Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Тестування та розробка Frontend частини медіа-сховища із використанням фреймворку Nuxt

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

_____ Легка Х. А.
(підпис) (прізвище та ініціали)

Керівник _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2024

РЕФЕРАТ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2024, Сторінок 53, таблиць -, рисунків 25, презентація.

Тема: Тестування та розробка Frontend частини медіа-сховища з використанням фреймворку Nuxt**.**

Кваліфікаційна робота бакалавра присвячена процесу розробки та тестування системи медіа-сховища. Проєкт реалізовано у вигляді веб-сайту, що обумовлено його доступністю та зручністю для користувачів, а також легкістю реалізації та гнучкістю при подальшому розширенні функціоналу. Проведено дослідження конкурентних сервісів, що надають подібні послуги, проаналізовано переваги та недоліки наявних платформ.

У ході роботи сплановано архітектуру frontend частини веб-сайту. Використовуючи фреймворк Nuxt, було створено клієнтську частину сайту. Застосування цього фреймворку дозволило забезпечити серверний рендеринг і покращити SEO-оптимізацію, що позитивно вплинуло на користувацький досвід і швидкість роботи сайту.

Крім того, у роботі детально описано процес тестування та відлагодження клієнтської частини. Для цього застосовувались різноманітні інструменти та методи тестування, що забезпечило високу якість кінцевого продукту. Також було враховано можливості подальшого масштабування та модернізації системи, що гарантує її довготривалу актуальність та функціональність.

Ключові слова: веб-сайт, медіа-сховище, Nuxt Веб-розробка, тестування, SEO-оптимізація, клієнтська частина.

ANNOTATION

Bachelor's qualification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2024, Pages 53, tables -, figures 25, presentation.

Topic: Testing and development of the frontend part of a Media Storage System using the Nuxt framework

This bachelor's thesis is dedicated to the process of developing and testing a media storage system. The project is implemented as a website, which is justified by its accessibility and convenience for users, as well as the ease of implementation and flexibility for future functionality expansion. A study of competing services providing similar functionalities was conducted, and the advantages and disadvantages of existing platforms were analyzed.

During the work, the architecture of the frontend part of the website was planned. Using the Nuxt framework, the client-side of the site was developed. The application of this framework allowed for server-side rendering and improved SEO optimization, which positively affected the user experience and the website's performance.

In addition, the thesis describes in detail the process of testing and debugging the client-side. Various tools and testing methods were employed to ensure the high quality of the final product. Furthermore, the possibilities for future scaling and modernization of the system were considered, ensuring its long-term relevance and functionality.

Keywords: website, media storage, Nuxt, web development, testing, SEO optimization, client-side.

ЗМІСТ

РЕФЕРАТ 4
ANNOTATION 5
ВСТУП 7
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Аналіз конкурентних засобів 9
1.2 Порівняльний аналіз конкурентів 10
1.3 Вибір напрямку дослідження 14
1.4 Методологія розробки 16
1.5 Формування вимог до системи 17
2 РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ 19
2.1 Архітектура системи 20
2.2 Розробка бізнес моделі 21
2.3 Проєктування архітектури 24
2.4 Послідовності процесів у системі 26
2.5 Розробка системи 27
2.6 Тестування системи 32
2.6.1 Функціональне тестування 33
2.6.2 Навантажувальне тестування 35
2.6.3 Автоматизоване тестування 38
3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ 41
3.1 Працездатність людини – оператора 41
3.2 Характеристика небезпечних зон обладнання і розробка заходів безпеки 42
ВИСНОВКИ 46
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 48
ДОДАТКИ 50

ВСТУП

У сучасному цифровому світі обсяг медіа-контенту постійно зростає, що вимагає наявності ефективних рішень для його зберігання та управління. Медіа-сховища відіграють важливу роль у забезпеченні можливості користувачів зберігати, організовувати, поширювати та відтворювати різноманітні мультимедійні файли. В умовах постійного збільшення обсягу даних, забезпечення зручного, швидкого та надійного доступу до медіа-контенту стає надзвичайно актуальним завданням. Крім того, зростаючі вимоги до безпеки даних та захисту приватності користувачів додають додаткових викликів при розробці таких систем.

Одним із найбільш перспективних підходів до реалізації медіа-сховищ є використання веб-технологій, які дозволяють забезпечити доступність та зручність для широкого кола користувачів. Веб-сайти, на відміну від десктоп чи мобільних додатків, не вимагають спеціальних установок та оновлень, що значно спрощує процес їх використання. Крім того, завдяки розвитку інтернет-технологій та підвищенню швидкості передачі даних, веб-додатки стають все більш потужними та функціональними. Сьогодні користувачі очікують від веб-додатків високої продуктивності, миттєвого відгуку та інтуїтивно зрозумілого інтерфейсу.

Розробка frontend частини медіа-сховища з використанням фреймворку Nuxt є важливим етапом у створенні ефективної системи управління медіа-контентом. Nuxt – це прогресивний фреймворк для створення універсальних Vue.js додатків, який забезпечує серверний рендеринг, покращену SEO-оптимізацію та високу продуктивність. Використання Nuxt дозволяє створювати швидкі, реактивні та зручні у використанні веб-додатки, що відповідають сучасним вимогам користувачів.

У ході роботи над даною кваліфікаційною роботою було проведено аналіз конкурентних рішень, які пропонують схожі функціональні можливості, вивчено їх переваги та недоліки. Це дозволило визначити ключові аспекти, які слід врахувати при розробці власної системи медіа-сховища, щоб забезпечити її

конкурентоспроможність та високу якість. Аналіз конкурентів включав детальне вивчення існуючих платформ, таких як One Drive чи FexNet, та інших сховищ, щоб зрозуміти, які функції є найбільш важливими для користувачів, та як можна покращити користувацький досвід.

Ключовим аспектом даної роботи є тестування та відлагодження frontend частини веб-сайту. Тестування є невід'ємною частиною процесу розробки програмного забезпечення, оскільки дозволяє виявити та виправити помилки, що можуть виникнути під час його використання. У роботі описано різні методи та інструменти тестування, які були застосовані для забезпечення високої якості кінцевого продукту. Зокрема, використовувалися автоматизовані тести, які дозволяють швидко і ефективно перевіряти функціональність веб-сайту після внесення змін.

Окрім цього, важливим завданням було забезпечення можливості подальшого масштабування та модернізації системи. Це включає в себе розробку архітектури, яка дозволить легко додавати нові функціональні можливості та забезпечувати безперебійну роботу системи при зростанні кількості користувачів та обсягу даних. Наприклад, враховувались можливості інтеграції з іншими сервісами, додавання нових форматів медіа-файлів та покращення алгоритмів пошуку та організації даних.

Таким чином, дана кваліфікаційна робота демонструє повний цикл розробки frontend частини медіа-сховища з використанням фреймворку Nuxt, від аналізу конкурентів та проектування архітектури до тестування та забезпечення можливості подальшого масштабування. Реалізація цього проекту дозволяє створити ефективну та зручну систему для зберігання та управління медіа-контентом, яка відповідатиме сучасним вимогам користувачів. Успішне виконання цієї роботи також сприятиме розвитку навичок та знань у сфері веб-розробки, що є важливим для подальшої професійної діяльності.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз конкурентних засобів

OneDrive - це хмарне сховище та сервіс файлообміну від Microsoft. Він дозволяє користувачам зберігати файли в хмарі, щоб мати до них доступ з будь-якого пристрою з підключенням до Інтернету [1]. OneDrive пропонує безкоштовне сховище для особистих користувачів, а також платні плани для бізнесу та користувачів Office 365.

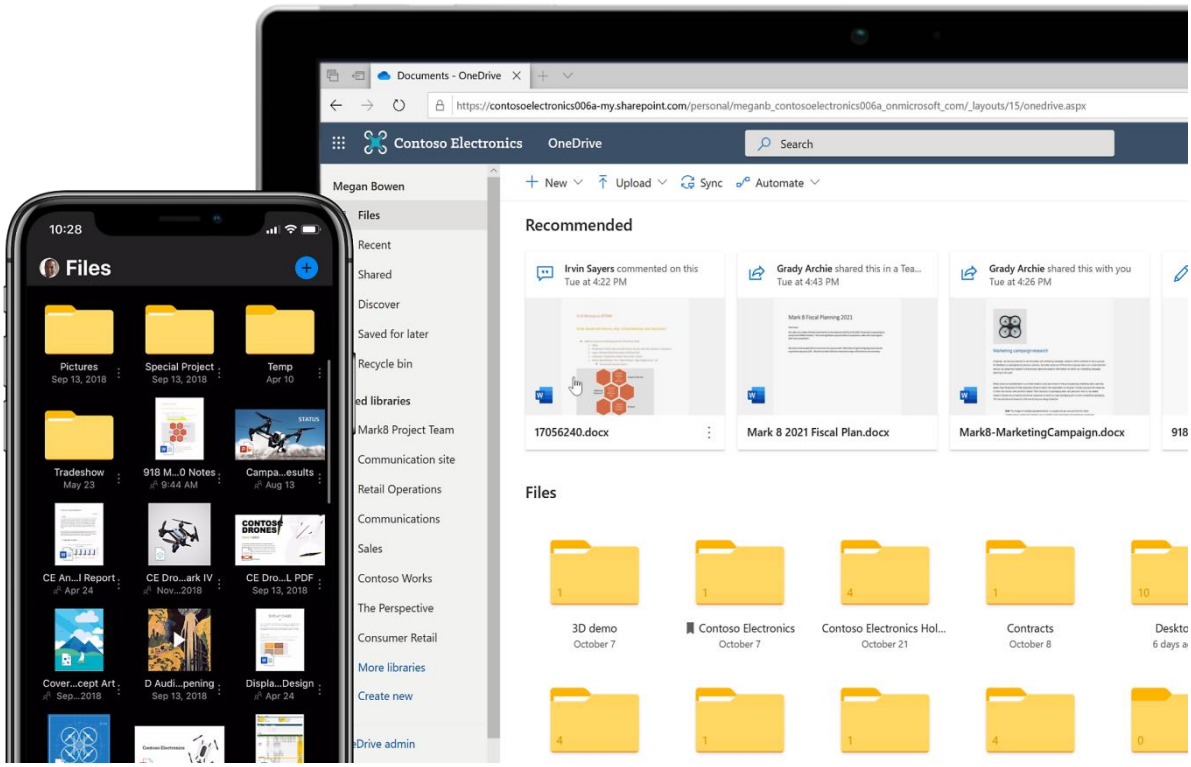


Рис. 1.1 – Вигляд сервісу Microsoft OneDrive [2].

OneDrive є популярним сервісом для зберігання та обміну файлами, особливо серед користувачів, які використовують продукти Microsoft. Його простий інтерфейс та інтеграція з іншими програмами роблять його зручним інструментом для роботи з даними в хмарі.

OneDrive - це надійне та зручне хмарне сховище від Microsoft, яке забезпечує ефективне зберігання, синхронізацію та обмін файлами. Завдяки інтеграції з пакетом офісних програм Microsoft Office та простому інтерфейсу,

OneDrive є ідеальним інструментом для користувачів, які шукають рішення для роботи з документами, фотографіями та іншими файлами в хмарі. Можливість резервного копіювання, спільного доступу та відновлення попередніх версій файлів робить OneDrive незамінним для особистого використання та бізнесу.

Іншим конкурентом можна розглянути сервіс FexNet. Fex.net – це український хмарний сервіс для обміну файлами, який дозволяє користувачам надсилати та отримувати файли будь-якого розміру. Має як безкоштовну версію, у якій 50 гігабайт файлів може зберігатися протягом семи днів, так і тарифні плани на різний обсяг пам’яті та безлімітний час зберігання.

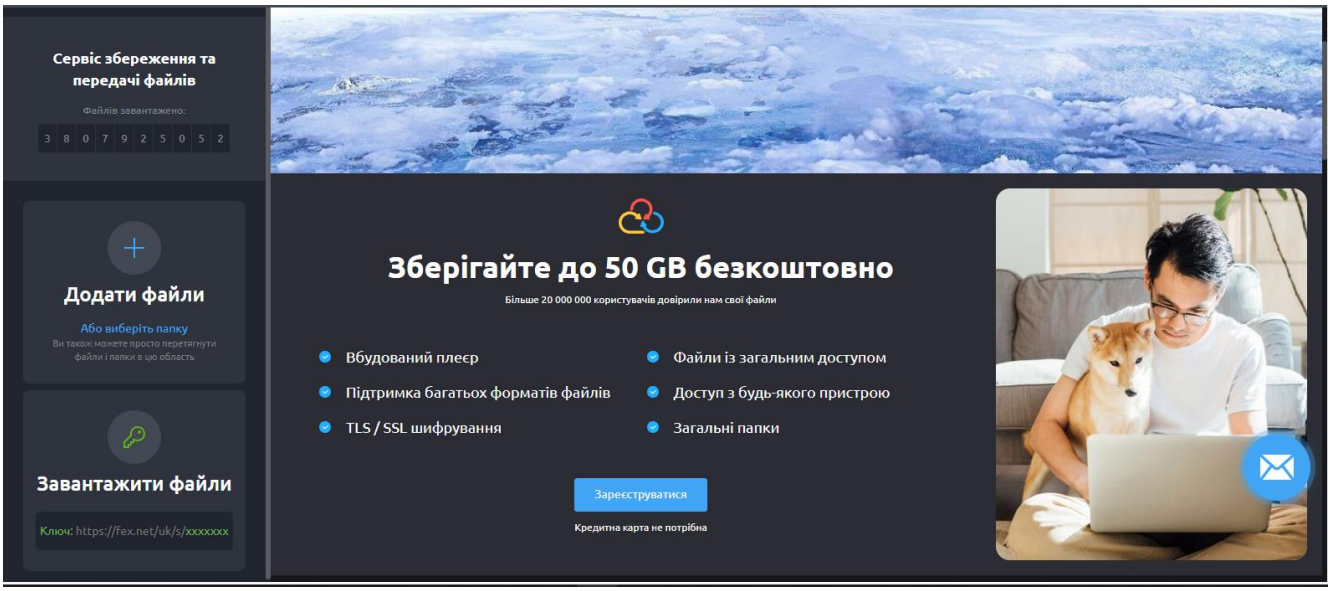


Рис. 1.2 – Вигляд сервісу FexNet

1.2 Порівняльний аналіз конкурентів

Проведення детального аналізу конкурентів є критично важливим для розробки будь-якого продукту або послуги, оскільки він дозволяє виявити сильні та слабкі сторони існуючих рішень на ринку. Такий аналіз допомагає зрозуміти потреби та очікування користувачів, визначити унікальні торгові пропозиції, які можуть виділити продукт серед конкурентів, а також уникнути помилок, які вже були зроблені іншими. Крім того, він надає цінну інформацію для стратегічного

планування та прийняття обґрунтованих рішень щодо функціональності, дизайну та маркетингу продукту.

Розглядаючи переваги сервісу OneDrive, то можна виділити такі:

- Інтеграція з Microsoft 365: OneDrive добре інтегрований з Microsoft 365 (раніше Office 365), що дозволяє легко зберігати та спільно редагувати документи Word, Excel, PowerPoint та інші файли прямо з хмари.
- Автоматичне збереження та синхронізація: файли автоматично зберігаються і синхронізуються на всіх пристроях, що дозволяє отримувати доступ до них з будь-якого місця та в будь-який час.
- Безпека та конфіденційність: OneDrive пропонує різні рівні безпеки, включаючи шифрування файлів та можливість встановлювати паролі на спільні посилання, що забезпечує захист даних користувачів.
- Гнучкі плани зберігання: є різні плани зберігання, включаючи безкоштовний план з обмеженим обсягом пам'яті та платні підписки з більшими обсягами сховища.
- Інтерфейс користувача: зручний і інтуїтивно зрозумілий інтерфейс, що дозволяє легко організовувати та керувати файлами.
- Розширені можливості для спільної роботи: можливість спільної роботи над документами в реальному часі, що підвищує ефективність комунікації та співпраці.

Хоча у таких великих корпорацій як Microsoft важко знайти недоліки програм та все ж це можливо. Ось недоліки сховища OneDrive:

- Обмежена безкоштовна пам'ять: безкоштовна версія OneDrive пропонує лише 5 ГБ пам'яті, що може бути недостатнім для користувачів з великими обсягами даних.
- Залежність від інтернет-з'єднання: для доступу до файлів потрібне стабільне інтернет-з'єднання. Хоча файли можуть бути доступні офлайн, синхронізація все одно потребує мережі.

- Приватність даних: хоча OneDrive пропонує різні заходи безпеки, деякі користувачі можуть мати занепокоєння щодо збереження своїх даних у хмарі, яка контролюється великою корпорацією.
- Обмежені функції для бізнес-користувачів у безкоштовній версії: деякі розширені функції доступні тільки в платних підписках, що може обмежувати можливості для малих бізнесів або окремих користувачів.
- Складність налаштувань для новачків: новим користувачам може знадобитися час для ознайомлення з усіма можливостями та налаштуваннями OneDrive.

Таким чином, OneDrive є потужним та зручним інструментом для зберігання та управління файлами, особливо для тих, хто вже використовує продукти Microsoft, але має деякі обмеження, які можуть бути критичними для певних користувачів.

Далі розглянемо переваги та недоліки FexNet. Переваги:

- Великий обсяг зберігання: FEX.NET пропонує значні обсяги пам'яті для зберігання даних, що дозволяє користувачам зберігати великі файли, включаючи мультимедійний контент.
- Зручний обмін файлами: можливість легко ділитися файлами через прямі посилання без необхідності реєстрації одержувача. Це спрощує процес обміну великими файлами.
- Високий рівень безпеки: FEX.NET пропонує опції для захисту файлів, включаючи встановлення паролів та термінів дії для посилань, що забезпечує додатковий рівень захисту даних.
- Мультимедійний плеєр: платформа дозволяє відтворювати аудіо та відео файли безпосередньо з хмари, що зручно для користувачів, які хочуть переглядати контент без необхідності завантаження.
- Доступність з будь-якого місця: FEX.NET забезпечує доступ до файлів з будь-якої точки світу, що дозволяє користувачам керувати своїми даними незалежно від місцезнаходження.

- Гнучкі тарифні плани: платформа пропонує різноманітні тарифні плани, включаючи безкоштовний доступ з певними обмеженнями та платні підписки з розширеними можливостями.

Недоліки:

- Обмежена функціональність у безкоштовній версії: безкоштовний доступ має обмеження щодо обсягу пам'яті та деяких функцій, що може бути недостатнім для користувачів з великими обсягами даних.
- Інтерфейс користувача: деякі користувачі можуть вважати інтерфейс менш інтуїтивним порівняно з іншими популярними хмарними сервісами.
- Обмежені інтеграції з іншими сервісами: FEX.NET може не мати такої широкої інтеграції з іншими популярними сервісами та інструментами, як інші великі хмарні платформи.
- Залежність від інтернет-з'єднання: як і інші хмарні сервіси, FEX.NET потребує стабільного інтернет-з'єднання для доступу та синхронізації файлів.
- Обмеження для бізнес-користувачів: деякі функції та можливості можуть бути обмеженими або менш зручними для корпоративних користувачів, що потребують розширених функцій для спільної роботи та управління даними.

FEX.NET є потужним інструментом для зберігання та обміну файлами, особливо підходить для зберігання великих мультимедійних файлів та забезпечення безпеки даних. Однак, платформа має деякі обмеження, які можуть бути достатньо проблемними для користувачів, які потребують інтеграції з іншими сервісами або розширених функцій для бізнесу.

1.3 Вибір напрямку дослідження

Фронтенд бібліотеки та фреймворки допомагають розробникам створювати інтерфейси користувача ефективніше та зручніше. Вибір конкретного фреймворку або бібліотеки залежить від багатьох факторів, включаючи вимоги проєкту, досвід розробника та специфічні особливості кожного інструменту. Тому при виборі засобів розробки важливо врахувати всі аспекти.

Для даного проєкту обрано Nuxt та Vue.js.

Vue.js є прогресивним фреймворком для побудови користувацьких інтерфейсів. Легкий для освоєння, з плавною кривою навчання, має гнучкість та може бути інтегрований поступово в існуючі проєкти. Vue.js має простий і зрозумілий синтаксис, що робить його доступним для розробників з різним рівнем досвіду. Документація Vue.js є дуже детальною та зрозумілою. Як і React, Vue.js використовує компонентний підхід, що дозволяє створювати багаторазові та легко підтримувані частини коду. Він може бути використаний як для маленьких компонентів у великому проєкті, так і для створення повноцінних односторінкових додатків.

Рис. 1.3 – Логотип Vue.js

Nuxt.js додає ще більше можливостей, таких як серверний рендеринг (SSR), генерація статичних сайтів та управління маршрутизацією. Він є мета-фреймворком для Vue.js, який спрощує створення універсальних (із серверним рендерингом) додатків. Nuxt.js має багато вбудованих функцій, таких як автоматична генерація маршрутів, покращене управління станом, обробка метаданих сторінок, а також підтримка статичної генерації сайтів (SSG). Важливо зазначити, що серверний рендеринг значно покращує SEO-оптимізацію та швидкість завантаження сторінок, що є критичним для користувацького досвіду та рейтингу сайту в пошукових системах.

Рис. 1.4 – Логотип Nuxt

Vue.js та Nuxt.js мають велику та активну спільноту розробників, що забезпечує наявність багатьох ресурсів, бібліотек та інструментів для розробки. Це значно полегшує вирішення потенційних проблем, оскільки є можливість знайти рішення або поради від інших розробників. Крім того, активна спільнота сприяє регулярному оновленню та вдосконаленню фреймворків, що забезпечує їх відповідність сучасним вимогам та стандартам.

Вибір Nuxt та Vue як основних інструментів для фронт-енд розробки має багато переваг завдяки їх легкості освоєння, гнучкості, компонентному підходу та великій спільноті. Ці інструменти дозволяють швидко та ефективно створювати сучасні, динамічні веб-додатки з мінімальними зусиллями. Більш того, їх використання сприяє створенню продуктивних, легко масштабованих та підтримуваних рішень, що є важливим для успіху будь-якого сучасного веб-проєкту.

1.4 Методологія розробки

Методи розробки програмного забезпечення (ПЗ) - це стратегії, підходи та процеси, які використовуються для розробки високоякісного програмного забезпечення. Ці методи допомагають забезпечити успішне завершення проекту та відповідність вимогам користувача. Ось деякі з найпопулярніших методів розробки програмного забезпечення:

- Каскадна модель: це традиційний метод, в якому розробка відбувається послідовно, кожен етап (аналіз, проектування, реалізація, тестування, впровадження) виконується після завершення попереднього.
- Розробка по вимогах: цей підхід акцентується на визначенні та вдосконаленні вимог користувача протягом усього циклу розробки.
- Методологія Agile: Agile використовує ітеративний підхід до розробки, дозволяючи команді швидко адаптуватися до змін. Складається з набору підходів, таких як Scrum, Kanban, Extreme Programming (XP) тощо.
- DevOps: DevOps поєднує розробку (Dev) та операції (Ops), створюючи єдиний процес, який полегшує автоматизацію та прискорення поставки ПЗ.
- Spiral Model (Спіральна модель): Це ітеративна модель, яка поєднує в собі елементи каскадної моделі та ітеративних методів розробки.
- Lean Software Development (Підхід Lean у розробці ПЗ): Цей підхід спрямований на мінімізацію витрат та максимізацію цінності для користувача шляхом усунення зайвих процесів та оптимізації робочих процесів.

Кожен метод має свої переваги та недоліки, і вибір методу зазвичай залежить від конкретного проекту, вимог клієнта та умов роботи команди.

Для реалізації проекту було обрано методологію Agile. Agile - це набір цінностей та принципів, спрямованих на покращення процесу розробки програмного забезпечення. Основні принципи Agile ставлять індивідуальність та взаємодію в команді вище, ніж процеси та інструменти; акцент на робочому

продукті важливіше за вичерпні документи; співпрацю з клієнтом вважається цінішою ніж угода про обслуговування; та готовність реагувати на зміни більш важлива, ніж дотримання плану.

Обрання Agile дозволило ефективно керувати проектом, реагувати на зміни вимог та відгуки клієнтів, а також швидко впроваджувати нові функції. Ітеративний підхід розробки, характерний для Agile, дозволяє швидко створювати та випробовувати прототипи, залучати клієнтів до процесу розробки та забезпечувати високу якість продукту.

Рис. 1.5 – схема методології Agile

1.5 Формування вимог до системи

Вимоги до системи (або вимоги до програмного забезпечення) - це опис того, як система повинна працювати або які функції вона повинна виконувати, щоб задовольнити потреби користувачів чи бізнесу. Вимоги бувають різними за призначенням та характером. Як правило їх поділяють на функціональні та нефункціональні.

Функціональні вимоги – це конкретні функції або сервіси, які система повинна забезпечити. Наприклад, можуть бути вимоги до можливості користувача створювати облікові записи або до можливості системи генерувати звіти.

Нефункціональні вимоги – це вимоги, що описують якісні аспекти системи, такі як продуктивність, безпека, надійність, зручність використання та інші. Наприклад, вимоги до часу відгуку системи або до рівня захисту від несанкціонованого доступу.

Іноді виділяють інші типи вимог, якщо їх необхідно виокремити за певними ознаками.

Функціональні вимоги:

1. Реєстрація користувачів за адресою електронної пошти та паролем.
2. Надання різних прав доступу до контенту залежно від ролі (адміністратор, редактор, користувач).
3. Керування ролями адміністратором через сторінку адміністрування.
4. Завантаження та видалення контенту редактором.
5. Пошук/фільтрація/перегляд контенту та скачування файлів користувачем.
6. Перевірка файлів на предмет графічного, аудіо чи відеоформату при завантаженні.
7. Зберігання графічних файлів у форматах jpeg, tiff, raw, bmp, png, gif, pcx; відео у форматах avi, mpeg, mov, asf, wmv, 3gp, flv, rm, vob; аудіо у форматах wav, aiff, au, pcm, mp3, aac, wma.
8. Створення контенту з подією, датою події, описом, мітками-тегами, даними про автора.

Нефункціональні вимоги:

1. Інтерфейс адміністрування повинен бути зручним та інтуїтивно зрозумілим.
2. Система повинна працювати стабільно та швидко навіть при великому обсязі даних.
3. Забезпечення безпеки даних та доступу до системи.
4. Сумісність з різними браузерами та пристроями.
5. Забезпечення можливості розширення системи та динамічного збільшення дискового простору для зберігання контенту.

2 РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ

Проєктування є одним із найважливіших етапів розробки програмного забезпечення, що визначає його майбутню функціональність, ефективність та зручність використання. Це процес, що охоплює створення моделей, які відображають основні аспекти системи, її архітектуру та взаємодію між компонентами. Метою проєктування є створення чіткого та зрозумілого плану, за яким можна реалізувати програмне забезпечення, що відповідає вимогам користувачів і бізнесу.

Проєктування включає в себе аналіз вимог, визначення основних елементів системи та їх взаємозв'язків, а також розробку відповідних моделей. На початковому етапі розробляється модель предметної області, що дозволяє краще зрозуміти контекст та специфіку системи. Наступним кроком є розробка бізнес-моделі, яка описує взаємодію користувачів із системою та основні бізнес-процеси.

Архітектурне проєктування є критично важливим етапом, під час якого створюється структура системи, визначаються основні компоненти та їх взаємодія. Діаграми класів, варіантів використання та інші UML-діаграми допомагають візуалізувати та спланувати архітектуру системи. Це забезпечує структурованість та гнучкість програмного забезпечення, полегшуючи його подальшу реалізацію та підтримку.

Проєктування також включає врахування нефункціональних вимог, таких як масштабованість, продуктивність, безпека та зручність використання. Всі ці аспекти враховуються на етапі проєктування, щоб гарантувати, що кінцевий продукт буде не тільки функціональним, але й ефективним у використанні.

Проєктування є фундаментальною частиною процесу розробки програмного забезпечення, що визначає його якість та успішність. Якісне проєктування дозволяє створити системи, що відповідають потребам користувачів, легко підтримуються та адаптуються до змін, забезпечуючи тривалу життєздатність програмного забезпечення.

2.1 Архітектура системи

Мікросервісна архітектура (Microservices Architecture) є підходом до розробки програмного забезпечення, при якому велика монолітна система розбивається на набір невеликих, незалежних сервісів. Кожен з цих сервісів виконує чітко визначену функцію і може розроблятися, розгортатися та масштабуватися окремо від інших.

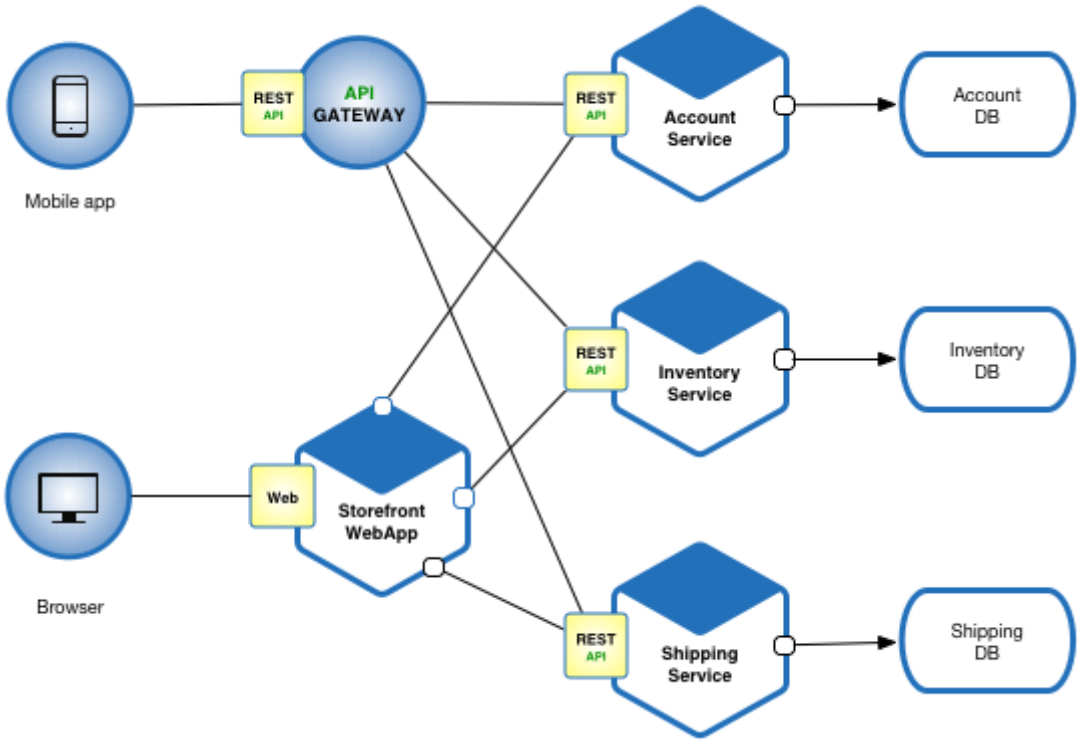


Рис. 2.1 – схема мікросервісної архітектури

Основні принципи мікросервісної архітектури полягають у декомпозиції системи на сервіси, незалежному розгортанні кожного з них, забезпеченні масштабованості та використанні різних технологій для різних сервісів. Сервіси взаємодіють один з одним через стандартизовані інтерфейси, зазвичай за допомогою протоколів HTTP/HTTPS і REST або повідомлень у форматах, таких як JSON або XML.

Серед переваг мікросервісної архітектури можна виділити гнучкість у розробці та розгортанні, покращену масштабованість, стійкість до збоїв, а також

легкість у підтримці та оновленні. Незалежність сервісів дозволяє командам працювати над різними частинами системи одночасно, без конфліктів та затримок. Можливість масштабувати лише ті сервіси, які цього потребують, дозволяє ефективніше використовувати ресурси. Відмова одного сервісу не призводить до краху всієї системи, інші сервіси продовжують працювати, що підвищує загальну надійність системи. Незалежні сервіси простіше підтримувати та оновлювати, оскільки зміни в одному сервісі не впливають на інші.

Однак мікросервісна архітектура має і свої недоліки. Складність у розробці та управлінні зростає, оскільки розбиття системи на багато сервісів ускладнює її розробку та управління, вимагаючи додаткових інструментів для моніторингу та управління. Взаємодія між сервісами може стати проблемним місцем, особливо при великому обсязі трафіку або при неправильній організації API. Тестування мікросервісних систем є складнішим, оскільки потребує тестування як окремих сервісів, так і їх взаємодії. Потреба у злагодженій роботі різних команд може викликати організаційні труднощі, особливо у великих компаніях.

Мікросервісна архітектура пропонує багато переваг, таких як гнучкість, масштабованість та стійкість до збоїв. Однак її впровадження вимагає значних зусиль, додаткових інструментів та уваги до управління комунікацією між сервісами. Вибір мікросервісної архітектури має базуватися на ретельному аналізі вимог проєкту та можливостей команди.

2.2 Розробка бізнес моделі

Unified Modeling Language (UML) – це стандартна мова для моделювання систем, яка використовується для візуалізації, специфікації, конструювання та документування артефактів програмного забезпечення. UML надає широкий спектр діаграм для моделювання різних аспектів системи. Основні типи діаграм UML включають діаграми класів, об'єктів, компонентів, послідовностей, станів, активностей, комунікацій та варіантів використання.

Діаграми варіантів використання є одним із найпоширеніших типів діаграм UML і використовуються для моделювання функціональних вимог до системи. Вони показують взаємодію між користувачами (акторів) та системою, а також описують, як користувачі використовують систему для досягнення своїх цілей.

Діаграми варіантів використання є потужним інструментом для моделювання функціональних вимог системи. Вони допомагають розробникам і зацікавленим сторонам зрозуміти, як система повинна працювати з точки зору користувачів, і які функції вона повинна надавати. Вони також служать основою для створення інших видів діаграм UML, таких як діаграми класів та послідовностей, які деталізують внутрішню структуру і поведінку системи.

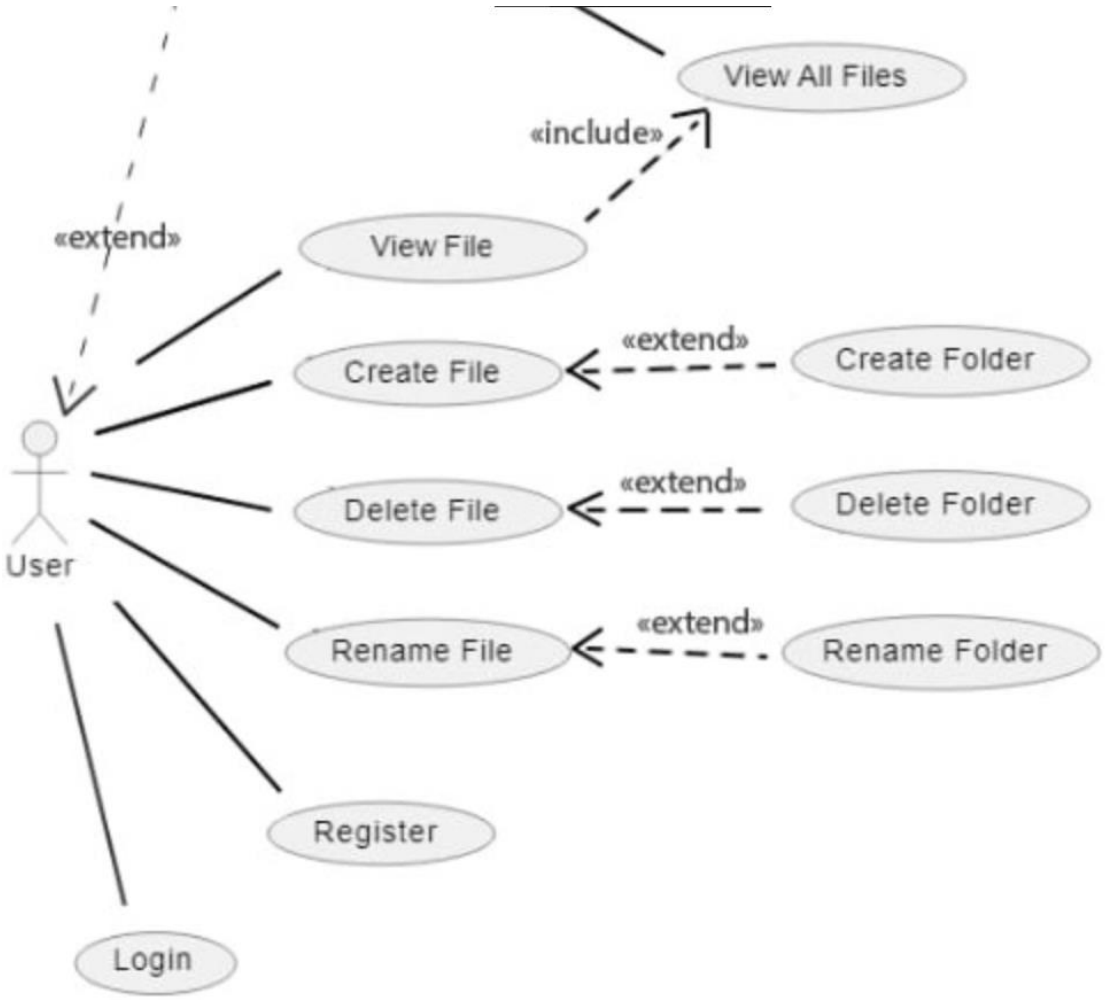


Рис. 2.2 – фрагмент діаграми варіантів використання для актора Користувач

На діаграмі представлено два актори – Користувач та Адміністратор який наслідується від користувача, оскільки має такий самий функціонал як і він, та крім цього – додаткові можливості.

Користувач може реєструватися та здійснювати вхід у систему, створювати файли, перейменовувати їх та видаляти. Також користувач може створювати перейменовувати та видаляти папки і відповідно додавати у папки файли .

Адміністратор у свою чергу може виконувати абсолютно ті ж самі дії, але окрім цього він має можливість блокувати інших користувачів, призначати інших адміністраторів або забирати у них це право та переглядати усі можливі файли.

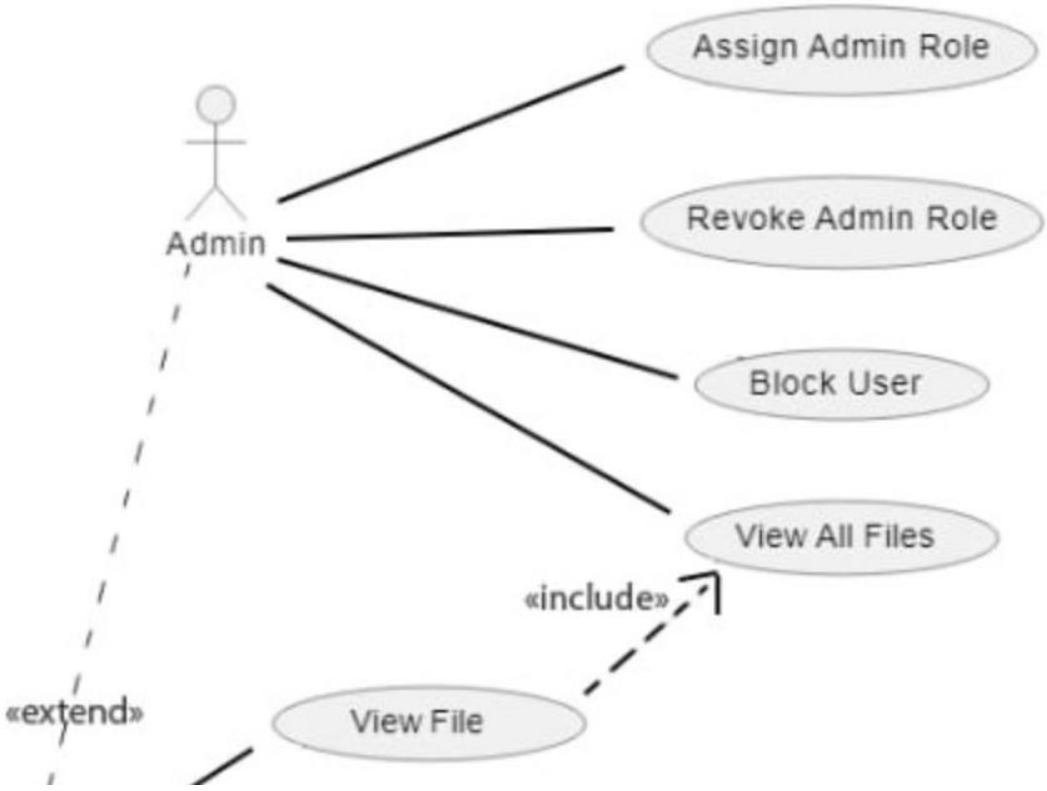


Рис. 2.3 – Фрагмент діаграми варіантів використання для актора Адміністратор

Повну діаграму варіантів використання можна переглянути в Додатку А

2.3 Проєктування архітектури

Для бачення загального вигляду системи необхідно зрозуміти які об’єкти та зв’язки між ними можуть бути. Для цього створено діаграму об’єктів. Діаграми об'єктів є важливим інструментом для моделювання, перевірки, документування і комунікації при розробці програмного забезпечення. Вони допомагають розробникам зрозуміти і візуалізувати взаємодію об'єктів у системі, забезпечуючи більш ефективний процес розробки і підтримки програмного забезпечення.

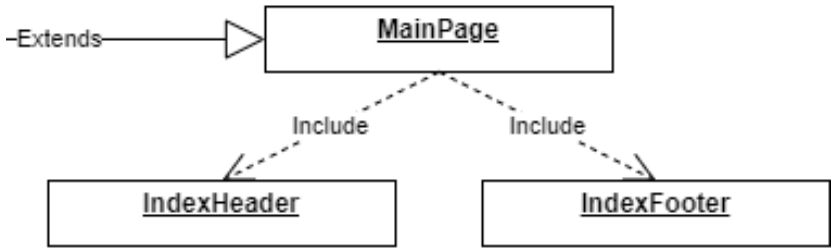


Рис. 2.4 – Фрагмент діаграми об’єктів 1

Повну діаграму можна переглянути в Додатку Б

Діаграма відображає структуру об'єктів у фронтенд частині медіасховища, створену з використанням Vue.js.

App - це головний макет, який є основою всього фронтенду. Усі інші компоненти є частинами або дочірніми компонентами App. MainPage є компонентом, що наслідується від App і є основним макетом для сторінок. Він включає в себе компоненти IndexHeader та IndexFooter. IndexHeader- компонент, що відповідає за заголовок на основній сторінці, IndexFooter - компонент, що відповідає за нижню частину (футер) основної сторінки.

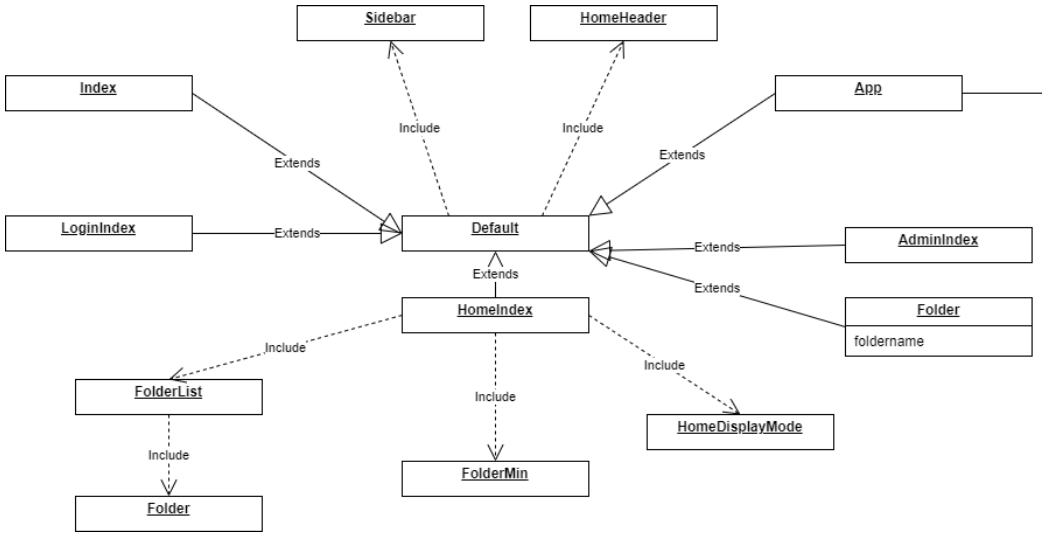


Рис. 2.5 – Фрагмент діаграми об’єктів 2

Default - це також макет, який наслідується від App. Використовується для інших сторінок, які не є частиною основного макету. Index наслідується від Default і є головною сторінкою вебсайту. LoginIndex - компонент для сторінки входу, також наслідується від Default. AdminIndex - компонент для адміністративної панелі, наслідується від Default.

FolderList відповідає за відображення списку папок. Він включає в себе компонент Folder, що представляє окрему папку та має параметр foldername, який використовується для доступу до конкретної папки. FolderMin - компонент, що відображає іконку папки. HomeDisplayMode – це кнопка, що змінює вигляд файлів або папок (списком чи плиткою). HomeIndex: Компонент, що наслідується від Default. Він включає в себе компонент Sidebar і HomeHeader.

Sidebar: Бокове меню, що забезпечує навігацію по сайту. HomeHeader - заголовок головної сторінки.

Таким чином, діаграма показує, як компоненти Vue.js взаємодіють один з одним у рамках медіасховища. Всі компоненти організовані в ієрархічну структуру, де кожен компонент має свою роль і взаємодіє з іншими компонентами, забезпечуючи функціональність та зручність користування додатком.

2.4 Послідовності процесів у системі

Ще одним способом візуалізації роботи системи є діаграми послідовностей. Діаграми послідовностей використовуються для моделювання динамічних аспектів системи. Вони допомагають візуалізувати, як об'єкти взаємодіють між собою протягом певного періоду часу. Діаграми послідовностей зосереджуються на порядку обміну повідомленнями між об'єктами і надають детальне уявлення про взаємодію компонентів системи.

Діаграми послідовностей допомагають зрозуміти і уточнити вимоги до системи, показуючи, як користувачі або інші системи взаємодіють з нею для виконання певних завдань. Вони використовуються для проектування логіки взаємодії між об'єктами, забезпечуючи чітке уявлення про послідовність викликів методів і обмін повідомленнями. Діаграми послідовностей служать як засіб документування поведінки системи, що допомагає у підтримці, оновленні та розширенні системи в майбутньому. Вони допомагають у розробці тестових сценаріїв, забезпечуючи чітке уявлення про послідовність дій, які потрібно перевірити. Також діаграми послідовностей покращують комунікацію між членами команди розробників, надаючи візуальне уявлення про взаємодію об'єктів у системі. Завдяки діаграмам послідовностей розробники можуть краще зрозуміти логіку роботи системи і забезпечити її коректне функціонування.

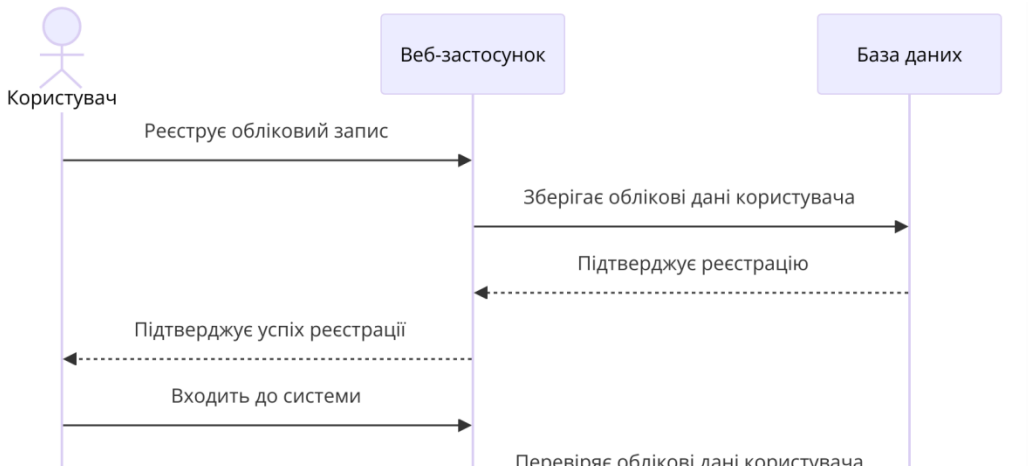


Рис. 2.6 – Фрагмент діаграми послідовностей

На цій діаграмі розміщений один актор Користувач та об'єкти з якими він може взаємодіяти, а саме Веб-застосунок та База даних. Повну діаграму послідовностей можна переглянути в Додатку В.

В першу чергу будь-який користувач повинен зареєструвати обліковий запис. Тому користувач у веб-застосунку реєструє обліковий запис, реєстраційні дані зберігаються на базі даних і вона надсилає підтвердження реєстрації, для користувача надходить повідомлення про успіх реєстрації. Після цього користувач може увійти до системи. Веб-застосунок надсилає запит на базу даних щоби перевірити облікові дані користувача, база даних повертає токен аутентифікації і веб-застосунок надає доступ користувачу.

Потім користувач створює нову папку, веб-застосунок зберігає інформацію про папку на базі даних та підтверджує користувачу, що папка створена. Після цього користувач може завантажити файл. Застосунок зберігає файл на базі даних, отримує підтвердження збереження файлу, зберігає метадані цього ж файлу і знову отримує підтвердження про збереження, відповідно до користувача надходить сигнал про завантаження файлу.

А коли користувач захоче завантажити файл із сайту, то веб-застосунок отримує мета дані файлу та отримує файл, а потім файл з метаданими надсилає користувачеві.

2.5 Розробка системи

Більшість фронтенд частини проєкту складається з таких основних елементів як сторінки, компоненти, мутації, query, а також лейаути. Нижче наведена базова структура всіх vue-файлів (компонентів, лейаутів та сторінок) які використовуються у проекті. Всі файли складаються з трьох частин: скрипти, темплейт і стилі.

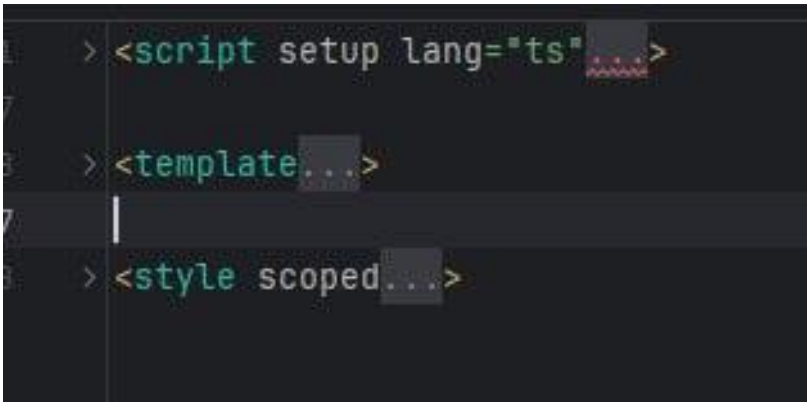


Рис. 2.7 - Базова структура

Для детальнішого огляду структури проекту розглянуто приклади кожної із основних частин проєкту, включаючи шаблони, логіку компонентів, та стилі, а також способи їх взаємодії та інтеграції в єдине ціле.

Самі файли сторінок є сторінками сайту, наприклад головна сторінка, адміністративна сторінка, сторінка папки чи сторінка списку папок.

```
<PContextMenu ref="menu" :model="contextMenuObjects">
  <template #itemicon="{item}">
    <Icon :name="item.icon" v-if="item.icon" class="mr-2"/>
  </template>
</PContextMenu>
<div class="px-5 xl:pl-16"...>
<div class="selecto_container py-2 pl-3 pr-5 xl:pl-14 ml-2 duration-400 relative transition-all bg-transparent
    :class="{ 'bg-secondary/20 !ring-2 ring-secondary': isOverDropZone }"
    @drop="onDragDrop" @dragover.prevent @dragenter.prevent
>
  <VueSelecto...>
  <FoldersList :folders="initialData?.folders || null" class="mb-5"
               :context-callback="onRightClick"
               :selected-folders-ids="selectedFoldersIds"
               @selectFolder="onFolderSelect"
               @dragStart="dragStart"
               @dragEnd="dragEnd"
               @onDrop="onFolderDrop"
               :is-dragging="isDragging"
  />
  <div class="text-base mb-2">Файли</div>
</div>
<div class="drag_image" ref="dragImage"></div>
```

Рис. 2.8 – template сторінки файлової системи

На цьому рисунку зображено фрагмент шаблону сторінки файлової системи. Тут можна побачити HTML розмітку сторінки, а також включення компонента в сторінку та виклик скриптів за допомогою оператора «@».

```
Show usages  ± psina home *
const onFolderSelect = (event: any, id: number) => {
  const element = (event.target.classList.contains('folder_min')) ?
      event.target : event.target.closest('.selecto_target');
  console.log("onFolderSelect", event.target);
  const boundingRect = element.getBoundingClientRect();
  if (selectedFoldersIds.value.includes(id)) {
    selectedFoldersIds.value = selectedFoldersIds.value.filter((folderId) => folderId !== id);
    selectoRef.value.$_selecto.selectedTargets = selectoRef.value.$_selecto.selectedTargets.filter((el: any)
        => parseInt(el.dataset.id) !== id);
    selectedFoldersCoords.value = selectedFoldersCoords.value.filter((folder) => folder.id !== id);
  } else {
    if (!selectMultiple.value) {...} else {
      selectedFoldersIds.value.push(id);
      selectoRef.value.$_selecto.selectedTargets.push(event.target);
      selectedFoldersCoords.value.push({id: id...});
    }
  }
}
```

Рис. 2.9 – Приклад скрипту

Тут продемонстровано приклад скриптів. В даному випадку це функція onFolderSelect. Вона приймає в себе івент за допомогою якого була викликана та id папки. Ця функція відповідає за виділення папок на сторінці та оновлення відповідних списків та об'єктів з їх координатами і розмірами.

```
<style scoped>
.raise-enter-active, .raise-leave-active {
  transition: all 0.3s ease;
}

.raise-enter-from, .raise-leave-to {
  transform: translate(-50%, 100%);
  opacity: 0;
}


</style>
```

Рис. 2.10 – Стилі

Для написання стилів зазвичай на проекті використовується бібліотека talewind, але деякі стилі написані вручну на SCSS як показано на рисунку вище.

Лейаути – це макети сторінок, від них наслідуються всі або більшість сторінок на сайті і в них вставлені основні характеристики сторінок: загальні налаштування, змінні, кольори і т. д.

```
<script setup lang="ts"> Show component usages
useHead( input: {
  bodyAttrs: {
    class: 'bg-light'
  }
})
</script>

<template>
  <div class="flex overflow-hidden w-full h-full bg-background text-sm">
    <Sidebar/>
    <div class="content overflow-y-auto flex flex-col flex-grow custom_scrollbar">
      <main class="flex-grow py-5">
        <HomeHeader/>
        <slot/>
      </main>
    </div>
  </div>
</template>

<style scoped>

</style>
```

Рис. 2.11 – Приклад лейаута

На рисунку представлено приклад лейаута для сторінок файлової системи проекту. В ньому продемонстровано як працює лейаут а в ньому є тег slot замість цього слоту будуть розміщатися код тієї сторінки яка наслідується від цього лейаута. Також в ньому включено елементи homeheader і sidebar(шапка сайту і бокове меню) – ці елементи будуть на всіх сторінках які наслідуються від цього лейаута.

Компоненти схожі до сторінок, але не є сторінками. Вони є частинками, певними елементами, що вставляються всередину сторінок. Наприклад, файл для папки чи іконки, хедер. Їх часто використовують тоді, коли потрібно відтворити одну і ту саму частину сторінки на різних сторінках, замість дублювання коду, або як виокремлення певних елементів для зручності роботи.

Приклад компонентів можна переглянути в Додатку Г.

Досить важливим елементом фронт-частини є query-файли. Ці файли слугують для того, щоб отримувати інформацію з бекенду, а саме з бази даних. Це можуть бути об'єкти або інформація про них і т. п.

```
query folderContent($link: String!) {
    folder(link: $link) {
        id
        link
        name

        childrens {
            edges {
                node {
                    id
                    link
                    name
                    createdAt
                    updatedAt
                }
            }
        }
        breadcrumbs {
            id
            link
            name
        }
    }
}
```

Рис. 2.12 – Код query запиту

Приклад query елемента в даному випадку демонструє отримання папки і її контенту використовуючи фільтр по полю link, тобто отримується конкретно та папка в якої лінк співпадає з рядком переданим в запит. Також в самому query запиті описано які саме поля потрібно отримати.

Мутації дуже схожі до query-файлів, але замість отримання інформації з бекенду в мутаціях передається запит на бекенд щоб змінити певний об’єкт або створити новий у базі з фронтенду.

```
mutation tokenAuth($username: String!, $password: String!) {
  tokenAuth(username: $username, password: $password) {
    token
    refreshToken
    refreshExpiresIn
    payload
    errors {
      field
      messages
    }
  }
}
```

Рис. 2.13 – Приклад мутації

На зображенні мутація, яка використовується для того щоб увійти в профіль користувача передається username і пароль і якщо є користувач з таким username і паролем то відбувається вхід в систему.

2.6 Тестування системи

Тестування програмного забезпечення (ПЗ) є ключовим етапом у розробці, спрямованим на забезпечення якості та надійності кінцевого продукту. Метою тестування є виявлення помилок, дефектів або вразливостей у програмному коді, щоб гарантувати його відповідність вимогам користувачів та специфікаціям проекту.

Сучасні інструменти для автоматизації тестування допомагають підвищити ефективність процесу. Деякі з популярних інструментів включають Selenium для веб-тестування, JUnit для модульного тестування Java, TestNG для тестування на базі Java, Jest для тестування JavaScript, та багато інших.

Якісне тестування програмного забезпечення допомагає виявити та виправити помилки на ранніх етапах розробки, що дозволяє уникнути додаткових витрат та зусиль на їх виправлення у майбутньому. Це також забезпечує високу

якість кінцевого продукту, підвищує задоволеність користувачів та зменшує ризики, пов’язані з можливими збоями або вразливостями системи.

2.6.1 Функціональне тестування

Функціональне тестування - це перевірка функціональних вимог. Тобто чи відповідає система свою функціональним вимогам і чи може їх виконати. Виконане тестування системи складено у вигляді тест-кейсів:

1. Вхід на сайт

Попередньо зареєстрований користувач повинен відкрити сайт та ввести у відповідному вікні своє користувацьке ім’я та пароль і натиснути кнопку «Авторизуватись». Як результат здійснено вхід у систему.

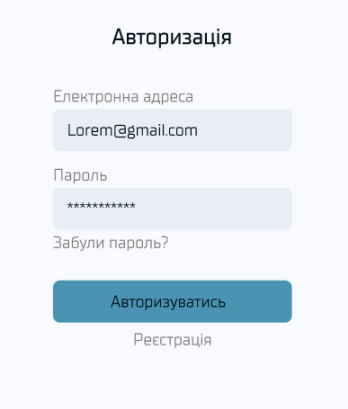


Рис. 2.14 – Авторизація користувача

Для перевірки цієї та інших функцій було використано спеціальні скрипти, а вже потім ручна перевірка шляхом виконання дій користувача.

```python
class Command(BaseCommand):
    Leonid *
    def handle(self, *args, **options):
        client = Client()
        response = client.post( path: "http://127.0.0.1:8000/admin/login/?next=/admin/",
                                data: {"username": "admin", "password": "admin"})
        if response.status_code == 200:
            print("Запити виконано успішно")
        else:
            print("Помилка при виконанні запиту")

        response = client.get("http://127.0.0.1:8000/admin/product/category/")
        print(response.content)
```

Рис. 2.15 – Код для функціонального тестування

2. Додавання файлу

Користувач, який здійснив вхід у систему на головній сторінці натискає праву кнопку мишки після чого із контекстного меню вибирає пункт «Завантажити». У вікні яке відкриється обирає файл який потрібно завантажити та підтверджує вибір. У результаті на сайт завантажуються обрані файли.

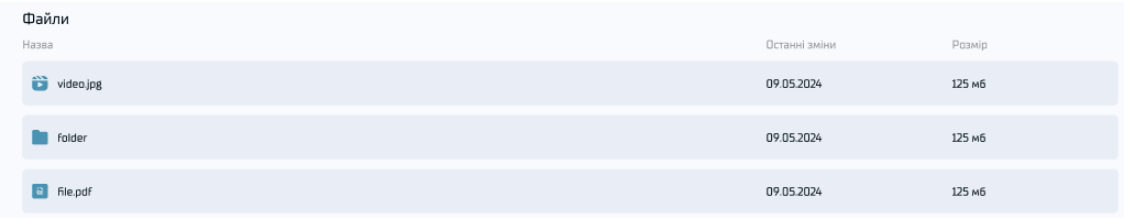


Рис. 2.16 – Завантажені файли

3. Зміна статусу користувача

Дана функція належить лише адміністраторам, отже саме він відкриває адміністративну панель після чого навпроти імені користувачів може обрати статус, який їм надаватиметься. Після підтвердження змін обрані ролі закріплюються за користувачами.

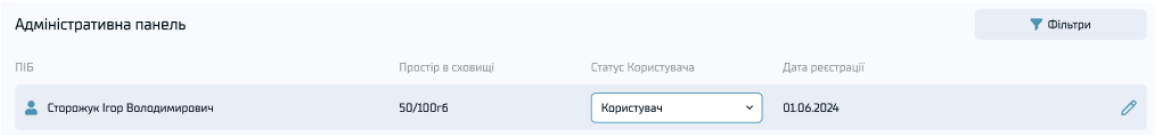


Рис.2.17 – Адміністративна панель

2.6.2 Навантажувальне тестування

Навантажувальне тестування є невід'ємною частиною процесу тестування програмного забезпечення, яке має на меті виявити проблеми та оцінити продуктивність та стійкість системи при великому обсязі навантажень. Цей процес включає імітацію великої кількості вхідних запитів, які можуть бути створені штучно або використовуючи реальні дані та сценарії використання. В результаті тестування виявляються слабкі місця системи та визначаються її максимальні межі функціонування.

Для виконання навантажувального тестування було обрано спеціалізовано програмне забезпечення – Blazemeter. Це платформа для тестування продуктивності, яка дозволяє розробникам і тестувальникам оцінювати та покращувати продуктивність веб-додатків, веб-сайтів і API. Вона забезпечує можливість проведення масштабних тестів навантаження та стрес-тестування в хмарному середовищі.

BlazeMeter дозволяє створювати, запускати та аналізувати тести продуктивності для різних типів додатків. Це допомагає виявити вузькі місця та оптимізувати систему для роботи під високим навантаженням. Цей сервіс підтримує інтеграцію з популярними інструментами для тестування продуктивності, такими як Apache JMeter, Gatling, Selenium і багатьма іншими. Це дозволяє користувачам використовувати знайомі інструменти для створення тестів. Платформа дозволяє легко масштабувати тести навантаження, симулюючи сотні тисяч або навіть мільйони користувачів з різних географічних регіонів. Це дає можливість оцінити, як система поводиться під високим навантаженням. BlazeMeter надає детальні звіти та аналітику щодо результатів тестування. Це включає в себе графіки, діаграми та метрики, що допомагають зрозуміти продуктивність системи та виявити потенційні проблеми. Платформа підтримує інтеграцію з інструментами безперервної інтеграції та доставки (CI/CD), такими як Jenkins, Bamboo, TeamCity тощо. Це дозволяє автоматизувати тестування продуктивності в процесі розробки та забезпечити швидке виявлення проблем.

BlazeMeter працює в хмарному середовищі, що дозволяє уникнути необхідності в налаштуванні та обслуговуванні власної інфраструктури для проведення тестів. Це знижує витрати та спрощує процес тестування.

Переваги сервісу:

- Інтуїтивно зрозумілий інтерфейс та можливість швидкого створення тестів без необхідності глибоких технічних знань.
- Підтримка різних інструментів для тестування та інтеграція з існуючими процесами розробки.
- Потужні інструменти для аналізу результатів тестів, що допомагають виявити та усунути проблеми з продуктивністю.
- Можливість проведення тестів під великим навантаженням з різних географічних регіонів.

Недоліки BlazeMeter:

- Використання BlazeMeter може бути дорогим, особливо для великих команд або підприємств з великими потребами в тестуванні.
- Оскільки BlazeMeter працює в хмарі, якість та стабільність тестування залежать від надійності інтернет-з'єднання

BlazeMeter є потужним інструментом для тестування продуктивності, який надає широкі можливості для розробників та тестувальників. Його можливість масштабування, інтеграція з популярними інструментами та детальна аналітика роблять його цінним у процесі розробки програмного забезпечення. Однак, вартість та залежність від інтернет-з'єднання можуть бути стримуючими факторами для деяких команд.

Тестування на BlazeMeter проходить через кілька основних етапів, які включають підготовку тестових сценаріїв, запуск тестів, моніторинг виконання та аналіз результатів. BlazeMeter підтримує сценарії, створені в Apache JMeter, популярному інструменті для тестування продуктивності. Ви можете створити JMeter-скрипт локально і завантажити його на BlazeMeter. Також він пропонує можливість створення сценаріїв за допомогою їхнього інтерфейсу Scriptless. Це

дозволяє створювати тести, використовуючи лише графічний інтерфейс без необхідності писати код.

Після цього іде налаштування: введення URL-адреси, визначення кількості віртуальних користувачів (VU), тривалість тесту, розподіл користувачів по різних географічних регіонах та інші параметри.

Після закінчення тестування отримуємо результати у вигляді списку та графіків які можна змінювати для відображення необхідної інформації.

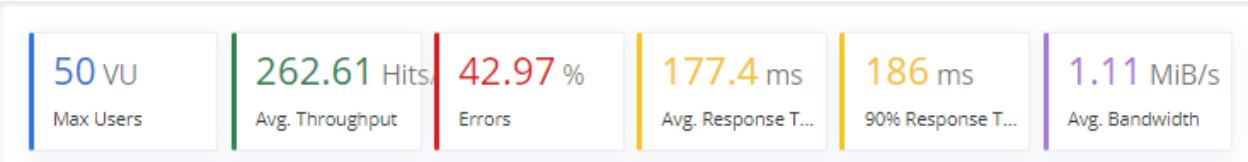


Рисунок 2.18 – Висновки тестування.

Розглянувши таймлайн тестувння можна відслідкувати як і коли змінювалась відповідь сайту залежно від кількості користувачів.

Таким чином маємо наступні результати:

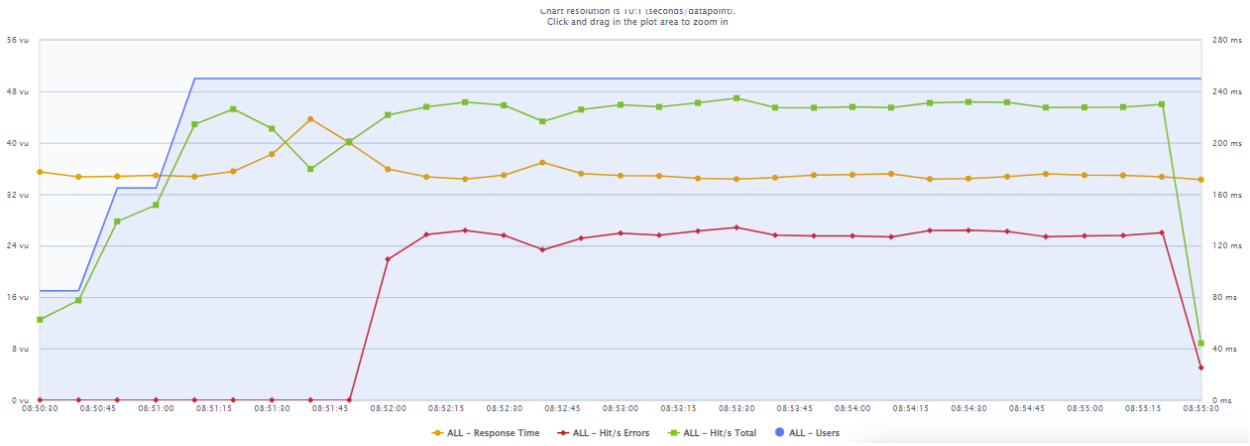


Рисунок 2.19 – Результати тестування.

2.6.3 Автоматизоване тестування

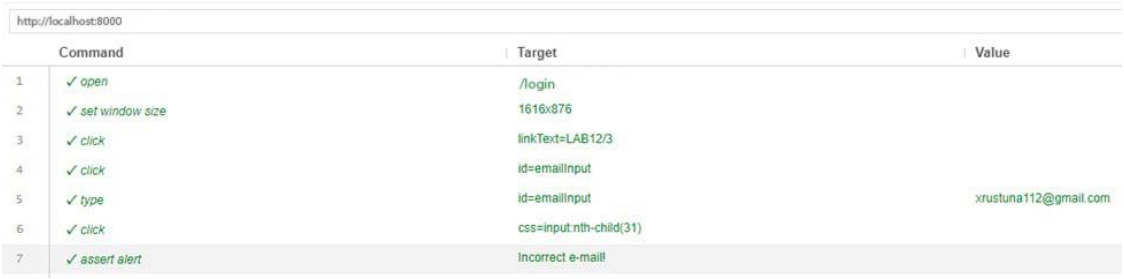
Автоматизоване тестування програмного забезпечення (ПЗ) є процесом використання спеціальних програмних інструментів для автоматичного виконання тестових сценаріїв на додатках, що розробляються. Метою автоматизованого тестування є підвищення ефективності тестування, зменшення людського фактору, та забезпечення стабільної якості продукту.

Для виконання автоматизованого тестування обрано засіб Selenium. Selenium — це популярний інструмент для автоматизованого тестування веб-додатків. Він надає можливість взаємодіяти з веб-сторінками так, як це робить користувач: відвідувати сторінки, натискати кнопки, заповнювати форми, переміщатися між вкладками і багато іншого. Selenium є потужним і гнучким інструментом, який підтримує багато мов програмування і браузерів.

Selenium підтримує автоматизацію для більшості популярних браузерів, що робить його універсальним інструментом для крос-браузерного тестування. Також він дозволяє писати тести на різних мовах програмування, що дає розробникам гнучкість у виборі інструментів і технологій. Selenium легко інтегрується з іншими інструментами та фреймворками, такими як TestNG, JUnit, Jenkins, Maven та багато інших. Крім цього даний засіб є проектом з відкритим кодом, тому його можна використовувати безкоштовно і модифікувати відповідно до своїх потреб. Щоправда Selenium призначений для автоматизації веб-додатків і не підтримує автоматизацію десктопних додатків. Для створення складних тестових сценаріїв потрібні знання мов програмування і досвід у розробці. Іноді виникає затримка у підтримці нових функцій браузерів, що може викликати проблеми з сумісністю.

Selenium є потужним інструментом для автоматизації тестування веб-додатків, який надає розробникам і тестувальникам можливості для ефективного створення та виконання тестових сценаріїв. Завдяки своїй гнучкості, розширюваності та підтримці багатьох браузерів і мов програмування, Selenium залишається популярним вибором для автоматизації тестування в сучасній розробці програмного забезпечення.

За допомогою даного розширення можна складати тести, які програма відтворює, тим самим перевіряючи якість програмного забезпечення та надійність системи.


http://localhost:8000

	Command	Target	Value
1	✓ open	/login	
2	✓ set window size	1616x876	
3	✓ click	linkText=LAB12/3	
4	✓ click	id=emailInput	
5	✓ type	id=emailInput	xrustuna112@gmail.com
6	✓ click	css=input:nth-child(31)	
7	✓ assert alert	Incorrect e-mail!	

Рис. 2.20 – Результат тестування на Selenium

На рисунку можна побачити скрипт автоматизованого тестування для веб-додатку, виконаний за допомогою інструмента Selenium IDE. Скрипт містить команди, що виконуються в браузері для перевірки певного функціоналу.

Скрипт автоматизованого тестування виконує послідовність дій, що перевіряють функціонал входу користувача на веб-сторінку. Усі команди, включаючи відкриття сторінки, встановлення розміру вікна, натискання на посилання, введення електронної адреси та перевірку сповіщення, були успішно виконані, що підтверджено зеленими галочками поруч з кожною командою.

Результат тестування показує, що система правильно реагує на введення некоректної електронної адреси, відображаючи відповідне сповіщення "Incorrect e-mail!". Це свідчить про те, що функціонал обробки входу користувачів працює коректно і система здатна виявляти та повідомляти про помилки користувача.

Таким чином, автоматизоване тестування підтвердило правильну роботу основних компонентів проекту, що забезпечує надійність і стабільність функціоналу входу. Це є важливим етапом у процесі розробки, оскільки дозволяє впевнитися у відсутності критичних помилок, які можуть вплинути на користувацький досвід.

Результати тестування:

1. Функціональне тестування: всі функції системи були ретельно перевірені для підтвердження їх відповідності заданим вимогам. Зокрема,

перевірялися основні компоненти інтерфейсу користувача, такі як відображення папок, навігація між сторінками та функціонал входу користувача.

2. Навантажувальне тестування: за допомогою інструментів, таких як BlazeMeter, було перевірено стійкість системи до навантажень, що дозволило оцінити її продуктивність під високими навантаженнями та забезпечити стабільність роботи в умовах пікових запитів.

3. Автоматизоване тестування: для зменшення часу на рутинні перевірки та забезпечення стабільності системи використовувалися автоматизовані тести з використанням інструментів, таких як Selenium. Це дозволило швидко виявляти та виправляти помилки.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

3.1 Працездатність людини – оператора

Робота оператора електронно-обчислювальних машин (ЕОМ) є важливим і відповідальним завданням, яке вимагає високої концентрації уваги, напруженості зору та нервового напруження. З огляду на це, організація робочого місця та створення належних умов праці мають вирішальне значення для збереження працездатності оператора та запобігання професійним захворюванням.

Згідно з НПАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», облаштування робочих місць повинно забезпечувати належні ергономічні характеристики. Робоче місце оператора повинно відповідати таким вимогам:

- Ергономічні характеристики: Основні елементи робочого місця повинні мати відповідні розміри та розташування для комфортної роботи оператора.
- Освітлення: Робоче місце повинно бути добре освітлене без відблисків, що можуть спричиняти дискомфорт або пошкодження зору.
- Мікроклімат: Оптимальні параметри температури, відносної вологості та швидкості руху повітря забезпечують комфортні умови для роботи.

Робоче місце оператора з ВДТ повинно бути організоване так, щоб забезпечувати комфортні умови для роботи. Робочі місця повинні бути розташовані так, щоб природне світло падало збоку, переважно зліва. Відстань між бічними поверхнями ВДТ повинна бути 1,2 м, а від тильної поверхні одного ВДТ до екрана другого – 2,5 м. Якщо робота вимагає особливої концентрації, робочі місця операторів слід відділяти перегородками заввишки 1,5–2 м.

Робочий стілець оператора є важливим елементом, який впливає на його працездатність. Стілець повинен мати підіймально-повертальний пристрій, що регулюється за висотою, кутом нахилу сидіння та спинки. Поверхня сидіння повинна бути плоскою, передній край – заокругленим. Висота сидіння повинна регулюватися в межах 400–500 мм. Для зниження статичного напруження м’язів

слід використовувати підлокітники, що регулюються за висотою. Висота робочої поверхні столу повинна регулюватися в межах 680–800 мм, а ширина та глибина повинні забезпечувати можливість виконання операцій у зоні досяжності. Оптимальні умови мікроклімату є важливими для збереження працездатності оператора. Температура повітря в робочій зоні повинна відповідати нормам для певної категорії робіт. Робочі приміщення не повинні межувати з приміщеннями, де рівні шуму та вібрації перевищують допустимі значення.

Оператори піддаються впливу електростатичного поля між екраном і оператором, що може призвести до нервово-емоційних напружень, головного болю, порушення сну та інших негативних наслідків. Тому важливо забезпечити належний захист та зниження впливу цих факторів.

У приміщеннях з ПК оператори піддаються впливу пилу, що притягується до обладнання. Це призводить до забруднення екранів та інших елементів устаткування, а також до збільшення потоку дрібних частинок пилу на обличчя та органи дихання оператора. Важливо забезпечити належну вентиляцію та регулярне очищення робочих місць.

Забезпечення оптимальних умов праці для оператора ЕОМ є важливою складовою збереження його працездатності та здоров’я. Ергономічне оформлення робочого місця, належні мікрокліматичні умови, захист від електростатичних полів та запобігання пилу – усе це сприяє комфортній та ефективній роботі оператора. Важливо дотримуватися встановлених норм та правил для забезпечення безпечної та продуктивної праці.

3.2 Характеристика небезпечних зон обладнання і розробка заходів безпеки

Безпека технічних процесів визначається збереженням виробничого обладнання, оснащення, сировини і використовуваних матеріалів. Вона повинна бути забезпечена комплексом проектних і організаційно-технічних рішень, що складаються з розумного вибору як всього технічного процесу в цілому, так і

окремих технічних операцій. Підбір виробничого обладнання та споруд при виборі умов транспортування і зберігання матеріалів, напівфабрикатів, готової продукції та відходів.

Для забезпечення вимог безпеки виробниче обладнання повинно бути безпечним при монтажі, експлуатації та ремонті, як окремо, так і в складі комплексів і технічних схем.

Небезпечна зона обладнання – це простір, де працівники можуть піддаватися впливу небезпечних або шкідливих виробничих факторів [14]. Небезпечні зони можуть бути локалізовані навколо або поблизу рухомих елементів обладнання (зубчастих коліс, ремінних або ланцюгових передач, обертових деталей, ріжучих інструментів і т.д.).), а також через можливість ураження електричним струмом, електромагнітного, іонізаційного, лазерного, ультрафіолетового та інфрачервоного випромінювання, шуму, вібрації, ультразвуку, шкідливих впливів. гази, пара і пил, а також можливість отримання травм через розлітання стружки.

Розміри небезпечної зони в просторі і в часі можуть бути постійними (наприклад, зона між пуансоном і матрицею в пресах) і змінними (наприклад, радіус відльоту стружки може змінюватися в залежності від властивостей матеріалу, що обробляється і режимів різання).

Для забезпечення безпеки необхідно передбачити пристрій, що виключає можливість потрапляння працівників в небезпечні зони обладнання або послаблює або виключає вплив небезпечних факторів.

Конструктивні елементи обладнання повинні виключати можливість випадкового пошкодження, яке може викликати небезпеку. Конструкційні матеріали не повинні бути небезпечними або шкідливими для здоров'я. Використання нових матеріалів і субстанцій, які не пройшли належну перевірку, не допускається.

Рухомі частини обладнання, що становлять небезпеку для працівників, повинні бути захищені або оснащені іншими засобами захисту. Конструкція пристрою повинна виключати можливість випадкового контакту працівника з гарячими або переохолодженими деталями, а також елементами з гострими

краями і кутами. Повинен бути передбачений захист від ураження електричним струмом, в тому числі при неправильних діях персоналу, і виключена можливість накопичення небезпечної кількості електростатичного заряду.

Конструкція обладнання повинна забезпечувати виключення або зниження до регульованого рівня шуму, вібрації і шкідливого випромінювання. Пристрій не повинен виступати джерелом викиду шкідливих речовин, що перевищують гранично допустиму концентрацію, в робочу зону виробничих приміщень, а також великої кількості тепла і вологи, що погіршують погодні умови.

Обладнання повинно бути оснащене засобами сигналізації про порушення нормальної роботи і, при необхідності, засобами аварійної зупинки.

Для запобігання ризику несподіваного відключення електроенергії всі робочі, підйомні і захватні органи і механізми мають бути оснащені захисними пристроями, що запобігають вивільнення або втрату продуктів або інструментів. Також слід виключити можливість мимовільного включення приводу робочого органу при повторному включенні живлення після випадкового відключення.

Органи управління повинні забезпечувати надійну швидкість запуску і зупинки, простоту використання, а зусилля для їх переміщення повинно бути невеликим. Вони повинні мати зручну і безпечну форму і розмір, а їх конструкція повинна виключати можливість мимовільного включення і виключення, а також виконання послідовності неправильних операцій.

Для контролю потрібно символічне позначення або відповідний напис. Вони повинні бути пофарбовані в червоний колір, забезпечені відповідними етикетками і розміщені в легкодоступних місцях [15].

Засоби захисту, які є конструктивним елементом обладнання, завжди повинні виконувати захисні функції. Він спрацьовує, коли людина входить в небезпечну зону обладнання, коли з'являються небезпечні або шкідливі фактори. Їх дія повинна тривати протягом усього періоду дії цього фактора.

Якщо засоби захисту відключені, несправні або зняті, обладнання не повинно працювати. Тобто він повинен відключатися автоматично і виключати можливість його включення до тих пір, поки не буде відновлено захисне

обладнання. Захисне обладнання повинно бути самоконтрольованим або легкодоступним для моніторингу та технічного обслуговування.

Розробка заходів безпеки для технологічних процесів включає:

1. Виявлення та аналіз небезпечних і шкідливих виробничих факторів, які можуть виникати під час виконання технологічних операцій.
2. Визначення небезпечних зон і розробка заходів для їх локалізації або виключення можливості перебування в них людей.
3. Розробка систем сигналізації, блокування і захисту, які забезпечують своєчасне виявлення і усунення небезпеки.
4. Вибір безпечних методів і режимів виконання технологічних операцій, включаючи підбір обладнання, інструментів і матеріалів.
5. Організація навчання і інструктажу персоналу з питань охорони праці та безпеки технологічних процесів.
6. Забезпечення засобами індивідуального і колективного захисту, включаючи спеціальний одяг, взуття, засоби захисту органів дихання, зору, слуху та інші.
7. Проведення регулярного контролю і оцінки ефективності заходів безпеки, аналіз аварій і нещасних випадків для їх запобігання в майбутньому.

Тому комплексний підхід до забезпечення безпеки технічних процесів, що включає розробку і впровадження заходів безпеки, регулярний моніторинг і навчання персоналу, є ключем до збереження життя і здоров'я співробітників і підвищення ефективності і надійності виробництва.

ВИСНОВКИ

У цій кваліфікаційній роботі було досягнуто всіх основних цілей і вирішено поставлені завдання. Перш за все, проведений детальний аналіз предметної області дозволив сформувати чітке розуміння вимог до розроблюваної системи. Це дало змогу розробити модель предметної області, що стала основою для подальшого проєктування та реалізації. Зазначена модель включала всі необхідні компоненти, процеси і взаємозв’язки, які було враховано під час розробки архітектури системи.

Архітектурний підхід, заснований на мікросервісній архітектурі, забезпечив гнучкість, масштабованість та легкість оновлення окремих компонентів системи. Цей підхід дозволив розділити систему на незалежні сервіси, що взаємодіють між собою через чітко визначені інтерфейси. Це забезпечило можливість розробки, тестування і розгортання окремих сервісів без негативного впливу на інші компоненти системи. Крім того, використання мікросервісної архітектури спрощує масштабування системи, оскільки кожен сервіс може бути масштабований незалежно залежно від потреб у ресурсах.

Важливим аспектом розробки був вибір технологій для фронтенду. Використання Vue.js та Nuxt.js дозволило створити сучасний, динамічний та зручний інтерфейс користувача. Завдяки компонентному підходу було можливе створення багаторазових та легко підтримуваних частин коду, що значно спростило процес розробки та подальшої підтримки системи. Компоненти були розділені на логічні частини, що відповідали за різні функціональні аспекти системи, такі як управління папками, навігація між сторінками, аутентифікація користувачів та інші ключові елементи.

Процес тестування системи складався з кількох етапів, включаючи функціональне, автоматизоване та навантажувальне тестування. Функціональне тестування було спрямоване на перевірку коректності роботи окремих функцій системи. Автоматизоване тестування, зокрема за допомогою Selenium, дозволило автоматизувати процес перевірки функціоналу, що зменшило час, необхідний на

ручне тестування, та підвищило його ефективність. Навантажувальне тестування за допомогою інструментів, таких як Blazemeter, дозволило оцінити продуктивність системи під високими навантаженнями та оптимізувати її для забезпечення стабільної роботи в умовах пікових запитів.

Результати тестування показали високу стабільність і надійність системи. Всі виявлені помилки були успішно виправлені, що забезпечило високу якість кінцевого продукту. Система продемонструвала здатність обробляти великий обсяг запитів без втрати продуктивності, що свідчить про її готовність до експлуатації в реальних умовах. Важливим аспектом було також забезпечення безпеки системи, включаючи захист даних користувачів та запобігання несанкціонованому доступу.

Загалом, розроблена система відповідає всім вимогам і є готовою до використання в реальних умовах. Високий рівень продуктивності, зручність використання та надійність системи забезпечують її конкурентоспроможність та довготривалу експлуатацію. Вона може бути легко адаптована до змінних вимог та масштабована в разі необхідності. Виконана робота підтверджує доцільність обраного підходу до розробки та успішне застосування сучасних технологій і методологій. Це дозволяє стверджувати, що мета дослідження була досягнута, а результати можуть бути використані як основа для подальших наукових досліджень та практичних розробок у цій галузі.

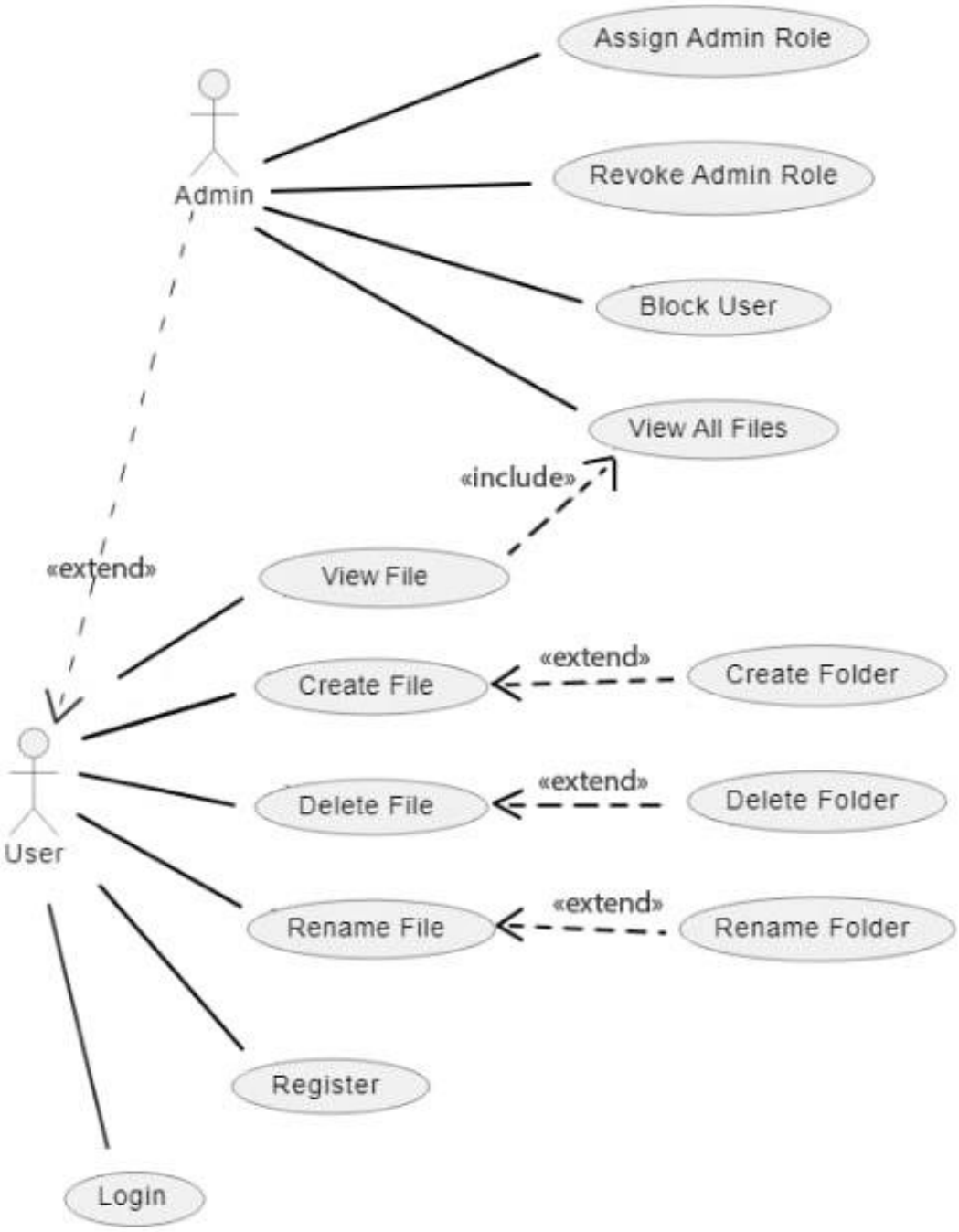
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Що таке хмарне сховище та як цим користуватись [Електронний ресурс] – Режим доступу до ресурсу: https://tribun.com.ua/uk/105054-scho-take-xmarne-sxovische-ta-jak-tsim-koristuvatis
2. OneDrive для ПК [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/microsoft-365/onedrive/download
3. Сервіс збереження та передачі файлів FexNet [Електронний ресурс] – Режим доступу до ресурсу: https://fex.net/
4. Thomas T. The Pragmatic Programmer / T. Thomas, A. Hunt., 2019. – 320 с.
5. Macrae C. Vue.js: Up and Running: Building Accessible and Performant Web Apps / Callum Macrae., 2018. – 164 с. – (1)
6. Nuxt Guide [Електронний ресурс] – Режим доступу до ресурсу: https://nuxt.com/docs/guide
7. Мартін Р. Чистий Agile. Назад до основ / Роберт Мартін., 2021. – 224 с.
8. Вігерс К. Розробка вимог до ПЗ / Карл Вігерс., 2021. – 736 с.
9. Microservice Architecture pattern [Електронний ресурс] – Режим доступу до ресурсу: https://microservices.io/patterns/microservices.html
10. UML [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml.org/
11. Майерс Г. Мистецтво тестування програм / Гленфорд Майерс., 2020. – 272 с. – (3)
12. Запорожець О. Основи охорони праці. Підручник / О. Запорожець, О. Протоєрейський. – Харків: Центр учбової літератури, 2021. – 264 с.
13. Серіков Я. Основи охорони праці / Я. Серіков, Б. Халмурадов. – Харків: Центр учбової літератури, 2024. – 250 с.
14. Запорожець О. Безпека життєдіяльності / О. Запорожець, В. Заплатинський. – Харків: Центр учбової літератури, 2024. – 448 с.
15. Вахонєва Т. Основи охорони праці в Україні / Тетяна Вахонєва. – Київ: Дакор, 2019. – 508 с.

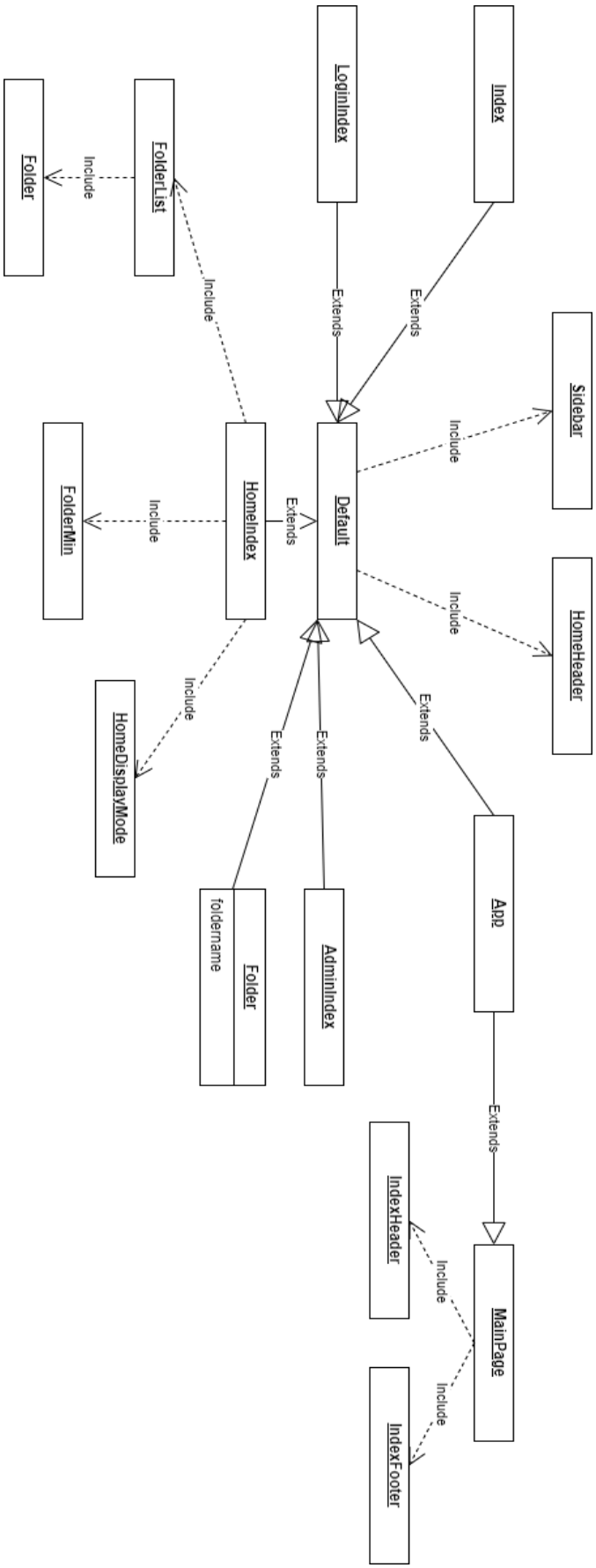
16. Мягченко О. Безпека життєдіяльності людини та суспільства / Олександр Мягченко. – Харків: Центр учбової літератури, 2023. – 384 с.
17. Пастух О. Estimation of shape memory alloys functional properties by methods of artificial intelligence [Електронний ресурс] / О. Пастух, О. Ясній. – 2024. – Режим доступу до ресурсу: https://elartu.tntu.edu.ua/handle/lib/39148
18. Петрик М. Two-stage random sequential adsorption of discorectangles and disks on a two-dimensional surface [Електронний ресурс] / М. Петрик, І. Бойко. – 2023. – Режим доступу до ресурсу: https://elartu.tntu.edu.ua/handle/lib/44653
19. Михалик Д. Intellectual information technologies for the study of filtration in multidimensional nanoporous particles media [Електронний ресурс] / Д. Михалик, М. Петрик. – 2022. – Режим доступу до ресурсу: https://elartu.tntu.edu.ua/handle/lib/42018?locale=gl

ДОДАТКИ

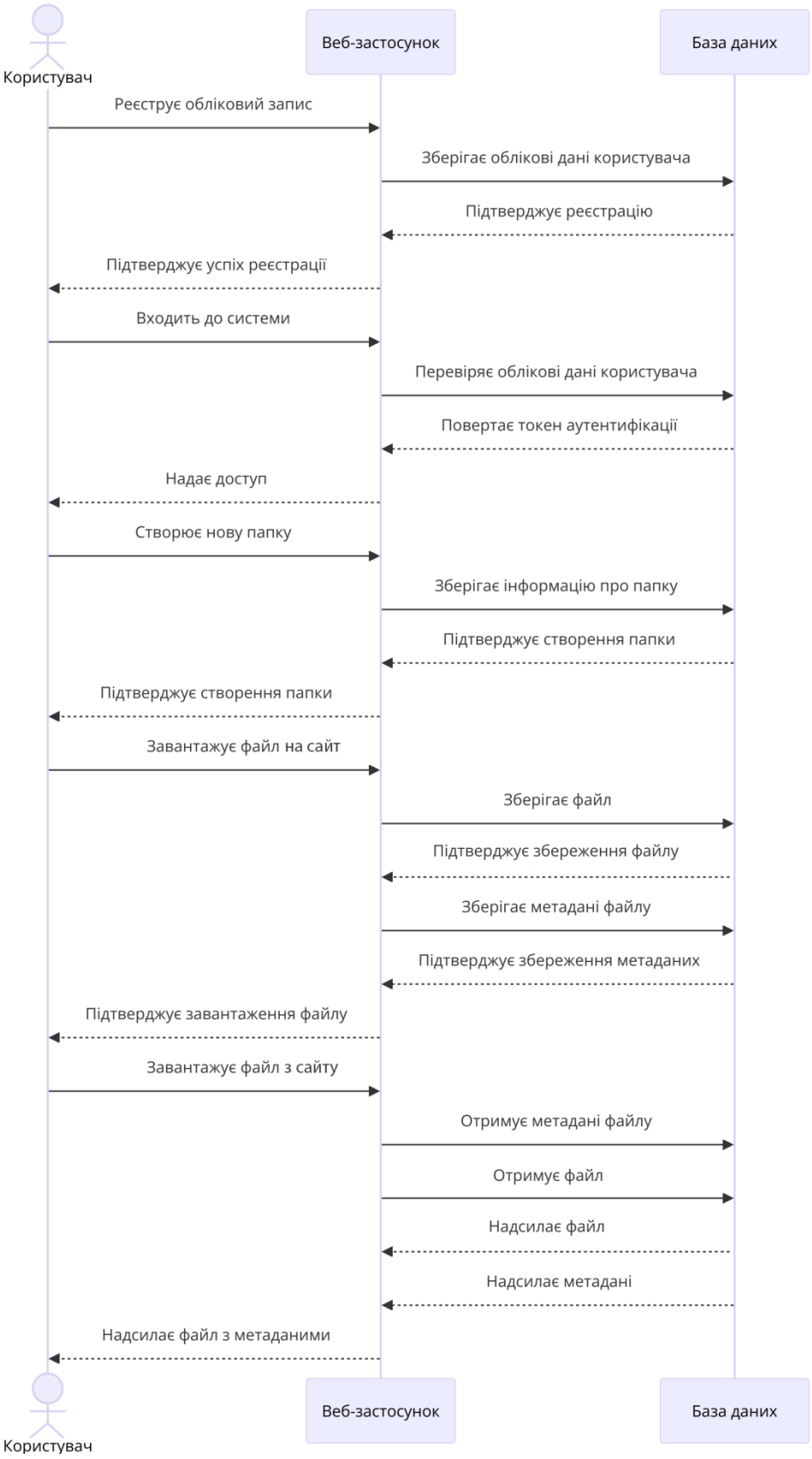
Додаток А. Повна діаграма варіантів використання



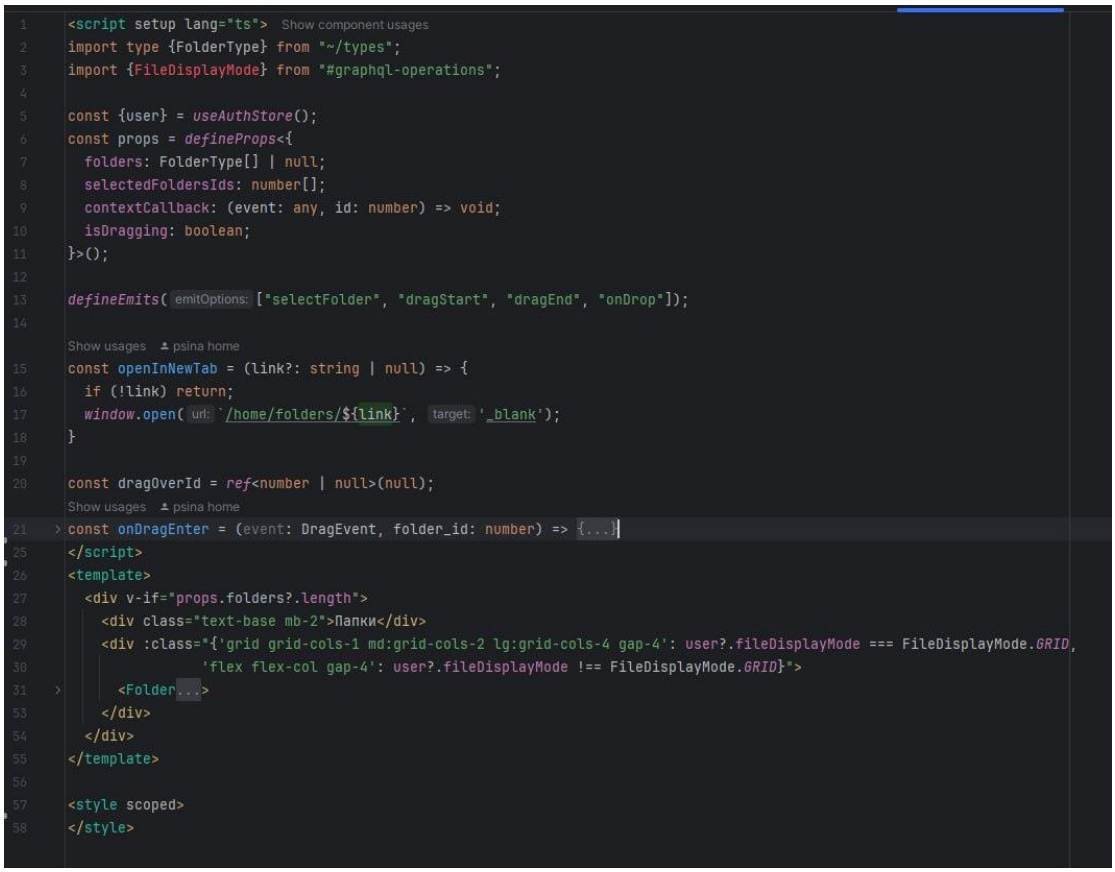
Додаток Б. Повна діаграма об’єктів



Додаток В. Повна діаграма послідовностей



Додаток Г. Приклад компоненту



```vue
<script setup lang="ts"> Show component usages
import type {FolderType} from "~/types";
import {FileDisplayMode} from "#graphql-operations";

const {user} = useAuthStore();
const props = definePropsr<{
  folders: FolderType[] | null;
  selectedFoldersIds: number[];
  contextCallback: (event: any, id: number) => void;
  isDragging: boolean;
}>();

defineEmits( emitOptions: ["selectFolder", "dragStart", "dragEnd", "onDrop"]);

Show usages  psina home
const openInNewTab = (link?: string | null) => {
  if (!link) return;
  window.open( url: `/home/folders/${link}`,  target: '_blank');
}

const dragOverId = ref<number | null>(null);
Show usages  psina home
const onDragEnter = (event: DragEvent, folder_id: number) => {...}
</script>
<template>
  <div v-if="props.folders?.length">
    <div class="text-base mb-2">Папки</div>
    <div :class="{'grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4': user?.fileDisplayMode === FileDisplayMode.GRID,
                  'flex flex-col gap-4': user?.fileDisplayMode !== FileDisplayMode.GRID}">
      <Folder...>
    </div>
  </div>
</template>

<style scoped>
</style>
```

Додаток Д. Диск