**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___" _____2016 р.

# Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код та назва спеціальності)

на тему: «Клієнтська частина системи зберігання корпоративного контенту»
Виконав: студент IV курсу, групи групи ДА-21
(шифр групи)
Медведський Анрій Миколайович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент. к.т.н. Безносик О.Ю. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст.. викладач Бритов О.А. _____

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

Київ – 2016 рік

**Національний технічний університет України**

**«Київський політехнічний інститут»**

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський) (перший
(бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування
**7.050103, 8.050103 Системне проектування**
(код і назва)

**ЗАТВЕРДЖУЮ**

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____2016 р.

**ЗАВДАННЯ на дипломний проект (роботу) студенту**

**Медведському Андрію Миколайовичу**
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) ) Клієнтська частина системи зберігання корпоративного контенту.

керівник проекту (роботи) ) Безносик О.Ю. к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 2016 р. №_____

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)

Файл для завантаження
Назва файлу для видалення
Пошуковий запит

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Проаналізувати існуючі рішення. Спроектувати архітектуру.
2. Сформувати вимоги до інтерфейсу. Проаналізувати реалізацію клієнтської частини
3. Розробити інтерфейс.

4. Проаналізувати результати.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Архітектура системи – плакат.
2. Шаблон інтерфейсу - плакат.
3. Архітектура AngularJS – плакат.

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання 01.02.2016

Календарний план

| № з/п | Назва етапів виконання дипломного проекту (роботи) | Строк виконання етапів проекту (роботи) | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 01.02.2016 | |
| 2 | Збір інформації | 15.02.2016 | |
| 3 | Вивчення існуючих архітектрних рішень | 28.02.2016 | |
| 4 | Розробка структури системи | 10.03.2016 | |
| 5 | Розробка плану тестування | 15.03.2016 | |
| 6 | Розробка програмної моделі | 25.03.2016 | |
| 7 | Розробка опису | 25.04.2016 | |
| 8 | Тестування додатку та отримання результатів | 30.04.2016 | |
| 9 | Оформлення дипломної роботи | 31.05.2016 | |
| 10 | Отримання допуску до захисту та подача роботи в ДЕК | 08.06.2016 | |

Студент _____ Медведський А.М.
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Безносик О.Ю.

# АНОТАЦІЯ

бакалаврської дипломної роботи Медведського Андрія Миколайовича

на тему «Клієнтська частина системи

збереження корпоративного контенту»

Дана робота присвячена розробці системи зберігання корпоративного контенту, а саме інтерфейсу. Метою є розробка інтерфейсу, що відповідає стандартам, вимогам користувачів та задовільняє весь функціонал сховища.

В роботі розглянуто віддалені системи зберігання даних та їх інтерфейси. Проаналізовано переваги та недоліки кожного з них, створений список вимог. В роботі розглянута повна архітектура сховища, описані технології та інструменти, необхідні для створення інтерфейсу. Розроблена безспосередньо сама клієнтська частина та описано функціонал головних її частин.

Загальний обсяг роботи 77 с., 19 рис., 10 табл., 16 посилань.

**Ключові слова:** Web ui, AngularJS, SPA, Bootstrap, файлове сховище.

# АННОТАЦИЯ

бакалаврской дипломной работы Медведского Андрея Николаевича

на тему: «Клиентская часть системы хранения корпоративного контента»

Данная работа посвящена разработке системы хранения корпоративного контента, а именно интерфейса. Целью является разработка интерфейса, соответствующего стандартам, требованиям пользователей и удовлетворяет весь функционал хранилища.

В работе рассмотрено отдаленные системы хранения данных и их интрфейс. Проанализированы преимущества и недостатки каждого из них, созданный список требований. В работе рассмотрна полная архитектура хранилища, описаны технологии и инструменты, необходимые для создания интерфейса. Разработана непосредственно сама клиентская часть и описано функцониал главных ее частей.

Общий объем работы  77 с., 19 рис., 10 табл., 16 источников.

**Ключевые слова:** Web ui, AngularJS, SPA, Bootstrap, файловое хранилище.

# ABSTRACT

on Andrii Medvedskyi bachelor’s degree

thesis: “Client side of enterprise content storage system”

The given work is dedicated to the development of a system to store corporate content, in particular, an interface. The aim is to create an interface that corresponds to standards and user demands, and satisfies all functioning aspects of a storage system.

In this work, distance storage systems for data and their interfaces were reviewed. The advantages and disadvantages of each one of them were analyzed, and the list of requirements was compiled. In this work, the entire storage architecture was examined, and technologies and tools necessary for the creation of the interface were described. There was direct development of a client's part and its functioning of the main segments.

Bachelor's work size 77 p., 19 pic., 10 tables, 16 sources.

**Keywords:** We ui, AngularJS, SPA, Bootstrap, file storage.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

HTML – HyperText Markup Language

ПЗ – Програмне забезпечення

URI – Uniform Resource Identifier

URL - Uniform Resource Locator

HTTP – HyperText Transfer Protocol

API – Application programming interface

Web – система доступу до пов'язаних між собою документів на різних комп'ютерах, підключених до Інтернету

HTML – HyperText

Markup Language — Мова розмітки гіпертекстових документів

CSS – Cascading Style Sheets (Каскадні таблиці стилів)

SPA – (Single Page Application) Одностор інковий додаток

MVC – Model-view-Controller

UI - User Interface

UX - User Experience

# ВСТУП

Практика зберігання документів у сховищах широко поширилася за останні п'ять років. Підприємства переходять від традиційного персонального комп'ютера с кількома вбудованими жорсткими дисками до ноутбуків з невеликими ssd накопичувачами, вважаючи віддалене сховище пріоритетним.

З розповсюдженням онлайн файлообмінних рішень, поширення інформації стало простіше, ніж коли-небудь. І працівники почали користуватись такими сервісами для поширення корпоративних даних.

Збільшення використання мобільних пристроїв змінює вимоги способу організації доступе до корпоративних даних. Працівники прагнуть отримати доступ до файлів в будь-який час і з різних пристроїв. Вони не хочуть бути залежними від Інтернет з’єднання, об’єму доступної пам’яті чи навіть від працездатності зовнішньої системи.

Синхронізація та поширення файлів на основі хмарних сервісів стають все популярнішими. Ці послуги є простими для використання, але рівень контролю та безпеки даних компанії є неоднозначним.

Для підприємств будь-якого типу і розміру, еволюція робочих процесів висуває вимоги, які кидають виклик традиційним методам обміну файлами. Якщо співробітники не мають доступу до системи зберігання файлів з можливістю інтенсивного обміну інформацією, використовуючи будь-які пристрої, інформація буде переміщатися по неконтрольованим і потенційно небезпечним каналам зв'язку

Зростаючий попит на обмін знаннями та продуктивність означає, що підприємства без надійних і простих у використанні інструментів для спільної роботи може або поставити під загрозу безпеку даних або понести втрати в операційній ефективності. Тим паче, що результат може поставити ці підприємства в невигідне конкурентне положення. Підприємства повинні забезпечити необхідні інструменти для підвищення продуктивності, мобільності та спільної роботи.

Співробітники вимагають ефективного і зручного способу організації спільної роботи с файлами, і організації, які не реалізують цей функціонал внутрішньо будуть все частіше виявляти що конфіденційні дані передаються по каналах поза їхнім контролем - в той час, коли методи викрадення даних досягли безпрецедентного рівня складності.

Реалізуючи мобільні системи зберігання та спільного використання файлів, які використовують моделі розгортання на місці, підприємства можуть зберегти конфіденційну інформацію під наглядом без шкоди для продуктивності праці співробітників, ефективності роботи і конкурентної переваги.

Більшість співробітників підприємств зберігають корпоративні дані на своїх пристроях, або використовують сторонні хмарні сервіси. У першому випадку подібна практика може привести до втрати даних при апаратному збої, в другому дані можуть передаватися по незахищених каналах зв'язку, що може привести до витоку важливих даних. Крім того дані зберігаються в неструктурованому вигляді і розподілені по різних пристроїв що ускладнює доступ до них.

Вирішення цих проблем дозволить підприємствам збільшити контроль над даними і надати співробітникам зручний інструмент для зберігання і спільного доступу до даних. Всі наявні програмні продукти є дорогими або не володіють достатньою функціональністю для усунення всіх цих проблем.

# 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ЗБЕРІГАННЯ КОНТЕНТУ

Наше сьогодення кардинально відрізняється від того що було 10, 5 чи навіть 1 рік тому. Світ технологій повністю проник в наше життя. Більшість населення вже не уявляє себе без різного роду пристроїв, а вихід в Інтернет стало повсякденною потребою. Відповідно до такого розвитку, збільшився і об’єм використовуваних даних.

Щоб зрозуміти масштаби цього зростання, вважають, що за даними дослідження проведеного International Data Corporation (IDC) в 2013 році, по всьому світу встановлена ємність для зберігання збільшиться з 2,596 екзабайт (EB) в 2012 році до приголомшливих 7235 EB в 2017. IDC припустила, що, до 2020 року обсяг даних в світі досягне 35,840 екзабайт, що в розрахунку на людину становить близько 4 терабайт.

Екзабайт еквівалентний одній тисячі петабайт (ПБ), одному мільйону терабайт (ТБ) або одному мільярду гігабайт (ГБ). Важко провести якусь паралель для порівняння, однак середня книга займає близько 1 мегабайта. Найбільша бібліотека в світі, Бібліотека Конгресу, має близько 23 мільйонів томів, які становлять близько 23 ТБ даних. Тому тре було б взяти 43000 бібліотеки розміром Бібліотеки Конгресу, щоб сформувати екзабайт. Іншим прикладом можуть бути фотографії з високою роздільною здатністю, які займають приблизно 2 МБ; екзабайт буде становити майже 537 млрд. фотографій з високою роздільною здатністю. Ємність накопичувачів продовжує зростати, що вимагає нових міток для ще більшої кількості даних.

Після екзабайт, існує  зетабайт і йотабайт. У відповідь на це швидке зростання обсягу даних, поширювались і терміни. Деякі з них визначали, що ми просто використовуємо "hellabyte".

Виходячи з цього, можна стверджувати, що і в різного роду компаніях збільшилась кількість використовуваних даних. Коли це стосується саме файлів з кодом, то існує велика різноманітність систем, що дозволяють керувати написаними матеріалами та зберігати на віддалених сховищах. Прикладом таких систем є GIT - розподілена система керування версіями файлів та спільної роботи. Проект створив Лінус Торвальдс для управління розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів. В якості сховищ зачасту використовуються Github, один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Однак це не локальне сховище, прикладом саме локального сховища(базується на власному сервері в межах локальної мережі) може бути Gitlab, що й до всього є українським стартапом.

## 1.1 Системи зберігання даних

В компаніях існують різного роду дані. До таких можна віднести різні технічні завдання, договори, документація до систем. Зазвичай до цих даних

також потрібен доступ великому колу людей, в більшості випадків - всім працівникам компаній. Звісно, можна зберігати все на одному робочому комп'ютері і обмінювати файлами за допомогою електронною пошти, ssh чи в крайньому разі за допомогою Skype. Як вирішення цього завдання на допомогу прийшли різного роду віддалені сховища. Найпопулярніші серед них такі:

1. Dropbox — файлообмінник та синхронізатор файлів від компанії Dropbox Inc. Dropbox має кросплатформений клієнт (Windows, Mac і Linux), за допомогою якого користувачі можуть завантажити файли на сервер Dropbox. Власні файли на Dropbox можна зробити доступними для інших користувачів чи для всіх охочих.
2. Диск Google (англ. *Google Drive*) — сховище даних, яке належить компанії Google Inc., що дозволяє користувачам зберігати свої дані на серверах у хмарі і ділитися ними з іншими користувачами в Інтернеті. Google Drive включає Google Docs, Sheets, and Slides, офісний пакет, який дозволяє спільно редагувати документи, електронні таблиці, презентації, малюнки, форми, і багато іншого. Після активації замінює собою Документи Google.
3. Хмара Mail.Ru - хмарне сховище даних російської компанії Mail.Ru Group. Дозволяє зберігати музику, відео, зображення та інші файли в хмарі, а також ділитися ними з іншими користувачами Інтернету.

## 1.2 Локальні віддалені сховища

Однак використання зовнішніх віддалених сервісів в першу чергу не завжди надійно. Ці системи хоч і дуже захищені, але все ж знаходяться у

відкритому доступі в мережі. А так як корпоративний контент зазвичай інформація приватна, то втрата її може призвести до негативних наслідків. Не варто забувати й про те, що ці всі системи в безкоштовному варіанті мають обмежений об’єм доступного простору. При невеликій кількості даних, це припустимо, але вже при виході за межу в 15 ГБ це може поставати проблемою.

Головним недоліком використання таких систем є залежність. Це означає, що технічні роботи або відсутність Інтернет з’єднання будуть створювати незручності в доступі до спільних даних. Нині існує тенденція до ізоляції внутрішнього робочого простору, навіть написання своїх власних бібліотек, при існуючих схожих, стало поширеним явищем. Нині існує велика кількість сервісів, які вирішують ці проблеми. Найпопулярніші серед них:

1. ownCloud — система для організації зберігання, синхронізації й обміну даними, розміщеними на зовнішніх серверах. Спочатку проект розвивався спільнотою KDE, але згодом засновники проекту створили комерційну компанію ownCloud Inc, яка взяла в свої руки розробку ownCloud і розпочала надання платних сервісів та Enterprise-версії платформи. Для доступу до даних, збережених в ownCloud, можна використовувати веб-інтерфейс або протокол WebDAV. Додатково до зберігання даних можна відзначити функції підтримки засобів для забезпечення спільного доступу і можливість синхронізації між різними машинами таких даних, як адресна книга, календар-планувальник і закладки, з можливістю їхнього перегляду і редагування з будь-якого пристрою в будь-якій точці мережі. Сервер ownCloud можна розгорнути на будь-якому хостингу, який підтримує виконання PHP-скриптів і надає доступ до SQLite, MySQL або PostgreSQL.

2. Git-annex - сервіс синхронизації файлів направлений на розв’язання проблем загального доступу до файлів і синхронізації, але не залежить від будь-якої комерційної служби або центрального сервера. Він написаний на Haskell і доступний для Linux, Android, OS X і Windows. Git-annex зберігає лише посилання на файли в git репозиторії і керує ними в окремому місці. Він також створює дуплікати файлу для відновлення втраченої інформації. Слід зазначити, що git-annex доступний на різних Linux дистрибутивах, включаючи: Fedora, Ubuntu, Debian і т.д.
3. Seafile — відкрита платформа для створення сервісу хмарного зберігання даних. Крім базових функцій зберігання на віддаленому сервері і забезпечення синхронізації даних між комп'ютерами, Seafile надає гнучкі можливості з організації спільної роботи з контентом. Для зручності спільної роботи підтримується створення робочих областей, в яких члени групи можуть розміщувати довільну інформацію, цікаву для учасників групи. Набори файлів можуть об'єднуватися в бібліотеки, до яких може відкриватися доступ для окремих користувачів або груп, а також публічний доступ. Кожна бібліотека у сховищі представлена у формі, що нагадує Git-репозиторій. Ця особливість дає можливість використання версійного контролю, в тому числі підтримку доступу до минулих редакцій збереженого контенту, можливість відстежити всі внесені зміни (хто, коли і що міняв), повернути колишній стан файлу або відновити випадково вилучений файл. В основі Seafile лежать технології, застосовувані в системі управління сирцевими текстами Git. При цьому Seafile не залежить від Git і самостійно реалізує потрібні методи, які

спрощені і перероблені для виконання завдань автоматичної синхронізації даних, забезпечення поновлення передачі даних у випадку розриву з'єднання і підтримки різних бекендів зберігання на стороні сервера. Дані зберігаються з розбиттям на блоки, що підвищує ефективність зберігання і дає можливість прискорення передачі файлів за рахунок паралельного завантаження блоків з різних серверів зберігання.

## 1.3 Формування вимог до сховища

Локальні сховища вирішуються низку проблем, притаманних для віддалених сервісів. Приватні сховища уникають багатьох тверджень, що стосуються безпеки. Оскільки власні налаштування здійснюються безпечно всередині корпоративного брандмауера, приватне сховище забезпечує більший контроль над даними компанії, і це гарантує безпеку, хоча і з великим потенційним ризиком для втрати даних через стихійні лиха. Однак головною перевагою локальних ресурсів є уникнення залежності. Ця залежність може проявлятись в багатьох видах:

1. Інтернет з’єднання
2. Необхідність оплати послуг
3. Залежність від робочого стану сервісу
4. Інші

Приватні системи надають такий же стек функціоналу, що й зовнішні, однак володіють масою переваг. Головними вимогами до приватних сховищ є такі:

1. Стійкість - ступінь, в якій система зберігання може гарантувати, що дані ніколи не будуть бути безповоротно втрачені або пошкоджені, незважаючи на окремі невдачі компонентів (наприклад, як втрата диска або комп'ютера). Дані, ймовірно, є найбільш важливою (і часто незамінною) річчю, що ми створюємо. Багато типів даних повинні зберігатися вічно.
2. Доступність відноситься до безвідмовної роботи і чутливості пам'яті систем зберігання даних в умовах окремої несправності компонентів або важкого навантаження на систему. Дані зазвичай повинні бути доступні в одну мить через безліч пристроїв.
3. Керованість - рівень зусиль і ресурсів, необхідних для підтримки функціонування в довгостроковій перспективі. Концепція може включати в себе персонал, час, ризик, гнучкість, і багато інших міркувань, які важко піддаються кількісній оцінці. Невелика кількість адміністраторів повинні бути в змозі підтримувати сервер зберігання.
4. Низька вартість - дані необхідно зберігати з найменшою вартістю. При наявності достатньої кількості ресурсів, ця проблема може бути проігнорована. Проте, ми живемо в світі обмежень. Бізнес-моделі і доступні бюджети вимагають недорогих рішень для зберігання даних. Є багато факторів, які повинні бути враховані в загальних витратах. Це включає в себе початкове авансові витрати, такі як витрати на придбання апаратних засобів. Але є і поточні витрати, які, можливо, потрібно буде врахувати. Ці витрати можуть включати додаткові ліцензії на програмне забезпечення, витрати на персонал для управління системою, а також витрати на центр даних.

Ці всі вимоги спрямовані саме до властивостей самого сховища. Однак, під час користування та підтримки сховища з’являються й інші. Їх створюють користувачі та той персонал, який займається підтримкою. До таких вимог можна віднести такі:

1. Можливість зручного завантаження файлів на комп’ютер та сервер
2. Розширений функціонал пошуку
    - За мітками
    - Повнотекстовий пошук
    - Зміна логічного оператору поєднання тегів
3. Можливість створення тегів
4. Легкий процес аутентифікації
5. Відображення файлів у вигляді звичної файлової структури

Ті функції, які повинен мати змогу виконувати користувач можна відобразити в діаграмі на рис. 1.1.

В зв’язку з тим, що жоден з цих сервісів не відповідає поставленим вимогам, з’являється потреба в створеннф нового, який би включав в себе всі існуючі можливості та задовільняв необхідні потреби.

## 1.4 Архітектура системи

В ході досліджень, була прийнята на озброєння класична архітектура створення продуктів даного типу, приведена на рис. 1.2, діаграма послідовностей на рис. 1.3. Головними складовими системи є клієнтська та серверна частини.  Зв’язок між цими частинами відбувається за допомогою REST API. REST - архітектурний стиль взаємодії компонентів розподіленого

додатка в мережі. REST є узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи.

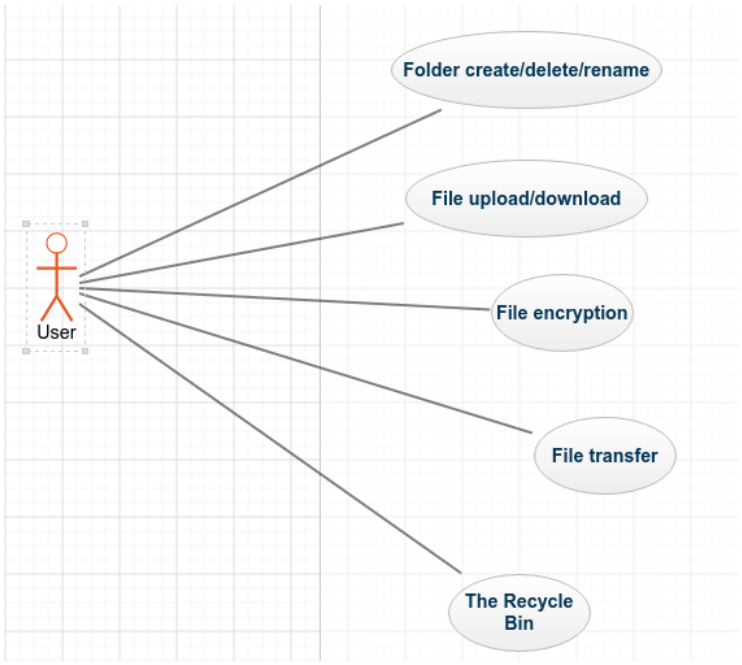


Рисунок 1.1 – Use case діаграма

У певних випадках це призводить до підвищення продуктивності і спрощення архітектури. У мережі Інтернет виклик віддаленої процедури може являти собою звичайний HTTP-запит (зазвичай GET або POST; такий запит називають REST-запит), а необхідні дані передаються в якості параметрів запиту.

### 1.4.1 Серверна частина

В основі системи лежить сховище певного типу. Існують три основні категорії сховищ даних: блочне сховище, файлове сховище і об’єктне сховище. Блочне сховище зберігає структуровані дані, які представлені у вигляді блоків однакового розміру, не змінюючи інтерпретацію збережених бітів. Часто цей

вид зберігання корисний, коли програма має жорстко контролювати структуру даних.

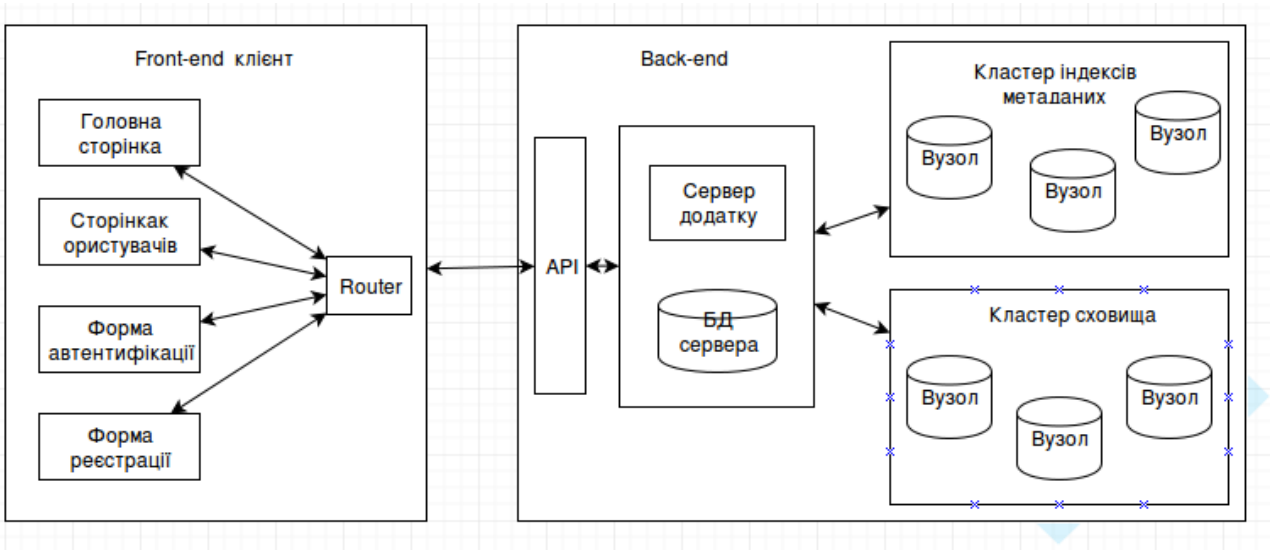


Рисунок 1.2 – Архітектура сховища

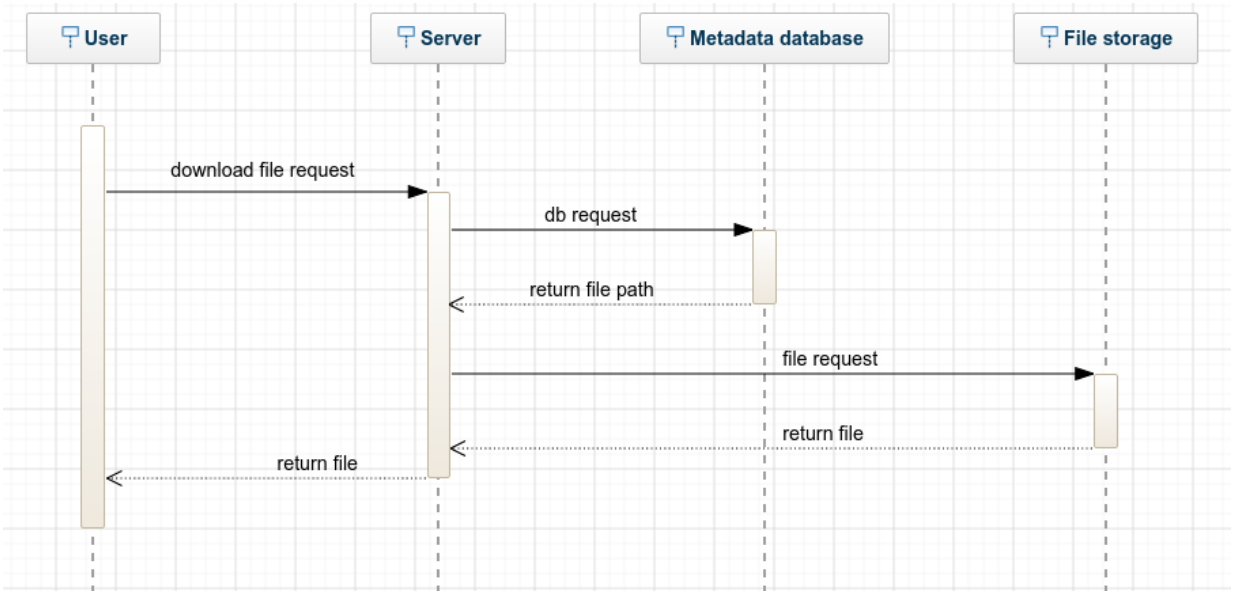


Рисунок 1.3 – Діаграма послідовностей

Зазвичай воно використовується базами даних, які можуть використовувати блоковий пристрій для ефективного читання і запису

структурованих даних. Файлове сховище - це те, що клієнти найбільше звикли бачити, як користувачі настільних комп'ютерів. У своїй простій формі, файлове сховище використовує жорсткий диск і розгортає на ньому файлову систему для зберігання неструктурованих даних. Центри зберігання даних містять системи, які розгортають файлову систему в межах мережі. Хоча файлове сховище забезпечує зручну модель даних, існують проблеми з масштабуванням. Файлове сховище потребує суворої узгодженості, що створює труднощі коли система зростає і збільшується кількість запитів. Крім того, файлові системи часто вимагають інших функцій (наприклад, блокування файлів), які створюють перешкоду для роботи з великими обсягами даних.

Щодо об’єктного сховища, то його особливістю є те, що кожен файл має унікальний глобальний ідентифікатор крім користувацького. Якщо фізичне розташування об'єкта змінюється, зміни обробляються внутрішнім механізмом об'єктного сховища, а ідентифікатор, призначений користувачем або додатком залишається незмінним, що дозволяє легко отримати доступ до нього в будь-який інший час.

Об’єктне сховище може зберігати велику кількість даних та доступ до них є більш простим, оскільки воно має декілька незалежних вузлів, які зберігають дані і центральний вузол не має знати місцезнаходження об’єкта для того, щоб отримати його.

Ідентифікатори можна використовувати для того, щоб легко порівняти два файла без необхідності завантаження. Об'єктне сховище також дає можливість використання великої кількості простого, більш дешевого апаратного забезпечення різних типів, пов'язуючи все разом в єдину систему. Кожен об'єкт, як правило, має кілька реплік, можливо в географічно

розподілених кластерах. Кожен об'єкт також містить контрольну суму, яка дозволяє легко виявити пошкодження даних, в цьому випадку можлива генерація нової копії об'єкта для заміни пошкодженої. Об'єктні сховища вже знайшли застосування в галузі охорони здоров'я, фінансів та індустрії розваг. Вони стають привабливими для ще більш широкої групи підприємств, в той час як технічні, управлінські, а також архівні вимоги ввели попит на масивні сховища даних в межах IT відділів.

Facebook, Instagram і Twitter використовують дану технологію для зберігання мільйонів користувальницьких фотографій кожен день. Spotify використовує її для зберігання мільйонів музичних треків. Dropbox і інші сервіси зберігають мільйони завантажених документів в об'єктному сховищі, відображеному у звичному, що містить файли і папки, інтерфейсі.

Використовуючи об’єктне локальне файлове сховище, організації зберігають повний контроль над безпекою та фізичним місцезнаходженням своїх даних, і в той же час проявляються ключові переваги хмари і об'єктного сховища такі, як масштабованість, ефективність, відмовостійкість і простота. Також локальне об’єктне сховище дозволяє значно знизити вартість зберігання об'єкта, в порівнянні з публічною хмарою.

### 1.4.2 Метадані

Метадані є неоднозначним і загальним терміном, але це найчастіше відноситься до імен атрибутів, типів даних, відносин, основних показників якості даних, статистики використання та управління доступом. Метадані - це

дані про використання даних, які часто не записуються а лише заповнюються працівниками підприємства.

З тих пір, як дані мають велику цінність, метадані, що описують їх, стали одним з ключових активів сучасних підприємств, що грає велику роль у виявленні дійсного значення даних. Організований процес управління корпоративними метаданими може запропонувати цінну інформацію їх неявних джерел. Схема можливих типів метаданих приведена на рис. 1.4. Бізнес метадані додають контекст до даних, зберігаючи інформацію про їх ролі в бізнес процесах з урахуванням предметної області і службові дані про стан файлів, такі як формат, місце розташування, інформацію про те, хто, коли і яким чином використовував файл.

Сховище метаданих дозволяє зберігати всю інформацію про структуру контейнерів з цими організації в одному місці. Це надає велику кількість матеріалу для обробки і аналізу з подальшою оптимізацією бізнес процесів. Зазвичай підприємства витрачають велику кількість часу і коштів на дослідження і прийняття рішень щодо впровадження та управління новими структурами даних. Сховища метаданих грають велику роль в організації великих обсягів неструктурованих даних, надаючи інфраструктуру для обробки метаданих та складання бізнес глосарію.

Добре складений бізнес глосарій є незамінним ресурсом для надання інформації про зв'язки та відносини між даними в базах даних, графах, моделях і т. д. Управління метаданими спрощує процес побудови бізнес лексикону в разі, якщо кількість визначень виростає до сотень і тисяч, спрощуючи взаємодію між різними організаційними структурами підприємства.

Фіксація і освоєння цих метаданих в легко доступному каталозі може відкрити великі можливості для організації. Зокрема, каталог метаданих може підвищити доступність даних підприємства. Працівники можуть швидко і впевнено зібрати необхідні дані для аналізу, зрозуміти, як взаємодіють дані. Користувацькі інтерфейси дозволяють легко для фахівцям по роботі з даними, власникам і користувачам мати зручний доступ до каталогу, що може допомогти в організації спільної роботи і отримання загального розуміння джерел даних по всьому підприємству.

Метадані можуть зберігатися або всередині, в тому ж файлі , або зовні, в окремому файлі або полі з описом даних. Сховище даних зазвичай зберігає метадані окремо від даних, але воно може бути спроектовано для підтримки мішаних підходів до зберігання метаданих. Кожен варіант має свої переваги і недоліки. Внутрішнє зберігання означає, що метадані завжди переміщуються як частина описуваних ними даних. Таким чином, метадані завжди доступні з даними, і ними можна маніпулювати локально. Цей метод створює надлишковість і не дозволяє управляти всіма метадані системи в одному місці. Це, можливо, збільшує цілісність, так як метадані легко змінюються щоразу, коли змінюються дані.

Відокремлене сховище метаданих дозволяє локалізувати метадані для всього сховища, наприклад, в базі даних, для більш ефективного пошуку і управління. При такому підході, метадані можуть бути об'єднані з контентом, коли інформація передається, або можна вставляти веб посилання.

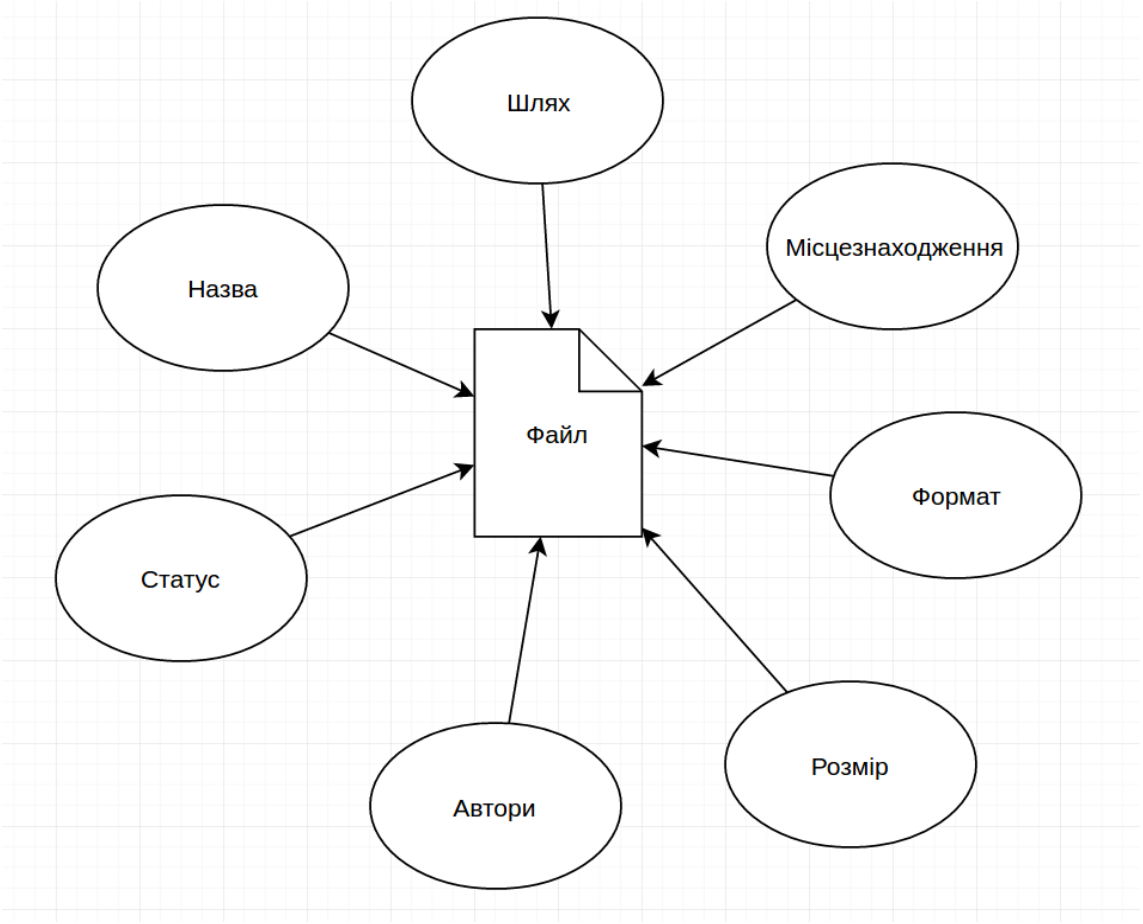


Рисунок 1.4 - Метадані

Метадані можуть бути збережені в бінарному вигляді або зручному для читання форматі. У другому випадку, наприклад, використовуючи формати XML або JSON, користувачі можуть зрозуміти і редагувати його без спеціальних інструментів. З іншого боку, ці формати рідко оптимізовані для ємності, часу передачі, і швидкості обробки даних.

## 1.5 Висновок до розділу 1

В результаті було проаналізовано предметну область та визначено вектор подальших досліджень. Було розглянуті причини використання локальних

систем зберігання даних та проведене порівняння з зовнішніми сервісами такого типу.

Також було розглянуто функціонал існуючих рішень. У розділі було сформовано головні вимоги до системи. Вони базуються на вимогах безпосередньо до самих сховищ та потреб користувачів. Це посприяло детальному дослідженню архітектури системи задля забезпечення повного функціоналу. Була затверджена архітектура всієї системи та її компонентів.

# 2 ФОРМУВАННЯ ВИМОГ ДО ІНТЕРФЕЙСУ. АНАЛІЗ РЕАЛІЗАЦІЇ КЛІЄНТСЬКОЇ ЧАСТИНИ

## 2.1 Клієнтський інтерфейс

UI - це різновид інтерфейсів, в якому одна сторона представлена людиною (користувачем), інша - машиною та пристроєм. Являє собою сукупність засобів і методів, за допомогою яких користувач взаємодіє з різними, найчастіше складними, машинами, пристроями та апаратурою. Іншими словами, призначений для користувача інтерфейс є системою, за допомогою якої люди (користувачі) взаємодіють з машиною. Інтерфейс включає в себе два компоненти: обладнання і програмне забезпечення. Існують інтерфейси для різних систем. UI є засобом для виконання двох типів взаємодії:

1. Введення (Input), що дозволяє користувачам управляти системою;
2. Висновок (Output), що дозволяє системі демонструвати ефект від вироблених користувачем маніпуляцій.

Основною метою при розробці інтерфейсу для взаємодії людини з машиною є створення UI, який дозволить легко (інтуїтивно), ефективно, і з задоволенням (user friendly) домагатися бажаного результату під час роботи з машиною. Ідеальним результатом вважається той, при якому оператор (або користувач) виробляє мінімальні зусилля для введення, отримуючи від машини бажаний висновок, при цьому машина мінімізує у висновку зайві дані. Існує безліч типів призначеного для користувача інтерфейсу, однак до тих, що відносяться до інформаційних технологій варто віднести такі:

1. Графічний інтерфейс користувача (GUI). Користувач здійснює операцію введення через пристрої, такі як комп'ютерна клавіатура і миша, а машина і забезпечує графічний висновок на моніторі комп'ютера.
2. Веб-інтерфейс (WUI). Це сукупність засобів, за допомогою яких користувач взаємодіє з веб-сайтом або будь-яким іншим додатком через

браузер. Нові реалізації використовують Java, JavaScript, AJAX, Adobe Flex, Microsoft. NET, або аналогічні технології. З їх допомогою можливий контроль в режимі реального часу, що виключає необхідність оновлення, що складається в основі HTML-браузерів.

3. Адміністративні веб-інтерфейси. Використовуються для роботи з серверами і віддаленими комп'ютерами. Часто називаються панеллю керування.
4. Сенсорні екрани = це дисплеї, які беруть введення при торканні пальцями або стилусом. Використовується в мобільних пристроях, вуличних автоматах різних типів і т. д.

Розробка UI - це створення інтерфейсу, який забезпечує найкращий, простий, приємний і не обтяжує спосіб взаємодії користувача з продуктом. Більшу частину роботи під час створення інтерфейсу становить спостереження за поведінкою користувача, що дозволяє приймати рішення, засновані на зібраних даних.

## 2.2 Досвід взаємодії з користувачем

Досвід взаємодії з користувачем - це сприйняття і дії у відповідь користувача, що виникають в результаті використання подальшого використання продукції, системи або послуги. Досвід користувача включає всі емоції, переконання, переваги, відчуття, фізичні та психологічні реакції користувача, поведінку і досягнення, які виникають до, під час і після використання системи. Досвід користувача поєднує образ торгової марки, спосіб представлення, функціональність, продуктивність системи, інтерактивну поведінку і допоміжні можливості інтерактивної системи, фізичний і психологічний стан користувача, що є результатом попереднього досвіду,

звичок, навичок і індивідуальності. Даний термін широко застосовується в інформаційних технологіях для опису суб'єктивного ставлення, що виникає у користувача в процесі використання як інформаційної системи в цілому, так і окремої її частини. Досвід користувача пов'язаний з такими поняттями як людино-комп'ютерна взаємодія і usability(зручність кристування), застосовуваним при розробці та аналізі призначених для користувача інтерфейсів і додатків. Розробники приділяють значну увагу вивченню і проектуванню досвіду користувача на всіх етапах створення продукту, починаючи з самого раннього - етапу планування до проведення його тестування.

П'ять рівнів - це концептуальна модель, запропонована Дж. Гарреттом для проектування досвіду користувача веб-додатків. Модель пропонує основу для обговорення проблем, пов'язаних з досвідом користувача, а також можливих шляхів і засобів їх вирішення. Розробка програми починається з верхнього рівня (стратегії), на якому досить абстрактно описується майбутній програмний продукт з точки зору очікувань як користувачів, так і замовника. В ході роботи над проектом - тобто просування вниз по рівням - рішення, пов'язані з досвідом користувача стають конкретніше і знаходять більш високий ступінь деталізації. Кожен наступний рівень тісно пов'язаний з попереднім (верхнім) і передбачає сувору узгодженість рішень. Очевидно, що при такому підході діапазон можливих рішень значно скорочується з переходом на кожний наступний рівень. Втім, це не означає, що всі рішення на конкретному рівні повинні бути прийняті до переходу на наступний рівень.

Між рівнями існує як пряма, так і зворотна залежність - тобто проблеми, з якими розробники стикаються на нижніх рівнях, часом вимагають переоцінки

і зміни рішень, прийнятих на більш високих рівнях. Слід зазначити, що процес розробки та планування досвіду взаємодії починається з верхнього рівня (стратегії) і поступово проходить через всі рівні до самого нижнього. Описати рівні можна таким чином:

1. Рівень поверхні являє собою зовнішній вигляд продукту з точки зору кінцевого користувача, тобто набір тексту, картинок, посилань, форм, вкладок, кнопок та іншого.
2. Під поверхнею знаходиться рівень компонування, що представляє конкретну реалізацію абстрактної структури продукту. На цьому рівні вирішуються питання найбільш ефективного розташування різних елементів UI.
3. На рівні структури описується взаємне розташування сторінок веб-сайту, програмних форм, вікон та ін. Тобто він відповідає на питання "звідки", "куди" і "як" зможе переміщатися користувач. Ефективна структура полегшує навігацію і робить її інтуїтивно зрозумілою для користувачів.
4. Рівень набору можливостей представляє з себе простий перелік набору функціональних можливостей, які будуть доступні для користувачів. Спосіб реалізації та взаємної організації цих можливостей буде описаний детальніше вже на рівні структури.
5. Рівень стратегії - це найвищий і найбільш абстрактний рівень представленої моделі. На цьому рівні необхідно отримати відповіді на питання, що стосуються бажань і очікувань щодо майбутнього програмного продукту, як з боку потенційних користувачів, так і

замовника. Відповіді на ці питання будуть сформовані і подані у вигляді конкретного списку на рівні набору можливостей.

## 2.3 Формування вимог до інтерфейсу

Ключовою частиною сервісу зберігання даних є саме веб-інтерфейс. З одінєї сторони, використання десктопних або мобільних додатків було б зручніше, однак використання саме веб дозволяє користуватись всіма можливостями і не бути залежним від пристрою. Враховуючи поставлені вимоги до ПЗ, веб-інтерфейс варто розробляти як окремий компонент. Такий підхід реалізовує певну автономність, що в свою чергу забезпечує більшу швидкодію, стійкість та масштабованість. Окрім переліченого, процес розробки в такому випадку простіше організувати для команди. Однак з точки зори користувача, йому не є принциповою архітектура додатку, тому він створює свої певні вимоги:

1. Можливість реєстрації
2. Адміністратор повинен мати змогу підтвердити/заблокувати обліковий запис
3. Користувач повинен легко авторизуватись
4. Вигляд файлової структури повинен бути таким же, як і звична для користувача структура на його комп’ютері
5. Користувач повинен мати можливість завантажити/відвантажити документ
6. Можливість легкого пошуку за мітками
7. Пошук за семантичним ядром

Відповідно, всі дії, які може виконати користувач повинні бути легкими, зрозумілими і не вимагати складних послідовностей дій.

## 2.4 Створення додатку у вигляді SPA

Враховуючи дані вимоги, можна стверджувати, що в якості інтерфейсу варто розробити односторінковий додаток (SPA), також відомий як односторінковий інтерфейс (SPI) - це веб-застосунок чи веб-сайт, який вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою. За даних умов реалізація серверного рендерингу буде не оптимальним рішенням. В односторінковому додатку весь необхідний код (HTML, JavaScript, та CSS) завантажується разом із сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача на іншу сторінку в процесі роботи з нею. Взаємодія з односторінковим додатком часто включає в себе динамічний зв'язок з веб-сервером. Розробка такого типу сторінок - досить класичне завдання для front-end розробника. Інструментами для розробки в такому випадку будуть:

1. JavaScript (JS) — динамічна, об'єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. Мова JavaScript також

використовується для програмування на стороні сервера (подібно до таких мов програмування, як Java і C#), розробки ігор, стаціонарних та мобільних додатків, сценаріїв в прикладному ПЗ, всередині PDF-документів тощо. JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну, декларативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу. Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування. Синтаксис обох мов отриманний «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme.

2. HTML5 — наступна версія мови HTML. До складу робочої групи з HTML5 AOL, Apple, Google, IBM, Microsoft, Mozilla, Nokia, Opera та кілька сотень інших виробників. 28 жовтня 2014 консорціум W3C оголосив про надання набору специфікацій HTML5 статусу рекомендованого стандарту. Цікаво, що у цьому вигляді специфікації HTML 5.0 були сформовані ще два роки до того, після чого робота була зосереджена на проведенні тестування та оцінки сумісності доступних реалізацій. На час стандартизації HTML5 вже давно став стандартом де-факто і активно використовується у веб-

застосунках. Фактичне затвердження стандарту лише формально поставило крапку в просуванні HTML5 і підтвердило повсюдність і коректність його реалізації. Специфікації HTML5 не обмежуються тільки розміткою і включають в себе низку веб-технологій, котрі у сукупності формують відкриту веб-платформу — програмне оточення для роботи крос-платформенних застосунків, здатних взаємодіяти з обладнанням, і які підтримують засоби для роботи з відео, графікою і анімацією, що надає розширені мережеві можливості.

3. CSS — спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних. CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

Саме цей стек мов зазвичай дозволяє розробляти веб-інтрейфейси. Однак за останній час з’явилась велика кількість додаткових засобів, що прискорюють та полегшують розробку.

Якщо говорити про такі інструменти, то спершу варто зазначити, що більшість з них неможливо використовувати без наявного Node.js інтерпритатора. Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Платформа Node.js перетворила мову JavaScript, що здебільшого

використовувалась в браузерах на мову загального використання з великою спільнотою розробників. Використання цієї платформи необхідне в першу чергу через пакетний менеджер npm. npm - це пакетний менеджер Node.js. З його допомогою можна керувати модулями і залежностями. Саме він дозволяє налаштувати робоче середовище відповідно до поставлених вимог. Відомо, що Node.js використовується здебільшого для back-end розробки, однак npm містить велику кількість пакетів, призначених і для розробки інтерфейсів.

## 2.5 Підбір інструментів розробки

Необхідною частиною для налаштування свого робочого середовища є таск-менеджери. Вони використовуються для автоматизації однотипних дій програміста під час розробки та збирання проектів. Це значить, що, задавши необхідну конфігурацію, процеси мінімізації коду, компресії, препроцесорна обробка можуть запускатись всі разом однією командою. Пряме визначення - це інструмент для автоматизації роботи (конкатенація, мінімізація файлів, робота з препроцесорами, відстеження помилок в коді, робота з зображеннями і багато іншого) розробників. Однак постає питання вибору між двома найбільш відомими, а саме - gulp чи grunt. З огляду на те, що gulp стабільніший і містить більшу кількість різного роду додатків, очевидно, що вибір повинен схилятись саме в його сторону. Одним головних сегментів, необхідних для запуску менеджера - наявність gulpfile.js, файлу, що містить всю конфігураційну інформацію. Зазвичай ці файли для різних проектів не сильно відрізняються, адже в будь-якому разі існує певний стек завдань, необхідних в кожному проекті, серед них такі: мініфікація, конкатинація, використання препроцесори.

Використання таких засобів значно пришвидшують розробку та оптимізують. Однак, окрім того, що було сказано вище про схожість налаштувань для всіх проектів, постає питання також структуризації свого проекту. Зазвичай кожний js-фреймворк несе з собою певні домовленості щодо організації проекту. Саме ці домовленості переважно однакові всюди і немає сенсу щоразу створювати одну і таку ж структуру для різних проектів. Для пришвидшення роботи в таких випадках використовується генератор коду початкового скелету Yeoman. Отже, для початку варто визначитися, який програмний код він генерує. В програмування існує таке поняття — скаффолдинг. Воно означає автоматичну побудову структури проекту. За деякими даними воно вперше було використано в Ruby on Rails. Там була спеціальна команда, яка генерувала код головних компонентів проекту, а саме - контролерів і моделей. Зазвичай робота програміста досить однотипна. Вона найчастіше полягає в тому, щоб вже заздалегідь установлені межі бібліотеки, фреймворку або движка внести власні корективи для реалізації потрібного функціоналу. Таким чином, перший крок роботи по створенню чогось нового у будь-якій системі відтворення базового шаблону тієї або іншої конструкції, яка закладена в її фундамент. Генератори пропонують позбутися трати часу на початкові кроки створення проекту. Як приклад - позбутися від рутинної операції тиражування вихідних текстів. Однак, така можливість генерації каркаса частин програми вбудована далеко не в кожну систему. Отже, Yeoman - якісне рішення для побудови скелету. Він надає велику кількість різних генераторів для різних фреймворків. І саме для таких можна знайти готові рішення:

1. AngularJS це фреймворк JavaScript з відкритим програмним кодом, який розробляє Google. Він призначений для розробки

односторінкових додатків, що складаються з однієї HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних додатків на основі шаблону модель-вид-контролер (MVC), а також спрощення їх тестування та розробки.

2. React.js - JavaScript бібліотека з відкритим програмним кодом для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових додатків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.
3. Ember.js - вільний JavaScript каркас веб-додатків, який реалізує MVC шаблон, призначений для спрощення створення масштабованих односторінкових веб-додатків. Фреймворк використовується такими компаніями як TED, Yahoo!, Twitch.tv і Groupon.

## 2.6 Вибір фреймворку

На цьому етапі постає питання вибору фреймворку, який би найкраще відповідав поставленим вимогам. Найпопулярнішим на даний час, є React.js, однак саме для заданих умов найкращим вибором є AngularJS. Простота написання коду, масштабованість, пристосованість до роботи з Rest API - його основні переваги. Варто також зауважити, що після своєї появи він досить швидко завоював ринок і на разі є не менш актуальним при написанні веб-додатків за React.js. Тим паче, що поява Angular 2.0 повинна взагалі забезпечити йому надійне місце серед використовуваних фреймворків. Додаток Angular 2 тепер являє собою дерево компонентів. Ідея використання

незалежних функціональних, зі своїми стилями, блоків HTML схожа до ідеї створення веб-додатків - web-components, а розмітка Angular 2 компонентів поміщається в Shadow DOM – внутрішній DOM кожного складного елемента, що відображається як один. При чому при використанні в додатку web-components або елементів бібліотеки webcomponents.js - Polymer синтаксис нічим не буде відрізнятися від використання власних створених Angular 2 компонентів. Самі компоненти являють собою класи стандарту мови JavaScript EcmaScript6 з анотаціями. Не потрыбно ніякого спеціального синтаксису як, наприклад, для директив у версії 1.x не потрібно. Сервіси тепер теж стали звичайними класами. В загальному AngularJS базується на MVC патерні.

## 2.7 Аналіз базового патерну

Шаблон проектування MVC передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: Модель, Представлення та Контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно. У наведеному визначенні під компонентом слід розуміти якусь окрему частину коду, кожна з яких грає одну з ролей контролера, моделі або представлення, де модель дозволяє отримувати і маніпулювати даними додатка. Представлення відповідає за видиме користувачеві відображення цих даних (тобто, в застосуванні до інтернету, формує відповідь сервером браузеру користувача HTML / CSS), а контролер керує всім цим. Класична схема веб-додатку зображена на рис. 2.1-2.3. На рисунках пунктирними лініями показана керуюча інформація (така, наприклад, як ID запитуваного запису блогу або товару в

магазині), а суцільними - власне дані додатку. Важливе зауваження: концепція MVC не тільки не прив'язана до якоїсь конкретної мови програмування, вона також не прив'язана і до використовуваних парадигм програмування.

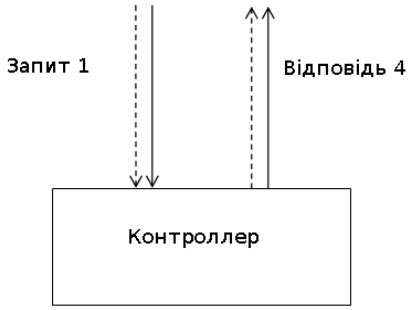


Рисунок 2.1 – Запит до контролера[4]

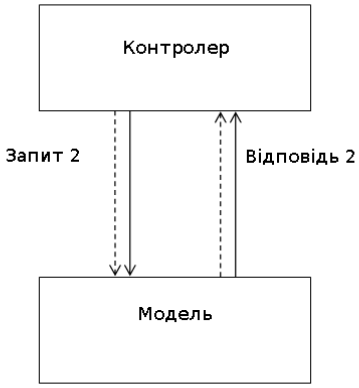


Рисунок 2.2 – Зв’язок моделі з контролером[4]


Рисунок 2.3 – Зв’язок представлення та контролера[4]

Питання про те, хто повинен перевіряти на валідність і права доступу вхідних даних (Контролер або Модель), є предметом досить численних суперечок, оскільки патерн MVC не описує таких деталей. Це означає, що в цьому питанні вибір за розробником.

## 2.8 Додаткові засоби розробки

В такому випадку, зосереджуючи свій вибір на AngularJS, варто звернути увагу на існуючі генератори в Yeoman. Серед існуючих, найкраще відповідає посталеним вимогам angular-gulp генератор. Він містить в собі все необіхдне для налаштування робочого середовища. Сюди варто віднести залежності описані у файлах package.json і bower.json, повністю нашалаштований gulfile.js, який в свою чергу довзоляє запустити розробницький сервер виключно для інтерфейсу. Також під час встановлення можна вказати додаткові бібліотеки, які є необхідними. Серед основної маси можна виділити такі.

1. Bootstrap — це безкоштовний набір інструментів з відкритим вихідним кодом, призначений для створення веб-сайтів та веб-додатків, який

містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-застосунків.

2. Sass — скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS). Спроектована Гемптоном Кетліном та розроблена Наталі Вейзенбаум. Sass призначений для підвищення рівня абстракції коду та спрощення файлів CSS.

Серед перерахованих, необхідним для вирішення поставленої задачі є Bootstrap. Він чудово поєднується з AngularJS, який містить адаптовані під фреймворк плагіни. Саме такий набір технологій та засобів необхідний для створення продукту, який буде відповідати поставленим вимогам.

## 2.9 Допоміжні модулі

### 2.9.1 Адресація сторінок в AngularJS

AngularJS - фреймворк, що містить велику кількість компонентів. Саме вони надають масштабованості та гнучкості. Головною функціональною одиницею частиною є модуль. Він може включати в себе контролери, директиви, фабрики, сервіси, фільтри. Також існує набір singletone об’єктів, що несуть в собі відокремлену логіку або дані. Їх використання дозволяє логічно структурувати код, що підвищує його читабельність та працездатність. Умовно його склад відображено на рис. 2.4. Кожний його компонент несе в собі певний функціонал та смислове навантаження. Для розробки багатофункціонального та масштабованого інтерфейсу необхідне використання додаткових модулів. Базуючись на створення SPA, потрібна адресація сторінок, реалізована виключно на клієнтській стороні. Аналізуючи роботу браузера, варто розглянути властивості url адреси сторінки. Результати наведені а

таблиці 2.1. Зміна майже всіх параметрів адреси призводить до повторного запиту та перезавантаження сторінки. Однак, при зміні hash, запит не надсилається, що дає змогу реалізувати адресацію виключного на стороні клієнта. Саме це є одним із основних принципів побудови SPA. В AngularJS для створення адресації існує модуль ngRoute, що зображений на рис. 2.5, а об'єкт $routeProvider дозволяє задати конфігурацію.

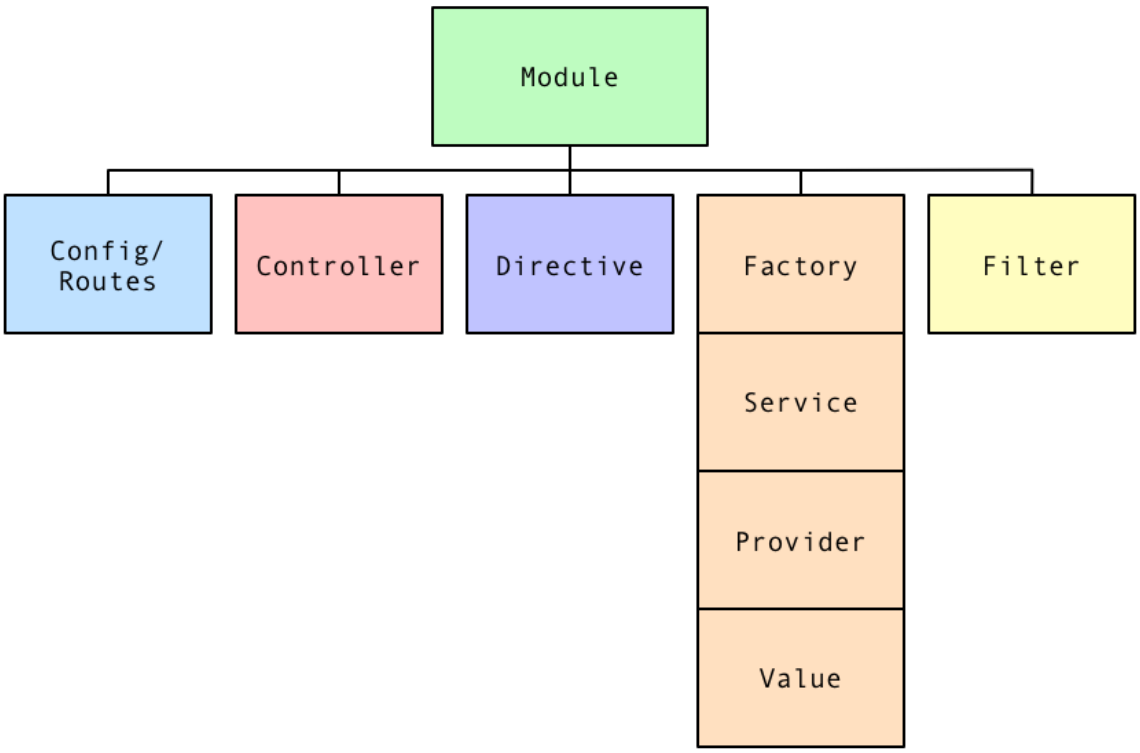


Рисунок 2.4 – Архітектура AngularJS[6]

Для використання інших частин адреси, використовується об'єкт $location. Він містить такі методи:

1. url - повертає всю адресу
2. protocol - повертає протокол
3. host - визначає хост
4. port - повертає порт
5. path - дає змогу отримати шлях

6. search - відображає параметри запиту

Стандартним засобом в Angular для надсилання AJAX запитів використовуються $http. Однак для створення додаткового рівня абстакції більш якісним є модуль ngResource.

Таблиця 2.1 - Властивості адреси сторінки

| Властивість | Опис | Приклад |
|---|---|---|
| hash | частина URL, яка йде після символу решітки “#” | #test |
| host | хост і порт | www.google.com:80 |
| pathname | шлях (відноста хоста) | /search |
| protocol | протокол | http: |
| query string | get-параметри | ?q=search |

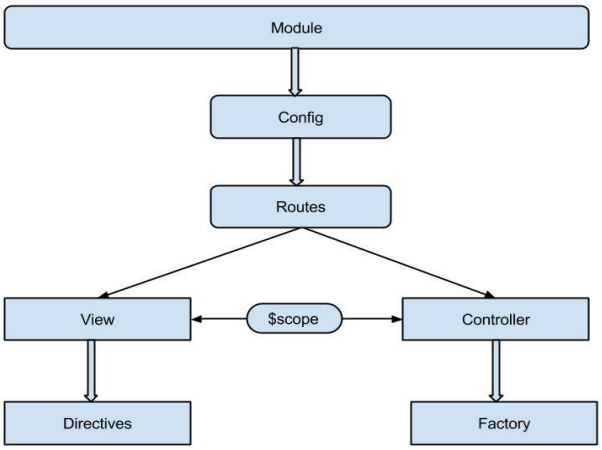


Рисунок 2.5 – Архітектура AngularJS з адресацією[8]

В ньому присутнє API $resource, зручне для операцій з базами даних CRUD. $resource створює об’єкт, через який з’являється можливість спілкування з джерелами даних по протоколам REST.

### 2.9.2 Модулі інтерфейсу

AngularJS був вперше випущений у 2009 році. Він зумів себе добре зарекомендувати і зібрав велику кількість прихильників, які в свою чергу допомагали набирати подальшу популярність. В зв'язку з цим була створена маса різних додатків для інтерфейсу, що б відповідали користувацьким вимогам. Враховуючи функціональні можливості системи, з'являється необхідність у використанні готових рішень. Адже варто враховувати користувацький досвід, який є першочерговим при розробці веб-додатків даного типу. Велику кількість інтерфейсних додатків були написані розробниками AngularJS на базі Bootstrap. До таких можна віднести:

1. Accordion - директива, що складається із заголовків та прихованої частини, яка розгоряється після кліку.
2. Alert - директива, що використовується для генерації повідомлень із статичних або динамічних даних.
3. Modal - директива для створення модальних вікон та надання їм функціоналу.
4. Rating - засіб для зручного відображення рейтингів у вигляді значків.
5. Інші

Враховуючи функціональну можливість сховища та потреби користувача, зі списку можливих директив необхідними є:

1. Datepicker - директива, яка буде надавати можливість легкого введення дати. Це дасть можливість зручного відображення лише тих файлів, завантаження яких потрапляє у вибраний часовий проміжок
2. Button - набір стилів та функціональності, необхідна для приведення інтерфейсу до певних стандартів.
3. Progressbar - директива для відображення рівня завантаження файлу.

Однією з головних функцій для користувача є можливість завантаження файлу у сховище. Серед існуючих рішень найзручнішим є Angular File Upload.

Він являє собою модуль для AngularJS. Підтримка завантаження за допомогою drag-n-drop, прогрес завантаження  і черга завантаження файлу – головні властивості. Він працює з будь-якою серверною платформою, яка підтримує стандартне завантаження HTML-форм. Головними його перевагами є:

1. Сумісність
2. Контроль завантаження
3. Звичність для користувача

Завдяки використанню цього модуля завантаження є простим, швидким і зрозумілим.  Серед функціоналу, доступного користувачу є також і додавання міток та фільтрування за декількома однотипними критеріями одночасно. Станадртне рішення - тег select, що дозволяє вибирати кілька пунктів за допомогою клавіши Ctrl є незручним та ненадійним. Окрім такого такий підхід мінімізує можливість зміни стилю. Як вирішення цих проблем, існує модуль Angular ui-select, зображений на рис. 2.6. Він надає можливість вибирати кілька пунктів одночасно, слідкувати та керувати ними. Окрім того, звичайний select також стає зручнішим, зрозумілішим та простішим у використанні.

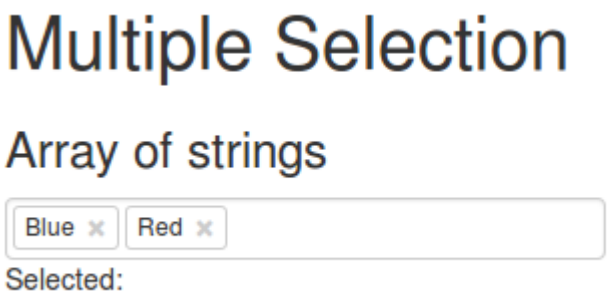


Рисунок 2.6 – Модуль ui-select

У вимогах до сховища закладений функціонал адміністратора, який матиме можливість дозволяти та блокувати доступ до системи іншим користувачам. Однак при збільшенні кількості користувачів, оперувати даними буде майже неможливо. Навіть при кількості в 70 записів, операції вимагатимуть занадто багато зусиль, через відсутність структуризації. Для

підвищення зручності необхідний модуль angular-datatables. Він відображає дані у вигляді таблиці, надає можливість керування відображенням. До основних його переваг варто віднести:

1. Швидкість
2. Зрозуміле відображення
3. Наявність розподілу на сторінки
4. Можливість керувати кількістю записів на сторінку
5. Пристосований для динамічних даних та пошуку

Необхідною властивістю для додатку є динамічний дизайн. Навіть у межах локальної мережі не виключене використання смартфонів та інших пристроїв, що вимагає адаптованості від інтерфейсу. Реалізувати такий дизайн можна за допомогою сітки Bootstrap. Її принцип полягає у поділі екрану на 12 частин і можливості визначення ширини блоку в частинах для різних екранів. Така можливість базується на медіа запитах, що з’явились у CSS3. Вони дозволяють в залежності від пристрою задати велику кількість параметрів відображення, а саме таких:

1. max-width - максимальна ширина дисплею, для якої будуть застосовані властивості
2. max-device-width - максимальна ширина пристрою, для якої будуть застосовані властивості
3. orientation - орієнтація, для якої будуть застосовані властивості
4. Інші

## 2.10 Висновок до розділу 2

В даному розділі проведено глибокий аналіз безпосереднього розробки інтерфейсу. В основу роботи було закладено користувацький досвід. Веб-додаток, що реалізовує інтерфейс системи збереження даних майже не несе інформативного навантаження. Його головними завданнями є:

1. Відповідність всьому функціональному стеку системи
2. Зручність інтерфейсу
3. Зрозумілість функціоналу
4. Звичність виконуваних дій для виконання операції

В зв’язку з тенденцією збільшення навантаження на клієнтську частину і створення нових інструментів, у розділі було розглянуто та проаналізовано необхідний набір засобів. В першу чергу це стосувалось вибору фреймворку. В результаті досліджень, було доведено, що найкращим в цьому випадку є AngularJS. В порівняння з популярним нині ReactJS він є повільнішим, особливо при роботі з великими об’ємами даних. Однак, описана система зберігання даних не передбачає відображення на клієнтській стороні великої кількості даних. Для прикладу, при обробці списку із 10 000 елементів, працездатність AngularJS значно падає, але так як система відображає користувацькі дані, то ситуація відображення такої великої колекції файлів фактично неможлива. Жоден користувач в здоровому глузді не буде створювати в одній директорії 10 000 фалів чи інших директорій. В цій ситуації необхідними є переваги саме AngularJS, головними з яких є:

1. Наявність великої кількості інтерфейсних модулів
2. Швидкість розробки
3. Зручна структуризація коду для подальшої масштабованості.

Так, як в основу фреймворку покладений патерн MVC, частина розділу містить інформацію про цей шаблон проектування. Також в цій частині роботи вказано, що для створення середовища найкраще вибрати gulp. В порівнянні з іншим такс-менеджером він є швидшим та зрозумілішим. Серед існуючих генераторів Yeoman існує якісна зв’язка Angular та gulp, яка і була обрана для розробки. В розділі описано вибір засобів для стилізації в результаті було обрано Bootstrap та Sass. В зв’язку з тим, що в основу вибору AngularJS лягло існування великої кількості плагінів, то в цій частині роботи також

проаналізовані плагіни, що необхідні для створення якісного інтерфейсу. Головними модулями, використаними у роботі є такі:

1. Angular File Upload
2. Angular ui-select
3. Angular datatables
4. Angular-ui модулі

В загальному розділ спрямований на аналіз розробки інтерфейсу, а саме на принципи реалізації та вибір необхідного інструментарію.

# 3 ОГЛЯД РЕЗУЛЬТАТІВ

Головним завдання було створення зручного та зрозумілого інтерфейсу, який би задовільнив всі функціональні можливості системи зберігання. Для цього головними були такі компоненти:

1. Блок завантаження файлів на сервер
2. select з можливістю вибору кількох пунктів
3. Компонент відображення вмісту поточної директорії
4. Таблиця для відображення списку користувачів

## 3.1 Інтерфейс для роботи з файлами

Блок завантаження файлів на сервер, реалізований за допомогою модуля Angular File Upload, зображений на рис. 3.1. У користувача є можливість вибору окремого набору файлів вручну або за допомогою Drag-n-Drop, а також наявна керованість чергою завантаження. Він може завантажувати файли кожен окремо або всі одразу при цьому додаючи певні метадані для полегшення пошуку в майбутньому. Головним завданням інтерфейсу є імітація руху по реальній файловій структурі. Адже для користувача є звичним відображенням файлів на його персональному комп’ютері. Для цього найзручніше зобразити контент у вигляді таблиці. Для цього існують готові закладені у фреймворк Bootstrap. Використання класу .table надасть для html-структури табличних стилів. Приклад зображено на рис. 3.2. Select з можливістю вибору кількох пунктів необхідний під час виконання таких операцій:

1. Завантаження файлів в сховище - для задання метаданих, а саме великої кількості міток.
2. Пошук файлів - для пошуку файлів за кількома мітками одночасно.

3. Пошук користувачів - для фільтрації користувачів, у яких відкритий та закритий доступ.

Результат наведено на рис. 3.3

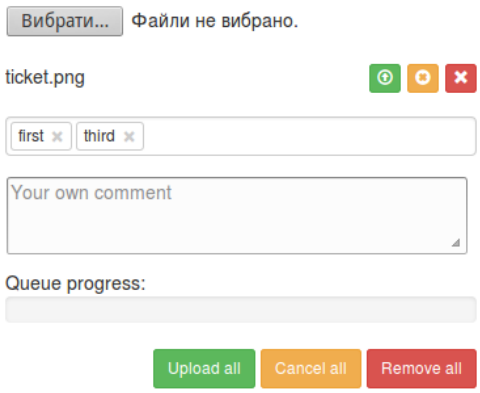


Рисунок. 3.1 – Модуль Angular File Upload


Рисунок. 3.2 – Структура директорій

## 3.2 Допоміжні засоби

В даному випадку за замовчуванням для міток буде спрацьовувати оператор “or”, змінивши значення чекбокса, можна встановити функцію “and”. Однак, для того, щоб набір міток був динамічним, також потрібне поле для їх додавання. Вигляд форми зображено на рис. 5.4.


Рисунок 3.3 – Фільтр за тегами

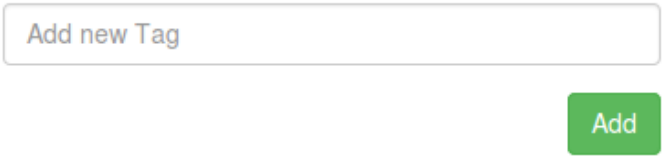


Рисунок 3.4 – Форма створення тегів

Таблиця для відображення користувачів повинна мати функціонал динамічного оновлення, сортування та структуризації списку за сторінками. Усіх цих вимог було дотримано за допомогою модуля Angular Datatable. Результат приведено на рис. 3.5

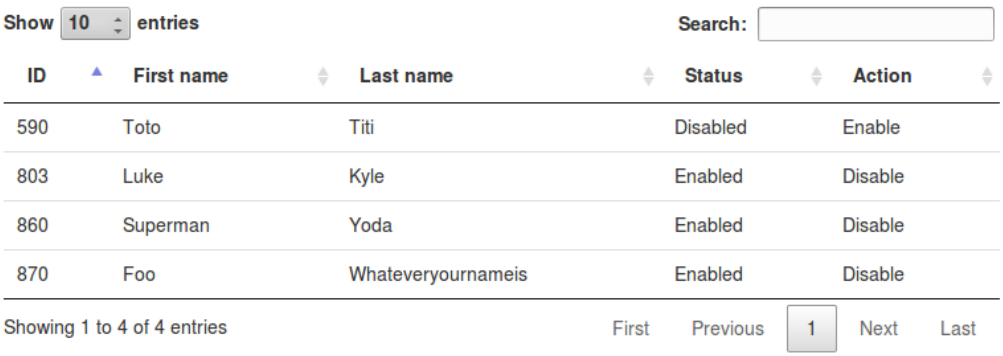


Рисунок 3.5 – Модуль AngularJS datatable

Серед 10 тестових користувачів, 9 зазначили, що інтерфейс повністю відповідає їхнім вимогам, 1 - вказали, що інтерфейс сприйнято як “досить зручний”.

# 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даній роботі проводиться оцінка основних характеристик програмного продукту, що полягає в створенні мобільного застосування для операційної системи iOS, використовуючи мову програмування Swift.

Для техніко-економічного обґрунтування програмного продукту буде використаний метод функціонально-вартісного аналізу (ФВА). Основою ФВА є функціональний підхід, згідно з яким об'єктом аналізу тобто не сам продукт, а функції, які він виконує. ФВА проводиться в два етапи: функціональний аналіз і вартісний аналіз.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій. Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

## 4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати у всіх сучасних браузерах;
- забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем;
- передбачати мінімальні витрати на впровадження програмного продукту.

## 4.2 Обґрунтування функцій програмного продукту

Головною метою програмного продукту є створення адаптивного інтерфейсу для користувачів за допомогою сучасних засобів. Ґрунтуючись на цьому, виділимо основні функції:

F1 – вибір найбільш зручного фреймворку;

F2 - інтерфейс користувача.

F3 – вибір середовища розробки;

Кожна з основних функцій може мати кілька варіантів рішення:

для F1:

а) EmberJS;

б) ReactJS;

в) AngularJS;

для F2:

а) інтерфейс за допомогою чистого CSS3;

б) інтерфейс створений з використанням Bootstrap;

для F3:

а) середовище розробки WebStorm;

б) середовище розробки IntelliJ IDEA;

За розглянутими варіантами будуємо морфологічну карту (Рисунок 4.1). На основі цієї карти побудуємо позитивно-негативну матрицю (Таблиця 4.1).

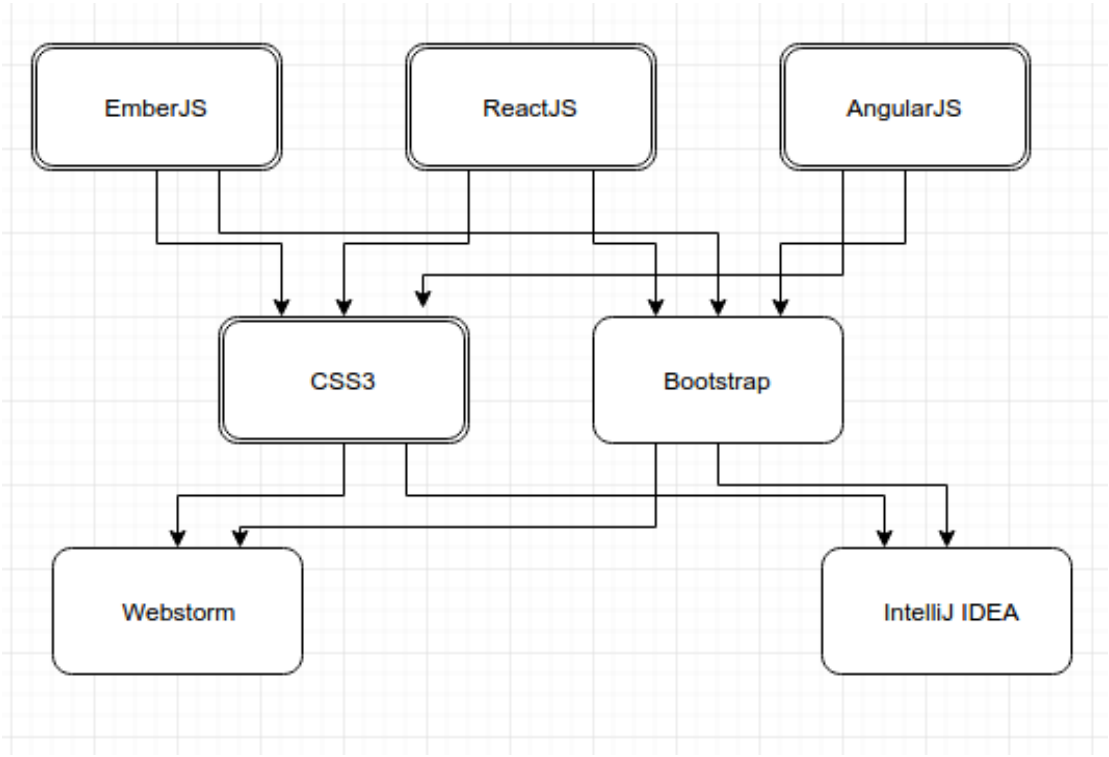


Рисунок 4.1 – Морфологічна карта

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам.

F1: Оскільки програмний продукт повинен швидко розроблятись і бути масштабованим, найкращий варіант - AngularJS.

F2: Через відсутність конкретного дизайну, обидва варіанти прийнятні

F3: Найкращим середовищем розробки є варіант а), оскільки він є швидким і орієнтованим лише на веб.

Табл.4.1 Позитивно-негативна матриця

| Основні функції | Варіанти реалізації | Переваги | Недоліки |
|---|---|---|---|
| *F1* | а) | Швидкий | Складність розробки |
| | б) | Відносно нешвидкий | Складність у програмуванні |
| | в) | Легкість у програмуванні | Повільний |
| *F2* | а) | Відсутність обмежень | Складний у розробці |
| | б) | Швидкодія, масштабованість | Нав’язування стандартів |
| *F3* | а) | Орієнтованість на веб | Відсутність додаткових можливостей |
| | б) | Наявність додаткових можливостей | Громісткий |

У зв’язку із оглядом основних функцій ПП наведеним вище, будемо розглядати наступні варіанти реалізації:

А: F1в) – F2а) – F3а)

Б: F1в) – F2б) – F3а)

## 4.3 Обґрунтування системи параметрів програмного продукту

На підставі даних про основні функції, що повинен реалізувати програмний продукт, визначаються основні параметри виробу, наведені в Таблиці 4.2. Будемо використовувати наступні параметри:

Табл.4.2 Визначення параметрів продукту

| | | | |
|---|---|---|---|
| *X1* – швидкість виконання | *X2* – час виконання | *X3* – ресурсоємність | *X4* – потенційний об’єм програмного коду |
| Відображає швидкість виконання по часу в залежності від фреймворку | Відображає час виконання | Відображає навантаженість на систему, спричинену роботою програмного продукту | Відображає розмір коду, який необхідно написати розробнику |

### 4.3.1 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів, що наведені в Таблиці 4.3, вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП:

Табл.4.3 Основні параметри продукту

| Назва параметра | Умовні позначення | Одиниці виміру | Значення параметру | | |
|---|---|---|---|---|---|
| | | | Гірші | Середні | Кращі |
| Швидкість виконання | X1 | мс | 1000 | 400 | 100 |
| Час виконання | X2 | с | 10 | 5 | 2 |
| Ресурсоємність | X3 | % | 100 | 80 | 20 |
| Об'єм програмного коду | X4 | Кількість рядків коду | 3000 | 2000 | 1000 |

За даними табл. 4.3 побудуємо графіки залежності бальної оцінки параметра від його основного значення (рис. 4.2 – 4.5).

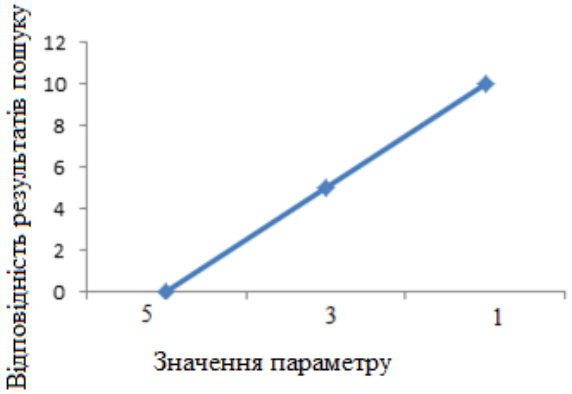


Рисунок 4.2 – Відповідність результатів пошуку

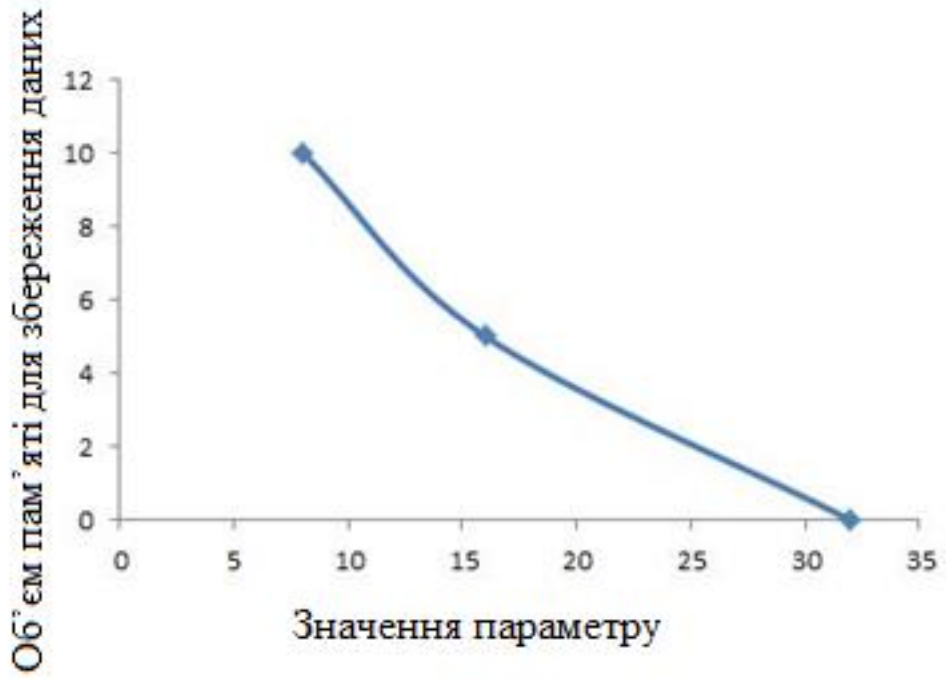


Рисунок 4.3 – Об’єм пам’яті для збереження даних


Рисунок 4.4 – Потенційний об’єм програмного коду

### 4.3.2 Аналіз експертного оцінювання параметрів

Важливість кожного параметра в загальній кількості розглянутих під час оцінювання параметрів, що наведені в таблиці 4.4 знаходять за методом попарного порівняння. Оцінювання проводить експертна комісія із семи осіб. Визначення коефіцієнта важливості передбачує:

- визначення рівня важливості параметра через присвоєння різних рангів;
- перевірити придатність експертних оцінок у подальшому використанні;
- визначити оцінки попарного пріоритету параметрів;
- обробити результати і знайти коефіцієнт важливості.

Після детального обговорення й аналізу кожний експерт оцінює рівень важливості, присвоюючи їм ранг. Результат експертного ранжування наведено в табл. 4.4.

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів за формулою (4.1):

$$R_i = \sum_{j=1}^{N} * r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105, \quad (4.1)$$

де $N$ – число експертів, $n$ – кількість параметрів;

б) середня сума рангів зі формулою (4.2):

$$T = \frac{1}{n} R_{ij} = 26{,}25. \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів за формулою (4.3):

$$\Delta_i = R_i - T \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення за формулою (4.4):

$$S = \sum_{i=1}^{N} * \Delta_i^2 = 420{,}75. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості за формулою (4.5):

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 420{,}75}{7^2(5^3-5)} = 1{,}03 > W_k = 0{,}67 \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.5:

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, $a_{ij}$ визначається по формулі (4.6).

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за формулами (4.7, 4.8):

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \tag{4.6}$$

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.7)$$

$$\text{де } b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами(4.9, 4.10).

Табл. 4.4 Результати ранжування параметрів.

| Позначення параметра | Назва параметра | Одиниці виміру | Ранг параметра за оцінкою експерта | | | | | | | Сума рангів Ri | Відхилення Δi | Δi2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | | |
| X1 | Швидкість виконання | мс | 4 | 3 | 4 | 4 | 4 | 4 | 4 | 27 | 0,75 | 0,56 |
| X2 | Час завантаження і обробки даних | с | 4 | 4 | 4 | 3 | 4 | 3 | 3 | 25 | -1,25 | 1,56 |
| X3 | Ресурсоємність | % | 2 | 2 | 1 | 2 | 1 | 2 | 2 | 12 | -14,25 | 203,06 |
| X4 | Об’єм програмного коду | кількість строк коду | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 41 | 14,75 | 217,56 |
| Разом | | | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 105 | 0 | 420,75 |

Табл. 4.5 Попарне порівняння параметрів.

| Параметри | Експерти | | | | | | | Кінцева оцінка | Числове значення |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | |
| X1 і X2 | = | > | = | < | = | < | < | < | 0,5 |
| X1 і X3 | < | < | < | < | < | < | < | < | 0,5 |
| X1 і X4 | > | > | > | > | > | > | > | > | 1,5 |
| X2 і X3 | < | < | < | < | < | < | < | < | 0,5 |
| X2 і X4 | > | > | > | > | > | > | > | > | 1,5 |
| X3 і X4 | > | > | > | > | > | > | > | > | 1,5 |

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$\text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

## 4.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 (швидкість виконання) та X4 (кількість строк) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра Х3 (ресурсоємність) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 20 % або варіанту б) 80%.

Абсолютне значення параметра Х2 (час виконання) також обрано на основі експертних знань: а) 2 с; Б) 5с.

## 4.5 Розрахунок показників якості варіантів реалізації

Коефіцієнт технічного рівня, що наведений у таблиці 4.6 для кожного варіанта реалізації ПП розраховується за формулою (4.11):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \tag{4.11}$$

де $n$ – кількість параметрів; $K_{\text{в}i}$– коефіцієнт вагомості $i$–го параметра; $B_i$ – оцінка $i$–го параметра в балах.

За даними з таблиці за формулою (4.12):

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \tag{4.12}$$

визначаємо рівень якості кожного з варіантів:

1: $К_1 = 1.344 + 0.729 + 2.589 + 1.001 = 5.663$

2: $К_2 = 1.344 + 2.264 + 0.661 + 1.001 = 5.27$

Як видно з таблиці 4.7, кращим є варіант А, для якого коефіцієнт технічного рівня має найбільше значення.

Табл. 4.6 Розрахунок вагомості параметрів ПП

| Параметри Xi | Параметри Xj | | | | Перша ітер. | | Друга ітер. | | Третя ітер. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | X1 | X2 | X3 | X4 | b_i | K_вi | b_i^1 | K_вi^1 | b_i^2 | K_вi^2 |
| X1 | 1,0 | 0,5 | 0,5 | 1,5 | 3,5 | 0,219 | 22,25 | 0,216 | 100 | 0,215 |
| X2 | 1,5 | 1,0 | 0,5 | 1,5 | 4,5 | 0,281 | 27,25 | 0,282 | 124,25 | 0,283 |
| X3 | 1,5 | 1,5 | 1,0 | 1,5 | 5,5 | 0,344 | 34,25 | 0,347 | 156 | 0,348 |
| X4 | 0,5 | 0,5 | 0,5 | 1,0 | 2,5 | 0,156 | 14,25 | 0,155 | 64,75 | 0,154 |
| Всього: | | | | | 16 | 1 | 98 | 1 | 445 | 1 |

Табл. 4.7 Рівень якості варіантів реалізації.

| Параметри | Реалізації функцій | Абсолютне значення параметра | Бальна оцінка параметра | Коефіцієнт вагомості параметра | Коефіцієнт рівня якості |
|---|---|---|---|---|---|
| X1(F1) | А, Б | 400 | 6.25 | 0,215 | 1.344 |
| X2(F3) | А | 2 | 2.8 | 0,283 | 0.729 |
| | Б | 5 | 8 | 0,283 | 2.264 |
| X3(F3) | А | 20% | 7.44 | 0,348 | 2.589 |
| | Б | 80% | 1.90 | 0,348 | 0.661 |
| X4(F2) | А, Б | 2000 | 6.5 | 0,154 | 1.001 |

## 4.6 Економічний аналіз варіантів розробки програмного продукту

Розрахуємо трудомісткість розробки ПП за різних умов реалізації за таблицею 4.8. Норми часу беремо відповідно до мов програмування, тому коефіцієнт $К_м = 1$. Якщо для розробки ПП використовують стандартні модулі чи пакети програм, стандартні програми, норми часу коригуються за допомогою коефіцієнта $К_{ст} =$ 0,6–0,8. У нашому випадку $К_{ст} = 0,7$. Якщо розробляють ПП, норму часу потрібно коригувати за допомогою коефіцієнта $К_{ст.м} = 1,2–1,6$. У нашому випадку $К_{ст.м} = 1,6$. У загальному випадку трудомісткість ПП розраховуємо за формулою (4.13):

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;$К_П$ – поправочний коефіцієнт;$К_{СК}$ – коефіцієнт на складність вхідної інформації; $К_М$ – коефіцієнт рівня мови програмування; $К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Табл. 4.8 - Трудомісткість варіантів

| Основні функції | Варіант реалізації функції | Рівень складності | Ступінь новизни |
|---|---|---|---|
| F1 (x1) | б | 1 | Б |
| F2 (x2) | б | 2 | В |
| F3 (x3) | а) | 1 | Б |
| | б) | 1 | А |

На основі даних з табл. 4.9 визначаємо трудомісткість кожного з варіантів реалізації ПП:

$T_I$ = (59,46 + 65,35 + 127,06) · 8 = 2014,96 людино-годин;

$T_{II}$ = (59,46 + 65,35 + 178,63) · 8 = 2427,55 людино-годин;

Більш високу трудомісткість має варіант ІІ.

В розробці бере участь один програміст з окладом 5 000 грн. Визначимо зарплату за годину за формулою (4.14):

$$СЧ = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;$T_m$ – кількість робочих днів на місяць;$t$ – кількість робочих годин в день.

$$СЧ = \frac{5000}{21 \cdot 8} = 29{,}76 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою (4.15):

$$СЗП = С_ч \cdot Т_i \cdot КД, \quad (4.15)$$

де $С_ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 29,76· 2014,96 · 1.2 = 71 962,85 грн.

II. $С_{ЗП}$ = 29.76· 2427,55 · 1.2 = 86 692,67 грн.

Табл.4.9 - Трудомісткість

| Основні функції | Варіант реалізації функції | Загальна норма, $T_{р(б)}$ | Коефіцієнт, $K_п$ | Коефіцієнт, $K_{сл}$ | Коефіцієнт, $K_{ст}$ | Коефіцієнт, $K_{ст.м}$ | Розрахункова працездатність, $T$ (чол./днів) |
|---|---|---|---|---|---|---|---|
| F1 (x1) | б | 52 | 1,021 | 1,00 | 0,7 | 1,6 | 59,46 |
| F2 (x2) | б | 52 | 0,72 | 1,00 | 0,7 | 1,6 | 65,35 |
| F3 (x3) | а) | 52 | 2,02 | 1,08 | 0,7 | 1,6 | 127,06 |
| | б) | 52 | 2,84 | 1,08 | 0,7 | 1,6 | 178,63 |

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 71\ 962,85 \cdot 0,22 = 15\ 583,39$грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 86\ 692,67 \cdot 0,22 = 19\ 072,39$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 5000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 5000 \cdot 0,2 = 12\ 000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 12\ 000 \cdot (1 + 0.2) = 14\ 400$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0,22 = 14\ 400 \cdot 0,22 = 3\ 168$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн. за формулою (4.16):

$$С_А = К_{ТМ} * К_А \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 20000 = 5\ 750 \text{ грн.}, \quad (4.16)$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою (4.17):

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 20000 \cdot 0,05 = 1150 \text{ грн.}, \quad (4.17)$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою (4.18):

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) * t_З \cdot К_В = (365 – 104 – 8 – 16) * 8 * 0,9 = 1706,4 \text{ годин}, \quad (4.18)$$

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} * N_С * К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,5 \cdot 0,2 \cdot 1,5682 = 267,6 \text{ грн.}, \quad (4.19)$$

де $N_C$ – середньо-споживча потужність приладу; $K_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою (4.20):

$$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 = 13\ 400 \text{ грн.} \quad (4.20)$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \quad (4.21)$$

$С_{ЕКС} = 14\ 400 + 3\ 168 + 5750 + 1150 + 267,6 + 13\ 400 = 38135,6$ грн.

Собівартість однієї машино-години ЕОМ за формулою (4.22) дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 43567,69/1706,4 = 22,3 \text{ грн/час.} \quad (4.22)$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, за формулою (4.23) складає:

$$С_М = С_{М\text{-}Г} * Т \quad (4.23)$$

I. $С_М = 22,3 \cdot 2014,96 = 45\ 031,5$ грн.;

II. $С_М = 22,3 \cdot 2427,55 = 54\ 134,37$грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 71\ 962,85 \cdot 0,67 = 42\ 215,1$ грн.;

II. $С_Н = 86\ 692,67 \cdot 0,67 = 58\ 084,1$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н}$

I. $С_{ПП} = 71\ 962,85 + 15\ 583,39 + 45\ 031,5 + 42\ 215,1 = 174\ 792,84$грн.;

II. $С_{ПП} = 86\ 692,67 + 19\ 072,39 + 54\ 134,37 + 58\ 084,1 =$ 217 983, 53 грн.;

## 4.7 Вибір кращого варіанта програмного продукту техніко-економічного рівня

Коефіцієнт техніко-економічного рівня розраховують за формулами(4.24, 4.25):

- $К_{тер1} = 5,663 / 174\ 792,84 = 3,2 \cdot 10^{-5}$; (4.24)
- $К_{тер2} = 5.27 / 217\ 983, 53 = 2,4 \cdot 10^{-5}$; (4.25)

Отже, найбільш ефективним є перший варіант реалізації програми.

## 4.8 Висновок до розділу 4

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 3,1 • 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Фреймворк – AngularJS;
- Створення стилів – CSS3;
- Програмне середовище - Webstorm.

# ВИСНОВКИ

Результатом даної роботи є аналіз існуючих віддалених систем зберігання, а також їх інтерфейсів. Продуктом даного аналізу є набір вимог до системи з точки зору користувача.

Серед розглянутих систем були і зовнішні сервіси та системи, і ті, що можна використовувати локально. Було виявлено, що за заданих умов, а саме для використання в сховища для невеликих компаній використання зовнішніх ресурсів - не найкраще рішення. Це пов’язано в першу чергу з безпекою та встановлення залежності. В такому випадку виходом є використання локальних сховищ. Однак їх функціонал та інтерфейс не відповідає користувацьким вимогам, а значній частині випадків їх використання не є безкоштовним. В зв’язку з цим з’являється потреба у створенні системи, яка відповідала функціональним та інтерфейсним потребам.

В роботі детально розглянуто процес створення клієнтської частини сервірсу. В зв’язку з тим, що користувачі навіть у межах локальної мережі бажають використовувати сервіс з різних пристроїв і не бажають встановлювати собі додаткове програмне забезпечення, з’являєтсья потреба у веб-інтрфейсі. У розділах роботи детально описано процес розробки цієї частини сервісу. Було проаналізовано вибір інструментів розробки, виходячи з поставлених вихідних умов. Головними з яких є:

1. Можливість завантаження файлу на власний пристрій або віддалений сервер
2. Переміщення по директоріям аналогічно руху по звичній файловій структурі комп’ютера
3. Функціонал пошуку
4. Здатність адміністратора надавати або блокувати доступ користувачів до системи

5. Засоби авторизації та реєстрації

В результаті досліджень в якості фреймворку було обрано AngularJS, так як він найкраще підходить для розробки клієнтської частини, що відповідала б поставленим вимогам. Було з’ясовано, що для створення зручного середовища розробки варто використати Gulp, а для стилізації - Bootstrap та Sass.

В якості результаті всіх досліджень була створена віддалена система зберігання корпоративних даних.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Kapadia, A. Implementing cloud storage with Openstack swift: Design, implement, and successfully manage your own cloud storage cluster using the popular OpenStack swift software. / Kapadia, A., Varma, S., Rajana, K. // United Kingdom: Packt Publishing, 2014 – pp. 134 – 136.
2. Arnold, J. OpenStack swift: Using, administering, and developing for swift object storage. / Arnold, J. // O'Reilly Media, 2014 – pp. 254 – 256.
3. Gormley, Elasticsearch: The definitive guide. / Gormley, C., Tong, Z. // O'Reilly Media, 2015 – pp. 134 – 138..
4. Seshardi S., AngularJS: Up and Running. // Seshadri S., Green B. - O'Reilly Media, 2014 – 322 с.
5. Williamson K., Learning AngularJS. // Williamson K. - O'Reilly Media, 2015 – 212 с.
6. Козловский П., Разработка веб-приложений с использованием AngularJS // Козловский П., Дарвин П. - ДМК Пресс, 2014 – 394 с.
7. Миковски М., Разработка одностраничных веб-приложений // Миковски М. - ДМК Пресс, 2014 – с. 512
8. Knol A., Dependency Injection with AngularJS // Knol A. - Packt Publishing, 2015 – с. 78
9. Toigo, J.W. The holy Grail of enterprise data storage. / Toigo, J.W. // Prentice-Hall, 1999 – pp. 64 – 69.
10. Ali Babar. Guidelines for Building a Private Cloud Infrastructure. / Ali Babar. // IT-Universitetet i København, 2012 – pp. 274 – 276.
11. Girish L S. Building Private Cloud using OpenStack. / Girish L S., Dr. H. S. Guruprasad. // International Journal of Emerging Trends & Technology in Computer Science, 2014 – pp. 134 – 138.
12. Marc Farley. Rethinking Enterprise Storage: A Hybrid Cloud Model. / Marc Farley. // Microsoft Press, 2013 – pp. 24 – 28.

13. Kapadia, Amar, Kris Rajana, Sreedhar Varma. OpenStack Object Storage (Swift) Essentials. / Kapadia, Amar, Kris Rajana, Sreedhar Varma // Packt Publishing, 2015 – pp. 184 – 187.
14. Single page apps in depth. – Режим доступу: http://singlepageappbook.com/goal.html. – Дата доступу : 27.04.2016.
15. Що таке SPA або односторінковий портал. – Режим доступу: http://www.calabonga.net/blog/post/141. – Дата доступу : 27.04.2016.
16. Офіційний сайт AngularJS. – Режим доступу https://angularjs.org/ – Дата доступу : 27.04.2016.