

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ЗБЕРІГАННЯ ФАЙЛІВ
КОРИСТУВАЧІВ
WEB APPLICATION DEVELOPMENT
FOR USER FILE STORAGE

Виконав: студент 4 курсу, групи КНТ-117

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Нескоромний І.М.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 20__ року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ),

Нескоромного Івана Михайловича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для зберігання файлів користувачів. Web Application Development for User File Storage

керівник проєкту (роботи) Федорончак Т. В., к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 17 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Реферат. Перелік скорочень та умовних позначок. Вступ. 1. Аналіз предметної області. 2. Опис програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми. Додатки.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Федорончак Т.В., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “ 20 ” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	1 тиждень	
2	Розробка та затвердження технічного завдання	2 тиждень	Технічне завдання
3	Проектування структури системи	3-4 тижні	Структурна схема
4	Розробка схеми функціонування системи та модулів програми	5-6 тижні	Функціональна схема. Модулі
5	Створення програмного коду системи	7-8 тижні	Текст програми
6	Тестування та налагодження програми	9 тиждень	Працездатна система
7	Розробка розділів пояснювальної записки	10 тиждень	ПЗ
8	Розробка слайдів презентації	11-12 тижні	Слайди презентації
9	Захист роботи	13 тиждень	

Студентка

Керівник проєкту (роботи)

(підпис)

(підпис)

Нескоромний І. М.

(прізвище та ініціали)

Федорончак Т.В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
102 с., 25 рис., 1 табл., 3 дод., 15 джерел.

ВЕБ-СХОВИЩЕ, ОРГАНІЗАЦІЯ ЗБЕРІГАННЯ ДАНИХ,
ЗАВАНТАЖЕННЯ ФАЙЛІВ, JAVASCRIPT, NODE JS, REACT JS, MONGODB.

Об'єкт дослідження – системи для зберігання файлів користувачів.

Предмет дослідження – вебсайт, що надає користувачам можливість зберігати файли.

Мета роботи – програмна реалізація сховища файлів користувачів у веб-просторі.

Матеріали, методи та технічні засоби: основні інструменти для веб-розробки HTML/CSS, мова програмування JavaScript, технологія створення односторінкового застосунку React, бібліотека для управління станом Redux, платформа NodeJS, нереляційна база даних MongoDB та інтегроване середовище розробки Visual Studio Code.

Результати. Створено програмний вебзастосунок для віддаленого зберігання файлів.

Висновки. Проведено аналіз та порівняння з існуючим подібними застосунками. Розроблено програмний продукт, який призначений для зберігання файлів користувачів.

Галузь використання - проект може буде використано для малого та великого бізнесу, підприємств, які хочуть мати свою власну платформу для збереження конфіденційної інформації.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 102 pages, 25 figures, 1 tables, 3 appendixes, 15 sources.

WEB STORAGE, DATA STORAGE ORGANIZATION, FILE DOWNLOAD, JAVASCRIPT, NODE JS, REACT JS, MONGODB.

The object of study is systems for storing user files.

The subject of study is the development of a web application for storing user files.

The purpose of the work - software implementation of a web application for storing user files.

Materials, methods and technical tools: basic tools for HTML / CSS web development, JavaScript programming language, React one-page application development technology, Redux state management library, NodeJS platform, non-relational MongoDB database and Visual Studio Code integrated development environment.

Results. A software application for remote file storage has been created.

Conclusions. An analysis and comparison with existing similar applications. Developed a software product that is designed to store user files.

Scope - the project can be used for small and large businesses, businesses that want to have their own platform to keep confidential information.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	10
1.1 Основні поняття хмарних технологій.....	10
1.1.1 Види розгортання хмарних сховищ.....	11
1.1.2 Переваги використання.....	12
1.2 Сервіси зберігання даних.....	12
1.3 Огляд існуючих рішень проблеми	15
1.4 Висновок до розділу	20
2 Опис програми	21
2.1 Основні засоби веброзробки.....	21
2.1.1 Мова гіпертекстової розмітки HTML	21
2.1.2 Каскадна таблиця стилів – CSS.....	22
2.1.3 Мова програмування JavaScript	24
2.2 Вибір клієнтського середовища проектування.....	25
2.2.1 Управління станом	28
2.2.2 Virtual DOM React	29
2.3 Вибір серверного середовища проектування	30
2.4 Вибір бази даних.....	31
2.5 Вибір допоміжних інструментів	32
2.6 Архітектура вебзастосунку.....	32
2.7 Вимоги до програмного забезпечення, що розробляється.....	33
2.8 Висновки до розділу.....	34

3 Розробка програми.....	36
3.1 База даних.....	36
3.2 Алгоритм функціонування вебсайту	37
3.3 Структурна схема проекту.....	38
3.3.1 Структурна схема клієнтської частини	38
3.3.2 Структурна схема серверної частини	40
3.4 Опис реалізованих функцій	41
3.4.1 Функції клієнтської частини (front-end).....	42
3.4.2 Функції серверної частини (back-end).....	43
3.5 Висновки до розділу.....	44
4 Експлуатація, тестування та експериментальне дослідження програми.....	45
4.1 Виконання програми	45
4.2 Повідомлення користувачу.....	50
4.3 Висновок до розділу	51
Висновок.....	52
Перелік джерел посилання	53
Додаток А Технічне завдання.....	55
Додаток Б Текст програми.....	59
Додаток В Слайди презентації	95

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

IDE – Integrated Development Environment;

БД – база даних;

ПЗ – програмне забезпечення;

ХС– хмарні сервіси.

ВСТУП

Зовнішні системи зберігання даних за свою історію сильно змінилися. Гнучкі магнітні диски, DVD-диски, зовнішні жорсткі диски, USB-накопичувачі - всі ці технології швидко вдосконалювалися і змінювалися разом з апаратною базою комп'ютерів [1].

Створення вебдодатку сховища файлів має багато переваг, серед наведених технологій. Наприклад, таке сховище доступно в будь-який час і з будь-якого пристрою (при наявності швидкого Інтернету). Тому отримувати дані можна там, де вони потрібні в той чи інший момент [1].

Актуальність та проблематика роботи полягає в тому, що з популярністю мережевих додатків є великий запит суспільства на створення та імплементацію вебзастосунків для зберігання файлів користувачів у віддаленому просторі. Таким чином, інформацію можуть завантажувати на інший пристрій, маючи тільки дані облікового запису з якого було додані файли до сервісу. Також це значно зменшить витрату локальної пам'яті, що для користувача зменшить потребу на придбання накопичувачів.

Метою дипломного проекту є оцінка та аналіз існуючих методів та програмна реалізація вебдодатку сховища файлів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття хмарних технологій

Хмарні сервіси — це модель онлайн-сховища, що пропонує віддалену обробку для зберігання інформації. Ця технологія надає користувачам мережі Інтернет, доступ до комп'ютерних ресурсів сервера і використання програмного забезпечення як онлайн-сервісу. При з'єднанні до мережі Інтернет можна виконувати важкі розрахунки, обробляти дані застосовуючи потужності віддаленого сервера [2].

Прихід хмарних технологій є можливим завдяки зростанню методів хмарних обчислень, які здійснюються завдяки динамічно масштабованому підходу до розподілених зовнішніх мережеских засобів. Присвоєння відповідного доступу, як окремий сервіс, продовжує бути різновидом хмарних технологій [3].

ХС як програмний продукт, в основному реалізують в мережі Інтернет завдяки актуальним інтернет-браузерам. Що до створення ХС, застосовують обчислювальне середовище, такі як віртуальні машини, які працюють з громіздкими центрами зберігання даних та зменшують потребу користувачів на сервери та комп'ютерні пристрої. Основною різницею від простого застосування ПЗ в хмарному сховищі полягає в тому, що споживач має можливість з'єднувати власні ресурси фізичного персонального комп'ютеру та програмні доповнення, що надаються користувачу в ролі додатку у мережі. Звичайно, користувачу надається можливість до повного доступу на керування персональною інформацією, але він не має можливості керувати програмною базою чи операційною системою, завдяки котрим усі процеси відпрацьовують коректно [3].

Хмарні технології мають багато плюсів: споживач має можливість залучити персональний комп'ютер фактично будь-яких системних вимог для вирішення ресурсоємних робіт; сервіс можна використовувати при з'єднанні з

мережею Інтернет в будь-якому місці за умов використання комп'ютерного пристрою; користувач застрахований від збоїв у роботі пристрою і може за потреби ділитися результатами роботи з іншими користувачами. Також, одною з користувацьких переваг є те, що, на відміну від користування не безплатними сервісами на власному пристрої, більшість ХС при встановленні з різних платформ є безкоштовними, а вбудовані платні послуги проводять у виді тарифних планів. Зменшення витрат на підтримку, обслуговування, модернізацію та адміністрування комп'ютерного обладнання і програмного забезпечення є перевагою використання ХС для організацій [3].

1.1.1 Види розгортання хмарних сховищ

В цьому розділі, я хочу виділити три основних різновидів хмарних сховищ: приватне, публічне та гібридне.

Персональне хмарне сховище забезпечується з'єднанням до мережі персональним комп'ютером, який дає право завантажувати будь-які види власних файлів. Як приклад, сервіси для зберігання файлів у мережі допускають до системи графіку, фотографії, текст, музику та відео. Користувач має доступ до пристрою та управляє ним, а також може дати доступ іншому користувачу з будь-якої географічної точки через мережу. Персональний комп'ютер насправді мережевий накопичувач [4].

Для приватних хмарних сховищ застосовують персональні сервери завантаження, які повністю під керуванням організації, до якої вона належить. Аналогічно глобального хмарного додатка та центрів обробки файлів, приватне хмарне сховище застосовує потужність обчислювальних пристроїв. Найчастіше приватні вебзастосунки застосовують компанії, яким необхідна мобільність та зручність хмарного сервіса, хоча і під прямим застосуванням і контролем організації, до якої вона належить. Компанії, що турбуються про захист ПЗ, можуть надати перевагу для керування своєю

архітектурою хмарного сервісу для зберігання файлів, а не скористатися публічною програмою [4].

Змішане хмарне сховище - це таке собі поєднання публічної та приватної хмари та центру обробки файлами, яке вважають кращим для компаній. Здебільшого, він з'єднує матеріали, якими володіє та керує організація, з глобальними хмарними сервісами, які керуються третьою стороною. Компанії поєднують два метода, щоб збалансувати необхідність в збереженні дуже цінних активів з перевагами зручності, мобільності та гнучкості, які постачають глобальні хмарні сервіси [4].

1.1.2 Переваги використання

Переваги використання сервісів хмарного сховища:

- немає необхідності в потужних пристроях для зберігання інформації;
- користувач заощадить на придбанні платного програмного продукту та на його постійному оновленні;
- нелімітована кількість доступної пам'яті для користувача;
- доступність з різних пристроїв і відсутня прив'язка до робочого місця;
- забезпечення безпеки даних від втрат та виконання багатьох видів навчальної діяльності, контролю і оцінювання, тестування он-лайн, відкритості освітнього середовища;
- зберігання коштів на утримання технічних спеціалістів[5].

1.2 Сервіси зберігання даних

На даний момент, колосальний зріст в популярності хмарних додатків спостерігається у сфері інформаційно-комунікаційних технологій. Відповідно

до цього, у багатьох сферах людської діяльності в великій кількості стали неодноразово з'являтися хмарні сервіси. Найпопулярнішими з таких сервісів стали саме сховища даних у вигляді веб-застосунків. Концепція подібних додатків була реалізована ще в 60-х роках, завдяки знаменитим науковцям у сфері обчислювальної техніки та штучного інтелекту Джоном Маккарті та Джоозефом Ліклайде. Ще у недалекому минулому хмарні сервіси були виключно приватними, тобто незнайомими або недоступними для багатьох користувачів в мережі. Так чи інакше, змога використовувати подібні технології у малих та великих підприємствах та легкість застосування ХС для звичайних користувачів, привела до популярності їх у мережі Інтернет. Хмарне сховище даних – це парадигма, де інформація зберігається на багатьох, розповсюджених в різних точках Землі серверах, які надають право для розміщення файлів клієнтам, як правило, третьою стороною [6].

Таким чином, замість розміщення файлів на носіях зовнішньої пам'яті (або на вінчестерах комп'ютерів) інструменти і результати роботи поступово переносяться та розміщуються у хмарному сховищі даних або у "хмарі". Хмара представляє собою сукупність серверів (центр обробки даних, ЦОД), часто віддалених один від одного на великі відстані, об'єднаних високошвидкісною мережею і виконуючих специфічні завдання. Точне число серверів назвати важко (компанії тримають його в секреті), на сьогодні кількість серверів оцінюється в 2-2,5 млн і прогнозується їх збільшення до 10 млн. ЦОД підключені до Інтернету безліччю каналів, і коли користувач заходить почитати пошту або відредагувати фотографії, він потрапляє на найближчий і найменш завантажений вузол, який здійснює обробку інформації. Як взаємодіють між собою сервери всередині інфраструктури - таємниця розробника. А завдання користувача полягає в тому, щоб увійти в Інтернет, пройти авторизацію на обраному сервісі (див рис. 1.1) [6].



Рисунок 1.1 – Схема роботи хмарного сервісу зберігання даних [6]

У таких обставинах файли стають доступним для усіх віртуальних машин. Що треба зазначити, велику вагу певно має те, що великий обсяг сервісів є доступними або мало коштують. Серед плюсів використання вебзастосунків для збереження даних у мережі можна відокремити: перегляд файлів сховища можна виконати з будь-якого пристрою та в будь-якому місці за умовою безперервного під'єднання до мережі Інтернет; споживач платить тільки за той простір у сервісі, який просто таки використовує або користується певним місцем дискового простору додатку для збереження файлів безкоштовно; зменшення витрати дискового простору на пристроях зовнішньої пам'яті; усі дії для захисту цілісності інформації гарантуються провайдером сервісу [6].

До мінусів можна віднести: загроза у стадії завантаження та відправки файлів, зокрема приватних, таємних; загальний показник активності при змінних операціях з файлами в хмарному сервісі може бути нижче, ніж при операціях з персональними копіями дублікатів файлів; потрібно мати швидке та стабільне з'єднання до мережі Інтернет. Таким чином, загальною різницею між сервісами зберігання даних у мережі та пристроями зовнішньої пам'яті є: резервне копіювання файлів з пристрою до сервісу, синхронізація інформації між різними пристроями, колективна робота певної групи користувачів [6].

1.3 Огляд існуючих рішень проблеми

В цьому розділі розглянемо існуючі додатки для зберігання файлів в мережі.

Залежно від цілей використання хмарного диска, зверніть увагу на основні характеристики сховищ:

- вартість додаткового простору і його обмеження;
- редагування файлів на диску;
- шифрування;
- безкоштовний обсяг сховища;
- спільний доступ до файлів і папок;
- робота з різних пристроїв під одним аккаунтом;
- максимальний обсяг завантаження;
- список форматів файлів, що підтримують попередній перегляд.

1.3.1 Microsoft OneDrive

OneDrive підтримує будь-яку мову та дозволяє збільшувати сховище від 5 до 100 гігабайт. Придбати додатковий простір для бізнесу або сім'ї можливо тільки через покупку Microsoft 365 разом з додатковими програмами і функціями [7].

Функції:

- управління рівнем доступу;
- додатковий захист через особистий сейф: подвійна аутентифікація для входу;
- можливість перегляду обраних файлів без доступу до інтернет;
- використання сховища через веб-версію, мобільне і десктоп-додатки, підтримка XBOX[7].

Особливості:

- розмір завантаження не більше 15 Гбайт;
- важка процедура реєстрації бізнес-тарифу і покупки;
- безліч додаткових функцій крім зберігання[7].

Графічний інтерфейс програми наведено на рисунку 1.2.

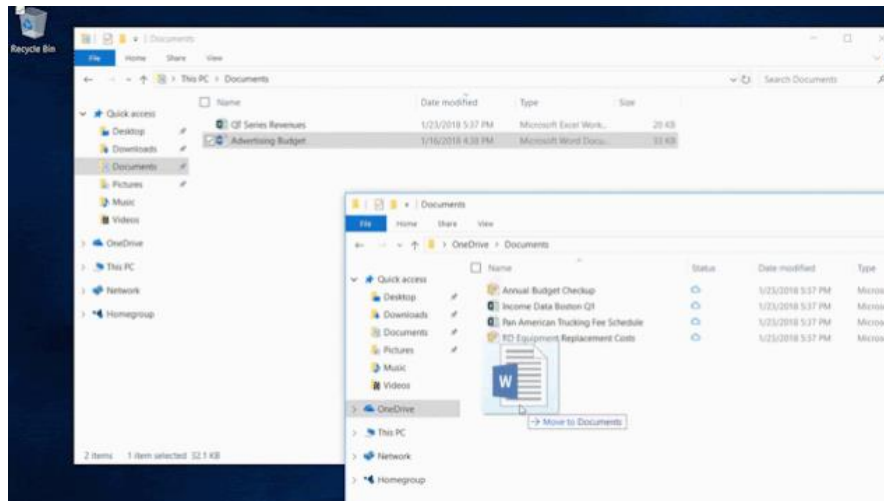


Рисунок 1.2 - Графічний інтерфейс системи “Microsoft OneDrive” [7]

1.3.2 Google Диск

Google Диск - сама містка хмара з усіх перерахованих. Максимально в нього можна завантажити до 30 Тбайт. Редагувати файли можна навіть без мережі: дані зберігаються автоматично. Рівні доступу дозволять керувати обміном папок і файлів і встановити заборону на копіювання важливі документи [7].

Функції:

- попередній перегляд і редагування більшості форматів через Google Docs;
- використання через веб-версію, десктопних і мобільне додатки;
- до окремих файлів можна підключити офлайн-доступ;

- безлімітне зберігання фотографій через застосунок на мобільних пристроях;
- зберігання історії змін файлів;
- шифрування при передачі і зберіганні даних [7].

Особливості:

- дисковий простір включає файли Gmail пошти;
- розмір завантаження - не більше 5 Тбайт [7].

Графічний інтерфейс програми наведено на рисунку 1.3.

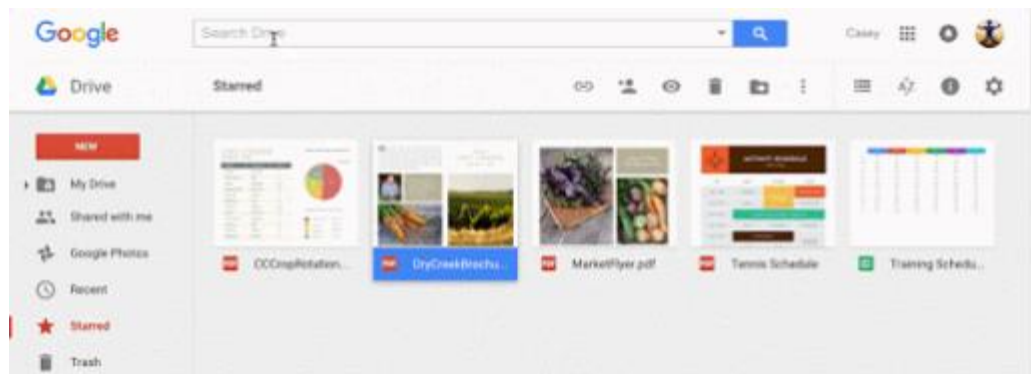


Рисунок 1.3 - Графічний інтерфейс системи “Google Диск” [7]

1.3.3 Dropbox

Хмара з невеликим безкоштовним сховищем, але можливістю покупки необмеженого простору в хмарі. Dropbox зберігає історію змін файлів, допомагає керувати рівнями доступу та коментувати відразу в сховище [7].

Функції:

- зберігання історії змін файлів і папок;
- необмежений розмір завантажуваних файлів (обмеження тільки в веб-версії - до 50 Гбайт) ;
- підключення сторонніх сервісів для управління командами: Trello, Slack, Jira;

- наявність реферальної програми по збільшенню дискового простору за запрошеного друга;
- використання через веб-версію, десктопних і мобільні додатки;
- доступ до файлів за паролем [7].

Особливості:

- на безкоштовному тарифі можна синхронізувати тільки три пристрої [7].

Графічний інтерфейс програми наведено на рисунку 1.4.

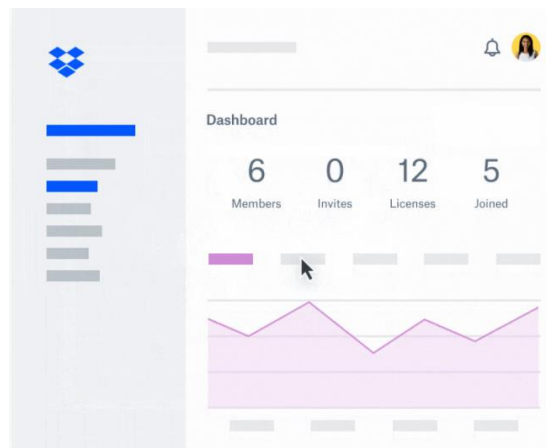


Рисунок 1.4 - Графічний інтерфейс системи “Dropbox ” [7]

1.3.4 Mega

Головна особливість сховища Mega - наскрізне шифрування даних для всіх користувачів. Це означає, що хмара не може дивитися ваші завантаження. У тарифах є обмеження на обсяг передачі даних в місяць від 1 до 16 Тбайт [7].

Функції:

- шифрування всіх даних навіть на безкоштовному тарифі;
- наявність програми з безкоштовного збільшення дискового простору: запрошення одного, установка програми та десктопної версії;
- використання через веб-версію, десктопних і мобільні додатки, розширення в браузері;

- інтеграція з поштовими клієнтами Mozilla Thunderbird [7].

Особливості:

- обмеження за обсягом передачі даних [7].

Графічний інтерфейс програми наведено на рисунку 1.5.

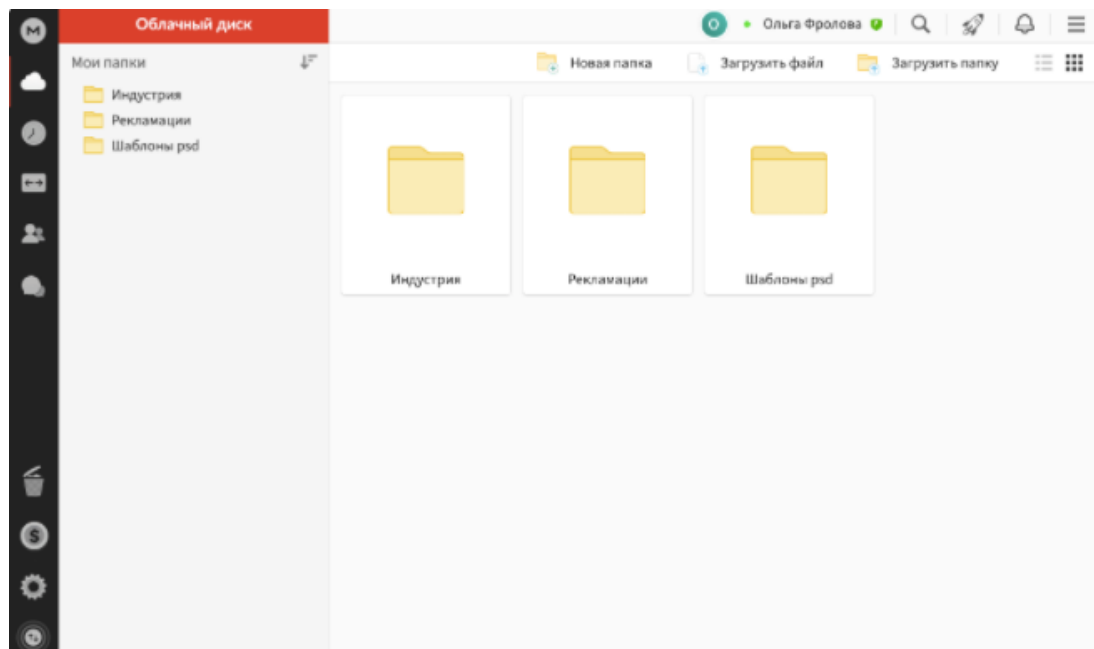


Рисунок 1.5 - Графічний інтерфейс системи “Mega ” [7]

1.3.5 MediaFire

В минулому файлообмінник MediaFire надає безкоштовне сховище до 10 гігабайт. Для збільшення простору є два тарифи: індивідуальний PRO і бізнес. На окремі файли і папки можна встановлювати пароль, обмежувати скачування [7].

Функції:

- захист файлів паролем;
- генерація одноразових посилань на скачування файлів;
- підтримує завантження папок;
- працює через веб-версію та мобільний додаток.

Особливості:

- при скачуванні з безкоштовного аккаунта буде реклама;
- розмір завантаження не більше 20 Гбайт [7].

Графічний інтерфейс програми наведено на рисунку 1.6.

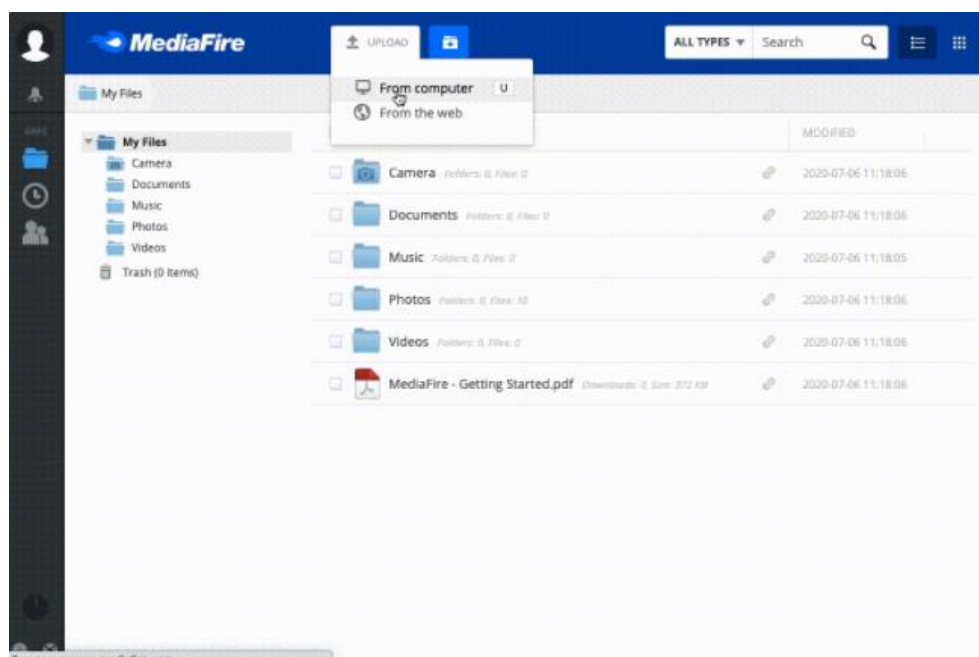


Рисунок 1.6 - Графічний інтерфейс системи “MediaFire” [7]

1.4 Висновок до розділу

В даному розділі було досліджено предметну область, а саме розглянуті існуючі рішення вебзастосунків для сховища файлів. Поставлена була задача на додавання та надійне збереження інформації користувачів. З збільшенням використання різних гаджетів (телефони, планшети, консолі, тощо.) зростає попит клієнтів на створення хмарних сервісів зберігання даних.

2 ОПИС ПРОГРАМИ

2.1 Основні засоби веброзробки

2.1.1 Мова гіпертекстової розмітки HTML

HTML - це стандартизована мова розмітки, що дозволяє користувачу працювати з блоками та структурувати елементи, списки, заголовки, посилання та блоки для веб-сторінок та веб застосунків. Суспільство не признає HTML професійною мовою програмування, за те що він не має можливості реалізувати динамічні функції [8].

Як і переважна більшість речей, які так чи інакше пов'язані з веб розробкою, HTML містить в собі як і плюси, так і мінуси:

Переваги:

- мова широко застосовується з величезною чисельністю ресурсів і має велику спільку розробників;
- на початку завжди виконується в будь-якому веб-браузері;
- легкий порок входження при вивченні;
- у розгорнутому доступі і абсолютно безкоштовний;
- загальні стандарти для веб-сторінок піддержуються користувальницькою умовою World Wide Web;
- логічна та прозора розмітка;
- зручно вбудовується з базисними мовами програмування, такими як JavaScript, PHP, тощо [8].

Недоліки:

- не повністю пропонує користувачу створити сторінку з логікою. У кінцевому етапі, усі сторінки треба додавати до мережі окремо, навіть якщо вони беруть ті ж самі елементи, наприклад: блоки чи зображення;
- в першу чергу використовують для реалізації статичних сторінок у мережі. Для додавання динамічності на сторінці необхідно використати такі

мови програмування як JavaScript або мову для створення серверної частини, як C#, Java;

- старі браузери не підтримують нові функції тому, працюють повільно;
- часом поведінку браузера складно передбачати (наприклад, старі браузери не всякий раз роблять нові теги) [8].

Отже, HTML є стандартизованою мовою розмітки в мережі. З початку роботи, він розгортається в кожному браузері та піддержуються користувальницькою умовою World Wide Web [8].

Користувач має можливість застосовувати його для реалізації структури вмісту веб-додатків та веб-сайтів. Серед знань розробників технологій у фронтенді, HTML розташовується на базовому етапі, який виступає основою для блокової стилізації, яку ви можете під'єднати шляхом додавання CSS до проекту та функціональність, яку ви можете під'єднати шляхом додавання JavaScript [8].

2.1.2 Каскадна таблиця стилів – CSS

Головною ціллю CSS – конструктор зовнішнього вигляду сторінки, її змісту. Якщо сторінка буде реалізована виключно на «чистому» HTML, то в такому документі буде визначений не тільки кожен елемент, але і метод її відображення кінцевому користувачеві (конкретний шрифт, колір, положення картинки і т. д.) [9].

Якщо ж крім HTML використовувати в документі ще і CSS (підключити каскадну таблицю стилів), то сам HTML буде вказувати браузеру виключно черговість відображення елементів на сторінці, а всі їхні властивості і параметри будуть виражені за допомогою CSS [9].

У HTML-документі не потрібно весь час перераховувати використовувані стилі CSS - досить вказати їх лише один раз, щоб підключити до документа [9].

Подібне рішення дозволяє вирішити такі завдання і проблеми:

- створювати веб-сторінки відносно швидко і легко, так як одного разу реалізоване оформлення можна використовувати в документі багаторазово - для різних сторінок;
- робить редагування максимально гнучким і зручним, так як досить лише раз поправити CSS, щоб змінити зовнішній вигляд одночасно кілька сторінок;
- значно спрощує код, зменшуючи кількість повторень його різних елементів, такий код стає просто читати навіть тим розробникам, які до цього його не бачили;
- прискорює завантаження веб-сторінки, що обумовлено кешуванням CSS при першому відкритті сайту, а в майбутньому ці дані будуть просто зчитуватися, відображаючи лише структуру документа;
- розширює список можливих графічних елементів, які можна використовувати для подання вмісту веб-сторінки;
- дозволяє швидко і легко змінювати стилі на багатьох сторінках (наприклад, верстати адаптовану під мобільні пристрої веб-сторінку або включати в документ стилі) [9].

Каскадні таблиці стилів CSS дозволяють не тільки реалізовувати на сайті все задумки веб-дизайнера, але і в значній мірі змінюють підхід до створення сайтів на HTML, спрощують роботу програмістам, дозволяючи їм використовувати всі доступні елементи з максимальною гнучкістю і простотою [9].

2.1.3 Мова програмування JavaScript

JavaScript - мова програмування, яка дозволяє вставити на сторінки веб-додатку будь-які інтерактивні компоненти: спливаючі вікна, анімації, міні-ігри, «плаваючі» форми, динамічний стилі оформлення і багато іншого [9].

JS являє собою динамічну, повноцінну мову програмування, яку можна використовувати спільно з HTML-документами. Ця мова забезпечує динамічну інтерактивність веб-ресурсу [9].

Творець мови JS - Брендан Ейх Ву, один з співзасновників таких великих проектів, як Mozilla Corporation і Mozilla Foundation. Саме він зумів створити JS - універсальна мова, яку легко освоїти навіть початківцю веб-майстру [9].

Професійні програмісти з великим досвідом можуть за допомогою JS створювати повноцінні гри, двомірну і тривимірну анімацію, потужні програми для роботи з великими обсягами даних, тощо [9].

Сам по собі JS відрізняється компактністю і одночасної гнучкістю. Поверх основної мови JavaScript, його розробниками було створено велику кількість додаткових інструментів і функцій [9].

Серед таких можна виділити наступні:

- API - програмні інтерфейси додатків, які вбудовані в інтернет-браузер і надають користувачеві масу додаткових можливостей у вигляді роботи з різними елементами сайту для генерації тривимірної графіки, обробки аудіо-семплів, динамічного створення HTML-сторінок, взаємодії з веб-камерою, установку додаткових стилів CSS і т. д;
- сторонні API від інших розробників, які дозволяють використовувати на своїх сайтах ширші можливості в плані впровадження готових рішень від Facebook, Twitter;
- можливість використовувати для свого HTML додаткові фреймворки і бібліотеки, що прискорюють створення додатків і веб-сайтів [9].

JavaScript, який використовують сучасні розробники, можна вважати «безпечною» мовою програмування. Він не відкриває доступ до процесора або пам'яті пристрою, адже спочатку був створений виключно для браузерів [9].

Можливості JavaScript дуже великі і безпосередньо залежать від оточення, де буде працювати мова програмування. Наприклад, Node.JS забезпечує підтримку запису і читання будь-яких типів файлів, а також «уміє» виконувати мережеві запити, тощо [9].

У браузері, що підтримує роботу з JavaScript, є все, що необхідно сучасному користувачеві, який часто відвідує веб-сторінки з динамічним вмістом, а також будь-яким чином взаємодіє з сервером [9].

Наприклад, JavaScript може в браузері:

- встановлювати нові HTML-коди, які здатні змінити вміст сторінки, модифікувати CSS;
- «відповідати» на певні дії користувача сайту, кліки кнопками миші, переміщення курсора, натискання на клавіші і т. д;
- відправляти мережеві запити на сервери, завантажувати відео, фото, аудіо контент, а також документи та інші файли;
- отримувати cookie-файли і виконувати їх установку, можуть поставити питання відвідувачеві сторінки, виводити певні повідомлення;
- зберігати будь-яку інформацію на стороні клієнта [9].

2.2 Вибір клієнтського середовища проектування

Frontend (фронтенд) представляє собою зовнішню частину сайту (веб-додатки), яка забезпечує пряму взаємодію користувача з веб-ресурсом. У поняття frontend входить:

- висновок на сайті інтерфейсу користувача;
- виконання функціональних завдань;
- обробка певних запитів відвідувачів [9].

Фронтенд - це буквально все, що бачить відвідувач сайту, відкриваючи будь-яку його сторінку. Frontend-розробка має на увазі роботу програміста над створенням видимої частини веб-ресурсу, з якої відвідувач буде контактувати. У frontend-розробку входить створення функціоналу сайту, який може виконуватися на стороні клієнта [9].

Frontend-розробник працює над тим, щоб на сайті працювали всі кнопки, іконки, форми зв'язку і підписки, правильно відображалися тексти і графічні елементи, щоб всі вони «стояли на своїх місцях» і виконували ті завдання, які перед ними були поставлені (наприклад, щоб при клієнті по кнопці «замовити» відкривалася форма для оформлення покупки, а при натисканні на кнопку «play» включалося відео) [9].

Щоб кінцевий продукт frontend-розробника був якісний, затребуваний серед користувачів і відповідав поставленим перед ним вимогам, необхідно забезпечити грамотне взаємодія цього розробника з усіма іншими фахівцями, які працюють над завданням: веб-дизайнерами, маркетологами, SMM-ками, аналітиками, тощо [9].

Фреймворки React.js, AngularJS, Vue.js – основні платформи, що дають фронтенд-розробнику основу для створення веб-додатків. У них присутні вже визначені і реалізовані функції з класами. Також в фреймворках можна додавати свій код до вже запропонованого [9].

Для створення свого вебдодатку, я вирішив зробити вибір між React, Angular, Vue.js та провести певний аналіз по основним характеристикам у виді таблиці, а також вибрати оптимальний варіант для розробки свого проекту.

Результати порівняння наведено у таблиці 2.1.

Таблиця 2.1 — Порівняння існуючих фронтенд-платформ

	React	Vue	Angular
Тип	Бібліотека	Фреймворк	Фреймворк
Складність вивчення	Середній	Легкий	Складний
Модель	Virtual DOM	Virtual DOM	MVC
Типи додатків	Односторінкові і мобільні додатки	Просунуті односторінкові додатки і підтримка нативних додатків	Нативні, гібридні та веб-додатки
Мова системи	JavaScript	JavaScript	TypeScript
Підтримка спільнотою	Спільнота розробників Facebook.	Проект з відкритим вихідним кодом, підтримуваний співтовариством.	Велике співтовариство розробників.
Використовується в компаніях	Twitter, Reddit, PayPal	Baidu, GitLab	Google, Forbes, Wix

Після проведення аналізу існуючих фронтенд-платформ, я вирішив обрати саме бібліотеку React. React – це потужна декларативна JavaScript-бібліотека, яку веб-розробники використовують для створення інтерактивних інтерфейсів веб-сайтів. Це інструмент з відкритим вихідним кодом, який був розроблений і підтримується компанією Facebook. За допомогою цього рішення можна створювати масштабні і складні веб-додатки, які мають справу з великим трафіком і часто змінюваними даними. React.js приділяє основну увагу незалежним компонентам, які спрощують роботу, так як їх можна використовувати повторно. Однією з найважливіших особливостей React.js є унікальний синтаксис, JSX, який в цілому спрощує процес програмування.

2.2.1 Управління станом

Redux - це інструмент для керування станом даних і призначеним для користувача інтерфейсом в додатках JavaScript. Він ідеальний для односторінкових додатків, в яких управління станом згодом може стати складним. Redux не прив'язаний до якогось фреймворку, хоча він написаний з орієнтацією на React, його можна використовувати навіть з Angular [10].

React не рекомендує використовувати безпосередню взаємодію компонентів. Навіть якби в React були можливості для підтримки цього підходу, він би розцінювався як погана практика, так як безпосередню взаємодію компонентів схильний до помилок і веде до спагетті-коду [10].

Redux пропонує вирішити це шляхом зберігання всього стану програми в одному місці - сховище. Компоненти передають зміни стану в сховище, а не безпосередньо іншим елементам. Компонент, яким важливі ці зміни може підписатися на них в сховищі [10].

З Redux очевидно, що всі компоненти отримують свої статки зі сховища. Також очевидно, що всі компоненти відправляють зміни свого стану в сховище. Компонент при запуску змін думає тільки про відправку змін в сховище і не турбується про всі компоненти, яким потрібні ці зміни. Так Redux робить потік даних простіше і зрозуміліше [10].

Архітектура Redux зображена на рис. 2.1.

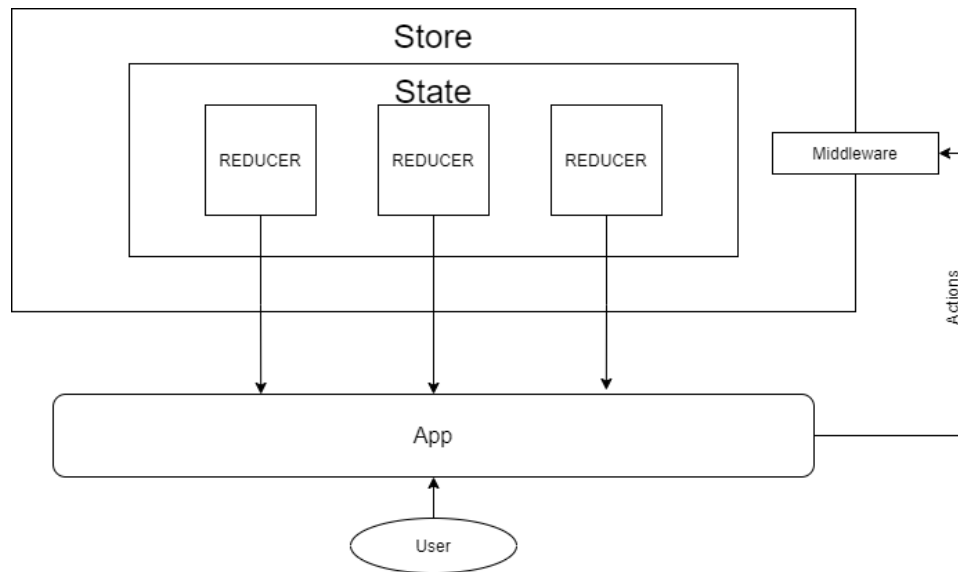


Рисунок 2.1 –Архітектура Redux

ACTION: це чистий об'єкт створений для зберігання інформації пов'язаної з подією користувача (Натискання кнопки миші на інтерфейсі, ..), він включає інформацію як: вид дії, місце походження, час, координати, в який стан (state) змінюється [11].

STORE: Управляє станом додатка і має функцію `dispatch (action)` [11].

MIDDLEWARE: (Проміжне програмне забезпечення) Надає спосіб взаємодії з об'єктами Action відправлені до STORE перед тим, як вони відправляються до REDUCER. У Middleware ви можете виконати такі завдання як запис журналів, оповіщення помилок, створення "асинхронізовані запитів" (asynchronous requests), або відправка (dispatch) нових action [11].

REDUCER: (Редуктор) Це чиста функція повертає новий стан з початкового стану [11].

2.2.2 Virtual DOM React

Віртуальний DOM (VDOM) - це концепція програмування, в якій ідеальне або «віртуальне» уявлення призначеного для користувача інтерфейсу

зберігається в пам'яті і синхронізується з «реальним» DOM за допомогою бібліотеки, такий як ReactDOM. Даний процес називається узгодженням [12].

Даний підхід забезпечується декларативним React API: ви говорите React, в якому стані повинен перебувати UI, і він гарантує, що DOM відповідає цьому стану. Це відволікає маніпуляції з атрибутами, обробку подій і ручне оновлення DOM, які в іншому випадку вам довелося б використовувати для створення вашого застосування [12].

«Віртуальний DOM» є скоріш шаблоном проектування, ніж конкретної технологією. У світі React термін «віртуальний DOM» зазвичай асоціюється з елементами React, оскільки вони є об'єктами, що представляють призначений для користувача інтерфейс. React, однак, також вживає внутрішні об'єкти, звані «волокнами» («fibers»), для зберігання додаткової інформації в дереві компонентів. Вони також можуть вважатися частиною реалізації «віртуального DOM» в React [12].

2.3 Вибір серверного середовища проектування

Node.JS - технологія, яка трансформувала знайомий всім веб-розробникам JavaScript з мови клієнтського скриптинга в мову загального призначення, який має велику кількість варіантів прикладного використання [13].

В основі роботи Node.js лежить асинхронність. На застосунок до неблокуючих вводу-виводу це дозволяє серверному додатку на Node.js обслуговувати набагато більше клієнтських запитів в одиницю часу, ніж аналогічного додатком, розробленим на більшості інших технологій серверної розробки [13].

Node дозволяє швидко розробити MVP. У Node вже розроблено велику кількість пакетів з різним функціоналом. Можна не витрачати час на розробку базового функціоналу, а відразу сконцентруватися на бізнес-логіці [13].

Технологія відмінно підходить для більшості проєктів реалізуються в сучасному світі веб-розробки. Засновники майже будь-якого проєкту на старті мріють про те, щоб їх продуктом користувалося якомога більше людей [13].

2.4 Вибір бази даних

В останні роки здобуває популярність технологія NoSQL бази даних. Одна з найпопулярніших серед них – MongoDB [14].

NoSQL-модель відрізняється від звичайної реляційної моделі. Про це необхідно знати, реалізуючи архітектуру програмного застосунку. Неправильний вибір дизайну погано позначиться на масштабованості та продуктивності [14].

MongoDB часто представляють як базу даних, у якій немає схеми. Але це невірно: схема є, але вона динамічна. Її називають поліморфной, так як вона може трансформуватися навіть на документах однієї і тієї ж колекції. Це дозволяє зберігати разом різні набори інформації, що в умовах бурхливого розвитку неструктурованих великих даних є потужним конкурентною перевагою [14].

Коли мова заходить про об'єктно-орієнтованому програмуванні (ООП) і спадкування, поліморфні можливості MongoDB стають особливо важливими і зручними. Завдяки їй можна серіалізувати екземпляри різних класів однієї ієрархії (батько-нащадок) в одну колекцію, а потім десеріалізувати їх назад в об'єкти. У реляційних базах даних такий прийом не вдалося б провести так просто, адже таблиці мають фіксовані схеми [14].

Для схеми бази даних це твердження так само вірно, як і для всього іншого. Необхідність змінити структуру традиційної реляційної бази може стати головним болем для розробника. Нормалізована таблична схема чудово усуває надмірність даних. Але при невеликій зміні однієї таблиці побічні ефекти можуть проявитися самим несподіваним чином [14].

Зазвичай це відбувається так:

- застосунок зупиняється і створюється резервна копія бази;
- запускаються складні сценарії міграції для підтримки нової схеми;
- код переробляється, щоб взаємодіяти з цією схемою;
- після внесення всіх змін, застосунок перезапускається [14].

Однак MongoDB з коробки підтримує зворотню сумісність. Розробники враховують ці зміни в коді на стороні сервера. Як тільки застосунок буде оновлено, щоб впоратися зі змінами бази, ми можемо перенести потрібну колекцію в фоновому режимі, не зупиняючи його. Після міграції можна замінити код додатка для остаточного впровадження змін [14].

2.5 Вибір допоміжних інструментів

Велику роль під час створення програмного забезпечення, крім інструментів для написання коду і розгортання системи, грають засоби для автоматизації процесу розробки.

У якості середовища для розробки (IDE) було обрано Visual Studio Code. VS Code - це середовище розробки від компанії Microsoft, створена зовсім недавно з використанням нових технологій (наприклад Node.js). На відміну від повного пакета Visual Studio, це середовище являє собою текстовий редактор з можливістю підключення великої кількості плагінів. Вона призначена для розробки на різних мовах програмування і для різних платформ. Підтримуються не тільки Windows, але і Linux, а також MacOS [15].

2.6 Архітектура вебзастосунку

Весь вебзастосунок я планую повністю реалізувати за допомогою мови програмування JavaScript.

Клієнтська частина додатку буде імплементована за допомогою бібліотеки React та інструментом управління станом Redux. Серверна частина сайту буде реалізована за допомогою платформи Node.js та фреймворка Express. Базою даних була обрана MongoDB та ODM Mongoose.

Загальна архітектура вебзастосунку зображена на рис. 2.2.

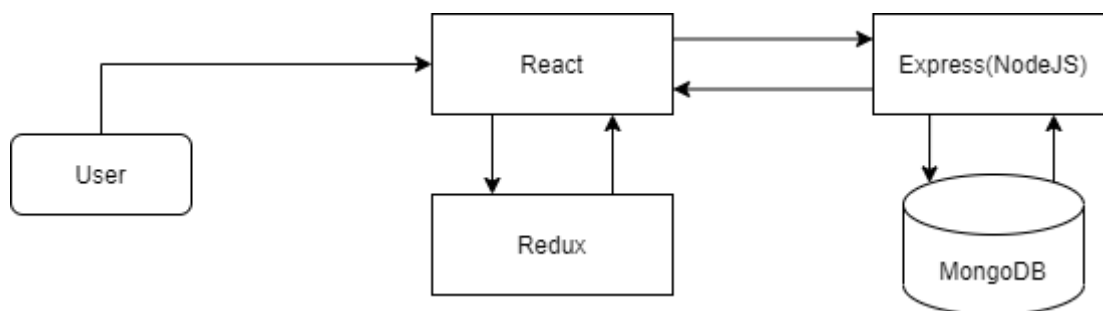


Рисунок 2.2 – Загальна архітектура вебзастосунку

2.7 Вимоги до програмного забезпечення, що розробляється

Розроблювана програма повинна виконувати наступні функції:

- реєстрація та авторизація користувача;
- зміна персональної інформації користувача;
- загрузка файлів на віддалений простір;
- завантаження та видалення файлів з сховища;
- пошук файлу по назві;
- фільтрація файлів по типу файлу, назві, даті;
- створення папок.

Діаграма функцій користувача представлена на рис. 2.3

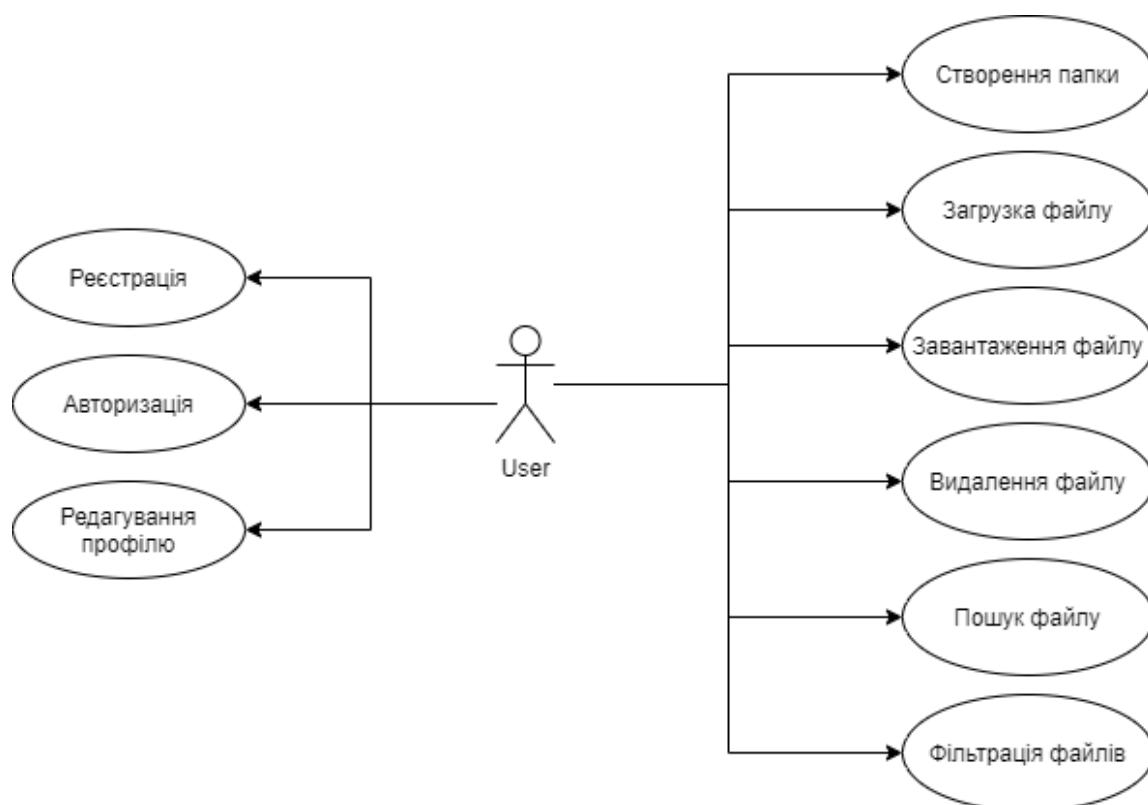


Рисунок 2.3 – Діаграма функцій користувача

При цьому ПЗ, що розробляється, повинно бути реалізовано у вигляді вебсайту, доступ до якого повинен бути можливий на будь-якому пристрої, у якому наявний доступ до мережи інтернет.

2.8 Висновки за розділом

В цьому розділі було розглянуто основні інструменти для імплементація веб-додатку, такі як HTML, CSS, JavaScript.

Було представлено загальну архітектуру для обраного стеку веб-розробки ПЗ.

Сформульовано функціональні вимоги до розроблюваного вебсайту, які стосуються авторизації та дій, які притаманні саме темі проекту: створення папки, загрузка файлу, завантаження файлу, видалення файлу, пошук файлу та фільтрація файлів.

Таким чином, для створення клієнтської частини було обрано бібліотеку React, Redux, для серверної частини – Express та Node.js. Базою даних було обрано MongoDB(NoSQL). Як середовище для розробки (IDE) було обрано Visual Studio Code.

3 РОЗРОБКА ПРОГРАМИ

3.1 База даних

Для проектування схеми бази даних використовується безкоштовний вебзастосунок draw.io .

База даних проекту складається з двох колекцій: користувач та файл. Схеми бази даних представлена на рисунку 3.1.

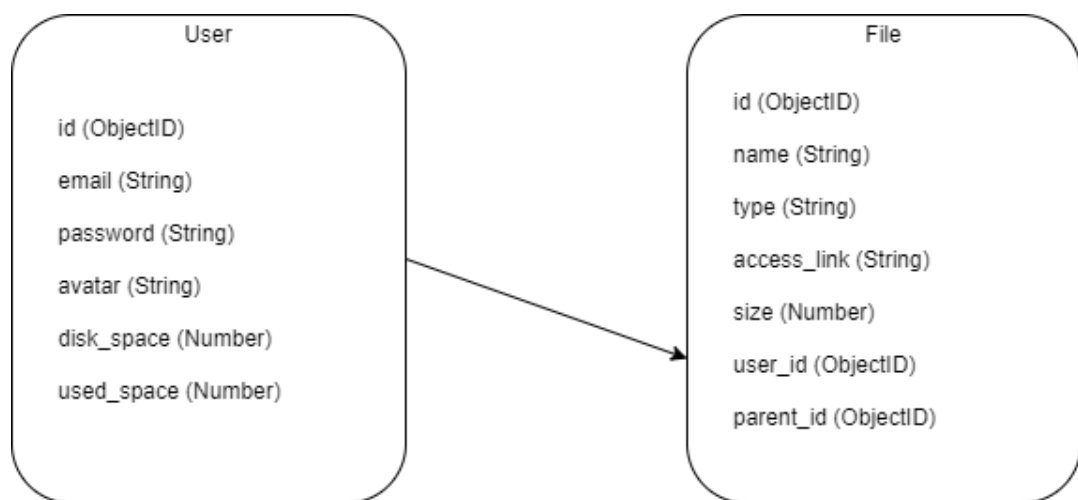


Рисунок 3.1 – Схеми бази даних

Колекція «Користувач» має наступні властивості: унікальний індекс, поштова адреса, пароль (зашифрований), аватар користувача, виділений для користувача простір, зайнятий користувачем простір.

Колекція «Файл» має наступні властивості: унікальний індекс, назва файлу, тип файлу, директорія файлу, розмір файлу, індекс власника файлу, індекс папки до якого належить файл.

Вигляд колекції у середовищі mongoDB представлено на рисунку 3.2.

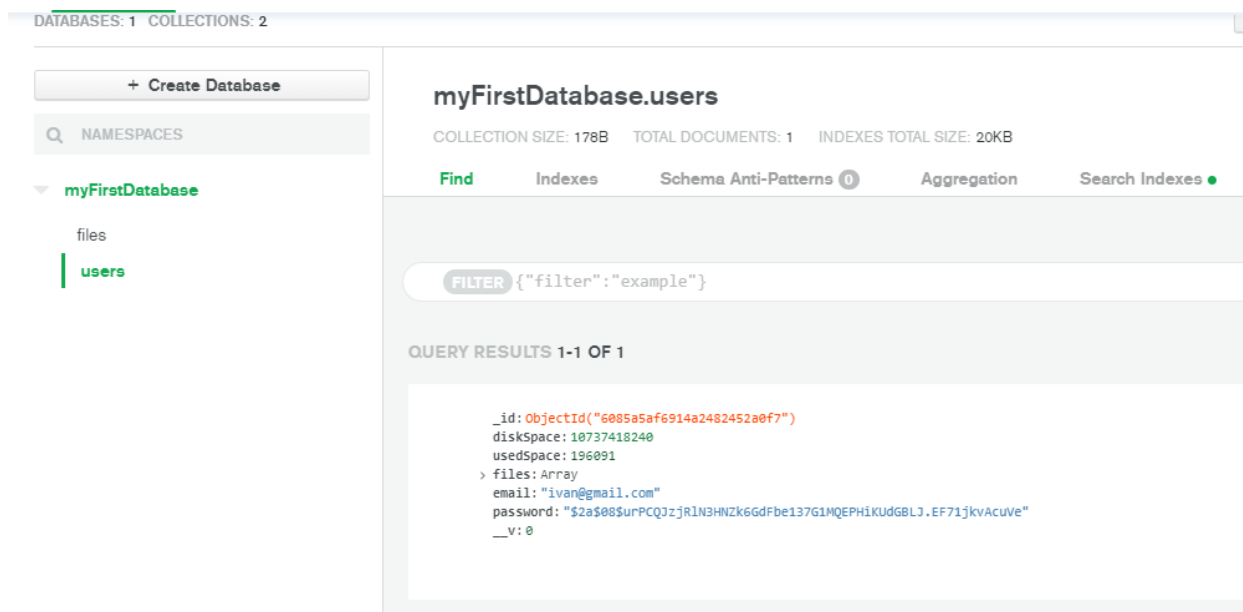


Рисунок 3.2 – Вигляд колекції «Користувач» у середовищі mongoDB

3.2 Алгоритм функціонування вебсайту

Взаємодія з програмним забезпеченням для неавторизованого користувача починається з сторінки авторизації. Серед доступних сторінок для гостя є тільки «Логін» та «Реєстрація», а для авторизованого користувач буде вже доступна головна сторінка з сховищем. У сховищі, перше, на що користувач має звернути увагу, так це на кнопку завантаження файлу та додавання папки. Створення папки дуже спрощує роботу з великою кількістю файлів, завдяки створенню зручної структури у середині сховища. Також для зручності було додано поле пошуку, зміна виду та сортування файлів. Основними функціями для маніпулювання з файлами є завантаження та видалення з сховища.

Функціональна схема взаємодії із вебсайтом представлена на рисунку 3.3.

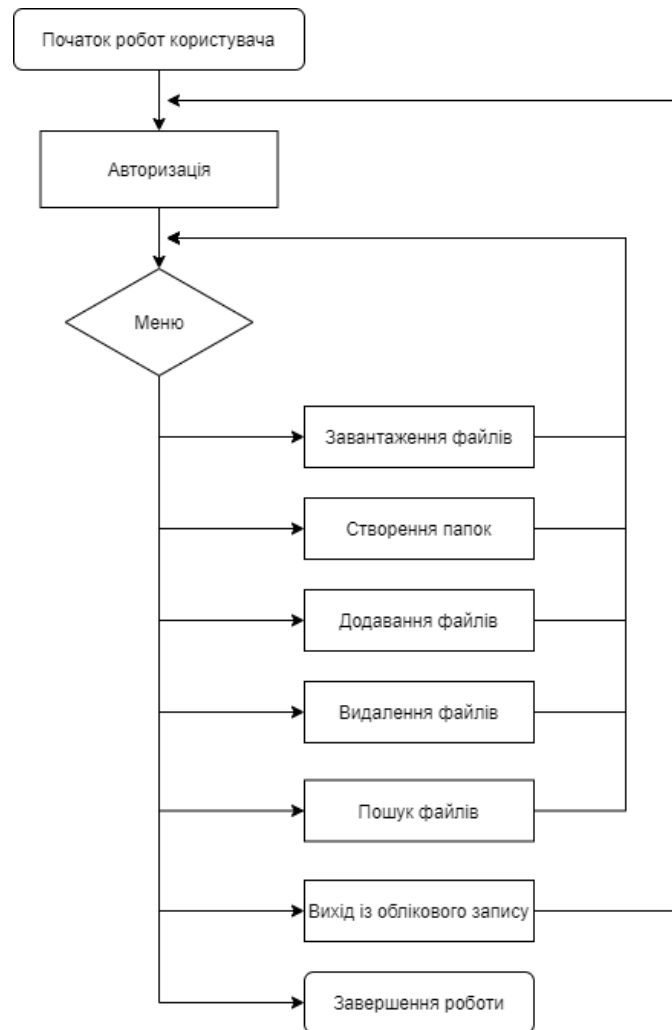


Рисунок 3.3 – Функціональна схема програми

3.3 Структурна схема проекту

В цьому підрозділі буде представлено структурні схеми клієнтської та серверної частини нашого проекту.

3.3.1 Структурна схема клієнтської частини

Структура директорій клієнтської частини проекту представлена на рисунку 3.4.

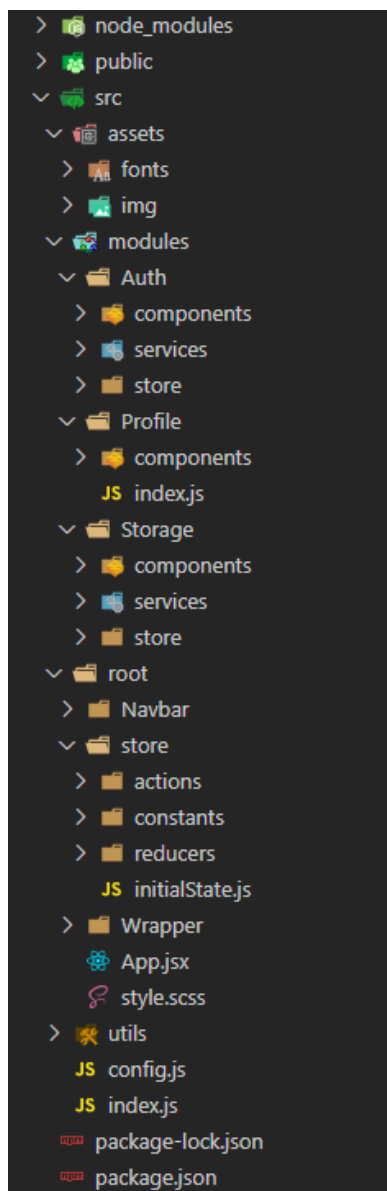


Рисунок 3.4 – Структура директорій клієнтської частини проекту

Директорія `assets` призначена для зберігання налаштувань для нашого проекту: логотипи, зображення, шрифти, звуки, тощо. В даному випадку, маємо дві папки: `fonts` (шрифти) та `img` (зображення).

Директорія `modules` має в собі усі сутності нашого проекту: `Auth`, `Profile`, `Storage`. Всередині кожного маємо: папку з React-компонентами (`components`); папку, що з'єднують зв'язок між фронтом і бекем та містить у собі запити, що формуються на клієнті та відсилаються до серверної частини (`services`); папка `store`, де ми працюємо зі станом нашої сутності. Директорія `store` має:

reducers – місце, де перерисовується дерево стану в залежності від обраного кейсу-типу; actions – приймає два аргумента (type та payload); constants – список змінних (types), що експортується в reducers та actions; файл initialState – стан, який в нас буде за замовчуванням.

Директорія root призначена для зберігання головних файлів проекту: navbar (навігаційна панель) та wrapper(місце, де зберігається маршрутизація по проекту та пов'язані з ними компоненти).

Директорія utils призначена для зберігання загальних стилів та стилізованих компонентів. В даному випадку маємо стилізований тег Input.

3.3.2 Структурна схема серверної частини

Структура директорій серверної частини проекту представлена на рисунку 3.5

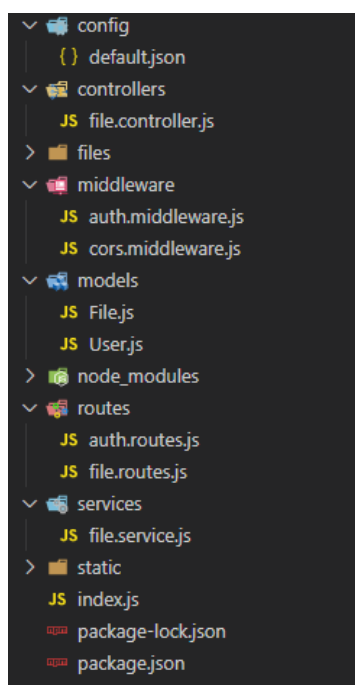


Рисунок 3.5 – Структура директорій серверної частини проекту

Директорія `config` призначена для зберігання інформації, яку треба не допустити у відкритому коді. Наприклад, токен, який не повинен попасти при публікації до репозиторію.

Директорія `controllers` призначена для файлів, які містять основну логіку кожного запиту. В тих файлах прописується усі змінні функції відносно до бази даних.

Директорія `files` містить усі файли, що були створенні користувачами веб-додатку.

Директорія `middleware` призначена для ролі посередника між вхідними даними та контролерами. Файл `auth.middleware` служить для дешифрування вхідного токена при авторизації користувача. Файл `cors.middleware` служить для отримання доступу зі сторони клієнта до запитів, що знаходяться на сервері.

Директорія `models` призначена для моделей колекцій, які будуть в нашій БД. У файлі моделі колекції прописані усі властивості та їх типи даних.

Директорія `routes` призначена для адрес до яких підв'язують певні контролери. Таким чином, ми встановлюємо посилання, завдяки якому ми можемо імплементувати коректний запит з клієнту.

Директорія `services` призначена для роботи з файловою системою, встановленням певного директорію, де будуть зберігатись усі файли та папки користувачів.

Директорія `static` містить у собі усі статичні файли, що були завантаженні користувачами веб-додатку.

3.4 Опис реалізованих функцій

В даному підрозділі розглянемо основні функції, які будуть використовуватись у проекті. Розглянемо функції, що взаємодіють між користувачами та клієнтською частиною сайту та функції, що знаходяться на серверній частині та приймають запити з клієнту.

3.4.1 Функції клієнтської частини (front-end)

Функції, що викликаються під час взаємодії користувача із клієнтською частиною:

- `showLoaderSpinner()` – функція, що відображає завантажувальний спіннер;
- `closeLoaderSpinner()` – функція, що вимикає спіннер при повному завантаженні сторінки;
- `getUserFiles()` – функція, що отримує усі файли авторизованого користувача;
- `createNewFolder()` – функція, що створює нову папку у сховищі користувача;
- `uploadNewFile()` – функція, що додає новий файл до сховища користувача;
- `downloadCurrentFile()` – функція, що завантажує файл із сховища до пристрою користувача.
- `deleteCurrentFile()` – функція, що видаляє файл із сховища користувача;
- `searchFiles()` – функція пошуку по файлам сховища користувача;
- `setPopupDisplay()` – функція, що вмикає модальне вікно з формою створення нового файлу;
- `setFileView()` – функція, що встановлює вид списку файлів користувача;
- `showUploader()` – функція, що відображає модальне вікно з прогресом завантаження файлів до сховища;
- `hideUploader()` – функція, що ховає модальне вікно з прогресом завантаження файлів до сховища;
- `addUploadFile()` – функція, що додає прогрес завантаження нового файлу до сховища;

- `removeUploadFile()` – функція, що видаляє прогрес завантаження файлу до сховища;
- `uploadAvatar()` – функція, що завантажує зображення з пристрою та встановлює до аватару користувача у системі;
- `deleteAvatar()` – функція, що видаляє поточний аватар користувача у системі;
- `registration()` – функція, що реєструє нового користувача до веб-додатку;
- `login()` – функція, що авторизує користувача та надає доступ до системи;
- `auth()` – функція, що встановлює токен з серверу до `localStorage`, для збереження стану авторизації у браузері;
- `logout()` – функція, для виходу з облікового запису з системи.

3.4.2 Функції серверної частини (back-end)

Функції, що викликаються під час взаємодії користувача із серверною частиною:

- `registration()` – функція, що отримує запит на створення нового користувача до системи;
- `login()` – функція, що перевіряє, чи існує такий користувач у базі даних та видає токен для доступу до системи;
- `auth()` – функція, що видає обліковий запис з актуальним токеном;
- `createDir()` – функція, що створює нову папку у директорії `files` та містить `id` користувача, якому приналежить цей файл;
- `uploadFile()` – функція, що додає файл до системи ;
- `uploadAvatar()` – функція, що завантажує новий аватар користувача у системі;

- `getFiles()` - функція, що відправляє усі файли авторизованого користувача у систему;
- `downloadFile()` - функція, що надає доступ на завантаження файлів з системи до пристрою;
- `searchFile()` - функція, що відправляє файли авторизованого користувача, залежно від пошукового запиту з клієнтської частини;
- `deleteFile()` - функція, що видаляє файл користувача з системи;
- `deleteAvatar()` - функція, що видаляє поточний аватар користувача з системи.

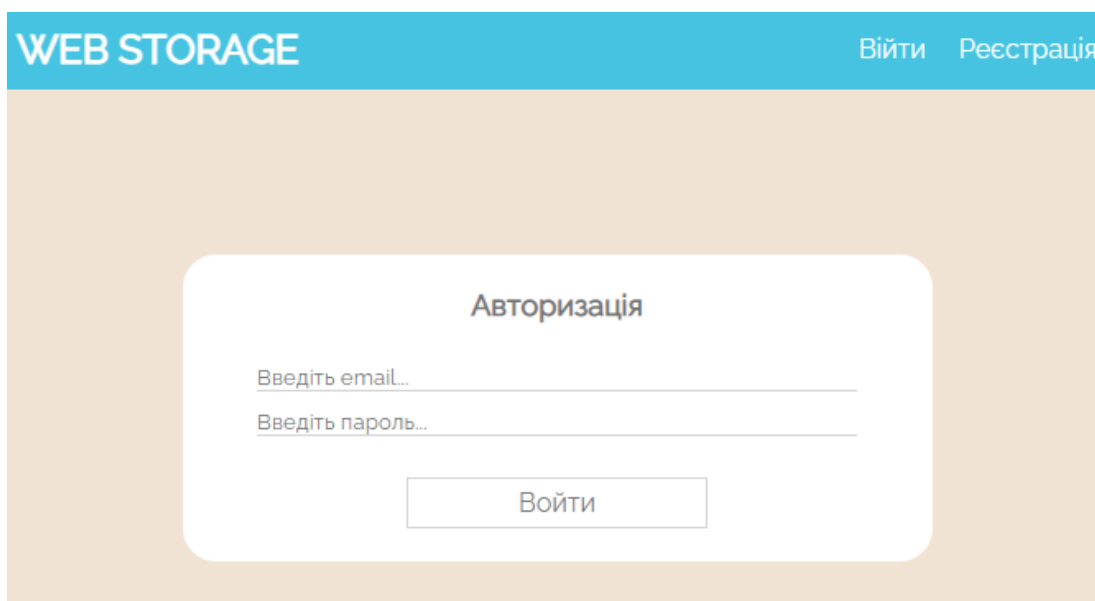
3.5 Висновки до розділу

В даному розділ ми розробили схему для бази даних проєктованого програмного забезпечення. Було прийнято рішення що до складової проєкту, створили зручну структуру для нашого вебзастосунку. Навели основні функції, які використовуються на клієнтській та серверній частині додатку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Виконання програми

Якщо користувач неавторизований у системі, при вході на сторінку веб-додатку, його автоматично перенаправляє на сторінку авторизації (рис. 4.1).



The image shows a web application interface for 'WEB STORAGE'. At the top, there is a blue header bar containing the text 'WEB STORAGE' on the left and two links, 'Війти' (Login) and 'Реєстрація' (Registration), on the right. Below the header, the background is a light beige color. In the center, there is a white rounded rectangle with the title 'Авторизація' (Authorization) at the top. Inside this rectangle, there are two input fields: the first is labeled 'Введіть email...' and the second is labeled 'Введіть пароль...'. Below these fields is a button labeled 'Войти' (Login).

Рисунок 4.1 – Сторінка для авторизації

Для користування системою необхідно бути зареєстрованим. Якщо користувач ще не має облікового запису у системі, він може перейти на форму реєстрації (рис. 4.2) та створити нового користувача на сайті. Для цього йому треба обрати розділ реєстрація у правому верхньому куті «шапки» застосунку.

Рисунок 4.2 – Сторінка для реєстрація

Після входу користувача до системи, на сервері генерується токен та записується у локальне сховище браузера. Кожен час токен відновлюється на сервері, таким чином ми гарантуємо користувачу захист його облікового запису у випадку, якщо про токен дізнається злоумисник. Переконайтесь у тому, що токен дійшов до системи, ми можемо завдяки панелі розробника у браузері Google Chrome, обравши розділ «Application» (рис. 4.3).

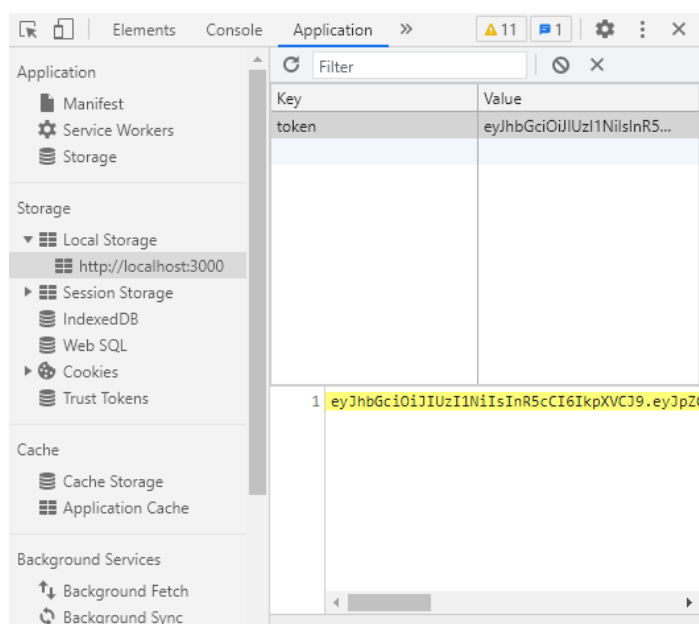


Рисунок 4.3 – Місце збереження токена

Після вдалої авторизації, на головній сторінці сайту для користувача з'являється сховище даних (рис. 4.4). Користувачу спливає список усіх файлів, що були додані саме з цього облікового запису. Якщо файлів у списку дуже багато, ми можемо використовувати поле вводу пошуку. Також, на сайті є фільтрація списку: по імені, по типу та по даті.

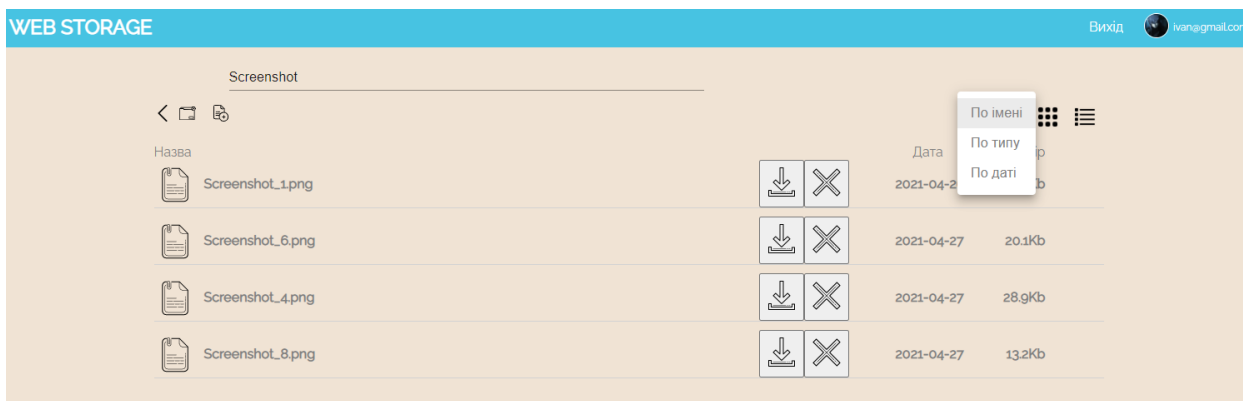


Рисунок 4.4 – Головна сторінка авторизованого користувача

Користувач може змінити для зручності вигляд списку файлів (рис. 4.5). Обирати вид можна завдяки іконкам в правому куті з двома режимами: у вигляді звичайного списку та з розширеними іконками файлів.

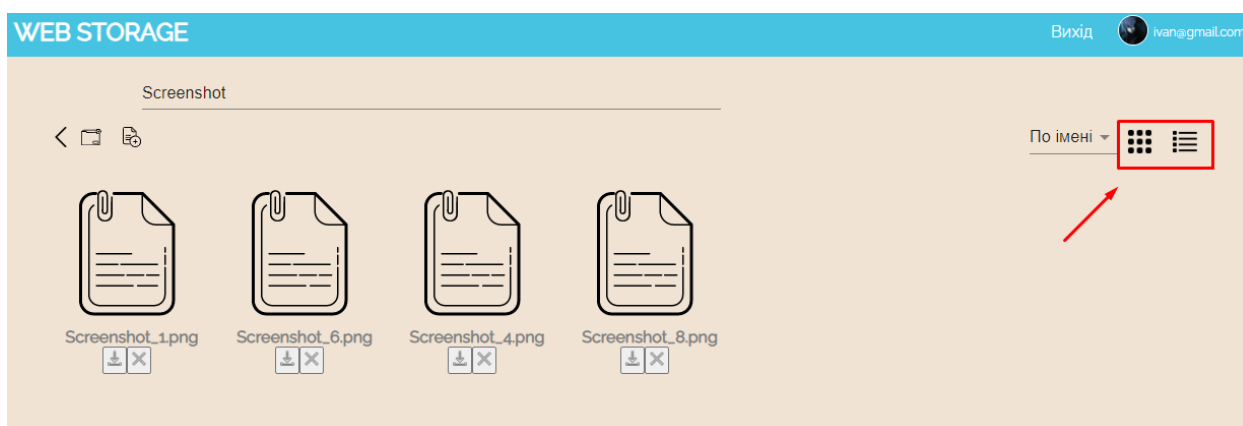


Рисунок 4.5 – Зміна вигляду файлів

Для створення більш складної структури сховища файлів було додано функцію створювання папок (рис. 4.6). Після натискання кнопки додавання

файлу, спливає модальне вікно, де треба вказати ім'я нової папки. Ім'я папки повинно бути унікальним.

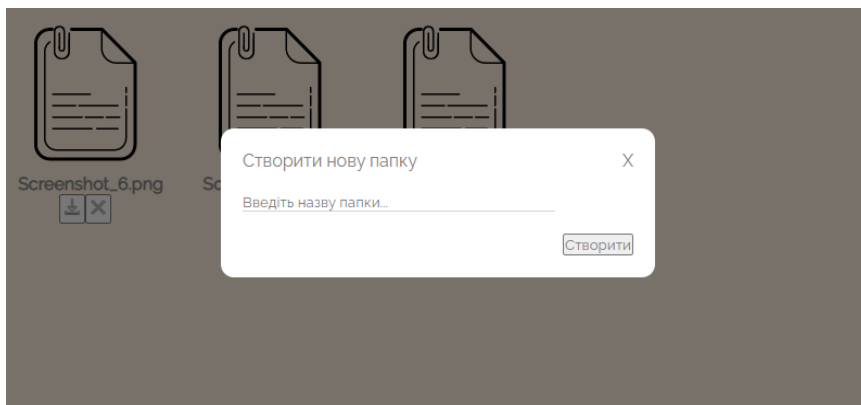


Рисунок 4.6 – Модальне вікно «Створити нову папку»

Для того щоб додати новий файл до сервісу, користувач повинен натиснути на кнопку додавання файлу, після чого з'являється вікно, де треба обрати файл на пристрої, який потім буде завантажено до веб-застосунку (рис. 4.7).

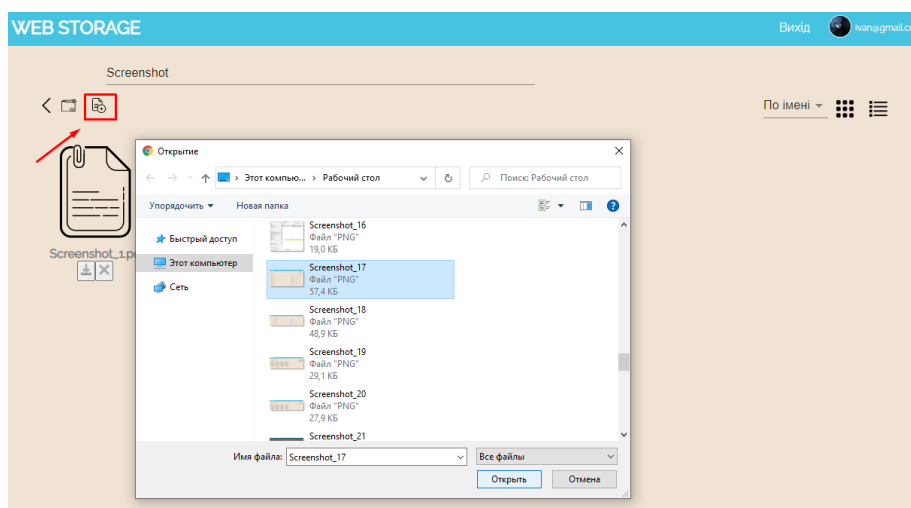


Рисунок 4.7 – Додавання файлу з пристрою до сервісу

Також, було реалізовано додавання файлу за допомогою технології Drag'n'Drop (рис. 4.8). Користувач може просто перетягнути іконку файлу з

пристрою до головної сторінки сховища у браузері та таким чином, файл буде додано до сервісу.

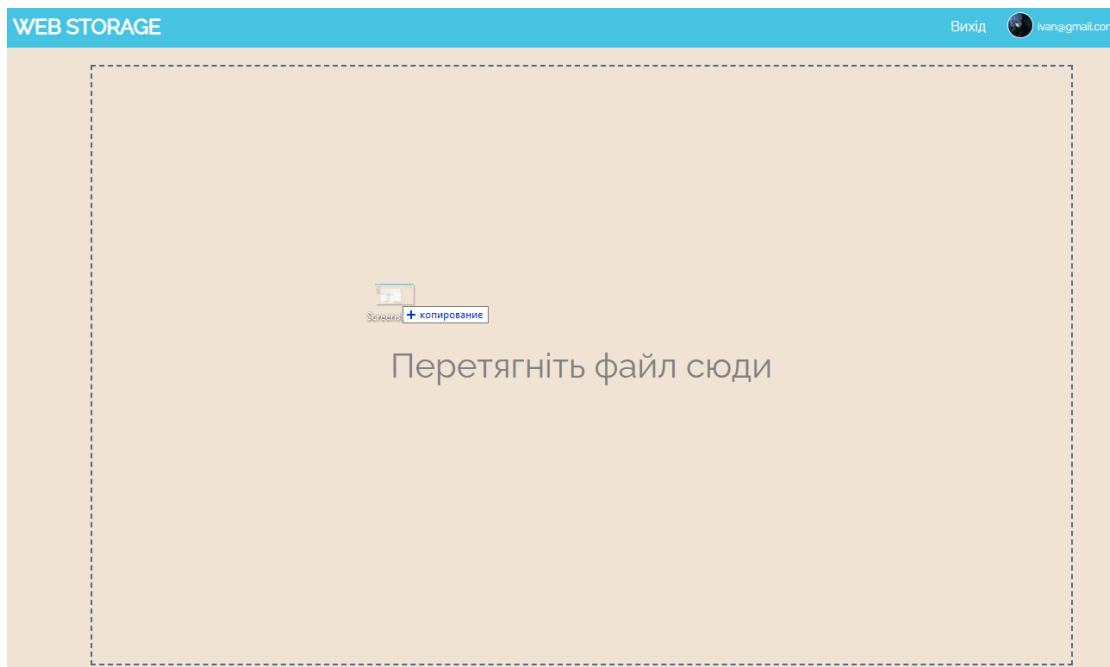


Рисунок 4.8 – Додавання файлу за допомогою технології Drag'n'Drop

Також, кожен користувач має можливість перегляду власного профілю на сервісі (рис. 4.9). На сторінці «Профіль», є інформація з доступним місцем пам'яті у додатку, а також інформація з вже використаним простором на диску веб-сховища. Змінною інформацією є відновлення аватару. Додано зручну форму додавання нового аватару до облікового запису, а також видалення нинішнього зображення.

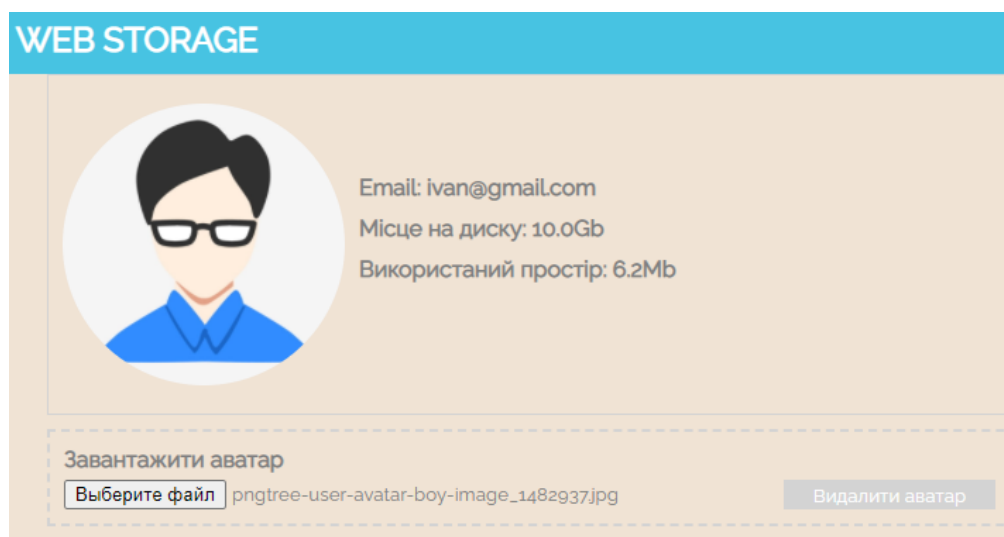


Рисунок 4.9 – Сторінка «Профіль»

4.2 Повідомлення користувачу

Для того щоб користувач міг вводити валідну інформацію, потрібно реалізувати появлення повідомлень, що відображаються при помилці введення даних.

Прикладом валідації у формі є форма реєстрації (рис. 4.10). При невалідних даних спливають такі повідомлення: «Поле обов'язкове», «Пароль повинен містити хоча б 1 літеру», «Пароль повинен містити хоча б 1 цифру».

Рисунок 4.10 – Форма реєстрації при введенні невалідних даних

Прикладом спливаючого повідомлення є повідомлення при додаванні файлу з неунікальним іменем (рис. 4.11).

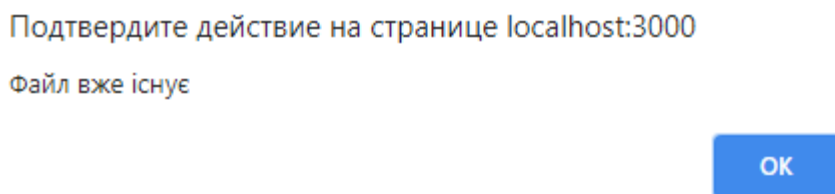


Рисунок 4.11 – Повідомлення про існуючий файл

4.3 Висновок до розділу

В даному розділі було продемонстрована взаємодія користувача з системою. Протестовані основні функції веб-додатку. Переконались в тому, що система попереджую користувачів у випадку вводу невалідної інформації.

ВИСНОВОК

В результаті виконання дипломної кваліфікаційної роботи було проведено аналіз предметної області та порівняння з існуючими подібним додатками. Були визначені переваги та недоліки найпопулярніших хмарних сховищ. На підставі порівнянь різних систем сформульовані функціональні вимоги. У вигляді діаграм були представлені вимоги до ПЗ.

Для реалізації ПЗ були обрані основні інструменти веб-розробника HTML/CSS та мова програмування JavaScript. Для серверної частини було обрано платформу NodeJS, для створення клієнтської частини вибір припав до бібліотеки React та Redux. Середою розробки програми було обрано Visual Studio Code.

У розробленому програмному забезпеченні реалізовано завантаження файлу з пристрою, створення папок, сортування файлів, пошук по символам, змінення виду списку, тощо. ПЗ було протестовано та перевірено на відображення коректних повідомлень про помилку.

Завдання, яке було поставлено до дипломної роботи, було виконано до кінця.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Облачные системы хранения [Электрон. ресурс]. – Режим доступа: <https://itelon.ru/blog/oblachnye-sistemy-khraneniya>.
2. Вакалюк, Т.А. Хмарні технології в освіті : навчально-методичний посібник для студентів фізико-математичного факультету / Т.А. Вакалюк – Житомир: ЖДУ, 2016. – 72с.
3. Хмарні сервіси [Електрон. ресурс]. – Режим доступу: <https://www.pharmencyclopedia.com.ua/article/7857/xmarni-servisi>.
4. Рознай, О. І. Домашнє портативне хмарне сховище / О. І. Рознай, А. М. Воронін. – Київ : НАУ, 2020. – 74 с.
5. Хмарні технології [Електрон. ресурс]. – Режим доступу: http://wiki.kubg.edu.ua/Хмарні_технології.
6. Шевченко, М. Хмарний сервіс зберігання / М. Шевченко, С.В. Дядюн – Харків : ХНУМГ, 2014. – 127 с.
7. 10 лучших облачных хранилищ для бизнеса [Электрон. ресурс]. – Режим доступа: <https://www.unisender.com/ru/blog/idei/oblachnye-hranilishcha>.
8. Что такое HTML? Основы языка разметки гипертекста [Электрон. ресурс]. – Режим доступа: <https://www.hostinger.ru/rukovodstva/shto-takoe-html>.
9. Глоссарий – drozd.red [Электрон. ресурс]. – Режим доступа: <https://drozd.red/glossary>.
10. Погружение в React: Redux [Электрон. ресурс]. – Режим доступа: <http://prgssr.ru/development/pogruzhenie-v-react-redux.html>.
11. Введение в Redux [Электрон. ресурс]. – Режим доступа: <https://betacode.net/12141/introduction-to-redux>.
12. Виртуальный DOM и внутренние механизмы [Электрон. ресурс]. – Режим доступа: <https://learn-reactjs.ru/faq/virtual-dom-and-internals>.

13. Почему вам стоит выбрать Node.js для разработки серверных приложений [Электрон. ресурс]. – Режим доступа: <https://tproger.ru/articles/pochemu-vam-stoit-vybrat-node-js-dlja-razrabotki-servernyh-prilozhenij>.

14. Паттерны работы с MongoDB [Электрон. ресурс]. – Режим доступа: <https://proglib.io/p/vstraivanie-ili-ssylki-patterny-raboty-s-mongodb-2020-03-21>.

15. Visual Studio Code [Электрон. ресурс]. – Режим доступа: <https://losst.ru/kak-polzovatsya-visual-studio-code>.

ДОДАТОК А
Технічне завдання

Вступ

Вебсайт для зберігання файлів користувача може використовуватися для завантаження файлу до іншого пристрою, отримання додаткового простору.

А.1 Підстави для розробки

Підставою для розробки програмного забезпечення є завдання на дипломну кваліфікаційну роботу бакалавра на тему: «Розробка вебзастосунку для зберігання файлів користувачів». Затверджене наказом № 86 від 20 березня 2021 р. по Національному університету «Запорізька політехніка».

А.2 Призначення розробки

Функціональне призначення: програма дозволяє зберігати файли користувачів у веб-сховищі.

А.3 Вимоги до програми чи програмному виробу

А.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати:

- створення папки;
- завантаження нового додату;
- видалення файлу з диску;
- реєстрація користувача;
- збереження токена авторизації у браузері;
- зміна персональної інформації.

А.3.2 Вимоги до надійності

Надійне функціонування системи має забезпечуватися наступними факторами:

- безперебійне живлення апаратно-технічних засобів;
- доступ до мережі Інтернет;
- необхідний рівень кваліфікації обслуговуючого персоналу.

Також, система повинна мати можливість оперативного відновлення працездатності при збоях.

А.3.3 Умови експлуатації

Для забезпечення експлуатації програмного забезпечення необхідно встановити наступні програмні застосунки:

- NodeJS;
- npm;
- ОС: MacOS, Windows, Android, IOS;
- Браузер Google Chrome, Mozilla Firefox, Safari.

А.3.4 Вимоги до складу й параметрів технічних засобів

Мінімальні системні вимоги:

- ОЗУ: 3 ГБ;
- Браузер: Google Chrome, Safari, Mozilla Firefox, Opera;
- ОС: MacOS, Windows, Android, IOS;
- Система: кросплатформена;
- доступ до Інтернету.

A.3.5 Вимоги маркування й упакування

Програма може бути записана на оптичному, жорсткому магнітному диску або USB-носії. На упаковці або відповідному пристрої має бути вказано назву роботи “Розробка вебзастосунку для зберігання файлів користувачів”.

A.3.6 Вимоги транспортування й збереження

Вимоги до транспортування та збереження повністю співпадають з вимогами до транспортування та збереження DVD-дисків.

A.4 Стадії та етапи розробки

Для роботи над дипломною кваліфікаційною роботою бакалавра заплановано виконання наступних етапів:

- аналіз проблеми;
- розробка архітектури програми;
- розробка програми;
- експлуатація, тестування та експериментальне дослідження програми;
- оформлення документації.

A.5 Порядок контролю й приймання

Дипломна кваліфікаційна робота бакалавра повинна бути виконана у відповідності з календарним планом. Дипломна кваліфікаційна робота допускається до захисту керівником, нормконтролером, затверджується користувачем, і захищається в присутності членів екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б.1 Файл index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider} from "react-redux";
import App from './root/App';
import {store} from "././root/store/reducers";
```

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Б.2 Файл calculateSize.js

```
let result;
let calNum;
let calNumToFixed;

const sizeCalculator = (occupiedMemory) => {
  if(occupiedMemory > 1073741824) {
    calNum = occupiedMemory/1073741824;
    calNumToFixed = calNum.toFixed(1)
    result = calNumToFixed+"Gb";
    return result;
  }
  if(occupiedMemory > 1048576) {
    calNum = occupiedMemory/1048576;
    calNumToFixed = calNum.toFixed(1);
    result = calNumToFixed+"Mb";
    return result;
  }
  if(occupiedMemory > 1024) {
    calNum = occupiedMemory/1024;
    calNumToFixed = calNum.toFixed(1);
    result = calNumToFixed+"Kb";
    return result;
  }
  return occupiedMemory +"B";
}
```

```
export default sizeCalculator;
```

Б.3 Файл Input.jsx

```
import React from 'react';
import styled from 'styled-components';

const StyledInput = styled.input`
  margin: 5px auto;
  width: 80%;
  color: black;
  border: none;
  outline: none;
  border-bottom: 1px solid lightgray;
  background-color: transparent;
`;

export default ({setValue, value, type, placeholder}) => {
  return (
    <StyledInput onChange={(e)=> setValue(e.target.value)}
      type={type}
      placeholder={placeholder}
      value={value}
    />
  );
};
```

Б.4 Файл App.jsx

```
import React, {useEffect} from 'react';
import {BrowserRouter} from "react-router-dom";
import {useDispatch} from "react-redux";
import NavComponent from "../root/Navbar/Navbar";
import ContentComponent from "../Wrapper/Wrapper";
import {auth} from "../modules/Auth/services/userService";
import styled from 'styled-components';
import './style.scss'

const App = () => {
  const dispatch = useDispatch()

  const StyledApp = styled.div`
    display: flex;
    flex-direction: column;
    align-items: center;
```

```

`;

useEffect(() => {
  dispatch(auth())
}, [])

return (
  <BrowserRouter>
    <StyledApp>
      <NavComponent/>
      <ContentComponent/>
    </StyledApp>
  </BrowserRouter>
);
}

export default App;

```

Б.5 Файл Wrapper.jsx

```

import React from 'react';
import SignInForm from "../../modules/Auth/components/SignIn";
import StorageComponent from "../../modules/Storage/components/Storage";
import ProfileComponent from "../../modules/Profile/components/Profile";
import styled from 'styled-components';
import {Redirect, Route, Switch} from "react-router-dom";
import {useSelector} from "react-redux";
import SignUpForm from "../../modules/Auth/components/SignUp";
import SignInForm from "../../modules/Auth/components/SignIn";
import StorageComponent from "../../modules/Storage/components/Storage";
import ProfileComponent from "../../modules/Profile/components/Profile";
import styled from 'styled-components';
import './style.scss';

const StyledWrapper = styled.div`
  display: flex;
  flex-direction: column;
  width: 100%;
`;

export default () => {
  const {authUser} = useSelector(state => state.user);
  const guest = !authUser;

  const authRoutes = [
    {

```

```

    path: "/registration",
    component: SignUpForm,
  },
  {
    path: "/login",
    component: SignInForm,
  },
];

const serviceRoutes = [
  {
    path: "/profile",
    component: ProfileComponent,
  },
  {
    path: "/",
    component: StorageComponent,
  },
];

return (
  <StyledWrapper>
    {guest ?
      <Switch>
        {authRoutes.map((item)=>{
          <Route path={item.path} component={item.component}/>
        })}
        <Redirect to="/login"/>
      </Switch>
      :
      <Switch>
        {serviceRoutes.map((item)=>{
          <Route path={item.path} component={item.component}/>
        })}
        <Redirect to="/" />
      </Switch>
    }
  </StyledWrapper>
);
}

```

Б.6 Файл loaderReducer.js

```
import { loaderPanelInitialValues } from "../initialState"
```

```

import {
  CLOSE_LOADER_SPINNER,
  SHOW_LOADER_SPINNER } from "../constants/actionTypes"

const loaderReducer = (state = loaderPanelInitialValues, action) => {
  switch (action.type) {
    case SHOW_LOADER_SPINNER:
      return {
        ...state,
        loader_spinner: true
      }
    case CLOSE_LOADER_SPINNER:
      return {
        ...state,
        loader_spinner: false
      }
    default:
      return state
  }
}

export default loaderReducer

```

Б.7 Файл root/reducers/index.js

```

import {applyMiddleware, combineReducers, createStore} from "redux";
import {composeWithDevTools } from 'redux-devtools-extension'
import thunk from "redux-thunk";
import userReducer from "../../modules/Auth/store/reducers/userReducer";
import fileReducer from "../../modules/Storage/store/reducers/fileReducer";
import uploadReducer from "../../modules/Storage/store/reducers/uploadReducer";
import loaderReducer from "../loaderReducer";

const rootReducer = combineReducers({
  user: userReducer,
  files: fileReducer,
  upload: uploadReducer,
  loader: loaderReducer
})

export const store = createStore(rootReducer, composeWithDevTools(applyMiddleware(thunk)))

```

Б.8 Файл loaderPanelActions.js

```
import {
  SHOW_LOADER_SPINNER,
  CLOSE_LOADER_SPINNER } from "../constants/actionTypes"

export const showLoaderSpinner = () => ({
  type: SHOW_LOADER_SPINNER
})

export const closeLoaderSpinner = () => ({
  type: CLOSE_LOADER_SPINNER
})
```

B.9 Navbar.jsx

```
import React from 'react';
import {NavLink} from "react-router-dom";
import {useDispatch, useSelector} from "react-redux";
import {logout} from "../modules/Auth/store/actions/userActions";
import userDefaultAvatar from '../assets/img/avatar.svg'
import {URL_SERVER} from "../config";
import styled from 'styled-components';
import './style.scss';

const NavbarWrapper = styled.div`
  display: flex;
  justify-content: center;
  background-color: #47c3e2;
  width: 100%;
  align-items: center;
`;

const StyledHeader = styled.div`
  margin: 10px;
  font-size: 24px;
  font-weight: 700;
`;

const StyledAvatar = styled.img`
  width: 30px;
  height: 30px;
  margin: 5px;
  border-radius: 50%;
  border: 1px solid #ffffff;
`;

const StyledEmail = styled.p`
```

```

    font-size: 12px;
    margin-right: 10px;
    color: white;
    text-decoration: 'none';
`;

const StyledLoginNavbar = styled.div`
  margin-left: auto;
  margin-right: 19px;
  cursor: pointer;
`;

const StyledNavbarRegistration = styled.div`
  margin-left: auto;
  margin-right: 19px;
  cursor: pointer;
`;

const StyledNavbarLogout = styled.div`
  margin-left: auto;
  margin-right: 19px;
  cursor: pointer;
`;

const Navbar = () => {
  const AuthorizedUser = useSelector(state => state.user.isAuth)
  const currentAuthorizedUser = useSelector(state => state.user.currentUser)
  const dispatch = useDispatch()

  const handleLogout = () => {
    dispatch(logout());
  }

  return (
    <NavbarWrapper>
      <StyledHeader>
        <NavLink style={{textDecoration: 'none'}} to="/" style={{color: 'white', textDecoration:
'none'}}>
          WEB STORAGE
        </NavLink>
      </StyledHeader>

      <!AuthorizedUser &&
        <StyledLoginNavbar>
          <NavLink to="/login" style={{color: 'white', textDecoration: 'none'}}>Війти</NavL
ink>
        </StyledLoginNavbar>
      </!AuthorizedUser &&
    </NavbarWrapper>
  )
}

```

```

    {!AuthorizedUser &&
      <StyledNavbarRegistration>
        <NavLink to="/registration" style={{color: 'white', textDecoration: 'none', marginRi
ght: '10px'}}>
          Реєстрація
        </NavLink>
      </StyledNavbarRegistration>
    }

    {AuthorizedUser &&
      <StyledNavbarLogout onClick={handleLogout} style={{color: 'white', textDecoration:
'none'}}>
        Вихід
      </StyledNavbarLogout>
    }

    {AuthorizedUser &&
      <NavLink to="/profile">
        <StyledAvatar src={currentAuthorizedUser.avatar ? `${URL_SERVER + currentAu
thorizedUser.avatar}` : userDefaultAvatar} alt=""/>
      </NavLink>
    }

    {isAuth &&
      <StyledEmail>{currentAuthorizedUser.email}</StyledEmail>
    }
  </NavbarWrapper>
);
};

export default Navbar;

```

Б.10 uploadReducer.js

```

import {
  ON_UPLOAD,
  OFF_UPLOAD,
  UPLOAD_NEW_DATA,
  CLEAN_UPLOAD_BAR } from "../constants/actionTypes"
import { UploadInitialValues } from "../initialState"

export default (state = UploadInitialValues, action) => {
  switch (action.type) {

    case CLEAN_UPLOAD_BAR:
      return {
        ...state,
        files: [...state.files.filter(file => file.id !== action.payload)]
      }
    }
  }

```

```

    ]]
  case UPLOAD_NEW_DATA:
    return {
      ...state,
      files: state.files.push(action.payload)
    }
  case ON_UPLOAD:
    return {
      ...state,
      isActive: true
    }
  case OFF_UPLOAD:
    return {
      ...state,
      isActive: false
    }
  default:
    return state
}
}

```

B.11 fileReducer.js

```

import { FileInitialValues } from "../initialState"
import {
  PUT_NEW_FILES,
  CURRENT_FILE,
  POST_NEW_FILE,
  SHOW_POPUP_MODAL,
  POST_ON_DIRECTORY,
  DELETE_CURRENT_FILE,
  VIEW_CHANGE } from "../constants/actionTypes"

export default function fileReducer(state = FileInitialValues, action) {
  switch (action.type) {

    case POST_ON_DIRECTORY:
      return {
        ...state,
        directory: state.directory.push(action.payload)
      }
    case CURRENT_FILE:
      return {
        ...state,
        currentFile: action.payload
      }

```

```

    case PUT_NEW_FILES:
      return {
        ...state,
        files: action.payload
      }
    case POST_NEW_FILE:
      return {
        ...state,
        files: state.file.push(action.payload)
      }
    case VIEW_CHANGE:
      return {
        ...state,
        view: action.payload
      }
    case SHOW_POPUP_MODAL:
      return {
        ...state,
        popupDisplay: action.payload
      }
    case DELETE_CURRENT_FILE:
      return {
        ...state,
        files: [...state.files.filter(file => file._id !== action.payload)]
      }
    default:
      return state
  }
}

```

B.12 uploadActions.js

```

import {
  ON_UPLOAD,
  OFF_UPLOAD,
  UPLOAD_NEW_DATA,
  CLEAN_UPLOAD_BAR } from "../constants/actionTypes"

export const showUploader = () => ({
  type: ON_UPLOAD
})
export const hideUploader = () => ({
  type: OFF_UPLOAD
})
export const addUploadFile = (data) => ({
  type: UPLOAD_NEW_DATA,

```

```

    payload: data
  })
export const removeUploadFile = (data) => ({
  type: CLEAN_UPLOAD_BAR,
  payload: data
})

```

B.13 fileActions.js

```

import {
  PUT_NEW_FILES,
  CURRENT_FILE,
  POST_NEW_FILE,
  SHOW_POPUP_MODAL,
  POST_ON_DIRECTORY,
  DELETE_CURRENT_FILE,
  VIEW_CHANGE } from "../constants/actionTypes"

export const setCurrentDir = (data) => ({
  type: CURRENT_FILE,
  payload: data
})
export const addFile = (data) => ({
  type: POST_NEW_FILE, payload: data
})
export const showPopup = (data) => ({
  type: SHOW_POPUP_MODAL,
  payload: data
})
export const deleteFileAction = (data) => ({
  type: DELETE_CURRENT_FILE,
  payload: data
})
export const setFileView = (data) => ({
  type: VIEW_CHANGE,
  payload: data
})
export const pushToStack = (data) => ({
  type: POST_ON_DIRECTORY,
  payload: data
})
export const setFiles = (data) => ({
  type: PUT_NEW_FILES,
  payload: data
})

```

B.14 Storage.jsx

```
import React, {useEffect, useState} from 'react';
import {useDispatch, useSelector} from "react-redux";
import FileList from "../FileList/FileList";
import { Input, Select, MenuItem } from '@material-ui/core';
import back from "../../assets/img/left-arrow.svg";
import add_file_icon from "../../assets/img/add-file.svg";
import add_folder from "../../assets/img/folder.svg";
import styled from 'styled-components';
import './style.scss'
```

```
const Storage = () => {
  const dispatch = useDispatch()
  const {currentDir} = useSelector(state => state.files)
  const {isAuth} = useSelector(state => state.user)
  const {loader_spinner} = useSelector(state => state.loader)
  const [searchValue, setSearchValue] = useState("")
  const [searchTimeout, setSearchTimeout] = useState(false)
  const dirStack = useSelector(state => state.files.dirStack)
  const [dragEnter, setDragEnter] = useState(false)
  const [sort, setSort] = useState('type')
```

```
  useEffect(() => {
    dispatch(getUserFiles(currentDir, sort))
  }, [dispatch,currentDir, sort])
```

```
  const StyledInput = styled.input`
    width: 30%;
    margin-bottom: 10px;
  `;
```

```
  const StyledSelect = styled.select`
    color: gray;
  `;
```

```
  export const StyledLoader = styled.div`
    display: flex;
    justify-content: center;
    align-items: center;
  `;
```

```
  export const StyledLoaderRing = styled.div`
    display: inline-block;
    width: 80px;
    height: 80px;
```

```

`;

export const StyledStorage = styled.div`
  margin-top: 20px;
`;

export const StyledBtnsBar = styled.div`
  display: flex;
`;

export const StyledDragDrop = styled.div`
  width: 100%;
  height: calc(100vh - 50px - 40px);
  margin: 20px;
  border: dashed #566885 2px;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 40px;
`;

if(loader_spinner) return <StyledLoader><StyledLoaderRing/></StyledLoader>;

const viewBtnsList = [
  {
    class: 'grid'
  },
  {
    class: 'list'
  }
]

const btnsList = [
  {
    src: back,
    handler: handleBack
  },
  {
    src: add_folder,
    handler: handlePopupOpen
  }
]

return ( !dragEnter ?
  <StyledStorage onDragEnter={dragEnterHandler} onDragLeave={dragLeaveHandler} onDrag
  Over={dragEnterHandler}>
    {isAuth && <Input
      value={searchValue}
      onChange={handleSearching}

```

```

        placeholder="Пошук файлу..." /> }
    <StyledBtnsBar>
        {btnsList.map(item =>
            <img src={item.src} onClick={item.handler} />
        )}
        <div className="storage__upload">
            <label htmlFor="storage__upload-input" className="storage__upload-label">
                <img src={add_file_icon} style={{marginTop: '7px'}} alt='img' className="storage__upload-label-img"></img>
            </label>
            <input onChange={fileUploadHandler} type="file" id="storage__upload-input" className="storage__upload-input" />
        </div>
        <Select value={sort}
            onChange={(e) => setSort(e.target.value)}
            className='storage-select'>
            <MenuItem value="name">По імені</MenuItem>
            <MenuItem value="type">По типу</MenuItem>
            <MenuItem value="date">По даті</MenuItem>
        </Select>
        {viewBtnsList.map(item =>
            <button className={item.class} style={{marginTop: '9px'}} onClick={() => dispatch(setFileView(item.class))} />
        )}
    </StyledBtnsBar>
    <FileList />
    <PopupComponent />
    <Uploader />
</StyledStorage>
:
    <StyledDragDrop onDrop={dropHandler} onDragEnter={dragEnterHandler} onDragLeave={dragLeaveHandler} onDragOver={dragEnterHandler}>
        Перетягніть файл сюди
    </StyledDragDrop>
);
};

export default Storage;

```

Б.15 Popup.jsx

```

import React, {useState} from 'react';
import Input from "../../utils/input/Input";
import {useDispatch, useSelector} from "react-redux";
import {showPopup} from "../../store/actions/fileActions";

```

```

import {createNewFolder} from "../services/fileService";
import {StyledClose,StyledTitle,StyledPopupHeader,StyledPopupContent,StyledPopupBtn,PopupSpace} from "./popup-style";

const PopupWrapper = styled(PopupSpace)`
  display: ${({popupDisplay})=>popupDisplay};
  justify-content: center;
  width: 100%;
  position: absolute;
  align-items: center;
`;

const Popup = () => {
  const dispatch = useDispatch();
  const [filename, setFilename] = useState("");
  const {popupDisplay, currentDir} = useSelector(state => state.files);

  const handleCreate = () => {
    dispatch(createNewFolder(currentDir, filename));
  }

  const handlePopup = () => {
    dispatch(showPopup('none'))
  }

  const handlePopupContent = (e) => {
    stopPropagation()
  }

  const handleClosePopup = (e) => {
    dispatch(showPopup('none'))
  }

  return (
    <PopupWrapper onClick={handlePopup} style={{display: popupDisplay}} displayPopup={popupDisplay}>
      <StyledPopupContent onClick={{e => handlePopupContent(e)}}>
        <StyledPopupHeader>
          <StyledTitle>Створити нову папку</StyledTitle>
          <StyledClose onClick={handleClosePopup}>X</StyledClose>
        </StyledPopupHeader>
        <Input type="text" placeholder="Введіть назву папки..." value={filename} setValue={setFilename}/>
        <StyledPopupBtn onClick={handleCreate}>Створити</StyledPopupBtn>
      </StyledPopupContent>
    </PopupWrapper>
  );
}

```

```
};
```

```
export default Popup;
```

Б.16 FileList.jsx

```
import React from 'react';
import './style.scss'
import {useSelector} from "react-redux";
import File from "./File/File";
import {CSSTransition, TransitionGroup} from "react-transition-group";
import {
  StyledGridView,
  StyledListView,
  StyledListViewDate,
  StyledListViewHeader,
  StyledListViewInfo,
  StyledListViewName,
  StyledListViewSize,
  StyledLoader } from './file-list-style';

const FileList = () => {

  const {files, view} = useSelector(state => state.files)

  if (files.length === 0) {
    return <StyledLoader>Файл не было найдено</StyledLoader>
  }

  switch(view) {
    case 'grid':
      return (
        <StyledGridView>
          {files.map(file =>
            <File key={file._id} file={file}/>
          )}
        </StyledGridView>
      )
    case 'list':
      return (
        <StyledListView>
          <StyledListViewHeader >
            <StyledListViewName>
              Назва
            </StyledListViewName>
            <StyledListViewInfo>
```

```

        <StyledListViewDate>Дата</StyledListViewDate>
        <StyledListViewSize>По3mip</StyledListViewSize>
      </StyledListViewInfo>
    </StyledListViewHeader >
    <TransitionGroup>
      { files.map(file =>
        <CSSTransition key={file._id} timeout={300} exit={false}>
          <File file={file}/>
        </CSSTransition>
      )}
    </TransitionGroup>
  </StyledListView>
);
default:
  return <></>;
}
};

export default FileList;

```

B.17 File.jsx

```

import React from 'react';
import './style.scss'
import folderSVG from '../assets/img/dir.svg'
import fileSVG from '../assets/img/file.svg'
import deleteSVG from '../assets/img/delete.svg'
import downloadSVG from '../assets/img/download.png'
import {useDispatch, useSelector} from "react-redux";
import {pushToStack, setCurrentDir} from "../store/actions/fileActions";
import {deleteCurrentFile, downloadCurrentFile} from "../services/fileService";
import calculateSize from "../utils/calculateSize";
import styled from 'styled-components';

const StyledFileWrapper = styled.div`
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  border-bottom: 1px solid lightgray;
`;

const StyledFileBox1 = styled.div`
  display: flex;
  align-items: center;
  cursor: pointer;
`;

const StyledFileName = styled.div`

```

```

    margin-left: 10px;
    color: gray;
`;
const StyledFileBox2 = styled.div`
  display: flex;
  align-items: center;
`;
const StyledFileBox3 = styled.div`
  display: flex;
  justify-content: space-between;
  width: 180px;
  margin-right: 70px;
`;
const StyledFileInfo = styled.div`
  display: flex;
  justify-content: space-between;
  align-items: center;
  width: 40%;
`;
const StyledFileData = styled.p`
  font-weight: 700;
  color: gray;
`;
const StyledFileSize = styled.p`
  font-weight: 700;
  color: gray;
`;

const StyledFileGrid = styled.div`
  width: 150px;
  height: 150px;
  cursor: pointer;
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-right: 20px;
  margin-bottom: 50px;
`;

const File = ({file}) => {
  const dispatch = useDispatch()
  const currentDir = useSelector(state => state.files.currentDir)
  const fileView = useSelector(state => state.files.view)

  function openDirHandler(file) {
    if(file.type === 'dir') {
      dispatch(pushToStack(currentDir))
      dispatch(setCurrentDir(file._id))
    }
  }

```

```

}

function downloadClickHandler(e) {
  e.stopPropagation()
  downloadCurrentFile(file)
}

function deleteClickHandler(e) {
  e.stopPropagation()
  dispatch(deleteCurrentFile(file))
}

if (fileView === 'list') {
  return (
    <StyledFileWrapper onClick={() => openDirHandler(file)}>
      <StyledFileBox1>
        <img width='50px' height='50px' src={file.type === 'dir' ? folderSVG : fileSVG} alt="
" className="file__img"/>
        <StyledFileName>{ file.name}</StyledFileName>
      </StyledFileBox1>
      <StyledFileInfo>
        <StyledFileBox2>
          {file.type !== 'dir' ?
            <button onClick={(e) => downloadClickHandler(e)} className="file__btn file__do
wnload">
              <img width='50px' src={ downloadSVG } alt="" className="file-plate__img"/>
            </button>
            :
            <div width='50px'/>
            <button onClick={(e) => deleteClickHandler(e)} className="file__btn file__delete
">
              <img width='50px' src={ deleteSVG } alt="" className="file-plate__img"/>
            </button>
          </StyledFileBox2>

          <StyledFileBox3>
            <StyledFileData>{ file.date.slice(0, 10)}</StyledFileData>
            <StyledFileSize>{ calculateSize(file.size)}</StyledFileSize>
          </StyledFileBox3>
        </StyledFileInfo>
      </StyledFileWrapper>
    );
  }
  if (fileView === 'grid') {
    return (
      <StyledFileGrid onClick={() => openDirHandler(file)}>
        <img width='150px' height='150px' src={file.type === 'dir' ? folderSVG : fileSVG} alt="
" className="file-plate__img"/>
        <StyledFileName>{ file.name}</StyledFileName>

```

```

    <div className="file-plate__btns">
      {file.type !== 'dir' &&
        <button onClick={ (e) => downloadClickHandler(e) } className="file-plate__btn file-plate__download">
          <img width='20px' src={ downloadSVG } alt="" className="file-plate__img"/>
        </button>}
      <button onClick={ (e) => deleteClickHandler(e) } className="file-plate__btn file-plate__delete">
        <img width='20px' src={ deleteSVG } alt="" className="file-plate__img"/>
      </button>
    </div>
  </StyledFileGrid>
);
}
};

export default File;

```

B.18 Profile.jsx

```

import React from 'react';
import {useDispatch, useSelector} from "react-redux";
import {deleteAvatar, uploadAvatar} from "../Auth/services/userService";
import avatarLogo from '../assets/img/avatar.svg'
import {API_URL} from "../config";
import calculateSize from "../utils/calculateSize";
import './style.scss';

const Profile = () => {
  const dispatch = useDispatch()
  const {email, diskSpace, usedSpace,} = useSelector(state => state.user.currentUser)
  const currentUser = useSelector(state => state.user.currentUser)
  const avatar = currentUser.avatar ? `${API_URL + currentUser.avatar}` : avatarLogo

  const uploadHandler = (event) => {
    const avatar = event.target.files[0]
    dispatch(uploadAvatar(avatar))
  }

  return (
    <div className = "Profile">
      <div className="profile-user">
        <img className="profile-avatar" src={avatar} alt=""/>
        <div className="profile-info">
          <div className="profile-email">
            Email: {email}

```

```

    </div>
    <div className="profile-disk-space">
      Місце на диску: {calculateSize(diskSpace)}
    </div>

    <div className="profile-used-space">
      Використаний простір: {calculateSize(usedSpace)}
    </div>
  </div>
</div>

<div className="profile-option-bar">
  <div className="profile-option-bar-1">
    Завантажити аватар
  </div>
  <div className="profile-option-bar-2">
    <input className = "profile-
upload" accept="image/*" onChange={e => uploadHandler(e)} type="file" placeholder="Завантаж
ити аватар"/>
    <button className = "profile-delete" onClick={() => dispatch(deleteAvatar())}>
      Видалити аватар
    </button>
  </div>
</div>

</div>
);
};

export default Profile;

```

Б.19 userReducer.jsx

```

import {
  LOGOUT,
  SET_USER } from "../constants/actionTypes"
import { UserInitialValues } from "../initialState"

export default (state = UserInitialValues, action) {
  switch (action.type) {
    case SET_USER:
      return {
        ...state,
        currentUser: action.payload,
        isAuth: true
      }
  }
}

```

```

    case LOGOUT:
      return {
        ...state,
        currentUser: {},
        isAuthenticated: false
      }
    default:
      return state
  }
}

```

B.20 userService.jsx

```

import axios from 'axios';
import {setUser} from "../store/actions/userActions";
import {URL_SERVER} from "../config";

const token = {headers:{Authorization:`Bearer ${localStorage.getItem('token')}`}}};

const handleSetToken = (res) => {
  localStorage.setItem('token', res.data.token);
};

const handleCleanToken = () => {
  localStorage.removeItem('token');
};

const convertFile = (file) => {
  const fileToForm = new FormData();
  fileToForm.append('file', file);
  return fileToForm;
}

export const uploadImg = (file) => {
  return async dispatch => {
    try {
      const form = convertFile(file);
      const res = await axios.post(`${URL_SERVER}api/files/avatar`, form, token);
      dispatch(setUser(res.data));
    } catch (e) {
      console.log(e);
    }
  }
};

export const deleteImg = () => {

```

```

return async dispatch => {
  try {
    const res = await axios.delete(`${URL_SERVER}api/files/avatar`, token);
    dispatch(setUser(res.data));
  } catch (e) {
    console.log(e);
  }
}
};

```

```

export const registration = async (email, password) => {
  try {
    let user = {
      email,
      password
    };
    await axios.post(`${URL_SERVER}api/auth/registration`, user);
  } catch (e) {
    console.log(e);
  }
};

```

```

export const login = (email, password) => {
  return async dispatch => {
    try {
      let user = {
        email,
        password
      };
      const res = await axios.post(`${URL_SERVER}api/auth/login`, user);
      dispatch(setUser(res.data.user));
      handleSetToken(res);
    } catch (e) {
      console.log(e);
    }
  }
};

```

```

export const auth = () => {
  return async dispatch => {
    try {
      const res = await axios.get(`${URL_SERVER}api/auth/auth`, token);
      dispatch(setUser(res.data.user));
      handleSetToken(res);
    } catch (e) {
      handleCleanToken();
    }
  }
};

```

Б.21 SignUp.jsx

```
import React from 'react';
import * as Yup from 'yup';
import {Formik, Form, Field, ErrorMessage} from 'formik';
import './style.scss'
import {registration} from "../services/userService";
import styled from 'styled-components';

const Wrapper = styled.div`
  display: flex;
  flex-direction: column;
  justify-content: center;
  background-color: white;
  width: 40%;
  margin: 100px auto;
  border-radius: 20px;
  text-align: center;
`;

const RegHeader = styled.div`
  text-align: center;
  margin: 20px;
  font-weight: 700;
  font-size: 18px;
  color: #757575;
`;

const required = "Поле обов'язкове";
const RegisterValidationSchema = () =>
  Yup.object({
    email: Yup.string().email().required(required).label('Email'),
    password: Yup.string().min(8)
      .matches(/^(?=.*[a-zA-Z]).+$/, 'Пароль повинен містити хоча б 1 літеру')
      .matches(/^\d/, 'Пароль повинен містити хоча б 1 цифру')
      .label('Пароль'),
  });

const SignUp = () => {
  let registrationInitialValues = {
    email: "",
```

```

    password: "
  }

  const submitHandle = (values) => {
    registration(values.email, values.password)
  }

  return (
    <Wrapper>
      <RegHeader>
        Реєстрація
      </RegHeader>
      <Formik
        initialValues={registrationInitialValues}
        validationSchema={RegisterValidationSchema()}
        onSubmit={submitHandle}>{() => (
          <Form className="form">

            <div className="mb-2">
              <Field
                name="email"
                placeholder="Введіть email..."
                type="email"
                className="input"
              />
              <ErrorMessage name="email" component="div" className="error-message"/>
            </div>

            <div className="mb-2">
              <Field
                name="password"
                placeholder="Введіть пароль..."
                type="password"
                className="input"
              />
              <ErrorMessage name="password" component="div" className="error-
message"/>
            </div>

            <button type="submit" className="submit">
              <span>Зареєструватись</span>
            </button>
          </Form>
        )}
      </Formik>
    </Wrapper>
  );
};

```

```
export default SignUp;
```

Б.22 SignIn.jsx

```
import React from 'react';
import {login} from "../services/userService";
import styled from 'styled-components';

const Wrapper = styled.div`
  display: flex;
  flex-direction: column;
  justify-content: center;
  background-color: white;
  width: 40%;
  margin: 100px auto;
  border-radius: 20px;
`;

const AuthHeader = styled.div`
  text-align: center;
  margin: 20px;
  font-weight: 700;
  font-size: 18px;
  color: #757575;
`;

const StyledButton = styled.div`
  text-align: center;
  margin: 20px auto;
  width: 40%;
  cursor: pointer;
  color: lightgray;
  border: 1px solid lightgray;
  color: #757575;
  padding: 5px;
`;

const required = "Поле обов'язкове";
const LoginValidationSchema = () =>
  Yup.object({
    email: Yup.string().email().required(required).label('Email'),
    password: Yup.string().min(8)
      .matches(/^(?=.*[a-zA-Z]).+$/, 'Пароль повинен містити хоча б 1 літеру')
      .matches(/^\d/, 'Пароль повинен містити хоча б 1 цифру')
      .label('Пароль'),
  });

const SignIn = () => {
  let loginInitialValues = {
    email: "",
  };

```

```

    password: "
  }

  const submitHandle = (values) => {
    login(values.email, values.password)
  }

  return (
    <Wrapper>
      <AuthHeader>
        Авторизація
      </AuthHeader>
      <Formik
        initialValues={loginInitialValues}
        validationSchema={LoginValidationSchema()}
        onSubmit={submitHandle}>{() => (
          <Form className="form">

            <div className="mb-2">
              <Field
                name="email"
                placeholder="Введіть email..."
                type="email"
                className="input"
              />
              <ErrorMessage name="email" component="div" className="error-message"/>
            </div>

            <div className="mb-2">
              <Field
                name="password"
                placeholder="Введіть пароль..."
                type="password"
                className="input"
              />
              <ErrorMessage name="password" component="div" className="error-
message"/>
            </div>

            <StyledButton type="submit" className="submit">
              <span>Війти</span>
            </StyledButton>
          </Form>
        )}
      </Formik>
    </Wrapper>
  );
};

```

```
export default SignIn;
```

B.23 server/index.js

```
import * as dotenv from 'dotenv';
dotenv.config();

import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { NestExpressApplication } from '@nestjs/platform-express';
import * as path from 'path';
import * as express from 'express';

async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>( AppModule, { cors: true } );
  app.useStaticAssets(path.join(__dirname, '../static'));
  app.use(express.static(path.join(__dirname, '../images')));
  await app.listen(process.env.PORT || 3000);
}

bootstrap();
```

B.24 file.controller.js

```
import { Injectable } from '@nestjs/common';
import config from require 'config'
import fileService from require './services/file.service'
import filesystem from require 'fs'
```

```
@Injectable()
export class FileController {

  async createDir(req, res) {
    try {
      const {data} = req.body
      const dir = new FileModel(data)
      await dir.save()
      return res.json(dir)
    } catch (e) {
      return res.json(e)
    }
  }
}
```

```

async getFiles(req, res) {
  try {
    const {sort} = req.query
    let files = await FileModel.find({user: req.user.id, parent: req.query.parent})
    return res.json(files)
  } catch (e) {
    return res.json({message: "Файл не отримано"})
  }
}

```

```

async uploadFile(req, res) {
  try {
    const file = req.files.file

    const user = await UserModel.findOne({_id: req.user.id})

    const dbFile = new FileModel(file);

    await dbFile.save()
    await user.save()

    res.json(dbFile)
  } catch (e) {
    return res.json({message: "Помилка завантаження"})
  }
}

```

```

async downloadFile(req, res) {
  try {
    const file = await FileModel.findOne({_id: req.query.id, user: req.user.id})
    const path = fileService.getPath(file)
    if (filesystem.existsSync(path)) {
      return res.download(path, file.name)
    }
    return res.status(400).json({message: "Помилка додавання"})
  } catch (e) {
    return res.json({message: "Помилка додавання"})
  }
}

```

```

async deleteFile(req, res) {
  try {
    const file = await FileModel.findOne({_id: req.query.id, user: req.user.id})
    fileService.deleteFile(file)
    await file.remove()
    return res.json({message: 'Файл було видалено'})
  } catch (e) {
    return res.json({message: 'Папка не пуста'})
  }
}

```

```

    }
  }

  async searchFile(req, res) {
    try {
      const searchName = req.query.search
      let files = await FileModel.find({user: req.user.id})
      files = files.filter(file => file.name.includes(searchName))
      return res.json(files)
    } catch (e) {
      return res.json({message: 'Пошук помилки'})
    }
  }

  async uploadAvatar(req, res) {
    try {
      const file = req.files.file
      const currentUser = await UserModel.findById(req.user.id)

      newAvatar = file.mv(`${process.env.PATH}\\${Uuid.v4()} .jpg`)
      currentUser.avatar = newAvatar
      await currentUser.save()
      return res.json(currentUser)
    } catch (e) {
      return res.json({message: 'Помилка відновлення аватару'})
    }
  }

  async deleteAvatar(req, res) {
    try {
      const currentUser = await UserModel.findById(req.user.id)
      currentUser.avatar = ""
      await currentUser.save()
      return res.json(currentUser)
    } catch (e) {
      return res.json({message: 'Помилка видалення зображення'})
    }
  }
}

```

Б.25 auth.middleware.js

```

import jwtToken from 'jsonwebtoken';

export default (req, res, next) => {
  switch(req.method){
    case 'OPTIONS': {

```

```

        return next();
    }
}
try {
    const token = req.headers.authorization.split(' ')[1]
    req.user = jwtToken.verify(token, ${process.env.JWT_KEY})
    next()
} catch (e) {
    console.log(e)
}
}

```

B.26 auth.controllers.js

```

import { Body, Controller, Post, Req } from '@nestjs/common';
import { registerSchema } from "register.validator";
import { YupOptions as ValidationFromYup } from "yup.validator";
import { IRegisterUser } from "../types";
import { RegisterService } from "../services/register.service";

@Controller('auth')
export class AuthController {
    constructor(
        private userService: UserService,
        private authService: AuthService,
        private registerService: RegisterService,
    ) {}

    @Post('registration')
    async registerCustomer(@Req() request, @Body() body: IRegisterUser){

        const data = await registerSchema.validate(body, ValidationFromYup);

        const user = await this.registerService.registerUser(request, data);

        return {user}
    }

    @Post('login')
    async login(@Body() body: ILoginUser){

        const payload = loginSchema.validateSync(body, ValidationFromYup);
        const user = await this.userService.findByEmail(payload.email);

        if (!user) {
            return {
                data: {

```

```

        message:new BadRequestException('User not found')
    }
}

const check = await HashService.check(body.password, user.password);

if(!check){
    return{
        data:{
            message:new BadRequestException('Password invalid')
        }
    }
}

const token = await this.authService.login( user.id } );

return {
    data: {
        token,
        user
    }
}
}

@Get('logout')
async logout(@Req() request) {
    try {
        await this.authService.logout( request['access_token']);
        return {message: 'Logout Success'};
    }
    catch (e) {
        return {message: 'Logout error'};
    }
}
}

```

B.27 file.routes.js

```

import { BadRequestException, Body, Controller, Get, Param, Patch, Post, Req } from '@nestjs/common';
import fileController from '../controllers/file.controller';
import { IFile } from "../types";

@Post('/')
async index(@Req() request, @Body() body: IFile) {
    return this.fileController.createDir()
}

```

```
}
```

```
@Post('/upload')
async index(@Req() request, @Body() body: IFile) {
  return this.fileController.uploadFile()
}
```

```
@Post('/avatar')
async index(@Req() request, @Body() body: IFile) {
  return this.fileController.uploadAvatar()
}
```

```
@Get('/')
async index(@Req() request) {
  return this.fileController.GetFiles()
}
```

```
@Get('/download')
async index(@Req() request) {
  return this.fileController.downloadFile()
}
```

```
@Get('/search')
async index(@Req() request) {
  return this.fileController.searchFile()
}
```

```
@Delete('/')
async index(@Req() request, @Body() body: IFile) {
  return this.fileController.deleteFile()
}
```

```
@Delete('/avatar')
async index(@Req() request, @Body() body: IFile) {
  return this.fileController.deleteAvatar()
}
```

B.28 models/File.js

```
import { UlidModel } from "../../lib/models/ulidModel";
import { ObjectId } from 'mongoose';

export class FileModel extends UlidModel {
  static get tableName() {
    return 'file';
  }

  name: string;
  type?: string;
  accessLink?: string;
  path: string;
  size: number;
  date: Date;
  user: ObjectId;
  parent: ObjectId;
  childs: Array<ObjectId>;
}
```

ДОДАТОК В
Слайди презентації

Національний університет “Запорізька Політехніка”

Кафедра програмних засобів

Дипломна кваліфікаційна робота

Розробка вебзастосунку для зберігання файлів користувачів

Виконав
ст. гр. КНТ-117

І.М. Нескоромний

Керівник
к.т.н., доц.

Т.В. Федорончак

Рисунок В.1 – Слайд 1

Об’єкт дослідження:

системи для зберігання файлів користувачів.

Предмет дослідження:

вебсайт, що надає користувачам можливість зберігати файли.

Мета роботи:

програмна реалізація сховища файлів користувачів у веб-просторі.

Рисунок В.2 – Слайд 2

Завдання до роботи:

- аналіз предметної області та порівняння з існуючими системами для збереження файлів
- розробка технічного завдання
- проектування структури системи
- розробка схеми функціонування
- розробка програмного продукту
- тестування та демонстрація програми

Рисунок В.3 – Слайд 3

Схема функцій користувачів

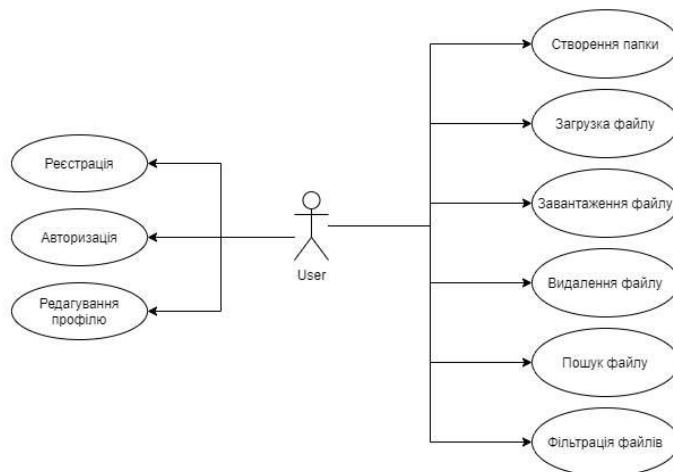


Рисунок В.4 – Слайд 4

Схема функціонування програми

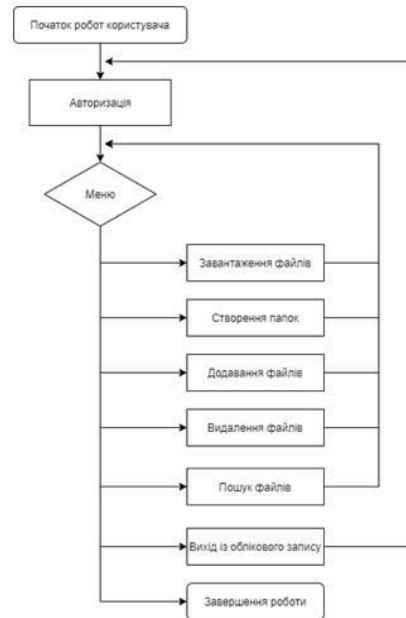


Рисунок В.5 – Слайд 5

Схема бази даних

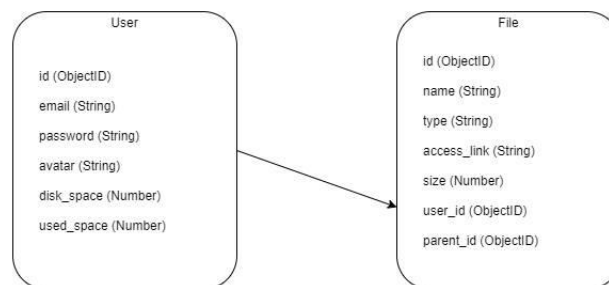


Рисунок В.6 – Слайд 6

Схема архітектури вебзастосунку

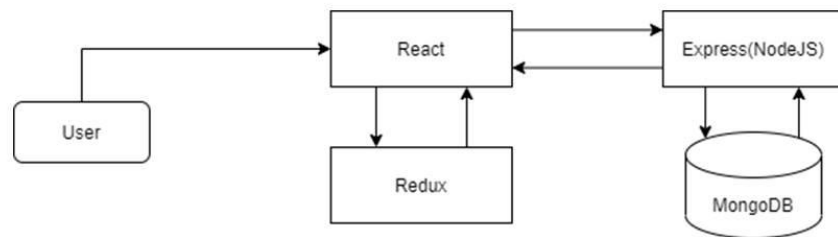


Рисунок В.7 – Слайд 7

Форма авторизації

The screenshot shows a web application interface for a login form. The header is blue and contains the text "WEB STORAGE" on the left and "Війти" (Login) and "Реєстрація" (Registration) on the right. The main content area is light beige. In the center, there is a white rounded rectangle containing the title "Авторизація" (Authorization). Below the title, there are two input fields: "Введіть email..." and "Введіть пароль..." (password). At the bottom of the white rectangle is a button labeled "Війти" (Login).

Рисунок В.8 – Слайд 8

Форма реєстрація

WEB STORAGE

Війти Реєстрація

Реєстрація

Введіть email...

Поле обов'язкове

.....

Пароль повинен містити хоча б 1 літеру

Зареєструватись

Рисунок В.9 – Слайд 9

Головна сторінка сховища

WEB STORAGE

Вийти  [myegymat.com](#)

Screenshot

<  

Нова

	Screenshot_1.png	 	2021-04-27	
	Screenshot_6.png	 	2021-04-27	20.1Kb
	Screenshot_4.png	 	2021-04-27	28.9Kb
	Screenshot_8.png	 	2021-04-27	13.2Kb

По імені

По типу

По дати

Рисунок В.10 – Слайд 10

Модальне вікно «Створити нову папку»

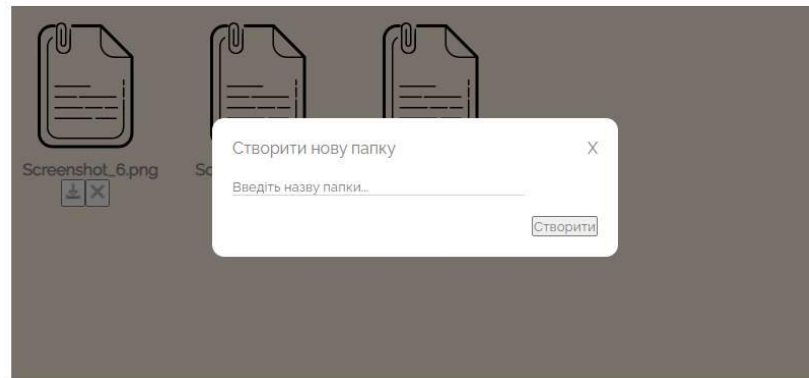


Рисунок В.11 – Слайд 11

Додавання файлу з пристрою до сервісу

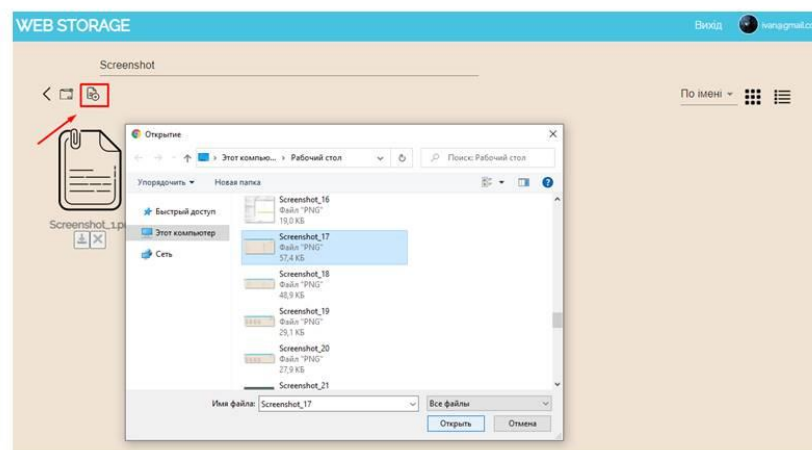


Рисунок В.12 – Слайд 12

Додавання файлу за допомогою технології Drag'n'Drop

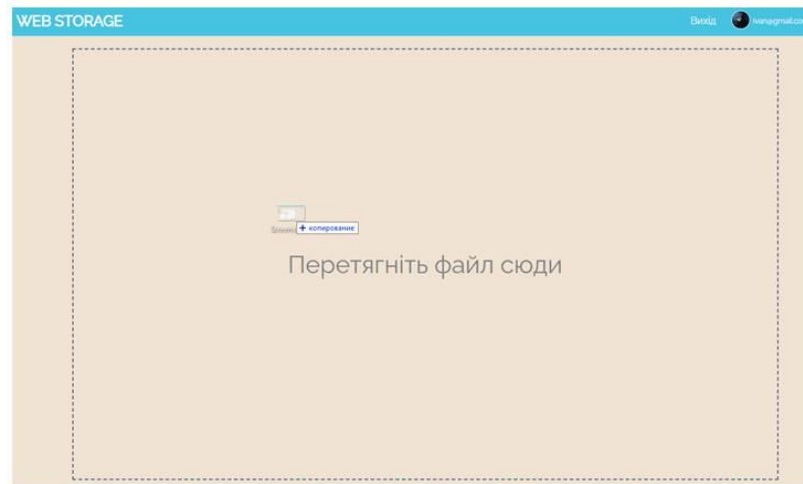


Рисунок В.13 – Слайд 13

Сторінка «Профіль»

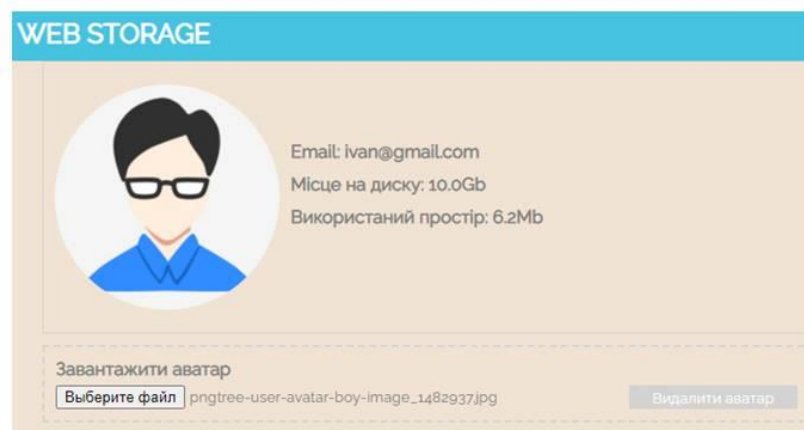


Рисунок В.14 – Слайд 14

Висновки

В результаті виконання дипломної кваліфікаційної роботи було проведено аналіз предметної області та порівняння з існуючими подібним додатками. Були визначені переваги та недоліки найпопулярніших хмарних сховищ. На підставі порівнянь різних систем сформульовані функціональні вимоги. У вигляді діаграм були представлені вимоги до ПЗ.

Для реалізації ПЗ були обрані основні інструменти веб-розробника HTML/CSS та мова програмування JavaScript. Для серверної частини було обрано платформу NodeJS, для створення клієнтської частини вибір припав до бібліотеки React та Redux. Середою розробки програми було обрано Visual Studio Code.

У розробленому програмному забезпеченні реалізовано завантаження файлу з пристрою, створення папок, сортування файлів, пошук по символам, змінення виду списку, тощо. ПЗ була протестована та перевірена на відображення коректних повідомлень про помилку.

Завдання, яке було поставлено до дипломної роботи, було виконано до кінця.

Рисунок В.15 – Слайд 15