МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Веб-сервіс зберігання та обміну файлами. Серверна частина»

Завідувач випускаючої кафедри — **Довбиш А. С.**

Керівник роботи — **Ободяк В. К.**

Студент групи ІН.м-92 — **Отрощенко М. С.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Отрощенку Михайлу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Веб-сервіс зберігання та обміну файлами. Серверна частина*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми. Постановка задачі дослідження. 2) Вивчення процесів хмарного зберігання файлів. 3) Моделювання та проектування web-додатку. 4) Практична реалізація.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження.*		
2.	*Вивчення процесів хмарного зберігання файлів.*		
3.	*Моделювання та проектування web-додатку.*		
4.	*Практична реалізація.*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Веб-сервіс зберігання та обміну файлами. Серверна частина». Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 22 найменувань, додатків. Загальний обсяг роботи – 86 сторінок, у тому числі 59 сторінок основного тексту, 2 сторінки списку використаних джерел, 24 сторінки додатків.

Метою даного проекту є розробка веб-додатку для зберігання та обміну файлами.

В роботі проведено аналіз аналогів, проектування і розробку додатку. Результатом проведеної роботи є веб-додаток для завантаження файлів та зберігання файлів на віддаленому сервері. Практичне значення роботи полягає у наданні більш надійного і безпечного методу зберігання файлів і даних.

Ключові слова: PHP, LARAVEL, JAVASCRIPT, ФАЙЛООБМІННИК, ПЛАГІНИ, ПРОТОТИП, СКРИПТИ, САЙТ

ЗМІСТ

ВСТУП

З розвитком технологій і збільшенням кількості носіїв інформації виникає питання зберігання та обміну даними між гаджетами та електронними носіями. Також з огляду на останні новини та пандемію виникає гостра необхідність в надійному зберіганні і швидкому зручному обміні великими обсягами найрізноманітніших даних між користувачами мережі, що знаходяться в самих різних куточках нашої планети.

Одним з найбільш оптимальних, ефективних, зручних і тому затребуваних рішень стали так звані хмарні сервіси для зберігання даних, що дозволяють зберігати, оперативно редагувати і швидко передавати значні обсяги найрізноманітніших файлів. Причому доступ до тих або інших даних можуть одночасно мати багато користувачів, або ж інформація може носити конфіденційних характер і бути призначеною тільки для особистого користування.

Виходячи із визначеної актуальності проблеми зберігання файлів, було визначено, що потрібно створити сайт для того, щоб кожен бажаючий міг з легкістю звантажити файли на сервіс і без проблем мати доступ до них. Для реалізації поставленої мети потрібно вирішити такі задачі:

- Вивчити процеси хмарного зберігання файлів.
- Провести аналів аналогів.
- Визначити задачі сервісу.
- Обрати та налаштувати інструменти реалізації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

Проблема своєчасного отримання інформації в бізнесі існувала завжди. Сьогоднішні облікові програми потужні і багатофункціональні, а на шляху потоку цієї інформації стає оператор ПК та засоби автоматичного введення-виведення даних.

Далі наведено найбільш актуальні і типові аспекти проблеми обміну даними:

– обмін між територіально віддаленими один від одного місцями введення інформації;

– обмін даними між системами обліку з різним призначенням (бухгалтерський облік, оперативний облік, управлінський облік);

– отримання консолідованого балансу з різних інформаційних баз (зведений баланс по дочірнім підприємствам);

Загальними вимогами, що пред'являються до систем обміну даними, є забезпечення одиничного введення інформації, використовуваної в декількох базах даних, дотримання загальних правил цілісності бази даних, стійкість системи до збоїв і захищеність від несанкціонованого доступу.

Для забезпечення даних вимог існують такі сервіси як файлоомінники або хмарні сервіси зберігання даних.

Хмарне сховище – це модель хмарних обчислень, яка передбачає зберігання даних в Інтернеті за допомогою постачальника обчислювальних ресурсів, який надає сховище даних як сервіс і забезпечує управління та обмін файлами. Це забезпечує гнучкість, глобальну масштабованість і надійність. Дані доступні в будь-який час і в будь-якому місці [1].

У загальних рисах хмарне сховище даних являє собою віртуальний файлообмінник або сервер з даними, доступ до яких можна отримати з будь-якого пристрою, підключеного до мережі і використовуваному хмарного сервісу.

Файлообмінник – сервіс, що надає користувачеві місце під його файли і цілодобовий доступ до них через інтернет. Такий сервіс дозволяє зручно обмінюватися файлами. На спеціальній сторінці файлообмінника (найчастіше на головній) користувач завантажує файл на сервер файлообмінника, а Файлообмінник віддає користувачеві постійне посилання, яку він може розсилати по електронній пошті, публікувати в блогах, на форумах або пересилати через системи миттєвого обміну повідомленнями. Перейшовши по такому посиланню, будь-який інший користувач може завантажити початковий файл [2].

Однак на відміну від традиційного сервера, що завантажується в хмарні сховища інформація розміщена одночасно на безлічі мережевих серверів, в тому числі, що знаходяться за тисячі кілометрів на іншому кінці Землі.

Зберігання даних в хмарі дозволяє принципово переглянути три аспекти своєї діяльності.

Таблиця 1.1 – Переваги хмарного сховища

Перевага	Опис
Сукупна вартість володіння	Завдяки хмарному сховищу не потрібно купувати устаткування, виділяти ресурси для сховища або витрачати кошти зберігання даних. Можна додавати або видаляти ресурси на вимогу, швидко змінювати продуктивність та термін зберігання. Це дозволяє забезпечити економію при великих обсягах.
Час для розгортання	Коли група розробників готова до запуску проекту, інфраструктура не повинна їх стримувати. Хмарне сховище дозволяє IT-фахівцям швидко виділяти необхідний простір для зберігання даних саме тоді, коли це потрібно. В результаті IT-фахівці можуть зосередитися на вирішенні

Перевага	Опис
	складних проблем, пов'язаних з додатками, а не на питаннях управління системами зберігання даних.
Управління інформацією	Централізоване сховище в хмарі створює величезні можливості для нових прикладів використання. Використовуючи політики управління життєвим циклом в хмарному сховищі, можна вирішувати важливі завдання, пов'язані з управлінням інформацією, включаючи автоматичний розподіл за рівнями або блокування даних з метою дотримання вимог [4].

Існує також три типи хмарних сховищ даних: об'єктні сховища, файлові сховища і блокові сховища. Кожен з них пропонує свої переваги, які підходять для конкретних прикладів використання.

Таблиця 1.2 – Типи хмарних сховищ

Тип	Опис
Об'єктне сховище	Додатки, розроблені в хмарі, як правило, використовують такі переваги об'єктного сховища, як широкі можливості масштабування і зберігання властивостей об'єктів у вигляді метаданих. Об'єктні сховища, наприклад Amazon Simple Storage Service (S3), ідеально підходять для розробки з нуля сучасних додатків, для яких потрібна гнучкість і можливість масштабування. Крім того, ці сховища можна використовувати для імпорту даних з існуючих сховищ з метою аналітики, резервного копіювання або архівації.

Тип	Опис
Файлове сховище	Деяким програмам потрібен доступ до передачі файлів, отже, їм необхідна файлова система. Даний тип сховища часто підтримується сервером сховищ, підключеним до мережі (NAS).
Блочне сховище	Інші корпоративні додатки, наприклад бази даних або системи планування ресурсів підприємства (ERP), часто потребують виділене сховище з низькими затримками для кожного з вузлів. Таке сховище працює аналогічно сховища з прямим підключенням (DAS) або мережі зберігання даних (SAN) [5].

Питання забезпечення надійного зберігання, безпеки та доступності критично важливих корпоративних даних мають першорядну важливість. При розгляді варіанту зберігання даних в хмарі існує кілька фундаментальних вимог.

Таблиця 1.3 – Вимоги, що пред'являються до хмарного сховища

Тип	Опис
Надійність	Дані повинні зберігатися з надмірністю. В ідеалі вони повинні бути розподілені між кількома об'єктами і кількома пристроями в рамках кожного з об'єктів. Стихійні лиха, людський фактор або механічні несправності не повинні призводити до втрати даних.
Доступність	Всі дані повинні бути доступними в разі необхідності, але існує різниця між виробничими даними і архівами. Ідеальне хмарне сховище пропонує оптимальне поєднання між часом отримання даних і вартістю.
Безпека	Всі дані повинні шифруватися – як при зберіганні, так і при передачі. Дозволи та контроль доступу повинні

Тип	Опис
	працювати в хмарі точно так же, як і в локальних сховищах даних [7].

Однак в роботі з «хмарами» справедливості заради можна відзначити і кілька основних мінусів:

Перший і головний – це недостатнє опрацювання питання забезпечення безпеки. Незважаючи на пропаговану політику конфіденційності, до інформації, що завантажується залишається відкритим доступ співробітників сервісу і його програмного забезпечення.

Другий недолік випливає частково з першого – при зломі сховища або сервісу, наприклад, в результаті хакерської атаки, конфіденційні файли можуть потрапити під загальний доступ або в руки до зловмисників.

Третій недолік пов'язаний з необхідністю очікування повної синхронізації пристрою і хмарного сховища, якщо така опція використовується. У свою чергу, тривалість очікування при зверненні до сервісу залежить від швидкості доступу в мережу. Переривати же процес синхронізації не рекомендується через високу ймовірність виникнення помилок і збоїв.

Отже сфера і можливості використання хмарних сервісів широкі. «Хмари» можуть використовуватися в корпоративних цілях для колективної командної роботи з певною інформацією, оперативного обміну актуальними даними, можуть служити файлообмінниках в особистих цілях для зберігання та обміну персональними даними. Також на відміну від локальних дата центрів хмари мають великий ряд переваг, а також деякі недоліки які було описано раніше.

1.2 Аналіз аналогів

Процес розробки нового продукту неможливий без попередньої постановки завдань і цілей а також ретельного аналізу ринкової ніші. Аналіз web-сайтів конкурентів, аналіз цільової аудиторії майбутнього сайту, виявлення

«сильних» і «слабких» сторін проекту. Розробка сайту завжди починається з виконання подібних завдань. Перед початком розробки поставленої задачі потрібно провести аналіз існуючих аналогів.

Для досягнення мети було проведено дослідження таких веб-сайтів:

- mega.dp.ua
- www.google.com/drive
- www.dropbox.com
- www.mediafire.com

Одним із ресурсів з подібною тематикою і призначенням являється сайт «Google Диск» призначений для завантаження, зберігання та поширення клієнтських фалів і інформації. Веб-сайт має досить зручний і сучасний інтерфейс, зручний і зрозумілий пошук інтерфейс якого показано на рисунку 1.1.

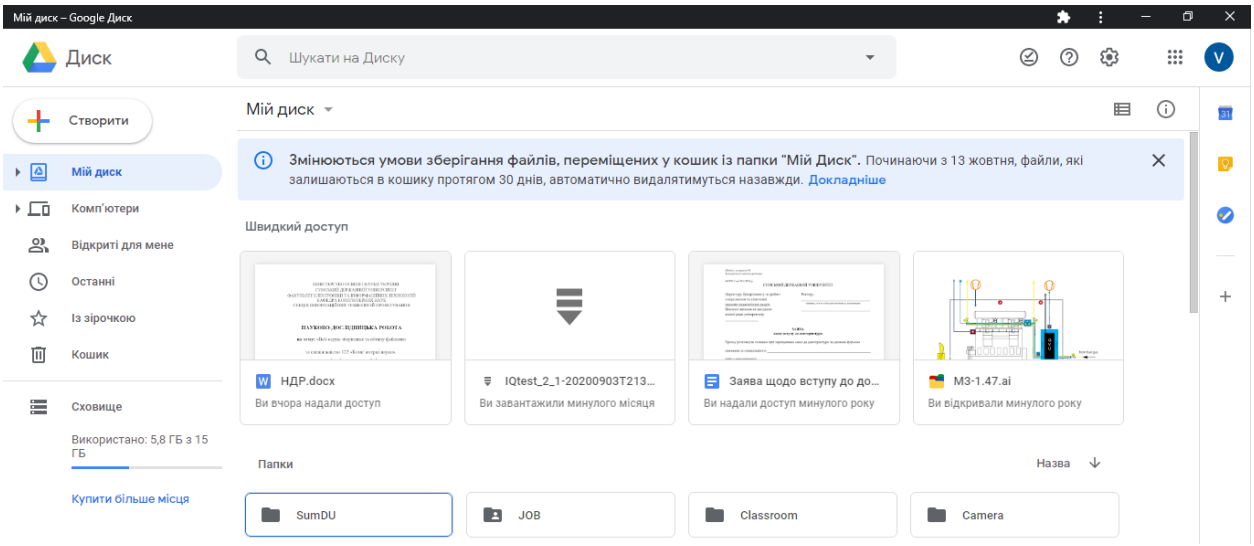

Рисунок 1.1 – головна сторінка Google диску

До переваг даного сайту можна віднести зручний інтерфейс, зручний пошук. Також на даному сайті можна працювати з різними типами файлів прямо на сайті.

Проте представлений сайт має декілька недоліків. До них можна віднести обмежена кількість пам'яті для зберігання щоб отримати великий обсяг пам'яті

потрібно заплатити кошти. Також на сайті відсутня можливість підписатися на користувача. Крім того відсутня можливість оцінювати файли.

Наступним сайтом для аналізу було обрано «Mega» рисунок 1.2.

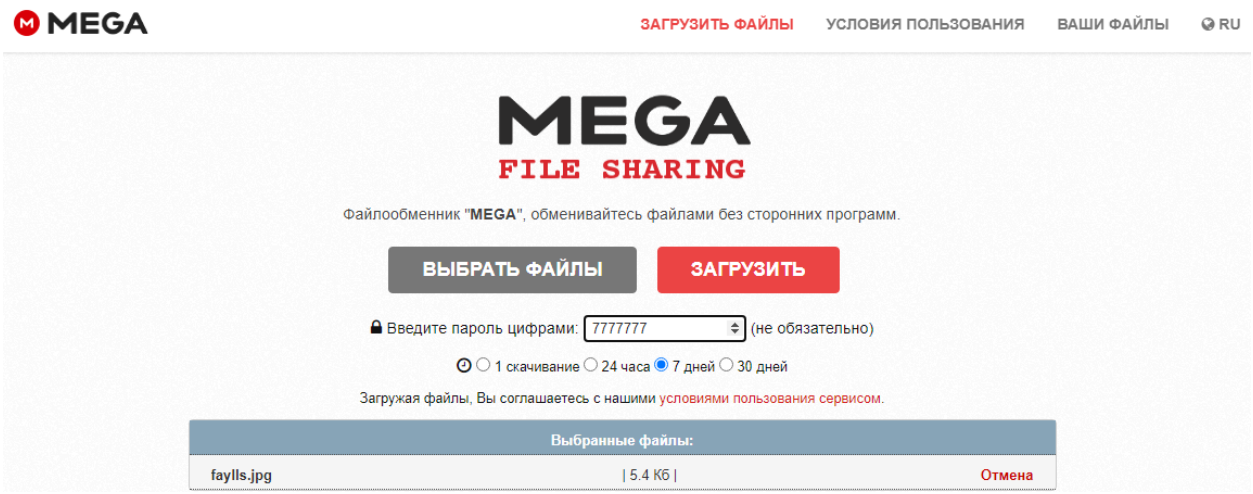

Рисунок 1.2 – головна сторінка «Mega»

Даний сайт має декілька переваг, а саме можливість завантажувати файли до 50GB. Проте представлений сайт має досить велику кількість недоліків. До них можна віднести застарілий дизайн, що є критичним для користувача. Також на сайті відсутня можливість підписатися на користувача, відсутність пошуку популярних публікацій з можливістю оцінки файлу, а також відсутність детальної інформації про файл.

Для аналізу також було обрано сайт «Dropbox» домашня сторінка якого зображена на рисунку 1.3.

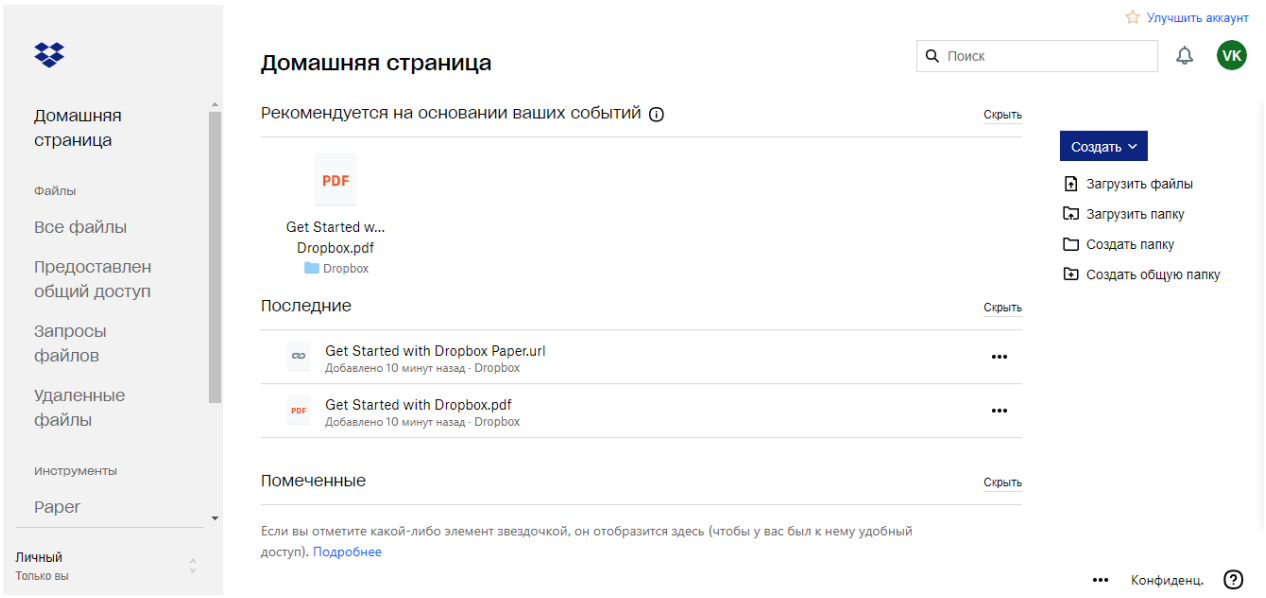

Рисунок 1.3 – домашня сторінка «Dropbox»

Даний сайт вирізняється сучасним дизайном, зручним інтерфейсом та наявність зворотного зв’язку з адміністрацією сайту.

Проте даний сайт також має ряд недоліків. На даному сервісі доступно безкоштовно лише 2GB пам`яті для завантаження та зберігання файлів.

Для аналізу також було обрано сайт mediafire.com головна сторінка якого зображена на рисунку 1.4.

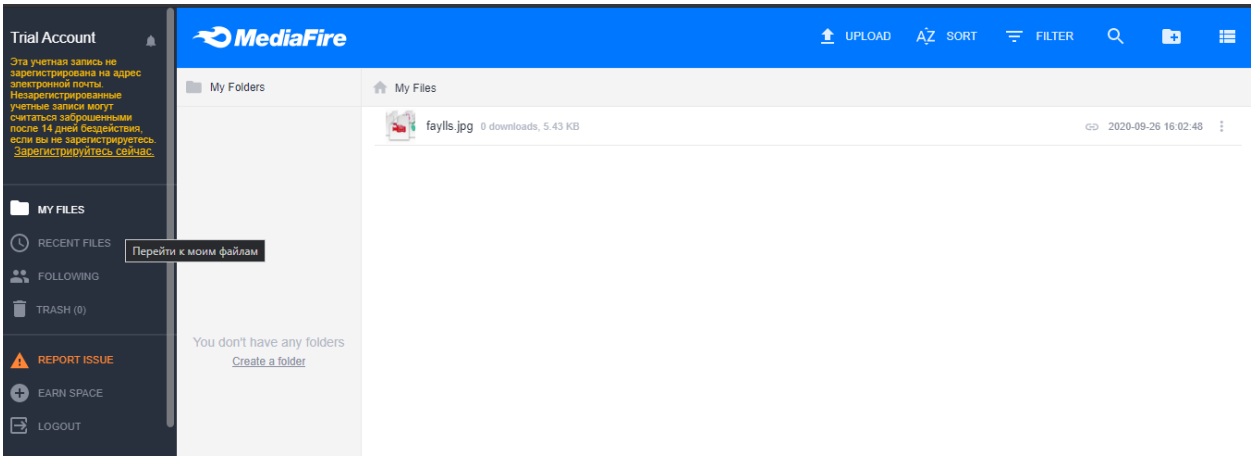

Рисунок 1.4 – головна сторінка mediafire.com

Даний сайт вирізняється унікальним дизайном, зручним пошуком, швидкістю роботи та наявністю можливості завантажувати файли без реєстрації.

Проте навіть даний сайт також має ряд недоліків. А саме базовий розмір сховища 10GB, публікувати лише за посиланням, неможливо оцінити файл. Також на даному сайті не має можливості підписуватись на інших користувачів.

Зробивши аналіз перелічених сайтів була зроблена таблиця 1.2 переваг та недоліків.

Таблиця 1.4 – Аналіз розглянутих сайтів

Критерії	Google Диск	Mega	Dropbox	Mediafire
Опис завантажених файлів	-	-	-	-
Зручний інтерфейс	+	-	-	-
Не обмежений розмір сховища	-	-	-	-
Зручний пошук	+	-	+	+
Можливість підписатись на інших користувачів	-	-	+	-
Рейтингове оцінювання	-	+/-	+	-
Публікувати не лише за посиланням	-	-	+	-

Отже, в результаті проведеного аналізу сайтів зі схожою тематикою і призначенням було вирішено розробити сайт який не матиме недоліків виявлених на сайтах представлених в таблиці 1.4.

Дана розробка матиме всі переваги даних сайтів, а також матиме унікальний дизайн та зручний інтерфейс.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1. Мета та задачі дослідження

Дана інформаційна система представляє з себе сервіс, що надає користувачеві місце під його файли і цілодобовий доступ до них через інтернет, як правило по протоколу https. Такий сервіс дозволяє зручно обмінюватись файлами між іншими користувачами інтернет мережі. На кожній сторінці файлообмінника користувач матиме можливість завантажувати файли на сервер файлообмінника, а додаток віддає користувачеві постійне посилання, яку він може розсилати по e-mail, публікувати в блогах, на форумах або інших порталах. Перейшовши по такому посиланню будь-який інший користувач може завантажити початковий файл.

Сервіс повинен бути реалізований у вигляді сайту, доступного в мережі Інтернет. Сайт повинен складатися із взаємозалежних розділів із чітко розділеними функціями.

Головна мета інформаційної системи – створення повноцінної платформи для зберігання файлів з можливістю доступу до них з різних типів операційних систем і гаджетів, надання надійного зберігання файлів на сервері.

Перелік вимог до сервісу:

Всі користувачі перед використанням повинні зареєструватись в системі.

Користувацька частина повинна складатися із наступних сторінок:

– Головна.
– Мої файли.
– Популярні файли.
– Обране.
– Кошик.

На головній сторінці сайту повинні бути такі елементи:

– Пошук файлів.

– Перегляд останніх публікацій користувачів.
– Перегляд інформації про файл.
– Можливість завантажувати файл на сервіс.

2.2. Вибір засобів реалізації

Розробка веб-сайтів – це процес створення веб-сайтів та додатків для інтернету. Від найпростіших, статичних веб-сторінок до платформ і додатків соціальних медіа, від веб-сайтів електронної комерції до систем управління контентом (CMS) і файлообмінників; всі інструменти, якими ми користуємось через інтернет щодня, були розроблені веб-розробниками.

Веб-розробка може бути розбита на три шари (табл. 1.5): кодування на стороні клієнта (фронтенд), кодування на стороні сервера (бекенд) та технології баз даних.

Таблиця 2.1 – Частини веб розробки

Назва	Опис	Приклад технологій розробки
Клієнтська сторона	Сценарії на стороні клієнта або розробка інтерфейсу стосується всього, що переживає кінцевий користувач безпосередньо. Код клієнта виконується у веб-браузері і безпосередньо стосується того, що бачать люди, відвідуючи веб-сайт. Такі речі, як компонування, шрифти, кольори,	HTML, CSS, Vue.js, Vuetify, JavaScript, React.js, Bootstrap.

Назва	Опис	Приклад технологій розробки
	меню та контактні форми, керуються фронтендом.	
На стороні сервера	Сценарії на стороні сервера або розробка бекенду – це все, що відбувається за кадром. Бекенд – це по суті частина веб-сайту, яку користувач насправді не бачить. Він несе відповідальність за зберігання та впорядкування даних та забезпечує те, що все на стороні клієнта працює безперебійно.	PHP, Laravel, Node.js, Python, Express.js, Docker.
Технологія баз даних	Веб-сайти також покладаються на технологію баз даних. База даних містить усі файли та вміст, необхідний веб-сайту для функціонування, зберігаючи його таким чином, щоб полегшити його роботу. База даних працює на сервері, і більшість веб-сайтів зазвичай використовують певну форму управління реляційною базою даних	MySQL, PostgresSQL, OracleSQL, MongoDB

2.2.1 Розробка інтерфейсу

Завдання розробника frontend – кодувати інтерфейс веб-сайту чи програми; тобто частину веб-сайту, яку бачить і взаємодіє користувач. Вони працюють над розробками, наданими веб-дизайнером, і реалізують їх за допомогою HTML, JavaScript та CSS.

Мови розмітки використовуються для визначення форматування текстового файлу. Іншими словами, мова розмітки повідомляє програмне забезпечення, яке відображає текст, як слід форматувати текст. Мови розмітки є повністю розбірливими для читання – вони містять стандартні слова, але теги розмітки приховані від користувача в кінцевому варіанті роботи.

Дві найпопулярніші мови розмітки – HTML та XML. HTML розшифровується як мова розмітки HyperText та використовується для створення веб-сайтів. Додані до звичайного текстового документа, всі теги HTML описують, як цей документ повинен відображатися веб-браузером.

Невідмінна частина будь-якої веб сторінки – каскадні стилі розмітки сторінки або CSS. Стилі – це в основному сукупність стилістичних правил. Мови аркушів стилів використовуються, доволі буквально, для оформлення документів, написаних мовами розмітки.

CSS розшифровується як каскадні таблиці стилів з акцентом на «стиль». Хоча HTML використовується для структури веб-документа, визначає такі речі, як заголовки та абзаци, і дозволяє вставляти зображення, відео та інші медіа, CSS визначає стиль створеного документа.

2.2.2 Розробка серверної частини

Код, який створюють розробники бекенду, гарантує, що все, що будує розробник frontend, є повністю функціональним. Головне завдання розробника серверної частини – переконатися, що сервер, програма та база даних взаємодіють між собою. Для вирішення такої складної задачі розробники

використовують серверні мови, такі як PHP, Ruby, Python та Java для створення програми.

Веб-технології стрімко розвиваються, і щоб встигати за швидкістю розвитку нових технологій, у веб-програмістів часто бракує часу для реалізації повсякденних завдань з нуля, таких як: автентифікація, API, тестування, дебаг, кешування і т.д. Саме для спрощення розробки сайтів і швидкого вирішення цих завдань були придумані веб-фреймворки.

Фреймворк – це каркас, призначений для створення динамічних веб-сайтів, веб-додатків, служб або ресурсів. Він спрощує розробку і позбавляє від необхідності писати звичайний код. Фреймворк спрощує доступ до бази даних, розробку інтерфейсу та зменшує дублювання коду.

Розглянемо один з популярних фреймворків Yii 2. Yii – це універсальний фреймворк, і він може бути задіяний у всіх типах веб-додатків.

Завдяки своїй складовій структурі та чудовій підтримці кешу, фреймворки особливо підходять для розробки великих проектів, таких як портали, форуми, CMS, магазини або програми RESTful. Yii – це командний проект. Він підтримується і розвивається сильною командою та великою спільнотою розробників. Автор цього фреймворку стежать за тенденціями розвитку веб-сайтів та інших проектів.

Найбільш підходящі можливості і кращі практики регулярно впроваджуються в фреймворк в вигляді простих і елегантних інтерфейсів:

– Як і багато інших PHP фреймворків, для організації коду Yii використовує архітектурний патерн MVC.

– Yii дотримується філософії простого і елегантного коду не намагаючись ускладнювати дизайн тільки заради проходження будь-якими шаблонами проектування.

– Yii є full-stack фреймворком і включає в себе перевірені і добре зарекомендовані в себе можливості: ActiveRecord для реляційних і NoSQL баз даних, підтримку REST API, багаторівневе кешування та інше.

– Yii відмінно масштабований. Можна налаштувати або замінити практично будь-яку частину основного коду. Використовуючи архітектуру розширень, легко ділитися кодом або використовувати код спільноти.

– Одна з головних цілей Yii – продуктивність.

Розглянемо також не менш популярний фреймворк Laravel 7. Laravel – це фреймворк для веб-додатків з виразним і елегантним синтаксисом. Він дозволить спростити вирішення основних наболілих завдань, таких як автентифікація, маршрутизація, сесії і кешування. Laravel – це спроба об'єднати все найкраще, що є в інших PHP фреймворк, а також Ruby on Rails, ASP.NET MVC і Sinatra.

Laravel – доступний, але потужний. Має безліч відмінних інструментів для великих, надійних додатків:

– Як і багато інших PHP фреймворків, для організації коду Laravel використовує архітектурний патерн MVC.

– Laravel дотримується філософії виразного, красивого синтаксису. Він призначений для людей, які цінують елегантність, простоту і читаність.

– Функціонал Laravel є простим для розуміння і використання.

– Більшість функцій Laravel чудово працюють, не вимагаючи додаткових налаштувань. Вони спираються на загальноприйняті стандарти написання коду, роблячи його інтуїтивно зрозумілим.

– Документація Laravel закінчена і постійно оновлюється.

– Чудова IoC (Інверсія управління).

– Зручна система міграцій.

– Інтегрована система модульного тестування.

Обидва фреймворка для організації коду використовують архітектурний патерн MVC. При детальному розгляді ці фреймворки складаються з трьох основних компонентів:

– Model – це сам додаток який нічого не знає про HTTP запитах.

– View – це HTTP запит / відповідь і уявлення даних, які вимагає клієнт від сервера.

– Controller – це кілька і більше класів, чиє завдання – абстрагування моделі від HTTP.

За архітектурою дані фрейморкі ідентичні, так як реалізують один і той же архітектурний патерн. Він дозволяє швидко реалізувати проект, а також легко супроводжувати і масштабувати його в подальшому.

Розширення важлива частина фреймворку, тому розглянемо їх докладніше. Розширення – це можливість підключати додаткові модулі або бібліотеки для фрейморка, які дозволять розширити його можливості. Yii і Laravel підтримують таку можливість. На даний момент, розширень набагато більше, звичайно ж, у Yii, так як фреймворк живе набагато довше, ніж Laravel, однак другий, в свою чергу, розвивається дуже великими темпами і включає в себе багато всього для роботи відразу «з коробки».

Розширення є абсолютно для будь-якого завдання, будь то панель адміністратора або ж платіжна система.

Чимало важливим фактором при виборі фреймворка є його документація. Через малу кількість років розробки або особистих переконань авторів документація Laravel дуже мізерна і неінформативна. Тому для навчання доведеться дивитися вихідний код, і як наслідок, не завжди буде зрозуміло, що робить та чи інша функція. В Yii ж є велика і корисна документація, де описана кожна функція із зазначенням її призначення і приклад використання. Так що, безсумнівно, це величезний плюс фреймворка Yii.

Для належної роботи сайту важлива підтримка REST API. REST – це архітектурний стиль взаємодії між компонентами розподілених додатків у мережі. REST – це набір послідовних обмежень, які слід враховувати при проектуванні розподіленої системи гіпермедіа. У деяких випадках (інтернет-магазини, пошукові системи, інші системи на основі даних) це може підвищити продуктивність та спростити архітектуру. Загалом, компоненти REST взаємодіють із клієнтами та серверами у всесвітній мережі.

В Інтернеті віддалені процедурні виклики можуть бути звичайними HTTP-запитами (зазвичай "GET" або "POST"; це називається "REST-запит"), а необхідні дані передаються як параметри запиту.

Для веб-служб, які відповідають REST тобто, не порушуючи жодних обмежень, що застосовуються до нього, використовується термін "RESTful". На відміну від веб-служб на основі SOAP, веб-API RESTful не має "офіційного" стандарту. Справа в тому, що REST – це архітектурний стиль, а SOAP – протокол. Хоча сам REST не є стандартом, більшість реалізацій RESTful використовують стандарти HTTP, URL, JSON та XML.

В кінцевому рахунку, в даний час всі сучасні PHP фреймворки зобов'язані підтримувати REST архітектуру. Нижче наведено список можливостей фреймворків для роботи з REST API.

Yii 2:

- Підтримка JSON, JSONP і XML.
- Роутинг відповідно до REST-запитами.
- Підтримка HATEAOS.
- Кешування запитів.
- Обмеження швидкості.

Laravel:

- Налаштування роутінга REST-запитів.

– Підтримка JSON, JSONP.

На основі проведеного аналізу був зроблений вибір на сторону, що активно розвивається Laravel, його структура дозволяє легко масштабувати веб-додаток. Yii є більш стабільним, але менш масштабованим.

2.2.3 Розробка бази даних

Для розробки бази даних використовують такі інструменти, як MySQL, Oracle чи SQL Server, щоб знаходити, зберегти або відредагувати дані та повернути їх користувачеві за допомогою коду візуальної частини.

Вище перелічені мови використовуються не лише для створення веб-сайтів, програмного забезпечення та додатків; вони також використовуються для створення та управління базами даних.

Бази даних не розроблені для розуміння тих же мов, на яких запрограмовані програми, тому важливо мати мову, яку вони розуміють – як SQL, стандартну мову для доступу та маніпулювання реляційними базами даних. SQL означає структурована мова запитів.

Вона має власну розмітку і в основному дозволяє програмістам працювати з даними, що зберігаються в системі баз даних. Розглянемо детальніше різні типи, переваги та недоліки баз даних (табл.1.6).

Таблиця 2.2 – Бази даних

№	Назва	Переваги	Недоліки
	Oracle	Має останні інновації та функції, що надходять від їх продукції, оскільки Oracle прагне встановити планку для інших інструментів управління базами даних.	Вартість Oracle може бути непосильною, особливо для менших організацій.

№	Назва	Переваги	Недоліки
		Інструменти управління базами даних Oracle також надзвичайно надійні, і ви можете знайти той, який може робити практично все, про що ви можете подумати.	Після встановлення система може вимагати значних ресурсів, тому для впровадження Oracle може знадобитися оновлення обладнання.
2	MySQL	Безкоштовний.	Ви можете витратити багато часу і сил, щоб змусити MySQL робити те, що інші системи роблять автоматично.
		Пропонує безліч функціональних можливостей.	Немає вбудованої підтримки для XML або OLAP.
		Існують різноманітні інтерфейси користувачів.	Підтримка доступна для безкоштовної версії, але за неї потрібно заплатити.
		Це може бути зроблено для роботи з іншими базами даних, включаючи DB2 та Oracle.	
3	Microsoft SQL Server	Це дуже швидко і стабільно.	Ціни на підприємства можуть перевищувати багато організацій.

№	Назва	Переваги	Недоліки
		Пропонує можливість регулювання та відстеження рівнів продуктивності, що може зменшити використання ресурсів.	Навіть при налаштуванні продуктивності Microsoft SQL Server може обробляти ресурси.
		Є можливість отримати доступ до візуалізації на мобільних пристроях.	Багато людей мають проблеми з використанням інтеграційних служб SQL Server для імпорту файлів.
		Дуже добре працює з іншими продуктами Microsoft.	
4	PostgreSQL	Ця система управління базами даних є масштабованою і може обробляти терабайти даних.	Не чітка документація.
		Підтримує JSON.	Конфігурація може бути заплутаною.
		Існують різноманітні заздалегідь визначені функції.	Швидкість може постраждати під час великих масових операцій чи запитів.
		Доступна низка інтерфейсів.	

№	Назва	Переваги	Недоліки
5	MongoDB	Швидка і проста у використанні.	Установки за замовчуванням не захищені
		Підтримує JSON та інші документи NoSQL.	Налаштування може бути тривалим процесом.
		Дані будь-якої структури можна зберігати та отримувати доступ до них швидко та легко.	Інструменти для перекладу SQL на запити MongoDB доступні, але вони додають додатковий крок до використання системи.
		SQL не використовується як мова запитів.	
		Система швидка і стабільна.	

В ході виконання даного проекту, вибір було зроблено на користь MySQL. По-перше, дана СУБД є безкоштовною. По-друге, супутній продукт MySQL Workbench дозволяє просто взаємодіяти зі встановленою БД, надаючи можливість через зручний інтерфейс користувача виконувати будь-які маніпуляції з даними.

Та нарешті, ця СУБД є однією з найбільш розповсюджених, займаючи друге місце за популярністю у світі, поступаючись лише корпоративній СУБД від Oracle. А це означає, що на випадок проблем чи збоїв в роботі цієї системи, в мережі можна знайти величезних кількість матеріалу для їх вирішення.

Для розробки клієнтської частини сайт було обрано мови програмування HTML, CSS і JavaScript так як вони є най популярнішими мовами для веб розробки і досить легкі у вивченні.

Розробка серверної частини буде відбуватись за допомогою мови програмування PHP з використанням фреймфорку Laravel. Ця мова активно використовується багатьма сайтами і досить легка у вивченні. Також найбільш популярні системи контролю контентом використовують саме цю мову програмування, що надає можливість розробляти додатковий функціонал.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Структура програмного додатку

У проектуванні інформаційної системи найважливішим етапом є саме розробка структури програмного додатку. Проектування інформаційних систем називається багатоступінчастим процесом створення та модернізації шляхом застосування впорядкованих методологій та різного інструментарію.

Якщо виділяти стадію проектування інформаційних систем в якості окремого етапу, то його можна розмістити між етапами аналізу і розробки. Однак на практиці чіткий поділ на етапи, як правило, ускладнене або неможливе, оскільки проектування, формально починаючи з визначення мети проекту, часто триває на стадіях тестування і реалізації.

Загальну структуру інформаційної системи можна розділити на декілька частин, а саме до та після авторизації. Головна відмінність – доступ до інформації та певних сторінок сайту.

До авторизації у системі користувачу відкриті деякі модулі (табл.3.1).

Таблиця 3.1 – Сторінки до авторизації

№	Сторінка	Опис
1	Головна	Сторінка із основною інформацією про інформаційну систему, головні переваги та етапи реєстрації.
2	Авторизація	Сторінка для авторизації зареєстрованих користувачів.
3	Реєстрація	Сторінка для реєстрації нових користувачів системи.

Після авторизації користувачеві відкриваються додаткові можливості. Перш за все, що саме залежить від типу акаунту – автовласник, майстер чи сервісний центр.

Панель меню складається із посилань на ключові модулі даної системи. Розглянемо детальніше, які можливості відкриті для кожного з типу акаунтів у табл.3.2.

Таблиця 3.2 – Функціонал за групами користувачів

№	Користувач	Можливості користувача
1	Адміністратор	Перегляд загальної інформації;
		Підтримка сайту;
		Блокування користувачів;
2	Звичайний користувач	Створення власної інформаційної сторінки;
		Редагування даних;
		Завантаження файлів;
		Обмін файлами;
		Відстежувати публікації інших користувачів;
		Залишити відгуки про файли;
		Написання повідомлення адміністратору.

Інформаційна система добре спроектована для якісної взаємодії всіх користувачів. Це добре продемонстровано у структурі інформаційної системи. Кожен користувач має можливість працювати з файлами, взаємодіяти з іншими користувачами та інше.

Крім того, було виконано всі задачі, поставлені на початку виконання даної практичної роботи.

3.2 Структурно-функціональне моделювання процесу

IDEF0 – це багато в чому дуже простий метод. Кожне поле представляє окремий процес, як і в інших підходах, але IDEF0 відрізняється використанням

та розміщенням стрілок. Крім звичайних входів і виходів, існують ще два типи стрілок, які представляють "елементи управління" та "механізми".

Елементи управління є формою введення, але які використовуються для керування діяльністю в процесі. Іноді виникає певна міра невизначеності щодо того, є елемент елементом введення чи контролю.

Розглянемо IDEF0 інформаційної системи (рис.3.1).

Рисунок 3.1 – IDEF0 інформаційної системи

Методологія IDEF1 розділяє елементів структури інформаційної галузі, їх властивості та взаємозв'язки на класи. Центральним поняттям методології IDEF1 є поняття сутності. Клас сутностей являє собою сукупність інформації, накопиченої і зберігається в рамках підприємства і відповідає певному об'єкту або групі об'єктів реального світу. Основними концептуальними властивостями сутностей в IDEF1 є:

– Стійкість. Інформація, що має відношення до тієї чи іншої суті постійно накопичується.

– Унікальність. Будь-яка сутність може бути однозначно ідентифікована з іншої сутності.

Кожна сутність має своє ім'я і атрибути. Атрибути представляють собою характерні властивості і ознаки об'єктів реального світу, які стосуються певної сутності. Клас атрибутів являє собою набір пар, що складаються з імені атрибута і його значення для певної сутності. Атрибути, за якими можна однозначно відрізнити одну сутність від іншої називаються ключовими атрибутами.

Кожна сутність може характеризуватися декількома ключовими атрибутами. Клас взаємозв'язків в IDEF1 являє собою сукупність взаємозв'язків між сутностями. Взаємозв'язок між двома окремими сутностями вважається існуючої в тому випадку, клас атрибутів однієї сутності містить ключові атрибути іншої сутності. Кожен з вищеописаних класів має своє умовне графічне відображення, згідно з методологією IDEF1. Розглянемо IDEF1 інформаційної системи (рис.3.2).

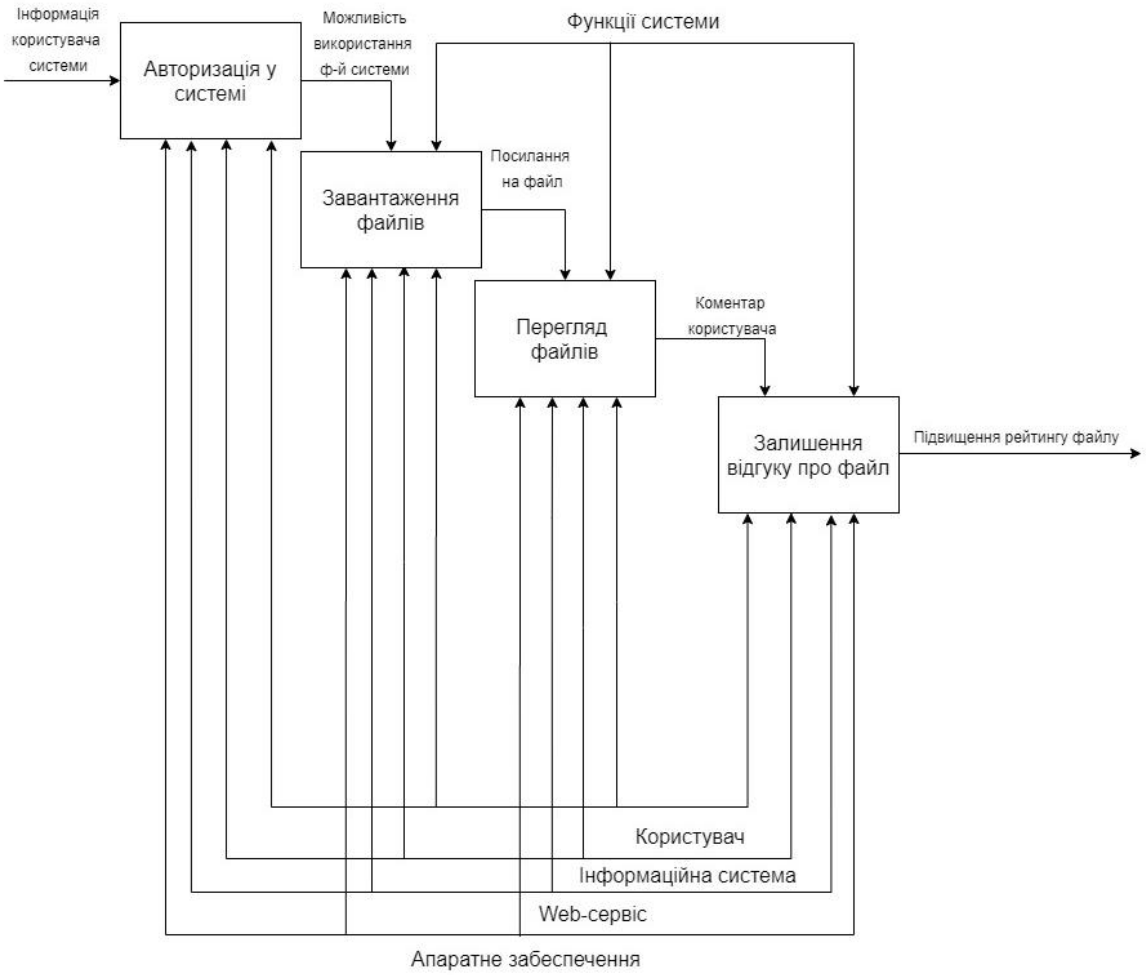

Рисунок 3.2 – IDEF1 проекту

3.3 Моделювання діаграми варіантів використання

Метою діаграми використання є відображення динамічного аспекту системи. Однак це визначення є занадто загальним, щоб описати мету, оскільки інші чотири діаграми (діяльність, послідовність, співпраця та діаграма стану) також мають те саме призначення. Ми розглянемо якесь конкретне призначення, яке відрізнятиме його від інших чотирьох діаграм.

Діаграми випадків використання використовуються для збору вимог системи, включаючи внутрішні та зовнішні впливи. Ці вимоги в основному є вимогами дизайну. Отже, коли система аналізується для збору її функціональних можливостей, готуються випадки використання та визначаються дійові особи.

Коли початкове завдання виконане, діаграми використання моделюються для подання зовнішнього виду.

Цілі діаграм випадків використання можуть бути такими:

– Використовується для збору вимог системи.
– Використовується для отримання зовнішнього вигляду системи.
– Визначте зовнішні та внутрішні фактори, що впливають на систему.
– Показати взаємодію між вимогами акторів.

Розглянемо детальніше акторів та варіанти використання в табл.3.3 та табл.3.4.

Табл. 3.4 – Опис акторів

№	Назва	Опис
1	Користувач	Користувач, що має можливість завантажувати та обмінюватись файлами.
3	Адміністратор	Користувач, що має найбільші можливості на сайті, а саме редагувати та видаляти дані на сайті.

Табл. 3.5 – Опис варіантів використання

№	Назва	Опис
1	Авторизація	Модуль дозволяє авторизуватися на сайті.
2	Реєстрація	Модуль дозволяє зареєструватися новому користувачеві на сайті.
3	Редагування інформації на сайті	Дозволяє змінювати загальну інформацію на сайті.
4	Перегляд завантажених файлів	Модуль дозволяє переглядати завантажені користувачами файли.

№	Назва	Опис
5	Завантаженні файлів	Модуль, що дозволяє користучвачам завантажвати файли та змінювати інформацію про них.
9	Залишення відгуків про файли	Кожен користувач який має доступ до файлу має можливість залишати відгуки про файл.
12	Редагування даних на сторінці	Кожен користувач має можливість редагувати особисту інформацію на сайті.

Сформувавши представлення про інформацію та функціонал інформаційної системи було сформовано діаграма варіантів використання (рис.3.3).

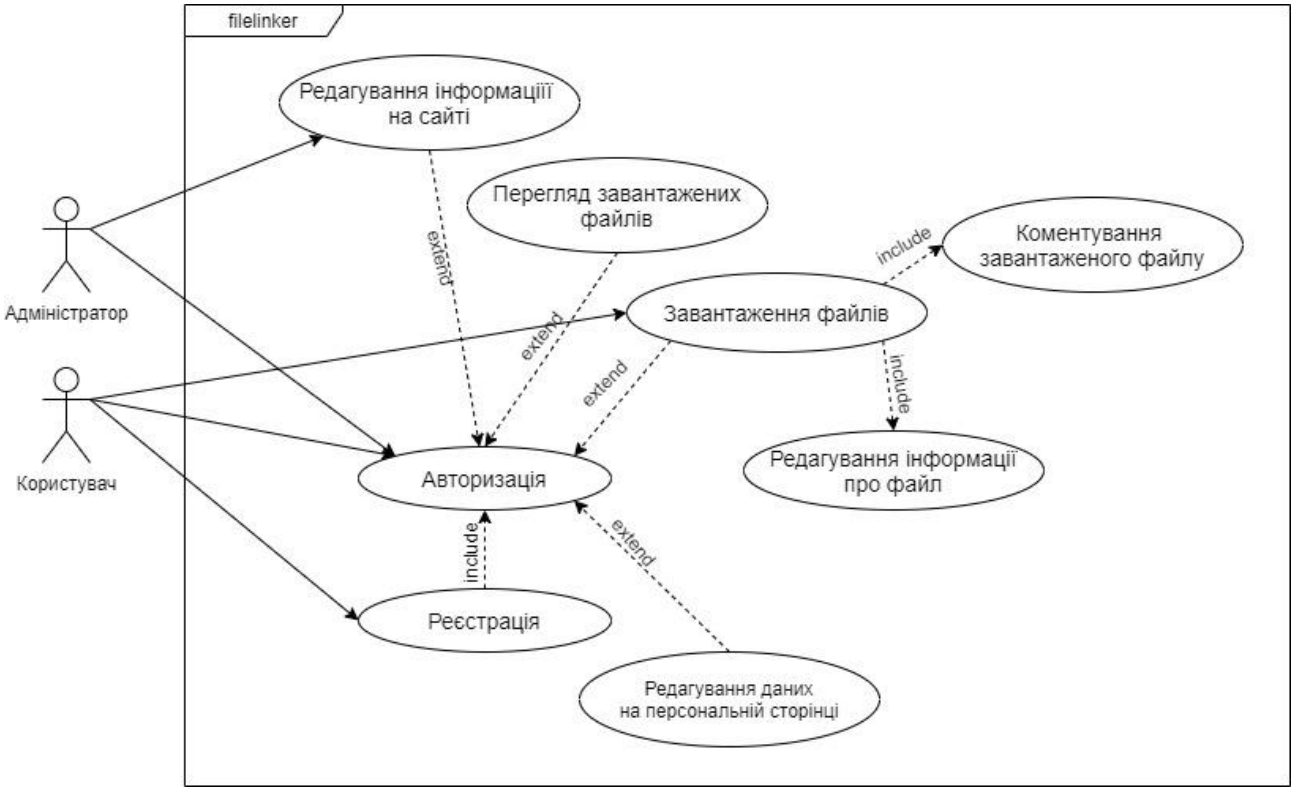

Рисунок 3.3 – Діаграма варіантів використання

3.4. Моделювання діаграм діяльності

Діаграма діяльності – це технологія, що дозволяє описувати логіку процедур, бізнес-процеси і потоки робіт. У багатьох випадках вони нагадують блок-схеми, але принципова різниця між діаграмами діяльності і нотацією блок-схем полягає в тому, що перші підтримують паралельне процеси.

Діаграма діяльності дозволяє будь-кому, хто виконує цей процес, вибирати порядок дій. Іншими словами, діаграма тільки встановлює правила обов'язкової послідовності дій, яким я повинен слідувати. Це важливо для моделювання бізнес-процесів, оскільки ці процеси часто виконуються паралельно. Такі діаграми також корисні при розробці паралельних алгоритмів, в яких незалежні потоки можуть виконувати роботу паралельно.

На рисунку 3.4-8 зображено діаграми діяльності модулів інформаційної системи фріланс-біржі.

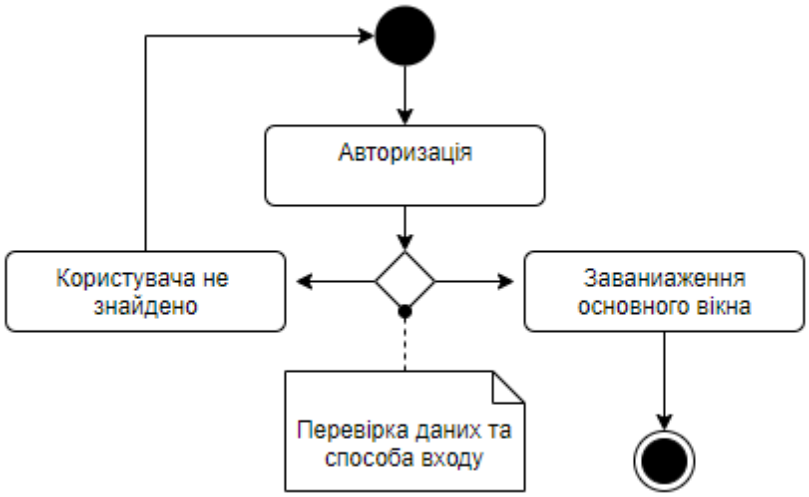

Рисунок 3.4 – Діаграма діяльності модулю авторизації

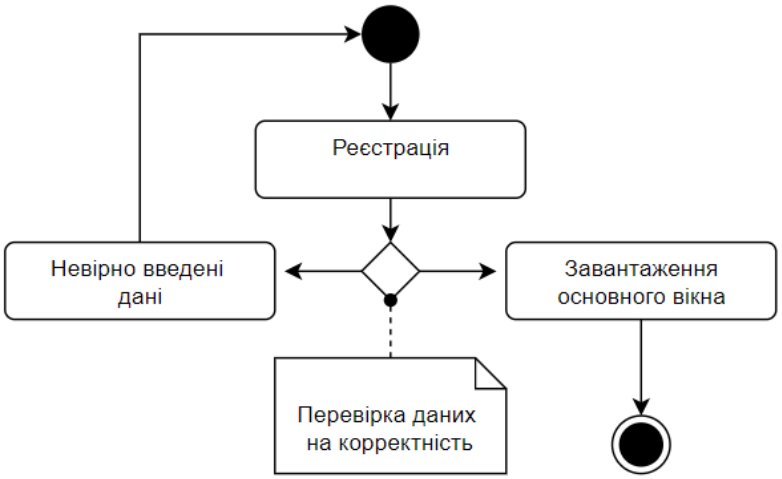

Рисунок 3.5 – Діаграма діяльності модулю реєстрації

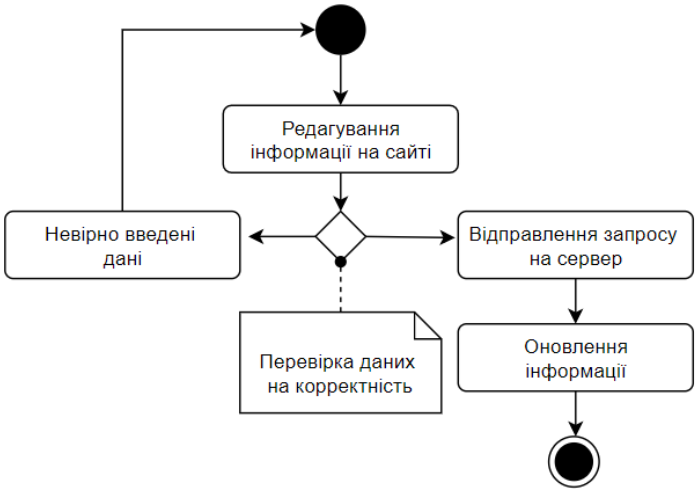

Рисунок 3.7 – Діаграма діяльності модулю перегляду інформації

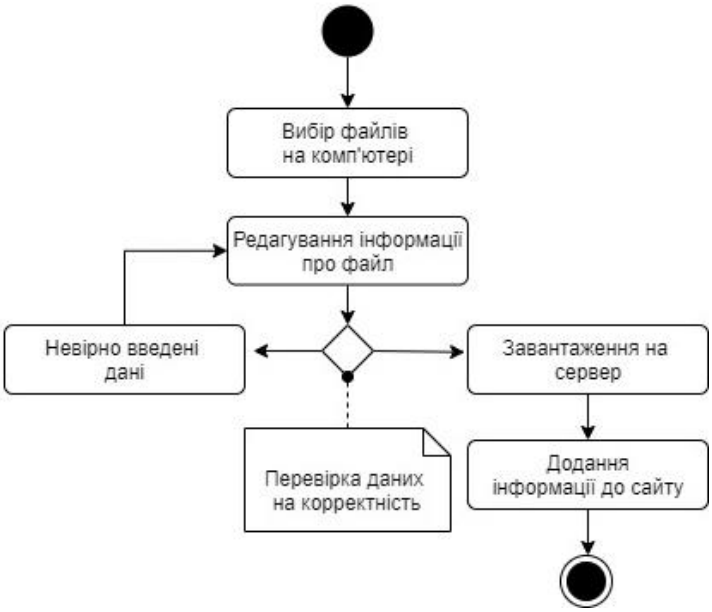

Рисунок 3.8 – Діаграма діяльності модулю завантаження файлів

3.5. Проектування бази даних

Основні користувачі системи – замовник та фахівець. За допомогою цієї системи вони можуть отримувати різні відомості про виконавців та замовників, послуги та інше.

Зобразимо відносини в ER-діаграмі (рис.3.9). Проаналізувавши сутність, використовувані в моделі інформаційної системи, перейдемо до реалізації структури БД. Для цього представимо імена необхідних таблиць, атрибутів, типів, їх призначення та обмеження в табл.3.6.

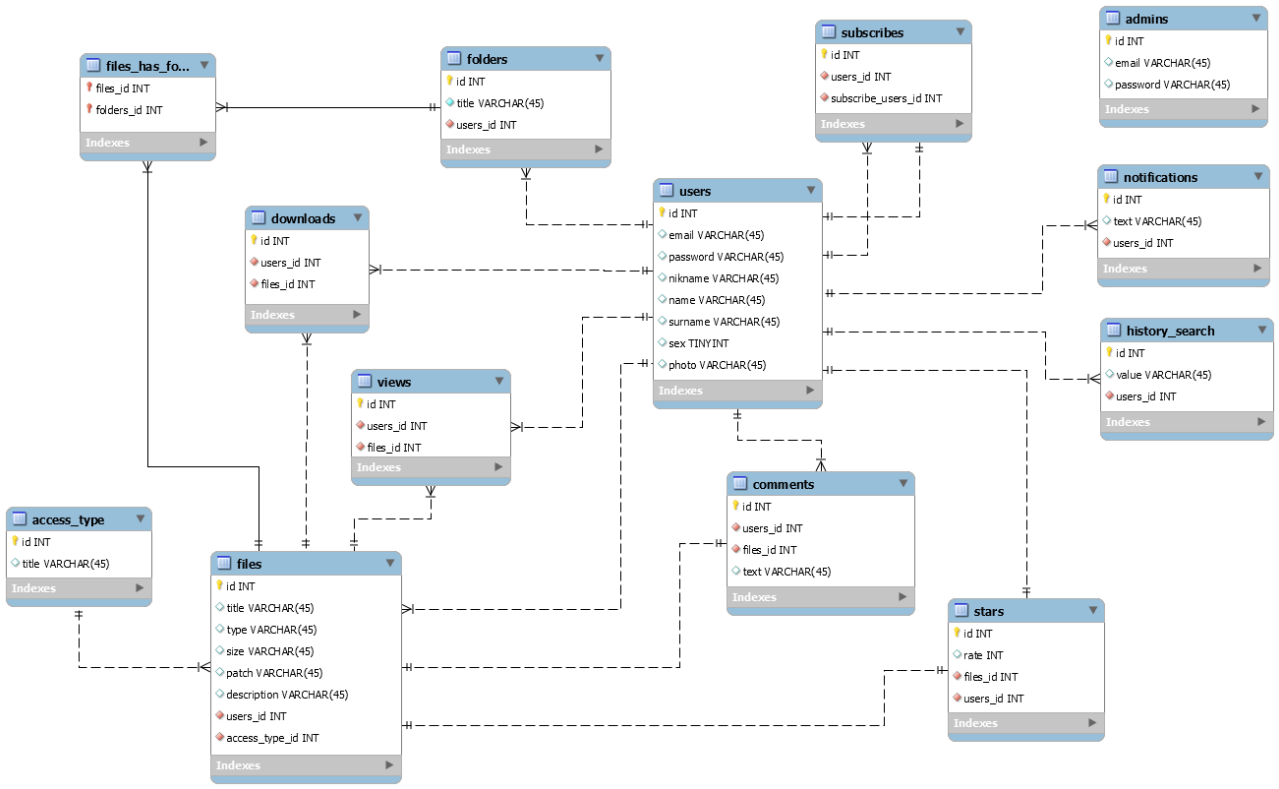

Рисунок 3.9 – ER-діаграма

Таблиця 3.6 – Інформація за таблицями ER-діаграми

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
1	users	id	Ідентифікатор аккаунта	INTEGER	PK	Не пустий
		email	Електронна пошта	VARCHAR(100)		Не пустий
		password	Пароль від аккаунта	VARCHAR(100)		Не пустий
		nikname	Псевдонім користувача	VARCHAR(100)		Не пустий
		name	Ім’я користувача	VARCHAR(100)		Не пустий
		surmane	Прізвище користувача	VARCHAR(100)		Не пустий
		sex	Пол	BOOLEAN		Не пустий
		photo	Фото користувача	VARCHAR(100)		Не пустий
2	Subscribes	user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		subscribe_users_id	Ідентифікатор аккаунта на який виконана підписка	INTEGER	FK	Не пустий
3	Notifications	id	Ідентифікатор повідомлення	INTEGER	PK	Не пустий
		Text	Текст повідомлення	TEXT		Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
4	history_search	id	Ідентифікатор пошукового запиту	INTEGER	PK	Не пустий

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
		Value	Пошуковий запит	VARCHAR(100)		Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
4	Folders	id	Ідентифікатор папки	INTEGER	PK	Не пустий
		Title	Назва папки	VARCHAR(100)		Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
5	files_has_folders	files_id	Ідентифікатор файлу	INTEGER	FK	Не пустий
		folders_id	Ідентифікатор папки	INTEGER	FK	Не пустий
6	files	id	Ідентифікатор файлу	INTEGER	PK	Не пустий
		title	Назва файлу	VARCHAR(100)		Не пустий
		Type	Тип файлу	VARCHAR(10)		Не пустий
		Size	Розмір файлу	FLOAT		Не пустий
		Patch	Посилання на файл	VARCHAR(100)		Не пустий
		Description	Опис файлу	TEXT		
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		access_type_id	Ідентифікатор типу доступу	INTEGER	FK	Не пустий
7	access_type	id	Ідентифікатор рівня доступу	INTEGER	PK	Не пустий
		Title	Назва типу доступу	VARCHAR(100)		Не пустий

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
8	Comments	id	Ідентифікатор коментару	INTEGER	PK	Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		Text	Текст комнтару	TEXT		Не пустий
9	Stars	id	Ідентифікатор балу	INTEGER	PK	Не пустий
		Rate	Кількість балів	INTEGER		Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
10	views	id	Ідентифікатор перегляду	INTEGER	PK	Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
11	downloads	id	Ідентифікатор завантаження	INTEGER	PK	Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		user_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1 Програмна реалізація

Розробка буде відбуватись у безкоштовному текстовому редакторі VS Code. Visual Studio Code – це крос-платформенний редактор скриптів, створений корпорацією Майкрософт. Поряд з розширенням PowerShell він надає широкі інтерактивні можливості редагування скриптів, спрощуючи написання надійних скриптів PowerShell.

Також для збереження вихідного коду, а також відслідковування версій сайту було використано систему контролю версій Git. Системи контролю версій дозволяють розробникам зберігати всі зміни, внесені в код [20]. Тому в разі, помилки, вони можуть просто відкотити код до робочого стану замість того, щоб витрачати години на пошуки маленької помилки або помилок, які ламають весь код.

Системи контролю версій також дають можливість декільком розробникам працювати над одним проектом і зберігати внесені зміни, щоб переконатися, що всі можуть стежити за тим, над чим вони працюють.

Проект буде розроблятись на віртуальному веб сервері OpenServer. Open Server Panel – це портативна серверна платформа і програмне середовище, створена спеціально для веб-розробників з урахуванням їх рекомендацій і побажань. Також буде використаний менеджер модулів для PHP – Composer.

Розробка сайту починається з створення нового проекту з використанням фреймворку laravel. Для цього в терміналі Composer необхідно написати наступну команду:

```
composer create-project --prefer-dist laravel/laravel filelinker
```

Де «filelinker» – ім'я проекту і може бути змінено на будь-яке. Після чого буде створено базову архітектуру проекту (рис. 4.1).

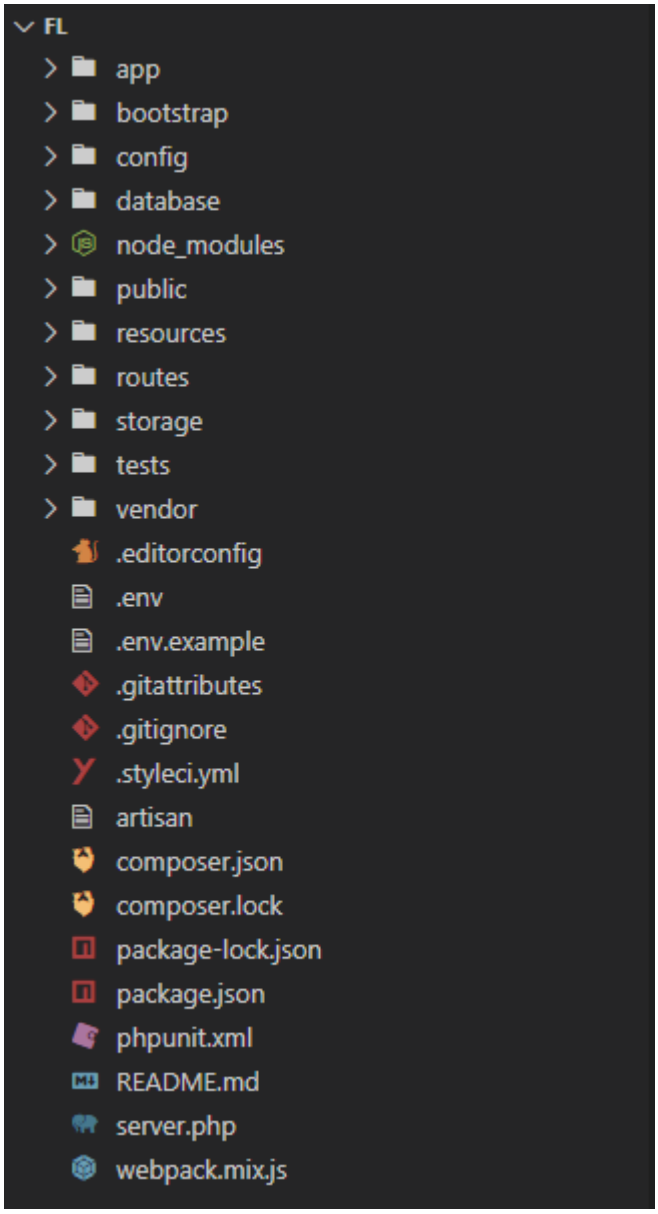

Рисунок 4.1 – Архітектура проекту

Таблиця 4.1 – Таблиця опису директорій проекту

№	Пакет	Короткий опис
1	«app»	Містить код ядра додатку.
2	«bootstrap»	Містить файли, які завантажують фреймворк і налаштовують автозагрузку.
3	«config»	Містить всі конфігураційні файли додатку
4	«database»	Містить міграції і класи для наповнення початковими даними БД.

5	«public»	Містить файл index.php, який є вхідною точкою для всіх запитів, що надходять в додаток. Також ця папка містить ресурси, такі як зображення, JavaScript, CSS.
6	«resources»	Містить початковий код, а також сирі, некомпільовані ресурси, такі як LESS, SASS, JavaScript.
7	«routes»	Містить всі визначення маршрутів додатку.
8	«storage»	Містить скомпільовані Blade-шаблони, файл-сесії, кешу файлів і інші файли, створені фреймворком.
9	«tests»	Містить автотести.

Потім починається розробка клієнтської частини сайту, а саме створення всіх необхідних компонентів, налаштування маршрутів доступу і верстка шаблонів та розробка модулів серверної частини сайту, створення всіх необхідних контролерів, моделей та налаштування API роутеру.

4.1.1 Розробка клієнтської частини

Написання сайту починається із створення «скелету» з використанням мови розмітки HTML. Після чого за допомогою мови каскадних стилів CSS формується зовнішній вигляд веб-сайту і надається візуальна форма елементів для зручного користування.

Потім створений шаблон розбивається на окремі компоненти, що надає можливість багаторазово використовувати один компонент на різних сторінках сайту. Приклад компоненту головної сторінки наведено в додатку В. Всі компоненти знаходяться в папці resources/js/.

Наступним кроком створюється єдина точка входу яка підключає всі необхідні компоненти і модулі JavaScript.

```
require('./bootstrap');
```

```
import Vue from 'vue'
import vuetify from './plugins/vuetify'
import router from "./routes"
import store from "./store"

import AppComponent from './views/site/App';

Vue.prototype.$http = axios;

const token = localStorage.getItem('token');
if (token) {
    Vue.prototype.$http.defaults.headers.common['Authorization'] = token;
}



const app = new Vue({
    el: '#app',
    components: {
        AppComponent
    },
    vuetify,
    store,
    router
});
```

Створивши всі необхідні шаблони і компоненти. Відбувається налаштування маршрутів для навігації між сторінками сайту. Далі наведено приклад коду маршруту для головної сторінки.

```
export default new Router({
    mode: "history",
    routes: [
        {
```

```
            path: '/',
            name: 'index',
            component: Index
        },
    ]
});
```

В результаті при кліку на посилання відбувається перехід на необхідний компонент вказаний в маршрутизаторі.

4.1.2 Розробка серверної частини

Розробка серверної частини починається зі створення та підключення бази даних. Для цього необхідно перейти до налаштувань підключення до бази даних. Необхідно змінити файл (.env), він знаходиться в кореневій папці проекту. Необхідно змінити 4 рядки: DB_HOST = localhost, DB_DATABASE = filelinker, DB_USERNAME = root, DB_PASSWORD = secret. Де DB_HOST – ім'я хоста, DB_DATABASE – ім'я бази даних, DB_USERNAME – ім'я користувача для доступу до бази даних, DB_PASSWORD – пароль користувача для доступу до бази даних.

Потім створюються міграції для того щоб автоматично було створено базу даних. Міграції – це система контролю версій для бази даних. Вони дозволяють команді програмістів змінювати структуру БД, в той же час залишаючись в курсі змін інших учасників. Далі наведено приклад міграції для створення таблиці користувачів.

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('email')->unique();
        $table->string('password');
```

```
            $table->string('nickname')->unique();
            $table->string('name');
            $table->string('surname');
            $table->boolean('sex');
            $table->string('photo')->default('/img/no-image.png');
            $table->boolean('is_admin')->default(0);
            $table->timestamps();
        });
    }
```

Після створення всіх необхідних міграцій потрібно написати в термінали Composer наступну команду:

```
php artisan:migrate
```

Далі створюються моделі для зручної роботи з базою даних, налаштувань залежностей між таблицями і підключення додаткових модулів. Далі наведено приклад моделі користувачів.

```
class User extends Authenticatable
{
    use Notifiable, HasApiTokens;

    protected $table = 'users';

    protected $fillable = [
        'email',
        'password',
        'nickname',
        'name',
        'surname',
        'sex',
        'photo',
```

```
        'is_admin'
    ];

    protected $hidden = [
        'password',
    ];

    function feed() {
        return $this->hasMany('App\Models\Feed', 'users_id');
    }

    function files() {
        return $this->hasMany('App\Models\Files', 'users_id')->where('basket', 0);
    }

    function subscribes() {
        return $this->HasMany('App\Models\Subscribes', 'subscribe_users_id');
    }

    function mySubscribes() {
        return $this->HasMany('App\Models\Subscribes', 'users_id');
    }
}
```

Всі запити що надходять на сервер обробляються API маршрутизатором, що знаходиться в файлі routes/api.php. Маршрутизація визначає, як додаток відповідає на клієнтський запит до конкретної адреси.

Потім в залежності від типу запиту підключається необхідний контролер з необхідним методом який опрацює дані отримані з бази даних. Контролер – з'єднує моделі, види і інші компоненти необхідні для роботи системи. А також відповідає за обробку запитів користувача, а саме отримання і обробку даних та

відправку готового результату клієнтській частині системи. Приклад такого контролеру для додання коментару :

```
class FileController extends Controller
{
    function commentFile(Request $request, $id) {
        $model = new Comments();
        $model->create([
            "files_id" => $id,
            "users_id" => Auth::id(),
            "text" => $request->comment
        ]);
        return response('ok', 200);
    }
}
```

Детальнішу інформацію можна знайти у Додатку В.

4.2 Використання програмного додатку

Робота з додатком розпочинається з головної сторінки, де користувач має можливість продивитися інформацію про сайт, та авторизуватися (рис.4.1-4.2).

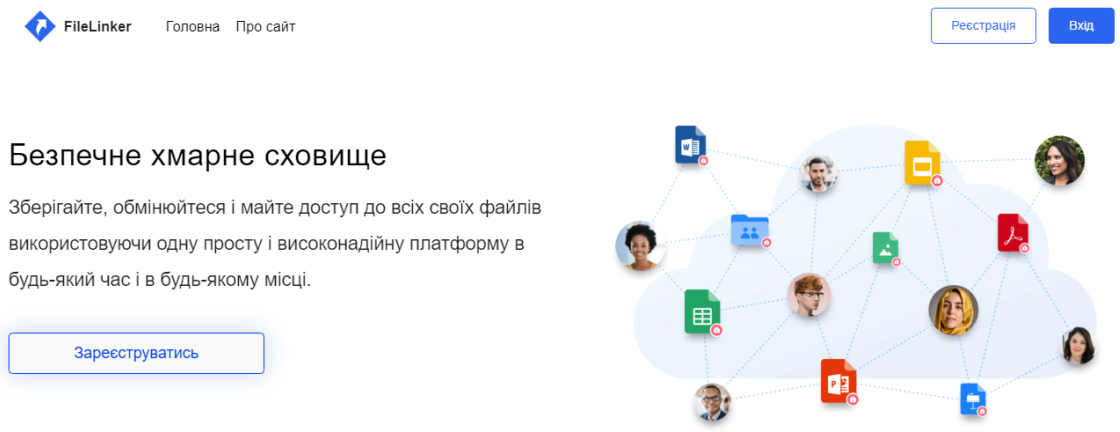

Рисунок 4.1 – Головна сторінка

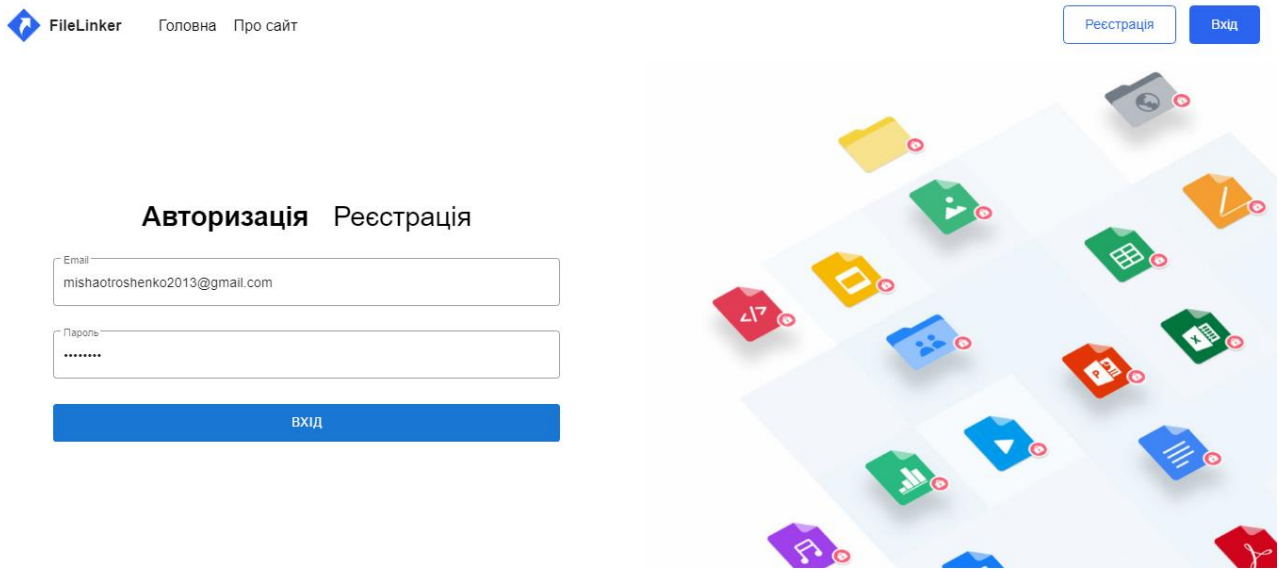

Рисунок 4.2 – Сторінка авторизації

Доступ до всіх функцій сайту мають лише авторизовані користувачі. Тому перед початком роботи потрібно зареєструватись.

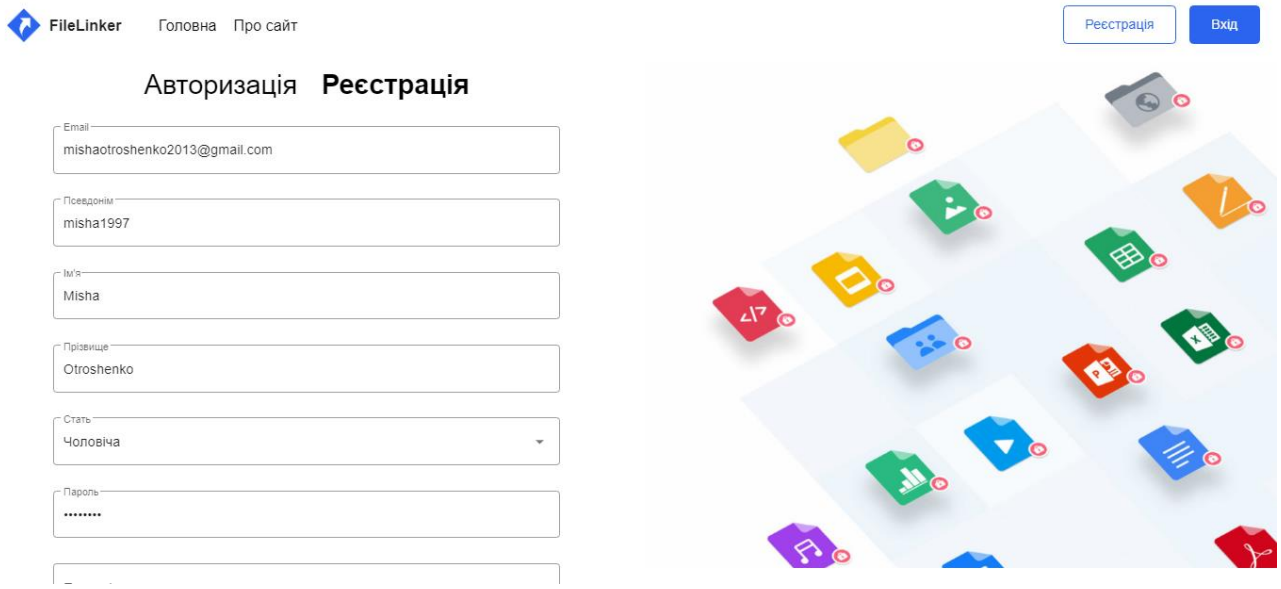

Рисунок 4.3 – Сторінка реєстрації

Після реєстрації або авторизації відкривається сторінка з власним кабінетом користувача. На сторінці є можливість відредагувати власну інформацію, змінити фото та пароль (рис.4.4).

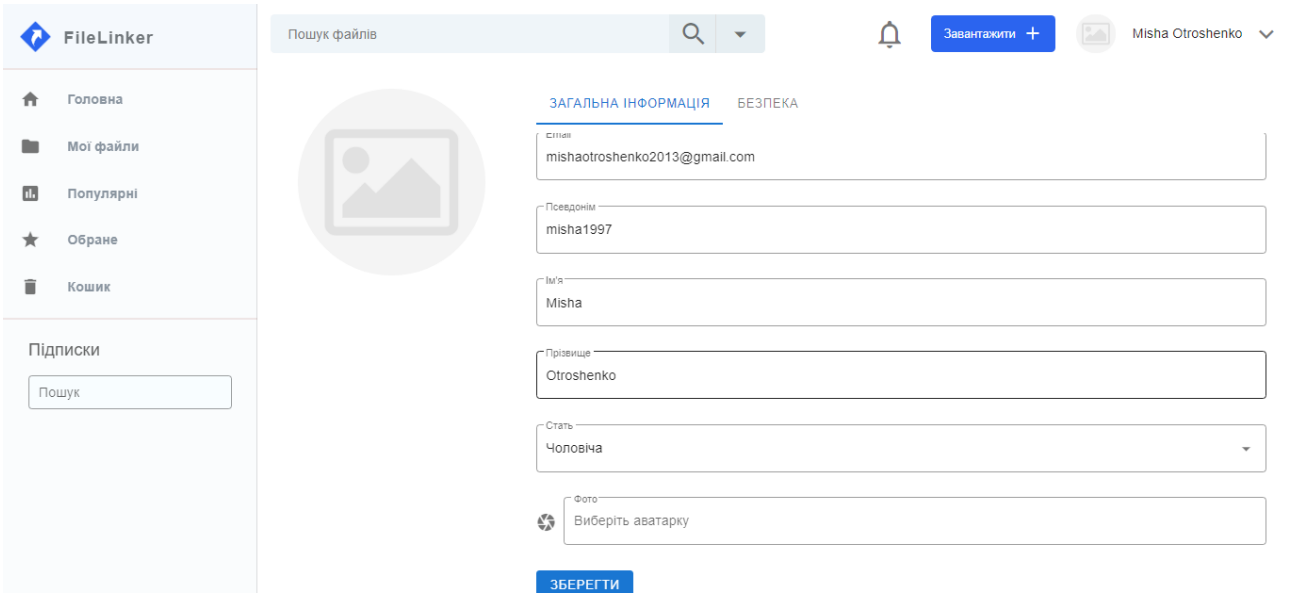

Рисунок 4.4 – Сторінка кабінету

Головна функція сайту це завантаження та обмін файлами. Для завантаження нових файлів необхідно натиснути кнопку «Завантажити +». Після натискання з’явиться форма для вибору файлів на комп’ютері (рис.4.5).

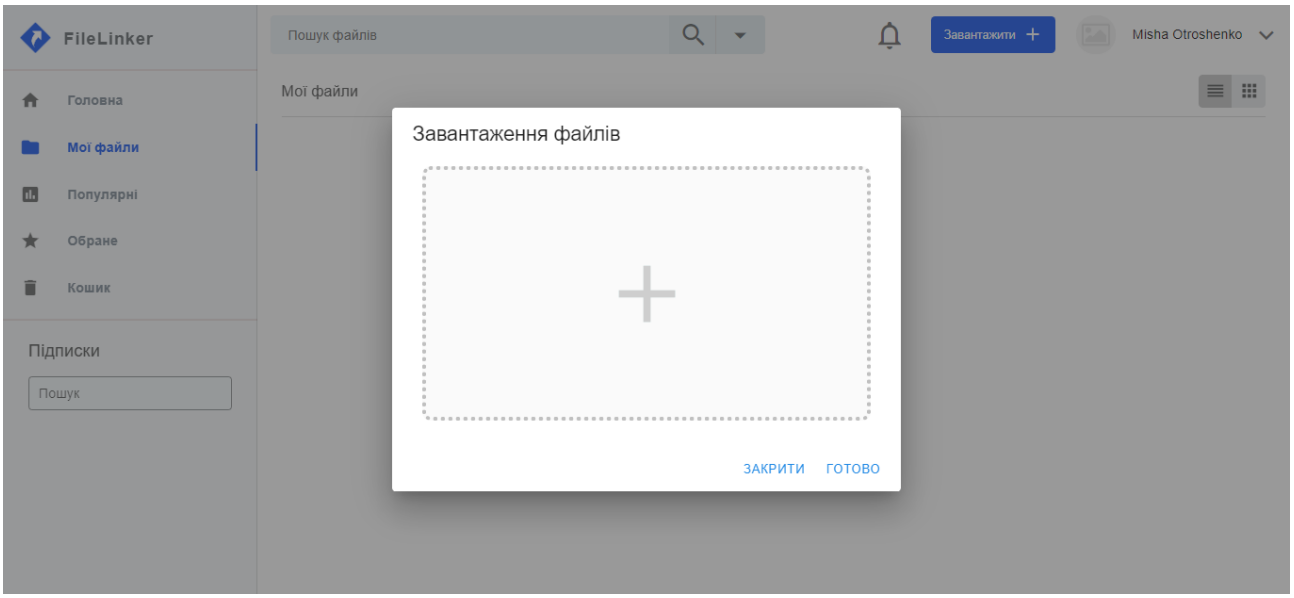

Рисунок 4.5 – Форма завантаження файлів

Після вибору бажаних файлів відобразиться список файлів з додатковими тестовими полями. Ці поля надають можливість змінювати назву та додати опис кожного з файлу (рис.4.6).

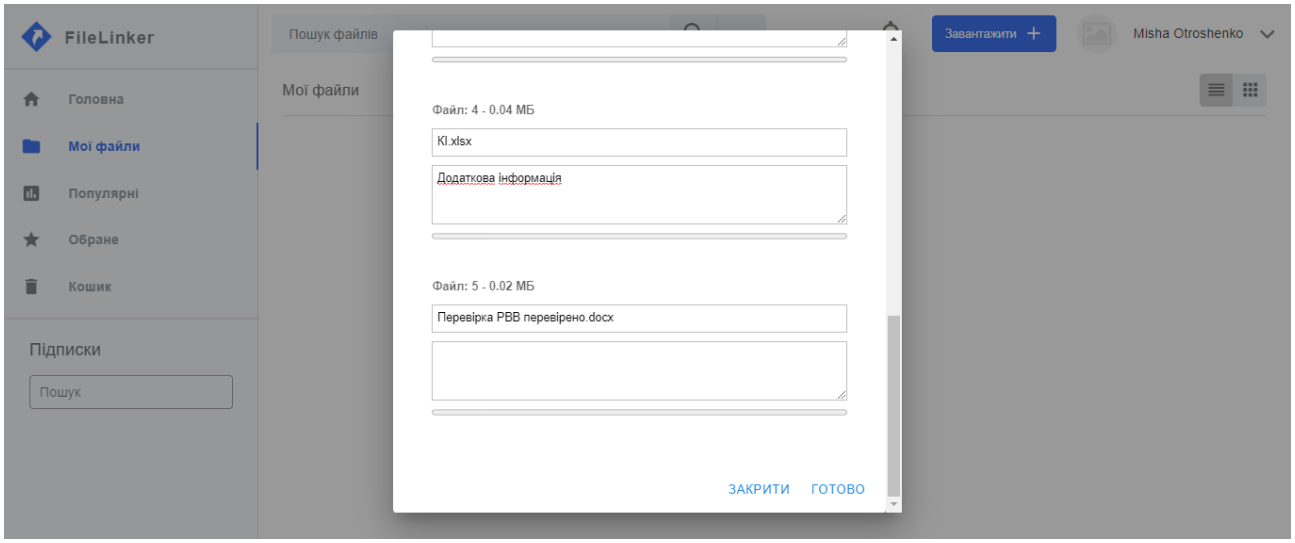

Рисунок 4.6 – Додання інформації про файли

Натиснувши кнопку «Готово» з’явиться повідомлення про успішну публікацію файлів та збереження даних.

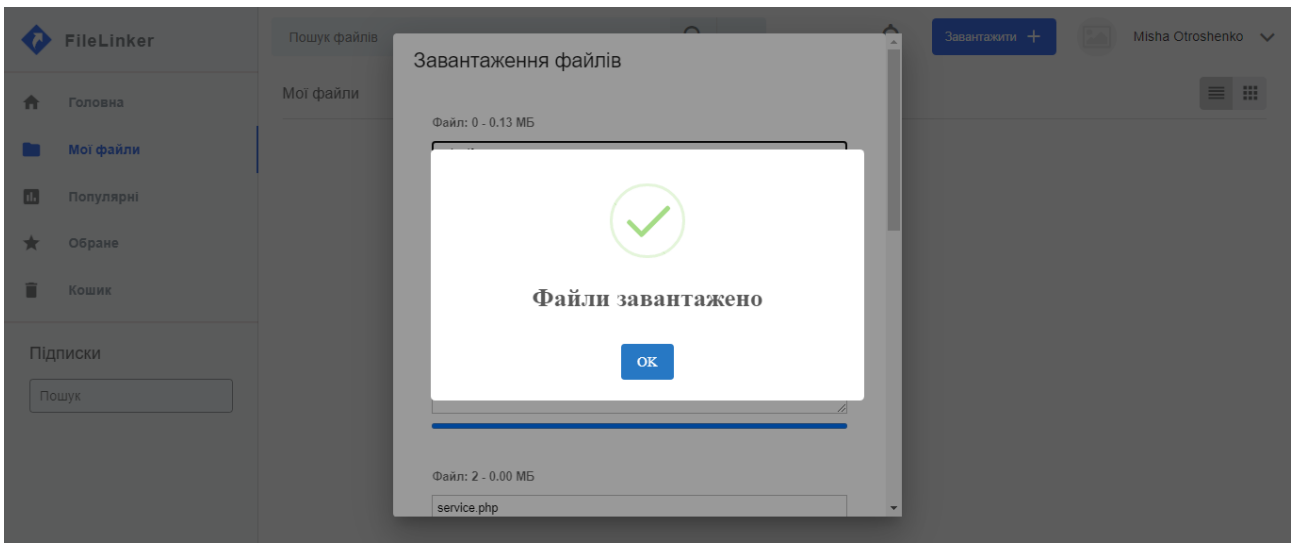

Рисунок 4.7 – Повідомлення про успішне завантаження

Всі завантажені користувачем файли можна побачити на сторінці «Мої файли». Кожна з сторінок надає можливість змінювати вид відображення файлів, а саме списком, або блоками. Для кожного з типів файлів було створено відповідне зображення для швидкого визначення типу файлу (рис. 4.8, 4.9).

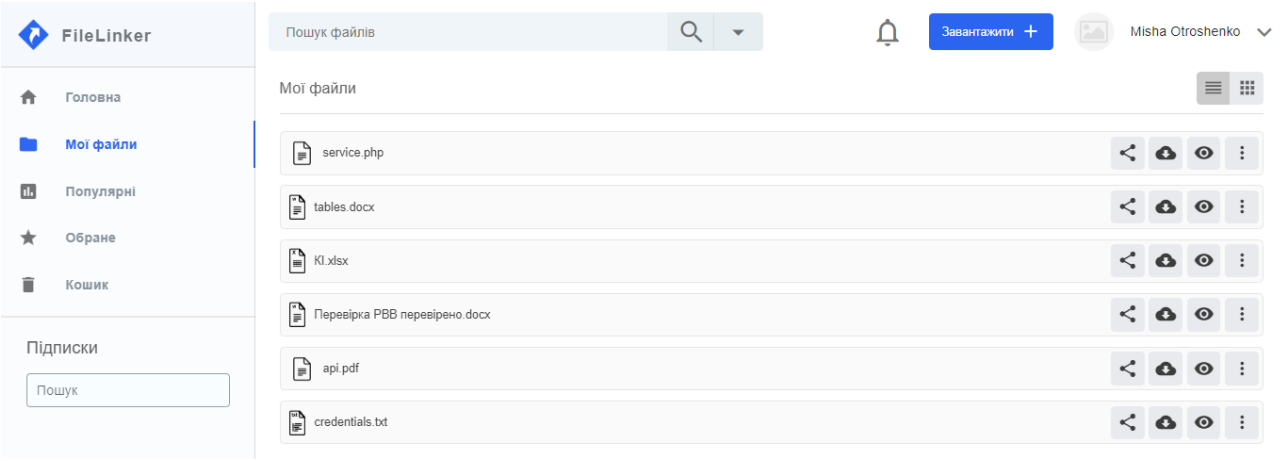

Рисунок 4.8 – Сторінка «Мої файли». Список з файлами

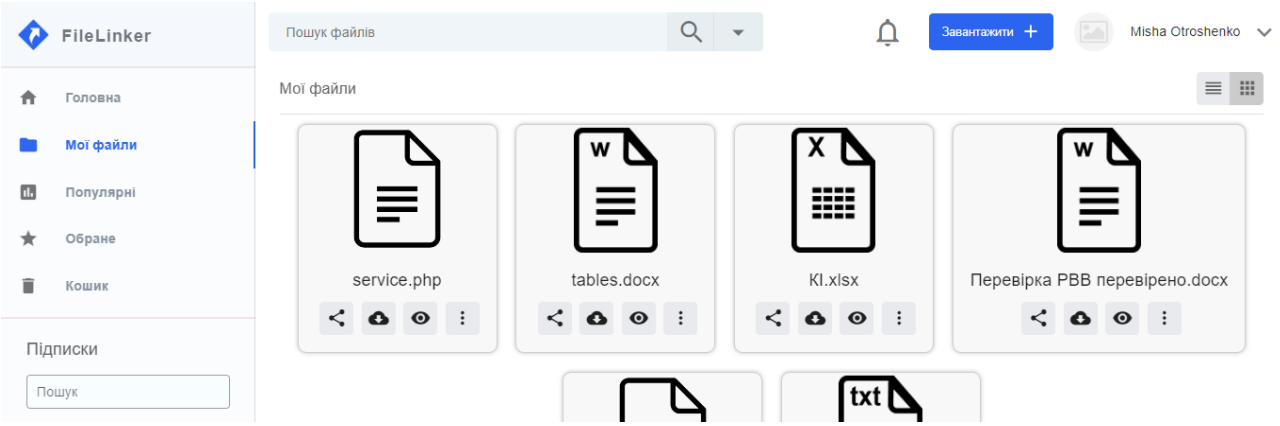

Рисунок 4.9 – Сторінка «Мої файли». Блоки з файлами

Web-сайт надає можливість завантажувати файл, копіювати посилання, переглядати додаткову інформацію.

Таблиця 4.1 – Опис функцій конок файлу.

Вид кнопки	Опис
	Копіювання посилання на файл.
	Завантаження файлу на комп’ютер.
	Перегляд повної інформації про файл.
	Видалити файл без можливості відновлення. . Доступна лише для власних файлів.
	Прибрати файл з кошику і зробити його доступним. Доступна лише для власних файлів.
	Кнопка з додатковим меню. Доступна лише для власних файлів. Дане меню складається з пунктів: • Видалити – перемістити файл до кошику, • Редагувати – змінення інформації про файл.

Редагування файлу відбувається в модальному вікні, в якому можна змінити назву файлу, опис та рівень поступу (рис. 4.10).

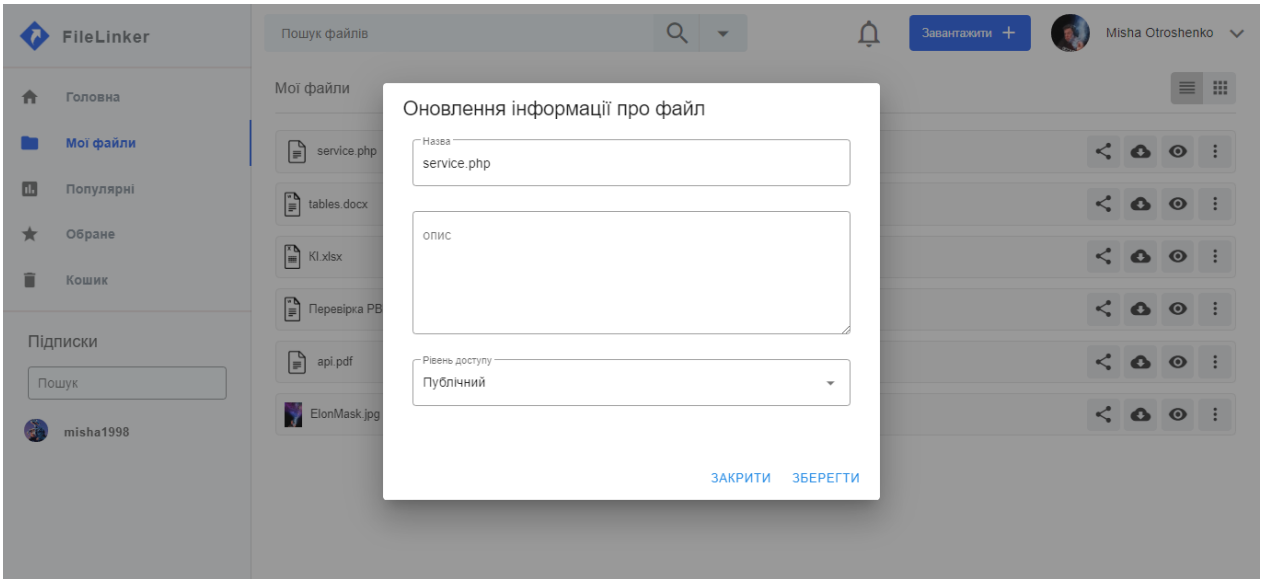

Рисунок 4.10 – Форма редагування файлу

Перегляд реалізований у вигляді додаткового правого блоку. Він складається з зображення типу файлу, назви, інформації про дату останньої зміцни, блоку «зірок» для можливості залишити бал фалу, що на далі вплине не його популярність. Також блок містить наступні кнопки:

- Відкрити – файл відкриється в новій вкладці,
- Завантажити – завантаження файлу на комп’ютер,
- Поділитись – скопіювати посилання на файл.

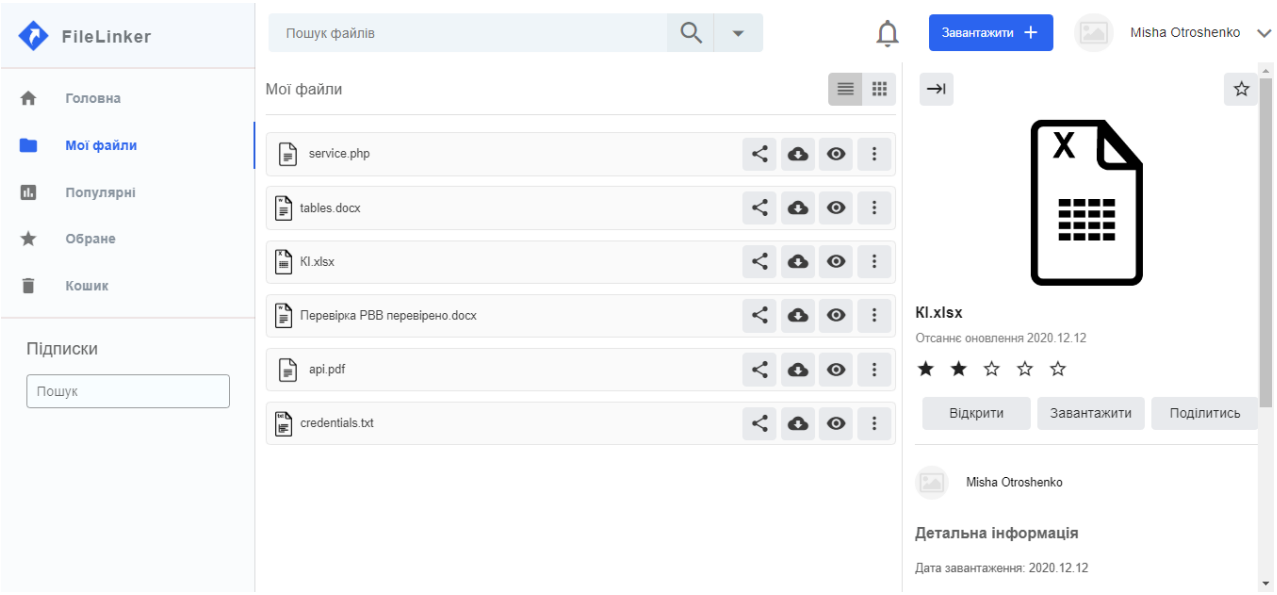

Рисунок 4.11 – Блок з інформацією про файл

Додатково блок містить інформацію про автора файлу, кількість переглядів та завантажень. Також є можливість залишати коментар (рис. 4.12).

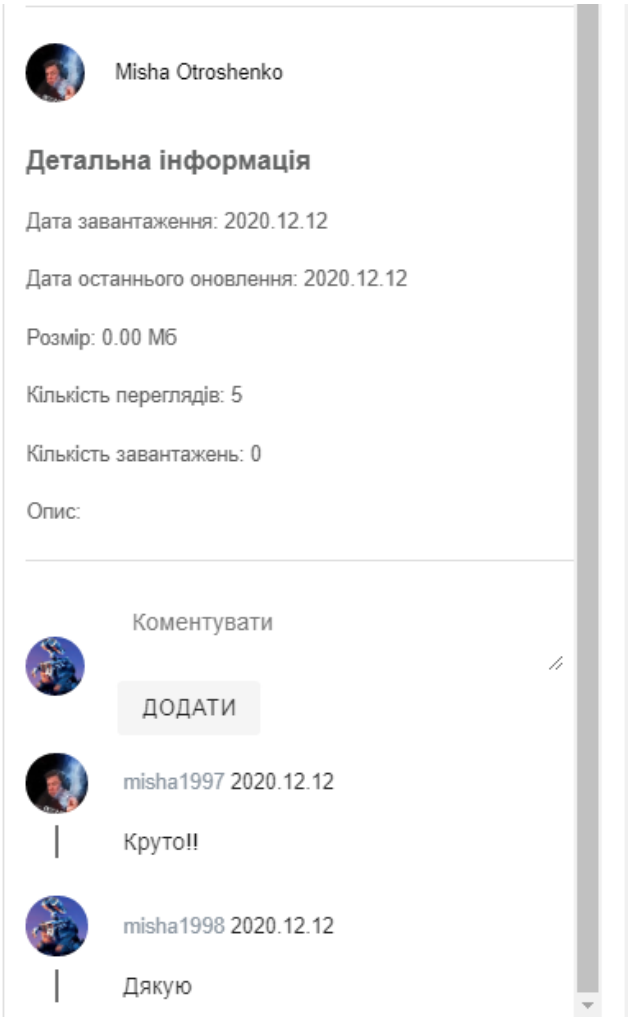

Рисунок 4.12 – Додаткова інформація про файл

На сайті також реалізований пошук файлів. Пошук можна виконати по назві файлу, типу та розміру. Кожен тим і максимальний розмір в формі пошуку змінюється динамічно в залежності від файлів завантаженими користувачами.

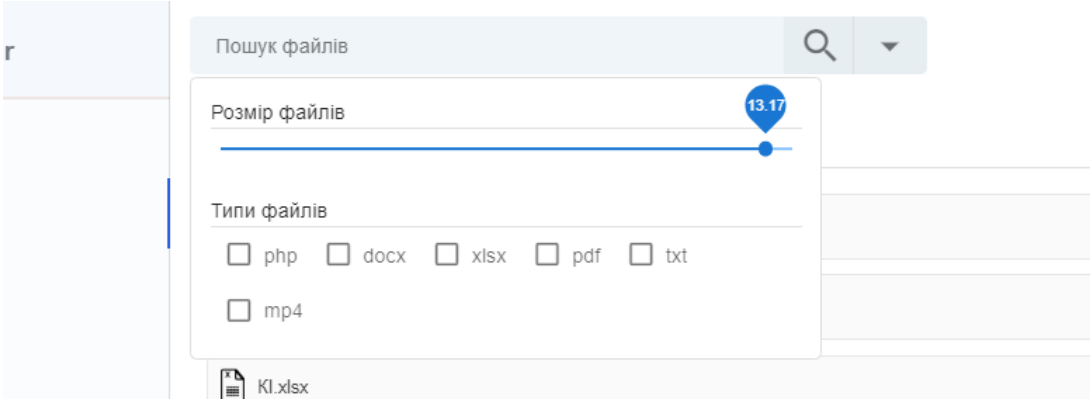

Рисунок 4.13 – Форма пошуку

За бажанням можна перейти на сторінку автора файлу і переглянути інформацію про нього, а також список файлів завантажених користувачем (4.14).

На сайті також реалізована система підписок на аккаунти користувачів. Кожен користувач має можливість підписатись на оновлення та публікації інших користувачів сайту.

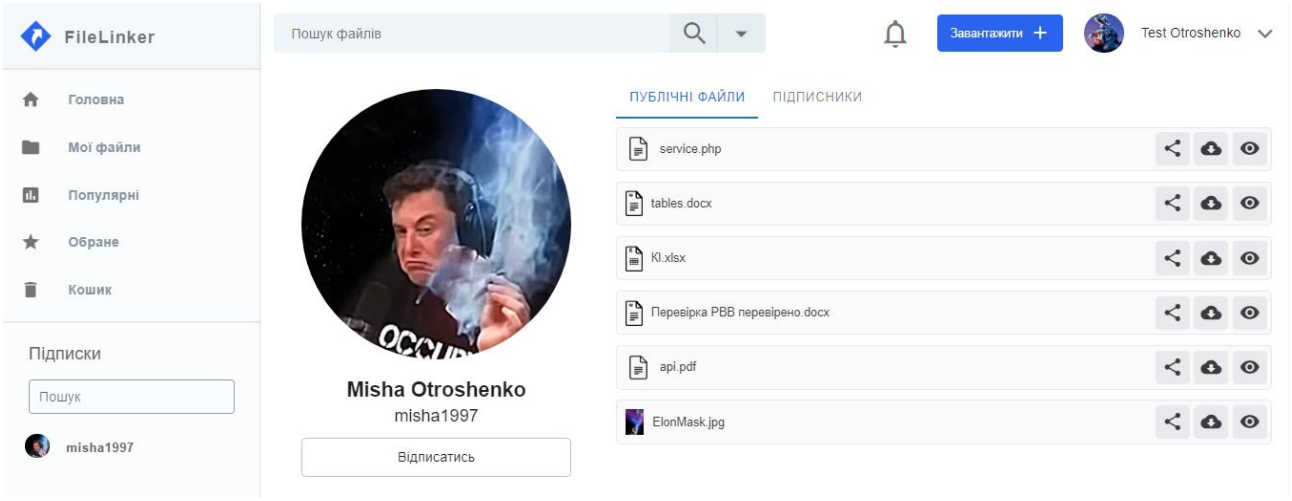

Рисунок 4.14 – Сторінка користувача

Всі оновлення користувачів на яких була виконана підписка будуть відображатись на головній сторінці сайту (рис. 4.15).

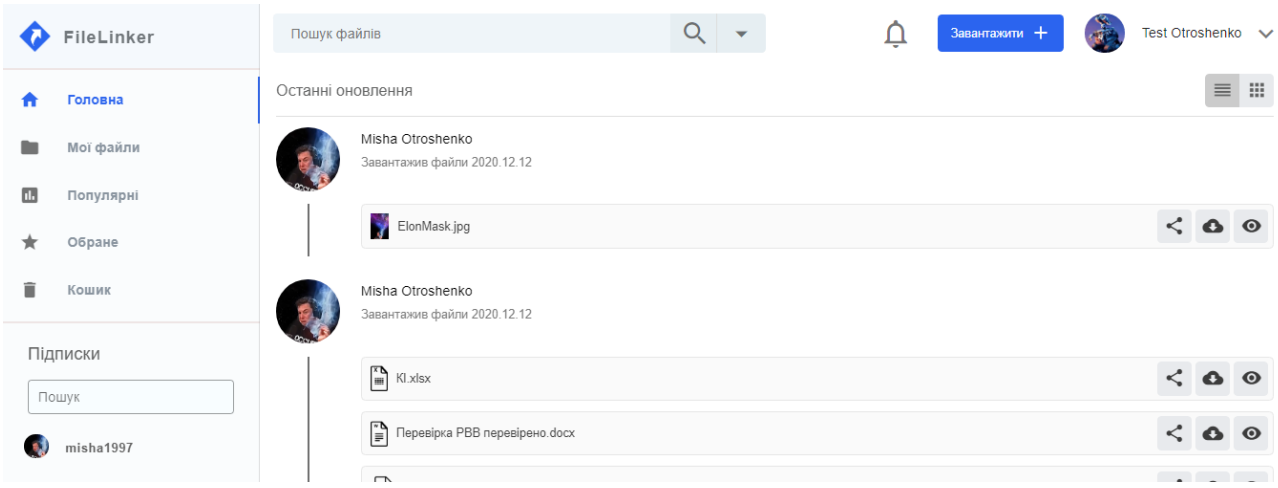

Рисунок 4.14 – Сторінка з останніми змінами

ВИСНОВКИ

При виконанні дипломної роботи та розробки інформаційної системи було проведено збір та аналіз актуальної інформації за даною тематикою. Результат даної роботи відіграв важливу роль при формуванні пріоритетів та функціоналу. А саме можливість завантаження обміну файлами між іншими користувачами сайту і не тільки, пошук файлів з можливістю вказувати додаткові параметри.

Було обрано функціонал для реалізації файлообміннику, візуальної та функціональної частини. Аналіз ринку та фреймворків допоміг обрати найбільш вдалі варіанти, а самеVue.js та Laravel.

При моделюванні інформаційної системи було створено відповідні діаграми та її декомпозиції. Розроблена діаграма варіантів використання повністю відображає функціонал створеного сайту. Дана інформація стала основою для прототипу додатку.

Проаналізувавши сутності, використовувані в моделі інформаційної системи, була виконана реалізація структура бази даних.

У результаті виконаної роботи було розроблено інформаційну систему для обміну файлами із реалізацією усіх можливостей користувача. Виконаний веб-додаток відповідає технічному завданню у повній мірі.

СПИСОК ЛІТЕРАТУРИ

1. Що таке хмарне сховище? – https://aws.amazon.com/ru/what-is-cloud-storage/.

2. Що таке файлообмінник і як ним користуватися? – https://drigin.com/article/chto-takoe-fajloobmennik.html.

3. Файлообмінники. Зберігання та обмін файлами? – http://www.introweb.ru/inews/soft/news9591.php.

4. Безпечний обмін даними через Інтернет? – http://www.taurion.ru/ie6/10/2.

5. Що таке файлообмінники і як ними користуватися – https://mypclife.ru/chto-takoe-fayloobmenniki-i-kak-imi-polzovatsya/.

6. 12 Best File Hosting Services (2020): Free Storage & Sharing? – https://www.hostingadvice.com/how-to/best-file-hosting-services/.

7. What is File Sharing Security? – https://digitalguardian.com/blog/what-file-sharing-security.

8. Що таке файлообмінник | Відповідь на питання – https://unotices.com/page-answer.php?id=34299.

9. Кращі безкоштовні файлообмінники без реєстрації 2019 – https://comhub.ru/luchshie-besplatnye-fajloobmenniki-bez-registratsii.

10. INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0). Draft Federal Information Processing Standards Publication 183 ,1993 December 21.

11. INTEGRATION DEFINITION FOR INFORMATION MODELING (IDEF1X), Draft Federal Information Processing Standards Publication 184 1993 December 21.

12. Sheer, A. ARIS-Business Process Modelling / Sheer A. – Springer-Verlag, Berlin,1998.

13. Booch, G. The Unified Modeling Language User Guide / Booch, G., Rumbaugh, J., Jacobson, I. – Second Edition, AddisonWesley, 2005.

14. Методичні вказівки до лабораторних робіт з курсу «Організація баз даних та баз знань». Укладач В.О. Нелюбов. – Ужгород: Видавничий центр ЗакДУ, 2010.

15. Система управління базами даних Access. Навчальний посібник «Організація баз даних і баз знань»/ Укладач В.О. Нелюбов. – Ужгород: Редакційно-видавничій відділ, 2015.

16. Круг Стів. Вебдизайн: Книга Стіва Кола або не змушуй мене думати [Текст] / Стів Круг. — Символ-Плюс, 2001.

17. Райс Ерік. Інформаційно архітектурний підхід до створення успішних веб-сайтів. [Текст] / Ерік Райс. — Addison Wesley, 2000.

18. Анді Гутманс, Стіг Баккен, Дерік Ретханс. Програмування живлення PHP5, 2015. — 704 с.

19. Документація Vue.js. https://ru.vuejs.org/v2/guide/.

20. Хеслоп П. HTML самого початку. С.-Пб: Санкт-Петербург, 2014. – 450с.

21. Що таке CSS. http://phpist.com.ua/css/5-whatcss.

22. Документація Laravel. https://laravel.com/.

ДОДАТОК А

2014_10_12_000000_create_users_table.php

Міграція створення таблиці аккаута.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('email')->unique();
            $table->string('password');
            $table->string('nickname')->unique();
            $table->string('name');
            $table->string('surname');
            $table->boolean('sex');
            $table->string('photo')->default('/img/no-image.png');
            $table->boolean('is_admin')->default(0);
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

2020_11_01_093618_create_files_table.php

Міграція створення таблиці файлів.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateFilesTable extends Migration
{
    public function up()
    {
        Schema::create('files', function (Blueprint $table) {
            $table->id();
```

```
            $table->string('title');
            $table->string('type');
            $table->integer('size');
            $table->string('patch');
            $table->text('description')->nullable();
            $table->foreignId('users_id')->unsigned();
            $table->foreignId('access_type_id');
            $table->boolean('basket')->default(0);
            $table->timestamps();
        });

        Schema::table('files', function (Blueprint $table) {
            $table->index('users_id');
            $table->foreign('users_id')->references('id')->on('users')->onDelete('cascade');
        });

        Schema::table('files', function (Blueprint $table) {
            $table->index('access_type_id');
            $table->foreign('access_type_id')->references('id')->on('access_type');
        });
    }
    public function down()
    {
        Schema::dropIfExists('files');
    }
}
```

2020_11_01_092557_create_subscribes_table.php

Міграція створення таблиці підписників.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateSubscribesTable extends Migration
{
    public function up()
    {
        Schema::create('subscribes', function (Blueprint $table) {
            $table->id();
            $table->foreignId('users_id');
            $table->foreignId('subscribe_users_id');
        });

        Schema::table('subscribes', function (Blueprint $table) {
            $table->index('users_id');
            $table->foreign('users_id')->references('id')->on('users')->onDelete('cascade');
            $table->index('subscribe_users_id');
            $table->foreign('subscribe_users_id')->references('id')->on('users')->onDelete('cascade');
```

```
        });
    }
    public function down()
    {
        Schema::dropIfExists('subscribes');
    }
}
```

User.php

Модель користувачів.

```
<?php

namespace App\Models;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravel\Passport\HasApiTokens;

class User extends Authenticatable
{
    use Notifiable, HasApiTokens;

    protected $table = 'users';

    protected $fillable = [
        'email',
        'password',
        'nickname',
        'name',
        'surname',
        'sex',
        'photo',
        'is_admin'
    ];

    protected $hidden = [
        'password',
    ];

    function feed() {
        return $this->hasMany('App\Models\Feed', 'users_id');
    }

    function files() {
        return $this->hasMany('App\Models\Files', 'users_id')->where('basket', 0);
    }

    function subscribes() {
        return $this->HasMany('App\Models\Subscribes', 'subscribe_users_id');
    }
```

```
    function mySubscribes() {
        return $this->HasMany('App\Models\Subscribes', 'users_id');
    }
}
```

Files.php

Модель файлів.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Files extends Model
{
    protected $table = 'files';

    protected $fillable = [
        'users_id',
        'title',
        'type',
        'size',
        'patch',
        'description',
        'access_type_id',
        'basket'
    ];

    protected $casts = [
        'created_at' => 'datetime:Y.m.d',
        'updated_at' => 'datetime:Y.m.d',
    ];

    function user() {
        return $this->belongsTo('App\Models\User', 'users_id');
    }

    function views() {
        return $this->hasMany('App\Models\Views', 'files_id');
    }

    function downloads() {
        return $this->hasMany('App\Models\Downloads', 'files_id');
    }

    function comments() {
        return $this->hasMany('App\Models\Comments', 'files_id');
    }
```

```
    function stars() {
        return $this->hasMany('App\Models\Stars', 'files_id');
    }

    function favorites() {
        return $this->hasMany('App\Models\Favorites', 'files_id');
    }
}
```

AuthController.php

Контроллер авторизації.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;

use App\Models\User;

class AuthController extends Controller
{
    // register
    function register(Request $request) {
        $request->validate([
            'email' => 'required|string|email|unique:users',
            'password' => 'required|string'
        ]);
        $user = new User();
        $data = $request->all();
        $data["password"] = Hash::make($request->password);
        $user->create($data);
        $credentials = request(['email', 'password']);
        if(!Auth::attempt($credentials)) {
            return response()->json(['message' => 'Unauthorized'], 401);
        }
        $authUser = Auth::user();
        $tokenResult = $authUser->createToken('Personal Access Token');
        $token = $tokenResult->token;
        $token->save();
        return response()->json([
            'access_token' => 'Bearer '.$tokenResult->accessToken,
            'user' => $authUser
        ]);
    }

    // login
    function login(Request $request) {
```

```
        if(Auth::attempt(['email' => $request->email, 'password' => $request->password, 'user_role_id'
=> 1]) || Auth::attempt(['email' => $request->email, 'password' => $request->password, 'user_role_id'
=> 2]) || Auth::attempt(['email' => $request->email, 'password' => $request->password, 'user_role_id'
=> 3])) {
            $user = Auth::user();
            $tokenResult = $user->createToken('Personal Access Token');
            $token = $tokenResult->token;
            if($request->remember_me) {
               $token->expires_at = Carbon::now()->addWeeks(1);
            }
            $token->save();
            return response()->json([
               'access_token' => 'Bearer '.$tokenResult->accessToken,
               'user' => $user
            ]);
        } else {
            return response()->json(['message' => 'Unauthorized'], 401);
        }
    }
    // loginAdmin
    function loginAdmin(Request $request) {
        if(Auth::attempt(['email' => $request->email, 'password' => $request->password,
'account_role_id' => 3])) {
            $user = $request->user();
            $tokenResult = $user->createToken('Personal Access Token');
            $token = $tokenResult->token;
            $token->save();
            return response()->json([
               'access_token' => 'Bearer '.$tokenResult->accessToken
            ]);
        } else {
            return response()->json(['message' => 'Unauthorized'], 401);
        }
    }
}
```

UserController.php

Контроллер користувачів.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Carbon\Carbon;

use App\Models\User;
use App\Models\UserHasServices;
use App\Models\UserHasServiceItems;
use App\Models\UserPhoto;
```

```
use App\Models\Reviews;
use App\Models\UserCars;

class UserController extends Controller
{
    protected $fileStorage = "userfiles/";

    //getPopularServices
    function getPopularServices() {
        $data = User::with(
            'services.service',
            'serviceItems.item',
            'photos'
        )->where("user_role_id", 2)->orWhere("user_role_id", 3)->limit(6)->get();
        return response()->json($data);
    }

    //postUserCar
    function postUserCar(Request $request) {
        $model = new UserCars();
        $data = $request->all();
        $data['user_id'] = Auth::id();
        $model->create($data);
        return response('ok', 200);
    }

    //delUserCar
    function delUserCar($id) {
        UserCars::find($id)->delete();
        return response('ok', 200);
    }

    //postReview
    function postReview(Request $request, $id) {
        $model = new Reviews();
        $model->create([
            "user_id" => $id,
            "client_id" => Auth::id(),
            "comment" => $request->comment
        ]);
        return response('ok', 200);
    }

    //delReview
    function delReview($id) {
        Reviews::find($id)->delete();
        return response('ok', 200);
    }

    //getServices
    function getServices(Request $request) {
```

```
        $data = User::with('services.service', 'serviceItems.item', 'photos')->where("user_role_id", 2)->orWhere("user_role_id", 3)->get();
        return response()->json($data);
    }
    //getServiceId
    function getServiceId($id) {
        $data = User::with('services.service', 'serviceItems.item', 'photos', 'reviews.user')->find($id);
        foreach ($data['reviews'] as $key => $value) {
            $value['date'] = Carbon::parse($value['created_at'])->format('d.m.Y');
        }
        return response()->json($data);
    }

    //profile
    function profile() {
        $data = User::with(
            'services.service',
            'serviceItems.item',
            'photos',
            'orders.user',
            'reviews.user',
            'cars'
        )->find(Auth::id());
        foreach ($data['orders'] as $key => $value) {
            $value['date'] = Carbon::parse($value['created_at'])->format('d.m.Y');
            $value['car'] = json_decode($value['car']);
            $value['services'] = json_decode($value['services']);
        }
        return response()->json($data);
    }

    // updateUser
    function updateProfile(Request $request) {
        $id = Auth::id();
        $data = $request->all();
        if(isset($data['newPassword'])) {
            $user = Auth::user();
            if (Hash::check($data['oldPassword'], $user->password)) {
                $data["password"] = Hash::make($request->newPassword);
            } else {
                return response('error', 401);
            }
        }
        if($request->services) {
            UserHasServices::where("user_id", $id)->delete();
            UserHasServiceItems::where("user_id", $id)->delete();
            foreach ($request->services as $key => $value) {
                $model = new UserHasServices();
                $model->create([
                    "user_id" => $id,
                    "service_id" => $value['id']
```

```
                ]);
                foreach ($value['items'] as $k => $v) {
                    if($v['selected'] == 1) {
                        $modelItem = new UserHasServiceItems();
                        $modelItem->create([
                            "user_id" => $id,
                            "service_item_id" => $v['id'],
                            "price" => isset($v['price']) ? $v['price'] : null
                        ]);
                    }
                }
            }
        }
        User::find($id)->update($data);
        return response()->json(User::with('services.service', 'serviceItems.item')->find($id));
    }

    // postProfilePhoto
    function postProfilePhoto(Request $request) {
        $uploadedFiles = $request->pics;
        foreach ($uploadedFiles as $file) {
            $model = new UserPhoto();
            $puth = $this->fileStorage.Auth::id();
            $name = 'http://'.$_SERVER['HTTP_HOST'].'/'.$puth."/".uniqid().'.'.$file-
>getClientOriginalExtension();
            $file->move(public_path().'/'.$puth, $name);
            $model->create([
                "user_id" => Auth::id(),
                "src" => $name
            ]);
        }
        return response('ok', 200);
    }
    // delProfilePhoto
    function delProfilePhoto($id) {
        UserPhoto::find($id)->delete();
        return response('ok', 200);
    }
    // img Account
    function img(Request $request) {
        if(isset($request['photo'])) {
            $arr = [];
            if($request['photo']) {
                $file = uniqid().'_photo_min.png';
                $uploadfile = $this->fileStorage . $request['id'] . '/' . $file;

                $img = str_replace('', '', $request['photo']);
                $img = str_replace(' ', '+', $img);
                $fileData = base64_decode($img);
                file_put_contents(public_path().'/'.$uploadfile, $fileData);
```

```
            $arr['status'] = 'success';
            $arr['path_mini'] = 'http://'.$_SERVER['HTTP_HOST'].'/'.$uploadfile;
            $arr['file_mini'] = $file;
        }
    }
    else {
        if(!file_exists("userfiles/".$request['id'])) {
            mkdir($this->fileStorage.$request['id']);
        }
        $uploadfile = $this->fileStorage. $request['id'] . '/' . uniqid().'_photo_original.png';
        $arr = array();
     if (move_uploaded_file($_FILES['file']['tmp_name'], public_path().'/'.$uploadfile)) {
            $arr['status'] = 'success';
            $arr['path_max'] = 'http://'.$_SERVER['HTTP_HOST'].'/'.$uploadfile;
            $arr['file_max'] = $_FILES['file']['name'];
        } else {
          $arr['status'] = 'fail';
        }
    }
    header('Content-type: application/json');
    return response()->json($arr);
  }
}
```

FilesController.php

Контроллер файлів.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

use App\Models\Files;
use App\Models\Views;
use App\Models\Downloads;
use App\Models\Comments;
use App\Models\Stars;
use App\Models\Feed;
use App\Models\UserFeed;
use App\Models\Favorites;
use App\Models\AccessType;

class FileController extends Controller
{
  protected $fileStorage = "/userFiles/";

  //searchFiles
  function popular() {
    $data = Files::with(
```

```
            'user',
            'views',
            'downloads',
            'comments.user',
            'stars',
            'favorites'
        )
        ->where('basket', 0)->limit(50)->get();

        return response()->json($data);
    }

    //searchFiles
    function searchFiles(Request $request) {
        $model = Files::with(
            'user',
            'views',
            'downloads',
            'comments.user',
            'stars',
            'favorites'
        )
        ->where('basket', 0);

        if($request->title != "") {
            $model->where('title', 'like', '%'.$request->title.'%');
        }

        if($request->types) {
            $model->whereIn('type', $request->types);
        }

        if($request->size != 0) {
            $model->where('size', '<=', $request->size);
        }

        $data = $model->get();

        return response()->json($data);
    }

    //deleteFile
    function deleteFile($id) {
        $model = Files::find($id)->delete();
        return response('ok', 200);
    }

    //updateFile
    function updateFile(Request $request, $id) {
        $model = Files::find($id);
        $data = $request->all();
```

```
    $model->update($data);
    return response('ok', 200);
}

// upload
function upload(Request $request) {
    $model = new Files();
    $modelUserFeed = new UserFeed();

    $puth = $this->fileStorage . Auth::id() . '/';
    $name = $puth . uniqid().'.'.$request['file']->getClientOriginalExtension();

    $response = $model->create([
        "title" => $request['file']->getClientOriginalName(),
        "type" => $request['file']->getClientOriginalExtension(),
        "size" => $request['file']->getSize(),
        "patch" => $name,
        "description" => $request->description,
        "users_id" => Auth::id(),
        "access_type_id" => 1
    ]);

    $modelUserFeed->create([
        "files_id" => $response['id'],
        "feed_id" => $request->idFeed
    ]);

    $request['file']->move(public_path().$puth, $name);
    return response('ok', 200);
}

//postFeed
function postFeed(Request $request) {
    $model = new Feed();
    $response = $model->create([
        "users_id" => Auth::id(),
        "text" => $request->text
    ]);
    return response()->json($response['id']);
}

//userFiles
function userFiles() {
    $data = Files::with(
        'user',
        'views',
        'downloads',
        'comments.user',
        'stars',
        'favorites'
    )
```

```
    ->where('users_id', Auth::id())
    ->where('basket', 0)
    ->get();
    return response()->json($data);
}

//userFilesBasket
function userFilesBasket() {
    $data = Files::with(
        'user',
        'views',
        'downloads',
        'comments.user',
        'stars',
        'favorites'
    )
    ->where('users_id', Auth::id())
    ->where('basket', 1)
    ->get();
    return response()->json($data);
}

//userFileId
function userFileId($id) {
    $data = Files::with(
        'user',
        'views',
        'downloads',
        'comments.user',
        'stars',
        'favorites'
    )->find($id);
    return response()->json($data);
}

//watchFile
function watchFile($id) {
    $model = new Views();
    $response = $model->create([
        "files_id" => $id,
        "users_id" => Auth::id()
    ]);
    return response()->json($response);
}

//downloadFile
function downloadFile($id) {
    $model = new Downloads();
    $response = $model->create([
        "files_id" => $id,
        "users_id" => Auth::id()
```

```
    ]);
    return response()->json($response);
}

//commentFile
function commentFile(Request $request, $id) {
    $model = new Comments();
    $model->create([
        "files_id" => $id,
        "users_id" => Auth::id(),
        "text" => $request->comment
    ]);
    return response('ok', 200);
}

//starFile
function starFile(Request $request, $id) {
    $model = new Comments();
    $response = $model->create([
        "files_id" => $id,
        "users_id" => Auth::id(),
        "text" => $request->rate
    ]);
    return response()->json($response);
}

// delFavorites
function delFavorites($id) {
    Favorites::where("users_id", Auth::id())->where('files_id', $id)->delete();
    return response('ok', 200);
}

// getFavorites
function getFavorites() {
    $data = Favorites::with(
        'files.user',
        'files.views',
        'files.downloads',
        'files.comments.user',
        'files.stars',
        'files.favorites'
    )->where("users_id", Auth::id())->get();
    return response()->json($data);
}

// postFavorites
function postFavorites(Request $request, $id) {
    $model = new Favorites();
    $model->create([
        "users_id" => Auth::id(),
        "files_id" => $id
```

```
        ]);
        return response('ok', 200);
    }

    // getAccessType
    function getAccessType() {
        $data = AccessType::get();
        return response()->json($data);
    }

    // getFileTypes
    function getFileTypes() {
        $types = Files::select('type')->distinct()->get();
        $maxSize = Files::max('size');

        return response()->json([
            "types" => $types,
            "maxSize" => $maxSize
        ]);
    }
}
```

ДОДАТОК Б

Index.vue

Компонент головної сторінки сайту.

```
<template>
  <v-container>
    <TitlePage :loading="loading" title="Останні оновлення"></TitlePage>

    <div class="font-weight-regular text-center pt-5" v-if="!loading && data.length == 0">
      Нічого не знайдено
    </div>

    <div v-for="(item, index) in data" :key="index">
      <v-row>
        <v-col cols="1" class="avatar pb-2">
          <img :src="item.user.photo">
        </v-col>
        <v-col class="nickname pb-2">
          <div class="name"><router-link :to="'/user/'+item.user.id">{{ item.user.name }} {{
item.user.surname }}</router-link></div>
          <div class="time">{{item.user.sex ? 'Завантажив' : 'Завантажила'}} файли {{
item.created_at }}</div>
        </v-col>
      </v-row>

      <v-row>
        <v-col cols="1" class="vertical-line pt-0">
          <div class="line"></div>
        </v-col>
        <v-col class="pt-0" v-if="typeView == 'line'">
          <FileLineItem v-for="(file, index) in item.files" :item="file.file"
:key="index"></FileLineItem>
        </v-col>
        <div style="display: flex" v-if="typeView == 'block'">
          <v-col style="padding: 11px" cols="auto" v-for="(file, index) in item.files"
:key="index">
            <FileItem :item="file.file"></FileItem>
          </v-col>
        </div>
      </v-row>
    </div>
  </v-container>
</template>

<script>
  import { mapGetters } from 'vuex';

  import FileLineItem from "../../components/site/FileLineItem";
  import FileItem from "../../components/site/FileItem";
```

```
    import TitlePage from "../../components/site/TitlePage";
    export default {
        components: {
            FileLineItem,
            FileItem,
            TitlePage
        },
        data() {
            return {
                loading: true,
                data: []
            }
        },
        created() {
            this.fetchData();
        },
        computed: {
            typeView() {
                return this.getTypeView();
            }
        },
        methods: {
            ...mapGetters([
                "getTypeView"
            ]),
            fetchData() {
                axios.get('/api/feed')
                .then((response) => {
                    this.loading = false;
                    this.data = response.data;
                })
                .catch((err) => {
                    this.loading = false;
                })
            }
        }
    }
</script>

<style lang="css" scoped>
    .nickname {
        font-weight: 400;
        line-height: 28px;
    }
    .nickname .name {
        font-size: 16px;
    }
    .nickname .name a {
        color: #000;
        text-decoration: none;
    }
```

```
  .avatar img {
    width: 100%;
    border-radius: 50%;
  }
  .vertical-line .line {
    width: 2px;
    background: #6F6F6F;
    height: 95%;
    margin: 0 auto;
  }
</style>
```

Header.vue

Компонент форми пошуку та завантаження файлів.

```
<template>
  <v-app-bar app clipped-right flat height="70" color="white">

    <v-dialog  v-model="formFiles" persistent max-width="600px">
      <v-card>
        <v-card-title>
          <span class="headline">Завантаження файлів</span>
        </v-card-title>
        <v-card-text>
          <v-container>
            <label class="input-block" v-if="files.length == 0">
              <input
                style="display: none"
                type="file"
                multiple
                id="file"
                ref="file"
                v-on:change="handleFileUpload()"
              >
            </label>
            <div v-else v-for="(item, index) in files" :key="index" class="file-item mt-4">
              <div class="name">Файл: {{ index }} - {{ ((item.size/1024)/1024).toFixed(2) + '
МБ' }}</div>
              <input type="text" v-model="item.title"><br>
              <textarea v-model="item.description"></textarea><br>
              <progress max="100" :value.prop="item.uploadPercentage"></progress>
            </div>
          </v-container>
        </v-card-text>
        <v-card-actions>
          <v-spacer></v-spacer>
          <v-btn
            color="blue darken-1"
            text
            @click="formFiles = false; files = []"
```

```
            >
              Закрити
            </v-btn>
            <v-btn
              color="blue darken-1"
              text
              @click="submitFile()"
            >
              Готово
            </v-btn>
          </v-card-actions>
        </v-card>
      </v-dialog>

      <div class="search py-2">
        <input type="text" class="search-form" @input="searchInput" v-model="filter.title" placeholder="Пошук файлів">
        <button class="search-button"><v-icon text large @click="search">search</v-icon></button>
        <button class="options-button"><v-icon text large @click="formFilter = !formFilter">arrow_drop_down</v-icon></button>
        <div class="searchResult" v-if="searchFiles.length > 0">
          <div class="resultItem" v-for="(item, index) in searchFiles" :key="index" @click="setFile(item)">
            {{ item.title }}
          </div>
        </div>
        <div class="formSearch" v-if="formFilter">
          Розмір файлів
          <v-divider></v-divider>
          <v-slider
            :max="maxSize"
            :min="0"
            v-model="filter.size"
            thumb-label="always"
          >
          <template v-slot:thumb-label="{ value }">
            {{ (value / 1024 / 1024).toFixed(2) }}
          </template>
          </v-slider>
          Типи файлів
          <v-divider></v-divider>
          <div class="typeBlock" v-for="(item, index) in typeFiles" :key="index">
            <v-checkbox
              @change="searchInput"
              :label="item.type"
              :value="item.type"
              hide-details
              v-model="filter.types"
            ></v-checkbox>
          </div>
```

```
            </div>
        </div>
        <v-spacer></v-spacer>
        <v-btn icon>
            <v-icon x-large>notifications_none</v-icon>
        </v-btn>
        <button @click="formFiles = true" class="download">Завантажити <span class="material-
icons ml-2">add</span></button>
        <img class="avatar py-2" :src="authUser.photo">
        <span>{{authUser.name}} {{authUser.surname}}</span>

        <v-menu offset-y>
            <template v-slot:activator="{ on, attrs }">
                <v-btn icon class="ml-1" v-bind="attrs" v-on="on">
                    <v-icon large>keyboard_arrow_down</v-icon>
                </v-btn>
            </template>
            <v-list>
                <v-list-item link to="/profile">
                    <v-list-item-title>Профіль</v-list-item-title>
                </v-list-item>
                <v-list-item link>
                    <v-list-item-title @click="logout">Вихід</v-list-item-title>
                </v-list-item>
            </v-list>
        </v-menu>
    </v-app-bar>
</template>
<script>
import { mapMutations } from 'vuex';
export default {
    data() {
        return {
            formFiles: false,
            formFilter: false,
            file: null,
            typeFiles: [],
            maxSize: 0,
            files: [],
            searchFiles: [],
            filter: {
                title: "",
                types: [],
                size: 0,
            }
        }
    },
    computed: {
        authUser() {
            return this.$store.getters.authUser
        }
```

```
},
created() {
    this.getTypeFiles();
},
methods: {
    ...mapMutations([
        "setFile"
    ]),
    searchInput() {
        if(this.filter.title.length > 3) {
            axios.get('/api/search', {
                params: this.filter
            }).then(response => {
                this.searchFiles = response.data;
            })
        } else {
            this.searchFiles = [];
        }
    },
    search() {
        if (this.$route.name !== 'search') {
            this.$router.push({ name: 'search', query: this.filter})
        }
    },
    getTypeFiles() {
        axios.get('/api/file-types').then(response => {
            console.log(response.data)
            this.typeFiles = response.data.types;
            this.filter.size = +response.data.maxSize;
            this.maxSize = +response.data.maxSize;
        })
    },
    handleFileUpload() {
        let selectFiles = this.$refs.file.files;
        for(let i = 0; i < selectFiles.length; i++) {
            this.files.push({
                title: selectFiles[i].name,
                description: "",
                file: selectFiles[i],
                uploadPercentage: 0,
                size: selectFiles[i].size
            })
        }
    },
    submitFile() {
        axios.post('/api/feed', {
            text: "додав файли"
        })
        .then((response) => {
            let formData = new FormData();
            for (let index = 0; index < this.files.length; index++) {
```

```
                this.uploadFile(this.files[index], response.data)
            }
        })
        .then(() => {
            // this.formFiles = false;
            swal.fire({
                icon: 'success',
                title: 'Файли завантажено'
            });
        })
    },
uploadFile(fileItem, idFeed) {
    let formData = new FormData();
    formData.append('file', fileItem.file);
    formData.append('title', fileItem.title);
    formData.append('description', fileItem.description);
    formData.append('idFeed', idFeed);
    axios.post('/api/file-progress', formData, {
        headers: {
            'Content-Type': 'multipart/form-data'
        },
        onUploadProgress: function(progressEvent) {
            fileItem.uploadPercentage = parseInt(Math.round((progressEvent.loaded / progressEvent.total) * 100));
        }.bind(this)
    }
    )
},
    logout() {
        this.$store.dispatch('logout').then(() => {
            this.$router.push('/')
        })
    }
}
}
</script>
```

app.js

Скрипт підключення головних модулів сайту.

```
require('./bootstrap');

import Vue from 'vue'
import vuetify from './plugins/vuetify'
import router from "./routes"
import store from "./store"

import AppComponent from './views/site/App';

const app = new Vue({
```

```
  el: '#app',
  components: {
    AppComponent
  },
  vuetify,
  store,
  router
});
```

store.js

Скрипт відправлення та отримання запитів від серверу на клієнтську частину.

```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    status: '',
    token: localStorage.getItem('token') || '',
    tokenAdmin: localStorage.getItem('tokenAdmin') || '',
    user: JSON.parse(localStorage.getItem('user')) || null
  },
  mutations: {
    auth_request(state) {
      state.status = 'loading'
    },
    auth_user_success(state, token, user_data) {
      state.status = 'success'
      state.token = token
      state.user = user_data
    },
    auth_admin_success(state, token) {
      state.status = 'success'
      state.tokenAdmin = token
    },
    auth_error(state) {
      state.status = 'error'
    },
    logout(state) {
      state.status = ''
      state.token = ''
      state.user = ''
    },
    user_data(state, user){
      state.user = user
    },
  },
  actions: {
    login({commit}, user) {
      return new Promise((resolve, reject) => {
```

```
        commit('auth_request')
        axios({url: '/api/login', data: user, method: 'POST' })
        .then(resp => {
            const token = resp.data.access_token
            const user_data = resp.data.user
            localStorage.setItem('user', JSON.stringify(user_data))
            localStorage.setItem('token', token)
            axios.defaults.headers.common['Authorization'] = token
            commit('auth_user_success', token, user_data)
            resolve(resp)
        })
        .catch(err => {
            commit('auth_error')
            localStorage.removeItem('token')
            localStorage.removeItem('user')
            reject(err)
        })
    })
},
loginAdmin({commit}, user) {
    return new Promise((resolve, reject) => {
        axios({url: '/api/login-admin', data: user, method: 'POST' })
        .then(resp => {
            const token = resp.data.access_token
            localStorage.setItem('tokenAdmin', token)
            axios.defaults.headers.common['Authorization'] = token
            commit('auth_admin_success', token)
            resolve(resp)
        })
        .catch(err => {
            commit('auth_error')
            localStorage.removeItem('tokenAdmin')
            reject(err)
        })
    })
},
register({commit}, user) {
    return new Promise((resolve, reject) => {
        commit('auth_request')
        axios({url: '/api/register', data: user, method: 'POST' })
        .then(resp => {
            const token = resp.data.access_token
            const user_data = resp.data.user
            localStorage.setItem('user', JSON.stringify(user_data))
            localStorage.setItem('token', token)
            axios.defaults.headers.common['Authorization'] = token
            commit('auth_success', token, user)
            resolve(resp)
        })
        .catch(err => {
            commit('auth_error', err)
```

```
                localStorage.removeItem('token')
                localStorage.removeItem('user')
                reject(err)
            })
        })
    },
    logout({commit}) {
        return new Promise((resolve, reject) => {
            commit('logout')
            localStorage.removeItem('token')
            localStorage.removeItem('user')
            localStorage.removeItem('tokenAdmin')
            delete axios.defaults.headers.common['Authorization']
            resolve()
        })
    },
    user({commit}, user) {
        localStorage.setItem('user', JSON.stringify(user))
        commit('user_data', user)
    }
},
getters: {
    isLoggedIn: state => !!state.token,
    isLoggedInAdmin: state => !!state.tokenAdmin,
    authStatus: state => state.status,
    authUser: state => state.user,
}
})
```