Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)

Кафедра обчислювальної техніки
(повна назва кафедри)

Пояснювальна записка

до кваліфікаційної роботи
_____магістра_____
(освітньо-кваліфікаційний рівень)

на тему: Онлайн сервіс для збереження інформації з оцінюванням степені схожості контенту

Виконав: студент _2_ курсу, групи 1КІ-18м
напряму підготовки (спеціальності)
123 – «Комп'ютерна інженерія»_
(шифр і назва напряму підготовки, спеціальності)

Котик А. М.
(прізвище та ініціали)

Керівник к.т.н., доцент Ткаченко О. М.
(прізвище та ініціали)

Рецензент к.т.н., доцент Карпінець В. В.
(прізвище та ініціали)

м. Вінниця - 2019року

Форма № Н-9.01

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень магістер
Напрям підготовки 123 – «Комп'ютерна інженерія»
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
_____проф., д.т.н. Т. Б. Мартинюк

«___»_____201_ р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Котику Антону Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Онлайн сервіс для збереження інформації з оцінюванням степені схожості контенту»
керівник роботи Ткаченко Олександр Миколайович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу
від"___"_____20__року №___
2. Строк подання студентом роботи_____

3. Вихідні дані до роботи: Формати файлів, що використовуються для зберігання, відео, аудіо, зображень, без обмежень. Технології, моделі побудови front-end, технології побудови back-end.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Аналіз існуючих технологій створення web-додатків і методів збереження та оробки інформації в мережі інтернет. Розробка web-додатка. Розрахунок економічної доцільності створення онлайн сервісу для збереження інформації. Розробка технічної документації та порівняння онлайн сервісу з аналогічними продуктами.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Модульна структура Angular додатка. Компонент Angular. Клієнт-серверна архітектура. MVC архітектура. Структурна схема розробленого додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Ткаченко О. М., к.т.н., доцент.		
4	Глущенко Л. Д. к.т.н., доцент.		

7.Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	03.09-15.09.19	
2	Огляд існуючих методів створення web-додатків	04.09-15.09.19	
3	Аналіз та вибір методів і технологій створення web-додатків	02.10-10.10.19	
4	Економічне обґрунтування роботи	11.10-20.10.19	
5	Розробка web-сервісу	21.10-30.10.19	
6	Проведення тестування створеного сервісу, та порівняння з провідними продуктами.	01.11-15.11.19	
7	Створення технічної документації	16.11-30.11.19	
8	Оформлення пояснювальної записки та ілюстративного матеріалу	02.12-08.12.19	
9	Аналіз виконання роботи, висновки, додатки	07.12-08.12.19	
10	Перевірка якості виконання магістерської роботи та усунення недоліків	09.12.19	

Студент _____ Котик А. М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ткаченко О. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній роботі було розглянуто актуальні на сьогодні технології розробки web-додатків. Проведено аналіз доцільності використання технологій для програмних продуктів різного розміру та складності, виявлено переваги та недоліки використання різних технологій, мов та фреймворків, в різних ситуаціях. Розглянуто технології та апаратні методи збереження та обробки інформації в мережі інтернет, проаналізовано принципи побудови систем, що забезпечують цілісність даних.

Розглянуто моделі побудови web-додатків та моделі функціонування додатків на web-серверах. Розглянуто кроки, що покликані підвищити продуктивність web-додатку, не змінюючи архітектуру web-додатка.

Розроблено web-сервіс для збереження даних з обмеженням доступу до файлів, та розділенням даних, що завантажуються за певною ознакою.

Розроблено інструкцію для системного адміністратора, яка покликана полегшити розгортання системи на web-сервері, а також інструкція користувача, що дозволить користувачеві отримати інформацію про основи використання сервісу.

Проведено тестування продукту на наявність помилок, та відповідність рекомендаціям до написання коду. Проведено аналіз та порівняння отриманого продукту з провідними сервісами, призначеними для виконання схожих завдань.

ANNOTATION

The current technologies of web-application development were considered in this paper. An analysis of feasibility of using technologies for software of different sizes and complexity was conducted, the advantages and disadvantages of using different technologies, languages and frameworks in different situations were identified. The technology and hardware methods for storing and processing information in the Internet are considered, the principles of the construction of systems that provide the integrity of the data were analyzed.

The models of building web-applications and models of application functioning, based on web-servers, were considered. The steps were examined to improve the web application perfomance without changing the architecture of the web application.

A web-service for storing data with restricted access to files, and the separation of data, downloaded by a certain characteristic, was developed.

A system administrator manual was developed to facilitate deployment of the system on a web server, as well as a user manual, which would allow to obtain information about the basics of using the service.

Product testing for errors and compliance with the recommendations for writing the code was conducted . The analysis and comparison of the product with the leading services, designed to perform similar tasks, was carried out.

ЗМІСТ

ВСТУП

У житті сучасної людини значну роль відіграють IT-технології, зокрема інтернет. На сьогоднішній день інтернет став доступним та швидким, що в свою чергу надає значно більше можливостей та простору для web-додатків, ми використовуємо web-технології щоденно і повсякчасно, для роботи, навчання чи будь-яких інших цілей.

Сьогодні сфера web-технологій пропонує безліч різних сервісів онлайн, деякі з них дозволяють переглядати відео, або прослуховувати музику не завантажуючи її на комп'ютер, інші дозволяють зберігати свої файли не витрачаючи на це дискового простору свого комп'ютера, багато з них повністю або частково повторюють десктопні додатки, надаючи їм специфічних можливостей, як синхронізація між кількома пристроями, це можливо завдяки мережі інтернет. Web-технології відіграють все більшу роль в житті людини, і дають їй все більші можливості, а тенденція збільшення швидкостей, здешевлення інтернету тільки сприяє цьому.

Сучасний web-працює над забезпеченням зручності використання web-додатків, наближеної до десктопних додатків, над побудовою дружніх інтерфейсів, а також над розширенням функціоналу та інтеграцією з онлайн джерелами, які можливо використовувати через мережу та не доцільно зберігати на одній машині. В цій роботі розроблено web-сервіс, який побудований як односторінковий web-додаток, що дозволяє зберігати свої файли онлайн, не витрачаючи місця на комп'ютері, дозволяє зручно групувати файли по їх типу, дає доступ до них з будь-якого пристрою, що має браузер. Особливостями сервісу є забезпечення потокового відтворення аудіо та відео, доступ до загальнодоступних файлів, забезпечення формування індивідуального профілю користувача використовуючи поведінкові дані такі як прослухані композиції, переглянуті зображення, контент який сподобався то що, а також формування індивідуальних рекомендацій на базі отриманої інформації, а також визначення схожості контенту.

Метою роботи є розширення функціональних можливостей web-сервісу для роботи з мультимедійною інформацією за рахунок введення додаткових функцій, об’єднання файлів у певні категорії для зручності роботи користувача та створення індивідуальних рекомендацій контенту.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

– провести аналіз існуючих на ринку провідних онлайн-сервісів розробок з аналогічними функціями;

– розробити web-сервіс у вигляді односторінкового додатку, з авторизацією;

– надати можливість завантажувати файли та групувати їх у певні категорії за призначенням файлів, плейлисти, колекції, відеотеки, галереї;

– для кожної з категорій розробити можливість попереднього перегляду, для аудіофайлів це має бути аудіопрогравач, для галереї – слайдер;

– забезпечити обмеження доступу до файлів та додати можливість перегляду загальнодоступних колекцій, розробити інтелектуальну рекомендацію контенту;

– провести тестування розробленого продукту та його порівняння з провідними аналогами;

– розробити технічну документацію на розроблений програмний комплекс.

Об’єкт дослідження – процес збереження й оброблення мультимедійної інформації у web-сервісах.

Предмет дослідження – методи та програмні засоби розробки web-додатків для збереження й оброблення мультимедійної інформації у web-сервісах.

Методи криптографії – для розрахунку криптографічних хешів, під час завантаження даних на сервер, для виявлення і усунення дублікатів завантажуваних данних.

Методи ООП – для реалізації програмного продукту а також методів визначення подібності і виявлення дублікатів контенту і реалізації алгоритмів що дозволяють побудувати індивідуальний профіль користувача, що буде відображати його вподобання.

Методи теорії штучного інтелекту – для визначення степені схожості контенту, формування індивідуального профілю користувача на базі поведінки, визначення вподобань користувача, та забезпечення інтелектуального підбору контенту.

Знайшов подальшого розвитку метод визначення подібності контенту, в якому, на відміну від існуючих використовується перцептивний хеш для порівняння контенту, що дозволило використати цей метод для обробки не тільки зображень, а й мультимедійного контенту, і таким чином збільшити час присутності користувача та розширити фунціональні можливості сервісу.

Практичною цінністю впроваджених методів являється:

– Розроблено алгоритм для побудови персонального відбитку користувача для визначення вподобань шляхом аналізу поведінки.

– Розроблено алгоритм для побудови списків рекомендованого контенту.

– Розроблено програмний засіб для реалізації методів та алгоритмів.

Основні положення роботи обговорювалися і були опубліковані у рамках таких конференцій:

Молодь в науці: дослідження, проблеми, перспективи (МН-2020) [1].

1 АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ СТВОРЕННЯ WEB-ДОДАТКІВ ТА МЕТОДІВ ЗБЕРЕЖЕННЯ І ОБРОБКИ ІНФОРМАЦІЇ В МЕРЕЖІ ІНТЕРНЕТ

1.1 Сучасні технології розробки Web-додатків

Сучасний web надзвичайно швидко розвивається, сьогодні термін сайт вже не може охопити ту величезну кількість різного роду продуктів, що створюються у сфері web-технологій, це поняття стало занадто вузьким, замість нього використовується поняття web-додаток.

У побудові web-додатків існує три основні моделі взаємодії і безліч архітектур, найбільш поширена модель клієнт-серверна, та вона не єдина, існують додатки побудовані на основі однорангових мереж, коли кожна машина одночасно виступає і клієнтом і сервером, також існує модель, що поєднує дві попередніх, гібридна модель, в ній передбачено багато машин, які можуть спілкуватись між собою, але управління такою взаємодією значно ускладнюється при збільшенні машин, тому для спрощення управління додають координаційний сервер.

Для типового web-додатку загальноприйнятим є використання клієнт-серверної моделі тому розглянемо саме її, архітектур додатка, побудованих на цій моделі існує дуже багато, вони можуть використовувати різні варіанти побудови сервера, використовувати різні бази даних або навіть декілька, але основа взаємодії клієнта і сервера у всіх таких додатках залишається однаковою. Клієнт-серверна архітектура, зображено на рисунку 1.1, передбачає існування клієнта в ролі якого виступає зазвичай браузер і (front-end) частина додатка, та у якості web-клієнта можуть виступати також десктопні додатки, ігри тощо. А також передбачається існування сервера, в ролі якого виступає віддалена машина з іншою (back-end) частиною додатка. Взаємодія між клієнтом та сервером відбувається за допомогою протоколу HTTP. Для того, щоб передати якусь інформацію на сервер, клієнт відправляє

йому запит. Запити, ще їх називають HTTP-методами можуть бути різних типів, наприклад, існують такі HTTP методи :

GET, POST, PUT, DELETE, OPTION, HEAD, PATCH, TRACE, CONNECT [2]. Сервер отримує запит і обробляє його, в залежності від того на що він направлений, видаляє, оновлює дані, звертається до бази даних або до інших серверів, після чого формує відповідь і відправляє її клієнту, у відповіді окрім даних міститься й службова інформація, наприклад, код, що позначає статус відповіді, так при неможливості виконати якісь операції сервер може відправити відповідь з кодом помилки.

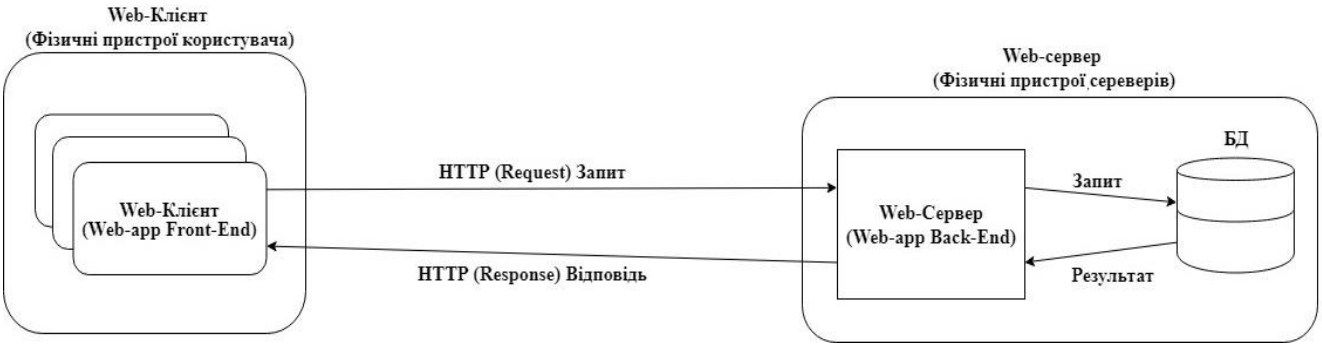


Рисунок 1.1 – Типова схема web-додатка

Клієнт – частина додатку, що відповідає за відображення даних, відображення інтерфейсу, взаємодію з користувачем та спілкування з сервером.

Сьогодні клієнтська частина відповідає не тільки за відображення статичних файлів або підставлення даних в шаблони, а й за попередню обробку даних перед відправленням на сервер, в тому числі валідацію, за пост-обробку отриманих даних з сервера. Всю логіку зв’язану з відображенням даних та елементів інтерфейсу перенесено на клієнтську частину [3].

Основа сторінок, все ще будується на html, що правда вже на версії стандарту HTML 5, для стилізації, як і раніше використовуються таблиці

стилів CSS, що розвинулись до стандарту CSS3. Значну роль у побудові логіки клієнта відіграє JavaScript, який використовується як в чистому вигляді, так і у вигляді бібліотек таких як JQuery, React, не менш популярними є і фреймворки написані на JavaScript, такі як, AngularJS, Vue.js, Angular 2+. JavaScript мова, яка стрімко розвивається і сьогодні ми використовуємо стандарт ECMAScript 6, що надає дійсно надзвичайні можливості для написання інтерфейсів будь-якої складності.

Сервер – частина web-додатка, що відповідає за обробку даних, взаємодію з базою даних, взаємодію з іншими web-сервісами, а також за відповіді на запити, отримані від клієнтської частини.

На відміну від клієнтської частини, де майже монополістом є JavaScript, для написання серверної частини використовують набагато більший перелік мов, технологій та підходів. Основними серверними мовами програмування є такі мови як Java, Python, JavaScript, C#, Rubi, PHP, для кожної з яких є свої фреймворки, бібліотеки та модулі.

1.1.1 Технології розробки web-серверів

Для побудови серверної частини, так званого API, використовується багато різних підходів, тобто немає якоїсь загальноприйнятої типової архітектури, яка б підійшла в будь-яких випадках, та є деякі загальні принципи як, наприклад, MVC (Model View Controller) та REST(Representational State Transfer), CRUD (Create, Reade, Update, Delete).

Вибір архітектури, стеку технологій та мови за допомогою яких буде розроблятись продукт дуже сильно залежить від специфіки продукту, складності обчислення, що мають проводитись з даними, розміру і функціоналу додатка [4].

В світі розробки програмного забезпечення, до якого також належить і розробка web-додатків, можна виділити кілька рівнів абстракції, а саме:

– Чиста мова – використовується одна з мов програмування в чистому вигляді, що дає максимум гнучкості, на чистій мові написані найбільші та найскладніші проекти.

– фреймворки – напівготовий продукт, інструментарій, що дає деякі блоки для побудови, пропонує типові архітектури, дозволяють пришвидшити і удешевити розробку, на них пишуться проекти середньої складності.

– коробочні рішення (CMS) – тут мова вже йде не про побудову додатка, а налаштування готового рішення, дає мінімум гнучкості підходить для маленьких типових проектів, з маленькими навантаженнями.

Для побудови back-end використовуються такі мови програмування:

– JavaScript (Node.js)

– Java (Spring)

– C# (ASP.NET core)

– Python (Django)

– Ruby (Ruby on Rails)

– PHP (Laravel, Symfony)

Node.js є основним способом запуску JavaScript за межами браузера. Додана вся специфікація ES6 досить швидко розвивається Node.js має каркаси для створення швидких API, серверів, настільних додатків, та навіть роботів, а велика спільнота створює різноманітні модулі, які тільки можна уявити. Представниками фреймворків Node являються такі як: Express, Koa, Next, Nodal.

PHP в першу чергу web-мова і тому має великий вибір web-фреймворків. Завдяки чудовій документації та функціям, Laravel сформував активну спільноту. Також вийшла третя версія Zend Framework, що стало значним оновленням для цього бізнес-орієнтованого фреймворка. Symfony також має багато змін, що робить його кращим вибором в якості рішення для повного стеку.

Ruby, "Rails" як і раніше залишається головним фреймворком.

Версія 5.0 була випущена в 2016 році, що забезпечила підтримку web-сокетів, API-інтерфейс та багато іншого. Sinatra також хороший вибір для невеликих додатків.

Python має свою власну комбінацію full-stek/minimal фреймворка у вигляді Django and Flask. Django 1.10 містить в собі повно-текстовий пошук і перероблений рівень проміжного програмного забезпечення.

Має надзвичайно розвинену екосистему з великим вибором фреймворків, також розвивається спільнота надбудови у вигляді Scala. Та поки рано говорити про майбутнє цієї мови, та багато фреймворків вже підтримують її. Популярними рішеннями на java є фреймворки Play, Spark, Spring.

C# – мова розроблена компанією Microsoft, дуже схожа на java, і призначена для тих самих задач, орієнтована в основному на бізнес. Має свій фреймворк Core, що набуває популярності.

Для проектів, що мають витримувати великі навантаження, бути прогнозованими, стабільними, або мають відповідати таким вимогам в подальшому хорошим вибором стане Java, C#, або ж Python. Тому що вони орієнтовані для побудови саме складних проектів. Для простіших проектів можна використовувати PHP, NodeJS, або ж Rubi. Вони доволі швидкі але не мають достатньої строгості написання [5].

Сьогодні для підвищення продуктивності web-систем в цілому можуть використовуватись різні структури функціонування серверів.

Наприклад для підвищення продуктивності можна виконати зміни, які напряму не стосуються архітектури web-додатка та значно зменшують навантаження на окремі його частини, що при великій завантаженості дає більш стабільну роботу.

У випадках коли в проектів слабким місцем стає база даних її можна виносити на окрему, більш потужну машину, зображено на рисунку 1.2

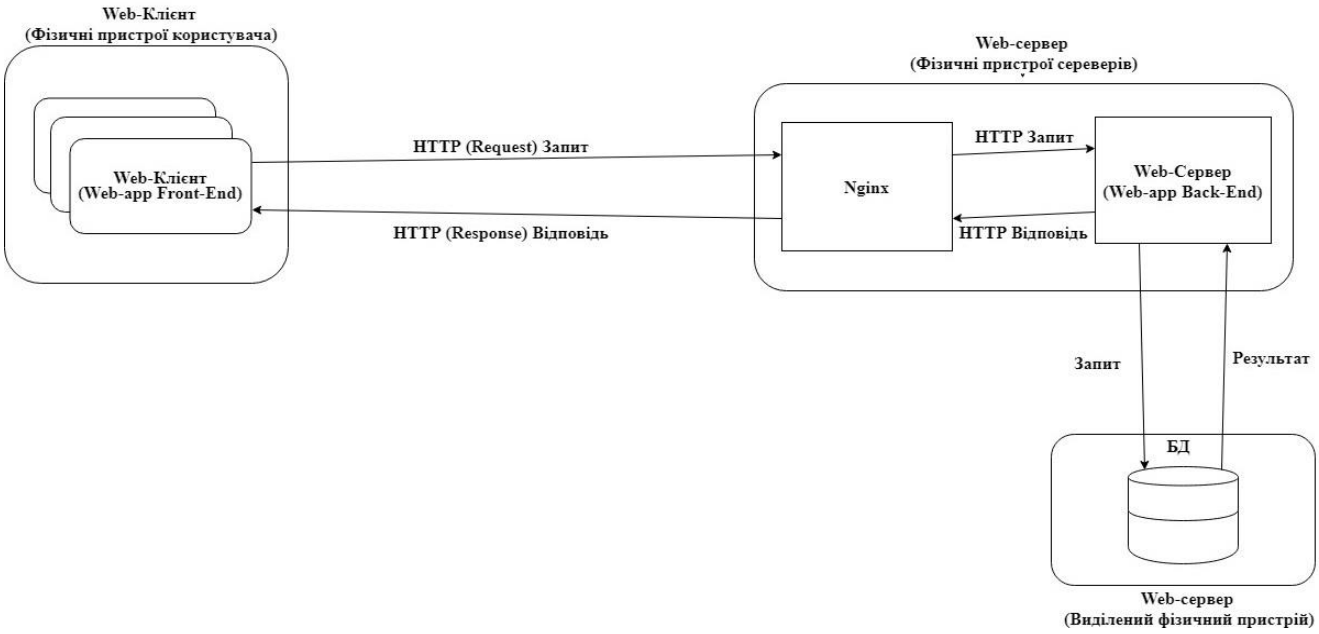


Рисунок 1.2 - Виділення бази даних

Це дозволяє використовувати всю її потужність лише на базу даних, не навантажуючи її ще роботою web-сервера додатку, отриманням та обробкою запитів.

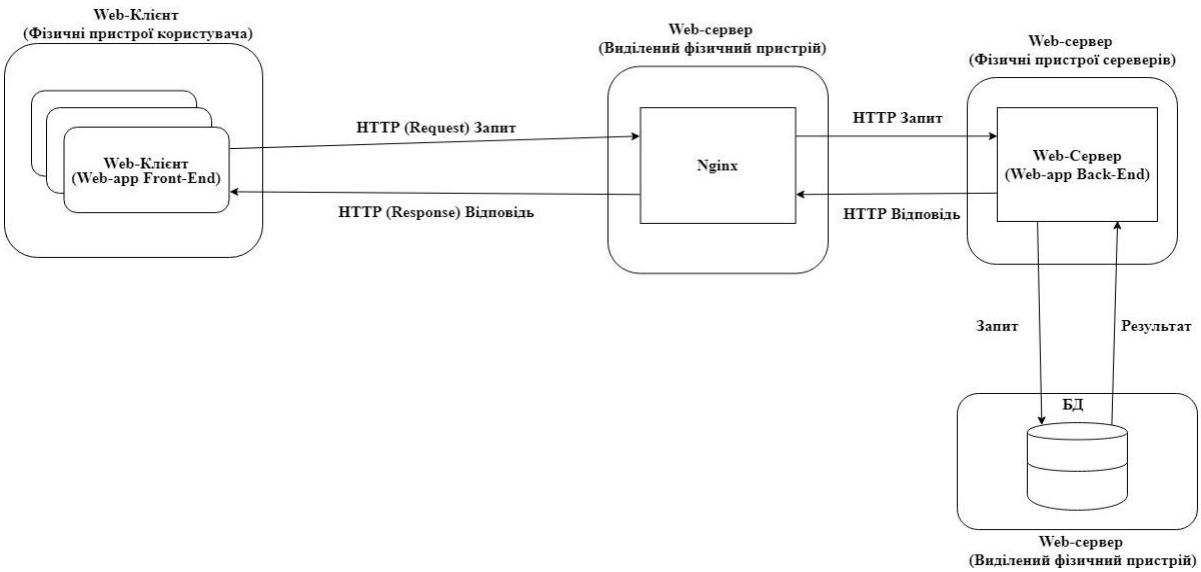


Рисунок 1.3 - Виділення Nginx на окрему машину.

Таке розділення усіх складових дозволяє використовувати для роботи кожної з них свою машину і дає деякий простір для вибору потужності такої

машини, ще одним плюсом для розвантаження back-end є можливість налаштування Nginx сервера і він буде сам віддавав статичні файли, таким чином API не буде навантажуватись зображено на рисунку 1.3

Та все ж таки можлива ситуація коли запитів буде надто багато і back-end справлятись не буде, а потужність машини, на якій він працює безкінечно збільшувати не вдасться для цього використовують схему зображену на рисунку 1.4

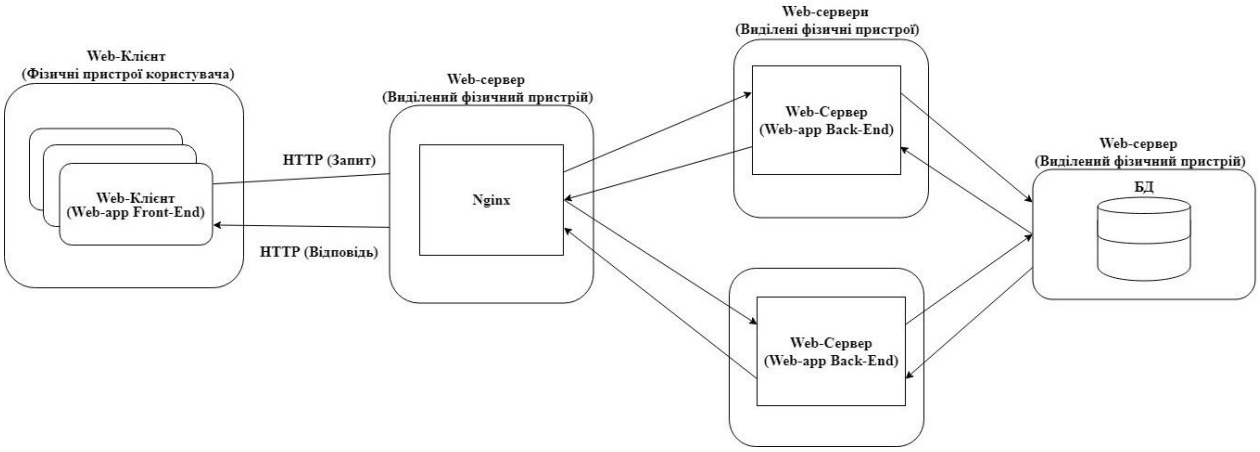


Рисунок 1.4 - Використання кількох Back-end

При такому підході можна настроїти Nginx для віддавання статичних файлів, а також для балансу навантажень і тоді він буде рівномірно завантажувати запитами усі back-end. Та з таким підходом виникають і складності, потрібно знати на який з серверів пішов запит [6].

1.1.2 Технології розробки web-клієнтів

Для побудови клієнтської частини сьогодні можуть бути використані різні підходи в залежності від складності задач та форми відображення даних на сторінках. Для простої візитки можна використати html сторінки з підключеними стилями, а якщо потрібно додати інтерактивності то, це можна зробити використовуючи мову JavaScript у чистому вигляді, або ж використати одну з бібліотек, написаних на ньому.

Оскільки багато сайтів використовують таку схему побудови клієнту, а JavaScript бібліотеки стали доволі популярними, та їх розмір часто доволі великий, тому скрипти популярних бібліотек розміщуються не на сервері, де працює beck-end, а на CDN-сервері, який дозволяє завантажити бібліотеку з географічно найближчого до користувача сервера.

Для побудови інтерфейсів, для обробки даних перед відображенням їх користувачу широко використовуються JavaScript-бібліотеки та фреймворки, найяскравішими представниками на сьогодні є JQuery, Angular, React [7].

Також широкого розповсюдження набула технологія Ajax, що дозволяє обмінюватись даними з сервером у фоні без перезавантаження сторінки і підставляти дані на льоту. Це дозволило зробити інтерфейс web-додатка набагато зручнішим. В результаті популярними стали SPA (Single Page Application), що дозволили побудувати інтерфейси, які по зручності не поступаються десктопним додаткам [8]. Для побудови інтерфейсу складного web-додатка хорошим вибором буде Angular 2+ або ж бібліотека React.

Angular створювався як enterprise-фреймворк і призначений для написання складних додатків з багаторівневою архітектурою, він має багато гнучкого та потужного функціоналу, для написання програм будь-якого напрямку, а його модульна архітектура спрощує тестування, супроводження, а також модернізацію додатку.

Використовуючи бібліотеку React також можна побудувати складні та гнучкі та продуктивні програми, але на відміну від Angular для написання складних програм потрібні грунтовні знання в побудові архітектури додатків. React не накладає ніяких обмежень на архітектуру додатків, а також може використовуватись з іншими бібліотеками або JavaScript-фреймворками в тому числі Angular. Таким чином, помилка в архітектурі, може призвести до проблем з продуктивністю, стабільністю роботи та багатьох інших.

Основою для побудови додатків на React є компоненти, а особливістю побудови інтерфейсів є поєднання html і JavaScript в одному файлі.

1.1.3 Бази даних у web-додатках

Для збереження даних у web-додатках сьогодні існують кілька підходів, в залежності від призначення та потреб додатку.

Зараз немає такої однозначності як колись, тепер, обираючи сховище для зберігання даних, архітектор не обмежується вибором лише між тим яку реляційну базу вибрати, останнім часом великої популярності набули, так завані, NoSQL бази даних, в першу чергу, через свою продуктивність бо вони призначені для зберігання великих об’ємів даних і вільність формату збереження даних.

Документо-орієнтовані бази даних такі як MongoDB, CouchDB, а також швидкі бази, що звичайно зберігають дані в форматі ключ-значення або ж колонкові бази, живуть в оперативній пам’яті представником таких «сховищ» є Redis та Casandra.

Вибір бази зумовлюється особливостями проекту. Для більш-менш складних проектів звичайною практикою є використання кількох баз в одному проекті зазвичай одна з них реляційна, яка використовується для нормалізованої інформації і коли потрібні різні складні вибірки «Join», транзакції і тому подібне.

Так як реляційні бази мають спеціальну, дуже потужну та гнучку мову запитів SQL. Паралельно використовуються storage типу Redis, Memcache, які значно швидші і використовуються для кешування та для операцій, в яких, реляційні бази поступаються в швидкості. NoSQL зазвичай не можуть робити складних вибірок, та прості CRUD операції зазвичай доступні [11].

1.2 Збереження та передача інформації в мережі інтернет

Web-додаток це все ж таки додаток, він також потребує збереження даних, проте зазвичай web-додатками користуються тисячі, чи навіть мільйони людей, тому існує доволі багато різних методів організації збереження даних на сервері.

1.2.1 Апаратні методи збереження даних

Web-додаток – це річ, якою користуються дуже багато людей, завдяки цьому навіть якщо кожен з них захоче зберегти кілька мегабайт в результаті загальна кількість інформації, яку потрібно зберігати досягає дуже великих обсягів. Більше того, кількість інформації в інтернет-просторі з кожним роком зростає в експоненційній прогресії. Тому до апаратури, що займається збереженням даних висуваються особливі вимоги, це мають бути швидкі і надзвичайно об’ємні носії.

На сьогодні цим критеріям, з деякими обмовками, відповідають 3 типи носіїв: це накопичувачі на жорстких магнітних дисках (HDD), так звані, твердотілі накопичувачі на флеш пам’яті (SSD), а також гібридні рішення (SSHD), в основі якого лежить компромісне рішення, що об’єднує HDD та SSD кеш відносно великого розміру, значно більшого в порівнянні з HDD. В ньому зберігаються часто використовувані файли, наприклад файли, яких потребує ОС для запуску, що дозволяє значно пришвидшити запуск ОС.

Існують і інші накопичувачі такі як оптичні диски, та магнітні стрічки та в сучасному web і такі носії не мають місця.

Накопичувачі на жорстких магнітних дисках використовують ефект магнітного запису, тобто існують жорсткі диски, що розбиті на сектори і сегменти, які намагнічуються або розмагнічуються завдяки чому зберігають дані. В сучасних накопичувачах використовується масив з кількох дисків.

Швидкість роботи такого пристрою напряму залежить від частоти з якою обертається шпиндель масиву з магнітними дисками, такий пристрій має затримку при старті, поки диск розкрутиться до робочих обертів, а також при зчитуванні або записі, час доки головки перемістяться до сектора з якого має проводитись зчитування.

Для збереження даних в сучасних системах використовуються такі жорсткі диски:

– 5200 об\хв так звані тихохідні диски, встановлюються в основному в ноутбуки так як споживають менше енергії, а також в деякі настільні комп’ютери. Такі диски мають довгий строк служби, та володіють відносно скромною продуктивністю.

– 7200 об\хв найпоширеніші диски мають дещо менший та прийнятний строк служби, споживають більше енергії та не критично більше, і мають доволі високі показники продуктивності. Установлюються на більшість настільних ПК.

– 10 – 15000 об\хв найпродуктивніші диски мають маленький строк служби, споживають багато енергії, піддаються підвищеному нагріву через, що потребують хорошого охолодження системи, такі диски установлюються на високопродуктивні серверні масиви де потрібна максимальна продуктивність, і за рахунок використання RAID-масивів резервного копіювання і реплікації вихід з ладу одного носія не призводить до втрати даних.

SSD – це твердотілий енергонезалежний накопичувач без рухомих механічних частин побудований з використанням флеш пам’яті, пам'ять флеш пам'ять, що використовується в сучасних SSD має обмежену кількість циклів запису та читання, через що використовуються контролери для контролю запису і читання, а також оптимізації розміщення інформації для забезпечення максимального строку служби. Окрім обмеженої кількості записів SSD має ще один недолік це його ціна, такі накопичувачі значно дорожчі за HDD.

Не зважаючи на всі свої недоліки SDD має вагому перевагу через що і використовується в комп’ютерній техніці, це швидкість читання та запису. Найбільший виграш у порівнянні з HDD твердо тілі накопичувачі мають при роботі з маленькими файлами.

Для прикладу швидкість читання на блоках по 4кб.

– HDD повільніше в 94 раза (0.68 МБ/с- HDD, 63.6 МБ/с – SSD) , в порівнянні з SSD

Операція запису на блоках 4кб.

– HDD повільніше в 178 раз (0.78 МБ/с – HDD, 139 МБ/с - SSD), в порівнянні з SSD

Виробники запам’ятовуючих пристроїв сьогодні працюють над мінімізацією недоліків SSD тому в подальшому використання цих швидких носіїв витіснить HDD. [12]

1.2.2 Програмна організація збереження даних

В залежності від розміру web-додатку, кількості даних які мають зберігатись та частоти їх використання, можуть застосовуватись різні підходи до збереження даних, для простих сайтів візиток можна використати топологію в якій сервер бази даних, проксуючий сервер nginx та back-end сайту знаходяться на одній машині.

Для великих і навантажених серверів використовується системи зберігання даних (storage).

По частоті використання даних, системи зберігання даних можна розділити на системи короткочасного збереження (online storage), збереження середньої тривалості (near-line storage) та системи довгострокового збереження (offline storage).

До першої категорії можна віднести HDD або ж SSD будь-якого персонального комп’ютера. Друга і третя категорії – це зовнішні системи збереження даних DAS (Direct Attached Storage), які можуть являти собою зовнішні по відношенню до комп’ютера масиви дисків (Disk Array). Їх в свою чергу також можна поділити на просто масиви і масиви з керуючим контролером.

Зовнішні системи збереження бувають трьох типів DAS (Direct Attached Storage), SAN (Storage Area Network) і NAS (Network Attached Storage). Системи SAN і NAS зображені на рисунку 1.7.

В SAN з системою збереження даних зв’язані самі сервери через мережу області збереження даних SAN. У випадку NAS – мережеві сервери

зв’язані через локальну мережу LAN з загальною файловою системою в RAID.

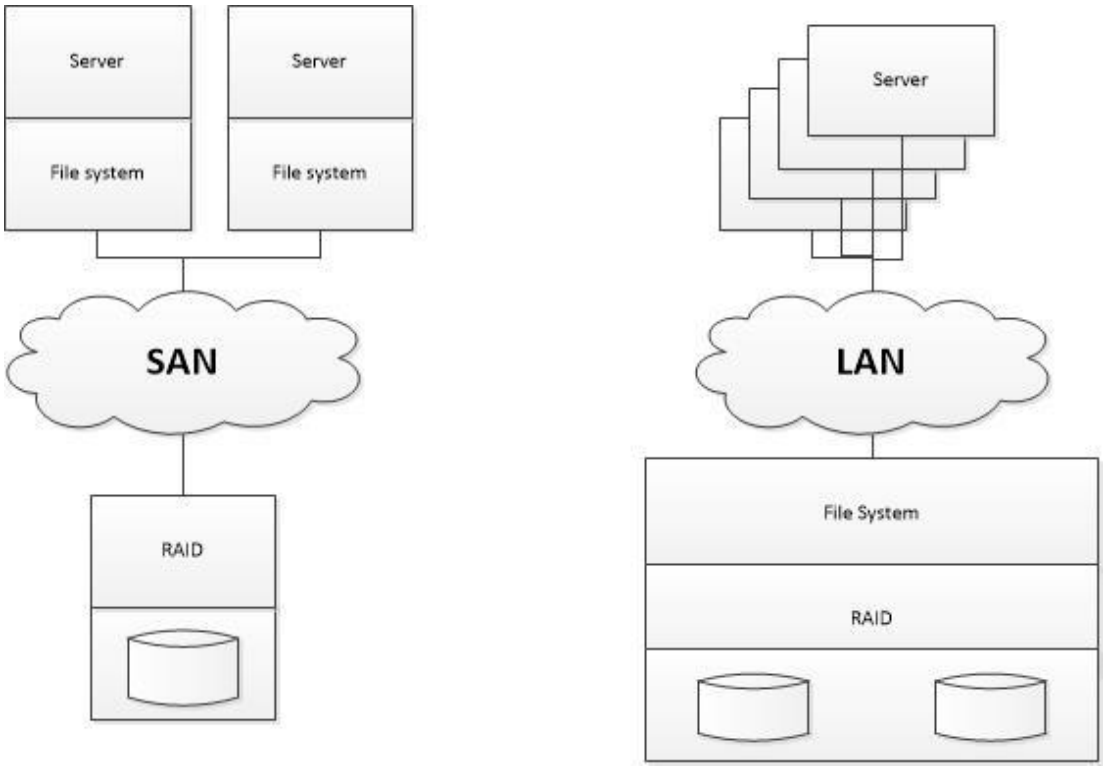


Рисунок 1.7 – Системи збереження даних SAN і NAS

Для побудови масивів дисків використовуються як HDD, так і SSD, проте які б надійні не були ці пристрої все ж зі збільшенням кількості дисків в масиві шанс того, що один з них вийде з ладу збільшується. Втрата інформації для web-додатка критична точка, якої допустити не можна, для запобігання такої ситуації використовується технологія для об’єднання дисків в RAID (Redundant Array of Independent Disks) – масив незележних дисків з надлишковістю зберігання даних. Основою технології RAID як сказано в назві є надлишковість, тобто дублювання і розділення даних. Існують декілька рівнів RAID основних їх 6 та існують також різного роду варіації.

- RAID-0 – може містити довільну кількість дисків, більше двох і так далі.
- RAID-1– один блок даних записується на всі диски.

– RAID-10 є комбінацією RAID-1 і RAID-0: де масиви RAID-1 об’єднуються в RAID-0.

– RAID-5 – слугує для роботи з трьома і більше дисків.

– RAID-6 – потребує, щоб було чотири і більше дисків.

У web-додатках широко розповсюджені різні бази даних, для доступу до кожної з них можна використати API бази даних або ж використати популярні в останній час ORM [13].

ORM або Objectrelational mapping *(укр. Об’єктно реляційне представлення)* — це технологія програмування, що дозволяє перетворити несумісні типи полів в ООП, наприклад між об’єктами програмування та моделями бази даних. ORM використовується для спрощення збереження об’єктів в реляційну базу даних та їх отримання, при цьому сама ORM турбується про перетворення даних між двома несумісними станами. Більшість ORM-інструментів в значній мірі покладаються на метадані бази даних і об’єктів, тому об’єктам нічого не потрібно знати про структуру бази даних, а базі даних – нічого про об’єкти, і як дані організовані в додатку. ORM забезпечує повне розділення задач в добре спроектованих додатках, при яких і база даних, і додаток можуть працювати з даними кожен в своїй вихідній формі.

Головною особливістю ORM є відображення, яке використовується для звязування об’єкта з його даними в БД.

ORM створює віртуальну схему бази даних в пам’яті і дозволяє маніпулювати даними вже на рівні об’єктів.

Відображення показує як об’єкт і його властивості зв’язані з одною або з кількома таблицями і їх полями в базі даних. ORM система використовує інформацію з цього відображення для управління процесом перетворення даних між базою і формами об’єктів, а також для створення SQL-запитів для автоматичної вставки, вибірки, оновлення і видалення даних у відповідь на зміни, які додаток вносить в ці об’єкти [14].

1.2.3 Особливості збереження та обробки великих обсягів даних

Для забезпечення стабільного та неперервного функціонування високонавантажених систем, які мають зберігати великі об’єми даних, а також мають великі навантаження на запис або зчитування цих даних, використовують декілька підходів. Деякі з них передбачають збільшення потужності серверного обладнання за рахунок установки продуктивніших комплектуючих, також можливе збільшення продуктивності за рахунок додавання нових серверів, які працюють паралельно, також виконують рознесення функціоналу, так розміщують проксуючі сервера Nginx, на окремих машинах, back-end теж на виділених машинах, бази даних використовують системи збереження даних, що також мають свої машини.

Високе навантаження на систему збереження даних, а саме часте зчитування та запис інформації не сприятливо ввідображається на довговічності носіїв тому для зменшення такого навантаження використовують кешування для збереження даних, які часто запитуються [12].

1.2.4 Потокові сервіси

Окремою гілкою в розвитку web-додатків стали потокові web-сервіси, це додатки, що дозволяють не завантажувати відео, або аудіофайли, а прослуховувати їх використавши при цьому лише браузер і з’єднання з інтернетом. Все це стало можливим завдяки збільшенню швидкості та доступності інтернету.

Одним з найяскравіших сервісів потокового відео являється продукт під назвою YouTube від компанії Google, а потокового аудіо Spotify, їх популярність зумовлена не лише можливістю переглядати відео без завантаження на комп’ютер в реальному часі, а й інтелектуальним підбором контенту для кожного з користувачів, який заснований на аналізі контенту, який вже був переглянутий [15,16].

1.2.5 Сервіси збереження даних онлайн

Також представниками сучасних web-додатків є сервіси збереження даних, це web-сервіси, що дозволяють зберігати користувацькі файли. Синхронізуючи їх між кількома комп'ютерами та мати доступ до файлів з різних комп'ютерів, іншими словами замість запису інформації на диск її можна зберегти в такому онлайн-сховищі.

Такі сервіси розвинулися завдяки збільшенню швидкості інтернету, тепер завантажити або навіть скачати файл не є великою проблемою.

Одним з представників сервісів збереження інформації є Dropbox, він дозволяє встановити на комп'ютер програмне забезпечення, в результаті встановлення якого, отримаємо спеціальну папку, яка і буде синхронізуватись між всіма комп'ютерами, на яких установлений Dropbox. Проте потреба установлювати додаткове програмне забезпечення і наявність папки на комп'ютері, яка займає місце на всіх комп'ютерах тобто фактично доступ не до одних даних, а дублювання їх створює деякі незручності користування таким сервісом. Більше того якщо доведеться змінити користувача, а в цій папці буде доволі багато інформації, виявиться, що синхронізація таких маніпуляцій доволі затратна, потрібно буде видалити всі дані користувача, який розлогінився з комп'ютера, а потім скачати всі дані нового користувача.

Таким самим принципом користується OneDrive від корпорації Microsoft.

Сервіс для збереження користувацьких файлів пропонує і компанія Google продукт називається Google Drive від двох інших його відрізняє те, що дані зберігаються повністю онлайн і вам не потрібно мати на комп'ютері деяку папку, яка буде займати місце. Доступ до файлів ви можете отримати з будь-якого пристрою який має браузер.

Загальним суттєвим недоліком всіх цих сервісів є орієнтованість під одну задачу, якщо ж спробуєте використати сервіс для наприклад зберігання

своєї музики, ви зможете лише згрупувати її в папку, немає можливості створити списки відтворення та навіть відтворити послідовно файли [17,18].

1.3 Попередня обробка завантажуваних даних, групування.

Попередня обробка даних що завантажуються дозволяє зменшити навантаження на апаратний комплекс обладнання в процесі використання даних, а також зменшити апаратні а тому і фінансові витрати на обладнання. Проте викликає невелике збільшення навантаження в процесі завантаження файлів. Таким чином незначні затрати в процесі завантаження файлів компенсуються виграшем в швидкодії та зручності використання сервісу в подальшому.

Для попередньої обробки даних користувачу пропонується вказати одну з заздалегідь визначених категорій до якої належать ці дані картинка, відео, аудіо. Таким чином можемо розділити контент і зменшити кількість вибірки для аналізу.

Одним із важливих етапів є розрахунок хешів для контенту що завантажується, в першу чергу потрібно розрахувати криптографічний хеш для кожного з файлів, ця операція потрібна для виключення завантаження дублікатів на сервер, це дозволить не зберігати надлишкову інформацію, для зменшення затрат трафіку та витрат дискового простору логічно буде змістити цю операцію на фронт-енд частину.

Наступним важливим етапом буде розрахунок перцептивного хеша, що дозволить робити висновки про степінь подібності контенту і в подальшому використовуючи поведінкові дані користувача, будувати списки рекомендованого контенту.

1.4 Методи персоналізації контенту.

Завдання інтелектуальної рекомендації контенту на ринку давно відома проблема яка активно вирішується, розробляються різні методи та шляхи,

основною галуззю використання таких систем є рекламна галузь, очевидно що потенційний покупець з набагато більшим бажанням відкриє рекламне оголошення, якщо оголошення пропонує користувачу потрібну йому річ в даний момент.

В рекламній галузі таке завдання вирішується за допомогою сервісів аналітики які збирають дані про поведінку користувача. Які банери були відкриті які товари в інтернет магазинах були переглянуті. Для вирішення проблеми інтелектуальної рекомендації контенту пропоную перенести досвід маркетологів на контентну частину інтернету.

Для побудови алгоритму інтелектуальних рекомендацій потрібно виконати два основних завдання, створити відбиток користувача який буде відображати його вподобання та смаки. Наступним важливим питанням є визначення подібності контенту.

Оскільки сервіс передбачає збереження трьох типів даних, зображень, аудіо та відео контенту, то ми маємо розробит механізм визначення подібності зображень, подібності відео та аудіо інформації.

Для формування профілю користувача нам не потрібно знати, або обробляти його особисті дані імена чи вік, для цього потрібно лише проаналізувати поведінку користувача, які зображення були прокоментовані, які зображення були відмічені як ті що подобаються, аналогічні дані потрібно збирати з прослуханих аудіо та відео файлів, що правда у відео та аудіо додається час прослуховування.

Якщо користувач прослухав аудіо файл на 90% очевидно, що в більшості випадків цей файл йому сподобався набагато більше ніж той який був вимкнений після 5% прослуховування.

Опираючись на зібрані данні ми зможемо сформувати образ користувача, що буде відображати його смаки. В подальшому ця інформація використовується для побудови списку рекомендованого до перегляду контенту.

1.5 Методи визначення схожості контенту.

Визначення подібності завантаженого контенту є невід'ємною частиною для інтелектуальної рекомендації, оскільки ми вже маємо інформацію що користувачу сподобалось, а що ні та його та всі його вподобання, зосталось визначити подібний контент. Для визначення схожості зображень, відео та аудіо файлів пропоную виділити певні ознаки за якими можна порівнювати контент та визначити степінь його подібності, для виділення такої ознаки пропоную використовувати ряд функцій для розрахунку перцептивного хеша.

Перцептивні хеші – це концепція розрахунку унікального відбитку об'єкта що відрізняється від концепції криптографічних шехів розрахованих функціями як MD5 або SHA1. В криптографії кожен хеш є випадковим. Дані що використовуються для генерації хеш, виконують роль джерела випадкових чисел, таким чином однакові данні дають однаковий результат, а різні дані – різний результат. Порівнюючи два хеша наприклад SHA1 можна зробити лише два висновки. Якщо хеші відрізняються то дані для їх генерації використовувались різні, якщо хеші співпадають значить данні для генерації швидше за все використовувались однакові, оскільки існує також ймовірність виникнення колізій, тому однакові хеші не гарантують спів падіння даних. На відміну від криптографічних хешів, перцептивні хеші можна порівнювати між собою і робити висновки про степінь схожості двох наборів даних.

Після обробки завантаженого об'єкта ми отримаємо базу з набором перцептивних хешів однакової довжини. Для порівняння рядків однакової довжини чудово підходить відстань Геммінга. Відстань Геммінга визначається числом позицій, в яких відповідні символи двох слів однакової довжини відрізняються між собою

Основою даного алгоритму є відображення середнього значення низьких частот. В зображенні високі частоти забезпечують деталізацію, а низькі частоти показують структуру зображення. Тому для побудови такої хеш-функції яка для схожих зображень буде видавати близький хеш, потрібно позбутись від високих частот. Принцип роботи алгоритму:

- **Зменшення розміру.** Найшвидший спосіб позбутись від високих частот зменшити розмір зображення. Зображення зменшується до розміру в діапазоні від 32x32 до 8x8.
- **Видалення кольору.** Зменшене зображення переводиться в зображення градацій сірого що забезпечує зменшення хешу втричі.
- **Визначення середнього значення кольору для всіх пік селів.**
- **Побудова ланцюжка бітів.** Для кожного пікселя виконується заміна кольру на 1 або 0 в зараженості від того воно більше середнього чи менше.
- **Побудова хешу.** Генерація 1024 бітів в одне значення. Порядок значення не має але зазвичай біти записуються з ліва на право, зверху вниз..

Отриманий хеш стійкий до масштабування, зжимання або розтягування зображення, а також до зміни яскравості, контрасту і маніпуляціх з кольорами зображення. Проте головна перевага алгоритму – швидкість роботи. Для порівняння хешів цього типу використовується функція нормованого відстані Хемінга.

Discrete Cosine Transform Based Hash, дискретне косинус не перетворення – одне з ортогональних перетворень, що тісно зв’язане з дискретним перетворенням Фурє і є гомоморфізмом його векторного простору. Як і будь яке Фурє-орієнтоване перетворення, виражає функцію або сигнал (послідовність з кінечної кількості точок даних) у вигляді суми синусоїд з різними частотами і амплітудами. Дискретне косинус не

перетворення використовує тільки косинусні фунції. Існує 8 типів дискретних косинус них перетворень.

Низькочастотні коефіцієнти ДКП найбільш стійкі до маніпуляцій з зображенням. Це відбувається тому, що більша частина інформації сигналу, як правило, зосереджена в кількох низькочастотних коефіцієнтах.

Генерація хешу використовуючи алгоритм дискретних косинус них перетворень:

- **Видалення кольору.** Для видалення непотрібних високих частот.
- **Застосування медіанного фільтру** для зменшення рівня шуму. При цьому зображення розбивається на так звані «вікна», після чого кожне вікно заміняється медіаною для сусідніх вікон
- **Зменшення зображення до 32x32;**
- **Застосування ДКП до зображення;**
- **Побудова хешу.**

Головною перевагою такого хеша є: стійкість до маленьких поворотів, розмиванню зображення і стиснення зображень, а також швидкість порівняння хешів, завдяки їх маленькому розміру. Для порівняння цього тиу хешів використовується функція відстані Хемінга.

Radial Variance Based Hash, ідея алгоритму Radial Variance Based Hash заснована на побудові променевого вектора дисперсії на основі перетворення Радона. Після чого до вектора застосовується дискретне косинус не перетворення і розраховується хеш.

Перетворення Радона – це інтегральне перетворення функції множини змінних вздовж прямої. Воно стійке до обробки зображень з використанням різних маніпуляцій наприклад стиснення, геометричних перетворень наприклад поворотів.

Для побудови хешу використовуючи даний алгоритм потрібно виконати наступні дії:

- **Видалити колір** для видалення непотрібних високих частот зображення;
- **Розмивання зображення** (blurring) використовуючи Гаусове розмивання. Зображення перетворюється з використанням функції Гаусса для змешення впливу деяких шумів.
- **Застосування гама-корекції** збільшення контрастності зображення.
- **Побудова променевого вектору дисперсії;**
- **Застосування ДКП до вектора дисперсії;**
- **Побудова хешу.**

Для порівняння хешів цього типу використовується пошуки піка взаємокореляційної функції

Marr-Hildreth Operator Based Hash, оператор Марра-Хилдрета дозволяє визначити границі на зображенні.

В загальному кажучи границі зображення можна визначити як край чи контур, розділяючий сусідні частини зображення які мають порівняно різні характеристики у відповідності з деякими особливостями. Такими особливостями можуть бути колір або текстура, але найчастіше використовується сіра градація кольору зображення тобто яскравість. Результатом виявлення границь на зображенні є карта границь. Карта границь описує класифікацію границь для кожного пікселя зображення. Якщо границі визначати як різку зміну яскравості, то для їх знаходження можна використовувати похідні або градієнт.

Для знаходження хешу використовуючи оператор Марра-Хилдрета потрібно виконати наступні кроки:

- **Видалення кольору** для зменшення впливу непотрібних високих частот;
- **Перетворення зображення в розмір 128x128;**

- **Розмиття зображення (blurring).** Зображення перетворюється з використанням функції Гауса для зменшення впливу деяких шумів.
- **Побудова оператора Марра-Хилдрета;**
- **Застосування дискретної згортки до LoG і зображення.** Отримаємо зображення на якому чітко видно стрибки яскравості.
- **Перетворення зображення в гістограму.** Зображення розбивається на маленькі блоки (5x5) в яких сумуються значення яскравості.
- **Побудова хешу з гістограми.** Гістограма розбивається а блоки 3x3. Для цих блоків розраховується середнє значення яскравості і використовується метод побудови ланцюга бітів. Отримуємо бінарний хеш розміром 64 байта. Розмір отриманого хеша не маленький, але порівняння двух хешів займає достатньо маленьку кількість часу в порівнянні с Radial алгоритмом, так як використовується функція ненормованої відстані Хемінга. Також такий алгоритм не стійкий до поворотів зображення, але стійкий до масштабування затемненню і стисненню.

1.6 Мета та задачі дослідження

На сьогоднішній день існує багато сервісів, які можуть виконувати багато різноманітних задач, деякі пропонують зберегти вашу інформацію, інші призначені для прослуховування музики, вони підбирають для вас найбільш підходящі композиції спираючись на раніше прослухані, є і сервіси, що дозволять вам продивлятись відео, не завантажуючи його на комп’ютер.

Всі ці сервіси охоплюють доволі широкі можливості. Та можливості завантажити свої файли, створювати зручні колекції файлів таких як плейлисти, і відразу ж послухати їх, отримати доступ до них з будь-якого пристрою, який має браузер, і при цьому отримувати рекомендації щодо контенту в одному сервісі такої можливості немає. Доведеться для збереження файлів використати Google Drive, OneDrive, Dropbox, для

прослуховування музики, доведеться використовувати Spotify, google play music, для перегляду відео доведеться використовувати Youtube, а для фото інший сервіс, а отримати весь функціонал в одному сервісі не вийде. Коли захочеться використати Google Drive для створення плейлиста оналайн виявиться, що цей сервіс не зможе навіть перемикати музику.

Доволі зручним був би сервіс в якому можна завантажити свої файли, де можна згрупувати їх за якимсь із критеріїв, або ж не групувати взагалі. Для згрупованих файлів наприклад аудіо, зручно було б запустити плеєр і прослухати весь набір таких файлів, а коли захочеться нового, було б чудово щоб сервіс запропонував тобі щось хороше. Так само для відео чи фото, зручно також було б відмітити тип доступу до файлів, можливо є файли, призначені для загального доступу, але існують приватні файли, які не мають бути доступні нікому.

Оскільки розширення функціональних можливостей web-сервісу для роботи з мультимедійною інформацією за рахунок введення додаткових функцій для авторизації, обмеження доступу до файлів та об’єднання їх у певні категорії для зручності роботи користувача є метою роботи. Групування файлів у певні категорії використовуючи теги, а також аналіз поведінки користувача та визначення схожості контенту між собою шляхом порівняння хешів, для забезпечення інтелектуальної рекомендації контенту

Для досягнення поставленої мети необхідно розв’язати такі задачі:

- провести аналіз існуючих на ринку провідних онлайн сервісів розробок з аналогічними функціями;
- розробити web-сервіс у вигляді односторінкового додатку, з авторизацією;
- надати можливість завантажувати файли та групувати їх у певні категорії за призначенням файлів, плейлисти, колекції, відеотеки, галереї з тегами на кожному з файлів.
- для кожної з категорій розробити можливість попереднього перегляду, для аудіофайлів це має бути аудіопрогравач, для галереї – слайдер;

- забезпечити обмеження доступу до файлів та додати можливість перегляду загальнодоступних колекцій, розробити інтелектуальну рекомендацію контенту;
- провести тестування розробленого продукту та його порівняння з провідними аналогами;
- розробити технічну документацію на розроблений програмний комплекс.

1.7 Висновки аналізу технологій для розробки

У даному розділі було розглянуто можливі архітектури побудови web-додатку. Проведено аналіз технологій побудови кожної з структурних складових, шаблонів проектування та технологій, що роблять можливим зберігання інформації в базі даних, в тому числі великих масивів даних. Розглянуто технології, що дозволяють побудувати дружній для користувача інтерфейс, а також технології, що дозволяють перенести деякий функціонал з серверної частини на клієнтську таким чином розвантажити сервер.

Також розглянуто технології, що використовують в побудові сучасних web-додатків на серверній стороні і дозволяють побудувати швидкий та ефективний API, що зможуть не тільки відповідати на запити клієнта, а й спілкуватись з іншими сервісами, та взаємодіяти з базою або з кількома базами даних різних типів.

2 РОЗРОБКА WEB-ДОДАТКА

2.1 Побудова серверної частини web-додатка

Серверну частину додатка було вирішено побудувати, використовуючи мову Java . Ця мова досить швидко розвивається, має чудову документацію, використовуючи її можна написати дійсно складні та стабільні програми. Широкого розповсюдження для побудови web-додатків корпоративного рівня набув фреймворк під назвою Spring, він, як і сама Java має велику спільноту, сам він обширний і розвинувся в багато побічних самостійних проектів таких як Spring Security, Spring Boot. Для розробки проектів на java зазвичай потрібно достатньо багато часу через деяку її надлишковість, саме тому в роботі було використано Spring Boot для прискорення процесу початкової розробки, а в подальшому, якщо потрібні будуть удосконалення, або зміна функціоналу, що пропонує Spring Boot, на щось інше, це завжди можна буде зробити.

Для розробки середніх та великих проектів, або таких, які в подальшому можуть розширюватись використовується об’єктно-орієнтовний підхід.

Проекти на java можуть доволі сильно відрізнятись по структурі або по способу конфігурації, та ферймворк накладає деякий відбиток і саме для того і створений, щоб диктувати деякі вдалі архітектурні рішення. Spring Boot пропонує використовувати MVC (Model View Controller), зображено на рисунку 2.1.

Такий підхід передбачає існування трьох шарів додатка, модель – це сутності і операції з ними, контролер - обробка запитів, відправка відповідей, представлення – все, що зв’язано з відображенням даних.

Тому спочатку потрібно визначитись з сутностями, що будуть фігурувати в даному проекті, щоб в подальшому можна було написати бізнес-логіку, тобто реалізувати Model-шар додатка.

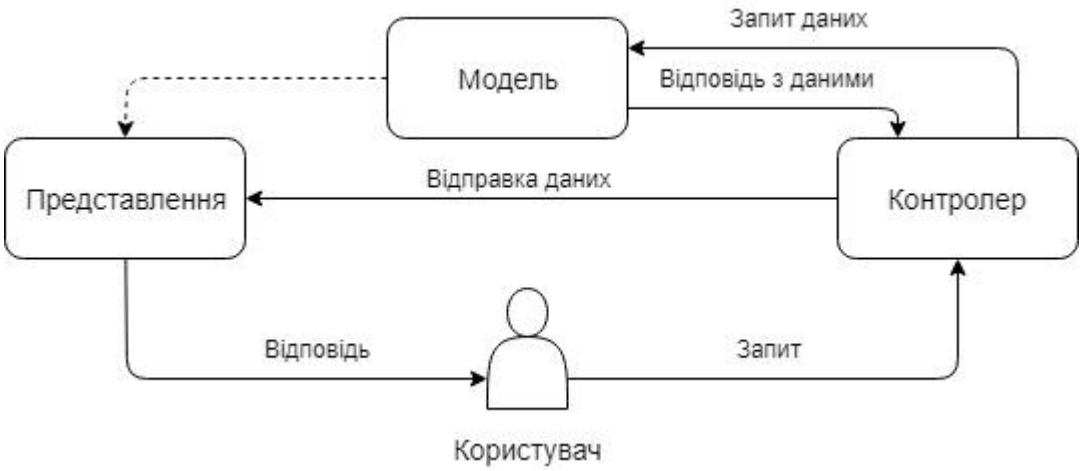


Рисунок 2.1 – MVC модель web-додатка.

Далі маємо написати контролери для отримання запитів від клієнтів та відправки відповідей, для програм середньої складності хорошою практикою є визначення DTO (data transfer object) оскільки шар представлення у нас буде дещо специфічний, то нам потрібно описати, які параметри клієнт має прислати серверу, а також описати, яку відповідь має відправити сервер клієнту [4,19].

2.1.1 Визначення сутностей

Для обмеження доступу до сервісу нам потрібно спочатку якось розрізняти користувачів, для цього, потрібно визначити сутність користувача User. В складних проектах може знадобитись розділити сутність User, що буде використовуватись при розрахунках і різних операціях в програмі, а також сутність бази даних UserEntity як буде мати деякі анотації і зберігатись в базу, проте на даному етапі це буде надлишковим, тому було вирішено одразу створити сутність User з анотаціями для збереження в базі, коли ж виникне ситуація, що клієнту не потрібно буде слати якісь із полів, можливе використання DTO для обмеження інформації, яку може отримати клієнт. Отже сутність User буде мати вигляд.

```
@Entity
@Table(name="users")
public class User {
```

```
    @Id
    @GeneratedValue
    private Long id;

    @Column(name="username")
    private String username;

    @Column(name="email")
    private String email;

    @Column(name="create_date")
    private Date createDate;

    @Column(name="password")
    private String hashPass;

    @Column(name="phone")
    private String phone;
public User(){}

public User(String username, String email, Date
createDate, String hashPass, String phone){

    this.username = username;
    this.email = email;
    this.createDate = createDate;
    this.hashPass = hashPass;
    this.phone = phone;

}
// getters and setters
}
```

@Entity – Анотація вказує Hibernate, що цей клас позначений нею це сутність(entity bean). Такий клас повинен мати конструктор за замовчуванням і він обов’язково має бути пустим.

@Table() – параметризована анотація, за допомогою якої, Hibernate точно знає з якою таблицею має бути зв’язана представлена сутність, анотація може приймати як параметр ім’я таблиці, каталог, унікальність стовпців.

@Id – вказує що дане поле виступає первинним ключем. @GeneratedValue – використовується разом з @Id і вказує що поле має автоматично генеруватись.

@Column() – вказує яким стовпчиком таблиці буде відопвідати поле класу, позначене такою анотацією. Анотація параметризована задати можна такі параметри як name, unique, nullable, length.

Звичайний клас який описує користувача з деякими покращеннями, так додані деякі анотації потрібні для розпізнавання і відповідного збереження полів у колонки таблиць SQL. По аналогії визначаємо модель File, також нам потрібна можливість групувати файли за різним призначенням, тому створюємо моделі колекцій, AudioCollection, FileCollection, VideoCollection, ImageCollection. В подальшому в такий клас потрібно буде додати спеціальні поля з анотаціями які будуть відображати зв’язки між таблицями. Як наприклад зв'язок колекції користувача та файлів, що додається в модель AudioCollection:

```
@ManyToOne()
private User user;
@OneToMany(mappedBy = "audioCollection")
private List<File> fileList;
```

@ManyToOne(), @OneToMany – анотації що дозволяють організувати тип звязку між таблицями один-до-багатьох, та багато-до-одного.

Відповідні поля потрібно додати і для сутностей User та File. Таким чином, будуються усі сутності, а спеціальні анотації дозволяють відображати структуру і взаємозв’язки сутностей на базу даних. [20]

2.1.2 Визначення контролерів

Контролери – це деякі класи, які визначаються для реагування на запити клієнтів, кожен контролер має один або декілька методів, методи мають свої url-адреси, а також тип запиту, на який метод має реагувати, таким чином, клієнт повинен відправити запит на конкретний url, певним

http методом, і коли сервер отримає такий запит, перевірить чи існує метод контролера з такими параметрами, якщо існує то управління буде передано конкретному методу. Окрім типу http-повідомлення та url-методи контролерів мають вхідні та вихідні параметри.

Розділяють контролери за їх функціональними можливостями наприклад можна створити контролер для авторизації користувача, контролер в спрощеному вигляді може виглядати таким чином:

```
@Controller
public class AuthController {

    @RequestMapping(value= BaseRequest.LOGIN, method=
RequestMethod.POST)
    public @ResponseBody UserResponse
login(@RequesstBody UserLoginRequest userLoginRequest,
HttpServletResponse response){
          // логіка авторизації користувача
    }

    @RequestMapping(value=BaseRequest.LOGOUT,
method=RequestMethod.GET)
    public String logout(HttpServletResponse response){
    // логіка виходу користувача з системи
    }
}
```

@Controller – анотація, що позначає клас як контролер Spring Boot.

@RequestMapping() – параметризована анотація, що позначає методи контролера Spring Boot має обов’язковий параметр value, який містить url на який буде реагувати метод.

@RequestBody, @ResponseBody – анотації для позначення DTO запиту та відповіді.

HttpServletResponse – об’єкт http-відповіді, що дозволяє взаємодіяти з http-заголовками.

Проте логіки в контролерах писати не варто – це є поганим тоном і суперечить ідеології Spring вся логіка має виноситись в сервіси, так можливе розміщення логіки зв’язаної з зміною http запитів або відповідей, та при складних розрахунках таку логіку теж варто винести в сервіси. Усі маніпуляції з сутностями також мають знаходитись в сервісах.

Щоб звернутись до сервіса широко використовується механізм ін’єкції залежностей (dependency injection) така ін’єкція залежностей в контролер може виглядати наступним чином:

```
private final UserService userService;
private final JwtService jwtService;
private final HashService hashService;

@Autowired
public AuthController(UserService userService,
JwtService jwtService,HashService hashService) {

    this.jwtService = jwtService;
    this.userService = userService;
    this.hashService = hashService;
}
```

@Autowired – помічає конструктор, поле або метод як такий що потребує автозаповнення ін’єкцією залежності.

Це дозволяє використовувати функції сервісів всередині контролера. В даному проекті було вирішено створити наступні контролери AuthController – для авторизації, автентифікації, та виходу користувача з системи. AudioColController, FileColController, VideoColController, ImageColController – відповідно для колекцій аудіо, файлів, відео та колекцій зображень. UserController – для отримання створення видалення користувачів та іншого.

А також контролерів для роботи з файлами, завантаження файлів віддавання файлів клієнтам.

Вхідними та вихідними параметрами для контролерів є спеціальні об’єкти DTO такі об’єкти описують параметри які мають прийти з клієнта, а також те, що контролер має відправити назад клієнту. Об’єкт для авторизації користувача може виглядати наступним чином:

```
public class UserLoginRequest {

    private String username;
    private String password;

    public UserLoginRequest(){
    }

    public UserLoginRequest( String username, String
password){
        this.username = username;
        this.password = password;
    }
 // getters and setters
}
```

Об’єкт відповіді може виглядати таким чином:

```
public class UserResponse {

    private String username;
    private String phone;
    private String email;
    private String token;

    public UserResponse(String username, String phone,
String email, String token){

        this.username = username;
        this.phone = phone;
        this.email = email;
        this.token = token;

    }

    public UserResponse(){}
    // getters and setters
}
```

Сервіси додатка можуть виконувати не лише маніпуляції з сутностями, а дуже різноманітні задачі, наприклад, генерацію jwt-токена для авторизації користувача, сервіс може виглядати таким чином:

```
@Service
public class JwtService {

    public String createJWT(User user){

        Date nowDate = new Date();

        String jwtStr = Jwts.builder()
                .setSubject("ME")
                .claim("id", user.getId())
                .claim("name",user.getUsername())
                .claim("email",user.getEmail())
                .claim("phone", user.getPhone())
                .claim("date", nowDate.toString())
                .signWith(SignatureAlgorithm.HS256,
TextCodec.BASE64.decode(GlobalConfig.SECRET))
                .compact();
        return jwtStr;

    }

    public String getUserEmail(String token){

        if(token.isEmpty()){
            return null;
        }else{

            Jws<Claims> jws = Jwts.parser()
.setSigningKey(TextCodec.BASE64.decode(GlobalConfig.SEC
RET)).parseClaimsJws(token);
        return jws.getBody().get("email").toString();

        }
    }

    public String getDate(String token){

        if(token.isEmpty()){
            return null;
        }else{
```

```
Jws<Claims> jws =
Jwts.parser().setSigningKey(TextCodec.BASE64.decode(Glo
balConfig.SECRET)).parseClaimsJws(token);
            return jws.getBody().get("date").toString();

        }
    }
}
```

Клас сервісу позначається відповідною спеціальною анотацією @Service. [21]

2.2 Взаємодія web-додатка з базою даних

Для доступу до баз даних в Java можна використовувати кілька шляхів. Та основою їх завжди є драйвер, написаний для кожної з баз, для реляційних баз даних таких як PostgreSQL, MySQL використовується JDBC(Java DataBase Connectivity), для нереляційних використовується драйвери для кожної свій. Використовуючи такий драйвер можна налаштувати з'єднання з будь-якою базою. А використовуючи API-інтерфейс можна надсилати запити до бази та отримувати відповідь у вигляді даних. При такій організації, щоб отримати дані, доведеться писати спеціальний запит на мові запитів до бази SQL для реляційних баз, та використовувати API драйвера для відправки їх.

В спрощеному вигляді запит на вибірку даних буде виглядати так:

```
String query = "SELECT id, name, author FROM songs";

rs = stmt.executeQuery(query);
```

NoSQL бази не підтримують мови запитів, але все ж можливо виконати прості запити, для цього використовується API драйвера. У випадку MongoDB створення з'єднання та вставка даних у колекцію буде виглядати таким чином :

```
MongoClient mongoClient = new MongoClient("localhost",
27017);

DB database = mongoClient.getDB("myMongoDb");

DBCollection collection =
database.getCollection("songs");
```

```
BasicDBObject document = new BasicDBObject();

document.put("name", "Годі говорити про того, кого
нема");
document.put("author", "Vivienne Mort");

collection.insert(document);
```

При використанні лише драйвера та API-інтерфейсу написання програми вимагає доволі великих витрат на написання запитів, обробки помилок та даних, тому для пришвидшення та спрощення розробки в невеликих проектах часто використовують ORM (Object Relation Mapping) такі як Hibernate для реляційних баз, та Mongoose для нереляційної MongoDB.

Ці системи надають єдиний інтерфейс для доступу до бази і дозволяють зосередити увагу на розробці програми, а не її взаємодії з базою. В програмі створюються спеціальні класи Entity та репозиторії для взаємодії з базою. [22]

Так як даний сервіс не потребує над-високої продуктивності запитів та не має витримувати великих навантажень, було обрано реляційну базу PostgreSQL, а для пришвидшення процесу розробки використаємо ORM-систему під назвою Hibernate, це дозволить нам створити сутності Entity, а не зосереджуватись на написанні SQL-запитів, а про перетворення класів Entity в таблиці буде займатись сам Hibernate-репозиторій, що має такий вигляд:

```
public interface UserRepository extends

CrudRepository<User, Long> {
    User findByLogin(String login);
    List<User> findFirst10ByIdNotIn(List<Long> users);
}
```

2.3 Побудова клієнтської частини web-додатка

Модель MVC, яка знайшла широке застосування в Spring, для побудови front-end використовує рзіні шаблонізатори. Шаблони – це певні макети схожі до звичайних html-сторінок з доповненнями, спеціальними позначками куди і як мають вставлятись дані, зазвичай вставкою даних займається сервер, після того як дані були підставлені сторінка повністю відправляється клієнту. З таким підходом існують багато додатків, і він відповідає моделі MVC. Недоліком такої організації є додаткове навантаження на сервер, додаткове навантаження на мережу оскільки потрібно при зміні даних повторно відправляти цілу сторінку.

Іншим варіантом є використання JavaScript-фреймворків. Шаблони завантажуються клієнтом в незаповненому вигляді як статичний контент, а далі працює JavaScript запитує дані з сервера, заповнює шаблони, оновлюючи застарілі дані.

Сучасні клієнти не обходяться без JavaScript, тому що він дає дуже багато функціональних можливостей і якщо потрібно буде на клієнті реалізувати якусь специфічну функцію з JavaScript ми можемо бути впевненими, що зможемо це зробити. Та використання JavaScript для великого проекту накладає деякий відбиток та складнощі, тому кращим вибором буде Angular 2 + версії, що спеціально проектувався для додатків рівня підприємства і використовуючи його можна написати дійсно великі швидкі та стабільні додатки, а мова Typescript робить розробку простішою, а поведінку додатку більш прогнозованою, також, за рахунок типізації, зменшує виникнення потенційних помилок. На додачу для передачі даних Angular використовує технологію Ajax, що робить наш інтерфейс зручним для користувача максимально.

Розробники Angular не тільки зробили чудовий продукт, а також подбали про розробників та разом з своїм фреймворком пропонують використовувати консольний додаток для розробки, який вміє генерувати проект, налаштовувати його, створювати сервіси, модулі, компоненти та

багато іншого, тому для побудови проекту було вирішено використовувати AngularCLI.

Хоч додатки на Angular можна розробляти використовуючи JavaScript проте розроблявся він для використання мови, написаної поверх JavaScript і призначеної зробити його передбачуванішим, тому було вирішено використовувати TypeScript з його динамічною типізацією. Typescript – також використовує стандарт ES6 тому ми можемо отримати доступ до найновіших можливостей. А той факт, що TypeScript в результаті компілюється в JavaScript дає нам можливість у випадку проблем звернутись прямо до JavaScript і переписати частину коду для правильного виконання. Хоча на практиці це буває доволі рідко, компілятор TypeScript генерує достатньо продуктивний код. [9,23]

Angular-додаток будується з модулів, які складаються з компонентів, каналів, та директив. Компоненти містять логіку зв'язану з відображенням даних, директиви – це спеціальний структурний інструмент що взаємодіє з шаблоном компонента і змінює його структуру, канали – спеціальні інструменти, що покликані формувати дані відображення, обробкою даних їх отриманням або ж відправкою займаються сервіси.

Модульна структура додатків Angular спрощує розробку великих проектів, з складною архітектурою. Схематично структура Angular додатка зображена на рисунку 1.5.

У додатках Angular компоненти можуть вкладатись один в одного, таким чином, компонент в який вкладається називається батьківським, а вкладений компонент дочірнім або ж нащадком, хоч компоненти і називаються батьківськими і дочірніми нічого спільного з наслідуванням і ООП вони не мають [9].

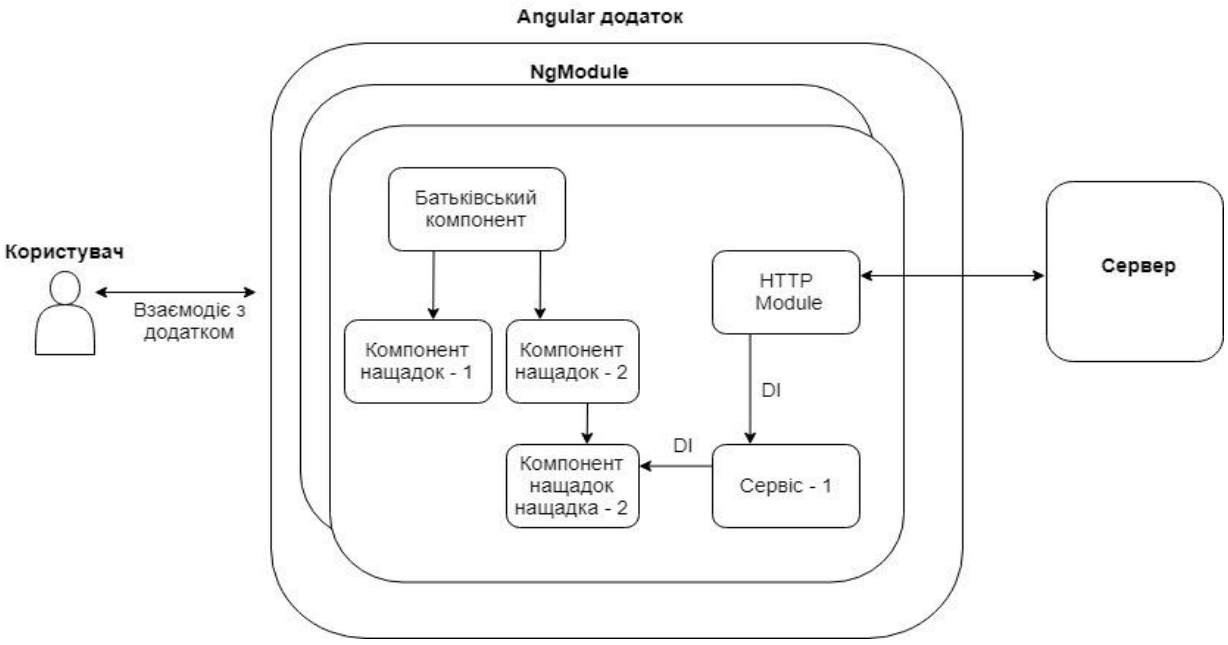


Рисунок 1.5 – Структурна схема додатка Angular

Між батьківськими компонентами і нащадкам можлива комунікація і передача даних як від батьківського дочірньому і від дочірнього батьківському. Структурна схема компонента зображена на рисунку 1.6.

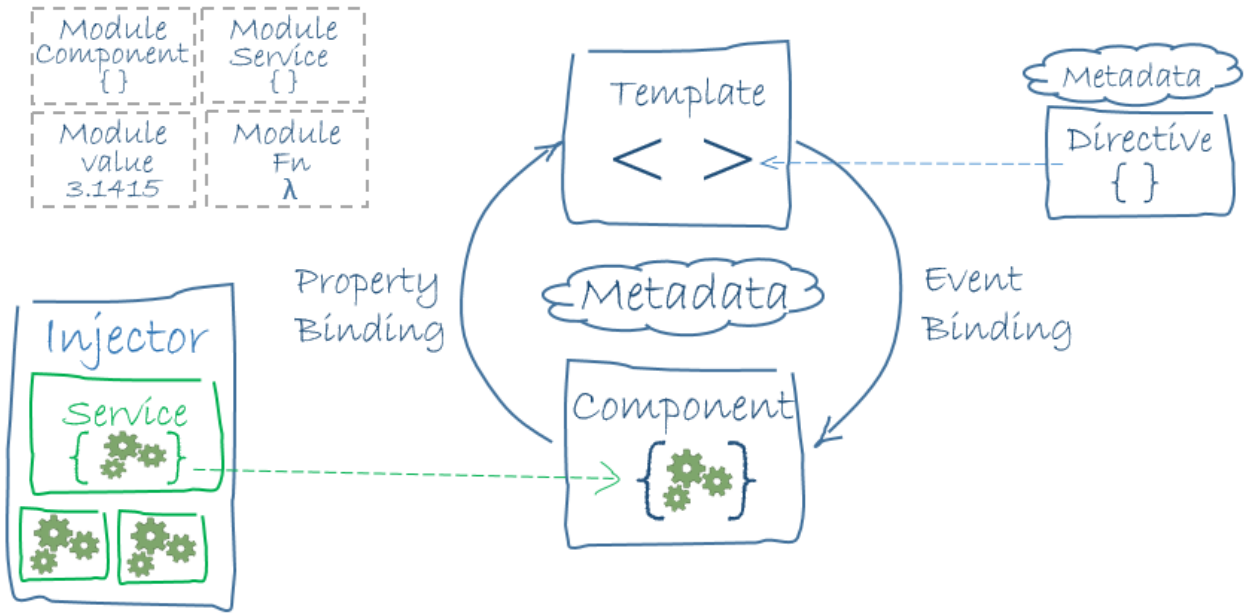


Рисунок 1.6 – Структурна схема компонента Angular

Широкого вжитку в Angular додатках також набув принцип впровадження в компонент зовнішньої залежності DI(Dependency Injection).

Впровадження залежностей дозволяє в парі з модульною архітектурою дозволяє писати набагато менш зв’язаний код, і тим самим надає більше можливостей для тестування та розширення додатка. [10]

Основою для побудови додатка Angular є модулі, більше того фреймворк сам складається з модулів, таким чином існують модулі Angular такі як BrowserModule, FormsModule, HttpModule, RouterModule та багато інших, а також існують користувацькі модулі. В загальному випадку, компоненти і сервіси об’єднуються в модулі за функціоналом або ж якоюсь ознакою. Модуль для відображення слкадових додатку виглядає так:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';

import { FooterComponent } from './footer/footer.component';
import { NavbarComponent } from './navbar/navbar.component';
import { SidebarComponent } from './sidebar/sidebar.component';

@NgModule({
  imports: [
    CommonModule,
    RouterModule,
  ],
  declarations: [
    FooterComponent,
    NavbarComponent,
    SidebarComponent
  ],
  exports: [
    FooterComponent,
    NavbarComponent,
```

```
    SidebarComponent
  ]
})
export class ComponentsModule { }
```

@NgModule() – параметризований декоратор, що позначає TypeScript клас як модуль Angular, має такі параметри:

imports: – приймає масив інших модулів, які використовються у компонентах цього модуля.

declarations: – приймає масив компонентів представлення таких як компоненти, директиви, канали.

exports: – Масив компонентів представлень, які належать цьому модулю і повинні використовуватись у інших модулях.

providers: – Масив сервісів, що використовуються у цьому модулі.

Angular-компонент – це деяка сутність, що являє собою Typescript клас і містить в собі параметризовану анотацію (декоратор) @Component() .

Загальна структура компонента виглядає таким чином:

```
import { Component, OnInit } from '@angular/core';

@Component({

  selector: 'app-detail-gallery',
  templateUrl: './detail-gallery.component.html',
  styleUrls: ['./detail-gallery.component.scss']

})
export class DetailGalleryComponent implements OnInit {

  constructor() { }
```

```
  ngOnInit() {}
}
```

selector – обов’язковий параметр, він описує тег в html-шаблоні, в якому буде рендеритись компонент, назва тега будь-якого компонента обов’язково повинна починатись з app- для прикладу 'app-detail-gallery';

templateUrl – вказує шлях до шаблону, який буде використовуватись для компонента , для простих компонентів можна не створювати окремого файлу і писати код шаблону прямо в компоненті, проте це не надто зручно коли компонент розростається. Шаблони фактично залишились звичайним HTML проте додалось багато можливостей, як наприклад цикли, зв’язування змінних, умовні розгалуження та багато іншого;

styleUrls – параметр, що вказує шлях до стилів які мають застосовуватись до цього компонента. Це може бути як один файл так і масив файлів, при чому зовсім не обов’язково CSS, це можуть також бути файли на скриптовій мові Sass, яка потім інтерпретується в каскадні таблиці стилів.

Конструктор, на відміну від багатьох інших мов, називається не іменем класу, а зарезервованим словом constructor, також компоненти можуть реалізовувати деякі інтерфейси, в даному випадку OnInit, який вимагає реалізувати один метод ngOnInit(), цей метод буде визваний після ініціалізації компонента.

Сервіси, на відміну від компонентів, не мають ні шаблона, ні стилів, там описуються лише методи для деяких операцій.

Сервіс Angular для авторизації користувача у спрощеному вигляді виглядає так.

```
@Injectable()
export class LoginService {
   private apiUrl: ApiUrl = new ApiUrl();
   constructor(public http: Http) {
   }
```

```
  public setLogin = (user: any) => {
    const body = {
      username: user.username,
      password: user.password
    };
    this.http.post(this.apiUrl.LOGIN_USER, body).subscribe(response => {
      if (response) {
        localStorage.setItem('user', JSON.stringify(response));
      } else {
        console.error('authorization is filed')
      }
    });
  }
  public logOut = () => {
    localStorage.removeItem('user');
    this.http.get(this.apiUrl.LOGOUT).subscribe(response => {
      if (response) {
        console.log('Logout request true');
      }
    });
  }
  public getLogin = () => {
    if (localStorage.getItem('user')) {
      return localStorage.getItem('user');
    } else {
      return null;
    }
  }
}
```

@Injectable – сервіс позначений таким декоратором (анотацією) може використовувати в собі інші сервіси. Окрім назви файлу в якій присутнє слово service, Angular сервіси ніяк більше не позначаються.

В Angular існують наперед визначені директиви як наприклад ngFor – використовується для циклічного повторення контенту, ngIF – використовується для виведення контенту за умови, а також директива може бути написана власноруч, структура директиви виглядає так:

```
@Directive({ selector: '[while]' })
export class WhileDirective {

   constructor()
   { }

   @Input() set while() {
   }
}
```

@Directive – декоратор, що позначає клас як директиву, має обов’язковий параметр селектор, що дозволяє відшукати елемент в шаблоні.

@Input – декоратор, який позначає вхідну властивість або метод.

Як і директиви в Angular існують наперед визначені канали це, наприклад канал для відображення формату дат. Також є можливість створювати свої канали. Структура канала виглядає так:

```
@Pipe({
   name: 'factorial'
})
export class FactorialPipe implements PipeTransform {
  transform():number{ // do something}
}
```

@Pipe – параметризований декоратор, що приймає обов’язковим параметром ім’я каналу.

Компоненти можуть об’єднуватись в модулі, а модуль в свою чергу використовувати сервіси, таким чином модульна структура дозволяє набагато простіше розробляти складні додатки, дозволяє спростити тестування частин програми, а в подальшому полегшує зміни, оскільки

зменшується зв’язаність коду тим самим зменшується взаємний вплив частин програми роблячи їх дещо ізольованими і незалежними.

В Angular існує поняття дочірній і батьківський компонент. Дочірній компонент це компонент, що рендериться всередині батьківського, термінологія переплітається з об’єктно-орієнтованим програмуванням та це не наслідування, а швидше вкладеність [23].

2.4 Висновки розробки програмного продукту

Для побудови серверної частини використовувалась мова програмування Java, а також фреймворк Spring, що базується на моделі додатків з MVC-архітектурою.

Для побудови клієнтської частини використався TypeScript фреймворк від компанії Google, орієнтований на розробку SPA додатків Angular 6, що дозволило побудувати клієнтську частину, яка по зручності користування не поступається звичайним додаткам, що досягнуто використанням технології Ajax.

Оскільки в розробленому сервісі не передбачається значних навантажень на базу даних, то для збереження даних використовується реляційна база PostgreSQL, вона дозволила використати ORM Hibernate, що дало можливість зменшити витрати часу на розробку, зменшити складність розробки, при забезпеченні достатньої гнучкості і продуктивності.

3 РОЗРОБКА ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ ТА ПОРІВНЯННЯ ОНЛАЙН СЕРВІСУ З АНАЛОГІЧНИМИ ПРОДУКТАМИ

3.1 Тестування створеного продукту

Для розробки front-end було використано один з JavaScript фреймворіків Angular, цей фреймворк створений для побудови складних додатків. Розробниками Angular також було створено консольний інструментарій, Angular CLI який рекомендовано для створення односторінкових додатків Angular. При створені додатка використовувалась консольна утиліта Angular CLI версії 6.0.7. Також розробниками були створені рекомендації до написання коду і в комплект Angular CLI входить аналізатор, який перевіряє відповідність написаного коду до стандарту. Розробники пропонують дотримуватись рекомендованого стилю, оскільки це знижує шанси на виникнення помилок, зв'язаних з неоднозначним трактуванням, модулів, компонентів чи змінних. Так для тестування спочатку був використаний аналізатор коду, яка запускається за допомогою Angular Cli і команди ng lint. Після перевірки, був проведений рефакторинг коду відповідно до рекомендацій, та повторно проведена перевірка яка дала результат, що всі файли відповідають стандарту. All files pass linting. [23,25]

Angular програма написана з використанням TypeScript в ній є класи-модулі, класи-сервіси та класи-компоненти, для всіх цих класів були написані unit-тести, використовуючи Jasmine версії 2.8.0 фреймворк, для написання тестів Angular. Unit-тест для найпростішого компонента буде виглядати так

```
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {

  beforeEach(async(() => {

    TestBed.configureTestingModule({
      declarations: [
```

```
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', async(() => {
 const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));

  it(`should have as title 'app'`, async(() => {
    const fixture =
TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app');
  }));

  it('should render title in a h1 tag', async(() => {
    const fixture =
TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled =
fixture.debugElement.nativeElement;

expect(compiled.querySelector('h1').textContent).toCont
ain('Welcome to app!');
  }));
});
```

beforeEach() – метод, що викликається перед початком тестування, проводить підготовку до тестування.

It() – метод теста, виконує код теста.

expect () – метод, що виконується після завершення тестування і перевіряє чи отриманий результат співпадає з очікуваним

Для запуску тестів використовувалось консольний інструмент, який виконує вихідний код разом з кодом тестів у браузері, та слідкує за змінами у вихідному коді під назвою Karma версії 2.0.0.

Щоб запустити комплексні тести в Angular-програмах, використовується фреймворк Protractor версії 5.1.2, він запускає тести на реальному браузері, на відміну від unit-тестів, взаємодіє з програмою в

цілому, як користувач, він вміє заповнювати форми, натискати на кнопки, переходити за посиланнями та перевіряти дані, що відображаються в вікні браузера. Такі тести описуються для кожної сторінки, для кожної з сторінок створюється 2 файли перший клас *.po.ts де описуються функції для навігації отримання даних з елементів натиснення кнопок, тощо. Найпростішим прикладом наповнення такого файлу є:

```
export class AppPage {
  navigateTo() {
    return browser.get('/');
  }

  getParagraphText() {
    return element(by.css('app-root h1')).getText();
  }
}
```

Другий файл *.e2e-spec.ts в якому власне використовуються дані функції для складання послідовностей тестів. Простим прикладом тесту є:

```
describe('workspace-project App', () => {
  let page: AppPage;

  beforeEach(() => {
    page = new AppPage();
  });

  it('should display welcome message', () => {
    page.navigateTo();
    expect(page.getParagraphText()).toEqual('Welcome to
app!');
  });
});
```

Створені тести не покривають повністю функціонал додатка, проте дозволяють при модернізації частин перевірити на працездатність основний його функціонал [25].

При написанні серверної частини додатка фреймворк Spring Boot автоматично створює точку входу і функцію main для запуску додатка без необхідності розгортання додатка на сервері додатків такому як Tomcat, Jetty, Glassfish та інші, для такого класу використовується анотація @SpringBootApplication також Spring Boot використовує Junit для написання

unit-тестів для класів Java-фреймворк автоматично створює клас, в якому розміщуються тести, він позначається відповідною анотацією @SpringBootTest. Та це лише контейнер для розміщення unit-тестів, написання тестів повністю покладається на програміста.

Найпростішим прикладом тесту для контролера є:

```
@Before
public void setup() {
   this.mockMvc = standaloneSetup(new
UserController(this.jwtService, this.hashService,
this.userService)).build();
}

@Test
public void testUserController() throws Exception {
this.mockMvc.perform(get(UserRequest.GET_USERS).accept(
MediaType.parseMediaType("application/json;charset=UTF-
8")))
.andExpect(status().isOk())
.andExpect(content()
.contentType("application/json;charset=UTF-8"));
}
```

Метод, позначений анотацією @Before, напряму не відноситься до тестів, але допомагає налаштувати оточення для виконання тесту. Анотація @Before означає, що даний метод буде виконуватись перед будь-яким з тестів, визначених в класі. В цьому випадку в методі створюється екземпляр контролера, що потрібен для тесту разом з його залежностями.

Вище визначений тест відправляє get-запит контролеру за адресою визначеною константою GET_USERS встановлюючи тип контенту, що відправляється «application/json;charset=UTF-8» і очікує у відповідь Http статус OK (200) і тип контенту, що повертається «application/json;charset=UTF-8»

Чим більше розростається програмний продукт, тим більша цінність таких тестів, при реалізації нового функціоналу або ж модернізації старого, можна запустити тести і бути впевненим, що ці зміни не вплинули на інші частини програми. [26]

3.2 Складання документації

– Інструкція з розгортання додатка

При умові відвідування сервісу не більше 20 тисяч унікальних відвідувань за добу до апаратних ресурсів сервера висуваються такі вимоги:

– Intel Core Quad 9550 (12MB кэша)
– 6 Гб ОЗУ
– HDD WD(7200об/хв)
– ОС Windows/Linux

Для функціонування сервера до програмного забезпечення висуваються такі вимоги:

– Java 1.8
– PostgreSQL 9.4.5
– Tomcat 7

Можливе функціонування сервера і на більш низькій версії Java, але, оскільки додаток розроблявся на основі java 1.8, то стабільність роботи не гарантується.

Для підготовки та розгортання додатка на сервері потрібно виконати такі кроки:

– Установити Java 1.8 та налаштувати константи змінних середовищ зокрема для ОС Windows це PATH, CLASSPATH і JAVA_HOME. Після установки перевірити версію Java можна в терміналі, командному рядку або ж Power Shell, виконавши команду java –version, результат виконання представлений на рисунку 3.1.

```
C:\Users\Anton>java -version
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)

C:\Users\Anton>
```

Рисунок 3.1 – Результат перевірки версії Java

– Наступним кроком буде установка PostgreSQL версії не нижче 9.4.5 та створення бази. Для роботи з базою даних напряму можливе використання інструменту для адміністрування баз даних Navicat Premium або ж інші аналоги.

– Для роботи web-додатка, написаного на Java, потрібно установити сервер додатків Tomcat, GlassFish, Jetty або інший. Для роботи програми на сервері додатків має бути розміщений JDBC Driver, а також додані налаштування для встановлення з'єднання з базою даних. Для різних версій та серверів додатків налаштування можуть відрізнятись, тому для коректного налаштування потрібно звернутись до документації вашого сервера, для сервера додатків Apache Tomcat налаштування виглядають так:

```
<Resource name="jdbc/saveDb" auth="Container"
type="javax.sql.DataSource"
        username="just_save"
        password="********"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost:5432/save"
        maxTotal="10"
        maxIdle="5"
        initialSize="5"
        maxWaitMillis="-1"
        factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
        testWhileIdle="true"
        testOnBorrow="true"
```

validationQuery="select 1"

validationInterval="60000"

removeAbandonedTimeout="600"

removeAbandonedOnBorrow="true"

logAbandoned="true"

timeBetweenEvictionRunsMillis="30000" />

– Наступним кроком буде налаштування серверної частини та побудова програми у war-архів. Після чого зібраний архів потрібно розмістити на сервері у директорії Tomcat сервера webapps.

– Інструкція користувача

Реєстрація та авторизація:

Для використання сервісу користувач має зареєструватись і увійти в систему, використовуючи логін та пароль. Сторінка авторизації зображена на рисунку 3.2.

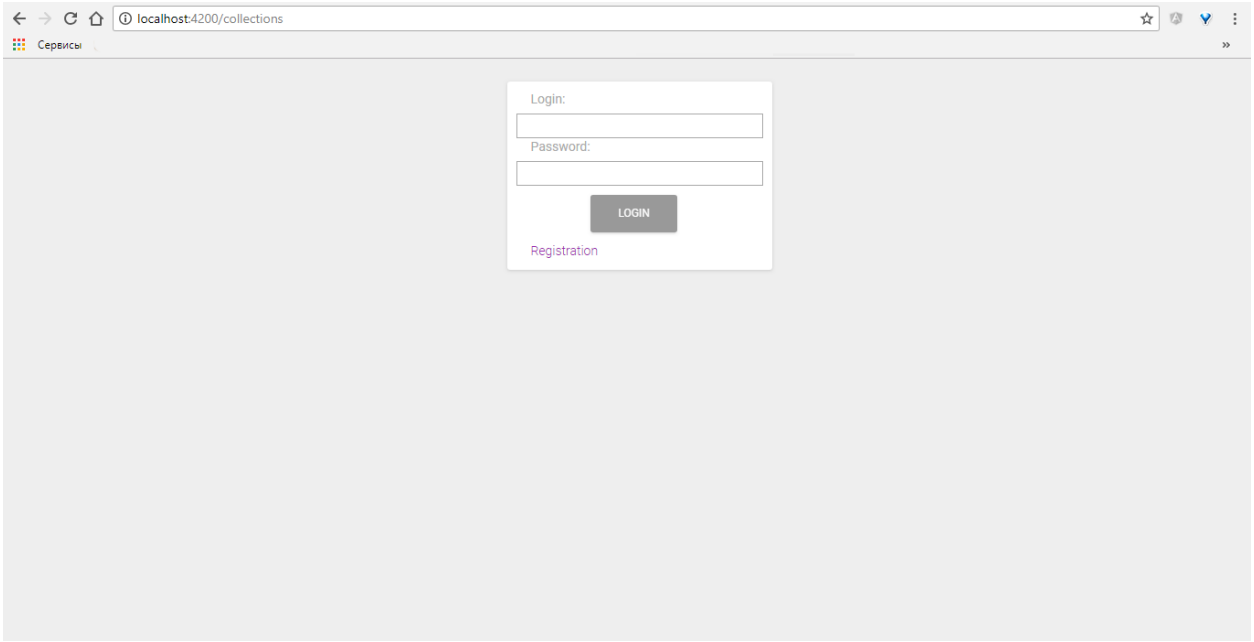


Рисунок 3.2 – Сторінка входу в систему.

При завантаженні додатка відображається вікно логіна, для реєстрації потрібно натиснути на відповідне посилання і перейти до сторінки реєстрації зображена на рисунку 3.3

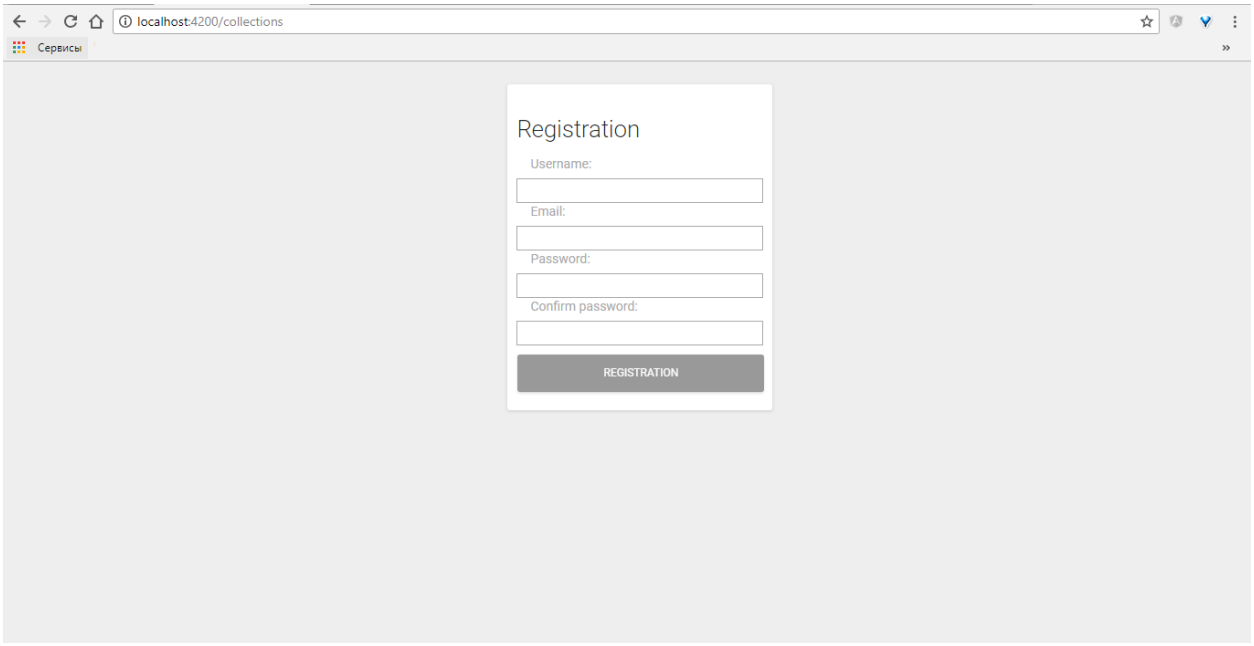


Рисунок 3.3 – Реєстрація нового користувача.

Після заповнення усіх полів натиснути кнопку Registration, після чого можна використовувати вказаний Email та пароль для авторизації в системі. Базовий інтерфейс:

Після авторизації користувач попадає на головну сторінку, зображена на рисунку 3.4, де представлений головний інтерфейс, який має такі складові:

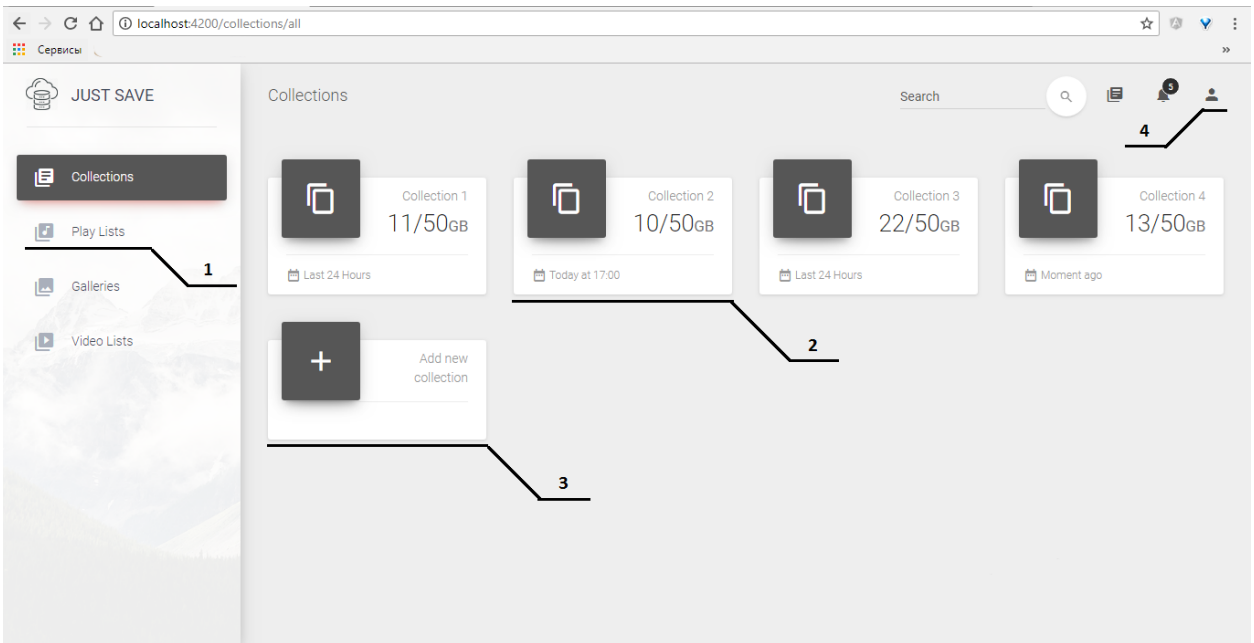


Рисунок 3.4 – Опис головної сторінки сервісу.

1 – Головне меню, представлені категорії, на які розділені файли в залежності від їх призначення. В системі можна створити колекції таких типів.

Collections – довільний набір файлів не зв’язаних єдиним типом, сюди можуть входити файли будь-яких форматів.

Playlists – згрупований набір аудіофайлів однакового або ж різних форматів.

Galleries – колекції картинок одного або ж різних форматів, згруповані за якоюсь ознакою.

Video Lists – колекція відеофайлів, згрупованих за певною ознакою.

2 – Карточка створеної колекції відображає деяку коротку інформацію, кількість файлів, назву колекції, ознаку групування, час останнього внесення змін.

3 – Карточка для додавання нової колекції.

4 – Особистий профіль користувача.

Профіль користувача:

Після натиснення піктограми профіля користувач потрапляє на сторінку свого профілю зображено на рисунку 3.5.

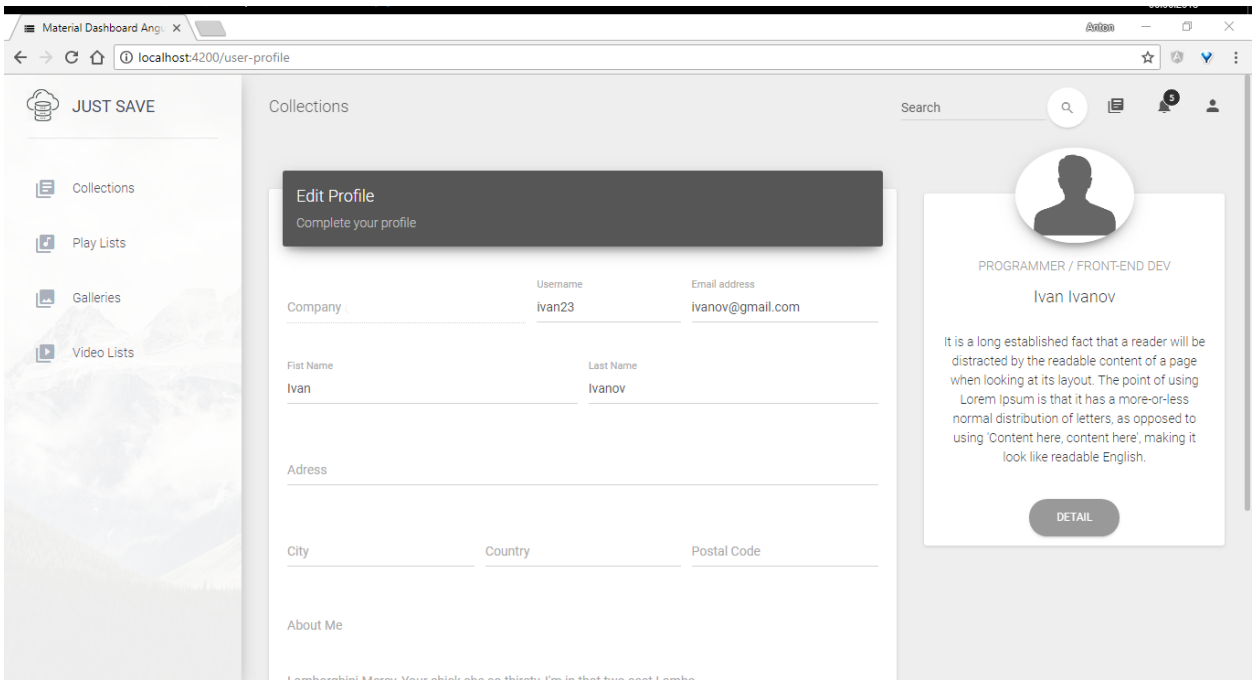


Рисунок 3.5 – Профіль користувача

На цій сторінці користувач може редагувати раніше заповнені дані, такі як, адреса електронної пошти, місце проживання, завантажити фото профілю або ж перейти до деталей, де можна вказати деякі налаштування, змінити пароль тощо.

Створення колекцій:

Для створення колекції користувач має спочатку вибрати тип колекції в головному меню зліва, після чого натиснути на карточку «add collection», після цих маніпуляцій користувач потрапляє на сторінку створення колекції, зображена на рисунку 3.6.

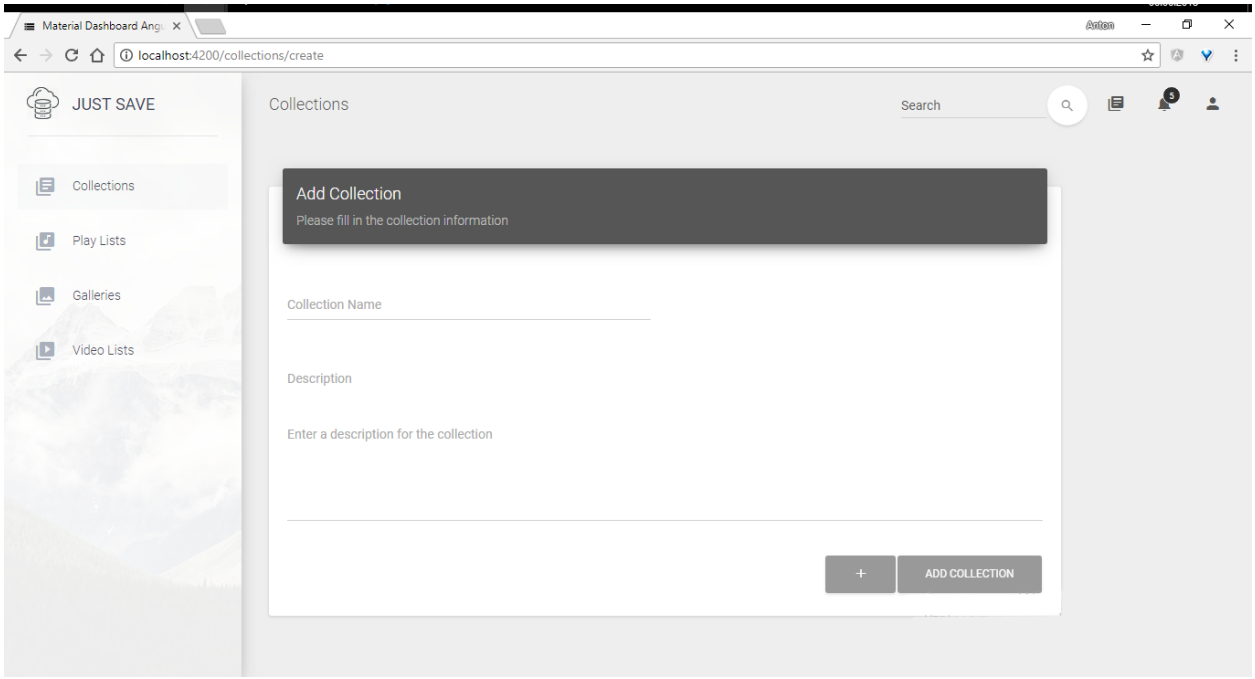


Рисунок 3.6 – Створення колекції

На цьому етапі потрібно заповнити назву колекції та короткий опис колекції, також натиснувши на кнопку «+» можна відразу додати до колекції файли, після чого натиснувши кнопку Add Collection створиться колекція з прикріпленими файлами, якщо вони були додані, або ж пуста без файлів, якщо не було прикріплено жодного файлу. В подальшому до пустої колекції можна додати файли або видалити уже додані з створеної колекції.

Для кожного з типів колекції, окрім файлової, передбачений свій тип відтворення, для аудіоколекції це аудіопрогравач, що відтворює послідовно

усі треки колекції, для галереї – це слайдер, що відображає зображення у вигляді слайд-шоу, для відеоколекції відеопрогравач. При виборі однієї з колекцій користувач потрапляє на сторінку детальної інформації про колекцію, де видно файли, які знаходяться в середині, та є можливість взаємодіяти з ними. Приклад створеної колекції разом з аудіопрогравачем зображено на рисунку 3.7

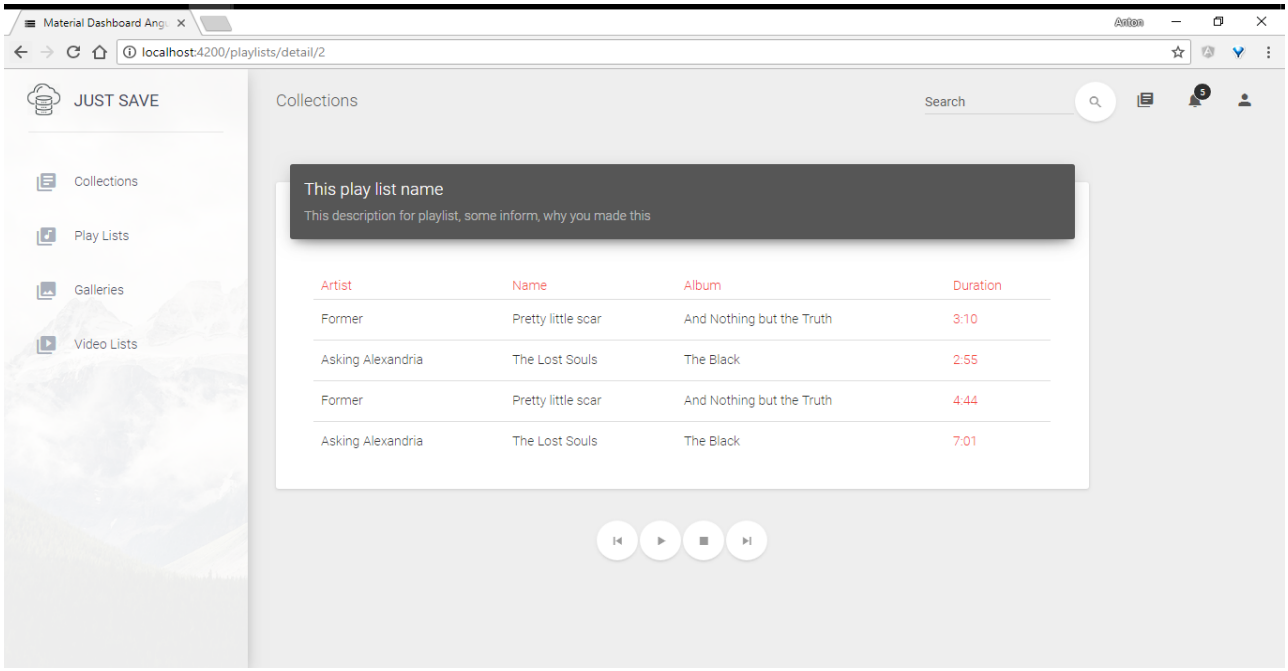


Рисунок 3.7 – Деталі колекції з можливістю відтворення.

3.3 Порівняння з провідними продуктами

В результаті розроблено web-додаток з обмеженням доступу, тобто для використання якого потрібна реєстрація, доступ до функціоналу можливий, використовуючи будь-який пристрій, який має web-браузер, на відміну від таких сервісів як Dropbox та OneDrive, розроблений сервіс не потребує встановлення додаткового програмного забезпечення на ваш комп’ютер, не створюються додаткові папки і не займається додаткове місце на вашому комп’ютері, а усі завантажені дані зберігаються на сервері, що дає миттєву синхронізацію між комп’ютерами. Порівняльна характеристика можливостей та характеристик створеного сервісу, а також ринкових сервісів представлена в таблиці 3.1.

В програмі передбачено створення колекцій файлів різного призначення з відповідними способами запуску файлів, тобто для запуску аудіо не потрібний програвач, все відбувається в web-браузері. Розроблений продукт кросплатформенний і кросдевайсний. А оскільки сервіс реалізований з використанням мови Java, то єдиною вимогою до платформи, на якій буде запускатись сервер є наявність реалізованої для нього Java Virtual Machine.

Таблиця 3.1 – Порівняльна характеристика можливостей сервісів збереження інформації

Характеристика	Spotify	Розроблений сервіс	OneDrive	Google Drive	Dropbox	YouTube
Завантаження файлів	-	+	+	+	+	-
Розміщення файлів	-	+	+	+	+	±
Доступність	$9.99/місяць	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно
Обмеження доступу	+	+	+	+	+	+
Персоналізація	-	+	-	-	-	±
Групування файлів	±	+	-	-	-	±
Тип доступу	Додаток	Web-інтерфейс	Додаток	Web-інтерфейс	Додаток	Web-інтерфейс
Розмір додатка	147мб	–	5,73 Мб	-	39,4 Мб	-
Синхронізація	-	Миттєва	Швидка	Миттєва	Швидка	-
Інтелектуальна рекомендація контенту	Музика	Зображення	–	–	-	Відео

3.4 Висновки

Для тестування back-end створеного web-додатка було написано unit-тести з використанням бібліотеки для модульного тестування програмного забезпечення написаного на мові Java, під назвою jUnit з сімейства бібліотек для різних мов програмування xUnit. Написання тестів дозволило впевнитись в коректності написаного коду, що програма працює як задумувалось. В перспективі при внесенні змін у вихідний код або ж додавання нового функціоналу тести дозволяють переконатись, що внесені зміни або доданий функціонал не вплинув на інший функціонал.

Для тестування front-end частини додатка використовувались Jasmine 2.8 при написанні unit-тестів для класів TypeScript. Protractor 5.1 та Karma 2 для написання і запуску тестів разом з вихідним кодом у реальному браузері. В перспективі тести дають перевагу при зміні або ж доповненні функціоналу, а також, оскільки Angular активно розвивається, тести дозволяють швидко перевірити роботу front-end після оновлення модулів.

Також було створено інструкцію для системного адміністратора по розгортанню сервісу на web-сервері, що допоможе адміністратору в установці та налаштуванні додатка. А також інструкцію користувача, яка дозволяє користувачеві зареєструватись та познайомитись з основами користування сервісом.

Окрім того був проведений аналіз продуктів, представлених на ринку, що покликані виконувати схожі з розробленим сервісом завдання. В результаті порівняння було визначено переваги та недоліки розробленого сервісу.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ДОЦІЛЬНОСТІ СТВОРЕННЯ ОНЛАЙН СЕРВІСУ ДЛЯ ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ

Виконання та впровадження будь-якої науково-дослідної роботи завжди вимагає певних витрат.

Витрати на виробництво та реалізацію будь-якого продукту повинні зменшуватись, цього вимагає процес рентабельного виробництва.

Якщо скорочення витрат на виробництво не спостерігається ніяка науково-технічна розробка не буде реалізована та впроваджена, тому що така розробка не буде мати виграшу у рентабельності, в порівнянні з розробками представленими на ринку, що в свою чергу нівелює необхідність в її впровадженні [27].

На основі економічних розрахунків, можна довести економічну доцільність та ефективність впровадження результатів, що були отримані в результаті виконаних науково-дослідних робіт у виробництві, тобто здійснити комерціалізацію наукових розробок.

Даний розділ магістерської роботи присвячений саме економічному обґрунтуванню впровадження представленої наукової розробки і передбачає такі етапи виконання (рис. 4.1):

Рисунок 4.1 - Складові економічної частини магістерської кваліфікаційної роботи

На такі складові буде поділено економічну частину даної магістерської роботи. Усі подальші економічні розрахунки, будуть висвітлені у згаданих

підрозділах економічної частини. У сукупності ці етапи дозволять побачити цілісну картину доцільності розробки та впровадження результатів.

4.1 Технологічний аудит розробки

Метою проведення оцінювання комерційного потенціалу розробки є оцінювання подальшого потенціалу розробленого в результаті науково-технічного дослідження продукту в комерційній сфері. За результатами оцінювання експертів робиться висновок щодо напрямів організації в майбутньому її впровадження з врахуванням встановленого рейтингу.

Оцінювання комерційного потенціалу розробки будемо здійснювати за 12-ма критеріями, наведеними в таблиці 4.1.

Таблиця 4.1 - Оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Багато аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
Практична здійсненність					
7	Активна конкуренція компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
8	Відсутні фахівці як з технічної, так і з комерційної реалізації	Необхідно наймати фахівців або витрачати значні кошти та час на навчання	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, та джерела які відсутні.	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси.Джерела фінансування є	Потрібні незначніфінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років.Термін окупності більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років.Термін окупності більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років.Термін окупності менше 3-х років

Закінчення таблиці 4.1

Критерій	0	1	2	3	4
12	Необхідно регламентні документи та велика кількість дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Спираючись на дані складеної таблиці ряду незалежних експертів, у нашому випадку керівник магістерської роботи та викладачі випускової кафедри, було запропоновано провести аналіз запланованого до розробки програмного продукту.

Після проведення аналізу і ознайомлення з продуктом незалежні експерти провели оцінку продукту і керуючись таблицею наведеною вище виставили наступну кількість балів.

В результаті оцінки було розраховано середню кількість балів кожного з викладачів, а також загальну середню кількість балів.

В результаті оцінки комерційного потенціалу отримані від експертів вбули внесені в таблицю 4.2

Таблиця 4.2 - Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 Семеренко В. П., к.т.н., доц. кафедри ОТ	2 Ткаченко О.М., к.т.н., доц. кафедри ОТ	3 Савицька Л.А., к.т.н., доц. кафедри ОТ
	Бали, виставлені експертами:		
1	2	2	2
2	4	4	4
3	4	4	3
4	4	4	4
5	3	3	3
6	4	4	4
7	3	3	3
8	4	4	3
9	2	2	3
10	3	3	3
11	4	3	3
12	4	4	4
Сума балів	$СБ_1 = 41$	$СБ_1 = 40$	$СБ_1 = 39$
Середньо-арифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{41+40+39}{3} = \frac{120}{3} = 40$		

Використовуючи дані таблиці 4.2, а також згідно із рекомендаціями, що наведені в таблиці 4.3, можна робити висновки, щодо рівня рівня

комерційного потенціалу, та доцільності впровадження результатів даної наукової роботи а також про подальші перспективи розвитку розробленого сервісу.

Таблиця 4.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 - 10	Низький
11 - 20	Нижче середнього
21 - 30	Середній
31 - 40	Вище середнього
41 - 48	Високий

Взявши до уваги, середньоарифметичну суму балів, $\overline{СБ} = 40$, що були виставлені експертами, можна стверджувати, що рівень комерційного потенціалу даної розробки є вище середнього.

В третьому розділі було проведено порівняння розробленого сервісу з провідними аналогами що представлені на ринку сьогодні, які надають схожі за функціоналом послуги і було виявлено, що найближчим аналогом представленим сьогодні на ринку виступає сервіс для збереження інформації Google Drive, тому в цьому розділі проведемо порівняння розробленого продукту з даним сервісом.

Результати порівняння розробленого продукту і сервісу для збереження інформації Google Drive наведено у таблиці 4.4

Таблиця 4.4 – Порівняння з провідним аналогом на ринку

	Розроблений сервіс	Google Drive
Зручність збереження файлів	80%	70%
Розмір безкоштовного простору	90%	80%
Розбиття файлів на категорії	100%	0%
Інтелектуальні рекомендації контенту	100%	0%
Персоналізація	90%	10%
Розповсюдженість	0%	80%

У порівнянні з вузько направленим сервісом Google Drive, розроблений сервіс має ширші можливості для збереження файлів, а саме можливість для розбиття збережених даних по категоріям за типами файлів. Розмір безкоштовного простору, що пропонує Google Drive, обмежений на цифрі в 15 GB, що продиктовано широким розповсюженням сервісу та його великим обсягом користувачів. В свою чергу розроблений сервіс не має таких обмежень, оскільки охоплює значно меньшу частину аудиторії.

Оскільки Google Drive вузькоспеціалізований сервіс що імітує локальний диск комп’ютера, однак зберігає данні в хмарному сховищі, то має дещо обмежений і вузькоспеціалізований функціонал. В розробленому сервісі цей недолік виправлений, окрім розбиття файлів на категорії, можливості створення колекцій і галерей, доданий функціонал для інтелектуального підбору контенту для користувача. Що значно розширює область застосування розробленого сервісу.

Одною з значних недоліків розробленого сервісу, на відміну від Google Drive, є відсутність широкої аудиторії і як результат не значна кількість користувачів.

Для розповсюдження продукту та розширення бази користувачів, заплановано використання рекламної компанії з залученням сторонніх сервісів соціальної направленості таких як, Instagram, YouTube та використання платформи Facebook.

Розроблений сервіс пропонується не як аналог представлених на ринку сервісів, з розширеними можливостями, а як принципово новий підхід що об'єднує характеристики різних сервісів в один новий сервіс, що покликаний забезпечити користувачів необхідним функціоналом в одному місці, без необхідності заводити кілька десятків профілів в різних сервісах, а також запропонувати окрім функціоналу рекомендації контенту, спеціально підібраного для конкретного користувача.

За результатами порівняння можна зробити висновок, що розроблений сервіс має переваги перед сервісами, представленими на ринку, тому даний сервіс може бути конкурентно спроможним. Прогнозований прибуток пропонується отримувати за рахунок продажу додаткового дискового простору, та реклами.

Потенційними покупцями можуть бути усі користувачі, що мають потребу у збереженні та синхронізації даних онлайн, а також користувачі зацікавленні в перегляді соціального контенту.

Розробака програмного забезпечення запланована з залученням інвестиційних коштів, з подальшими поверненням коштів та виплатами дивідендів інвесторам.

4.2 Прогнозування витрат на виконання та впровадження результатів наукової роботи

У магістерській кваліфікаційній роботі розглядається програмне забезпечення для збереженя користувацьких даних та їх обробки, тому

значна частина витрат - це витрати на розробку та на оренду дискового простору, а не на виробництво і відтворення. Звідси, й певна специфіка розрахунків [27].

Основна заробітна плата розробників, які працюють над проектом визначається за формулою 4.1:

$$Зо = \frac{М}{Тр} \cdot t \text{ [грн]}, \quad (4.1)$$

де М - місячний посадовий оклад розробника;

Тр - число робочих днів в місяці (T_p= 22дні);

t - число днів роботи розробника.

Над створенням розробки працювали керівник проекту та інженер-програміст , отже, виконаємо для них всі необхідні розрахунки:

$$Зо = \frac{13000}{22} \cdot 3 = 1772{,}73 \text{ (грн)}.$$

$$Зо = \frac{11000}{22} \cdot 66 = 33000 \text{ (грн)}.$$

Таблиця 4.5- Заробітна плата

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
1 Керівник	13000	590,91	3	1772,73
2 Старший інженер- програміст	11000	500	66	33000
Всього				34772,73

Додаткова заробітна плата (Зд) всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = (10 \dots 12\%) \cdot Зо \text{ [грн]}, \quad (4.2)$$

де Зо - основана заробітна плата.

$$Зд = \frac{11 \cdot 34772{,}73}{100} = 3825(\text{грн}).$$

Нарахування на заробітну плату (Нзп) розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Нзп = 22\% \cdot (Зо + Зд) \quad (4.3)$$

$$Нзп = \frac{22 \cdot (34772{,}73 + 3825)}{100} = 8491{,}5 \text{ (грн)}.$$

Амортизація обладнання, комп’ютерів та приміщень (А), які використовувались під час виконання даного етапу роботи.

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо[1].

У спрощеному вигляді амортизаційні відрахування (А) в цілому бути розраховані за формулою 4.4:

$$А = \frac{Ц \cdot На}{100} \cdot \frac{Т}{12} \text{ [грн]}, \quad (4.4)$$

де Ц - загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн;

На - річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що На = (10...25)%;

Т - термін, використання обладнання, приміщень тощо, місяці.

Всі розрахунки зводимо до таблиці 4.5.

Таблиця 4.6 - Амортизація обладнання та приміщень

Найменування обладнання, приміщень	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн.
ЕОМ	10000	20%	3	500
Приміщення	125000	15%	3	4687,5
Всього				5187,5

У магістерській кваліфікаційній роботі розробляється програмний продукт, який для споживача буде поширюватися через інтернет, а саме через веб-сторінку.

При розробці сторінки до послуг виробничого характеру сторонніх підприємств можна віднести надавання послуги «Хостинг», а також направлення обраного доменного імені на сервери хосту, тому на протязі всього існування проекту за ці послуги, щорічно потрібно сплачувати абонентську плату.

Таблиця 4.7 - Послуги, що використовуються при виготовленні програми

Найменування комплектуючих (робіт, послуг)	Кількість, шт.	Ціна за одиницю, грн.	Сума, грн.
1. Послуга «Хостинг», шт.	1	500	500
2. Послуга «Доменне ім'я», шт.	1	120	120
Всього	620 грн.		

Витрати на силову електроенергію $В_е$, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою 4.6:

$$В_е = В \cdot П \cdot Ф \cdot К_п [грн], \quad (4.6)$$

де В - вартість 1 кВт електроенергії, грн.;

П - установлена потужність обладнання, кВт/год;

Ф - фактична кількість годин роботи обладнання, яке задіяне на виготовлення одного виробу, годин;

$К_п$ - коефіцієнт використання потужності, $К_п \leq 1$.

$$В_е = 2{,}46 \cdot 0{,}12 \cdot 528 \cdot 0{,}7 = 109{,}11 \text{ (грн)}.$$

Інші витрати охоплюють: загально виробничі витрати (витрати управління організацією, ремонт та експлуатація основних засобів, витрати на опалення, освітлення тощо), адміністративні витрати (проведення зборів, оплата юридичних та аудиторських послуг, тощо), витрати на збут (витрати на рекламу, перепідготовка кадрів) на інші операційні витрати (штрафи, пені, матеріальні допомоги, втрати від знецінення запасів тощо).

Інші витрати можна розрахувати за нормативами відносно основної заробітної плати основних робітників, які виготовляють продукцію, за формулою 4.7

$$В_{ін} = Н \cdot Зо \text{ [грн]}, \quad (4.7)$$

де Н - норматив загальновиробничих витрат. Для ЕОМ Н = 230%.

$$В_{ін} = \frac{230 \cdot 34772{,}73}{100} = 79977{,}28(\text{грн}).$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи - В.

$$В = 34772{,}73 + 3825 + 8491{,}5 + 5187{,}5 + 620 + 109{,}11 + 79977{,}28 = 132983{,}12(\text{грн}).$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \text{ [грн]}, \quad (4.8)$$

де β - коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Оскільки, розробка знаходиться на стадії впровадження, то $\beta \approx 0{,}9$;

$В_{заг}$ - загальна вартість всієї наукової роботи. У даному випадку $В_{заг} = В$.

$$ЗВ = \frac{132983{,}12}{0{,}9} = 147759{,}02(\text{грн}).$$

Отже, розрахований кошторис витрат на розробку програмного забезпечення для зберігання і обробки користувацьких даних складає 147759,02грн.

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі здійснено прогнозування, яку вигоду можна отримати у майбутньому від впровадження результатів даної наукової роботи.

Передбачається, що виконання наукової роботи та впровадження результатів по розробці програмного забезпечення для зберігання і обробки користувацьких даних займе 1 рік.

Основні позитивні результати від впровадження розробки очікуються протягом 3 років після її впровадження.

Саме зростання чистого прибутку забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

$$\Delta П_{\text{і}} = \sum_{1}^{n} (\Delta П_{\text{я}} \cdot N + П_{\text{я}} \Delta N)_i \, , [\text{грн}], \qquad (4.9)$$

де $\Delta П_{\text{я}}$ - покращення основного якісного показника від впровадження результатів розробки у даному році;

N - основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN - покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_{\text{я}}$ - основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n - кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки покращується якість програмного продукту, що дозволяє підвищити ціну його реалізації на 100 грн., а кількість потенційних користувачів ресурсу збільшиться: протягом першого року - на 1000 шт., протягом другого року - ще на 2000 шт., протягом третього року - ще на 4000 шт.

Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 200 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки -200 грн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства протягом наступних трьох років складе:

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом першого року складе:

$$\Delta П_1 = 20 \cdot 200 \ + (200 + 100) \cdot 1000 = 304000 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом другого року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_2 = 20 \cdot 200 \ + (200 + 100) \cdot 2000 = 604000 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом третього року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_3 = 20 \cdot 200 \ + (200 + 100) \cdot 4000 = 1204000 \text{ (грн)}.$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахований комерційний ефект можливого впровадження розробки ще не означає, що розробка буде впроваджена. Якщо збільшення прогнозованого прибутку від впровадження результатів наукової розробки є вигідним для підприємства, то це ще не означає, що вона зацікавить інвесторів. Основні показники, що визначають рентабельність фінансування розробки певним інвестором, є відносна і абсолютна ефективність вкладених інвестицій та термін їх окупності [28].

Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт:

1-й крок. Розраховуємо теперішню вартість інвестицій PV, що вкладаються в наукову розробку. Такою вартістю, можна вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, розраховану нами раніше за формулою (4.8), тобто будемо вважати, що ЗВ = PV=147759,02.

2-й крок. Розрахуємо очікуване збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження.

3-й крок. Для спрощення подальших розрахунків побудуємо вісь часу, на яку нанесемо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів.

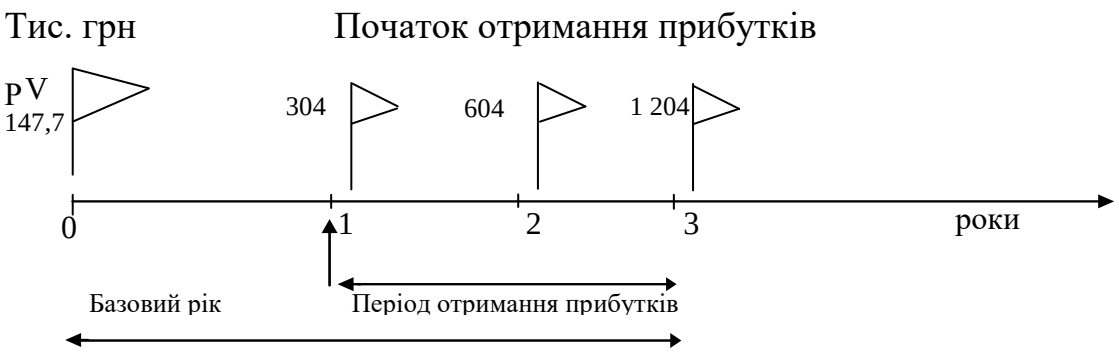
Платежі показуються у ті терміни, коли вони здійснюються. Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рис. 4.2.

4-й крок. Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$.

Для цього користуються формулою:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV), \quad (4.10)$$

де ПП - приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн;



Життєвий цикл наукової розробки $T_ж$

Рисунок 4.2 - Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

PV - теперішня вартість інвестицій PV = ЗВ=147759,02(грн).

У свою чергу, приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$\text{ПП} = \sum_{1}^{\text{T}} \frac{\Delta\text{П}_\text{i}}{(1+\tau)^t}, [\text{грн}], \quad (4.11)$$

де $\Delta\text{П}_\text{i}$ - збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

Т - період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ - ставка дисконтування [5], за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t - період часу (в роках) від моменту отримання чистого прибутку до точки «0».

Отримаємо:

$$ПП = \frac{304000}{(1+0,1)^1} + \frac{604000}{(1+0,1)^2} + \frac{1204000}{(1+0,1)^3} = 276363,64 + 499173,55 + 904583,02 = 1680120,21 \text{ (грн)}.$$

Тоді $E_{абс} = (1680120,21 - 147759,02\ ) = 1532361,19$ (грн).

Оскільки $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження може принести прибуток, але це також ще не гарантує те, що інвестор зацікавиться у фінансуванні даної роботи.

5-й крок. Розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1 \qquad (4.12)$$

де $E_{абс}$ - абсолютна ефективність вкладених інвестицій, грн;

PV - теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ - життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{1532361,19}{147759,02}} - 1 = 2,249 - 1 = 124,9\ \%$$

Далі, розрахована величина $E_в$ порівнюється з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$t = d + f\ [\%], \qquad (4.13)$$

де d - середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = (0,19...0,22);

f - показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,1), але може бути і значно більше.

$$t = d + f = 0{,}2 + 0{,}1 = 0{,}3 = 30\%$$

Величина $E_в > \tau_{мін}$, інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених коштів у реалізацію наукового проекту за формулою:

$$T_{ок} = \frac{1}{E_в} \text{ [років]}, \qquad (4.14)$$

$$T_{ок} = \frac{1}{1{,}249} = 0{,}8 \text{ роки.}$$

Оскільки $T_{ок}$ = 0,8 роки, то розробка являється доцільною.

4.5 Висновки економічного обгрунтування

У даному розділі магістерської кваліфікаційної роботи здійснено розрахунки, які доводять прибутковість та ефективність впровадження нового продукту.

Проведено оцінювання комерційного потенціалу розробки. На основі компетентної думки експертів було сформовано систему критеріїв та за 5-ти бальною шкалою, виставлено бали по кожному з них. Виставлені бали, показують, що рівень комерційного потенціалу є вище середнього.

Розраховано витрати на розробку. Розрахований кошторис витрат на розробку склав 147759,02грн.

Були спрогнозовані комерційні ефекти від реалізації розробки, тобто який дохід, можна отримати у майбутньому від впровадження виконаної наукової роботи. Доведено, що розробка отримає вигоду від впровадження.

Розраховано основі показники, які визначають доцільність фінансування наукової розробки інвестором. Такими показниками є абсолютна та відносна ефективність вкладених інвестицій, а також термін їх окупності.

Обрахована абсолютна ефективність становить 1532361,19 грн, що свідчить про те, що інвестор буде зацікавлений у фінансуванні даної розробки. Відносна (щорічна) ефективність становить 124,9 %, що більше мінімальної ставки дисконтування, що ще раз підтверджує зацікавленість інвестора.

Термін окупності вкладених коштів у реалізацію наукового проекту становить 0,8 роки, що означає, що вкладені кошти повернуться, приблизно, через 10 місяців.

Отже, можна стверджувати, що фінансування даної розробки є доцільним.

ВИСНОВКИ

В результаті виконання магістерської роботи на тему “Онлайн-сервіс з обмеженням доступу для зберігання даних та їх попереднього обробляння” було проведено аналіз існуючих технологій та методів створення web-додатків, зокрема, актуальних технологій для побудови інтерфейсів, а також високопродуктивних web-серверів і моделей їх роботи, що дозволило побудувати дружній для користувача інтерфейс, та забезпечити високу продуктивність на стороні back-end.

Розглянуто актуальні методи збереження та обробки інформації, а також технології, що забезпечують консистентність даних та високу продуктивність при записі або зчитуванні даних. Це дозволяє зробити обмін з базою даних швидким, а також захистити користувацькі дані від втрати.

Було розроблено web-сервіс що дозволяє зберігати дані користувачів, надає можливість зручного групування даних за певною ознакою, дозволяє обмежити доступ до цих даних, а також надає можливості попереднього відтворення. Для розробки сервера було використано мову програмування Java разом з фреймворком Spring Boot в основі якого лежить MVC-архітектура, використання MVC-архітектури дозволило побудувати продуктивний додаток, а використання мови Java разом з феріймворком Spring Boot зробити сервер не лише платформо незалежним, а й пришвидшити його розробку. Написання unit-тестів для Java-класів з використанням бібліотеки JUnit, дозволило бути впевненим в працездатності програми після внесення змін.

Використання реляційної бази даних з відкритим вихідним кодом PostrgeSQL разом з ORM Hibernate дозволило швидко організувати взаємодію з базою даних і не зациклюватись на написанні SQL-запитів, окрім того зменшило потенційні трудовитрати при потребі переходу від PostgreSQL до іншої реляційної бази.

При побудові front-end було використано мову програмування TypeScript та фреймворк від компанії Google орієнтований на розробку

складних інтерфейсів Angular 6, що дозволило побудувати зручний і продуктивний інтерфейс з широкими можливостями для масштабування. Для написання unit-тестів TypeScript класів було використано ферймворк Jasmine, за виконання вихідного коду та тестів відповідає консольний інструмент Karma. Для написання і запуску комплексних e2e тестів було використано фреймворк Protractor, що дозволяє запускати e2e тести, які імітують дії користувача в реальному браузері.

Також проведене тестування вихідного коду на відповідність рекомендаціям до написання коду від розробників Angular інтегрованим в Angular CLI консольним інструментом Lin. Таке тестування гарантує відповідність коду рекомендаціям і зменшує ймовірність потенційних та неочевидних помилок.

В ході виконання магістерської роботи було створено інструкції для системного адміністратора, що дозволить спростити установку програми на сервер, а також інструкція для користувача, яка дозволить кінцевому користувачеві познайомитись з основами використання сервісу.

Порівняння з провідними продуктами представленими на ринку і призначеними виконувати схожі задачі, показало, що більшість сервісів покликано виконувати вузькоспеціалізовані задачі, розробкою таких сервісів займаються команди програмістів, що несе за собою значні фінансові витрати. Тому розроблений сервіс має переваги у вартості розробки, переваги у виконанні широкого спектру задач, а також функціональні вдосконалення, призначені для підвищення комфорту використання додатку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Молодь в науці: дослідження, проблеми, перспективи (МН-2020) [Електронний ресурс] / Котик А.М. 2019 – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2020/paper/view/8413

2 Лекції з Веб-програмування [Електронний ресурс] / Грійо-Тукало О.Ф 2017 – Режим доступу до ресурсу: https://github.com/Ot-WebCourse/web_lections

3 Mark M. REST API Design Rulebook / Mark Masse. – USA : O'Reilly Media, 2012. – 96 с.

4 Крейг У. Spring в действии. / Крейг Уоллс. – М.: ДМК Пресс, 2013. – 752 с.: ил.

5 Выбор технологий для большого и не очень большого web-проекта. [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/company/SECL_GROUP/blog/315734/

6 Архитектура высоких нагрузок [Електронний ресурс] –Режим доступу до ресурсу: https://ruhighload.com/post/Архитектура+высоких+нагрузок

7 Brain M. Web API Design - Crafting Interfaces that Developers Love. / Brain Mulloy. – Google Group: 2011 – 38c.

8 Progresive web Apps [Електронний ресурс] – Режим доступу до ресурсу: https://netpeak.net/ru/blog/chto-takoe-progressive-web-apps-i-kakie-vozmozhnosti-oni-otkryvayut-dlya-vashego-biznesa/

9 Знакомство с Angular [Електронний ресурс] – Режим доступу до ресурсу: http://ogrik2.ru/b/fajn-ya-moiseev-a/angular-i-typescript-sajtostroenie-dlya-professionalov/31725/1-znakomstvo-s-angular/5

10 Architecture overview [Електронний ресурс] –Режим доступу до ресурсу: https://angular.io/guide/architecture

11 Фуллер М. Архитектура корпоративных программных приложений / Мартин Фуллер. – Москва – Санкт –Петербург – Київ: Вильямс, 2006 – 541с.

12 Шарапов Р.В. Апаратные средства хранения больших обёмов данных / Шарапов Р.В. – 2011 – 5с.

13 Системы хранения данных [Електронний ресурс] –Режим доступу до ресурсу: https://shalaginov.com/2014/10/21

14 Введение в ORM- [Електронний ресурс] – Режим доступу до ресурсу: http://internetka.in.ua/orm-info

15 Backend infrastructure at Spotify [Електронний ресурс] – Режим доступу до ресурсу: https://labs.spotify.com/2013/03/15/backend-infrastructure-at-spotify/

16 Архитектура YouTube [Електронний ресурс] – Режим доступу до ресурсу: https://www.insight-it.ru/highload/2008/arkhitektura-youtube/

17 Особенности архитектуры распределённого хранилища в Dropbox [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/321274/

18 Google Drive [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Google_Drive

19 Спинелис Д. Идеальная Архитектура ведущие спецыалисты о красоте программных архитектур. / Спинелис Д., Гусиос Г. – Санкт-Петербург – Москва: Символ-Плюс, 2010 – 529с.

20 Spring Boot [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-boot

21 Daniel M. B. Software Engineering for Modern Web Applications: Methodologies and Technologies / Daniel M. Brandon. – USA : Christian Brothers University, 2008 – 380c

22 Риккарди Г. Системы баз данных. Теория и практика использования в Internet и среде Java / Грег Риккарди.– Москва – Санкт –Петербург – Київ :Вильямс 2001 – 477с.

23 Ng-book The Complete Book on Angular 5 [Filipe Coury, Ari Lerner, Nate Murray, Carlos Taborda].– Leanpub, 2017 – 873 с.

24 Внедрение Зависимости (Dependency Injection) [Електронний ресурс] – Режим доступу до ресурсу: http://designpatternsphp.readthedocs.io/ru/latest/Structural/DependencyInjection/README.html

25 Тестирование Angular: введение для разработчика [Електронний ресурс] –Режим доступу до ресурсу: https://webformyself.com/testirovanie-angular-vvedenie-dlya-razrabotchika/

26 Hunt A. Pragmatic Unit Testing in Java with Junit / Hunt A., Thomas D.– USA: The Pragmatic Programmers, 2003 – 136с.

27 Економіка підприємства. Навчальний посібник. / Гринчуцький В.І.– ЦНЛ, 2018 – 304с.