НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
"___" __червня 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системне програмування»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: Cloud-based модуль розширеного пошуку документів на мікросервісній REST архітектурі. _____
_____

Виконав (-ла):

студент (-ка) IV курсу, групи КВ-62___
(шифр групи)

Штефанович Георгій Миколайович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доц.каф.СПСКС, к.т.н., доцент, Потапова К.Р. ____ _____
_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

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

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені
ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

«___» червня 2020 р.

ЗАВДАННЯ на дипломний
проєкт студента Штефановича
Георгія Миколайовича
(прізвище, ім'я, по батькові)

1. Тема проєкту «Cloud-based модуль розширеного пошуку документів на мікросервісній REST архітектурі», керівник проєкту Потапова Катерина Романівна, к.т.н., доц.каф.СПСКС,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «25» травня 2020 р. №1181-С

2. Термін подання студентом проєкту: дивись технічне завдання.

3. Вихідні дані до проєкту: Назва. Cloud-based модуль розширеного пошуку документів на мікросервісній REST архітектурі.

4. Зміст пояснювальної записки: перелік скорочень, умовних позначень та термінів; вступ; опис, аналіз та обгрунтування вибору мов та технологій для розробки серверної та клієнтської частини системи; аналіз існуючих рішень; опис розроблених підпрограм та алгоритмів; аналіз та тестування розробленої системи; висновки; список використаних літературних джерел.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) презентація; структурні схеми: структурна схема модулів системи, структура бахи; схеми взаємодії: пошук тексту, взаємодія модулів.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Клятченко Я.М., доц.каф.СПСКС, к.т.н.	17.05.2020	20.05.2020

7. Дата видачі завдання 12.11.2019.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проекту	18.11.2019	
2.	Розроблення та узгодження технічного завдання	28.01.2020	
3.	Розробка структури серверної та клієнтської частини	04.02.2020	
4.	Дизайн серверної частини	14.03.2020	
5.	Дизайн клієнтської частини	10.04.2020	
6.	Тестування всієї системи	24.04.2020	
7.	Підготовка матеріалів текстової та графічної частини проекту	30.04.2020	
8.	Оформлення технічної документації проекту	27.05.2020	

Студент _____ Штефанович Георгій
(підпис) (Ім’я та ПРІЗВИЩЕ)

Керівник проєкту _____ Катерина Потапова
(підпис) (Ім’я та ПРІЗВИЩЕ)

 Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Даний дипломний проект присвячується розробці програмного модулю для зручнішого та прогресивнішого користування існуючими сучасними програмними рішеннями для збереження документів.

Даний модуль складається із двох осоновних частин: серверної та клієнтської. Серверна частина, в свою чергу, складається з двох частин – локальної та віддаленої. Віддалена частина – Google Cloud знаходиться на серверах корпорації Google та використовується виключно у якості місця сховища, а локальна, як уже було сказано вище, знаходиться на власному пристрої та виконує основні задачі та алгоритми проекту. Також локальна серверна частина містить свою реляційну сучасну базу даних, для збереження метаінформації про систему, її користувачів та вміст.

Клієнтська програма – десктопний застосунок, призначений для додавання, видалення, перегляду та отримання документів. Документи можна отримувати за допомогою різних параметрів, таких як відсоток унікальності серед інших файлів, дата оновлення, шаблон назви і тд. Основне завдання серверної частини – формування бази даних з інформацією про унікальність документу в середньому та схожості його з іншими файлами, а клієнтської частини – забезпечення користувача сформованою інформацією та надання доступу до неї.

Результатом виконання даного дипломного проекту стали: розроблена системна архітектура, алгоритм взаємодії різних програмних компонентів та систем, алгоритм захисту даного програмого модуля та розроблений зручний інтерфейс для прикладу користування даним проектом.

ABSTRACT

This diploma project is dedicated to the development of a software module for more convenient and progressive use of existing modern software solutions for saving documents.

This module consists of two main parts: server and client. The server part, in turn, consists of two parts - local and remote. The remote part - Google Cloud is located on Google's servers and is used exclusively as a storage location. A local server, as mentioned above, is on your own device and performs the main tasks and algorithms of the project. Also, the local server part contains its non-relational modern database, to store meta-information about the system, its users, and content.

The client is a desktop application designed to add, delete, view, and retrieve documents. Documents can be retrieved using various parameters, such as the percentage of uniqueness among other files, the updated date, the name template, and so on. The main task of the server part is to form a database with information about the uniqueness of the document on average and its similarity to other files, and the client part - to provide the user with the generated information and provide access to it.

As a result of work on this project are: developed system architecture, an algorithm of the interaction of various software components and systems, an algorithm of protection of this program module, and developed a user-friendly interface for an example of using this project.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045480.002 ТЗ	Cloud-based модуль	4		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Технічне завдання			
	А4	ІАЛЦ.045480.003 ТП	Cloud-based модуль	1		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Відомість технічного			
			проєкту			
	А4	ІАЛЦ.045480.004 ПЗ	Cloud-based модуль	65		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Пояснювальна записка			
	А4	ІАЛЦ.045480.005 Д1	Cloud-based модуль	1		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Структура модулів системи			
			Схема структурна			

Зм	Лист	№ докум.	Підп	Дата	ІАЛЦ.045480.001 ОА		
Розроб.		Штефанович Г.М.			Cloud-based модуль розширеного пошуку документів на мікросервісній REST архітектурі.	Літ.	Лист 1 / Листів 2
Перев.		Потапова К.Р.					
Н. контр.		Клятченко Я.М.			*Опис альбому*	КПІ ім. Ігоря Сікорського, ФПМ, КВ-62	
Затв.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.006 Д2	Cloud-based модуль	1		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Взаємодія модулів			
			Схема взаємодії			
	А4	ІАЛЦ.045480.007 Д3	Cloud-based модуль	1		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Архітектура бази			
			Схема структурна			
	А4	ІАЛЦ.045480.008 Д4	Cloud-based модуль	1		
			розширеного пошуку			
			документів на			
			мікросервісній REST			
			архітектурі.			
			Пошук тексту			
			Схема взаємодії			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «Cloud-based модуль з розширеного пошуку документів на мікросервісній REST архітектурі»

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування, затверджене кафедрою системного програмування та спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для людей, чия професія вимагає постійної роботи з великою кількістю документів для пришвидшення ефективності їхньої праці за рахунок інформації, яку надає даний програмний модуль, що дозволяє користувачу визначити чи існує уже подібний матеріал, і який дозволяє витягнути підходящі.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1 Серверна чатина повинна забезпечувати такі основні функції:

- Збереження документів;
- Виведення метаінформації документів;
- Видалення існуючих документів;
- Оновлення існуючих документів;
- Вилучення документів за фільтром;
- Порівнювання документів відносно один одного.

4.2 Клієнтський застосунок повинен забезпечувати такі основні функції:

- Перегляд документів;
- Перегляд інформації про документи;
- Налаштування особливостей відображення документів;
- Завантаження та вилучення документів.

Розробку клієнтського застосунку виконати за допомогою технології Java FX.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

- Пояснювальна записка;
- Програма та методика тестування;
- Керівництво користувача;
- Схеми та діаграми:
  - «Архітектура проекту. Схема взаємодії компонентів системи»;
  - «Ієрархія взяємодії компонентів коду».

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	12.11.2019
2.	Вивчення літератури за тематикою роботи	14.11.2019
3.	Розробка структури серверної та клієнтської частини	04.02.2020
4.	Дизайн серверної частини	14.03.2020
5.	Дизайн клієнтської частини	10.04.2020
6.	Тестування всієї системи	24.04.2020
7.	Підготовка матеріалів текстової частини проекту	30.04.2020
8.	Підготовка матеріалів графічної частини проекту	17.05.2020
9.	Оформлення технічної документації проекту	05.06.2020

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Форм	Позначення	Найменування	Кількіс листі	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.045480.004 ПЗ	Пояснювальна записка	65	
3	А4	ІАЛЦ.045480.005 Д1	Структура модулів системи	1	
4	А4	ІАЛЦ.045480.006 Д2	Взаємодія модулів системи	1	
5	А4	ІАЛЦ.045480.007 Д3	Структура бази даних	1	
6	А4	ІАЛЦ.045480.008 Д4	Пошук тексту	1	

				ІАЛЦ.045480.003		
	ПІБ	Підп.	Дата			
Розробн.	Штефанович Г.М.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Потапова К.Р.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-62	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Романкевич В.О.					

Пояснювальна записка
до дипломного проєкту

на тему:Cloud-based модуль з розширеного пошуку документів на мікросервісній REST архітектурі

Київ – 2020 року

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Авторизація – керування рівнями та засобами доступу до певного захищеного ресурсу.

Автентифіка́ція – процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора.

Бібліотека – це сукупність ресурсів, що використовуються комп'ютерними програмами, часто для розробки програмного забезпечення.

Валідація – процес підтвердження відповідності або надання законної сили.

Веб-сервіс – це модуль коду, що дозволяє різним електронним пристроям комунікувати один з одним через мережу.

Відлагодження - це процес пошуку та усунення дефектів чи проблем у комп'ютерній програмі, які перешкоджають правильній роботі програмного забезпечення комп'ютера або системи.

Ідентифікація – встановлення тотожності невідомого об'єкта відомому на підставі збігу ознак; розпізнання.

Патерн – це опис або шаблон того, як вирішити проблему, яку можна використовувати у багатьох різних ситуаціях.

Компіляція – трансляція вихідного коду програми в бінарний код.

Конфігурація – сукупність налаштувань програми, що задається користувачем, а також процес зміни цих налаштувань відповідно до потреб користувача.

Клієнт – апаратний або програмний компонент обчислювальної системи, який надсилає запити серверу.

Кросплатформність – властивість програмного забезпечення працювати більш ніж на одній програмній або апаратній платформі.

Модульне тестування - це метод тестування програмного забезпечення, за допомогою якого окремі одиниці вихідного коду, набори одного або

декількох модулів комп'ютерної програми із вхідними даними, процедурами та функціями, щоб визначити, чи придатні вони для використання.

Тест-кейс – це ситуація, яка перевіряє конкретну умова з вимог.

Сервер – програма, що надає деякі послуги іншим програмам (клієнтам).

Фреймворк – заготовки, шаблони для програмної платформи, що визначають архітектуру програмної системи.

Хост – Будь-який комп'ютерний пристрій, що має доступ до IP мережі тобто синонім терміна вузол мережі.

API – опис способів, якими одна комп'ютерна програма може взаємодіяти з іншою програмою.

ООР - це модель комп'ютерного програмування, яка організовує дизайн програмного забезпечення навколо даних або об'єктів, а не функцій та логіки.

ВСТУП

У сучасному світі існує досить багато варіантів файлових менеджерів, проте одного разу, вивчаючи алгоритми нечіткого пошуку, я зіткнувся з думкою, що не зустрічав ні разу файлового менеджера, який уміє порівнювати між собою документи та відображає користувачу статистику використання. Поосмисливши, я зрозумів, що це був би чудовий інструмент, на приклад для викладачів університетів, вчителів у школах, студентів, юристів та людей із багатьох інших професіональних сфер, загалом, людей які постійно мають справу із текстовими документами.

Такий програмний продукт може бути дуже корисним для людей, адже людина, якій потрібно створити новий документ, може зберегти собі багато часу завдяки тому, що знайде уже існуючий або дуже схожий файл на той, що потрібний. Звісно, якщо такий продукт допоможе спеціалісту заощадити час – значить збільшиться продуктивність і ефективність його роботи, а значить і прибуток.

Клієнтський застосунок володіє зручним і дружнім для користувача інтерфейсом та, завдяки сучасним оптимізованим рішенням зі сторони серверної частини, вражає швидкістю роботи. Однією із переваг даного модуля являється універсальний інтерфейс серверної частини: REST архітектура. Такий архітектурний підхід дозволя створювати будь які клієнтські застосунки, починаючи, банально від простого консольного рішення та закінчуючи веб сайтом, і при цьому, взагалі не важливо яка мова програмування буде використовуватися для написання клієнтської частини.

Отже, метою даного проєкта є:

- Створення нового виду файлового менеджера
- Створення універсального, гнучкого рішення
- Покращення ефективності роботи із документами користувачів даної системи

1. ОПИС, АНАЛІЗ ТА ОБГРУНУТВАННЯ ВИБОРУ МОВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ

1.1. Опис та аналіз мови Java 11

Java - це мова програмування загального призначення, яка являється об'єктно-орієнтованою та розроблена таким чином, щоб мати якомога менше залежностей від реалізації. Вона призначений для того, щоб розробники додатків могли писати один раз, працювати в будь-якому місці, тобто компільований код Java може працювати на всіх платформах, які підтримують Java без необхідності перекомпіляції. Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від основної архітектури комп'ютера. Синтаксис Java схожий на C і C ++, але в ній є менше низькорівневих можливостей, ніж в обох названих. Станом на 2019 рік, Java була однією з найпопулярніших мов програмування, яка використовується GitHub, особливо для клієнт-серверних веб-додатків, з 9 мільйонами розробників.

У створенні мови Java було п'ять основних цілей:

- Вона повинна бути простою, об'єктно-орієнтованою та звичною.
- Вона повинна бути надійною і захищеною.
- Вона повинна не залежати від архітектури та бути кросплатформенною.
- Вона повинна виконуватись з високою продуктивністю.
- Вона повинна бути інтерпретованою, потоковою та динамічною.

1.1.1 Java JVM and bytecode

Однією з цілей дизайну Java є портативність, а це означає, що програми, написані на платформі Java, повинні працювати аналогічно на будь-якій комбінації апаратних та операційних систем з адекватною

підтримкою часу виконання. Це досягається шляхом компіляції коду мови Java до проміжного представлення під назвою Java bytecode, а не безпосередньо до специфічного для архітектури машинного коду. Інструкції щодо байт-коду Java аналогічні машинному коду, але вони призначені для виконання віртуальною машиною (VM), написаною спеціально для апаратів, які будуть її використовувати. Кінцеві користувачі зазвичай використовують середовище виконання Java (JRE), встановлене на їх пристрої.

Стандартні бібліотеки надають загальний спосіб доступу до специфічних для хоста функцій, таких як графіка, нарізка ниток та мережа.

Використання універсального байт-коду робить перенесення просто. Однак накладні витрати на інтерпретацію байт-коду в машинні інструкції, що робляться інтерпретованими програмами, майже завжди працюють повільніше, ніж рідні виконувані файли.Компілятори Just-In-Time (JIT), що компілюють байт-код до машинного коду під час виконання, були введені з раннього етапу розвитку мови. Сама Java не залежить від платформи і пристосована до конкретної платформи, на якій вона працює за допомогою віртуальної машини Java, яка переводить байт-код Java на машинну мову платформи.

1.1.2 Automatic memory management

Java використовує автоматичний збирач сміття для управління пам'яттю в життєвому циклі об'єкта . Програміст визначає, коли створюються об'єкти, а Java відповідає за час використання об'єкта та відновлення пам'яті, коли об'єкти більше не використовуються. Як тільки не залишається посилання на об'єкт, зайнята пам'ять автоматично звільняється сміттєзбірником. Щось подібне до витоку пам'яті може все-таки статися, якщо код програміста містить посилання на об'єкт, який більше не потрібен, як правило, коли

об'єкти, які більше не потрібні, зберігаються в контейнерах, які все ще використовуються.

Однією з ідей, що стоять за моделлю автоматичного управління пам'яттю Java, є те, що програмісти можуть позбавитися тягаря необхідності виконувати ручне управління пам'яттю. У деяких мовах пам'ять для створення об'єктів неявно виділяється у стеку або явно розподіляється та розміщується в купі. В останньому випадку відповідальність за управління пам'яттю покладається на програміста. Якщо програма не займається розміщенням об'єкта в пам'яті, відбувається її витік. Якщо програма намагається отримати доступ до пам'яті, яка вже була розміщена, результат не визначений і непередбачуваний, і програма, швидше за все, стане нестабільною або вийде з ладу. Частково це можна усунути за допомогою розумних вказівників, але це додає лишніх витрат часу, ресурсів і збільшує складність.

Збір сміття може відбутися в будь-який час. В ідеалі це відбудеться, коли програма не буде працювати. Він гарантовано спрацьовує, якщо в купі недостатньо вільної пам’яті для виділення її під новий об’єкт; це може спричинити миттєву зупинку програми. Явне управління пам’яттю неможливо при використанні Java.

Java не підтримує синтаксис вказівників C / C ++ , де адреси об’єктів можуть бути арифметично маніпульовані (наприклад, додаючи або віднімаючи зміщення). Це дозволяє сміттєзбірнику переміщати посилання на об'єкти та забезпечує безпеку.

Як і в C ++ та деяких інших об'єктно-орієнтованих мовах, змінні примітивних типів даних Java зберігаються безпосередньо в полях (для об'єктів) або в стеці (для методів), а не в купі. Це було свідоме рішення дизайнерів Java з міркувань продуктивності. Java містить кілька видів сміттєзбірників.

```java
public class AuditorAwareImpl {

    public static void main(String[] args) {
        System.out.println("Hello World!"); // Prints the string to the console.
    }
}
```

Рис. 1.1 Hello world приклад на Java

1.1.3 Збірник проекта Maven

Maven - це інструмент для автоматизації побудови проекту, який використовується в основному для проектів Java. Maven також може використовуватися для створення та управління проектами, написаними на C#, Ruby, Scala та інших мовах. Проект Maven знаходиться під ліцензією Apache Software Foundation, проте раніше був частиною проекту Джакарта.

В Maven розглядаються два аспекти побудови програмного забезпечення: побудова проекту та під'єднання його залежностей. На відміну від попередніх інструментів, таких як Apache Ant, він використовує конвенції для процедури збирання, і потрібно виписувати лише визначення цих залежностей. XML-файл описує програмний проект, його залежність від інших зовнішніх модулів та компонентів, порядок складання, каталоги та необхідні плагіни. Він використовується з заздалегідь визначеними цілями для виконання певних чітко визначених завдань, таких як компіляція коду та його упаковка. Maven динамічно завантажує бібліотеки Java та плагіни Maven з одного або декількох сховищ, таких як Maven 2 Central Repository, і зберігає їх у локальному кеші. Цей локальний кеш завантажених артефактів також може бути оновлений новими артефактами, створеними локальними проектами. Загальнодоступні сховища також можуть бути оновлені.

Maven побудований за допомогою plugin-based архітектури, яка дозволяє використовувати будь-які програми, керовані за допомогою стандартного вводу.

Існують також альтернативні технології, такі як Gradle та sbt для збірки проектів, які не покладаються на XML, але зберігають ключові поняття Maven. За допомогою Apache Ivy був розроблений спеціалізований менеджер залежностей, який також підтримує сховища Maven.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
          https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.5.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <!--        Spring dependencies-->
    <artifactId>gas-station</artifactId>

    <version>0.0.1-SNAPSHOT</version>
    <name>gas-station</name>
    <description>Automated gas station</description>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>

        </plugins>
    </build>

    <dependencies>
        <!--        Logging-->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

    </dependencies
</project>
```

Рис. 1.2 Приклад використання

1.2. Опис та аналіз технології для серверної частини Spring Framework

Spring Framework - це прикладний фреймворк та контейнер інверсії управління для платформи Java. Основною перевагою даного фреймворка є те, що він може використовуватися для написання будь-якого додатку Java, але широко використовується для побудови веб-додатків на платформі Java EE (Enterprise Edition). Хоча фреймворк не нав'язує жодної конкретної моделі програмування, проте він став популярним у спільноті Java як доповнення або навіть заміна моделі Enterprise JavaBeans (EJB). Код Spring Framework знаходиться у відкритому доступі.

Spring Framework пропонує комплексну модель програмування та конфігурації сучасних великих Java-проектів – його можна розгорнути на будь-якій платформі.

Ключовим елементом Spring є інфраструктурна підтримка на рівні прикладних програм: Spring фокусується на вживленні в enterprise додатки, щоб команди могли зосередитись на бізнес-логіці на рівні додатків без зайвих зв’язків із конкретними середовищами розгортання.

Переваги:

- Основні технології: dependency injection, події, ресурси, i18n, validation, зв’язування даних, конверсія типів, SpEL, AOP.
- Тестування: mock objects, TestContext framework, Spring MVC Test, WebTestClient.
- Доступ до бази: transactions, DAO підтримка, JDBC, ORM, Marshalling XML.
- Spring MVC та Spring WebFlux веб-фреймворки.
- Інтеграція:відаленість, JMS, JCA, JMX, електронна почта, завдання, визначення розкладу, кеш.
- Мови використання: Kotlin, Groovy, Java та інші динамічні мови.

1.2.1 Інверсія контролю

В інженерії програмного забезпечення інверсія управління (IoC) – це принцип програмування. IoC інвертує потік управління в порівнянні з традиційним способом передачі контролю над об’єктом. В IoC, написані на замовлення частини комп’ютерної програми отримують потік управління з загальних рамок. Архітектура програмного забезпечення з цим дизайном інвертує контроль не так як у процедурному програмуванні: у традиційному програмуванні користувацький код закликає до бібліотек багаторазового використання, щоб піклуватися про загальні завдання, але з інверсією управління - це структура що викликає в користувальницький або конкретний завдання код.

Інверсія управління використовується для підвищення модульності програми та її розширення та є одним із принципів в об'єктно-орієнтованому програмуванні та інших парадигмах програмування.

Цей термін пов'язаний з принципом dependency inversion, але відрізняється від нього тим, що стосується розв'язки залежностей між шарами високого та низького рівнів шляхом спільних абстракцій. Загальна концепція також пов'язана з керованням подій в програмуванні. Вона часто реалізовується за допомогою IoC, так що користувацький код зазвичай торкається лише обробки подій, тоді як цикл подій та відправлення подій / повідомлень обробляються фреймворками або середовищами виконання.

1.2.2 Аспектно-орієнтоване програмування

В обчислювальній роботі аспект-орієнтоване програмування (AOP) - це парадигма програмування, яка має на меті підвищити модульність, дозволяючи розділити міжсекторні проблеми. Це робиться шляхом

додавання додаткової поведінки до існуючого коду (поради) без зміни самого коду, замість цього окремо вказуючи, який код модифікується за допомогою специфікації "pointcut", наприклад "журнал усіх викликів функцій, де ім'я функції починається з" set " ". Це дозволяє додавати поведінку, яка не є звичною для ділової логіки (наприклад, ведення журналу), додавати до програми без захаращування коду, що є основним для функціоналу. AOP є основою для розробки програмного забезпечення, орієнтованого на аспекти.

AOP включає методи та інструменти програмування, які підтримують модуляризацію проблем на рівні вихідного коду, тоді як "аспектно-орієнтована розробка програмного забезпечення" відноситься до цілої інженерної дисципліни.

Аспектно-орієнтоване програмування тягне за собою розбиття програмної логіки на окремі частини (так звані проблеми, згуртовані області функціональності). Майже всі парадигми програмування підтримують певний рівень групування та інкапсуляції питань у окремі, незалежні сутності, надаючи абстракції (наприклад, функції, процедури, модулі, класи, методи), які можуть бути використані для реалізації, абстрагування та складання цих питань. Деякі проблеми стосуються "перерізання" декількох абстракцій у програмі та протидії цим формам реалізації. Ці проблеми називаються наскрізними або горизонтальними.

Як правило, аспект розкиданим або заплутаним як код, що ускладнює його розуміння та підтримку. Він розповсюджується внаслідок розширення функції (наприклад, ведення журналу) по ряду непов'язаних функцій, які можуть використовувати її функцію, можливо, у повністю незв'язаних системах, різних мовах джерел тощо. Це означає, що для зміни журналу може знадобитися зміна всіх постраждалих модулів . Аспекти заплутуються не лише основною функцією систем, в яких вони виражаються, але і однин з одним.

Наприклад, розглянемо банківську програму з концептуально дуже простим методом переказу суми з одного рахунку на інший:

```
void transfer(Account fromAcc, Account toAcc, int amount) throws Exception {
  if (fromAcc.getBalance() < amount)
      throw new InsufficientFundsException();

  fromAcc.withdraw(amount);
  toAcc.deposit(amount);
}
```

Рис. 1.3 Приклад AOP

1.2.3 Spring Data

Місія Spring Data полягає у наданні звичної та послідовної, весняної моделі програмування для доступу до даних, зберігаючи при цьому особливі риси базового сховища даних.

Це полегшує використання технологій доступу до даних, реляційних та нереляційних баз даних, фреймворків для зменшення карт та хмарних служб передачі даних. Це батьківський проект, який містить багато підпроектів, характерних для певної бази даних. Проекти розробляються спільно з багатьма компаніями та розробниками, які стоять за цими захоплюючими технологіями.

Переваги:

- Потужне сховище та спеціальні абстракції для відображення об'єктів
- Динамічне виведення запитів із імен методів репозиторію
- Реалізація базових класів доменів, що забезпечують основні властивості
- Підтримка прозорого аудиту (створено, востаннє змінено)
- Можливість інтеграції спеціального коду репозиторію
- Легка Spring інтеграція через JavaConfig та користувацькі простори імен XML
- Розширена інтеграція з контролерами Spring MVC
- Експериментальна підтримка persistence

Припустимо, що Spring Data фреймворк підключений до нашого проекту. Тоді для його роботи спочатку знадобиться задекларований клас-модель сутності бази даних. На приклад:

```
@Entity
@Table(name = "users")
public class User implements IUser {
    @Id
    @GeneratedValue
    private Integer id;
    private UUID guid;
    private String name;
    private String surname;
    private String login;
    private String password;
    private String email;
    private String city;

    public User(){

    }

    // … methods omitted
}
```

Рис. 1.4 Приклад моделі бази

Після створення класів-моделей сутностей ми можемо піти двома шляхами:

1) Використати стандартну вбудовану імплементацію методів взаємодії з базою

   В даному випадку імплементація методів взаємодії буде виглядати наступним чином:

```
@Transactional(readOnly = true)
public interface UserRepository extends JpaRepository<User, Long> {

  List<Account> findByName(String name);

  List<Account> findByNameAndSurname(String name, String surname);

  List<Account> findByNameLike(String name, String likeStatement);

}
```

Рис. 1.5 Приклад репозиторію

2) Написати власні методи взаємодії із базою даних, використовуючи методи інтерфейсів JPA

В даному випадку імплементація буде виглядати так:

```
@Repository
@Transactional(readOnly = true)
class UserService implements IUserService {

  @PersistenceContext
  private EntityManager em;

  @Autowired
  private UserRepository repository;

  @Override
  @Transactional
  public User save(User user) {
    return repository.save(user);
  }

  @Override
  public List<User> findByName(String name) {

    TypedQuery query = em.createQuery(
        "select u from User u where u.name = ?1",
        User.class);
    query.setParameter(1, name);

    return query.getResultList();
  }
}
```

Рис. 1.6 Приклад сервісу

Як можна побачити з прикладів, перевагою Spring Data є значне спрощення звертання до бази даних через програмний код. Даний фреймворк дозволяє значно зменшити кількість рядків коду, неймовірно збільшує читабельність коду, а також обробляє помилки та виключення, які можуть виникати у процесі роботи із базою.

Отже, розглядаючи вищенаведені приклади, можна зробити висновок, що перший варіант чудово підходить для простих методів які організовують звичайне звернення до бази. Можна скоротити величезну кількість часу не імплементуючи велику кількість коду і не роздумуючи над SQL запитами. Але у випадку коли потрібна гнучкість звернення до бази, чи потрібно створити специфічний запит, фреймворк надає всі можливості зробити це, написавши власні, чи перевизначивши уже існуючі методи.

1.2.4 Spring Data JPA

Spring Data JPA, який є частиною більшої родини Spring Data, дозволяє легко реалізовувати бази даних на базі JPA. Цей модуль стосується розширеної підтримки шарів доступу до даних на основі JPA. Це спрощує побудову Spring-based додатків, які використовують технології доступу до даних.

Реалізація рівня доступу до даних у програмі була громіздкою протягом досить тривалого часу. Для виконання простих запитів, а також виконання сторінок і аудиту потрібно писати занадто багато однакового, схожого коду. Spring Data JPA має на меті значно покращити реалізацію шарів доступу до даних за рахунок зменшення зусиль до фактично необхідної кількості. Як розробник, ви пишете інтерфейси вашого сховища, включаючи власні методи пошуку, і Spring забезпечить реалізацію автоматично.

Переваги:

- Ускладнена підтримка для створення сховищ на основі Spring та JPA

- Підтримка предикатів Querydsl, а також JPA-запитів
- Прозорий аудит доменного класу
- Підтримка сторінки, динамічне виконання запитів, можливість інтеграції спеціального коду доступу до даних
- Перевірка анотованих запитів @Query під час завантаження
- Підтримка відображення сутності на основі XML
- Конфігурація сховища на основі JavaConfig шляхом введення @EnableJpaRepositories.

1.2.5 Spring Web

Одним із модулів фреймворку Spring є Spring Web. Даний модуль надає можливість створення класів-контроллерів для реалізації веб сервісів. На приклад,

```
@RestController
public class UserController {

    public static final Logger LOGGER = LoggerFactory.getLogger(UserController.class);

    private static final String RESPONSE_FIELD_GUID = "guid";

    private final IUserService userService;

    @Autowired
    public UserController(IUserService userService) {
        this.userService = userService;
    }

    @GetMapping("/login")
    public String userLogin(@RequestBody User user){

        final String userLogin = user.getLogin();
        final IUser userByLogin = userService.getUserByLogin(userLogin);

        LOGGER.info("User {} guild retrieved!", userLogin);

        return getGuidJson(userByLogin).toString();
    }

    @PutMapping("/{userGuid}/personal")
    public void updateUserInfo(@RequestBody @Valid User newUserInfo,
                               @PathVariable UUID userGuid) {

        userService.updateUserByGuid(newUserInfo, userGuid);

        LOGGER.info("User {} info successfully updated in the database",
        newUserInfo.getLogin());
    }
}
```

Рис. 1.7 Приклад контроллеру

Клас позначений анотацією @RestController стає готовий до використання Spring Web для обробки веб-запитів. @RequestMapping переводить шлях «/login», чи «/{userGuid}/personal», чи інших шлях для відповідних їм методів. При ініціалізації фреймворку, він буде ‘під капотом’ парсити такі URL до потрібного виду, враховуючи всі параметри шляху, чи запиту. При виклику з браузера або за допомогою curl у командному рядку метод по вказаному ендпоінту повертає те, для чого він призначений. Це тому, що @RestController поєднує в собі @Controller та @ResponseBody, дві анотації, які призводять до того, що веб-запити повертають дані, а не перегляд.

Знову таки, даний фреймворк дуже сильно спрощує роботу із створення веб-сервісів. На відміну від стандартних рішень для роботи із REST-контроллерами, даний фреймоворк виконує майже всю рутинну роботу, а саме:

- парс URL до потрібного вигляду;
- парс сутностей, отриманих через веб запит;
- віддача правильної відповіді користувачу.

1.3. Опис та аналіз технології для клієнтської частини Java FX

JavaFX - це програмна платформа для створення та доставки десктопних додатків, а також багатих Інтернет-додатків (RIA), які можуть працювати на широкому спектрі пристроїв. JavaFX призначений замінити Swing як стандартну бібліотеку графічного інтерфейсу для Java SE, але обидві будуть включені в осяжному майбутньому. JavaFX має підтримку настільних комп'ютерів та веб-браузерів у Microsoft Windows, Linux та macOS.

До версії 2.0 JavaFX розробники використовували статично набрану декларативну мову під назвою JavaFX Script для створення JavaFX-додатків.

Оскільки JavaFX Script був зібраний в байт-код Java, програмісти також могли використовувати замість цього Java-код. Програми JavaFX можуть працювати на будь-якому робочому столі, на якому можна запустити Java SE, або на будь-якому мобільному телефоні, на якому можна запустити Java ME.

JavaFX 2.0 і пізніші версії реалізовані як "рідна" бібліотека Java, а програми, що використовують JavaFX, записуються в "рідний" код Java. JavaFX Script був знятий Oracle, але розвиток продовжується в проекті Visage. JavaFX 2.x не підтримує операційну систему Solaris або мобільні телефони; проте Oracle планує інтегрувати JavaFX до Java SE Embedded 8, а Java FX для процесорів ARM знаходиться у фазі попереднього перегляду розробника.

Десктоп JavaFX підтримує операційні системи Windows Vista, Windows 7, Windows 8, Windows 10, macOS та Linux. Починаючи з JavaFX 1.2, Oracle випустив бета-версії для OpenSolaris. На мобільних пристроях JavaFX Mobile 1.x здатний працювати в декількох мобільних операційних системах, включаючи Symbian OS, Windows Mobile та власні операційні системи в режимі реального часу.

Платформа JavaFX 2.x включає такі компоненти:

- JavaFX SDK: інструменти виконання. Графіка, веб-сервіси медіа та багаті текстові бібліотеки. Java FX 1.x також включав компілятор JavaFX, який тепер застарілий, оскільки код користувача JavaFX написаний на Java.
- NetBeans IDE для JavaFX: NetBeans з палітрою drag-and-drop для додавання об'єктів з перетвореннями, ефектами та анімацією плюс набір зразків та кращих практик. Для підтримки JavaFX 2 вам потрібно принаймні NetBeans 7.1.1. Для користувачів Eclipse існує плагін, що підтримується спільнотою, розміщений на e (fx) clipse.

- Конструктор сцен JavaFX: це було представлено для Java FX 2.1 та пізніших версій. Користувальницький інтерфейс (UI) створюється шляхом перетягування елементів управління з палітри. Ця інформація зберігається як файл FXML, спеціальний формат XML.
- Інструменти та плагіни для творчих інструментів (він же). Плагіни для Adobe Photoshop та Adobe Illustrator, які можуть експортувати графічні активи до коду сценарію JavaFX, інструменти для перетворення графіки SVG у код сценарію JavaFX та попередні перегляд активів, перетворених у JavaFX з інших інструментів (наразі немає підтримується у версіях JavaFX 2.x).

1.4. Опис та аналіз системи управління базами даних PostgreSQL

PostgreSQL, також відомий як Postgres, це безкоштовна та відкрита система управління реляційними базами даних (RDBMS), що підкреслює розширюваність та відповідність SQL. Спочатку вона мала назву POSTGRES, посилаючись на своє походження як спадкоємця бази даних Інгрес, розробленої в Каліфорнійському університеті, Берклі. У 1996 році проект було перейменовано на PostgreSQL, щоб відобразити його підтримку SQL. Після огляду в 2007 році команда розробників вирішила зберегти ім’я PostgreSQL.

PostgreSQL пропонує транзакції з властивостями Atomicity, Consistentcy, Isolation, Durability (ACID), автоматично оновлюються представлення даних, матеріалізовані подання, тригери, зовнішні ключі та збережені процедури. Він призначений для роботи з різними навантаженнями, від окремих машин до сховищ даних або веб-сервісів з багатьма одночасними користувачами. Це база даних за замовчуванням для сервера macOS, а також доступна для Linux, FreeBSD, OpenBSD та Windows.

Схема.

У PostgreSQL схема містить всі об'єкти, крім ролей та табличних просторів. Схеми ефективно діють як простори імен, дозволяючи об’єктам одного імені співіснувати в одній базі даних. За замовчуванням у новостворених баз даних є схема, що називається загальнодоступною, але будь-які подальші схеми можуть бути додані, і загальнодоступна схема не є обов'язковою.

Параметр search_path визначає порядок, у якому PostgreSQL перевіряє схеми на наявність некваліфікованих об'єктів (тих, що не мають попередньо встановленої схеми). За замовчуванням вона встановлена на $ user, public ($ user посилається на користувача, який зараз підключений). Цей за замовчуванням можна встановити на базі даних або на рівні ролей, але оскільки це параметр сеансу, його можна вільно змінювати (навіть кілька разів) під час сеансу клієнта, впливаючи лише на цей сеанс.

Неіснуючі схеми, перелічені в search_path, безшумно пропускаються під час пошуку об’єктів.

Нові об’єкти створюються в тій чи іншій дійсній схемі (тій, яка існує в даний час) спочатку з'являється в search_path.Schema - це контур бази даних.

Типи даних.

Підтримується широкий вибір власних типів даних, включаючи:

- Boolean
- Числа з плаваючою точкою
- Character (text, varchar, char)
- Binary
- Date/time (timestamp/time with/without time zone, date, interval)
- Money
- Enum
- Bit strings
- Text search type
- Composite

- HStore, розширення для збереження пари ключ-значення в PostgreSQL
- Arrays (змінної довжини і можуть бути будь-якого типу даних, включаючи текстові та складені типи) до 1 GB загального розміру
- Геометричні примітиви
- IPv4 та IPv6 адреси
- Блоки безкласового маршрутизації між доменами (CIDR) та MAC-адреси
- XML, що підтримує XPath запити
- Universally unique identifier (UUID)
- JavaScript Object Notation (JSON), та більш швидкий двійковий JSONB

PostgreSQL має багато розширених функцій, які пропонують інші системи управління базами даних, такі як:

- Типи, визначені користувачем
- Таблиця спадкування
- Ускладнений замикаючий механізм
- Foreign key вказівникова цілісність
- Views, rules, subquery
- Вкладені транзакції (точки збереження)
- Багатоверсійний контроль сумісності (MVCC)
- Асинхронна реплікація

Останні версії PostgreSQL підтримують такі функції:

- Рідна версія Microsoft Windows Server
- Простір імен таблиць
- Point-in-time відновлення

1.4.1 PostgreSQL драйвер

JDBC - це API для мови програмування Java, який визначає, як клієнт може отримати доступ до бази даних. Він надає методи запиту та оновлення даних у базі даних. JDBC орієнтований на реляційні бази даних. З технічної точки зору, API є набором класів у пакеті java.sql. Для використання JDBC з певною базою даних нам потрібен драйвер JDBC для цієї бази даних.

JDBC розшифровується як Java Database Connectivity, який є стандартним Java API для незалежності баз даних між мовою програмування Java та широким спектром баз даних.

Бібліотека JDBC включає API для кожної із задач, зазначених нижче, які зазвичай пов'язані з використанням баз даних.

- Встановлення підключення до бази даних.
- Створення операторів SQL або MySQL.
- Виконання запитів SQL або MySQL в базі даних.
- Перегляд та зміна отриманих записів.

Щоб підключити базу даних до проекту потрібно підключити JDBC драйвер для встановленя зв'язку системи із базою даних. Так як у даному проєкті використовується Maven build tool, то це робиться просто додавши таку залежність у файл pom.xml:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.0</version>
</dependency>
```

Рис. 1.8 Приклад підключення драйверу в Maven

Після цього потрібно створити конфігурацію, зчитуючи яку, система буде знати з якою саме базою працювати. Реалізувати таке можна трьома способами:

1) Створивши конфігураційний файл <fileName>.xml. На приклад:

```
<bean id="myDataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.postgresql.Driver" />
        <property name="url" value="jdbc:postgresql://localhost:5432/dbname" />
        <property name="username" value="postgres" />
        <property name="password" value="" />

        <property name="connectionProperties">
            <props>
                <prop key="socketTimeout">10</prop>
            </props>
        </property>

</bean>
```

Рис. 1.9 Приклад конфігураційного XML файлу.

2) Створивши конфігураційний файл <fileName>.properties, зчитавши його в коді програми та передавши цей файл відповідному фабричному методу. На приклад:

```
db.driver.class=org.postgresql.Driver
db.conn.url=jdbc:postgresql://localhost:5432/<db-name>
db.username=some-user
db.password=user-password
```

Рис. 1.10 Приклад конфігураційного .property файлу.

3) Задавши конфігурацію безпосередньо у коді програми. На приклад:

```
public DataSource dataSource() {
        final DriverManagerDataSource dataSource = new DriverManagerDataSource();

        dataSource.setDriverClassName("org.postgresql.Driver");
        dataSource.setUrl("jdbc:postgresql://localhost:5432/dbname");
        dataSource.setUsername("username");
        dataSource.setPassword("userPassword");

        return dataSource;
}
```

Рис. 1.11 Приклад конфігурації в коді.

Проте використання JDBC являється дуже низькорівневим, так як приходиться виконувати всі елементарні дії для маніпуляції базою даних вручну, таких як:

- Відкривання зв’язку з базою
- Створення рядку запиту в базу даних

- Отримання даних у вигляді ResultSet, який містить всі дані таблиць у відповідності до типів колонок
- Маппінг даних ResultSet-у до класів-моделів сутностей
- Відловлення і опрацювання помилок і вийнятків
- Закривання зв’язку з базою

Тому в проєкті дана технологія використовується неявно, всі вищеперечислені дії виконують фреймворки, які описані в звіті вище.

1.4.2 Hibernate PostgreSQL, JPA

Hibernate - найпопулярніша реалізація специфікації JPA (Java Persistense API), призначена для вирішення завдань об'єктно-реляційного відображення (ORM). Поширюється вільно на умовах GNU Lesser General Public License.

Метою Hibernate є звільнення розробника від значного обсягу порівняно низкоуровневого програмування при роботі в об'єктно-орієнтованих засобах в реляційної базі даних. Розробник може використовувати Hibernate як в процесі проектування системи класів і таблиць «з нуля», так і для роботи з уже існуючою базою даних.

Бібліотека не тільки вирішує завдання зв'язку класів Java з таблицями бази даних (і типів даних Java з типами даних SQL), але і також надає засоби для автоматичної генерації і оновлення набору таблиць, побудови запитів і обробки отриманих даних і може значно зменшити час розробки, яке зазвичай витрачається на ручне написання SQL- і JDBC-коду. Hibernate автоматизує генерацію SQL-запитів і звільняє розробника від ручної обробки результуючого набору даних і перетворення об'єктів, максимально полегшуючи перенесення (портирование) додатки на будь-які бази даних SQL.

Hibernate забезпечує прозору підтримку збереження даних (persistence) для «POJO» (тобто для стандартних Java-об'єктів); єдине сувора вимога для зберігається класу - наявність конструктора за замовчуванням (без

параметрів). Для коректної поведінки в деяких додатках потрібно також приділити увагу методам equals () і hashCode ().

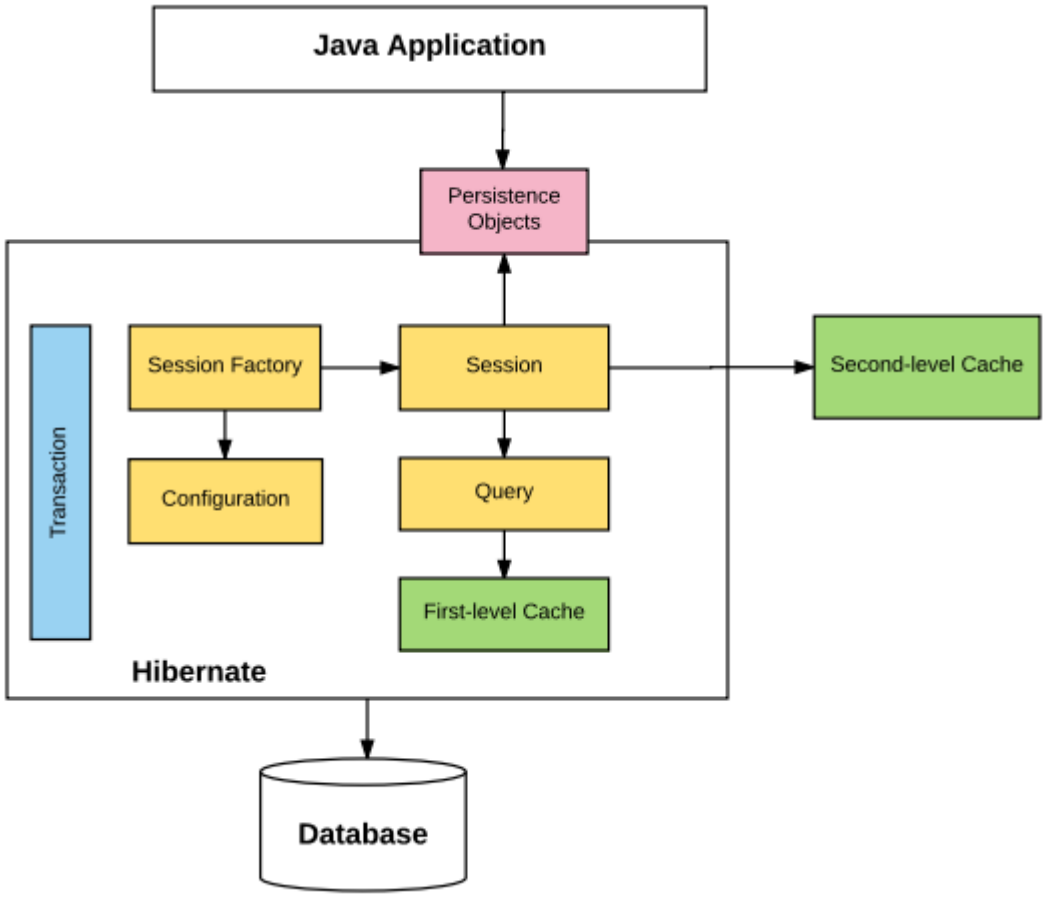


Рис. 1.12 Архітектура Hibernate

Найважливіші особливості Hibernate framework:

- Об'єктний / реляційний маппінг

Hibernate, як ORM, дозволяє відображати доменний об’єкт Java в таблицях баз даних і навпаки. Як результат, бізнес-логіка дозволяє отримувати доступ до об'єктів бази даних та керувати ними через об’єкти Java. Це допомагає прискорити загальний процес розробки, піклуючись про такі аспекти, як управління транзакціями, автоматичне генерування первинного ключа, керування підключеннями до бази даних та пов'язаними реалізаціями тощо.

- Провайдер JPA

Hibernate підтримує специфікацію API Java Persistence API (JPA) . JPA - це набір специфікацій для доступу, зберігання та управління даними між об'єктами Java та реляційними базами даних.

- Ідіоматичний persistence

Будь-який клас, який дотримується об'єктно-орієнтованих принципів, таких як успадкування, поліморфізм тощо, може використовуватися як стійкий клас.

- Висока продуктивність та масштабованість

Hibernate підтримує такі методи, як різні стратегії отримання, ледача ініціалізація, оптимістичне блокування тощо, для досягнення високої продуктивності, і це добре масштабує в будь-яких умовах.

- Легкий в обслуговуванні

Hibernate легше підтримувати, оскільки для нього не потрібно спеціальних таблиць баз даних або полів. Він генерує SQL під час ініціалізації системи. Це набагато швидше і простіше в обслуговуванні порівняно з JDBC.

Висновок.

По замовчуванню, Hibernate використовується як основна реалізація JPA у фреймворка Spring Data, але за бажанням можна обрати іншу ORM, тому для того щоб підключити Hibernate до проекту достатньо всього лише підключити залежність Spring Data в файлі pom.xml, який використовується Maven build tool:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>2.2.6.RELEASE</version>
</dependency>
```

Рис. 1.13 Приклад підключення Spring Data в Maven

Даний фреймворк значно спрощує роботу, так як за допомогою нього та його можливостей можна зекономити багато часу на досягнення бізнес цілей,

не вірушуючи і не виконуючи всю рутинну роботу, яку потрібно робити при використанні технології JDBC.

1.5. Опис та аналіз серверу Google cloud API

Платформа Google Cloud (GCP), запропонована Google, - це набір послуг хмарних засобів, які працюють на тій самій інфраструктурі, яку Google використовує внутрішньо для своїх комерційних продуктів, таких як Пошук Google, Gmail та YouTube. Поряд із набором інструментів управління, він пропонує низку модульних хмарних служб, включаючи обчислення, зберігання даних, аналітику даних та машинне навчання. Для реєстрації потрібні дані кредитної картки або банківського рахунку.

Платформа Google Cloud надає інфраструктуру як сервіс, платформу як службу та безсерверні обчислювальні середовища.

У квітні 2008 року компанія Google оголосила App Engine, платформу для розробки та розміщення веб-додатків в центрах обробки даних, керованих Google, що було першим сервісом хмарних обчислень від компанії. Початок сервіс став доступний у листопаді 2011 року. З моменту появи App Engine компанія Google додала на платформу кілька хмарних сервісів.

Платформа Google Cloud є частиною Google Cloud, яка включає загальнодоступну хмарну інфраструктуру платформи Google Cloud, а також G Suite, корпоративні версії ОС Android та Chrome і прикладні інтерфейси програмування (API) для машинного навчання та картування підприємств. послуги.

1.5.1. Google Cloud Storage

Google Cloud Storage - це RESTful веб-сервіс для зберігання файлів в Інтернеті для зберігання та доступу до даних на інфраструктурі платформи

Google Cloud. Послуга поєднує в собі ефективність та масштабованість хмари Google із розширеними можливостями безпеки та обміну. Це інфраструктура як послуга (IaaS), порівнянна зі службою зберігання в Інтернеті Amazon S3. На відміну від Диска Google та відповідно до різних специфікацій сервісу, Google Cloud Storage видається більш підходящим для підприємств.

Ключові особливості.

- Управління витратами на зберігання та продуктивністю за допомогою OLM

Налаштуйте свої дані за допомогою управління об'єктним життєвим циклом (OLM) для автоматичного переходу до класів зберігання за нижчими витратами, коли вони відповідають критеріям, визначеним вами, наприклад, коли досягнуть певного віку або коли ви зберегли нову версію даних.

- Типи місцеположення для різних потреб у надмірності та ефективності

Хмарне зберігання має постійно зростаючий перелік локацій у всьому світі, де ви можете зберігати свої дані за допомогою декількох варіантів автоматичного резервування. Незалежно від того, чи оптимізуєте ви час реакції за секунду або створюєте надійний план відновлення після аварій, налаштуйте, де і як ви зберігаєте свої дані.

1.6. Обгрунтування вибору технологій для розробки інформаційної системи

1.6.1. Spring фреймворк

1. Продуктивність

Spring Boot змінює підхід до завдань із програмування на Java, радикально впорядковуючи свій досвід. Spring Boot поєднує в собі такі потреби, як контекст програми та автоматично-налаштовуваний вбудований веб-сервер, щоб зробити розробку мікросервісу чіткішою та простішою. Щоб

працювати ще ефективніше, можна поєднувати Spring Boot з багатим набором підтримуючих бібліотек, серверів, шаблонів та шаблонів Spring Cloud, щоб безпечно розгорнути цілі архітектури на основі мікросервісів у хмару за короткий час.

2. Безпека

Spring має перевірене досвідом вирішення з питань безпеки та робить це швидко та відповідально. Контриб'ютори Spring працюють з професіоналами із сфери безпеки, щоб виправити і протестувати будь-які повідомлення про вразливості. Сторонні залежності також ретельно контролюються, і регулярно оновлюються, щоб зберегти ваші дані та програми максимально безпечними. Крім того, Spring Security полегшує вам інтеграцію зі стандартними галузевими схемами безпеки та забезпечує надійні рішення, які захищені за замовчуванням.

3. Гнучкість

Гнучкий і всебічний набір розширень і сторонніх бібліотек Spring дозволяє розробникам створювати практично будь-яку програму, яку можна уявити. По суті, функції Inversion of Control (IoC) та інжекції залежностей (DI) Spring Framework служать основою для широкого набору можливостей та функціональності. Незалежно від того, чи будуєте ви безпечні, реактивні, хмарні мікросервіси для Інтернету чи складні потокові потоки даних для підприємства, Spring має інструменти, які допоможуть.

4. Підтримка

Спільнота Spring є величезною, глобальною, різноманітною і охоплює людей різного віку та можливостей, від початківців до досвідчених професіоналів. Незалежно від того, куди ви їдете, ви можете знайти підтримку та ресурси, які вам потрібні, щоб перейти на наступний рівень: швидкі старти, путівники та навчальні посібники, відео, зустрічі, підтримка або навіть формальне навчання та сертифікація.

1.6.2. Spring Data фреймворк

1. No-code репозиторії

Патерн репозиторію - одна з найпопулярніших моделей, які стосуються збереження даних в базі даних. Він приховує конкретні деталі реалізації сховища даних та дозволяє реалізувати бізнес-код на більш високому рівні абстракції.

Реалізація цього патерну не надто складна, але написання стандартних операцій CRUD для кожного об'єкта створює багато повторюваних кодів. Spring Data JPA надає вам набір інтерфейсів сховища, який потрібно розширити лише для визначення конкретного сховища для однієї з ваших організацій.

2. Скорочення кількості одноманітності коду

Щоб зробити це ще простіше, Spring Data JPA забезпечує реалізацію за замовчуванням для кожного методу, визначеного одним із його інтерфейсів сховища. Це означає, що вам більше не потрібно здійснювати основні операції читання або запису. І навіть тому, що всі ці операції не потребують великого коду, а їхнє впровадження робить життя трохи простішим і зменшує ризик появи дурних помилок.

3. Згенеровані запити

Ще одна зручна особливість Spring Data JPA - генерування запитів до бази даних на основі імен методів. Поки ваш запит не надто складний, вам просто потрібно визначити метод у інтерфейсі вашого сховища з іменем, яке починається з пошуку… Ву. Потім Spring аналізує назву методу та створює запит до нього.

Ось простий приклад запиту, який завантажує об’єкт Книги із заданим заголовком. Внутрішньо Spring створює JPQL-запит на основі імені методу, встановлює надані параметри методу як значення параметрів прив'язки, виконує запит і повертає результат.

```
public interface BookRepository extends CrudRepository<Book, Long> {

    Book findByTitle(String title);
}
```

Рис. 1.14 Приклад репозиторію

1.6.3. JavaFX

Досвід свідчить про те, що реалізація інтерфейсної частини застосунку користувача за допомогою рідної технології є коректним підходом і що JavaFX добре підходить.

- JavaFX доступний у провідних операційних системах настільних ПК (Windows, Linux та Mac OS X).
- Незважаючи на те, що він пережив деякі болючі зміни, його еволюція підтверджує рівень відданості продавця.
- Як наступник Swing, його використовують все більше число розробників Java. Незалежно від свого майбутнього, це виграє від сильної спільноти розробників.
- Порівняно з Swing, він забезпечує чітку та чисту архітектуру та має багато вдосконалень: стилізацію, управління подіями, переходи, графік сцен - щоб назвати лише кілька.
- Він забезпечує можливість розробки сучасних інтерфейсів користувача з анімацією, мультитач тощо.
- Він заснований на чіткій та чистій мові: Java.
- Він надає всі професійні інструменти Java, необхідні для налагодження, аналізу, профілю та реєстрації клієнтської програми.
- Це дозволяє просту установку, подібну додатку, на стороні клієнта, без будь-яких передумов.

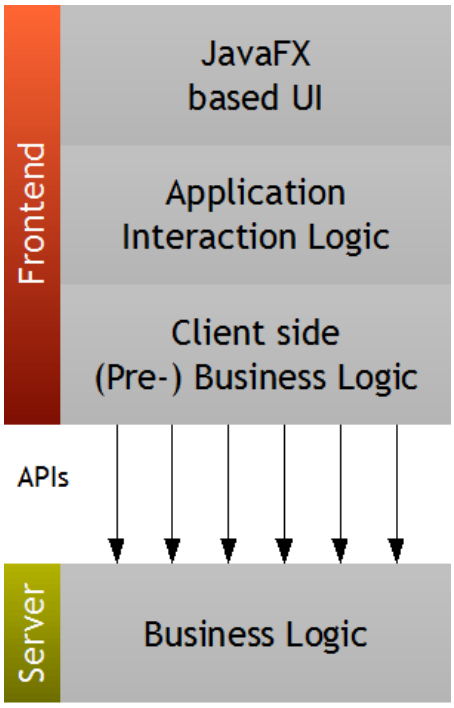


Рис. 1.15 Обширна архітектура клієнта, написаного на JavaFX

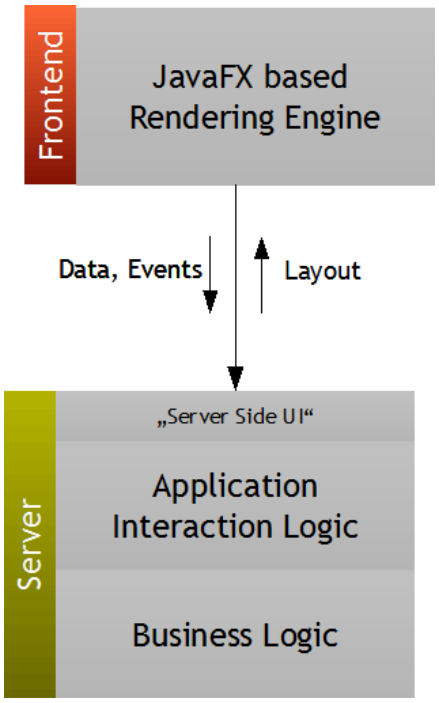


Рис. 1.16 Тонка архітектура клієнта, написаного на JavaFX

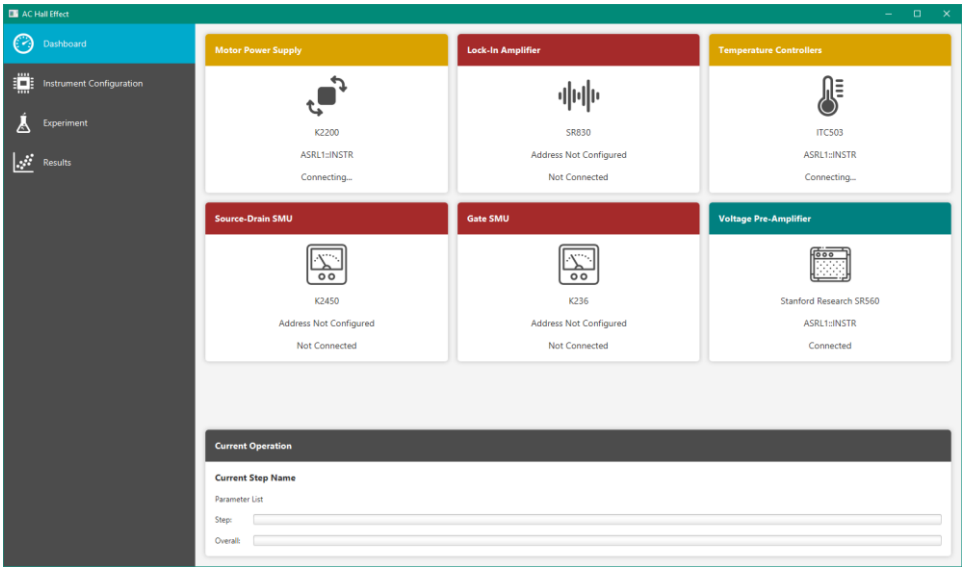


Рис. 1.17 Приклад клієнтського застосунку, написаного за допомогою JavaFX

2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

2.1. Аналіз існуючих рішень документних менеджерів

Файловий менеджер або браузер файлів - це комп'ютерна програма, яка забезпечує інтерфейс користувача для управління файлами та папками. Найбільш поширені операції з файлами або групами файлів включають створення, відкриття (наприклад, перегляд, відтворення, редагування чи друк), перейменування, переміщення або копіювання, видалення та пошук файлів, а також зміна атрибутів файлів, властивостей та дозволів на файли. Папки та файли можуть відображатися в ієрархічному дереві на основі їх структури каталогів. Деякі файлові менеджери містять функції, навіяні веб-браузерами, включаючи навігаційні кнопки вперед та назад.

Деякі файлові менеджери забезпечують підключення до мережі за допомогою протоколів, таких як FTP, HTTP, NFS, SMB або WebDAV. Це досягається за допомогою дозволу користувачу переглядати файловий сервер (підключення та доступ до файлової системи сервера, як локальна файлова система) або надання власних повних клієнтських реалізацій для протоколів файлового сервера.

2.1.1. Total Commander

Total Commander (раніше Windows Commander) - це файловий менеджер для Windows, Windows Phone, Windows Mobile / Windows CE та Android, розроблений швейцарцем Крістіаном Гіслером. Спочатку був написаний за допомогою Delphi, проте останні версії 64-бітних версій Windows були виконані з допомогою Lazarus. У ньому є вбудований FTP-клієнт, інтерфейс з вкладками, порівняння файлів, навігація до архіву файлів та інструмент для перейменування з регулярною підтримкою виразів. Він здебільшого сумісний з Linux за допомогою Wine.

Утиліта підтримує можливість розширення за допомогою плагінів і може прив’язувати зовнішні програми для перегляду чи редагування файлів. Багато плагінів є у вільному доступі, наприклад, різні формати пакування або переглядач файлів для спеціальних форматів файлів. Багато функцій, недоступних за замовчуванням, підтримуються і можуть бути призначені піктограмам.

З 1993 по 2002 рік Total Commander називався Windows Commander. Назва була змінена в 2002 році після того, як Microsoft вказала, що слово "Windows" було їх торговою маркою.

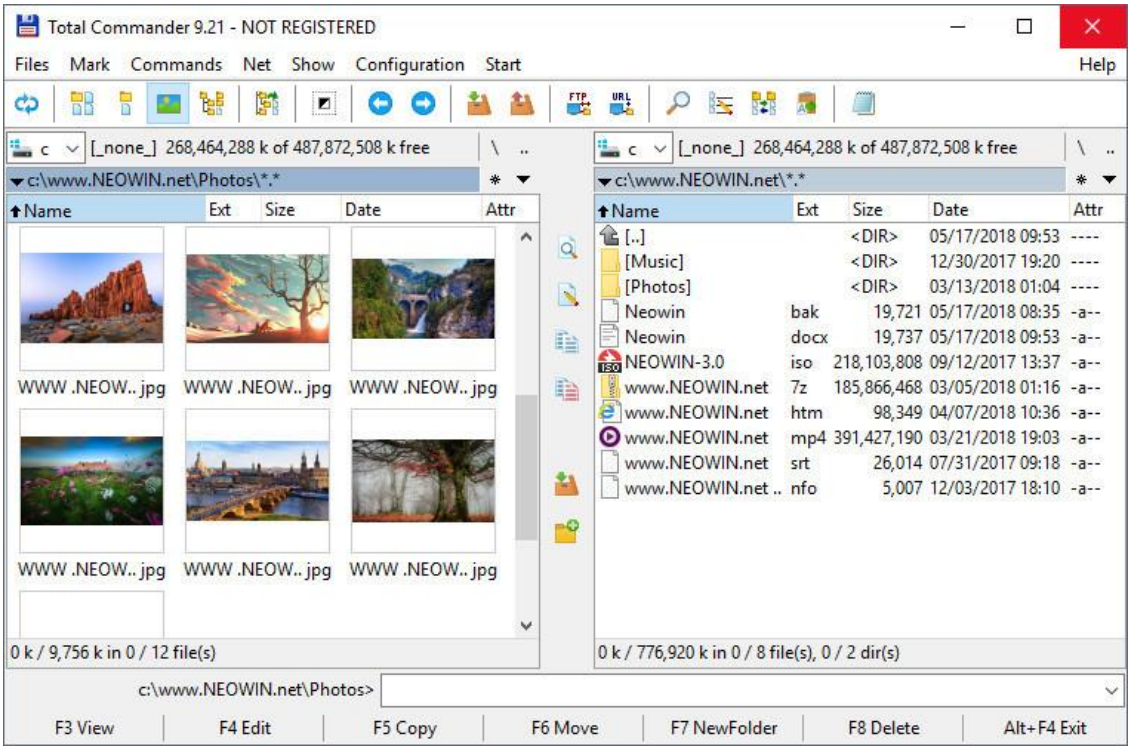


Рис. 2.1 Інтерфейс Total Commander-а

На сьогоднішній день, Total Commander є одним із найбільш використовуваних та найбільш потужних інструментів для роботи з файлами. Так як він не має плагінів чи модулів які виконують порівнювання документів, але дозволяє підключати до себе сторонні програми, то власноруч розроблений модуль з розширеного пошуку документів може бути підключеним за допомогою допоміжних засобів до Total Commander-а, розширивши таким чином його функціонал та зробивши його ще потужнішим. Таким чином можна зробити внесок у продукт світового використання. В

такому випадку користуватися даним модулем будуть тисячі, а то і міліони користувачів, а значить бізнес значення даного модуля буде не мізерне.

2.1.2. Провідник Windows

Файловий провідник , раніше відомий як Windows Explorer, - це програма для управління файлами, яка входить до випусків операційної системи Microsoft Windows від Windows 95 і далі. Він надає графічний інтерфейс користувача для доступу до файлових систем. Це також компонент операційної системи, який представляє на екрані багато елементів інтерфейсу користувача, таких як панель завдань та робочий стіл. Управління комп'ютером можливе без запуску Windows Explorer (наприклад, команда File | Run у диспетчері завдань у похідних NT версіях Windows буде функціонувати без нього, як і команди, введені у вікні командного рядка).

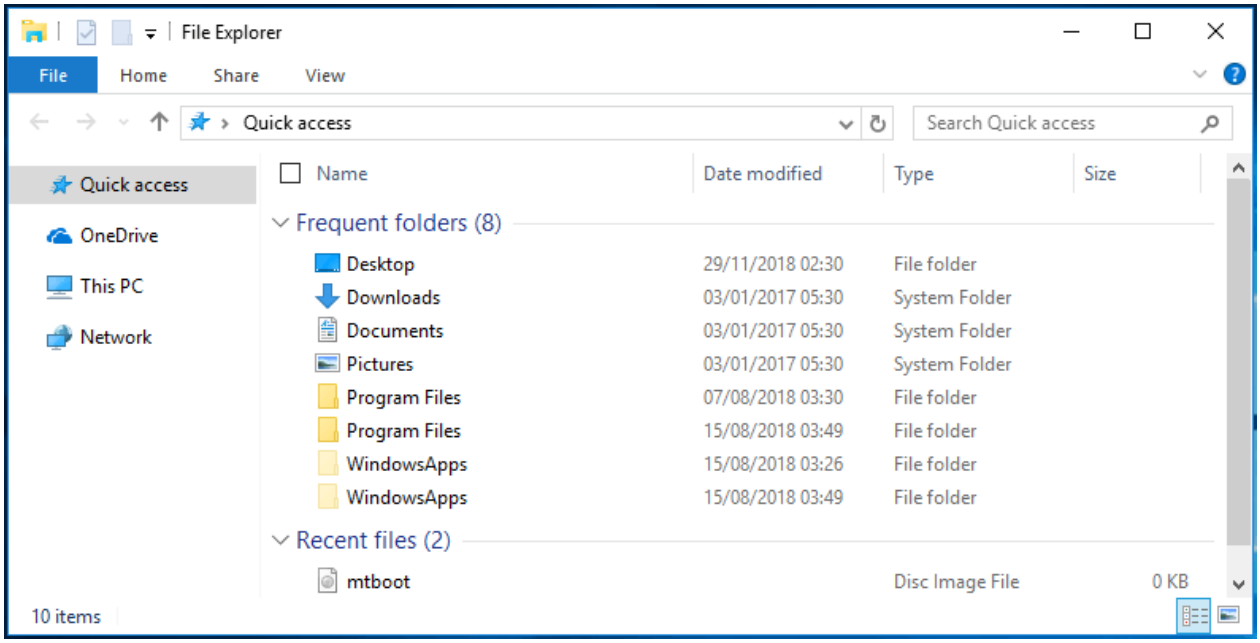


Рис. 2.2 Приклад файлового провідника Windows

Так як даний файловий менеджер є елементом операційної системи Microsoft Windows та повністю належить корпорації Microsoft і захищений їхньою ліцензією, то змінювати його чи доповнювати не можливо, так як це порушить політику використання програмного продукту. Тому єдиним

рішенням в такому випадку буде створення подібного менеджера, але з дещо іншим функціоналом. В такому випадку створений модуль буде цілим незалежним програмним продуктом, який можна буде окремо встановити на персональний комп'ютер.

2.2. Аналіз існуючих рішень програмних засобів з текстового пошуку

Повнотекстовий пошук позначає методи пошуку одного документа, що зберігається на комп'ютері, або колекції у повнотекстовій базі даних. Повнотекстовий пошук відрізняється від пошуку на основі метаданих або на частинах оригінальних текстів, представлених у базах даних (таких як заголовки, реферати, вибрані розділи чи бібліографічні посилання).

Під час повнотекстового пошуку пошукова система вивчає всі слова у кожному збереженому документі, намагаючись відповідати критеріям пошуку (наприклад, текст, визначений користувачем). Методи пошуку повнотекстового пошуку стали поширеними в онлайн-бібліографічних базах даних у 1990-х рр. Багато веб-сайтів та прикладних програм (наприклад, програмне забезпечення для обробки тексту) надають можливості пошуку в повному тексті. У деяких веб-пошукових системах, таких як AltaVista, використовуються методи пошуку в повнотекстовому пошуку, в той час як інші індексують лише частину веб-сторінок, розглянутих їх системами індексації.

2.2.1. Desktop Search, Copernic inc.

Інструменти десктопного пошуку шукають в файлах власного комп'ютеру користувача на відміну від пошуку в Інтернеті. Ці інструменти призначені для пошуку інформації на ПК користувача, включаючи історію веб-браузера, архіви електронної пошти, текстові документи, звукові файли, зображення та відео. Більшість програм пошуку настільних ПК є автономними

програмами. Продукти пошуку настільних комп'ютерів – це альтернатива програмному забезпеченню програмного забезпечення, включеного в операційну систему, що допомагає користувачам просіювати файли настільних ПК, електронні листи, вкладення тощо.

Інструмент Desktop Search дозволяє користувачеві шукати текст у власних документах за допомогою ключових слів. Він призначений для пошуку відповідної інформації на ПК користувача, включаючи файли Office, електронні листи та вкладення Outlook, текстові документи та мультимедійні файли. Як окремий додаток, він створює карту ключових слів (індекс) для розблокування неструктурованих даних. Замість того, щоб інвестувати у велике та дороге рішення для пошуку на підприємстві, пошук на робочому столі – це швидка та проста установка для підвищення вашої продуктивності, при цьому надзвичайно безпечна.

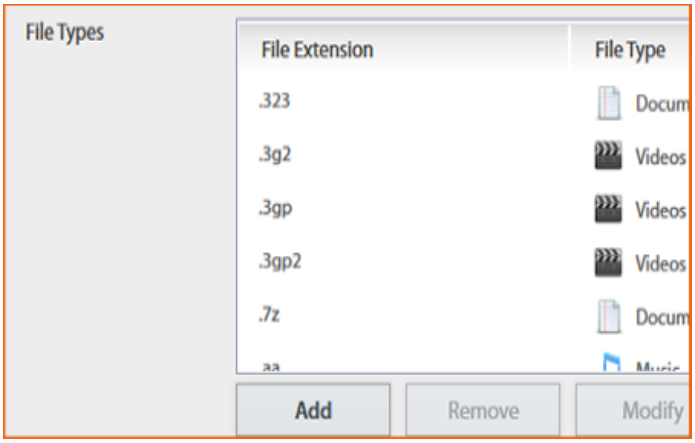


Рис. 2.3 Приклад типів файлів

Понад 150 типів файлів.

Безкоштовне видання пропонує індексувати та шукати понад 119 типів файлів. Під час переходу до повної версії ви отримуєте найбільш використовувані типи файлів, такі як Microsoft Office, Outlook, PDF, Cloud Services та багато іншого.

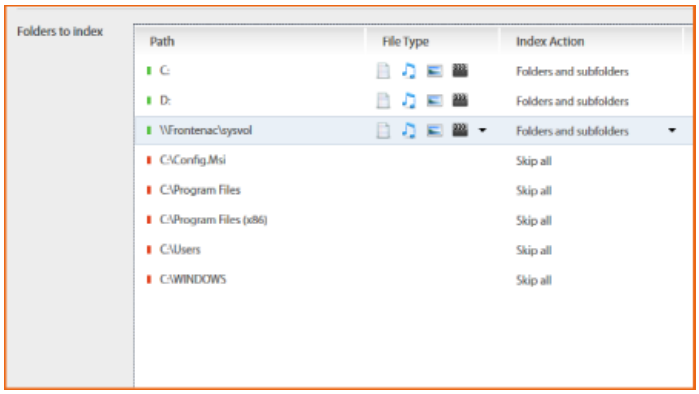


Рис. 2.4 Приклад пошуку всюди

Пошук «всюди».

Потужність Copernic Desktop Search дозволяє виконувати пошук «всюди», на кожному диску системи чи мережі. Пошук виконується швидко, так як Copernic будує свій індекс при першому використанні, а потім просто оновлює його у фоновому режимі.

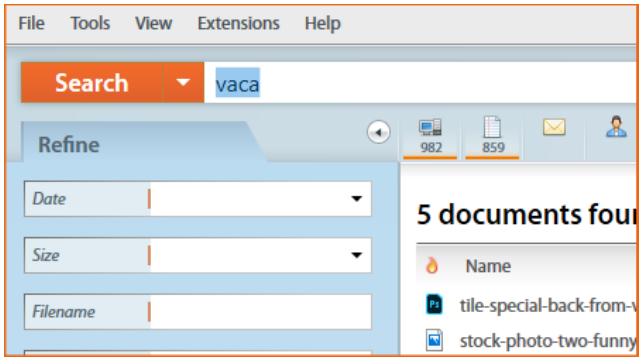


Рис. 2.5 Приклад пошуку під час введення

Пошук під час введення.

Можливість отримування результатів, як тільки вводиться текст на панелі пошуку. Результати пошуку з’являються на панелі попереднього перегляду.

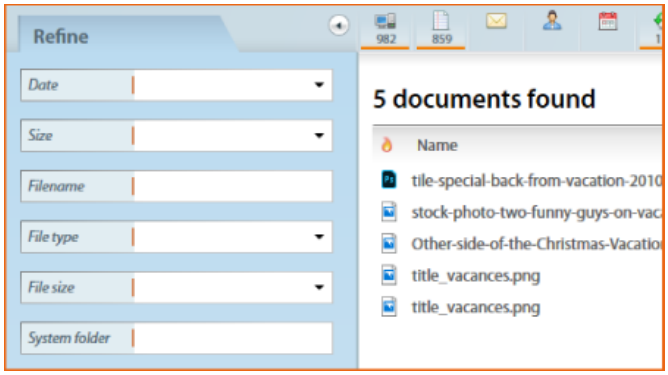


Рис. 2.6 Приклад введення полів пошукових параметрів

Введення полів параметрів пошуку.

Панель вдосконалення пошуку зліва від інтерфейсу дозволяє користувачеві заглиблюватися у файли комп'ютера та отримувати саме те, що шукає, дозволяє налаштувати пошукову роботу, додаючи або видаляючи критерії, пов'язані з кожною категорією пошуку.

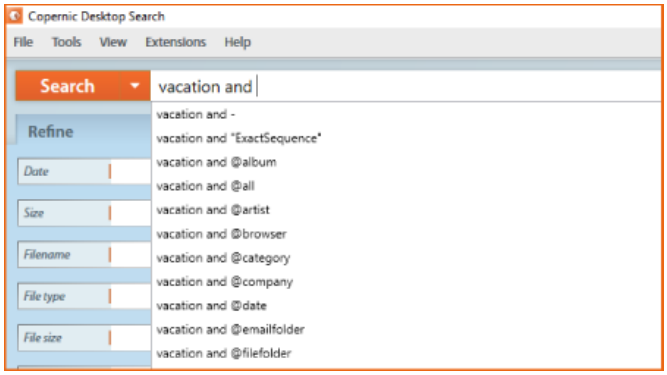


Рис. 2.7 Приклад використання операторів пошуку

Оператори пошуку.

Copernic Desktop Search повністю підтримує оператори Bolean, такі як AND, OR, NOT, BLED та багато іншого. Також на панелі пошуку використовується архітектура пошуку префікса. Тому він знаходить слова, що починаються із заданого префікса (ключового слова).

Отже, Copernic Desktop Search це серйозна програма, написана командою досвідчених розробників, яка надає досить широкий функціонал для маніпуляції документами. Власно створений програмний модуль не зможе надавати наскільки широкий спектр можливостей, так як всі вони потребують надто багато часу для реалізації. Проте даний програмний продукт не надає змогу отримати таблицю, яка показує на скільки файли схожі один з одним. А це і є саме той функціонал, який реалізований в темі дипломного проєкту.

2.2. Висновок

Отже, як можна помітити, існує велика кількість різноманітних програмних продуктів для маніпуляції документами у файловій системі. Вище

перечислені найпотужніші та найпопулярніші рішення. Безумовно, це хороші продукти, доведені із не одним роком їх реалізації до ідеалу.

Проте, темою дипломного проєкту є написання програмного продукту, який підходить конкретно для людей, чиї професії змушують працювати з великою кількістю текстових документів. Вищеперечислені програмні продукти не надають функціонала, який може бути дуже корисним у роботі людей такої категорії професій. Саме тому виникла необхідність створити модуль який буде надавати зручний інтерфейс користувачу для маніпуляції документами та перегляду їх у вигляді таблиць у відповідності до інших документів.

Власний програмний модуль повинний бути універсальним. Тобто, це має бути сервіс, який не прив’язаний до конкретного інтерфейсу, а який надає універсальний інтерфейс за допомогою REST - архітектури. В такому випадку цей модуль можна буде використовувати як для написання десктопних застосунків, браузерних застосунків, так і для створення плагінів для таких програм як Total Commander.

REST-архітектура – це архітектура, яка створена для того щоб надавати узагальнений інтерфейс для користувача. Основна її перевага – надання можливості використання незалежно від платформи, чи мови програмування, чи операційної системи.

3. ОПИС РОЗРОБЛЕНИХ ПІДПРОГРАМ ТА АЛГОРИТМІВ

3.1. Серверна частина

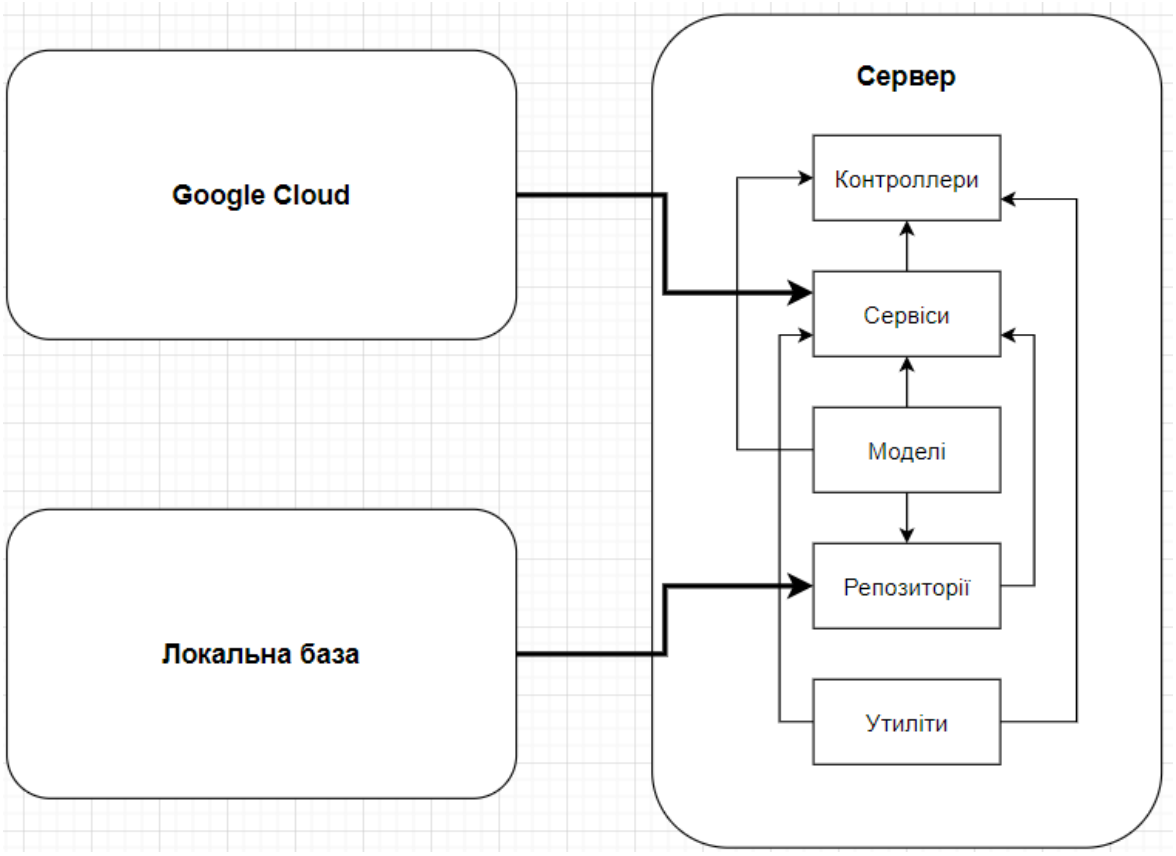


Рис. 3.1 Архітектура серверної частини

Серверна частина складається із 3-х частин:

- Сервер, який, в свою чергу, складається із 5 основних модулів:
  - controller (на діаграмі «контроллери»)
  - service (на діаграмі «сервіси»)
  - model (на діаграмі «моделі»)
  - repository (на діаграмі «репозиторії»)
  - util (на діаграмі «утиліти»)
- Віддалена база даних – Google Cloud
- Локальна база даних

3.2 Структура серверної частина

controller – основний модуль сурверу, який по своїй суті є його інтерфейсом. Саме через цей модуль відбувається комунікація серверу із всіма клієнтами.

Містить в собі:

- com.github.grishasht.docdropper.controller.DocumentController – клас, який відповідає за отримання всіх необхідних користувачу даних про документи. Містить в собі набір методів, які описують певний ендпоінт та які вміють обробляти дані по відповідних ендпоінтах.

Табл.3.1 Опис методу getAll(UUID userGuid)

Метод	getAll(UUID userGuid)
Описання	Вилучання всіх документів
HTTP-метод	GET
Шлях	{userGuid}/documents/

Response (json)	[
	{
	"id": 1123,
	"name": "Lab1",
	"guid": "5cfa6497c5ee5900046c92b9",
	"format": "pdf",
	"size": 35684296,
	"date_uploaded": "2020-01 01T00:00:00.000Z",
	"date_last_updated": "2020-03 01T00:03:34.000Z",
	"userGuid": "5cfa6497c5ee5900046c92b9"
	}
	]

Табл.3.2 Опис методу getByName(UUID userGuid, String name)

Метод	getByName (UUID userGuid, String name)
Описання	Вилучання конкретного документа по імені
HTTP-метод	GET
Шлях	{userGuid}/documents/{name}/

Response (json)	{
	"id": 1123,
	"name": "Lab1",
	"guid": "5cfa6497c5ee5900046c92b9",
	"format": "pdf",
	"size": 35684296,
	"date_uploaded": "2020-01 01T00:00:00.000Z",
	"date_last_updated": "2020-03 01T00:03:34.000Z",
	"userGuid": "5cfa6497c5ee5900046c92b9"
	}

Табл.3.3 Опис методу create(UUID userGuid, Document body)

Метод	create (UUID userGuid, Document body)
Описання	Створення нового докумету в базі
HTTP-метод	POST
Шлях	{userGuid}/documents/

Request (json)	{
	"name": "Lab2",
	"format": "docx",
	"userGuid": "5cfa6497c5ee5900046c92b9",
	"data": "……"
	}

Табл.3.4 Опис методу delete(UUID userGuid)

Метод	delete (UUID userGuid)
Описання	Видалення докуметну з бази
HTTP-метод	DELETE

Шлях	{userGuid}/documents/{name}/
Request (json)	**Empty**

Табл.3.5 Опис методу update (UUID userGuid, Document body)

Метод	update (UUID userGuid, Document body)
Описання	Оновлення існуючого документу в базі
HTTP-метод	PUT
Шлях	{userGuid}/documents/

Request (json)	<pre>{
	"name": "Lab2",
	"format": "docx",
	"userGuid": "5cfa6497c5ee5900046c92b9",
	"data": "……"
	}</pre>

- com.github.grishasht.docdropper.controller. SimilarityController – клас, який відповідає за отримання всіх необхідних користувачу даних про подібність документів. Містить в собі набір методів, які описують певний ендпоінт та які вміють обробляти дані по відповідних ендпоінтах.

Табл.3.6 Опис методу getAll(UUID userGuid)

Метод	getAll(UUID userGuid)
Описання	Вилучання всіх подібностей
HTTP-метод	GET
Шлях	{userGuid}/similarities /

Response (json)	<pre>[
	{
	"id": 1150,
	"count": 0,89,
	"id_first_doc": 11,
	"id_second_doc": 22
	}
	]</pre>

Табл.3.7 Опис методу getAllByDocId(UUID userGuid, Integer id)

Метод	getAllByDocId(UUID userGuid, Integer id)
Описання	Вилучання всіх подібностей за ідентифікатором документу
HTTP-метод	GET
Шлях	{userGuid}/similarity/{id}

Response (json)	[
	{
	"id": 1150,
	"count": 0,89,
	"id_first_doc": 11,
	"id_second_doc": 22
	},
	{
	"id": 1150,
	"count": 0,73,
	"id_first_doc": 11,
	"id_second_doc": 33
	}
	]

- com.github.grishasht.docdropper.controller.UserController – клас, який відповідає за отримання всіх необхідних даних про користувача. Містить в собі набір методів, які описують певний ендпоінт, та які вміють обробляти дані по відповідних ендпоінтах.

Табл.3.8 Опис методу signUp(User requestUser)

Метод	signUp(User requestUser)
Описання	Реєстрація нового користувача на сервері
HTTP-метод	POST
Шлях	/guest/sign-up
Request (json)	{ "name": "example-name", "surname": "example-surname", "email": "example-email",

	"login": "example-login", "password": "example-password" }

Табл.3.9 Опис методу login (User user)

Метод	login (User user)
Описання	Вхід користувача на сервер
HTTP-метод	GET
Шлях	login/

Response (json)	{
	"guid": "5cfa6497c5ee5900046c92b9"
	}

Табл.3.9 Опис методу login (User user)

Метод	login (User user)
Описання	Вилучання даних про користувача
HTTP-метод	GET
Шлях	/{userGuid}/personal

Response (json)	{
	"id": 1150,
	"guid": "5cfa6497c5ee5900046c92b9",
	"name": "example-name",
	"surname": "example-surname",
	"email": "example-email",
	"login": "example-login",
	"password": "example-password",
	}

service – допоміжний модуль сурверу. Даний модуль використовується для того, щоб інкапсулювати бізнес-логіку контроллерів і максимально спросити їх логічне навантаження. Контроллери повинні бути не великими, так як при великій кількості мікросервісів стає дуже важким їх читання.

Модуль service викликається із відповідних контроллерів та містить в собі методи для обробки даних, які приходять на сервер у запиті та даних, які вилучаються із бази. Методи даного модулю викликають в собі методи відповідних репозиторіїв та використовують моделі для відображення сутностей бази даних.

model – модуль, що містить в собі всі класи для відображення сутностей бази даних. Моделями керують та маніпулюють всі модулі проекту, по суті вони являються «мостом» між базою даних та кінцевим продуктом серверу для користувача. Кожна модель називається відповідно до сутності бази, яку вона відображає.

repository – модуль, що містить в собі інтерфейси з методами для зв'язку та управління базою даних. Працює на базі описаних вище фреймворків Spring Data та Hibernate. Викликається із модуля service та використовує модуль model для репрезентування даних, якими маніпулює в базі. Даний модуль являється «мостом» між сервером та базою.

util – модуль який містить в собі «класи-помічники». Використовується для того, щоб надати іншим модулям додатковий функціонал, який не може знаходитися в тих модулях через принципи об'єктно-орієнтованого програмування.

3.3. Клієнтська частина

Клієнтська частина написана на базі фреймворку JavaFX. Даний фреймворк складається із набору різноманітних структур інтерфейсу які в подальшому використовуються для того щоб ‘‘зліпити’’ інтерфейс застосунку.

В клієнській частині дипломного проєкту використовується FXML технологія JavaFX. Програма, написана за допомогою даної технології складається із таких частин:

- Описовий файл .fxml

- Контроллер
- Клас старту застосунку
- Моделі
- Сервіси

Основою даного застосунку є файл main.fxml, так як в ньому описується весь інтерфейс застосунку, а саме: продекларовані всі елементи інтерфейсу, вказані їх розміри і положення в основному вікні, встановлена їх взаємодія та описані обробники до даних елементів. Після того як всі елементи описані у файлі main.fxml прописані, можна згенерувати програмний інтерфейс програми.

Для створення файлу опису main.fxml використовувалася утиліта SceneBuilder.

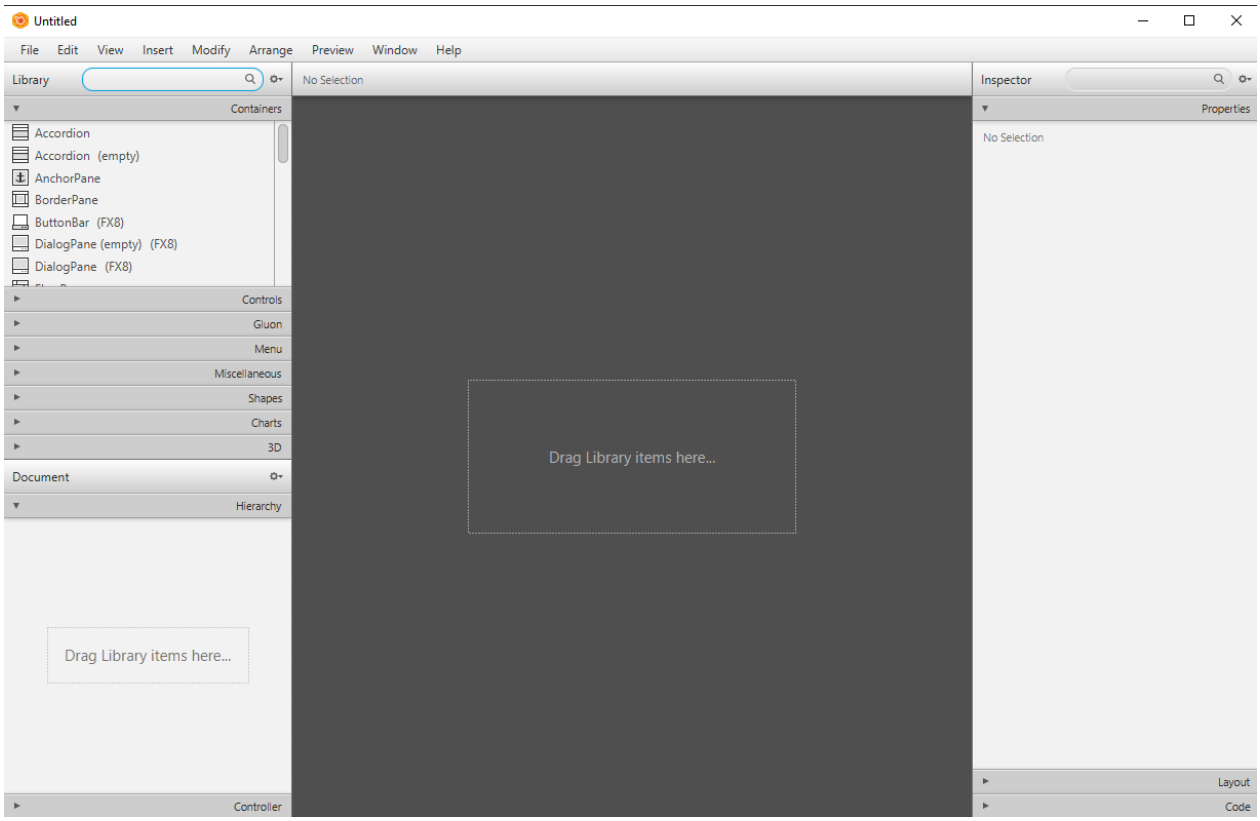


Рис. 3.2. SceneBuilder інтерфейс

Зліва – панель елементів інтерфейсу, по середині – макет інтерфейсу користувача, зправа – панель із налаштуванням властивостей елементів інтерфейсу.

Щоб створити новий елемент інтерфейсу потрібно з лівої частини перетягнути елемент на середню частину програми, тобто на макет, а потім встановити параметри даного елементу в правій панелі.

3.4. Опис інтерфейсу клієнтського застосунку

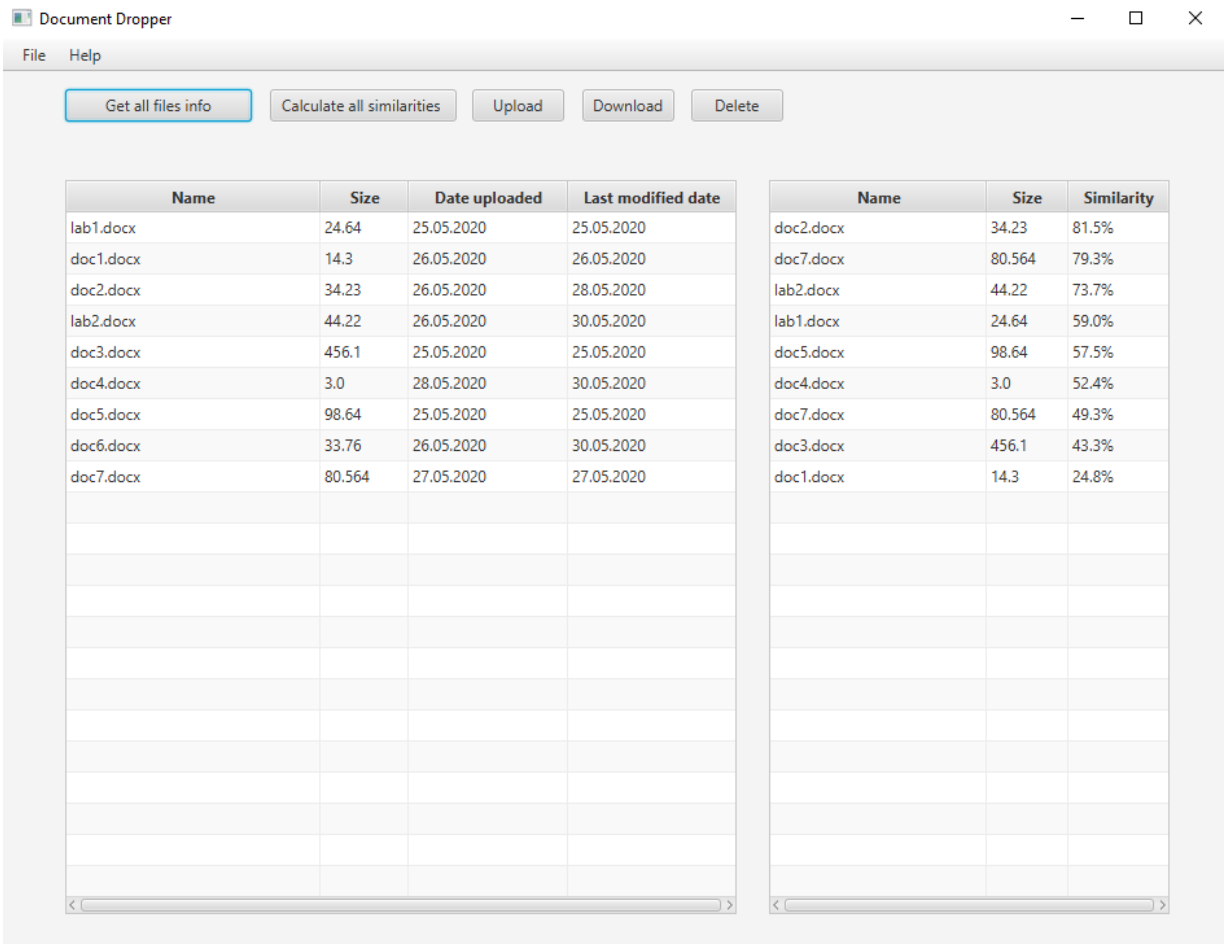


Рис. 3.3 Інтерфейс клієнтського застосунку

Інтерфейс користувача складається із таких частин:

- Панель заголовку, яка містить саму назву програми та меню із опціями:

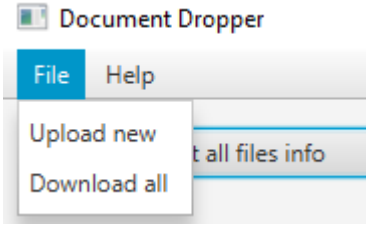


Рис. 3.4 Меню файлів

Upload new – завантажити новий файл на сервер.

Download all – завантажити всі файли із серверу на власний комп’ютер.

- Панель із кнопками для маніпуляції документами:

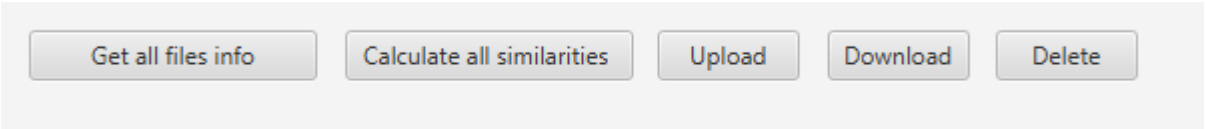


Рис. 3.5 Панель із кнопками керування

Get all files info – показати всі файли користувача, які є на сервері.

Calculate all similarities – визначити подібності для всіх фалйів.

Upload – завантажити свій файл на сервер. При натисканні на дану кнопку відкриється файловий провідник Windows, в якому потрібно буде вибрати файл, який користувач бажає завантажити.

Download – завантажити файл із серверу на власний комп’ютер. Щоб завантажити файл потрібно спочатку обрати його в лівій панелі із списку файлі а потім натиснути на цю кнопку.

Delete – видалити файл із серверу. Щоб видалити файл із серверу потрібно вибрати файл із списку файлів та натиснути на саму кнопку.

- Таблиці із списком файлів, які знаходяться на сервері:

Name	Size	Date uploaded ▲	Last modified date
lab1.docx	24.64	25.05.2020	25.05.2020
doc5.docx	98.64	25.05.2020	25.05.2020
doc3.docx	456.1	25.05.2020	25.05.2020
doc1.docx	14.3	26.05.2020	26.05.2020
doc2.docx	34.23	26.05.2020	28.05.2020
doc6.docx	33.76	26.05.2020	30.05.2020
lab2.docx	44.22	26.05.2020	30.05.2020
doc7.docx	80.564	27.05.2020	27.05.2020
doc4.docx	3.0	28.05.2020	30.05.2020

Рис. 3.6 Таблиця списку файлів.

Таблиця складається із колонок:

- Name – ім’я документу;
- Size – розмір документу;
- Date uploaded – дата, коли документ був відправлений на сервер;
- Last modified date – дата, коли документ змінювався востаннє.

- Таблиці із списком подібних файлів:

Name	Size	Similarity
doc2.docx	34.23	81.5%
doc7.docx	80.564	79.3%
lab2.docx	44.22	73.7%
lab1.docx	24.64	59.0%
doc5.docx	98.64	57.5%
doc4.docx	3.0	52.4%
doc7.docx	80.564	49.3%
doc3.docx	456.1	43.3%
doc1.docx	14.3	24.8%

Рис. 3.7 Таблиця списку подібностей

Імена колонок:

Name – ім’я файлу, який перевірився на подініть із вибраним файлом із попередньої таблиці;

Size – розмір файлу;

Similarity – відсоток подібності фалу на вибраний із попередньої таблиці.

- Форми входу в систему:

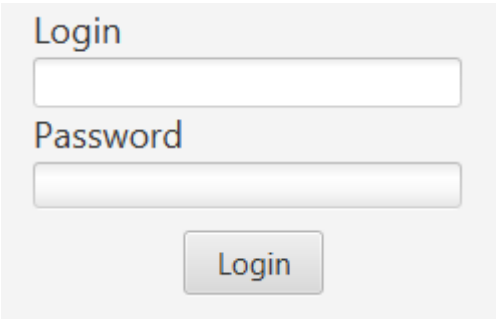


Рис. 3.8 Форма входу в систему

Login – поле для введення логіну для входу в систему;

Password – поле для введення паролю для входу в систему;

3.5. Алгоритм знаходження подіних документів

Для початку користувачу необхідно запустити клієнтський застосунок та авторизуватися в системі, якщо він у ній зареєструвався до цього. Після успішної авторизації відкриється головне відкно застосунку із всіма перечисленими файлами користувача. Коли користувач вибере конкретний файл у нього з'являється змога відкрити контекстне меню, в якому будуть варіанти маніпуляції документом. Серед варіантів:

- Find similarities – порівняння всіх файлів до вибраного, обчислення відсотку подібності та виведення списку у таблицю подібних файлів.
- Delete – видалення файлу із системи

Проте, нас цікавить саме кнопка Find similarities із контекстного меню.

Коли користувач вибрав даний пункт із меню, клієнтський застосунок відправляє на сервер HTTP -GET запит, в тілі якого містьтися ім'я файлу, як його ідентифікатор в базі даних, та унікальний ідентифікатор користувача userGuid, по якому також ідентифікується файл у базі.

Сервер, отримавши запит про необхідність обчислити унікальність документу, перенаправляє цей запит у SimilarityController модуль, який буде його обробляти. Витягнувши із запиту всі необхідні вхідні дані, цей контроллер викликає SimilarityService модуль, передавши в нього їх.

SimilarityService, отримавши дані, дістає по них метаінформацію цього файлу із бази даних використовуючи IDocumentRepository модуль. Якщо такого файлу в базі даних не виявляється, то формує повідомлення про помилку та відповідну відповідь серверу із кодом помилки 400. У випадку коли сервер знайшов такий файл у базі, отримав його метаінформацію, він дістає тіло всіх файлів із віддаленої бази даних та запускає механізм їх порівняння. Після закінчення роботи алгоритму порівняння, формується

список із отриманих результатів, а всі не потрібні більше файли видаляються із сервера з ціллю розвантаження пам’яті системи.

Список із результатами повертається назад у SimilarityController, який конвертує ці файли у формат json та, за допомогою REST мікросервісів, відправляє відповідь із таблицею результатів порівняння в тілі відповіді.

Клієнтський застосунок, отримавши відповідь серверної частини, та вилучивши дані із тіла відповіді, формує із цих даних таблицю, яка в застосунку називається «таблиця подібностей», в якій користувач уже має змогу їх побачити, та відсортувати так як йому потрібно, відштовхуючись від назв колонок.

При натисканні на кнопку Get all files info в інтерфейсі користувача, формується такий же запит на сервер про необхідність обчислення подібностей, але на цього разу уже без указання конкретного файлу, а лише унікального ідентифікатора користувача. Як вказувалося вище, все буде викону через SimilarityService, але цього разу цей сервій вилучить із бази всю метаінформацію документів, тіла всіх документів із віддаленої бази та буде порівнювати кожен документ із кожним. В результаті ми отримаємо структуру хеш-таблицю, в якій значенням ключа буде назва файлу, а ключем список із даних про унікальність відносно інших файлів користувача. Всі непотрібні більше дані видаляться із серверу, для розвантаження пам’яті серверу.

Отримана хеш-таблиця повернеться назад в SimilarityController, який сформує із неї тіло відповіді у форматі json та за допомогою REST-мікросервісів відправить цю відповідь клієнтському застосунку.

Клієнтський застосунок, отримавши відповідь серверної частини, та вилучивши дані із тіла відповіді, формує із цих даних таблицю, яка в застосунку називається «таблиця подібностей», в якій користувач уже має змогу їх побачити, та відсортувати так як йому потрібно, відштовхуючись від назв колонок.

Для відображення таблиці унікальності для бажаного файлу потрібно просто вибрати його у таблиці «файлів користувача».

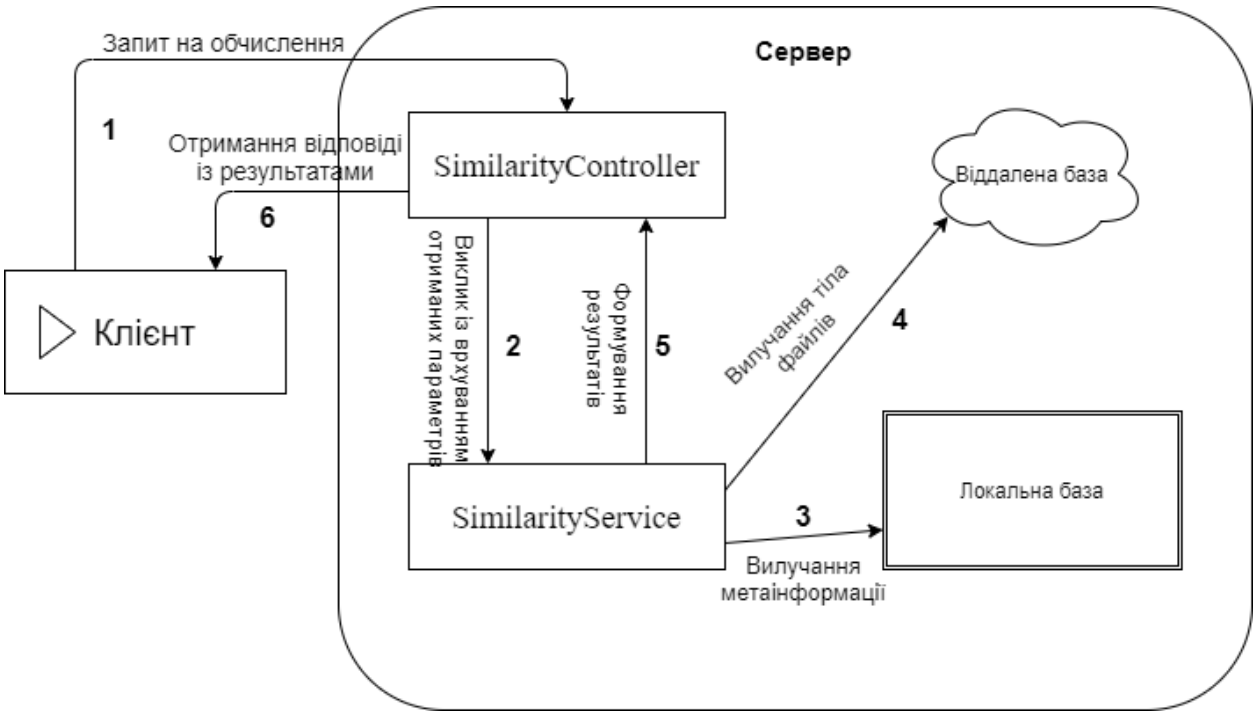


Рис 3.9. Загальний алгоритм роботи системи.

3.6. Висновок

Отже, в ході розробки дипломного проєкту було розроблено малу систему, яка складається із клєнтського додатку та серверної частини, які взаємодіють один з одним, використовуючи мережу Internet та HTTP протокол. А також було розроблено алгоритм визначення унікальності документу.

Основною особливістю даної системи є те, що ці модулі працюють відокремлено один від одного завдяки REST архітектурі. Така система надає можливість розробки одразу багатьох клієнтських застосунків із різним інтерфейсом. Інтерфейс користувача ні яким чином не є прив’язаним до архітектури серверної частини, а отже, його можна розробити за допомогою будь яких інструментів.

Для зручності реалізації клієнтського застосунку була використана технологія JavaFX, яка надає змогу побудувати досить лешко, швидко та ефективно клієнтський застосунок. Однією із переваг даної технології є описовий файл .fxml який надає можливість додати новий елемент інтерфейсу просто описавши його в такому файлі. Це пришвидшує розробку інтерфейсу, збільшує читабельність коду програми та робить архітектуру проекту зрозумілішою. Також серед переваг можна виділити ще той факт, що дана технологія розробляється за допомогою мови Java і не потрібно знати ні яких інших мов чи додаткових інструментів, для того щоб розробити застосунок.

Основною частиною системи є серверна частина. Це ядро всієї системи. Вся логіка виконується на серверній частині. Це значно розвантажує користувача, якому не потрібно думати про ресурси власного комп'ютору. Серверна частина розроблена спеціально таким чином, щоб бути гнучкою, щоб клієнт міг використовувати її якнайпростіше, та міг доповнювати свої додатки як йому заманеться.

4. АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Функціональні вимоги до системи

Звісно, на ринку існує багато рішень. Деякі із них відкриті для користування, а деякі розробляються конкретно під вимоги користувача. Даний проєкт розроблявся на власному ноутбуці, відповідно вимоги ставилися такими, щоб він без проблем міг плавно і без завистань працювати.

По-друге, система повинна бути така, щоб не споживати багато ресурсів кінцевого користувача. Саме тому було вирішено виконувати алгоритм порівнювання файлів та всю основну логіку на серверній частині. В такому випадку для користувача залишиться лише те навантаження яке буде створювати клієнтський застосунок для використання отриманих даних. Так як даних у користувача скоріше всього майже не буде, і враховуючи потужність сучасних обчислюваних машин, то клієнтський застосунок майже не буде споживати ресурсів системи.

Найбільшу кількість ресурсів користувача буде споживати інтерфейс клієнтського застосунку, що пов’язано із використанням JVM платформи та технології JavaFX.

Навантаженість серверної частини насамперед залежить від кількості користувачів. Проте, серверна частина розроблена таким чином, щоб також споживати мінімум. Наприклад, для того щоб зекономити кількість фізичної пам’яті використовується Google Cloud, який при досить не великій сумі дозволяє збільшувати об’єм пам’яті, тому при можливості можна легко скористатися такою можливістю. Найбільше навантаження буде складати саме обчислення відсотку подібності документів, тому до серверної частини ставляться вимоги саме до обчислюваної спроможності комп’ютера.

Також необхідно зазначити необхідність у з’єднанні з мережею інтернет для обох частин проєкту. Дякуючи теперішньому часу, наявність швидкого інтернету не є проблемою. Для користувача не потрібно надшвидкого

з’єднання, але для серверної частини потрібно. Тому найкращим рішенням буде розмістити серверну частину на певному хості, на приклад Heroku чи любий інший сервіс, який надає подібні послуги.

Також серед переваг системи можна виділити можливість встановлення на будь-яку операційну систему. Це зв’язано із кросплатформенністю мови програмування Java та всієї JVM платформи.

Даний проєкт розроблявся на операційній системі Windows 10.

Отож, для клієнтського застосунку необхідний такий мінімум:

- Будь-яка операційна система;
- Процесор – Intel Pentium, 2 ядра, 2 потоки, 2.5 ГГц;
- Оперативна пам’ять – 512 Мб;
- Графічний процесор – Intel HD Graphics;
- Фізична пам’ять – 512 Мб, HDD
- Необхідно встановити Java Runtime Environment для Java 11. Завантажити можна безкоштовно на сайті офіційного розробника платформи Oracle Corp.;
- Швидкість з’єднання із мережею Internet – 1 Мбіт

Для роботи серверного застосунку необхідно:

- Будь яка операційна система. Краще всього будь-яка ОС сімейства Linux;
- Процесор – Intel Core i5, 4 ядра, 8 потоків, 3.0 ГГц;
- Оперативна пам’ять 16 Гб;
- Графічний процесор – Intel HD Graphics;
- Фізична пам’ять – 10 Гб, SSD;
- Необхідно встановити Java Runtime Environment для Java 11. Завантажити можна безкоштовно на сайті офіційного розробника платформи Oracle Corp.;
- Швидкість з’єднання із мережею Internet – 100 Мбіт

4.2. Бізнес вимоги до системи

Клієнтський застосунок повинний мати зрозумілий та простий інтерфейс, щоб користувачу було легко розібратися із системою. Він повинний надавати можливості реєстрації на сервері та входу в систему. Також був вибраний мінімалістичний інтерфейс із холодними відтінками.

Одним із головних завдань серверної частини є захист даних користувача та всіх його документів. Тому однією із головних бізнес вимог було створення механізмів захисту інформації.

Клієнтський застосунок повинен забезпечувати такі основні функції:

- Перегляд документів, які знаходяться на сервері;
- Завантаження нових документів на сервер;
- Завантаження документів з серверу;
- Оновлення інформації про документи на сервері;
- Сортування документів по параметрах;
- Отримання інфомації про подібність документу на інші;
- Виклик функцій підраховування відсотків подібності документів один на одного;
- Низьке навантаження комп'ютера користувача.

Серверна частина повинна забезпечувати такі основні функції:

- Надання інтерфейсу API зовнішнім користувачам;
- Обробка запитів користувача;
- Реєстрація нових користувачів у системі;
- Вхід в систему для зареєстрованих користувачів;
- Захист інформації користувачів;
- Підраховування подібності файлів один до одного;
- Робота із локальною базою даних;
- Робота із віддаленою базою даних.

4.3. Тестування системи

Тестування системи проводилося на власному комп’юторі, який чудово справляється із подібними задачами. Опис його системних властивостей:

- Операційна система – Windows 10;
- Процесор – Intel Core i3, 2 ядра, 4 потоки, 2.4 ГГц;
- Оперативна пам’ять – 16 Гб;
- Відеопроцесор – AMD Radeon 8700M, відеопам’ять 2 Гб + Intel HD Graphics;
- Фізична пам’ять – 240 Гб, SSD;
- Швидкість з’єднання із мережею Internet – 100 Мбіт;
- Java Runtime Environment для Java 11.

Як можна помітити, потужності власного комп’ютора вистачає із запасом, але з цим також є проблема, яка проявляється в тестуванні всієї системи. Через таку потужність складно протестувати систему в складних умовах, коли системі не вистачає ресурсів. Із-за цього складно передбачити поведінку системи в умовах навантаження. Через це було вирішено запускати проєкт паралельно із багатьма іншими програмами, що не дало потрібного ефекту, так як проєктом одночасно користується лише одна особа, а значить змоделювати роботи при великій кількості запитів на сервер, а із задачами для одного користувача система чудово справляється при любих умовах.

4.3.1. Тестування клієнтського застосунку

Виходячи з бізнес та функціональних вимог, було створено застосунок який не споживає велику потужність системи та із простим інтерфейсом користувача. Нижче наведені приклади споживання потужності:

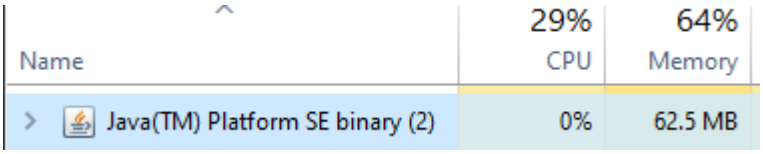


Рис. 4.1 Споживчість клієнтського застосунку.

Як можна помітити, застосунок працює в середовищі JRE, яке забирає найбільший відсоток потужності з тієї, що вимагає сам застосунок. При високому навантаженні комп'ютора, застосунок почуває себе чудово, без натяків на зависання.

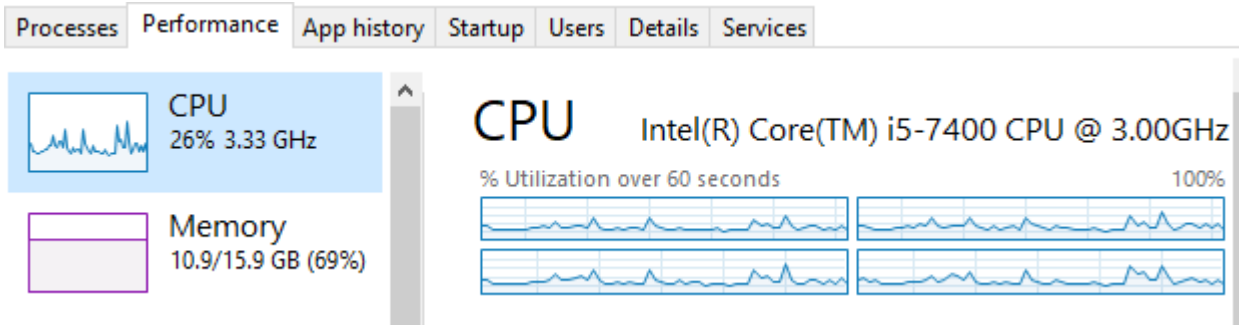
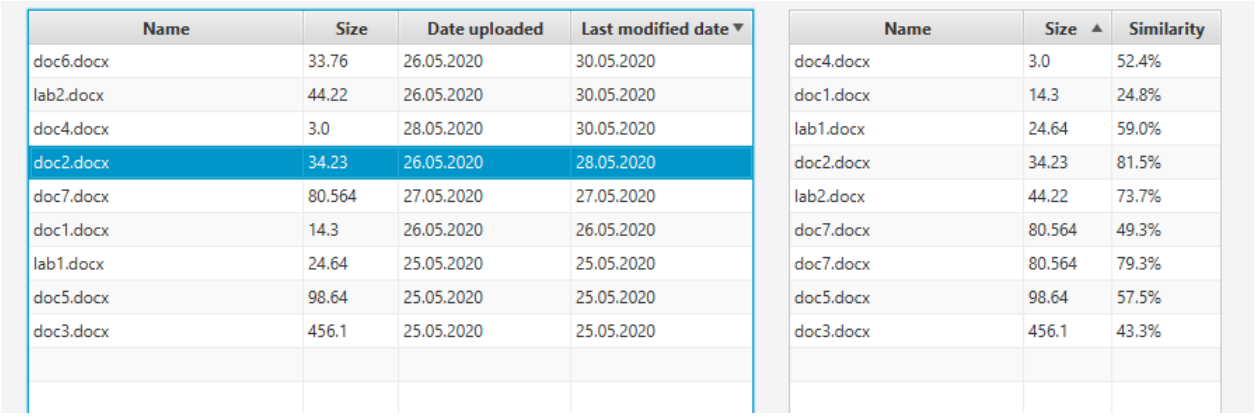


Рис. 4.2 Завантаженість системи в момент тестування клієнтського застосунку.

При використанні власне застосунку також ні яких дефектів не виявлено, всі таблиці завантажуються та відображаються коректно, а також працює сортування даних таблиць по указаних колонках.



Name	Size	Date uploaded	Last modified date
doc6.docx	33.76	26.05.2020	30.05.2020
lab2.docx	44.22	26.05.2020	30.05.2020
doc4.docx	3.0	28.05.2020	30.05.2020
doc2.docx	34.23	26.05.2020	28.05.2020
doc7.docx	80.564	27.05.2020	27.05.2020
doc1.docx	14.3	26.05.2020	26.05.2020
lab1.docx	24.64	25.05.2020	25.05.2020
doc5.docx	98.64	25.05.2020	25.05.2020
doc3.docx	456.1	25.05.2020	25.05.2020

Name	Size	Similarity
doc4.docx	3.0	52.4%
doc1.docx	14.3	24.8%
lab1.docx	24.64	59.0%
doc2.docx	34.23	81.5%
lab2.docx	44.22	73.7%
doc7.docx	80.564	49.3%
doc7.docx	80.564	79.3%
doc5.docx	98.64	57.5%
doc3.docx	456.1	43.3%

Рис 4.3. Демонстрація відображення таблиць

Із рисунку вище помітно, що таблиця зліва відображує всі файли які є на даний момент на сервері та одночасно сортує їх по колонці Las modified date (дата останнього оновлення файлу). Таблиця зліва також працює коректно, так

як вона демонструє список файлів які схожі на файл doc2.docx, а також сортує файли по колонці Size (розмір файлу).

Отже по даним рисункам видно, що клієнтський застосунок чудово відповідає функціональним та бізнес вимогам.

4.3.2. Тестування серверної частини

Виходячи з бізнес та функціональних вимог було створено серверний програмний модуль, який є мультипоточним, тобто може працювати на багатоядерному процесорі та обробляти декілька запитів одразу, не споживає багато оперативної пам'яті та велику кількість ресурсів системи. Як і планувалося, локальна база не займає багато фізичної та оперативної пам'яті.

Також був доданий сучасний ефективний алгоритм авторизації користувача, та механізми захисту інформації.

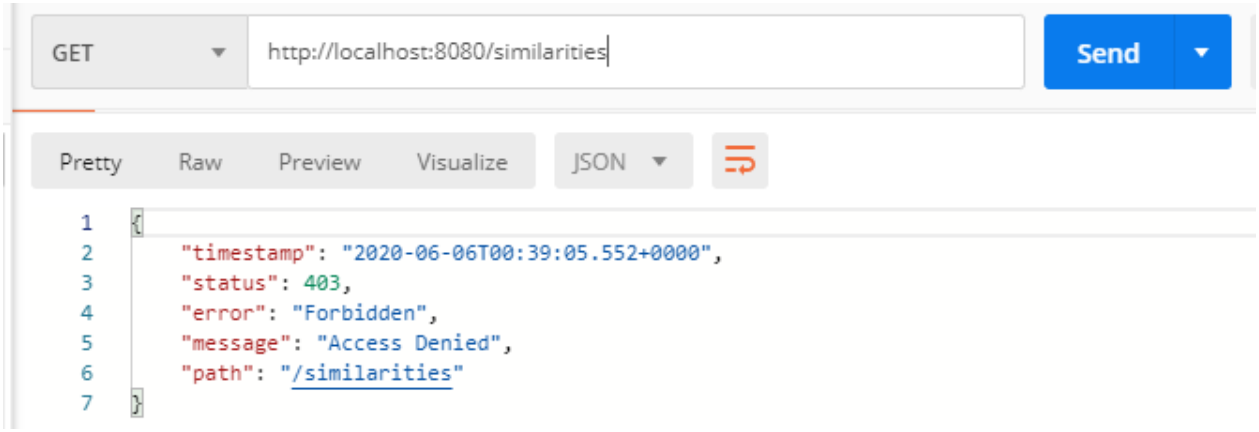


Рис. 4.4 Тестування авторизації серверу.

За допомогою інструменту Postman, який дозволяє зручно виконувати запити на сервер бачимо, що сервер не надає доступу до своїх ресурсів для незареєстрованих в системі користувачів.

Також за допомогою того ж інструменту, на рисунку нижче бачимо, що після успішної авторизації на сервер, після встановлення HTTP header-параметру Authorization значення, яке містить в собі зашифровані в форматі JWT за алгоритмом RSA256 логін та пароль користувача, та після

проходження всього процесу входження в систему ми можемо отримати відповідь у вигляді json із всіми заданими користувачем при реєстрації даними. А, отже, авторизація працює і це означає, що ні яка третя персона не зможе використати ресурси серверу.

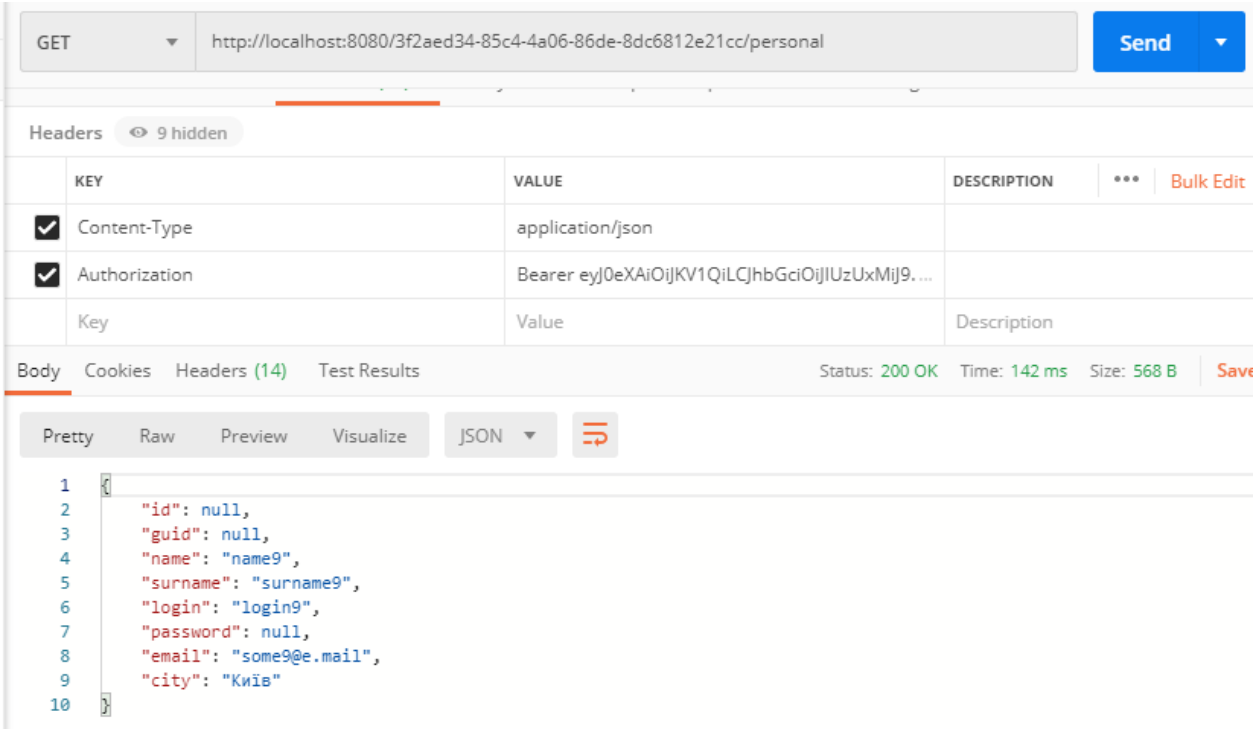


Рис. 4.5 Отримання даних авторизованого користувача.

На рахунок споживчості серверного модулю, то тут все цікавіше. Через використання не малої кількості фреймворків та інструментів для запуску і використання серверу, даний модуль споживає значно більше оперативної пам’яті, ніж клієнтський застосунок, але, як бачимо, процесор майже не навантажує у стані очікування.

Рис. 4.6 Споживчість серверного модулю.

Також тестування проводилися при не малому навантаженні комп’ютеру і серверний модуль проявив себе чудово.

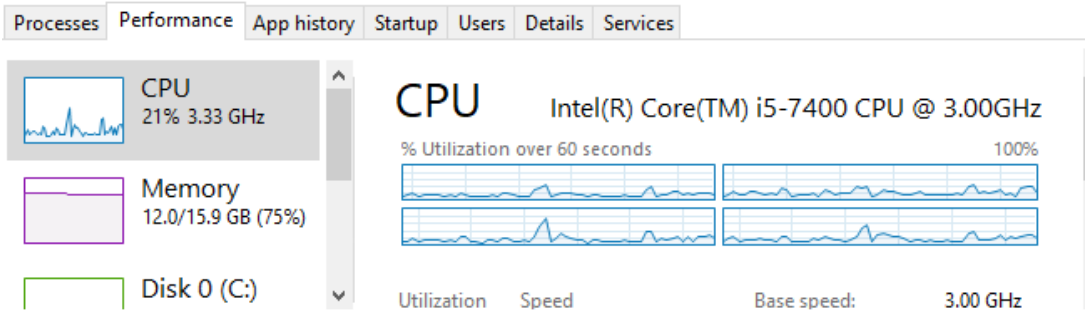


Рис. 4.7 Навантаженість комп'ютера при тестуванні серверного модулю.

Також слід зазначити, що не малий об'єм ресурсів комп'ютеру займає локальна база даних PostgreSQL, на що слід звернути увагу.

4.4. Висновок

Отже, в ході розробки дипломного проєкту враховувалися абсолютно усі бізнес та функціональні вимоги. Із приведених зображень можна зробити такі висновки:

- Система не є вибагливою до ресурсів процессору;
- Система не є вибагливою до оперативної та фізичної пам'яті;
- Однією з переваг системи є сумісність із буль-якою операційною системою;
- Клієнтський застосунок володіє зручним та зрозумілим інтерфейсом користувача;
- Серверний застосунок володіє основними механізмами інформації;
- Імплементована REST – архітектура як одна із основних переваг системи;
- Система оптимізована за рахунок мультипоточності та Google Cloud.

Для покращення результатів тестування системи необхідна велика кількість користувачів, які будуть одночасно «штурмувати» серверну частину. В такому випадку можна буде протестувати систему на стійкість у несприятливих умовах. Також було б чудово протестувати дану систему на різних операційних системах.

ВИСНОВКИ

В ході роботи над дипломнмим проєктом було розроблено систему, яка складається із двох частин: серверної – основної та клієнтської, які використовують REST-архітектуру. Така архітектура надає системі наступні переваги:

- Надання програмного інтерфейсу;
- Відокремлення серверної логіки від клієнтської;
- Стандартизованість серверних відповідей;
- Використання HTTP в ‘‘чистому’’ вигляді

Для розроблення системи було вибрано платформу JVM та мову програмування Java, а також всі технології, бібліотеки і фреймворки, які стосуються цієї платформи, що також надало переваг системі, таких як:

- Кросплатформеність;
- Високопродуктивність системи;
- Можливість використання інструментарію мов сімейства JVM

Завдяки використанню принципів об’єктно-орієнтованого програмування, дотримуванню принципів правильного написання коду програми та принципів SOLID, досяглася висока читабельність коду, в зв’язку з чим можна легко виявити помилки та / або доповнити існуючу систему новим функціоналом, що також є дуже великою перевагою з точки зору бізнес вимог.

Також у роботі були враховані всі функціональні та бізнес вимоги, поставлені до проєкту, серед яких основними можна виділити захищеність серверного модулю, можливість використання серверного модулю не лише для написання клієнтських застосунків, а і для написання плагінів для інших програм та їх розширення.

Клієнтський застосунок розроблений за допомогою технології JavaFX задля того, щоб зменшити навантаження на комплексність системи. Так як

дана технологія також входить до сімейства JVM, то клієнтський застосунок переймає такі ж переваги як і серверна частина.

Отже, основними перевагами даної системи є:

- Незалежність від типу операційної системи;
- Свобода вибору стеку інструментів та технологій для написання клієнтського застосунку;
- Не високе споживання ресурсів комп’ютера;
- Використання сучасний технологій;
- Оптимізація використання фізичної пам’яті за рахунок технології Google Cloud;
- Захищеність даних користувачів та ресурсів серверу.

Систему протестовано з урахуванням можливих потреб та проблем користувача. Було перевірено всі можливі вразливості системи такі як sql-ін'єкції та отримання доступу до ресурсів системи несанкціонованим користувачам. Під час тестування розроблена системи показала себе як високопродуктивною та нересурсоспоживчою.

Через те, що модуль був розроблений на базі сучасних технологій і більшість схожих рішень існують уже давно і , відповідно уже застаріли, то основною його перевагою є гнучкість, можливість легкого доповнення функціональних можливостей, кросплатформенність (чого не має майже у більшості подібних рішень) та можливість використання не лише як самостійної системи, а і як доповнення до уже існуючої.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

- Clean Code, Robert Cecil Martin, 2008
- Java: A Beginner's Guide, Herbert Schildt, 2002
- Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, Michael Sikorski, Andrew Honig, 2012
- Mastering JavaFX 11: Build Advanced and Visually Stunning Java Applications, Sergey Grinev, May 30, 2018
- Wikipedia.com
- https://wikipedia.org/
- https://www.oracle.com/index.html
- https://developers.google.com/api-client-library/java/google-api-java-client/oauth2
- https://developers.google.com/drive/api/v3/manage-uploads
- https://codete.com/blog/5-common-spring-transactional-pitfalls/
- https://refactoring.guru/
- https://docs.oracle.com/
- https://codelabs.developers.google.com/
- https://stackoverflow.com/