнопку “Logout” та вийти з системи, користувача буде направлено на сторінку логіну;

Отже, цей контрольний приклад показує, що отримане програмне забезпечення працює у відповідності до функціональних вимог, визначених раніше.

Висновки до розділу

У рамках цього розділу описані процеси тестування програмного забезпечення. У ньому були наведені конкретні приклади мануальних тестів, проведених у рамках тестування. За результатами тестування встановлено, що фактичний результат усіх тестів відповідає очікуваному.

У тестовому прикладі, детально описаному в останньому підрозділі, було покрито значну частину функціональних вимог до кінцевого програмного забезпечення.

Мета тестування та показники якості програмного забезпечення були представлені у початковій частині цього розділу. Отриманий результат показав, що програмне забезпечення задовольняє вимогам рівня якості, які висунув аналізатор за вказаними метриками (безпека, надійність, ремонтопридатність, відсоток дублювання коду). Отриманий

код був проаналізований за допомогою SonarCloud - аналізатора коду - на основі цих показників.

Дуже докладно та охоплюючи значну частину функціональних вимог отриманого програмного забезпечення, тестовий приклад був описаний в останньому підрозділі.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання даного програмного забезпечення було обрано технологію Docker [29].

Дана технологія надає можливість більш зручного розгортання програмного забезпечення завдяки процесу контейнеризації. Для створення контейнера, що містить програмний продукт, необхідно створити Docker-образ, на основі якого і буде створено контейнер. Контейнер можна створити на будь-якому пристрої під управлінням будь-якої операційної системи, на якому встановлено Docker.

Таким чином, розроблений застосунок буде легко розгорнути у будь-якому середовищі та на будь-якій хостинг платформі: на віртуальній машині AWS EC2 [30], Vercel [31], Heroku [32] чи з використанням інших подібних хмарних технологій.

Загалом, значних відмінностей між місцем розгортання Docker-контейнеру немає, тому розробникам потрібно лише правильно описати Dockerfile – інструкцію щодо того, як має відбуватись розгортання додатку та якими властивостями володіє віртуалізоване середовище, в якому він запускається.

```
FROM node:20.11-alpine AS base
ENV PNPM_HOME="/pnpm"
ENV PATH="$PNPM_HOME:$PATH"
RUN corepack enable
COPY . /app
WORKDIR /app

# also has dev dependencies
FROM base AS build
ARG BUILD_MODE
ENV VITE_BUILD_MODE=$BUILD_MODE
RUN --mount=type=cache,id=pnpm,target=/pnpm/store pnpm install --no-frozen-lockfile
RUN pnpm run build

FROM nginx:1.16-alpine
COPY ./nginx/config/default.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Рисунок 5.1 – Файл Dockerfile для клієнтської частини застосунку

З метою спрощення розгортання застосунку було використано технологію Docker Compose [36]. Дана технологія надає можливість шаблонізованого опису налаштувань Docker-контейнерів та взаємозв’язків між ними.

```yaml
version: "3.9"

name: kexit

services:

  eureka:
    image: zaranik/kexit-drive-eureka:latest
    restart: always
    ports:
      - "8761:8761"
    profiles:
      - prod

  gateway:
    image: zaranik/kexit-drive-gateway:latest
    restart: always
    ports:
      - "80:80"
    profiles:
      - prod
    depends_on:
      eureka:
        condition: service_healthy

  core:
    image: zaranik/kexit-drive-core:latest
    restart: always
    ports:
      - "8080:8080"
    profiles:
      - prod
    depends_on:
      mongo:
        condition: service_healthy
      eureka:
        condition: service_healthy
```

Рисунок 5.2 – Файл docker-compose.yml

```yaml
mongo:
  image: mongo:6.0.3
  restart: always
  ports:
    - "27017:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: password
  volumes:
    - ./data/mongo:/data/db
  healthcheck:
    test: echo 'db.runCommand("ping").ok' | mongosh localhost:27017/test --quiet
    interval: 10s
    timeout: 5s
    retries: 5

rabbitmq:
  image: rabbitmq:3.9.7-management
  restart: always
  environment:
    RABBITMQ_DEFAULT_USER: admin
    RABBITMQ_DEFAULT_PASS: password
  ports:
    - "5672:5672"
    - "15672:15672"
  volumes:
    - ./data/rabbit:/var/lib/rabbitmq

gotenberg:
  image: gotenberg/gotenberg:8
  restart: always
  ports:
    - "3000:3000"
```

Рисунок 5.3 – Файл docker-compose.yml

Для розгортання додатку, у кореневій папці проекту, що містить файл docker-compose.yaml, потрібно виконати команду docker-compose up, що створить всі контейнери та розгорне додаток. Для створення контейнерів усі необхідні образи повинні бути завчасно зібрані. Процес розгортання додатку показаний на рисунку 5.4.

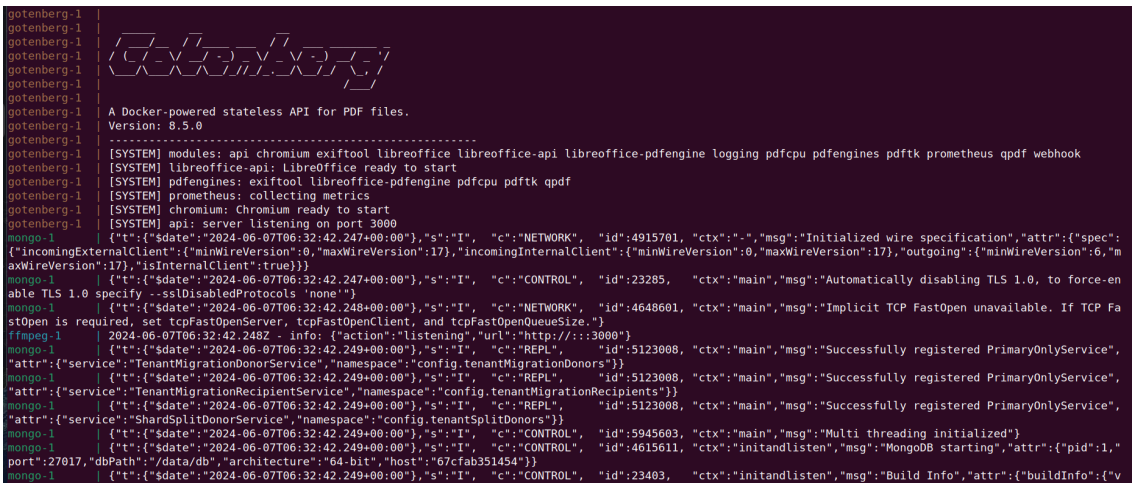


Рисунок 5.4 – Логування процесу розгортання додатку

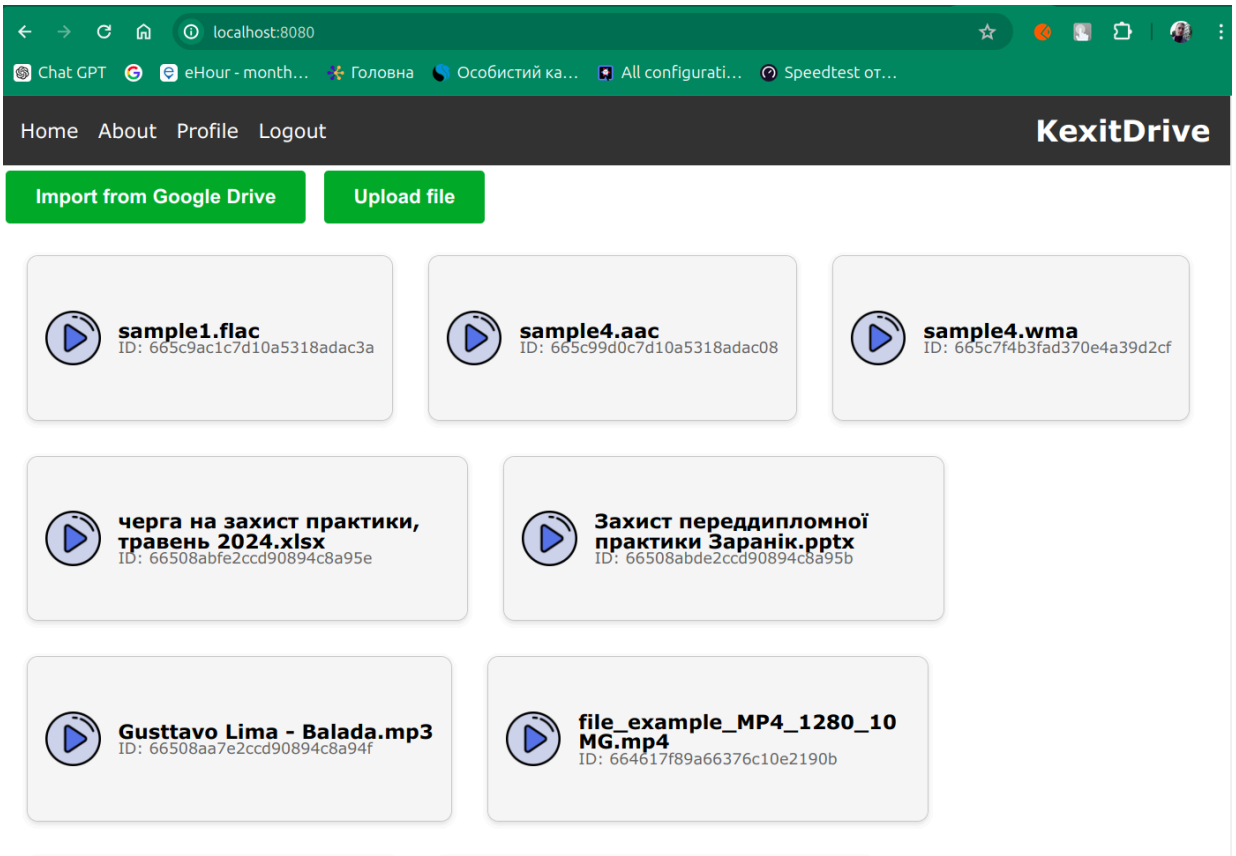


Рисунок 5.5 – Перевірка доступу до застосунку після розгортання

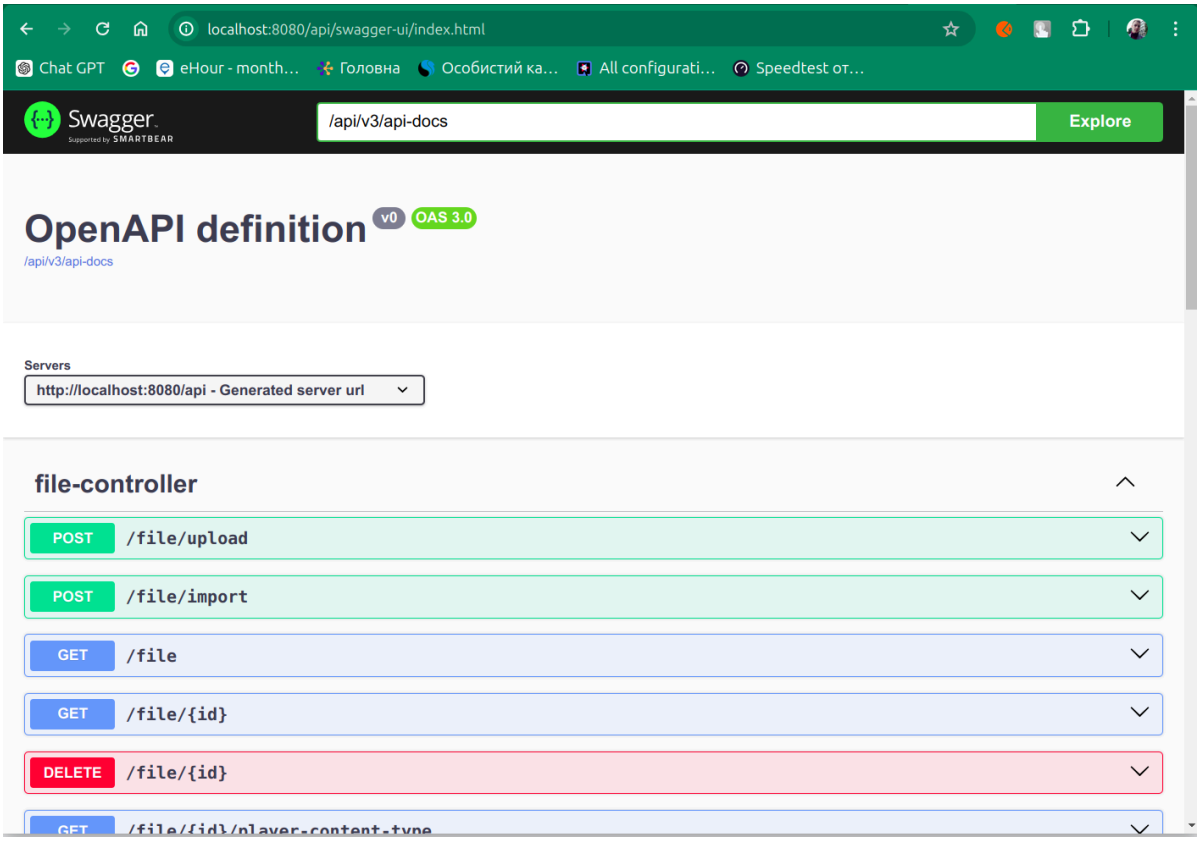


Рисунок 5.6 – Перевірка доступу до документації застосунку після розгортання

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Задля доставки оновлень програмного забезпечення необхідними кроками є створення оновлених образів програмного забезпечення та створення нових контейнерів з подальшим перезапуском усього додатку для того, щоби внесені зміни були застосовані. Для цього необхідно виконати команду docker build для кожного зміненого застосунку, яка створить новий образ, docker compose down для вимкнення застосунку та docker compose up для запуску застосунку заново.

У результаті даних маніпуляцій оновлена версія застосунку стане доступна. Таке оновлення контейнерів можна робити за допомогою сервісу Github Actions [34] або Gitlab CI [35] в автоматичному режимі, наприклад, під час додавання змін у головну гілку репозиторію.

Висновки до розділу

В цьому розділі розглядаються процеси, що стосуються розгортання та супроводу програмного забезпечення.

Перший підрозділ присвячений розгортанню програмного забезпечення. У ньому детально описані технології та інструменти, що використовуються для розгортання, їх особливості. Продемонстровано вміст файлів, які визначають правила розгортання Docker-контейнерів, а також сам процес розгортання додатку та перевірку його доступності. Заключною частиною цього підрозділу є діаграма розгортання, яка наочно показує фізичне розташування та стан компонентів системи після завершення розгортання.

Фокус уваги останнього підрозділу зосереджено на супроводі програмного забезпечення, де детально описано процес доставки оновлень коду системи.

Таким чином, результатом роботи над цим розділом стало визначення технологій, інструментів та процесів, що використовуються при розгортанні та супроводі розробленого програмного забезпечення.

ВИСНОВКИ

Метою розробки є спрощення процесу додавання нових підтримуваних форматів файлів, перегляд яких раніше був недоступний для програвання онлайн. Мета дипломного проектування була досягнута, за рахунок вирішення усіх функціональних задач.

Реалізовано функціонал аутентифікації користувачів за допомогою облікового запису Google. Це забезпечує зручний і безпечний спосіб входу в систему, використовуючи існуючі облікові дані.

Реалізація інтеграції з Google Drive дозволяє користувачам імпортувати файли безпосередньо з їхнього сховища Google. Це полегшує доступ до файлів, збережених у хмарі.

Застосунок підтримує програвання різних типів файлів онлайн, включаючи документи, зображення, відео та аудіо. Це дозволяє користувачам переглядати їхні файли без необхідності завантаження на пристрій

Реалізоване архітектурне рішення дозволяє легко додавати підтримку нових форматів файлів, що забезпечує гнучкість та масштабованість системи. Це здійснюється завдяки модульному додаванню нових сервісів-конвертерів або програвачів для нових форматів.

Усі функціональні задачі, що були поставлені для цього проекту, у повному обсязі були виконані. Застосунок забезпечує можливість імпорту файлів зі сховища Google Drive, завантаження з власної машини, програвання онлайн та скачування завантажених файлів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) RFC 9110: HTTP semantics. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc9110 (дата звернення: 03.06.2024).

2) HTML Standard. The Web Hypertext Application Technology Working Group. URL: https://html.spec.whatwg.org (дата звернення: 03.06.2024).

3) Cascading Style Sheets | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 03.06.2024).

4) JavaScript | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 03.06.2024).

5) MongoDB: The Developer Data Platform. MongoDB. URL: https://www.mongodb.com (дата звернення: 03.06.2024).

6) API - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/API (дата звернення: 04.06.2024).

7) Spring Documentation. https://docs.spring.io/ (дата звернення: 03.06.2024)

8) Плюси і мінуси HTTP Live Streaming. Facesacast. URL: https://facecast.net/ru/blog/http-live-streaming/ (дата звернення: 20.03.2024);

9) What Is MPEG-DASH? And MPEG-DASH vs. HLS. Cloudinary. URL:https://cloudinary.com/guides/video-formats/what-is-mpeg-dash-and-mpeg-dash-vshls (дата звернення: 20.03.2024);

10) OAuth 2.0. URL: https://oauth.net/2/ (дата звернення: 20.03.2024);

11) OpenID Connect простими словами. Habr. URL: https://habr.com/ru/companies/nixys/articles/566910/ (дата звернення: 20.03.2024);

12) Heckler, M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications. Київ : O'Reilly Media, 2021. 328 с.

13) Spilca, L. Spring Security in Action. Київ : Manning Publications, 2020. 560 с.

14) Spring Framework Documentation. URL: https://docs.spring.io/springframework/docs/ (дата звернення 24.05.2024).

15) Spring Boot. URL: https://spring.io/projects/spring-boot (дата звернення 25.05.2024).

16) Spring Security. URL: https://spring.io/projects/spring-security (дата звернення 30.05.2024).

17) Maven official site. URL: https://maven.apache.org/download.cgi (дата звернення 19.05.2024)

18) Google Drive. URL: https://drive.google.com/ (date of access: 01.06.2024).

19) BPMN. https://wikipedia.org/. URL: https://ru.wikipedia.org/wiki/BPMN (date of access: 01.06.2024).

20) C4 Diagram. URL: https://c4model.com/ (date of access: 02.06.2024).

21) RabbitMQ documentation. URL: https://www.rabbitmq.com/docs (date of access: 04.06.2024).

22) What is SOA? - SOA Architecture Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/en/what-is/service-oriented-architecture/#:~:text=you%20implement%20microservices?-,What%20is%20service-oriented%20architecture?,other%20across%20platforms%20and%20languages . (date of access: 08.06.2024).

23) Vue.js. Vue.js - The Progressive JavaScript Framework | Vue.js. URL: https://vuejs.org/guide/introduction.html (date of access: 08.06.2024).

24) Quick Start – React. React. URL: https://react.dev/learn (date of access: 08.06.2024).

25) SvelteKit • Web development, streamlined. SvelteKit • Web development, streamlined. URL: https://kit.svelte.dev/ (date of access: 08.06.2024).

26) ПО Java. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/cis/java/ (дата звернення: 08.06.2024).

27) GridFS - MongoDB Manual v7.0. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/core/gridfs/ (date of access: 08.06.2024).

28) Postman. URL: https://www.postman.com/ (date of access: 08.06.2024).

29) Docker: Accelerated Container Application Development. Docker. URL: https://www.docker.com/ (date of access: 08.06.2024).

30) Amazon EC2 - Cloud Compute Capacity - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/ec2/?nc1=h_ls (date of access: 08.06.2024).

31) Vercel: Build and deploy the best web experiences with the Frontend Cloud – Vercel. Vercel. URL: https://vercel.com/ (date of access: 08.06.2024).

32) Cloud Application Platform | Heroku. Cloud Application Platform | Heroku. URL: https://www.heroku.com/ (date of access: 08.06.2024).

33) dropbox.com. Dropbox.com. URL: https://www.dropbox.com/uk_UA/ (date of access: 08.06.2024).

34) Features • GitHub Actions. GitHub. URL: https://github.com/features/actions (date of access: 08.06.2024).

35) Get started with GitLab CI/CD | GitLab. GitLab Documentation. URL: https://docs.gitlab.com/ee/ci (date of access: 08.06.2024).

36) Docker Compose overview. Docker Documentation. URL: https://docs.docker.com/compose (date of access: 08.06.2024).

37) Google - About Google, Our Culture & Company News. Google - Alles Google, unsere Unternehmenskultur und Neuigkeiten zum Unternehmen. URL: https://about.google/intl/ALL_en/ (date of access: 09.06.2024).

38) Personal Cloud Storage & File Sharing Platform - Google. Personal Cloud Storage & File Sharing Platform - Google. URL: https://drive.google.com/ (date of access: 09.06.2024).

39) Posey B. What is a Server? - Definition from WhatIs.com. WhatIs. URL: https://www.techtarget.com/whatis/definition/server (date of access: 09.06.2024).

40) Client-Server Model - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/client-server-model (date of access: 09.06.2024).

41) Microservices vs. monolithic architecture | Atlassian. Atlassian. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (date of access: 09.06.2024).

42) Clean Coder Blog. Clean Coder Blog. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (date of access: 09.06.2024).

43) Чим відрізняється сайт від веб додатку: технічні аспекти. FoxmindEd. URL: https://foxminded.ua/chym-vidrizniaietsia-sait-vid-veb-dodatku/ (дата звернення: 09.06.2024).

44) Microservices Pattern: Pattern: Server-side service discovery. microservices.io. URL: https://microservices.io/patterns/server-side-discovery.html (date of access: 09.06.2024).

45) Chain of Responsibility. Refactoring and Design Patterns. URL: https://refactoring.guru/design-patterns/chain-of-responsibility (date of access: 09.06.2024).

46) Pulatjonov A. Microservices Design Patterns: API Gateway Design Pattern. Medium. URL: https://medium.com/@apulatjonov/microservices-design-patterns-api-gateway-design-pattern-ddba36700d84 (date of access: 09.06.2024).

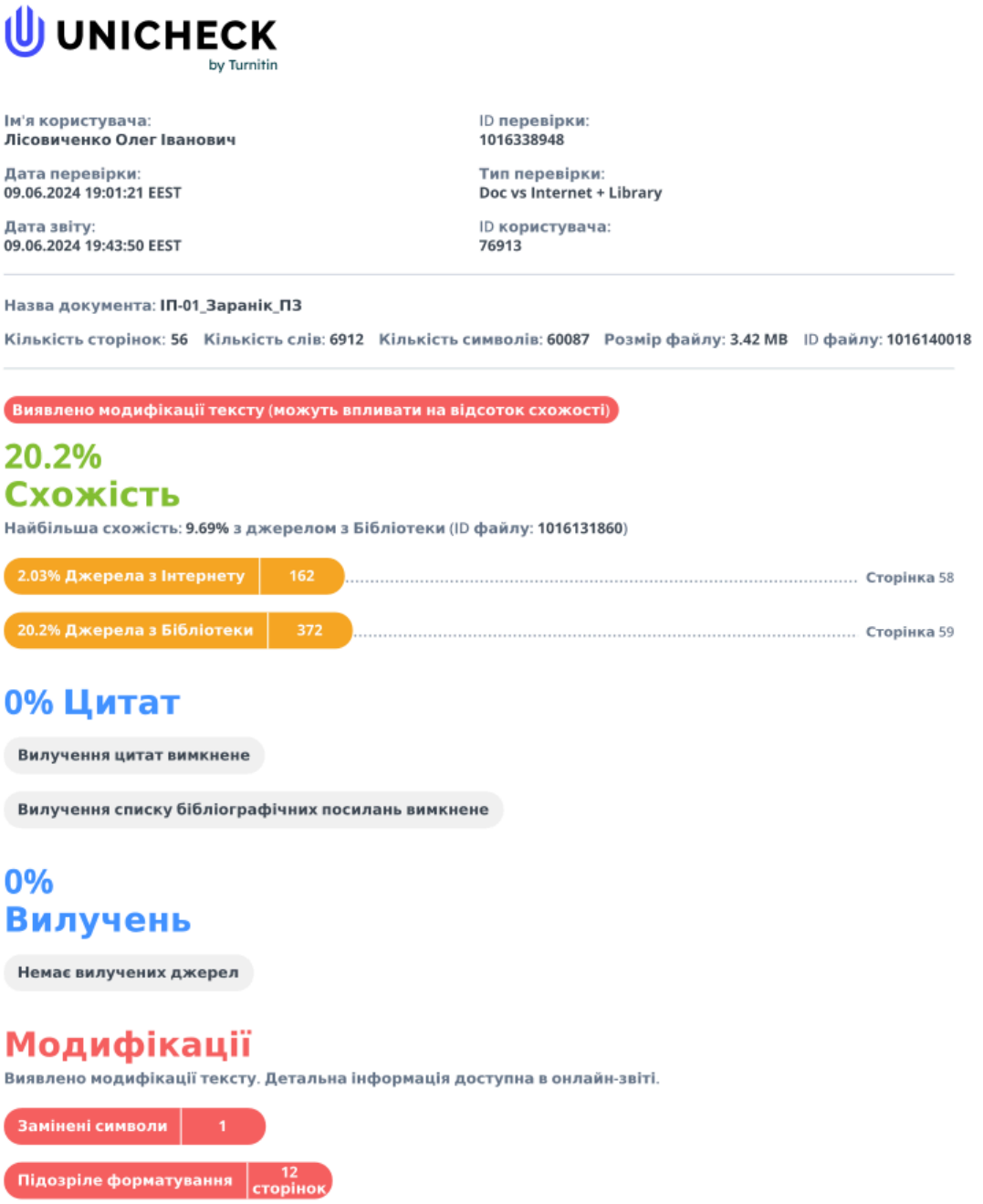
47) JetBrains. IntelliJ IDEA – the Leading Java and Kotlin IDE. JetBrains. URL: https://www.jetbrains.com/idea/ (дата звернення: 09.06.2024).

48) Microsoft. Visual Studio Code - Code Editing. Redefined. Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/ (date of access: 09.06.2024).

49) Postman. URL: https://www.postman.com/ (дата звернення: 09.06.2024).

50) MongoDB Compass. MongoDB. URL: https://www.mongodb.com/products/tools/compass (дата звернення: 09.06.2024).

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016338948

Дата перевірки:
09.06.2024 19:01:21 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 19:43:50 EEST

ID користувача:
76913

Назва документа: ІП-01_Заранік_ПЗ

Кількість сторінок: 56 Кількість слів: 6912 Кількість символів: 60087 Розмір файлу: 3.42 MB ID файлу: 1016140018

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

20.2%
Схожість

Найбільша схожість: 9.69% з джерелом з Бібліотеки (ID файлу: 1016131860)

2.03% Джерела з Інтернету 162 ..... Сторінка 58

20.2% Джерела з Бібліотеки 372 ..... Сторінка 59

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 1

Підозріле форматування 12 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

АРХІТЕКТУРНЕ РІШЕННЯ ДЛЯ ФАЙЛОВОГО СХОВИЩА

Текст програми

КПІ.ІП-0110.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Богдан ЗАРАНІК

Київ – 2024

Посилання на репозиторії з повним текстом програмного коду
https://github.com/kexit-drive/core
https://github.com/kexit-drive/gateway
https://github.com/kexit-drive/converter-audio
https://github.com/kexit-drive/kexit-frontend
https://github.com/kexit-drive/eureka
https://github.com/kexit-drive/platform

Файл AudioPlayer.svelte
Реалізація функціональної задачі «Можливість переглядати будь-які типи файлів онлайн»

```
<script>
 import { onMount, onDestroy } from "svelte";
 let audioFile;
 export let url;
 export let trackName;

 let isPlaying = false;
 let currentTime = "0:00";
 let totalTime = "0:00";
 let progress = 0;
 let volume = 50;

 onMount(() => {
   setupAudio();
 });

 $: if (url) {
   setupAudio();
 }

 function setupAudio() {
   if (audioFile) {
     audioFile.pause();
     audioFile.removeEventListener("timeupdate", updateProgress);
     audioFile.removeEventListener("loadedmetadata", onLoadedMetadata);
   }
   audioFile = new Audio(url);
   audioFile.volume = volume / 100;
   audioFile.addEventListener("timeupdate", updateProgress);
   audioFile.addEventListener("loadedmetadata", onLoadedMetadata);

   if (isPlaying) {
     audioFile.play();
   }
 }

 function onLoadedMetadata() {
   totalTime = formatTime(audioFile.duration);
 }

 function playPause() {
   if (audioFile.paused) {
     audioFile.play();
     isPlaying = true;
   } else {
```

```
    audioFile.pause();
    isPlaying = false;
  }
}

function updateProgress() {
  progress = (audioFile.currentTime / audioFile.duration) * 100;
  currentTime = formatTime(audioFile.currentTime);
}

function formatTime(seconds) {
  const mins = Math.floor(seconds / 60);
  const secs = Math.floor(seconds % 60);
  return `${mins}:${secs < 10 ? "0" + secs : secs}`;
}

function seek(event) {
  const newTime = (event.target.value / 100) * audioFile.duration;
  audioFile.currentTime = newTime;
}

function adjustVolume(event) {
  audioFile.volume = event.target.value / 100;
  volume = event.target.value;
}

onDestroy(() => {
  if (audioFile) {
    audioFile.pause();
    audioFile.removeEventListener("timeupdate", updateProgress);
    audioFile.removeEventListener("loadedmetadata", onLoadedMetadata);
  }
});
</script>

<main>
 <section class="audio-player">
  <h3>{trackName}</h3>
  <div class="controls">
   <button on:click={playPause}>
    {isPlaying ? "Pause" : "Play"}
   </button>
   <input
    type="range"
    value={progress}
    min="0"
    max="100"
    step="1"
    on:input={seek}
   />
   <span>{currentTime} / {totalTime}</span>
  </div>
  <div class="volume-control">
   <label for="volume">Volume</label>
   <input
    type="range"
    id="volume"
    min="0"
    max="100"
    value={volume}
    on:input={adjustVolume}
   />
  </div>
```

```
 </section>
</main>

<style>
 .audio-player {
   display: flex;
   flex-direction: column;
   align-items: center;
   background-color: #222;
   color: #bbb;
   padding: 20px;
   border-radius: 10px;
   box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
   width: 300px;
   margin: 20px auto;
 }

 .controls {
   display: flex;
   align-items: center;
   width: 100%;
 }

 .controls button {
   margin-right: 10px;
   padding: 5px 10px;
   background-color: #444;
   color: #bbb;
   border: none;
   border-radius: 5px;
   cursor: pointer;
 }

 .controls input[type="range"] {
   flex: 1;
   margin-right: 10px;
 }

 .volume-control {
   margin-top: 10px;
   width: 100%;
 }

 .volume-control label {
   display: block;
   margin-bottom: 5px;
 }

 .volume-control input[type="range"] {
   width: 100%;
 }
</style>
```

Файл FileCard.svelte

Реалізація функціональної задачі «Можливість керувати файлами та папками»

```
<script>
 import { fetchService } from "$lib/fetchService";
 import PlayerComponent from "../PlayerComponent.svelte";
 import { PUBLIC_BASE_URL } from "$env/static/public";
```

```
 import Modal from "../modal/Modal.svelte";
 export let title;
 export let fileId;

 let mediaType = "";
 $: url = `${PUBLIC_BASE_URL}/file/${fileId}/play`;

 let isModalOpen = false;

 function openModal() {
   console.log("Modal opened");
   isModalOpen = true;
 }

 function closeModalCallback() {
   console.log("Modal closed");
   isModalOpen = false;
 }

 const playFile = async () => {
   mediaType = await getFileMediaType();
   openModal();
 };

 async function getFileMediaType() {
   const responseJson = await fetchService(
     `/file/${fileId}/player-content-type`,
     "GET"
   );
   return await responseJson.dataType;
 }
</script>

<!-- svelte-ignore a11y-no-static-element-interactions -->
<div class="file-card">
 <!-- svelte-ignore a11y-click-events-have-key-events -->
 <!-- svelte-ignore a11y-no-noninteractive-element-interactions -->
 <img
   on:click={playFile}
   src="https://cdn-icons-png.freepik.com/512/183/183625.png"
   alt="File Icon"
   class="file-icon"
 />
 <div class="file-info">
   <div><span class="file-title">{title}</span></div>
   <div class="file-id">ID: {fileId}</div>
 </div>
</div>

<Modal isOpen={isModalOpen} closeCallback={closeModalCallback}>
 {#if isModalOpen}
   <PlayerComponent {mediaType} {url} />
 {/if}
</Modal>

<style>
 span {
   word-break: break-word;
 }

 .file-card {
   display: flex;
   align-items: center;
```

```
  border: 1px solid #ccc;
  border-radius: 10px;
  padding: 16px;
  background-color: #f9f9f9;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  max-width: 400px;
  height: 150px;
  margin: 8px;
 }

 .file-icon {
  width: 50px;
  height: 50px;
  margin-right: 16px;
  cursor: pointer;
 }

 .file-info {
  flex-grow: 1;
 }

 .file-title {
  font-size: 18px;
  font-weight: bold;
  margin-bottom: 4px;
 }

 .file-id {
  color: #666;
  font-size: 14px;
 }
</style>
```

Файл GoogleDriveImportButton.svelte

Реалізація функціональної задачі «Можливість імпорту файлів із Google Drive»

```
<script>
 import { fetchService } from "$lib/fetchService";
 import { onMount } from "svelte";

 onMount(async () => {
  console.log("onMount");
  await import("https://apis.google.com/js/api.js");
 });

 async function onApiLoad() {
  gapi.load("picker", { callback: await onPickerApiLoad });
 }

 async function getAccessToken() {
  const dto = await fetchService("/user/getAccessToken", "GET");
  const accessToken = dto.token;
  console.log("accessToken = " + accessToken);
  return accessToken;
 }

 async function onPickerApiLoad() {
  let picker = new google.picker.PickerBuilder()
    .addView(google.picker.ViewId.DOCS)
    .setOAuthToken(await getAccessToken())
    .setCallback(pickerCallback)
```

```
    .enableFeature(google.picker.Feature.MULTISELECT_ENABLED)
    .build();
  picker.setVisible(true);
 }

 function pickerCallback(data) {
  if (data.action == google.picker.Action.PICKED) {
   const fileIds = data.docs.map((doc) => doc.id);
   // Handle the selected file
   console.log("Selected File ID:", fileIds);
   importFiles(fileIds);
  }
 }

 function importFiles(fileIds) {
  console.log("importing fetch");

  const myHeaders = new Headers();
  myHeaders.append("Content-Type", "application/json");

  const rawBody = JSON.stringify({
   fileIds,
  });

  console.dir(fileIds);

  const requestOptions = {
   method: "POST",
   body: rawBody,
   headers: myHeaders,
   redirect: "follow",
  };

  return fetchService("/file/import", "GET", requestOptions);
 }
</script>

<button class="import-button" on:click={onApiLoad}>
 Import from Google Drive
</button>

<style>
 .import-button {
  padding: 0.75rem 1.5rem;
  margin: 5px;
  font-size: 1rem;
  font-weight: bold;
  cursor: pointer;
  background-color: #0a951f;
  color: white;
  border: none;
  border-radius: 4px;
  display: inline-block;
  transition: background-color 0.3s ease;
 }

 .import-button:hover {
  background-color: #357ae8;
 }

 .import-button:active {
  background-color: #3367d6;
 }
```

```
</style>
```

Файл ImagePlayer.svelte

Реалізація функціональної задачі «Можливість переглядати будь-які типи файлів онлайн»

```
<script>
 import { onMount, onDestroy } from 'svelte';
 let imageFile;
 export let url;
 export let imageName;

 let loading = true;
 let progress = 0;
 let zoom = 100;

 function setupImage() {
   if (imageFile) {
     imageFile.src = '';
   }
   imageFile = new Image();
   imageFile.src = url;
   imageFile.onloadstart = () => {
     loading = true;
     progress = 0;
   };
   imageFile.onprogress = (event) => {
     if (event.lengthComputable) {
       progress = (event.loaded / event.total) * 100;
     }
   };
   imageFile.onload = () => {
     loading = false;
     progress = 100;
   };
   imageFile.onerror = () => {
     loading = false;
     progress = 0;
   };
 }

 function adjustZoom(event) {
   zoom = event.target.value;
 }

 // Handle URL changes
 $: if (url) {
   setupImage();
 }

 onMount(() => {
   setupImage();
 });

 onDestroy(() => {
   if (imageFile) {
     imageFile.src = '';
   }
 });
</script>
```

```
<main>
 <section class="image-viewer">
  <h3>{imageName}</h3>
  {#if loading}
   <div class="loading">
    Loading... {progress}%
   </div>
  {:else}
   <div class="image-container">
    <img src={url} alt={imageName} style="width: {zoom}%; height: auto;" />
   </div>
  {/if}
  <div class="zoom-control">
   <label for="zoom">Zoom</label>
   <input
    type="range"
    id="zoom"
    min="10"
    max="200"
    value={zoom}
    on:input={adjustZoom}
   />
   <span>{zoom}%</span>
  </div>
 </section>
</main>

<style>
 .image-viewer {
  display: flex;
  flex-direction: column;
  align-items: center;
  background-color: #222;
  color: #bbb;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
  width: 300px;
  margin: 20px auto;
 }

 .loading {
  margin: 20px 0;
 }

 .image-container {
  width: 100%;
  height: auto;
  margin-bottom: 10px;
 }

 .image-container img {
  width: 100%;
  height: auto;
  border-radius: 10px;
 }

 .zoom-control {
  margin-top: 10px;
  width: 100%;
 }
```

```
 .zoom-control label {
   display: block;
   margin-bottom: 5px;
 }

 .zoom-control input[type="range"] {
   width: 100%;
 }
</style>
```

Файл Modal.svelte

Реалізація функціональної задачі «Можливість переглядати будь-які типи файлів онлайн»

```
<script>
 import { createEventDispatcher } from "svelte";

 export let isOpen = false;
 export let closeCallback = () => {};

 $: isActive = isOpen ? 'modal-active' : 'modal-hidden';

 const dispatch = createEventDispatcher();

 function closeModal() {
   dispatch("close");
   closeCallback();
 }
</script>

<div class="{isActive}" on:click={closeModal}>
 <div class="modal-content" on:click|stopPropagation>
   <button class="close-btn" on:click={closeModal}>&times;</button>
   <slot></slot>
 </div>
</div>

<style>
 .modal-active {
   display: flex;
   position: fixed;
   top: 0;
   left: 0;
   width: 100%;
   height: 100%;
   justify-content: center;
   align-items: center;
   background-color: rgba(0, 0, 0, 0.5);
 }

 .modal-hidden {
   display: none;
   position: fixed;
   top: 0;
   left: 0;
   width: 100%;
   height: 100%;
   justify-content: center;
   align-items: center;
   background-color: rgba(0, 0, 0, 0.5);
 }
```

```
 .modal-content {
   background-color: white;
   padding: 1rem;
   border-radius: 8px;
   max-width: 500px;
   width: 100%;
   box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
   position: relative;
 }

 .close-btn {
   position: absolute;
   top: 1rem;
   right: 1rem;
   background: none;
   border: none;
   font-size: 1.5rem;
   cursor: pointer;
 }
</style>
```

Файл PdfPlayerComponent.svelte

Реалізація функціональної задачі «Можливість переглядати будь-які типи файлів онлайн»

```
<script>
 export let sourceUrl;

 import { onMount } from "svelte";

 onMount(async () => {
   console.log("onMount");
   await import("pdfjs-viewer-element");
 });

 $: src = sourceUrl;
 const viewerPath = "/pdfjs-4.2.67-dist";
 const page = "1";
 const locale = "en";
 const zoom = "auto";
 const pagemode = "none";
 const search = null;
 const phrase = true;
 const textLayer = "hover";
</script>

<main>
 <pdfjs-viewer-element
   class="viewer"
   {src}
   viewer-path={viewerPath}
   {locale}
   {page}
   {search}
   {phrase}
   {zoom}
   {pagemode}
   text-layer={textLayer}
 />
</main>
```

```
<style>
 :global(body) {
   margin: 0;
   padding: 0;
 }
 .viewer {
   height: 80vh;
   height: 80dvh;
 }
</style>
```

Файл VideoPlayer.svelte

Реалізація функціональної задачі «Можливість переглядати будь-які типи файлів онлайн»

```
<script>
 import { onMount, onDestroy } from 'svelte';
 let videoFile;
 export let url;
 export let videoName;

 let isPlaying = false;
 let currentTime = "0:00";
 let totalTime = "0:00";
 let progress = 0;
 let volume = 50;

 function setupVideo() {
   if (videoFile) {
     videoFile.pause();
     videoFile.removeEventListener('timeupdate', updateProgress);
     videoFile.removeEventListener('loadedmetadata', onLoadedMetadata);
     videoFile.src = url;
     videoFile.volume = volume / 100;
     videoFile.addEventListener('timeupdate', updateProgress);
     videoFile.addEventListener('loadedmetadata', onLoadedMetadata);

     if (isPlaying) {
       videoFile.play();
     }
   }
 }

 function onLoadedMetadata() {
   totalTime = formatTime(videoFile.duration);
 }

 function playPause() {
   if (videoFile.paused) {
     videoFile.play();
     isPlaying = true;
   } else {
     videoFile.pause();
     isPlaying = false;
   }
 }

 function updateProgress() {
   progress = (videoFile.currentTime / videoFile.duration) * 100;
   currentTime = formatTime(videoFile.currentTime);
 }

 function formatTime(seconds) {
```

```
  const mins = Math.floor(seconds / 60);
  const secs = Math.floor(seconds % 60);
  return `${mins}:${secs < 10 ? "0" + secs : secs}`;
 }

 function seek(event) {
  const newTime = (event.target.value / 100) * videoFile.duration;
  videoFile.currentTime = newTime;
 }

 function adjustVolume(event) {
  videoFile.volume = event.target.value / 100;
  volume = event.target.value;
 }

 // Handle URL changes
 $: if (url) {
  setupVideo();
 }

 onMount(() => {
  // Initialize the video file on mount
  setupVideo();
 });

 onDestroy(() => {
  if (videoFile) {
   videoFile.pause();
   videoFile.removeEventListener('timeupdate', updateProgress);
   videoFile.removeEventListener('loadedmetadata', onLoadedMetadata);
  }
 });
</script>

<main>
 <section class="video-player">
  <h3>{videoName}</h3>
  <div class="video-container">
   <video bind:this={videoFile} controls></video>
  </div>
  <div class="controls">
   <button on:click={playPause}>
    {isPlaying ? "Pause" : "Play"}
   </button>
   <input
    type="range"
    value={progress}
    min="0"
    max="100"
    step="1"
    on:input={seek}
   />
   <span>{currentTime} / {totalTime}</span>
  </div>
  <div class="volume-control">
   <label for="volume">Volume</label>
   <input
    type="range"
    id="volume"
    min="0"
    max="100"
    value={volume}
    on:input={adjustVolume}
```

```
    />
   </div>
  </section>
</main>

<style>
 .video-player {
   display: flex;
   flex-direction: column;
   align-items: center;
   background-color: #222;
   color: #bbb;
   padding: 20px;
   border-radius: 10px;
   box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
   width: 300px;
   margin: 20px auto;
 }

 .video-container {
   width: 100%;
   height: auto;
   margin-bottom: 10px;
 }

 .video-container video {
   width: 100%;
   height: auto;
   border-radius: 10px;
 }

 .controls {
   display: flex;
   align-items: center;
   width: 100%;
 }

 .controls button {
   margin-right: 10px;
   padding: 5px 10px;
   background-color: #444;
   color: #bbb;
   border: none;
   border-radius: 5px;
   cursor: pointer;
 }

 .controls input[type="range"] {
   flex: 1;
   margin-right: 10px;
 }

 .volume-control {
   margin-top: 10px;
   width: 100%;
 }

 .volume-control label {
   display: block;
   margin-bottom: 5px;
 }

 .volume-control input[type="range"] {
```

```
  width: 100%;
 }
</style>
```

Файл PlayerComponent.svelte

Реалізація функціональної задачі «Можливість переглядати будь-які типи файлів онлайн»

```
<script>
 import ImagePlayer from './image/ImagePlayer.svelte';
 import AudioPlayer from "./audio/AudioPlayer.svelte";
 import PdfPlayerComponent from "./pdf/PdfPlayerComponent.svelte";
 import VideoPlayer from "./video/VideoPlayer.svelte";

 export let mediaType;
 export let url;
</script>

{#if mediaType === "VIDEO"}
 <VideoPlayer {url} videoName = {"some video name"}/>
{:else if mediaType === "AUDIO"}
 <AudioPlayer url={url} trackName={"some track name"}/>
{:else if mediaType === "IMAGE"}
 <ImagePlayer {url} imageName = "some image name"/>
{:else if mediaType === "APPLICATION_PDF"}
 <PdfPlayerComponent sourceUrl={url} />
{:else if mediaType === "UNSUPPORTED"}
 <p>This file format cannot be displayed.</p>
{:else}
 <p>Some error</p>
{/if}
```

Файл fetchService.svelte

Є реалізацією задачі інтеграції серверної та клієнтської частини застосунку

```
import { PUBLIC_BASE_URL } from "$env/static/public";

export async function fetchService(url, method, requestOptions={}) {
 const defaultRequestOptions = {
   method,
   redirect: "follow",
   credentials: "include",
 }
 const response = await fetch(`${PUBLIC_BASE_URL}${url}`, { ...defaultRequestOptions, ...requestOptions });
 if (response.ok) {
   return response.json();
 }
 throw new Error(response.status);
}
```

Файл +layout.server.js

Є реалізацією задачі інтеграції серверної та клієнтської частини застосунку

```
import { PUBLIC_BASE_URL } from "$env/static/public";
import { redirect } from '@sveltejs/kit';

export async function load({fetch, url}) {
 // List of routes to skip the load function
 const skipRoutes = ['/login'];
```

```
// Check if the current route is in the skip list
if (skipRoutes.some(route => url.pathname.startsWith(route))) {
  return {};
}

const response = await fetch(`${PUBLIC_BASE_URL}/user/current`, {
  method: "GET",
  redirect: "follow",
  credentials: "include",
});
console.log(`response status = ${response.status}`);
if (response.redirected || response.status === 401 || response.status === 403) {
  throw redirect(302, '/login');
}
const currentUser = await response.json();
return {
  currentUser
}
}
```

Файл me/+page.svelte

Реалізація функціональної задачі «Можливість перегляду власного профіля користувача»

```
<script>
 export let data;
 let profile = data.currentUser;
</script>

<style>
 .profile-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    border: 1px solid #ddd;
    border-radius: 8px;
    padding: 20px;
    max-width: 400px;
    margin: 20px auto;
    background-color: #f9f9f9;
 }

 .profile-image {
    border-radius: 50%;
    width: 96px;
    height: 96px;
    margin-bottom: 20px;
 }

 .profile-details {
    text-align: center;
 }

 .profile-details h2 {
    margin: 0;
    font-size: 1.5em;
    color: #333;
 }

 .profile-details p {
```

```
    margin: 5px 0;
    color: #666;
 }

 .profile-details span {
    display: block;
    font-weight: bold;
 }
</style>

<div class="profile-container">
 <img class="profile-image" src={profile.pictureLink} alt="user profile icon" />
 <div class="profile-details">
    <h2>{profile.name}</h2>
    <p><span>First Name:</span> {profile.firstName}</p>
    <p><span>Last Name:</span> {profile.lastName}</p>
    <p><span>External ID:</span> {profile.externalId}</p>
 </div>
</div>
```

Файл login/+page.svelte

Реалізація функціональної задачі «Можливість входу через Google»

```
<script>
 import { PUBLIC_BASE_URL } from "$env/static/public";
</script>

<main>
 <a href={`${PUBLIC_BASE_URL}/oauth2/authorization/google`}>Sign up with Google</a>
</main>
```

Файл GatewayApplication.java

Є реалізацією задачі інтеграції серверної та клієнтської частини застосунку

```
package kpi.zaranik.kexitdrive.gateway;

import kpi.zaranik.kexitdrive.gateway.config.properties.ServiceProperties;
import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;

@Slf4j
@EnableDiscoveryClient
@SpringBootApplication
public class GatewayApplication {

  public static void main(String[] args) {
    SpringApplication.run(GatewayApplication.class, args);
  }

  @Bean
  public RouteLocator customRouteLocator(RouteLocatorBuilder builder, ServiceProperties serviceProperties) {
    return builder.routes()
      .route("kexit-core", r -> r
        .path("/api/**")
```

```
            .filters(filters -> filters.rewritePath("/api/(?<segment>.*)", "/${segment}"))
            .uri(serviceProperties.url().get("kexit-core"))
        )
        .route("frontend", r -> r.order(100)
            .path("/**")
            .uri(serviceProperties.url().get("frontend"))
        )
        .build();
  }

}
```

Файл GoogleWebClientConfig.java

Реалізація функціональної задачі «Можливість імпорту файлів із Google Drive»

```
package kpi.zaranik.kexitdrive.core.config.googleclient;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.util.unit.DataSize;
import org.springframework.web.reactive.function.client.WebClient;

@Configuration
public class GoogleWebClientConfig {

   @Bean
   public WebClient fileImportingWebClient(
      @Value("${spring.servlet.multipart.max-file-size}") DataSize maxInMemorySize,
      @Value("${google-apis-url}") String googleApisUrl
   ) {
      int maxInMemorySizeInBytes = Math.toIntExact(maxInMemorySize.toBytes());
      return WebClient.builder()
         .baseUrl(googleApisUrl)
         .codecs(configurer -> configurer.defaultCodecs().maxInMemorySize(maxInMemorySizeInBytes))
         .build();
   }

}
```

Файл RabbitConfig.java

Реалізація функціональної задачі «Можливість імпорту файлів із Google Drive»

```
package kpi.zaranik.kexitdrive.core.config.rabbit;

import kpi.zaranik.kexitdrive.core.misc.Constants;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitConfig {
```

```
    @Bean
    public Queue importFiles() {
        return new Queue(Constants.IMPORT_FILES_QUEUE, false);
    }

    @Bean
    public RabbitTemplate rabbitTemplate(final ConnectionFactory connectionFactory) {
        RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
        rabbitTemplate.setMessageConverter(rabbitJackson2MessageConverter());
        return rabbitTemplate;
    }

    @Bean
    public Jackson2JsonMessageConverter rabbitJackson2MessageConverter() {
        return new Jackson2JsonMessageConverter();
    }

}
```

Файл SecurityConfig.java

Реалізація функціональної задачі «Можливість входу через Google»

```
package kpi.zaranik.kexitdrive.core.config.security;

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

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
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.core.Authentication;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    private static final String[] PERMIT_ALL = {
        "/user/redirect",
        "/swagger-ui/**",
        "/swagger-ui.html",
        "/v3/api-docs/**"
    };

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(withDefaults())
            .logout(l -> l.logoutSuccessHandler(customLogoutSuccessHandler()))
            .oauth2Login(login -> login.successHandler(customOAuth2LoginSuccessHandler()))
            .oauth2Client(withDefaults())
            .authorizeHttpRequests(requests ->
```

```
            requests
                .requestMatchers(PERMIT_ALL).permitAll()
                .anyRequest().authenticated()
        );

    return http.build();
  }

  @Bean
  AuthenticationSuccessHandler customOAuth2LoginSuccessHandler() {
    return (HttpServletRequest request, HttpServletResponse response, Authentication authentication) -> {
      response.sendRedirect("/");
    };
  }

  @Bean
  LogoutSuccessHandler customLogoutSuccessHandler() {
    return (HttpServletRequest request, HttpServletResponse response, Authentication authentication) -> {
      response.sendRedirect("/");
    };
  }

}
```

Файл FileContfoller.java

Реалізація функціональних задач «Можливість керувати файлами та папками», «Можливість переглядати будь-які типи файлів онлайн», «Можливість завантаження файлів із компьютера», «Можливість імпорту файлів із Google Drive»

```
package kpi.zaranik.kexitdrive.core.controller;

import java.nio.charset.StandardCharsets;
import java.util.List;
import kpi.zaranik.kexitdrive.core.config.security.CurrentUser;
import kpi.zaranik.kexitdrive.core.dto.UserInfo;
import kpi.zaranik.kexitdrive.core.dto.file.FileResponse;
import kpi.zaranik.kexitdrive.core.dto.file.PlayableResourceResponse;
import kpi.zaranik.kexitdrive.core.dto.file.PlayerDataTypeResponse;
import kpi.zaranik.kexitdrive.core.dto.file.UploadedFileResponse;
import kpi.zaranik.kexitdrive.core.dto.file.importing.ImportFilesRequest;
import kpi.zaranik.kexitdrive.core.service.file.FileService;
import lombok.RequiredArgsConstructor;
import org.springframework.core.io.Resource;
import org.springframework.http.ContentDisposition;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

@RestController
@RequestMapping("file")
```

```
@RequiredArgsConstructor
public class FileController {

  private final FileService fileService;

  @GetMapping
  public List<FileResponse> getAllFiles() {
    return fileService.getAllFiles();
  }

  @GetMapping("{id}")
  public FileResponse getFileById(@PathVariable String id) {
    return fileService.getFileById(id);
  }

  @GetMapping("{id}/download")
  public ResponseEntity<Resource> downloadFileById(@PathVariable String id) {
    FileResponse file = fileService.getFileById(id);
    Resource resource = fileService.getFileResourceById(id);
    return ResponseEntity.ok()
      .contentType(MediaType.parseMediaType(file.metadata().contentType()))
      .header(HttpHeaders.CONTENT_DISPOSITION,
ContentDisposition.attachment().filename(resource.getFilename(), StandardCharsets.UTF_8).build().toString())
      .body(resource);
  }

  @GetMapping("{id}/player-content-type")
  public PlayerDataTypeResponse getPlayerContentType(@PathVariable String id) {
    return fileService.getPlayerContentType(id);
  }

  @GetMapping("{id}/play")
  public ResponseEntity<Resource> getPlayableResource(@PathVariable String id) {
    PlayableResourceResponse response = fileService.getPlayableResource(id);
    Resource resource = response.resource();
    String contentType = response.contentType();
    return ResponseEntity.ok()
      .contentType(MediaType.parseMediaType(contentType))
      .body(resource);
  }

  @PostMapping("upload")
  public UploadedFileResponse uploadFile(@RequestPart MultipartFile file, @RequestParam String directoryId) {
    return fileService.uploadFile(file, directoryId);
  }

  @PostMapping("import")
  public void importFilesFromGoogleDrive(@CurrentUser UserInfo user, @RequestBody ImportFilesRequest
request) {
    fileService.importFilesFromGoogleDrive(user, request);
  }

  @DeleteMapping("{id}")
  public void deleteFileById(@PathVariable String id) {
    fileService.deleteFileById(id);
  }

}
```

Файл UserController.java

Реалізація функціональної задачі «Можливість перегляду власного профіля користувача»

```
package kpi.zaranik.kexitdrive.core.controller;

import kpi.zaranik.kexitdrive.core.config.security.CurrentUser;
import kpi.zaranik.kexitdrive.core.dto.UserInfo;
import kpi.zaranik.kexitdrive.core.dto.security.TokenResponse;
import kpi.zaranik.kexitdrive.core.service.auth.AuthorizedClientFactory;
import kpi.zaranik.kexitdrive.core.service.user.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.security.oauth2.client.annotation.RegisteredOAuth2AuthorizedClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("user")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;
    private final AuthorizedClientFactory authorizedClientFactory;

    @GetMapping("current")
    public UserInfo currentUser(@CurrentUser UserInfo user) {
        return user;
    }

    @GetMapping("getAccessToken")
    public TokenResponse getAccessToken(@RegisteredOAuth2AuthorizedClient OAuth2AuthorizedClient
authorizedClient) {
        return userService.getAccessToken(authorizedClient);
    }

}
```

Файл GlobalExceptionHandler.java

Реалізація функціональної задачі «Можливість перегляду власного профіля користувача»

```
package kpi.zaranik.kexitdrive.core.controller.handler;

import java.time.LocalDateTime;
import kpi.zaranik.kexitdrive.core.exception.AccessToResourceDeniedException;
import kpi.zaranik.kexitdrive.core.exception.NoSuitableConverterServiceFoundException;
import kpi.zaranik.kexitdrive.core.exception.ResourceNotFoundException;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ProblemDetail;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@Slf4j
@RestControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {
```

```
    @ExceptionHandler(AccessToResourceDeniedException.class)
    public ProblemDetail handleAccessToResourceDeniedException(AccessToResourceDeniedException exception)
{
        log.warn(exception.getMessage(), exception);
        var problemDetail = ProblemDetail.forStatusAndDetail(HttpStatus.FORBIDDEN, exception.getMessage());
        problemDetail.setProperty("timestamp", LocalDateTime.now());
        return problemDetail;
    }

    @ExceptionHandler(UnsupportedOperationException.class)
    public ProblemDetail handleUnsupportedOperationException(UnsupportedOperationException exception) {
        log.warn(exception.getMessage(), exception);
        var problemDetail = ProblemDetail.forStatusAndDetail(HttpStatus.BAD_REQUEST, exception.getMessage());
        problemDetail.setProperty("timestamp", LocalDateTime.now());
        return problemDetail;
    }

    @ExceptionHandler(NoSuitableConverterServiceFoundException.class)
    public ProblemDetail
handleNoSuitableConverterServiceFoundException(NoSuitableConverterServiceFoundException exception) {
        log.warn(exception.getMessage(), exception);
        var problemDetail = ProblemDetail.forStatusAndDetail(HttpStatus.BAD_REQUEST, exception.getMessage());
        problemDetail.setProperty("timestamp", LocalDateTime.now());
        return problemDetail;
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ProblemDetail handleResourceNotFoundException(ResourceNotFoundException exception) {
        log.warn(exception.getMessage(), exception);
        var problemDetail = ProblemDetail.forStatusAndDetail(HttpStatus.NOT_FOUND, exception.getMessage());
        problemDetail.setProperty("timestamp", LocalDateTime.now());
        return problemDetail;
    }

}
```

Файл FileEntityMapper.java

Реалізація функціональної задачі «Можливість керувати файлами та папками»

```
package kpi.zaranik.kexitdrive.core.mapper;

import kpi.zaranik.kexitdrive.core.dto.file.FileResponse;
import kpi.zaranik.kexitdrive.core.dto.file.FileResponse.FileMetadata;
import kpi.zaranik.kexitdrive.core.dto.file.FileResponse.FileResponseBuilder;
import kpi.zaranik.kexitdrive.core.entity.FileEntity;
import org.springframework.stereotype.Service;

@Service
public class FileEntityMapper {

    public FileResponse mapToResponse(FileEntity entity) {
        FileResponseBuilder builder = FileResponse.builder();

        builder
            .id(entity.id())
            .ownerUserExternalId(entity.ownerUserExternalId())
            .filename(entity.filename())
            .created(entity.created())
            .isDirectory(entity.isDirectory())
```

```
            .containingDirectoryId(entity.containingDirectoryId())
            .metadata(new FileMetadata(
                entity.metadata().gridFsFileId(),
                entity.metadata().contentType(),
                entity.metadata().isImported()
            ));

        return builder.build();
    }

}
```

Файл UserMapper.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.mapper;

import kpi.zaranik.kexitdrive.core.dto.UserInfo;
import org.springframework.security.oauth2.core.oidc.user.OidcUser;
import org.springframework.stereotype.Service;

@Service
public class UserMapper {

  public UserInfo mapFromOidcUser(OidcUser user) {
      return UserInfo.builder()
          .externalId(user.getSubject())
          .name(user.getFullName())
          .firstName(user.getGivenName())
          .lastName(user.getFamilyName())
          .pictureLink(user.getPicture())
          .build();
  }

}
```

Файл AuthorizedClientFactory.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.service.auth;

import static kpi.zaranik.kexitdrive.core.misc.Constants.GOOGLE_CLIENT_REGISTRATION_ID;

import org.springframework.security.oauth2.client.AuthorizationCodeOAuth2AuthorizedClientProvider;
import org.springframework.security.oauth2.client.AuthorizedClientServiceOAuth2AuthorizedClientManager;
import org.springframework.security.oauth2.client.DelegatingOAuth2AuthorizedClientProvider;
import org.springframework.security.oauth2.client.OAuth2AuthorizeRequest;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientService;
import org.springframework.security.oauth2.client.RefreshTokenOAuth2AuthorizedClientProvider;
import org.springframework.security.oauth2.client.registration.ClientRegistrationRepository;
import org.springframework.stereotype.Service;

@Service
public class AuthorizedClientFactory {

    private final AuthorizedClientServiceOAuth2AuthorizedClientManager manager;

    public AuthorizedClientFactory(OAuth2AuthorizedClientService authorizedClientService,
ClientRegistrationRepository clientRegistrationRepository) {
```

```
        this.manager = new AuthorizedClientServiceOAuth2AuthorizedClientManager(clientRegistrationRepository,
authorizedClientService);
        this.manager.setAuthorizedClientProvider(new DelegatingOAuth2AuthorizedClientProvider(
            new AuthorizationCodeOAuth2AuthorizedClientProvider(),
            new RefreshTokenOAuth2AuthorizedClientProvider()
        ));
    }

    public OAuth2AuthorizedClient getClientByExternalUserId(String userExternalId) {
        OAuth2AuthorizeRequest request =
OAuth2AuthorizeRequest.withClientRegistrationId(GOOGLE_CLIENT_REGISTRATION_ID).principal(userExter
nalId).build();
        return manager.authorize(request);
    }

}
```

Файл FileService.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.service.file;

import static kpi.zaranik.kexitdrive.core.misc.Constants.CONVERT_FROM_PROPERTY_NAME;
import static kpi.zaranik.kexitdrive.core.misc.Constants.CONVERT_TO_PROPERTY_NAME;

import com.mongodb.client.gridfs.model.GridFSFile;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import kpi.zaranik.kexitdrive.core.dto.UserInfo;
import kpi.zaranik.kexitdrive.core.dto.file.FileResponse;
import kpi.zaranik.kexitdrive.core.dto.file.PlayableResourceResponse;
import kpi.zaranik.kexitdrive.core.dto.file.PlayerDataTypeResponse;
import kpi.zaranik.kexitdrive.core.dto.file.UploadedFileResponse;
import kpi.zaranik.kexitdrive.core.dto.file.importing.ImportFilesRequest;
import kpi.zaranik.kexitdrive.core.dto.file.importing.ImportingMessage;
import kpi.zaranik.kexitdrive.core.entity.FileEntity;
import kpi.zaranik.kexitdrive.core.entity.PlayerSupportedContentType;
import kpi.zaranik.kexitdrive.core.exception.AccessToResourceDeniedException;
import kpi.zaranik.kexitdrive.core.exception.NoSuitableConverterServiceFoundException;
import kpi.zaranik.kexitdrive.core.exception.ResourceNotFoundException;
import kpi.zaranik.kexitdrive.core.mapper.FileEntityMapper;
import kpi.zaranik.kexitdrive.core.misc.Constants;
import kpi.zaranik.kexitdrive.core.repository.FileEntityRepository;
import kpi.zaranik.kexitdrive.core.service.user.UserService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.core.io.Resource;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.gridfs.GridFsTemplate;
import org.springframework.http.HttpEntity;
import org.springframework.http.client.MultipartBodyBuilder;
import org.springframework.stereotype.Service;
import org.springframework.util.MultiValueMap;
import org.springframework.web.multipart.MultipartFile;
```

```
import org.springframework.web.reactive.function.client.WebClient;

@Slf4j
@Service
@RequiredArgsConstructor
public class FileService {

    private final RabbitTemplate rabbitTemplate;
    private final GridFsTemplate gridFsTemplate;
    private final TikaService tikaService;
    private final FileEntityRepository fileRepository;
    private final UserService userService;
    private final FileEntityMapper fileEntityMapper;
    private final DiscoveryClient discoveryClient;
    @Qualifier("loadBalancedWebClientBuilder")
    private final WebClient.Builder webClientBuilder;

    public UploadedFileResponse uploadFile(MultipartFile file, String containingDirectoryId) {
        try (InputStream fileInputStream = file.getInputStream()) {
            String originalFilename = file.getOriginalFilename();
            String contentType = tikaService.getDataType(file);
            String gridFsFileId = gridFsTemplate.store(fileInputStream, originalFilename, contentType).toString();
            FileEntity uploadedFile = FileEntity.createUploadedFile(originalFilename, gridFsFileId, contentType,
containingDirectoryId);
            uploadedFile = fileRepository.save(uploadedFile);
            return new UploadedFileResponse(uploadedFile.id());
        } catch (IOException exception) {
            log.error("Error while uploading file", exception);
            throw new RuntimeException(exception); // TODO
        }
    }

    // 111401278168521168568
    public void importFilesFromGoogleDrive(UserInfo user, ImportFilesRequest request) {
        List<String> fileIds = request.fileIds();
        fileIds.stream()
            .map(id -> new ImportingMessage(user.externalId(), id))
            .forEach(message -> rabbitTemplate.convertAndSend(Constants.IMPORT_FILES_QUEUE, message));
    }

    public FileResponse getFileById(String fileId) {
        FileEntity fileEntity = getFileEntity(fileId);
        return fileEntityMapper.mapToResponse(fileEntity);
    }

    public Resource getFileResourceById(String fileId) {
        FileEntity fileEntity = getFileEntity(fileId);
        String gridFsFileId = fileEntity.metadata().gridFsFileId();
        GridFSFile gridFsFile = Optional.ofNullable(gridFsTemplate.findOne(new
Query(Criteria.where("_id").is(gridFsFileId))))
            .orElseThrow(() -> new ResourceNotFoundException("Unable to find file with fileId " + fileEntity.id() + "
and gridFsFileId = " + gridFsFileId));
        return gridFsTemplate.getResource(gridFsFile);
    }

    public void deleteFileById(String fileId) {
        FileEntity fileEntity = getFileEntity(fileId);
        if (!fileEntity.isDirectory()) {
            String gridFsFileId = fileEntity.metadata().gridFsFileId();
            gridFsTemplate.delete(new Query(Criteria.where("_id").is(gridFsFileId)));
        }
        fileRepository.delete(fileEntity);
    }
```

```
    private FileEntity getFileEntity(String fileId) {
        FileEntity fileEntity = fileRepository.findById(fileId).orElseThrow(ResourceNotFoundException::new);
        String currentUserExternalId = userService.getCurrentUserExternalId().orElseThrow();
        if (!fileEntity.ownerUserExternalId().equals(currentUserExternalId)) {
            throw new AccessToResourceDeniedException("user with id " + currentUserExternalId + " is not allowed to
access file " + fileEntity.id());
        }
        return fileEntity;
    }

    public PlayerDataTypeResponse getPlayerContentType(String fileId) {
        FileEntity fileEntity = getFileEntity(fileId);
        if (fileEntity.isDirectory()) { // TODO move to AOP validation
            throw new UnsupportedOperationException("This operation is not supported for directory with id " + fileId);
        }
        Optional<PlayerSupportedContentType> supportedContentType =
PlayerSupportedContentType.getForMimeType(fileEntity.metadata().contentType());
        if (supportedContentType.isPresent()) {
            return new PlayerDataTypeResponse(supportedContentType.get());
        }

        PlayerSupportedContentType playerSupportedContentType =
getConverterServiceInstance(fileEntity.metadata().contentType())
            .flatMap(instance ->
PlayerSupportedContentType.getForMimeType(instance.getMetadata().get(CONVERT_TO_PROPERTY_NAME))
)
            .orElse(PlayerSupportedContentType.UNSUPPORTED);
        return new PlayerDataTypeResponse(playerSupportedContentType);
    }

    public PlayableResourceResponse getPlayableResource(String fileId) {
        FileEntity fileEntity = getFileEntity(fileId);
        if (fileEntity.isDirectory()) {
            throw new UnsupportedOperationException("This operation is not supported for directory with id " + fileId);
        }

        Resource resource = getFileResourceById(fileId);

        String originalContentType = fileEntity.metadata().contentType();
        Optional<PlayerSupportedContentType> supportedContentType =
PlayerSupportedContentType.getForMimeType(originalContentType);
        if (supportedContentType.isPresent()) {
            return new PlayableResourceResponse(resource, originalContentType);
        }

        ServiceInstance converterServiceInstance = getConverterServiceInstance(originalContentType)
            .orElseThrow(() -> new NoSuitableConverterServiceFoundException("No suitable converter service found
for " + fileEntity.id()));
        Resource convertedResource = requestResource(converterServiceInstance, resource);
        String convertToContentType =
converterServiceInstance.getMetadata().get(CONVERT_TO_PROPERTY_NAME);
        return new PlayableResourceResponse(convertedResource, convertToContentType);
    }

    private Resource requestResource(ServiceInstance converterServiceInstance, Resource originalResource) {
        String serviceId = converterServiceInstance.getServiceId();
        WebClient webClient = webClientBuilder.baseUrl("lb://" + serviceId).build();

        MultipartBodyBuilder builder = new MultipartBodyBuilder();
        builder
            .part("file", originalResource);
        MultiValueMap<String, HttpEntity<?>> multipartBody = builder.build();
```

```
        return webClient.post()
            .uri("convert")
            .bodyValue(multipartBody)
            .retrieve()
            .bodyToMono(Resource.class)
            .block();
    }

    private Optional<ServiceInstance> getConverterServiceInstance(String contentType) {
        List<ServiceInstance> convertFrom = discoveryClient.getServices().stream()
            .flatMap(serviceId -> discoveryClient.getInstances(serviceId).stream())
            .filter(instance -> {
                Map<String, String> instanceMetadata = instance.getMetadata();
                List<String> convertFromContentTypes = instanceMetadata.keySet().stream()
                    .filter(key -> key.startsWith(CONVERT_FROM_PROPERTY_NAME))
                    .map(instanceMetadata::get)
                    .toList();
                return convertFromContentTypes.contains(contentType);
            })
            .toList();
        return convertFrom.stream().findFirst();
    }

    public List<FileResponse> getAllFiles() {
        String currentUserExternalId = userService.getCurrentUserExternalId().orElseThrow();
        return fileRepository.findByOwnerUserExternalIdOrderByCreatedDesc(currentUserExternalId)
            .map(fileEntityMapper::mapToResponse)
            .toList();
    }
}
```

Файл ImportingService.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.service.file;

import kpi.zaranik.kexitdrive.core.config.properties.MimeTypesToExportMappingProperties;
import kpi.zaranik.kexitdrive.core.config.properties.MimeTypesToExportMappingProperties.ExportMapping;
import kpi.zaranik.kexitdrive.core.dto.DriveFile;
import kpi.zaranik.kexitdrive.core.dto.file.importing.ImportingMessage;
import kpi.zaranik.kexitdrive.core.entity.FileEntity;
import kpi.zaranik.kexitdrive.core.misc.Constants;
import kpi.zaranik.kexitdrive.core.repository.FileEntityRepository;
import kpi.zaranik.kexitdrive.core.service.GoogleApiService;
import kpi.zaranik.kexitdrive.core.service.auth.AuthorizedClientFactory;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.core.io.Resource;
import org.springframework.data.mongodb.gridfs.GridFsTemplate;
import org.springframework.scheduling.annotation.Async;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.stereotype.Service;

@Slf4j
@Service
@RequiredArgsConstructor
public class ImportingService {

    private final AuthorizedClientFactory authorizedClientFactory;
```

```
    private final GoogleApiService googleApiService;
    private final TikaService tikaService;
    private final FileEntityRepository fileEntityRepository;
    private final GridFsTemplate gridFsTemplate;
    private final MimeTypesToExportMappingProperties mimeTypesToExportMapping;

    @Async
    @RabbitListener(queues = Constants.IMPORT_FILES_QUEUE)
    public void importFilesRabbitListener(ImportingMessage message) {
        try {
            OAuth2AuthorizedClient authorizedClient =
authorizedClientFactory.getClientByExternalUserId(message.userExternalId());
            DriveFile driveFile = googleApiService.getDriveFileInfo(authorizedClient, message.fileId());

            Resource resource;
            String fileName;
            if (mimeTypesToExportMapping.mapping().containsKey(driveFile.mimeType())) {
                ExportMapping exportMapping = mimeTypesToExportMapping.mapping().get(driveFile.mimeType());
                resource = googleApiService.exportFileResource(authorizedClient, driveFile.id(),
exportMapping.exportMimeType());
                fileName = driveFile.name() + "." + exportMapping.extension();
            } else {
                resource = googleApiService.getFileResource(authorizedClient, driveFile.id());
                fileName = driveFile.name();
            }
            String dataType = tikaService.getDataType(resource.getInputStream());
            String gridFsFileId = gridFsTemplate.store(resource.getInputStream(), fileName, dataType).toString();
            FileEntity importedFile = FileEntity.createImportedFile(fileName, gridFsFileId, dataType, null,
message.userExternalId());

            fileEntityRepository.save(importedFile);

            log.info("Successfully isImported file with external id {} for user {}", message.fileId(),
message.userExternalId());
        } catch (Exception e) {
            log.warn("Failed to import file with external id {} for user {}", message.fileId(), message.userExternalId(),
e);
        }
    }

}
```

Файл TikaService.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.service.file;

import java.io.InputStream;
import lombok.SneakyThrows;
import org.apache.tika.Tika;
import org.springframework.data.mongodb.gridfs.GridFsResource;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

@Service
public class TikaService {

    private final Tika tika;

    public TikaService() {
        this.tika = new Tika();
```

```
    }

    @SneakyThrows
    public String getDataType(InputStream inputStream)  {
        return tika.detect(inputStream);
    }

    @SneakyThrows
    public String getDataType(MultipartFile multipartFile)  {
        return tika.detect(multipartFile.getInputStream());
    }

    @SneakyThrows
    public String getDataType(GridFsResource resource)  {
        return tika.detect(resource.getInputStream());
    }

}
```

Файл UserService.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.service.user;

import java.security.Principal;
import java.util.Optional;
import kpi.zaranik.kexitdrive.core.dto.security.TokenResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class UserService {

  public TokenResponse getAccessToken(OAuth2AuthorizedClient authorizedClient) {
      return new TokenResponse(authorizedClient.getAccessToken().getTokenValue());
  }

  public Optional<String> getCurrentUserExternalId() {
      return Optional.ofNullable(SecurityContextHolder.getContext())
          .map(SecurityContext::getAuthentication)
          .map(Principal::getName);
  }

}
```

Файл GoogleApiService.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.core.service;

import kpi.zaranik.kexitdrive.core.dto.DriveFile;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
```

```
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;

@Service
@RequiredArgsConstructor
public class GoogleApiService {

   @Qualifier("fileImportingWebClient")
   private final WebClient webClient;

   public DriveFile getDriveFileInfo(OAuth2AuthorizedClient authorizedClient, String fileId) {
      String accessToken = authorizedClient.getAccessToken().getTokenValue();
      return webClient.get()
         .uri("drive/v3/files/{fileId}", fileId)
         .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
         .retrieve()
         .bodyToMono(DriveFile.class)
         .block();
   }

   public Resource getFileResource(OAuth2AuthorizedClient authorizedClient, String fileId) {
      String accessToken = authorizedClient.getAccessToken().getTokenValue();
      return webClient.get()
         .uri("drive/v3/files/{fileId}?alt=media", fileId)
         .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
         .retrieve()
         .bodyToMono(Resource.class)
         .block();
   }

   public Resource exportFileResource(OAuth2AuthorizedClient authorizedClient, String fileId, String
exportMimeType) {
      String accessToken = authorizedClient.getAccessToken().getTokenValue();
      return webClient.get()
         .uri(uriBuilder -> uriBuilder
            .path("drive/v3/files/{fileId}/export")
            .queryParam("mimeType", exportMimeType)
            .build(fileId))
         .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
         .retrieve()
         .bodyToMono(Resource.class)
         .block();
   }

}
```

Файл docker-compose.yaml.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
version: "3.9"

name: kexit

services:

  eureka:
    image: zaranik/kexit-drive-eureka:latest
    restart: always
    ports:
      - "8761:8761"
```

```
  profiles:
    - prod

gateway:
  image: zaranik/kexit-drive-gateway:latest
  restart: always
  ports:
    - "80:80"
  profiles:
    - prod
  depends_on:
    eureka:
      condition: service_healthy

core:
  image: zaranik/kexit-drive-core:latest
  restart: always
  ports:
    - "8080:8080"
  profiles:
    - prod
  depends_on:
    mongo:
      condition: service_healthy
    eureka:
      condition: service_healthy

converter-audio:
  image: zaranik/converter-audio:latest
  restart: always
  profiles:
    - prod
  depends_on:
    eureka:
      condition: service_healthy

converter-video:
  image: zaranik/converter-video:latest
  restart: always
  profiles:
    - prod
  depends_on:
    eureka:
      condition: service_healthy

converter-microsoft-office:
  image: zaranik/converter-video:latest
  restart: always
  profiles:
    - prod
  depends_on:
    eureka:
      condition: service_healthy

mongo:
  image: mongo:6.0.3
  restart: always
  ports:
    - "27017:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: password
  volumes:
```

```
    - ./data/mongo:/data/db
  healthcheck:
    test: echo 'db.runCommand("ping").ok' | mongosh localhost:27017/test --quiet
    interval: 10s
    timeout: 5s
    retries: 5

 rabbitmq:
  image: rabbitmq:3.9.7-management
  restart: always
  environment:
    RABBITMQ_DEFAULT_USER: admin
    RABBITMQ_DEFAULT_PASS: password
  ports:
    - "5672:5672"
    - "15672:15672"
  volumes:
    - ./data/rabbit:/var/lib/rabbitmq

 gotenberg:
  image: gotenberg/gotenberg:8
  restart: always
  ports:
    - "3000:3000"

 ffmpeg:
  image: surebert/docker-ffpmeg:latest
  restart: always
  ports:
    - "9025:3000"
  volumes:
    - ./data/upload:/usr/src/app/uploads
```

Файл ConvertToPdfController.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.convertermicrosoftoffice.converter;

import java.util.Objects;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.ContentDisposition;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.http.client.MultipartBodyBuilder;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.reactive.function.client.WebClient;

@Slf4j
@RestController
public class ConvertToPdfController {

   @Autowired
   WebClient webClient;
```

```
    @PostMapping("convert")
    public ResponseEntity<Resource> getFileConvertedToPdf(@RequestPart MultipartFile file) {
        MultipartBodyBuilder builder = new MultipartBodyBuilder();

        boolean singlePageSheets = Objects.equals(file.getContentType(),
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");

        log.info("singlePageSheets = {}", singlePageSheets);

        builder
            .part("singlePageSheets", singlePageSheets);

        builder
            .part("file", file.getResource());


        MultiValueMap<String, HttpEntity<?>> multipartBody = builder.build();

        Resource converted = webClient.post()
            .uri("forms/libreoffice/convert")
            .bodyValue(multipartBody)
            .retrieve()
            .bodyToMono(Resource.class)
            .block();

        ContentDisposition contentDisposition = ContentDisposition.inline().build();
        HttpHeaders headers = new HttpHeaders();
        headers.setContentDisposition(contentDisposition);

        return ResponseEntity.ok()
            .contentType(MediaType.APPLICATION_PDF)
            .headers(headers)
            .body(converted);
    }

}
```

Файл ConvertToAudioController.java

Реалізація функціональної задачі «Керування профілями користувачів»

```
package kpi.zaranik.kexitdrive.converteraudio.converter;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.ContentDisposition;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.http.client.MultipartBodyBuilder;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.reactive.function.client.WebClient;

@Slf4j
@RestController
public class ConvertAudioController {
```

```
    @Autowired
    WebClient webClient;

    @PostMapping("convert")
    public ResponseEntity<Resource> getFileConverted(@RequestPart MultipartFile file) {
        MultipartBodyBuilder builder = new MultipartBodyBuilder();

        builder
            .part("file", file.getResource());

        MultiValueMap<String, HttpEntity<?>> multipartBody = builder.build();

        Resource converted = webClient.post()
            .uri("mp3")
            .bodyValue(multipartBody)
            .retrieve()
            .bodyToMono(Resource.class)
            .block();

        ContentDisposition contentDisposition = ContentDisposition.inline().build();
        HttpHeaders headers = new HttpHeaders();
        headers.setContentDisposition(contentDisposition);

        return ResponseEntity.ok()
            .contentType(MediaType.parseMediaType("audio/mpeg"))
            .headers(headers)
            .body(converted);
    }
}
```

Факультет інформатики та обчислювальної техніки Кафедра
інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ “_”
_____2024 р.

ВЕБСЕРВІС ДЛЯ ПІДТРИМКИ ДІЯЛЬНОСТІ СЛУЖБИ ДОСТАВКИ

Програма та методика тестування

КПІ.ІП-0109.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____Олександр ПАВЛОВ

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

Виконавець:
_____Богдан ЗАРАНІК

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблений в рамках дипломного проєкту архітектурного рішення для файлового сховища, що розроблений мовою Java під управлінням фреймворку Spring Boot та JavaScript під управлінням фреймворку Svete Kit.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка правильності збереження даних та збереження структури бази даних;

– перевірка сумісності застосунку з різними операційними системами (Windows, Linux);

– знаходження проблем, помилок і недоліків з метою їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

− статичне тестування − перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

− динамічне тестування − застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

− мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

− функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально, з метою знаходження помилок та недоліків у функціональній частині програмного забезпечення. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на виявлення недоліків, що стосуються збереження даних і цілісності бази даних;
- тестування валідації запитів;
- тестування виконання контрольного прикладу.

Факультет інформатики та обчислювальної техніки Кафедра
інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ “_”
_____2024 р.

ВЕБСЕРВІС ДЛЯ ПІДТРИМКИ ДІЯЛЬНОСТІ СЛУЖБИ ДОСТАВКИ

Керівництво користувача

КПІ.ІП-0110.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____Олександр ПАВЛОВ

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

Виконавець:
_____Богдан ЗАРАНІК

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Розробка призначена для забезпечення файлового сховище засобами онлайн-перегляду раніше недоступних форматів файлів користувача із можливістю імпорту файлів із сховища Google Drive на основі запропонованого оригінального архітектурного рішення.

Головною метою розробки є спрощення процесу додавання нових підтримуваних форматів файлів, перегляд яких раніше був недоступний онлайн.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесора: Intel Core i3;
– об‘єм ОЗП: 8 Гб;
– об’єм постійної пам’яті: 512 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип процесора: Intel Core i7;
– об‘єм ОЗП: 32 Гб;
– об’єм постійної пам’яті: 2 Тб;
– підключення до мережі Інтернет зі швидкістю від 300 мегабіт;

2.2 Завантаження застосунку

На даний момент вебсервіс можна розгорнути локально, використавши його вихідний код.

Для розгортання необхідно завантажити вихідний код додатку з публічно доступного репозиторію, створити Docker-образи усіх сервісів та використати Docker Compose, який автоматично створить контейнери для клієнтського, серверного застосунків і бази даних та виконає запуск системи.

2.3 Перевірка коректної роботи

При коректному розгортання введенні посилання на сайт (при локальному розгортанні http://localhost:80) відкривається сторінка входу до системи.

3 ВИКОНАННЯ ПРОГРАМИ

У якості клієнту під час виконання контрольного прикладу був використаний Postman [28] і браузер Google. Для виконання успішної перевірки коректності роботи необхідно виконати такі кроки:

– крок 1: перейти на сторінку логіну;

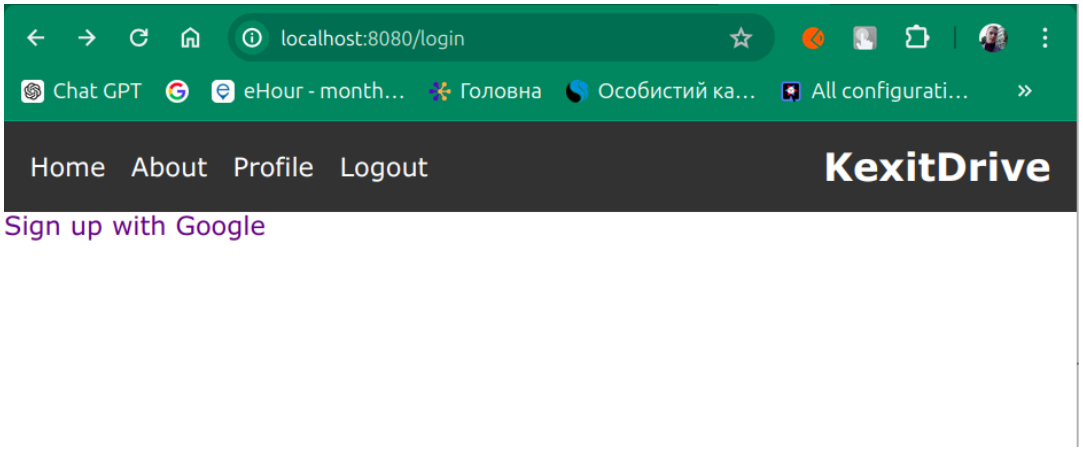


Рисунок 4.2 – Сторінка логіну

– крок 2: натиснути кнопку “Sign Up With Google”;

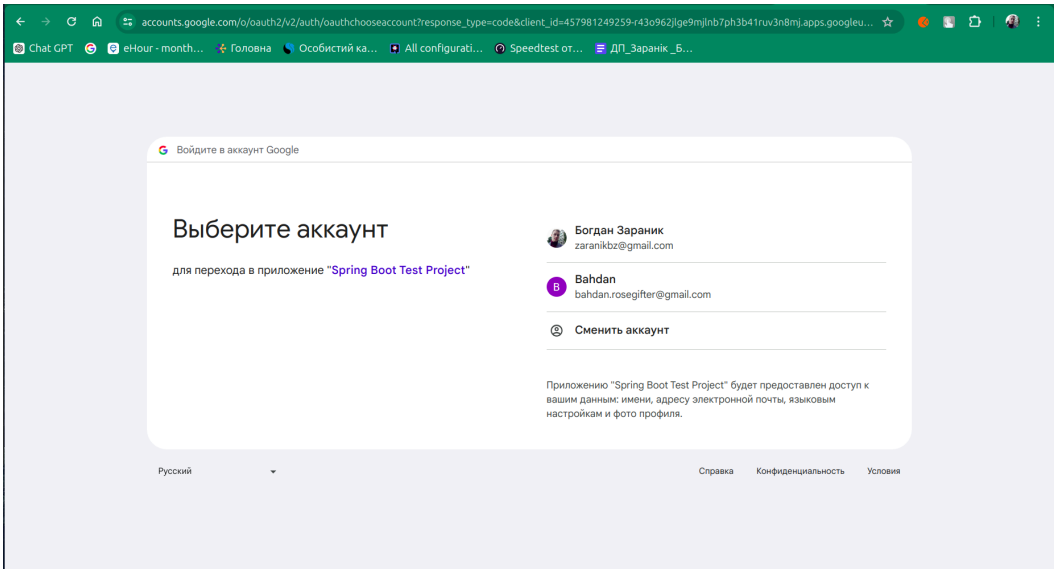


Рисунок 4.3 – Сторінка вибору акаунта

– крок 3: обрати акаунт та погодитись із наданням застосунку прав доступу до Google акаунта користувача;

– крок 4: автоматичний перехід на сторінку кореневого каталогу;

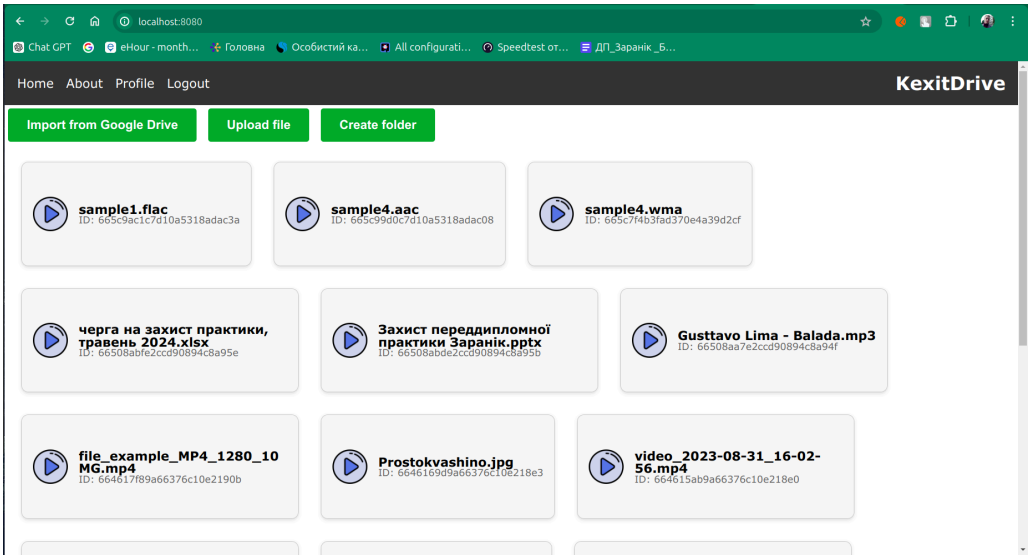


Рисунок 4.4 – Сторінка кореневого каталогу

– крок 5: натиснути кнопку “Import from Google Drive”;

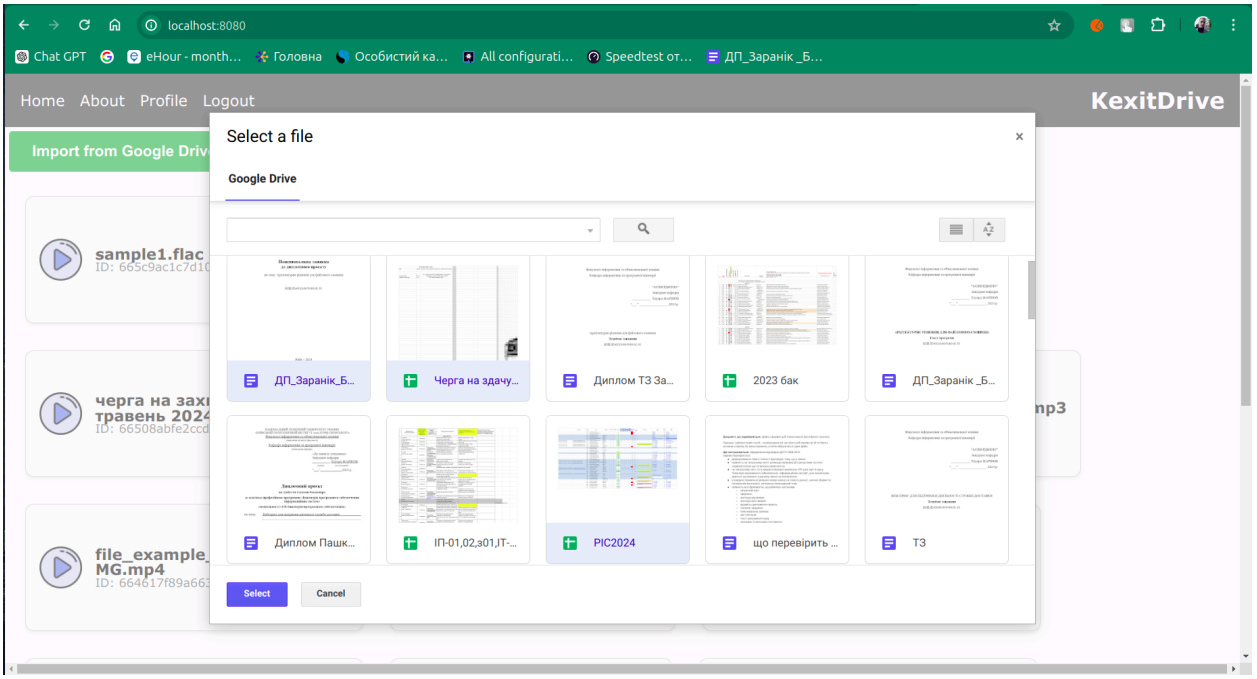


Рисунок 4.5 – Модальне вікно вибору файлів з Google Drive

– крок 6: у відкритому модальному вікні обрати будь-які файли та натиснути кнопку “Select”

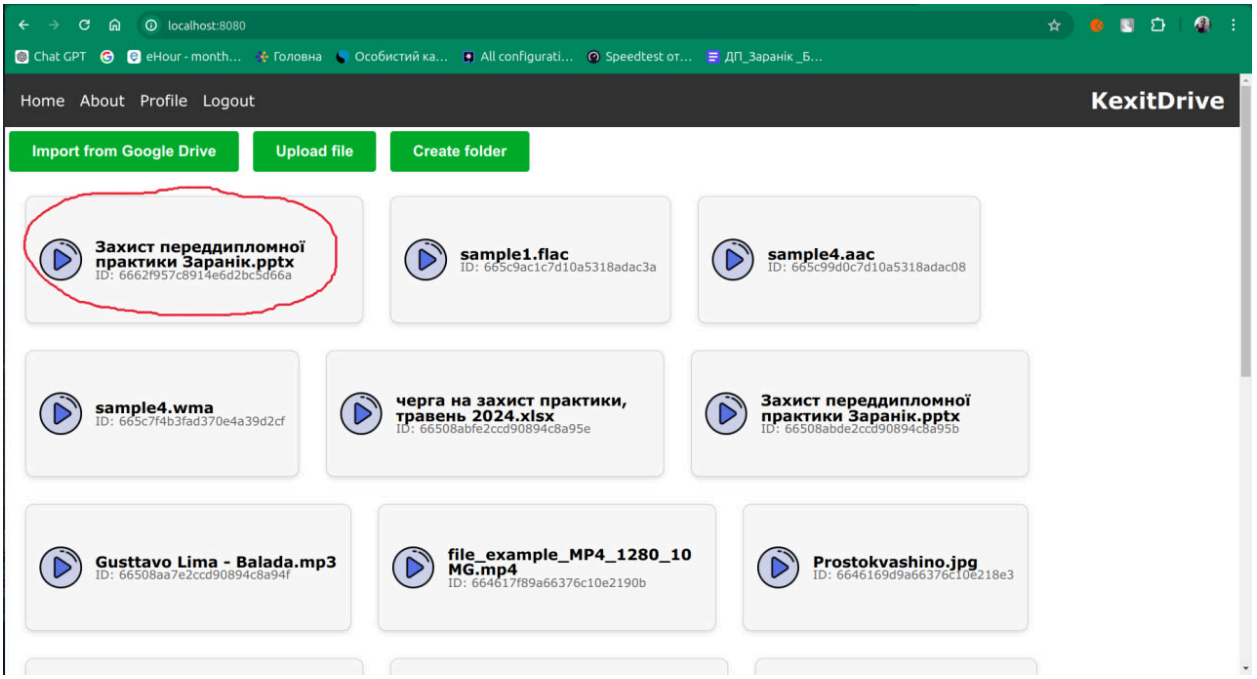


Рисунок 4.6 – Файл імпортовано з Google Drive до поточного каталогу

– крок 7: натиснути кнопку “Upload file”;

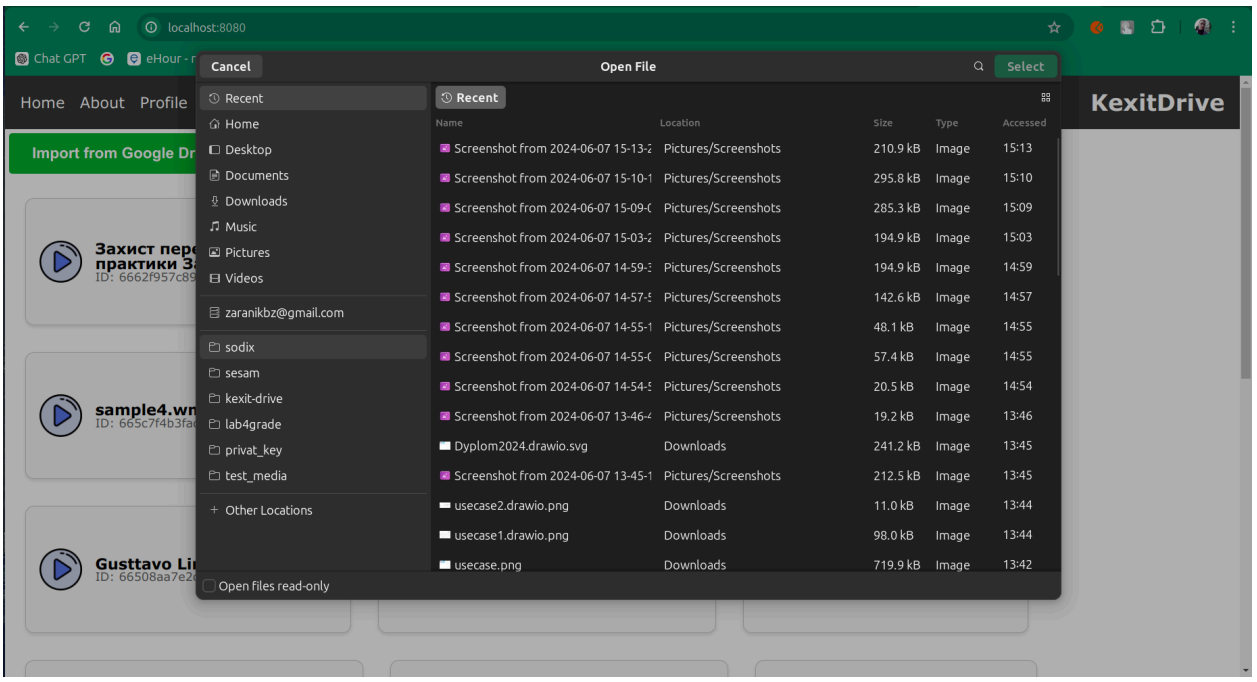


Рисунок 4.7 – Вікно файлового провідника

– крок 8: у відкритому вікні файлового провідника обрати файл та завантажити його до системи;

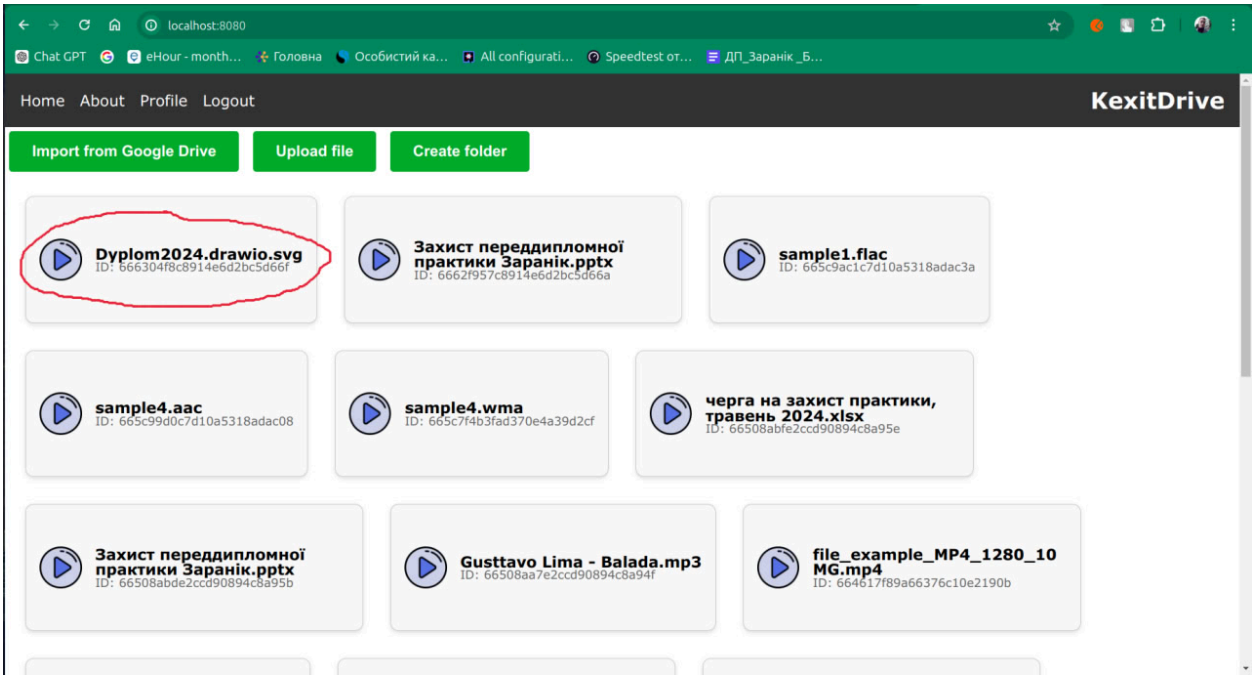


Рисунок 4.8 – Файл завантажено з локального сховища до поточного каталогу

– крок 9: натиснути на кнопку плеера на файлі (або двічі клацнути на файл), відкриється плеєр і файл можна програти онлайн;

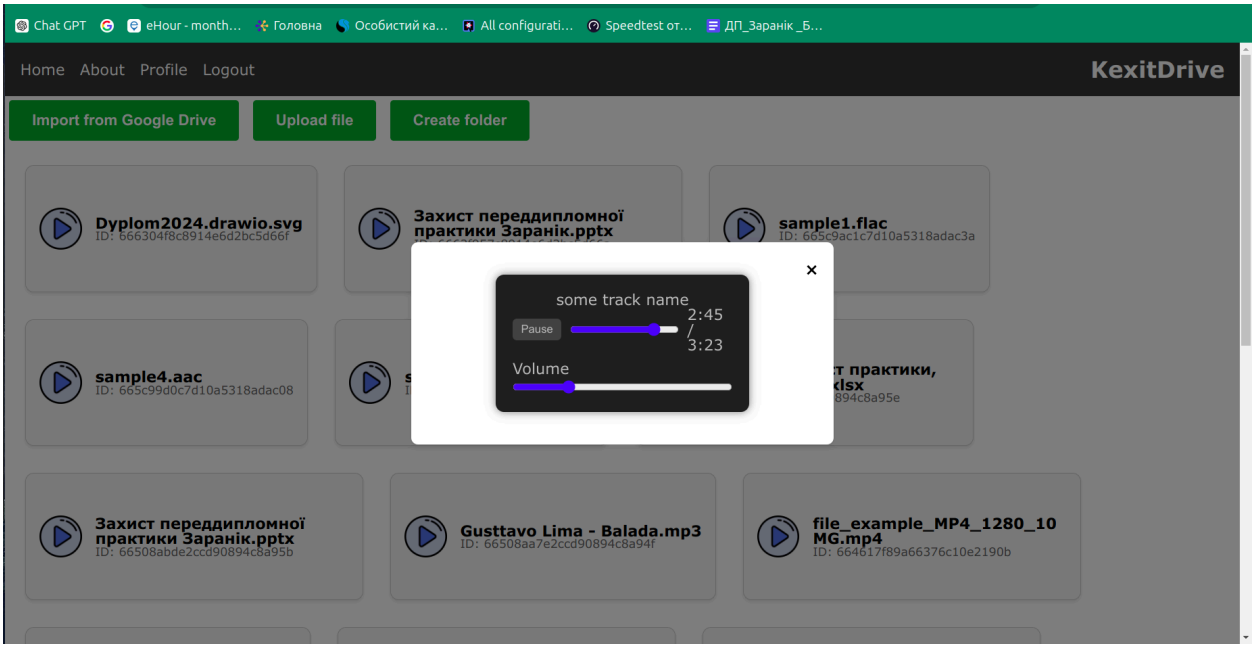


Рисунок 4.9 – Модальне вікно з плеєром, який відкриває файл даного типу

– крок 10: натиснути кнопку “Create folder” та у відкритому модальному вікні ввести назву нової папки;

– крок 11: натиснути вкладку “Profile” - відкривається сторінка із особистою інформацією користувача;

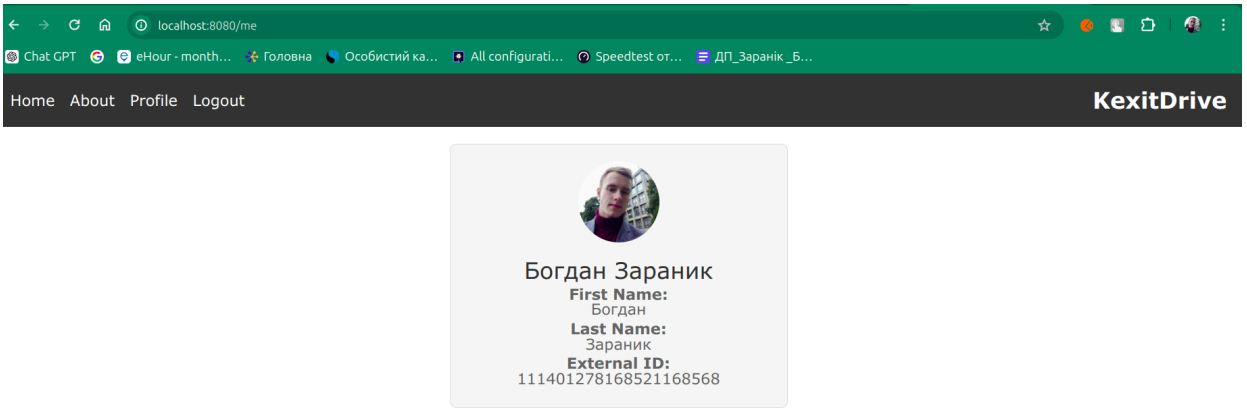


Рисунок 4.10 – Сторінка із особистою інформацією користувача

– крок 12: натиснути на кнопку “Logout” та вийти з системи, користувача буде направлено на сторінку логіну;

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Архітектурне рішення для файлового сховища

Графічний матеріал

КПІ.ІП-0110.045440.06.99

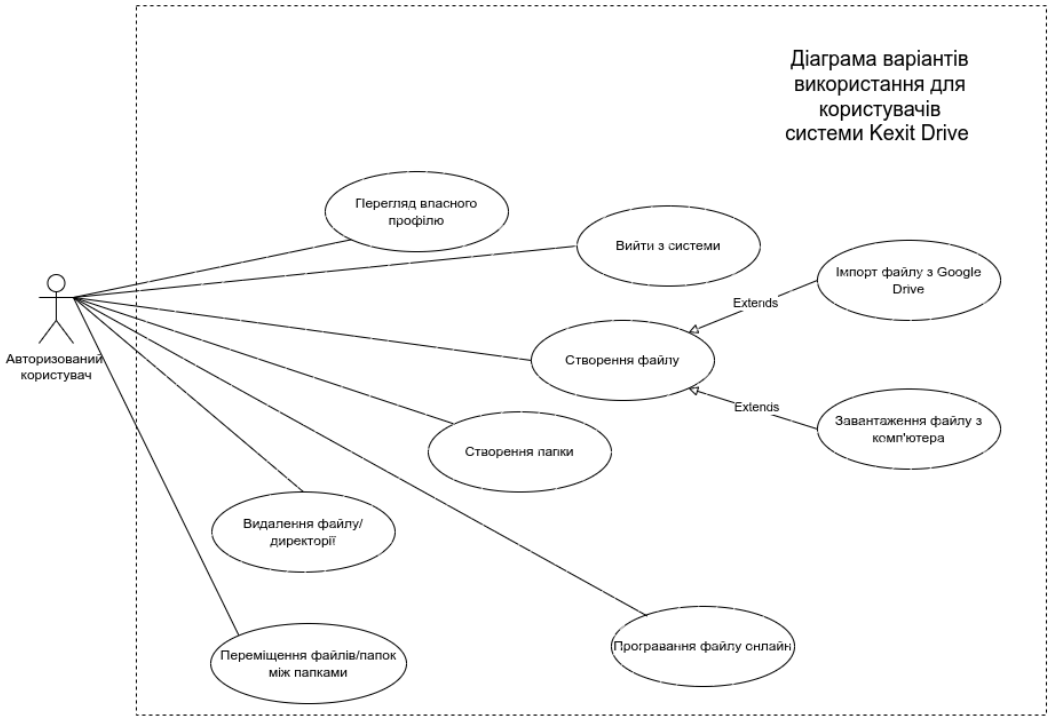
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

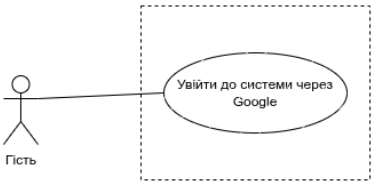
Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Богдан ЗАРАНІК

Київ – 2024

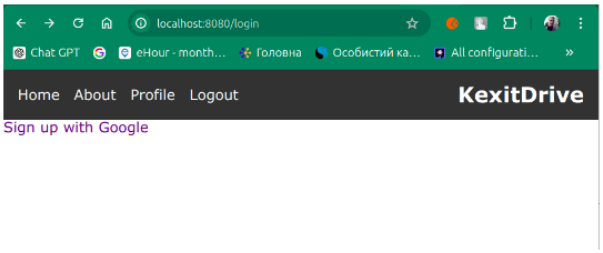
КПІ.ІП-0110.045440.06.99.ССМ

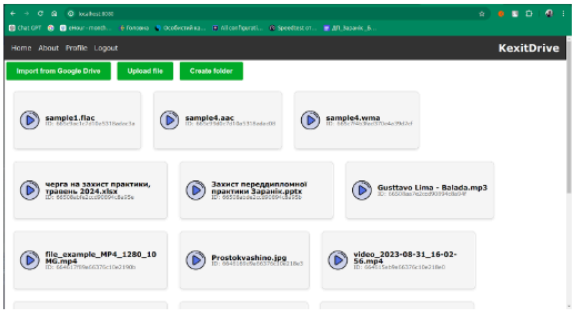


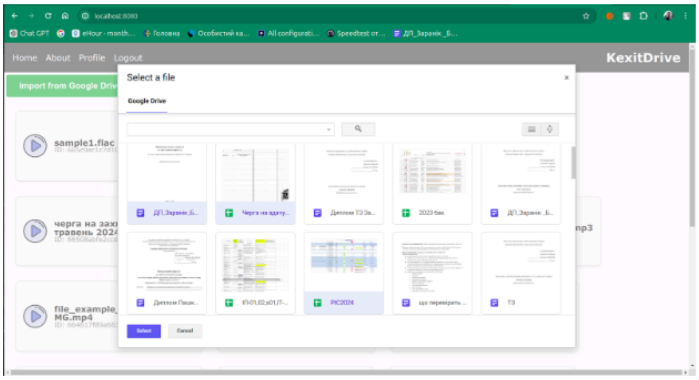


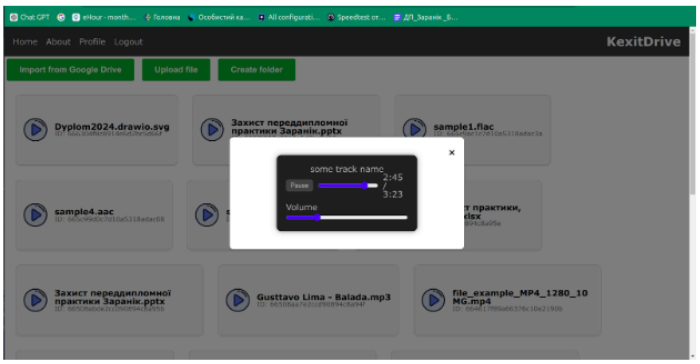
					КПІ.ІП-0110.045440.06.99.ССМ	
					Схема структурних компонентів програмного забезпечення	Лит. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш / Аркушів
Розробив		Заранік Б.Ю.			Архітектурне рішення для файлового сховища	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01
Перевірив		Павлов О.А.				
Т. контр.						
Н. контр.		Головченко М.М.				
Затвердив		Жаріков Е.В.				

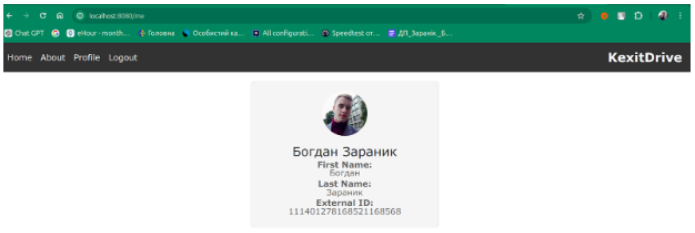
КПІ.ІП-0110.045440.06.99.ЕК











					КПІ.ІП-0110.045440.06.99.ЕК			
Зм.	Арк.	№ докум.	Підп.	Дата				
Розробив		Заранік Б.Ю.			Схема структурних компонентів програмного забезпечення	Лит.	Маса	Масштаб
Перевірив		Павлов О.А.						
Т. контр.						Аркуш	Аркушів	
Н. контр.		Головченко М.М.			Архітектурне рішення для файлового сховища	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01		
Затвердив		Жаріков Е.В.						

КПІ.ІП-0110.045440.06.99.ССМ

					КПІ.ІП-0110.045440.06.99.ССМ
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурних компонентів програмного забезпечення
Розробив		Заранік Б.Ю.			
Перевірив		Павлов О.А.			
Т. контр.					
Н. контр.		Головченко М.М.			Архітектурне рішення для файлового сховища
Затвердив		Жаріков Е.В.			

Літера	Маса	Масштаб
Аркуш	Аркушів	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-01