КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки
на тему:

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СЕРВІСУ ПРАЦЕВЛАШТУВАННЯ СОЦІАЛЬНОЇ МЕРЕЖІ

Виконав студент 2-го курсу магістратури
Микола СИЗЬКО

(підпис)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Володимир ШЕВЧЕНКО

(підпис)

Науковий консультант:
доцент, кандидат фізико-математичних наук
Віктор ВОЛОХОВ

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«08» травня 2023 р.,
протокол № 16

Завідувач кафедри
Микола НІКІТЧЕНКО

(підпис)

Київ-2023

РЕФЕРАТ

Обсяг роботи 60 сторінок, 16 ілюстрацій, 1 таблиця, 17 джерел посилань, 2 додатки.

СОЦІАЛЬНА МЕРЕЖА, СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА, JAVA, SPRING FRAMEWORK, БАЗА ДАНИХ, ПРАЦЕВЛАШТУВАННЯ, СУБД POSTGRESQL, АРХІТЕКТУРА, АЛГОРИТМ, RSQL, DOCKER, SWAGGER, POSTMAN.

Об’єктом розробки є сервіс працевлаштування соціальної мережі. **.** Метою кваліфікаційної роботи є розробка сервісу соціальної мережі для роботи з такими сутностями, як користувацькі профілі та публікації.

Інструментами створення програмного продукту було обрано мову програмування Java 17, Spring Framework. База даних PostgreSQL. Сервісно-орієнтована архітектура. Також у проєкті використовуються наступні засоби та технології: Docker, Swagger, Postman, RSQL.

Результати роботи: було досліджено існуючі соціальні мережі, розглянуто статистику соціальних мереж за популярністю, розроблено вимоги та змодельовано поведінку сервісу, досліджено сучасні архітектурні шаблони програмного забезпечення, обрано мову програмування Java, Spring Framework, створено сервіс працевлаштування соціальної мережі, реалізовано створення нових користувачів з персональною інформацією і аватаром, створення публікацій, пошук з фільтром.

Дана робота може бути використана для створення нового програмного забезпечення зі зручним та ефективним інтерфейсом користувачів, що допоможе їм знайти потрібне місце працевлаштування, яке відповідає їхнім потребам та кваліфікації.

Зважаючи на тенденції швидкого розвитку соціальних мереж з’являтимуться потреби у створенні сервісів з новими, важливими для користувачів функціями, тому розпочаті у кваліфікаційній роботі дослідження варто продовжувати.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

SOA – Service-oriented architecture, сервісно-орієнтована архітектура;

ESB – Enterprise service bus, корпоративна шина даних;

RSQL – RESTful Service Query Language, мова запитів для параметричної фільтрації записів у RESTful API;

REST – REpresentational State Transfer, «передача репрезентативного стану»

URI – Uniform Resource Identifier, уніфікований ідентифікатор ресурсів;

API – Application Programming Interface, інтерфейс програмування додатків;

HTTP – HyperText Transfer Protocol, протокол передачі даних;

XML – Extensible Markup Language, розширювана мова розмітки;

JSON – JavaScript Object Notation, запис об'єктів JavaScript;

JDBC – Java Database Connectivity, підключення до бази даних Java;

SDLC – Software Development Lifecycle, життєвий цикл програмного забезпечення;

СУБД – система управління базами даних;

БД – база даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогоднішній день соціальні мережі є популярним інструментом для працевлаштування. Деякі з них, такі як LinkedIn, можуть надати велику кількість вакансій та можливостей для спілкування з роботодавцями. Проте існують деякі проблеми, які можуть знизити їх ефективність, наприклад, обмежений доступ до вакансій, недостатній захист особистої інформації користувачів та низька якість спілкування між користувачами та роботодавцями, недостатньо розвинута система пошуку та фільтрації. Сервіс, розроблений у рамках кваліфікаційної роботи, побудований на основі сучасних технологій у сфері розробки програмного забезпечення. У ньому реалізовані методи для роботи з профілем користувача та публікаціями, розширена система пошуку з можливістю фільтрації по різним полях. Перевагами сервісно-орієнтованого підходу, що використовується, є можливість швидкого додавання нового функціоналу з гарантією того, що уже реалізовані функції будуть і надалі коректно працювати.

Актуальність роботи та підстави для її виконання. Розвиток можливостей і зростання популярності соціальних мереж та месенджерів, зручність їх інформаційного спілкування суттєво змінили кадрову роботу на ринку праці, зокрема такий важливий її компонент як пошук місця працевлаштування.

Звідси виникає необхідність дослідження та розробки сервісу працевлаштування соціальної мережі, яка зможе задовольнити потреби користувачів у працевлаштуванні.

Мета й завдання роботи. Метою кваліфікаційної роботи є розробка сервісу соціальної мережі для роботи з такими сутностями, як користувацькі профілі та публікації. Було поставлено такі завдання:

- дослідити можливості існуючих соціальних мереж для працевлаштування;

- розробити вимоги та змоделювати систему сервісу для соціальної мережі;
- дослідити сучасні архітектурні шаблони таких сервісних систем;
- розробити та протестувати сервіс у відповідності з вимогами.

Об’єкт, методи й засоби розроблення. Об’єктом розробки є сервіс працевлаштування соціальної мережі. Інструментами створення програмного продукту було обрано мову програмування Java 17, Spring Framework. База даних PostgreSQL. Сервісно-орієнтована архітектура. Також у проєкті використовуються наступні засоби та технології: Docker, Swagger, Postman, RSQL.

Можливі сфери застосування. Дана робота може бути використана для створення нового програмного забезпечення зі зручним та ефективним інтерфейсом користувачів, що допоможе їм знайти потрібне місце працевлаштування, яке відповідає їхнім потребам та кваліфікації.

Апробація роботи та публікації з теми роботи. За результатами роботи було оприлюднено та опубліковано тези на конференції «Шевченківська весна» 14.04.2023 року [1].

РОЗДІЛ 1. СОЦІАЛЬНІ МЕРЕЖІ

1.1 Означення

Соціальна мережа (англ. Social networks) – це вебсайт, який дозволяє зареєстрованим на ньому користувачам розміщувати інформацію про себе й спілкуватися між собою, встановлюючи соціальні зв'язки. Інформація на цьому сайті створюється самими користувачами.

Соціальні мережі об'єднують людей різних країн, релігій, професій, соціальних або вікових груп тощо. Зазвичай вони мають інструменти для створення співтовариств за інтересами, де спілкування відбувається у вужчих колах. [2]

Сучасні соціальні мережі - це місце, де мільйони користувачів з усього світу обмінюються інформацією, взаємодіють, спілкуються та співпрацюють один з одним. Залежно від своїх цілей, користувачі можуть використовувати соціальні мережі для знайомства з новими людьми, обговорення тем, які їх цікавлять, публікації фотографій та відео, спільної роботи над проектами та багато іншого.

Однією з найбільш популярних соціальних мереж є Facebook, який налічує більше 2 мільярдів активних користувачів щомісяця. Інші популярні соціальні мережі - це Twitter, Instagram, LinkedIn, TikTok та Snapchat.

Сучасні соціальні мережі також впливають на політичні, економічні та соціальні процеси у світі. Вони можуть бути використані для поширення пропаганди, організації масових протестів, збору пожертв та координації гуманітарних допомог.

Однак, соціальні мережі також викликають суттєві проблеми, такі як поширення неправдивої інформації, кібербулінг, порушення приватності та залежність від соціальних мереж. У зв'язку з цим, соціальні мережі все частіше стають об'єктом досліджень та регулювання з боку урядових органів та різних організацій.

Загалом, сучасні соціальні мережі мають великий вплив на життя людей та суспільство в цілому, і вони продовжують розвиватись та змінюватись з кожним роком.

1.2 Статистика

Детальний аналіз, проведений командою Kepios, показує, що в січні 2023 року у світі налічувалося 4,76 мільярда користувачів соціальних мереж, що становить 59,4 відсотка від загальної кількості населення планети.

Кількість користувачів соціальних мереж продовжувала зростати і за останні 12 місяців: 137 мільйонів нових користувачів приєдналися до соціальних мереж з цього часу минулого року.

Це дорівнює щорічному зростанню на 3 відсотки, що в середньому становить понад 4 нових користувачів щосекунди.

Останні дані свідчать про те, що понад 9 з 10 інтернет-користувачів користуються соціальними мережами щомісяця.

Однак зауважте, що цифри користувачів соціальних мереж можуть не відображати унікальних людей, і через такі проблеми, як дублікати акаунтів, цифри користувачів соціальних мереж можуть перевищувати цифри, які ми публікуємо для користувачів інтернету - або навіть для всього населення в цілому.

З іншого боку, варто також зазначити, що порівняння кількості користувачів соціальних мереж із загальною кількістю населення може не відображати повну картину використання соціальних мереж, оскільки більшість соціальних мереж обмежують використання своїх платформ особами віком від 13 років і старше.

Для контексту, останні дані свідчать про те, що кількість користувачів соціальних мереж у всьому світі зараз дорівнює майже 78 відсоткам населення планети, що має право користуватися ними.

Дані GWI показують, що типовий користувач соціальних мереж активно використовує або відвідує в середньому 7,2 різних соціальних платформ щомісяця

і проводить в середньому понад 2,5 години на день, користуючись соціальними мережами.

Якщо припустити, що люди сплять від 7 до 8 годин на добу, то ці останні дані свідчать про те, що люди проводять приблизно 15 відсотків свого неспання в соціальних мережах.

Разом, світ витрачає майже 12 мільярдів годин на використання соціальних платформ щодня, що еквівалентно майже 1,4 мільйонам років існування людства.

Facebook залишається найпоширенішою у світі соціальною мережею, але зараз існує шість соціальних мереж, кожна з яких налічує мільярд або більше активних користувачів щомісяця.

Три з цих семи платформ належать компанії Meta.

Крім того, 15 соціальних медіа-платформ мали щонайменше 400 мільйонів активних користувачів у січні 2023 року (рисунок 1):

1. Facebook має 2,958 мільярда активних користувачів щомісяця.
2. Потенційне рекламне охоплення YouTube становить 2,514 мільярда.
3. WhatsApp має щонайменше 2 мільярди активних користувачів щомісяця.
4. Instagram має 2 мільярди активних користувачів щомісяця.
5. WeChat (включно з Weixin) має 1,309 мільярда активних користувачів щомісяця.
6. Реклама в TikTok потенційно може охопити 1,051 мільярда дорослих людей старше 18 років щомісяця.
7. Потенційне рекламне охоплення Facebook Messenger становить 931 мільйони.
8. Douyin має 715 мільйонів(с) активних користувачів щомісяця.
9. Telegram має 700 мільйонів активних користувачів щомісяця.
10. Потенційне рекламне охоплення Snapchat становить 635 мільйонів.
11. Kuaishou має 626 мільйонів активних користувачів щомісяця.
12. Sina Weibo має 584 мільйони активних користувачів щомісяця.
13. QQ має 574 мільйони активних користувачів щомісяця.

14. Потенційне рекламне охоплення Twitter становить приблизно 556 мільйонів.
15. Pinterest має 445 мільйонів активних користувачів щомісяця.

LinkedIn не публікує щомісячні дані про активних користувачів, тому ми не можемо включити його до цього списку. [3]

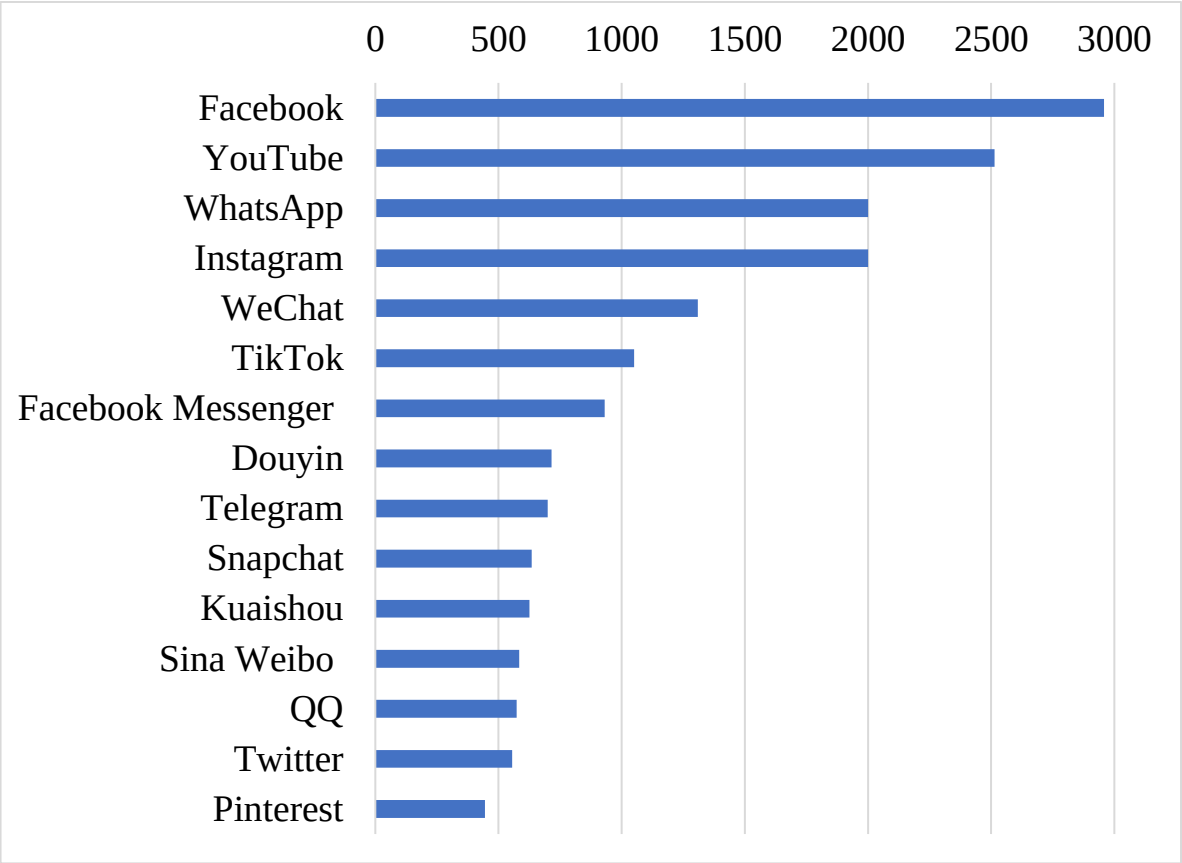

Рисунок 1 – Найбільші соціальні мережі у світі

РОЗДІЛ 2. РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Моделювання вимог

Зважаючи на сучасний ринок праці, велика кількість людей зацікавлені у працевлаштування. Існує попит на нові технології та сервіси, які можуть допомогти в працевлаштуванні більш ефективно та зручно, ніж уже існуючі.

Таким чином, бізнес-потреба полягає у створенні сервісу соціальної мережі для працевлаштування, яка забезпечуватиме ефективний інтерфейс для користувачів, які шукають роботу. Цей сервіс соціальної мережа має забезпечити можливість створити профіль користувача, де користувачі можуть вказати свої професійні навички, досвід та інші важливі дані, що відображають їхню кваліфікацію.

Крім того, мережа повинна мати можливість створювати та переглядати оголошення про вакансії різного типу та об'єму робіт. Також, зручний та ефективний механізм працевлаштування повинен бути включеним у функціональні можливості сурвісу соціальної мережі.

Також необхідно забезпечити можливість спілкування між користувачами, наприклад, чат або систему обміну повідомленнями, щоб сприяти взаємодії між роботодавцями та кандидатами на роботу.

Узагальнюючи, сервіс соціальної мережі для працевлаштування має забезпечувати зручний та ефективний інтерфейс для користувачів, що допоможе їм знайти роботу, яка відповідає їхнім потребам та кваліфікації.

Бізнес-вимоги:

- можливість модифікування сервісів відповідно до тенденцій на ринку;
- система має бути конкурентноспроможною;

Функціональні вимоги - це вимоги до системи, які описують функціональність, яку система повинна надавати користувачам або іншим системам. Ці вимоги описують, як система повинна поводитися при виконанні певних функцій, які функції повинні бути доступні та які параметри повинні бути

відображені на виході. Функціональні вимоги є основою для розробки функціонального дизайну системи та тестування її функціональності.

Функціональні вимоги соціальної мережі:

1. Реєстрація користувачів: користувачі повинні мати можливість створювати облікові записи на платформі.
2. Інформації профілю користувача: система має надавати можливість користувачеві додавати свої ім'я, прізвище, вік, адресу, основну інформацію про користувача, позиція, аватар та фонову картинку.
3. Пошук вакансій: система повинна надавати користувачам зручний та швидкий пошук вакансій за критеріями, такими як категорія роботи, місцезнаходження, розмір зарплати та інші параметри.
4. Надання вакансій: роботодавці повинні мати можливість додавати вакансії на платформу з детальним описом робочих місць, вимогами до кандидатів та іншою необхідною інформацією.
5. Комунікація між користувачами: система повинна надавати можливість спілкуватися між користувачами, в тому числі і між роботодавцями та кандидатами, зокрема через внутрішні повідомлення та чат.
6. Рекомендації вакансій: система повинна надавати користувачам персоналізовані рекомендації вакансій, зокрема на основі їхнього досвіду, кваліфікації та інших факторів.
7. Огляд резюме: роботодавці повинні мати можливість переглядати резюме потенційних кандидатів на платформі та зв'язуватися з ними напряму.
8. Аналітика та звітність: система повинна надавати аналітичні звіти про активність користувачів, динаміку пошуку та надання вакансій, а також інші ключові показники.

Нефункціональні вимоги до системи - це вимоги, які не стосуються функціональності системи, але впливають на її якість та ефективність:

1. Надійність: система повинна бути стійкою та надійною, інакше користувачі можуть втратити довіру до сервісу та перейти до конкурентів.
2. Швидкодія: система повинна працювати достатньо швидко, щоб користувачі не втрачали зацікавленість через довгі часи очікування.
3. Масштабованість: система повинна бути здатною працювати з великою кількістю користувачів та обробляти великі обсяги даних, що дозволить розширювати функціональність сервісу та розвивати бізнес.
4. Безпека: система повинна мати надійну систему безпеки для захисту користувачів та їхніх даних від зловмисників.
5. Сумісність: система повинна бути сумісною з різними браузерами та пристроями, що дозволить користувачам зручно користуватися сервісом з будь-якого пристрою.
6. Доступність: система повинна бути доступною для людей з різними потребами, включаючи людей з обмеженими можливостями, для цього необхідно розробити дизайн, який дозволяє використовувати сервіс людям з будь-якими особливостями.
7. Локалізація: система повинна підтримувати різні мови та локалізацію для різних країн, що дозволить залучати користувачів з різних куточків світу.

2.2 Постановка та алгоритм розв’язання задачі

2.2.1 Постановка задачі

Постановка задачі для сервісу соціальної мережі для працевлаштування полягає в розробці системи, що дозволяє користувачам створювати профілі, додавати свої дані, шукати вакансії та, таким чином, знаходити роботу.

Інформаційна модель - це модель даних, що описує структуру даних та зв'язки між ними, що використовуються в даний момент у системі. Інформаційна модель може бути представлена у вигляді діаграми, таблиці або текстового опису. Вона дозволяє описати потреби користувачів та бізнес-процесів, що впливають на структуру даних в системі. Інформаційна модель забезпечує зрозумілість та однозначність у роботі з даними та допомагає забезпечити їх цілісність та якість.

Інформаційна модель для соціальної мережі зображена на рисунку 2.

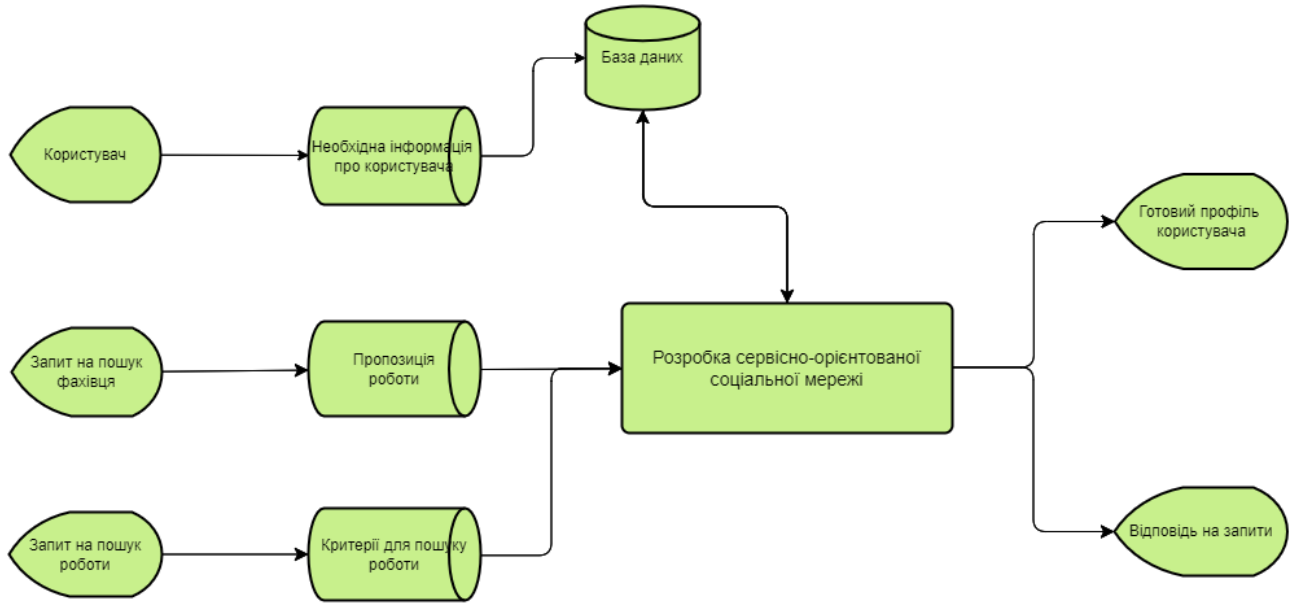

Рисунок 2 – Інформаційна модель соціальної мережі

На вході система отримує від користувача інформацію про нього, а на виході маємо готовий профіль користувача.

За допомогою окремого інтерфейсу компанії можуть створювати запити для пошуку фахівців, які зберігаються та обробляються в системі.

Користувач має можливість надіслати запит на працевлаштування, а на виході з системи має відповідь на запит.

Вихідна інформація соціальної мережі для працевлаштування включає наступне:

1. Користувачі: інформація про користувачів, така як ім'я, прізвище, вік, адреса, основна інформація про користувача, позиція, аватар, фонова картинка та інші деталі профілю.
2. Вакансії: інформація про вакансії, така як назва, опис, вимоги до кандидатів, умови роботи та інші деталі.
3. Компанії: інформація про компанії, які розміщують вакансії, така як назва, розташування, галузь діяльності, інформація про компанію та інші деталі.
4. Рекомендації: рекомендації для користувачів, які можуть допомогти знайти підходящі вакансії або кандидатів.
5. Сповіщення: інформація про сповіщення для користувачів, що стосуються нових вакансій, змін у вакансіях, рекомендацій та інші повідомлення.
6. Статистика: інформація про статистику використання, така як кількість користувачів, кількість вакансій, кількість рекомендацій, кількість сповіщень та інші показники.
7. Інша інформація: інші дані, такі як додаткові деталі про користувачів, вакансії та компанії, можуть також збиратись та зберігатись у соціальній мережі для працевлаштування.

Інформація на вході в соціальну мережу може бути різного типу і форми. Це особисті дані користувача, такі як ім'я, прізвище, електронна пошта, пароль, вік та місце проживання. Крім того, користувач може надати інформацію про свої професійні навички, досвід роботи, освіту та інші важливі дані, які допоможуть знайти йому відповідну роботу. Користувач може ввести інформацію про компанії, які йому цікаві, та підписатися на новини та оголошення цих компаній. Також на вхід можуть надходити інформація від інших користувачів, такі як коментарі та відгуки про компанії та роботодавців.

2.2.2 Алгоритм розв’язання задачі

Алгоритм розв'язання задачі для реєстрації користувачів, алгоритм розв'язання задачі для працевлаштування та алгоритм розв'язання задачі для пошуку фахівця зображено на рисунках 3, 4 та 5 відповідно.

Рисунок 3 – Алгоритм розв'язання задачі 1

Алгоритм розв'язання задачі для реєстрації користувачів:

- Відображення сторінки реєстрації: система відображає сторінку реєстрації з необхідними полями для заповнення (наприклад, електронна пошта, ім'я користувача, пароль, тощо).

- Перевірка введеної інформації: система перевіряє, чи введена інформація відповідає вимогам (наприклад, чи вказано дійсну електронну адресу, чи пароль містить достатньо символів, тощо). Якщо введена інформація некоректна, система повідомляє про помилку та просить користувача ввести правильні дані.
- Збереження даних: якщо введена інформація відповідає вимогам, система зберігає дані про користувача в базі даних.
- Підтвердження реєстрації: система повідомляє користувача про успішну реєстрацію та пропонує ввійти до облікового запису.
- Вхід у систему: якщо користувач погодився з підтвердженням реєстрації, система автоматично перенаправляє його на сторінку входу до системи, де користувач може ввійти до свого облікового запису з використанням введеного раніше логіну та паролю.
- Підтвердження входу: система перевіряє введені користувачем дані та дозволяє входити до облікового запису, якщо логін та пароль введені правильно. Якщо введені дані неправильні, система повідомляє користувача про помилку та просить ввести правильні дані.

Основні кроки алгоритму розв'язання задачі працевлаштування:

- Реєстрація та авторизація користувача в системі.
- Заповнення профілю користувача з вказанням його освіти, досвіду роботи та інших професійних навичок.
- Пошук вакансій за ключовими словами та фільтрування їх за різними параметрами, такими як місце розташування, тип роботи, заробітна плата, рівень досвіду тощо.
- Перегляд детальної інформації про вакансії та можливість відправлення резюме на вибрану вакансію.
- Оцінка користувачем вакансії за допомогою системи рекомендацій, яка базується на раніше вказаних навичках та досвіді користувача.

- Зв'язок користувача з роботодавцем через вбудований чат або електронну пошту для обговорення деталей вакансії та проведення співбесіди.

Рисунок 4 – Алгоритм розв'язання задачі 2

Основні кроки алгоритму для пошуку фахівця:

- Збір вхідних даних: отримання запиту на пошук фахівця та параметрів, за якими потрібно здійснювати пошук (наприклад, область знань, рівень досвіду тощо).
- Аналіз вхідних даних: перевірка введених параметрів на коректність та відповідність вимогам системи.

Рисунок 5 – Алгоритм розв'язання задачі 3

- Пошук фахівців: на основі введених параметрів, система здійснює пошук фахівців в базі даних.
- Ранжування фахівців: після знаходження фахівців, система виконує їх ранжування відповідно до введених параметрів. Наприклад, фахівці з вищим рівнем досвіду можуть мати більший пріоритет.
- Виведення результатів: система виводить результати пошуку фахівців у зручному форматі для користувача.
- Завершення пошуку: користувач може продовжити пошук або зупинити його та вибрати потрібного фахівця.

- Оновлення бази даних: після вибору фахівця, система оновлює базу даних, зберігаючи інформацію про користувача та фахівця.

2.3 Моделювання поведінки системи

Коли аналітик намагається зрозуміти основну прикладну область проблеми, він повинен враховувати як структурні, так і поведінкові аспекти проблеми. На відміну від інших підходів до розробки інформаційних систем, об'єктно-орієнтовані підходи намагаються розглядати основну прикладну область цілісно. Розглядаючи проблемну область як набір варіантів використання, які підтримуються набором взаємодіючих об'єктів, об'єктно-орієнтовані підходи дозволяють аналітику мінімізувати семантичний розрив між реальним набором об'єктів і об'єктно-орієнтованою моделлю предметної області, що розвивається.

Одна з основних цілей поведінкових моделей - показати, як базові об'єкти в предметній області будуть працювати разом, щоб сформувати співпрацю для підтримки кожного з варіантів використання. У той час як структурні моделі представляють об'єкти і взаємозв'язки між ними, поведінкові моделі відображають внутрішній вигляд бізнес-процесу, який описує варіант використання. Процес може бути показаний взаємодією, яка відбувається між об'єктами, що співпрацюють для підтримки варіанту використання, за допомогою діаграм взаємодії (послідовності та зв'язку). Також можна показати вплив, який набір варіантів використання, що складають систему, має на об'єкти в системі за допомогою поведінкових машин станів.

Створення поведінкових моделей - це ітеративний процес, який повторюється не тільки над окремими поведінковими моделями, але також над функціональними і структурними моделями. Під час створення поведінкових моделей часто доводиться вносити зміни у функціональні та структурні моделі [4].

Ніхто, навіть творці UML, не розуміє і не використовує її повністю. Більшість людей використовують невелику підмножину UML і працюють з нею. Потрібно знайти ту підмножину UML, яка працює для вас і ваших колег.

Треба спочатку зосередитися на базових формах діаграм класів і діаграмах послідовності. Це найпоширеніші і найкорисніші типи діаграм. Після того, як їх освоїли, можна почати використовувати більш складні діаграми класів класів та ознайомитися з іншими типами діаграм [5].

Use case діаграма - це діаграма, яка моделює функціональність системи з точки зору її користувачів. Вона використовується для визначення функцій, які система повинна виконувати та ролей, які взаємодіють з системою. Ця діаграма є важливим інструментом в аналізі вимог та проектуванні системи, оскільки допомагає зрозуміти взаємодію користувачів з системою та визначити, які функції повинна виконувати система.

Use case діаграма для соціальної мережі зображена в додатку А.

Згідно з діаграмою, користувач має можливість здійснювати такі маніпуляції над системою:

- робота з обліковим записом,
- пошук інформації,
- створення повідомлень,
- працевлаштування.

Діаграма послідовностей (Sequence diagram) - це графічний інструмент для представлення взаємодії між об'єктами або компонентами в рамках конкретного сценарію або процесу. Вона показує послідовність виконання дій та передачу повідомлень між об'єктами в часі. Діаграма послідовностей є частиною UML (Unified Modeling Language) і широко використовується для моделювання взаємодії систем та їх компонентів.

Діаграма послідовностей соціальної мережі для працевлаштування зображена на рисунку 6.

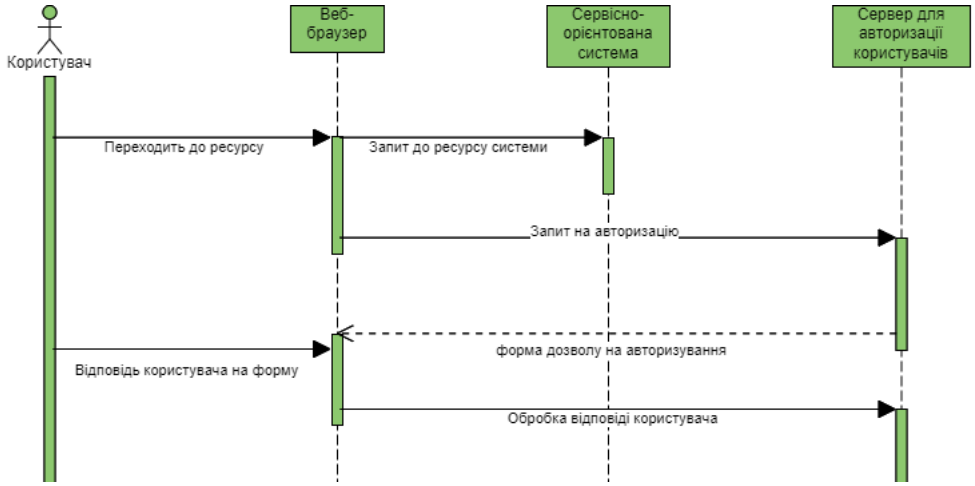

Рисунок 6 – Діаграма послідовностей соціальної мережі

Діаграма класів - це структурна діаграма, що відображає класи, їх атрибути та методи, а також зв'язки між ними. Діаграма класів дозволяє відобразити структуру системи, її компоненти, їх взаємозв'язки та ієрархію.

Діаграма класів соціальної мережі для працевлаштування зображена на рисунку 7.

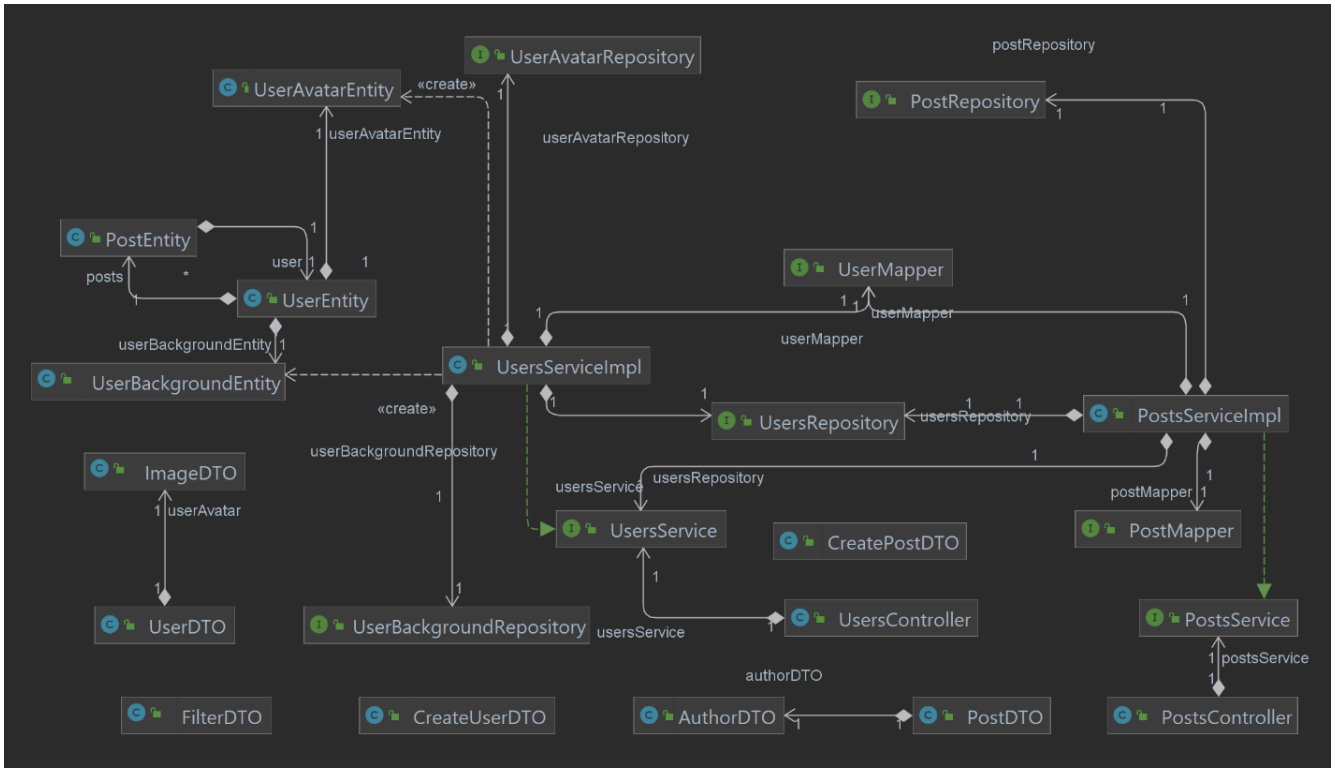

Рисунок 7 – Діаграма класів соціальної мережі

РОЗДІЛ 3. АРХІТЕКТУРНІ ШАБЛОНИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура клієнт-сервер

Розвиток програмних засобів для проектування та розробки різноманітних ресурсів, включаючи веб-додатки з користувацьким інтерфейсом, призвів до вдосконалення підходів до створення систем. Веб-додаток - це інтерактивний ресурс, що надає користувачу можливість керувати даними та вирішувати задачі в рамках конкретного ресурсу. На відміну від звичайних веб-сайтів, що переважно призначені для представлення інформації, веб-сервіси є більш ресурсоємними. Веб-сервіс - це клієнт-серверний додаток, що міститься на віддаленому сервері та виконується там, а його візуальна частина - веб-інтерфейс, запускається та відображається у браузері (рисунок 8). Для створення веб-ресурсу потрібно розробити архітектуру веб-додатку, протоколи зв'язку та сховища даних. Сучасні веб-застосунки є розподіленими системами.

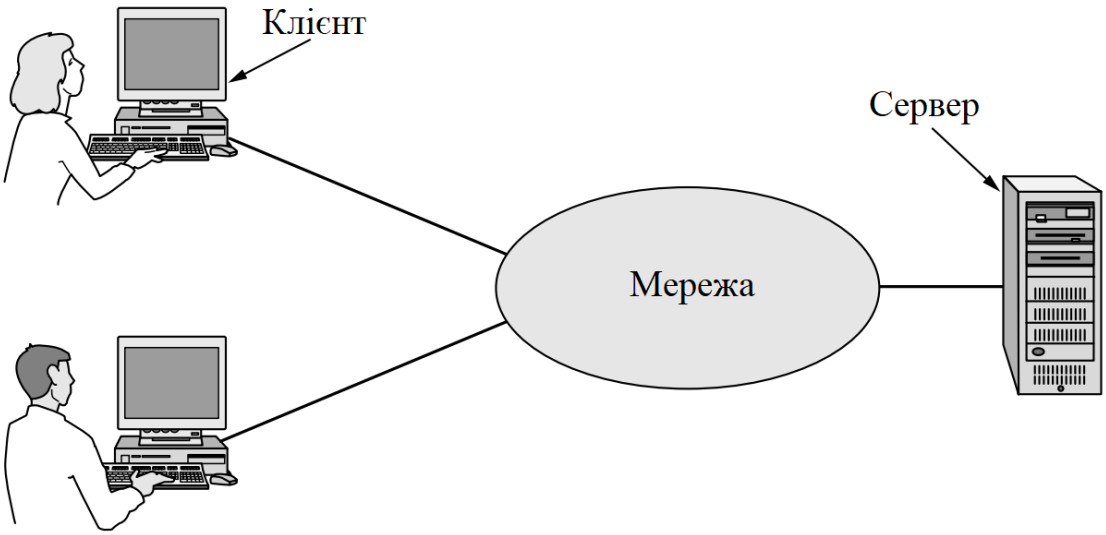

Рисунок 8 – Взаємодія клієнтів з сервером у мережі

Розподілена система - це система, в якій компоненти розташовані на різних вузлах та виконують різні функції залежно від своїх можливостей. Веб-додатки мають дві складові: клієнтську та серверну. Клієнт робить запит до серверної частини та отримує відповідь. Серверна частина обробляє запит, формує відповідь та надсилає її клієнту у вигляді графічної форми, такої як HTML або XML. Модель клієнт-сервер має два окремі процеси, що працюють на клієнтській та серверній машині (рисунок 9).

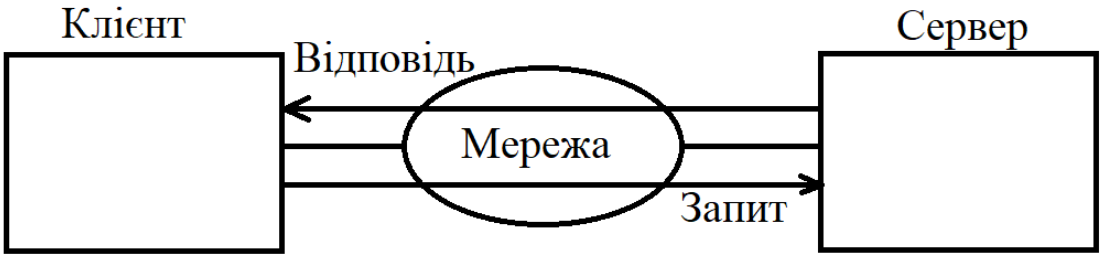

Рисунок 9 – Запит та відповідь між клієнтом та сервером

Клієнтська частина - це користувацький інтерфейс, тоді як серверна частина - це центральний вузол, який зберігає, оброблює та видає дані. Один сервер може обробляти сотні або тисячі клієнтів одночасно. Комунікація між клієнтом та сервером відбувається через мережу за допомогою HTTP-протоколу. Використання такої архітектури дозволяє розділити зони відповідальності між двома підсистемами та зробити їх більш незалежними. Крім того, зміна серверної логіки не впливає на клієнтів, поки дотримується встановлений API. Відповідно використання клієнт-серверної архітектури у веб-додатках передбачає використання браузера як клієнта, що відправляє запити (HTTP-request) та отримує відповіді (HTTP-response) за допомогою протоколу HTTP [4].

Існує декілька типів веб-клієнтів, які можна виділити залежно від наявних характеристик. Перш за все, можна розділити їх за використанням протоколу передачі даних, такими як HTTP або WAP. Крім того, їх можна класифікувати за типом використовуваного додатку - веб-браузер або будь-яка інша клієнтська програма, за типом рендерингу даних і за обробкою логіки застосунків. Однак,

особливу увагу слід звернути на підхід до рендерингу даних. Рендеринг може відбуватися на стороні клієнта або на стороні сервера, і ця характеристика часто формулюється як "товстий клієнт - тонкий сервер" або "тонкий клієнт - товстий сервер".

Роль клієнта та сервера може бути розподілена по-різному залежно від концепції додатку, використовуваного стеку технологій, мов програмування та фреймворків. Тому виникає питання розподілення ролей між компонентами веб-застосунку у контексті клієнт-серверної архітектури. Підхід "товстий клієнт - тонкий сервер" орієнтується на перенесення рендерингу даних та обробки бізнес-логіки на більшу міру на сторону клієнта. Сучасні інтерпретовані мови програмування, зокрема JavaScript та її фреймворки, здатні виконувати більшу частину бізнес-логіки безпосередньо у браузері. Отже, питання розподілення ролей між компонентами веб-застосунку набуває особливої важливості в контексті клієнт-серверної архітектури.

JavaScript, що виконується в браузері користувача, відповідає за запит даних з сервера та взаємодію з веб-сторінкою при рендерингу на стороні клієнта. Наприклад, якщо користувач введе неприпустимі дані у форму, на стороні клієнта достатньо буде мати код, що оновить сторінку з повідомленням про помилку, без необхідності створювати нові запити на сервер та перезавантажувати сторінку.

На відміну від «товстого клієнта – тонкого сервера», підхід «тонкий клієнт – товстий сервер» є класичним методом розробки веб-сервісів. Цей підхід передбачає, що логіка веб-додатку, обробка даних та інші операції здійснюються на сервері, що має значний обсяг ресурсів. При рендерингу сторінок звертання до сервера здійснюється безпосередньо з клієнта. Це означає, що для кожної веб-сторінки або після взаємодії користувача з компонентами, клієнт створює запит на сервер. Якщо користувач введе невірні дані в форму на веб-ресурсі, код на стороні клієнта звернеться до сервера для отримання нової сторінки.

Поняття **дворівневої архітектури** відображає підхід "товстий клієнт-тонкий сервер". Будь-яка інформаційна система повинна мати як мінімум три базові функціональні частини: модулі для зберігання даних; модулі для обробки даних;

інтерфейс для представлення даних користувачу. Кожна з цих функціональних частин має бути реалізована незалежно одна від одної. Система повинна мати можливість змінювати користувацький інтерфейс та відображення таблиць даних, не змінюючи середовища для зберігання та обробки даних.

Отже, двошарова архітектура дозволяє зберігати дані та їх обробку в окремих модулях, що робить систему гнучкою і забезпечує можливість змінювати користувацький інтерфейс або платформу зберігання даних без втручання в самі дані. Проте, як і у будь-якій іншій архітектурі, є і певні недоліки та переваги.

Архітектура з простотою та мінімалізмом надає розробникам та користувачам необхідний функціонал. Розподіл даних та можливість одночасної обробки даних забезпечують безпеку та цілісність даних. Проте, визначення ролі проміжного вузла між сервером та клієнтом під назвою «обробка даних» може бути складним, а потужність у підтримці бізнес-логіки може бути обмеженою. Крім того, оновлення клієнтської сторони може створити додаткову складність та перевантажити сервіс, на який застосовна така архітектура.

У **трирівневій архітектурі** застосовується підхід "тонкий клієнт - товстий сервер", який полягає у тому, що клієнт не виконує обробку даних, а лише відображає інформацію, отриману з сервера додатків. Для цього можна використовувати стандартний стек веб-технологій, такий як браузер, CGI і Java. Такий підхід дозволяє зменшити обсяг передаваних даних, що дозволяє підключати клієнтські комп'ютери навіть через повільні лінії зв'язку, а також реалізувати дуже прості клієнтські інтерфейси. Крім того, трирівнева архітектура дозволяє більш точно визначати ролі користувачів і забезпечує високий рівень захищеності системи від навмисного нападу та помилкових дій персоналу, оскільки користувачі отримують доступ до певних функцій сервера додатків, а не безпосередньо до бази даних.

3.2 REST архітектура

REST (REpresentational State Transfer) - це стиль архітектури для розподілених гіпермедійних систем, що базується на принципах та обмеженнях, визначених Роєм Філдінгом у його дисертації, яку він представив у 2000 році. Для того, щоб інтерфейс сервісу відповідав REST, необхідно дотримуватися цих принципів. REST API - це веб-сервіс, що відповідає архітектурному стилю REST.

Принципи REST наведено в таблиці 1.

Таблиця 1 – Принципи REST

Назва принципу	Опис
Уніфікований інтерфейс	Шляхом застосування принципу узагальненості до інтерфейсу компонентів можна спростити загальну архітектуру системи та покращити зрозумілість взаємодій. Для отримання уніфікованого інтерфейсу та керування поведінкою компонентів існують кілька архітектурних обмежень. Для досягнення REST-інтерфейсу необхідно дотримуватися наступних чотирьох обмежень: • Ідентифікація ресурсів - це вимога до того, щоб інтерфейс однозначно ідентифікував кожен ресурс, який взаємодіє між клієнтом та сервером. • Маніпулювання ресурсами через представлення - є обмеженням, за яким ресурси мають мати єдині представлення у відповіді сервера, які споживачі API використовують для зміни стану ресурсів на сервері. • Самоописні повідомлення - це вимога до того, щоб кожне представлення ресурсу містило достатньо інформації, щоб описати, як обробити повідомлення, а також повинно надавати інформацію про додаткові дії, які клієнт може виконати над ресурсом.

	• Гіпермедіа як рушій стану додатку - це обмеження, за яким клієнт повинен мати лише початковий URI додатку, а всі інші ресурси та взаємодії повинні бути керовані динамічно за допомогою гіперпосилань.
Клієнт-сервер	Клієнт-серверний патерн проектування забезпечує розділення завдань, що допомагає клієнтським і серверним компонентам розвиватися незалежно. Відокремлюючи проблеми користувацького інтерфейсу (клієнт) від проблем зберігання даних (сервер), ми покращуємо переносимість користувацького інтерфейсу на різні платформи та покращуємо масштабованість за рахунок спрощення серверних компонентів. У той час як клієнт і сервер розвиваються, ми повинні переконатися, що інтерфейс між клієнтом і сервером не розривається.
Без збереження внутрішнього стану	У такій архітектурі, кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для виконання запиту, тому сервер не може використовувати контекстну інформацію, яка була збережена раніше на сервері. Це означає, що клієнтська програма повинна повністю зберігати стан сеансу, щоб забезпечити виконання запиту належним чином.
Кешабельність	Обмеження щодо кешування вимагає, щоб відповідь чітко або приховано позначалась як кешабельна або некешабельна. У випадку, коли відповідь можна кешувати, клієнт має можливість повторно використовувати дані відповіді в майбутньому для подібних запитів на певний проміжок часу.
Багаторівнева система	Стиль багатошарової системи надає можливість архітектурі мати ієрархічні рівні, обмежуючи функціонування компонентів.

	Наприклад, у такій багатошаровій системі кожен компонент може взаємодіяти лише з рівнем, на якому він розташований, і не може сприймати рівні, що знаходяться далі.
Код на вимогу (необов'язково)	REST також надає можливість розширити функціонал клієнта за допомогою завантаження та запуску коду у вигляді аплетів чи скриптів. Цей завантажений код полегшує роботу клієнтів, знижуючи кількість функцій, які мають бути реалізовані на початку. Сервери можуть передавати деякі функції клієнту у формі коду, який клієнт має лише виконати.

Основною абстракцією інформації в REST є ресурс. Це може бути будь-яка інформація, яку можна ідентифікувати. Наприклад, документ, зображення, тимчасовий сервіс, колекція ресурсів або навіть неелектронний об'єкт (такий як людина) можуть бути ресурсами REST. Стан ресурсу в даний момент часу називається його представленням.

Представлення ресурсу складається з:

- даних;
- метаданих, які характеризують дані;
- гіпермедійних посилань, які допомагають клієнтам перейти до наступного потрібного стану.

REST API включає в себе набір пов'язаних ресурсів. Ця сукупність ресурсів називається ресурсною моделлю REST API.

REST використовує ідентифікатори ресурсів для розпізнавання кожного ресурсу, задіяного у взаємодії між клієнтськими та серверними елементами.

Формат даних для представлення називається типом медіа. Тип медіа вказує на специфікацію, яка визначає обробку представлення.

RESTful API має схожість з гіпертекстом. Кожен адресований елемент інформації має адресу, яку можна визначити явно (наприклад, через атрибути

посилань та ідентифікатори) або неявно (отриману від типу медіа та структури представлення).

«Гіпертекст (або гіпермедіа) передбачає одночасне відображення інформації та контролю, завдяки чому інформація стає доступною для користувача (або автомату) для вибору та визначення дій.

Важливо зазначити, що гіпертекст не обов'язково повинен бути HTML (або XML чи JSON) у браузері. Машини можуть переходити за посиланнями, якщо вони розуміють формат даних і види зв'язків.» - Рой Філдінг.

Більше того, представлення ресурсів мають бути самодокументованими: клієнт не повинен знати, чи є ресурс співробітником або пристроєм. Він має діяти, базуючись на типі медіа, асоційованому з ресурсом.

Таким чином, на практиці ми розробляємо багато специфічних типів медіа - зазвичай один тип медіа відповідає одному ресурсу.

Кожен тип медіа встановлює модель обробки за замовчуванням. Наприклад, HTML визначає процес відображення гіпертексту та поведінку браузера для кожного елемента.

Типи медіа не мають прямого стосунку до методів ресурсу, таких як GET/PUT/POST/DELETE/..., окрім ситуацій, коли деякі компоненти типу медіа встановлюють модель обробки, яка має такий вигляд: "елементи зв'язку з атрибутом href створюють гіпертекстове посилання, яке при активації ініціює запит на отримання (GET) на URI, відповідно до значення атрибуту href, закодованого в CDATA".

Ще одним важливим аспектом, пов'язаним з REST, є методи ресурсів. Ці методи використовуються для здійснення бажаного переходу між різними станами ресурсів. Багато людей неправильно вважають методи ресурсів синонімами HTTP-методів (тобто, GET/PUT/POST/DELETE). Рой Філдінг ніколи не надавав конкретних рекомендацій щодо використання певного методу для певного стану. Він лише підкреслював, що інтерфейс має бути уніфікованим.

Для використання REST API ми повинні розпочати з початкового URI (закладки) та набору стандартних типів медіа, придатних для цільової аудиторії

(тобто, як очікується, зрозумілих для будь-якого клієнта, який може використовувати API).

Починаючи з цього моменту, усі переходи між станами програми мають бути визначені вибором клієнта серед опцій, наданих сервером у отриманих представленнях, або передбаченими діями користувача з цими представленнями.

Переходи можуть бути визначені (або обмежені) знаннями клієнта про типи медіа та механізми взаємодії з ресурсами. Обидва аспекти можуть вдосконалюватися "на льоту" (наприклад, завантаження коду за запитом). Необхідно уникати ситуацій, коли взаємодію керують зовнішні дані, а не гіпертекст.

Чимало людей мають схильність порівнювати HTTP з REST, але варто пам'ятати, що вони - різні речі.

REST ≠ HTTP.

Хоча метою REST також є зробити веб-простір більш організованим та стандартним, Рой Філдінг пропагує строге дотримання принципів REST. Саме через це люди намагаються порівняти REST з інтернетом.

У своїй дисертації Рой Філдінг не згадує жодного конкретного напрямку реалізації, у тому числі не акцентує увагу на протоколах або HTTP. Головне, щоб дотримуватися шести основних принципів REST, і тоді можна вважати наш інтерфейс RESTful.

Кажучи простими словами, в архітектурі REST дані та функціональні можливості вважаються ресурсами, якими можна керувати за допомогою уніфікованих ідентифікаторів ресурсів (URI).

Операції над ресурсами здійснюються через прості та чітко визначені дії. Також ресурси мають бути відокремлені від їхніх представлень, що дозволяє клієнтам отримувати контент у різних форматах, таких як HTML, XML, текст, PDF, JPEG, JSON тощо.

Клієнти та сервери обмінюються представленнями ресурсів через стандартний інтерфейс та протокол. Зазвичай використовується HTTP, але REST не наполягає на цьому.

Метадані про ресурс доступні та використовуються для керування кешуванням, виявлення помилок передачі, підбору відповідного формату представлення та автентифікації або регулювання доступу.

І найважливіше - будь-яка взаємодія з сервером має бути безстатусною.

Усі ці принципи сприяють тому, щоб RESTful застосунки були простими, легкими та швидкими [5].

3.3 Сервісно-орієнтована архітектура

SOA, або сервісно-орієнтована архітектура, встановлює методологію перетворення програмних компонентів на багаторазово використовувані та взаємодіючі сервісні інтерфейси. Використовуючи загальні стандарти інтерфейсу та архітектурні шаблони, сервіси можуть легко інтегруватися в нові програми, звільняючи розробників від необхідності переписувати або дублювати існуючу функціональність або забезпечувати сумісність з наявними можливостями.

Сервісно-орієнтована програма – це програма, яка в основному складається зі служб, які часто знаходяться в ієрархії.

Кожен сервіс в SOA містить код та дані, потрібні для виконання окремої бізнес-функції (наприклад, перевірки кредитної здатності клієнта, розрахунку щомісячного платежу за кредитом або обробки іпотечного клопотання). Сервісні інтерфейси надають слабкий зв'язок, дозволяючи їх викликати без докладного знання реалізації, зменшуючи залежність між додатками.

Цей інтерфейс є договором про надання послуг між провайдером та споживачем послуг. Базові додатки сервісного інтерфейсу можуть бути створені на Java, Microsoft .Net, Cobol або будь-якій іншій мові програмування, поставляти упаковані програмні додатки від постачальника (наприклад, SAP), SaaS-додаткі (наприклад, Salesforce CRM) або отримані як додатки з відкритим вихідним кодом. Сервісні інтерфейси зазвичай визначаються за допомогою веб-сервісів мови визначення (WSDL), які є стандартними тегами на основі XML.

Сервіси надаються через стандартні мережеві протоколи, такі як SOAP (простий протокол доступу до об'єктів)/HTTP або Restful HTTP (JSON/HTTP), що дозволяють надсилати запити для читання або зміни даних. Управління сервісами відповідає за життєвий цикл розробки, і на відповідному етапі сервіси реєструються у реєстрі, що дозволяє розробникам легко знаходити та повторно використовувати їх для створення нових додатків чи бізнес-процесів.

Ці сервіси можуть бути створені з нуля, але часто розробники використовують функції старих систем обліку як сервісні інтерфейси.

Таким чином, SOA є значним кроком у розвитку розробки та інтеграції додатків протягом останніх десятиліть. До появи SOA у кінці 1990-х років, забезпечення доступу до даних або функцій, розташованих в інших системах, вимагало складної "точка-точка" інтеграції, яку розробники доводилося повторювати, частково або повністю, для кожного нового проекту розробки. Застосування цих можливостей через сервіси SOA дозволило розробникам просто повторно використати існуючі функції та підключатися через архітектуру SOA ESB (Enterprise Service Bus).

ESB, або корпоративна сервісна шина, представляє собою архітектурний патерн, який використовує централізований програмний компонент для інтеграції додатків. Він забезпечує перетворення моделей даних, обробку підключень/обміну повідомленнями, маршрутизацію, перетворення протоколів комунікації та можливе управління композицією кількох запитів. ESB може надавати ці інтеграції та перетворення у вигляді сервісного інтерфейсу для повторного використання новими додатками. Зазвичай, патерн ESB реалізується за допомогою спеціалізованого середовища інтеграції та набору інструментів для оптимальної продуктивності.

SOA може бути втілена без ESB, але такий варіант буде схожий на простий набір сервісів. У цьому випадку, кожному власнику програми доведеться безпосередньо підключатися до потрібного сервісу та виконувати відповідні перетворення даних для відповідності різним сервісним інтерфейсам. Це створює значну роботу (навіть якщо інтерфейси повторно використовуються) і може

призвести до проблем з підтримкою в майбутньому, оскільки кожне з'єднання є точковим. Фактично, згодом ESB стали настільки важливою частиною будь-якої реалізації SOA, що ці два поняття іноді використовуються як взаємозамінні, що призводить до плутанини.

У порівнянні з попередніми архітектурами, SOA пропонує важливі переваги для підприємства:

- Підвищена гнучкість бізнесу та скорочений час виведення на ринок: Повторне використання є ключовим елементом. Збірка додатків з повторно використовуваних сервісів, тобто з основних компонентів, замість переписування та реінтеграції при кожному новому проекті розробки, дозволяє розробникам створювати додатки набагато швидше у відповідь на нові бізнес-можливості. Сервіс-орієнтований підхід підтримує сценарії інтеграції додатків, інтеграції даних та автоматизації бізнес-процесів або робочих процесів у стилі оркестрування сервісів. Це сприяє прискоренню проектування та розробки програмного забезпечення, дозволяючи розробникам витрачати менше часу на інтеграцію та більше часу на створення та поліпшення своїх додатків.
- Здатність адаптувати застарілі функції для нових ринків: Вдало розроблена SOA дозволяє розробникам легко брати функціональність, "закріплену" в одному обчислювальному середовищі або платформі, та розповсюджувати її на нові середовища та ринки. Наприклад, багато компаній використовують SOA для перенесення функцій фінансових систем, які базуються на мейнфреймах, у нові веб-додатки, надаючи своїм клієнтам можливість самостійного доступу до процесів та інформації, які раніше були доступні тільки через безпосередній контакт з праці вниками компанії або бізнес-партнерами.
- Покращення взаємодії між бізнесом та IT: В рамках SOA, послуги можуть бути визначені у бізнес-термінології (наприклад, "створити

страховий план" або "рахувати прибутковість основних засобів"). Це сприяє більш ефективній співпраці між бізнес-аналітиками та розробниками у вирішенні важливих питань, таких як обсяг бізнес-процесів, визначених через послуги, або бізнес-наслідки змін у процесах, що можуть призвести до кращих результатів. Таким чином, SOA допомагає забезпечити краще розуміння між бізнесом та IT, сприяючи ефективнішому виконанню проектів та досягненню загальних цілей підприємства.

Експерти створили кілька тисяч друкованих та електронних сторінок, аналізуючи SOA та мікросервіси, та вивчаючи тонкощі їхніх взаємовідносин. В контексті даної статті основні розбіжності між ними стосуються комбінації компонентів та області застосування:

- SOA - це інтеграційний архітектурний підхід та концепція, які охоплюють весь бізнес. Він дозволяє представляти існуючі програми за допомогою слабко зв'язаних інтерфейсів, кожен з яких відповідає окремій бізнес-функції, що забезпечує повторне використання функціоналу в різних програмах підприємства.
- Мікросервісна архітектура - це стиль архітектури програм і концепція, спрямована на додатки. Вона дає змогу розділяти внутрішні компоненти однієї програми на маленькі частини, які можна окремо модифікувати, масштабувати та керувати. Вона не встановлює способи взаємодії між програмами - для цього ми повертаємося до корпоративного рівня сервісних інтерфейсів, які надає SOA.

Архітектура мікросервісів почала розвиватися та збільшуватися завдяки віртуалізації, хмарним обчисленням, гнучким підходам до розробки та DevOps. В основному, переваги мікросервісів у цих контекстах випливають з розділення компонентів, що спрощує та покращує таке:

- Спритність та ефективність розробників: Мікросервіси дозволяють розробникам втілювати нові технології в частину програм и, не

впливаючи на решту програми. Кожен компонент може бути змінений, протестований та розгорнутий незалежно від інших, що прискорює цикли розробки.

- Масштабованість: Мікросервіси дозволяють ефективно використовувати переваги хмарної масштабованості - будь-який компонент може бути масштабований окремо від інших для швидкого адаптування до вимог робочого навантаження та оптимального використання обчислювальних ресурсів.
- Стійкість: Завдяки розділенню компонентів, збій одного мікросервісу не впливає на роботу інших. Також кожен мікросервіс може працювати згідно з власними вимогами до доступності, не обмежуючи інші компоненти або весь додаток загальними вимогами до доступності.

Так само, як мікросервісна архітектура може покращити гнучкість, масштабованість та стійкість дизайну додатків, ці ж методи можуть бути використані для інтеграції. Це має важливе значення, оскільки з часом сильно централізована модель ESB і пов'язана з нею централізована команда фахівців з інтеграції можуть стати слабким місцем. Застосовуючи принципи мікросервісів, ми можемо потенційно розбити ESB на менші, децентралізовані інтеграційні компоненти. Це одна з основних передумов для гнучкої інтеграції [6].

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ

4.1 Мова програмування Java

Розробка соціальної мережі представляє собою frontend та backend, тобто клієнтську та серверну частину проєкту. Оскільки клієнтська частина не може повноцінно функціонувати без серверної, першими кроками була реалізація backend на Java 17, Spring Framework, що покликало за собою дослідження можливостей мови програмування Java та фреймворку Spring.

Java має значні переваги над іншими мовами та середовищами, що робить її придатною для виконання практично будь-яких завдань програмування. Основні переваги Java в порівнянні з іншими мовами та середовищами програмування можуть бути сформульовані так:

- Легкість вивчення: Java була розроблена з метою надання простоти використання. Це дозволяє легко писати, компілювати, налагоджувати та вивчати цю мову програмування.
- Об'єктно-орієнтованість: Java є об'єктно-орієнтованою мовою програмування. Це дозволяє створювати модульні програми та код для багаторазового використання, що дозволяє зменшити витрати на розробку програмного забезпечення.
- Платформонезалежність: Однією з найважливіших переваг Java є її здатність легко переноситися з однієї комп'ютерної системи на іншу. Це означає, що програми, написані на Java, можуть працювати на різних операційних системах, що дозволяє вирішити проблему кросплатформеності.
- Надійність та безпека: Java має вбудовану функцію безпеки, що дозволяє запобігати вразливостям безпеки програм. Це робить Java однією з найбільш надійних мов програмування для створення рішень для Інтернету по всьому світу.

Алан Кей підсумував п'ять основних характеристик Smalltalk, першої успішної об'єктно-орієнтованої мови і однієї з мов, на яких базується Java. Ці характеристики представляють чистий підхід до об'єктно-орієнтованого програмування:

1. Все є об'єктом. Уявіть собі об'єкт як химерну змінну; він зберігає дані, але ви можете "робити запити" до цього об'єкта, просячи його виконати операції над самим собою. Теоретично, ви можете взяти будь-який концептуальний компонент проблеми, яку ви намагаєтеся вирішити (собаки, будівлі, послуги тощо), і представити його як об'єкт у вашій програмі.

2. Програма - це сукупність об'єктів, які повідомляють один одному, що робити, за допомогою надсилаючи повідомлення. Щоб зробити запит до об'єкта, ви "надсилаєте повідомлення" цьому об'єкту. Більш конкретно, ви можете думати про повідомлення як про запит на виклик методу, який належить певному об'єкту.

3. Кожен об'єкт має власну пам'ять, яка складається з інших об'єктів. Інакше кажучи, ви створюєте Інакше кажучи, ви створюєте новий тип об'єкта, створюючи пакет, що містить вже існуючі об'єкти. Таким чином, ви можете вбудовувати складність у програму, приховуючи її за простотою об'єктів.

4. Кожен об'єкт має тип. Говорячи мовою програмування, кожен об'єкт є екземпляром класу, де "клас" є синонімом "типу". Найважливішою відмінною характеристикою класу є характеристикою класу є "Які повідомлення ви можете йому надсилати?".

5. Всі об'єкти певного типу можуть отримувати однакові повідомлення. Це фактично є навантажувальним твердженням, як ви побачите пізніше. Оскільки об'єкт типу "коло" є також є об'єктом типу "фігура", коло гарантовано прийматиме повідомлення про фігури. Це означає, що ви можете писати код, який розмовляє з фігурами, і автоматично обробляти все, що відповідає опису фігури. Така заміннність є однією з найпотужніших концепцій ООП [7].

Java ніколи не була просто мовою. Існує багато мов програмування, але лише деякі з них стають популярними. Java - це ціла платформа з величезною бібліотекою, що містить багато коду, який можна використовувати повторно, і

середовищем виконання, яке надає такі послуги такі як безпека, переносимість на різні операційні системи та автоматичне збирання сміття.

Як програмісту, вам потрібна мова з приємним синтаксисом і зрозумілою семантикою. Java підходить для цього, як і десятки інших чудових мов. Деякі мови дають вам портативність, збирання сміття і тому подібне, але вони не мають великої бібліотеки, що змушує вас створювати власні, якщо вам потрібна вигадлива графіка, мережева робота або доступ до баз даних. Що ж, у Java є все - гарна мова, якісне середовище виконання та велика бібліотека. Ця комбінація робить Java непереборною пропозицією для багатьох програмістів [8].

4.2 Spring Framework як інструмент розробки

Проєкт соціальної мережі створюється з допомогою Spring Framework. Spring є одним з найпопулярніших фреймворків для розробки додатків на Java. Розробники з усього світу використовують Spring для створення надійних та якісних додатків. Spring був розроблений Родом Джонсоном і з того часу став альтернативною технологією у світі Java для моделі EJB.

З використанням фреймворку Spring можна створювати різні типи додатків, включаючи веб-додатки та додатки, що взаємодіють з базою даних. Spring надає розробникам різноманітні можливості та функції, що допомагають збільшити продуктивність та підвищити якість додатків. Тому Spring є популярним вибором серед розробників Java.

Неможливо зрозуміти, що таке Spring Framework, не розуміючи, що таке впровадження залежностей і інверсія керування . Ін’єкція залежностей, яка також називається DI, є одним із типів інверсії управління (IoC).

Інверсія управління - це принцип об'єктно-орієнтованого програмування, в якому об'єкти програми не залежать від конкретних реалізацій інших об'єктів, але можуть мати знання про свої абстракції (інтерфейси) для подальшої взаємодії.

Ін'єкція залежностей - це композиція структурних шаблонів проектування, в якій для кожної функції програми існує один, умовно незалежний об'єкт (сервіс), який може мати потребу використовувати інші об'єкти (залежності), відомі йому через інтерфейси. Залежності передаються (впроваджуються) в сервіс під час його створення. Це ситуація, коли ми вводимо елемент одного класу в інший. На практиці DI реалізується шляхом передачі параметрів у конструктор або за допомогою сеттерів. Бібліотеки, які реалізують цей підхід, також називають контейнерами IoC. Аспектно-орієнтоване програмування - парадигма програмування, яка дозволяє виділити наскрізну функціональність у додатку. Ці функції, які охоплюють кілька вузлів програми, називаються наскрізними проблемами, і вони відокремлені від безпосередньої бізнес-логіки програми. В ООП ключовою одиницею є клас, тоді як в АОП ключовим елементом є аспект. DI допомагає розділити класи додатків на окремі модулі, а AOP допомагає відокремити наскрізні проблеми від об’єктів, на які вони впливають.

Доступ до даних Spring Framework. Контейнер доступу до даних/інтеграції складається з JDBC, ORM, OXM, JMS і модуля Transactions.

- JDBC надає абстрактний рівень JDBC і позбавляє розробника необхідності вручну реєструвати одноманітний код, пов’язаний із підключенням до бази даних.
- Spring ORM забезпечує інтеграцію з популярними ORM, такими як Hibernate, JDO, які є реалізаціями JPA.
- Модуль OXM відповідає за зв'язування Object / XML - XMLBeans, JAXB тощо.
- Модуль JMS (Java Messaging Service) відповідає за створення, надсилання та отримання повідомлень.
- Transactions підтримує керування транзакціями для класів, які реалізують певні методи та POJO.

Spring також включає ряд інших важливих модулів, таких як AOP, Aspects, Instrumentation, Messaging і Test.

AOP реалізує аспектно-орієнтоване програмування і дозволяє використовувати весь арсенал можливостей АОП. Модуль Aspects забезпечує інтеграцію з AspectJ, який також є потужною структурою AOP. Instrumentation відповідає за підтримку інструментів класу та завантажувача класів, які використовуються в серверних програмах. Модуль обміну повідомленнями забезпечує підтримку STOMP. Нарешті, модуль Test забезпечує тестування за допомогою TestNG або JUnit Framework.

4.3 Використання Docker та СУБД PostgreSQL

Docker - це платформа з відкритим вихідним кодом для створення, розгортання та управління додатками. Використовуючи Docker, можна легко відокремити додатки від інфраструктури та швидко доставляти програмне забезпечення. Docker дозволяє керувати вашою інфраструктурою на рівні, схожому на керування додатками. Використовуючи Docker, можна ефективно використовувати методології швидкої доставки, тестування та розгортання коду, що дозволяє зменшити затримки між написанням коду та запуском його в реальному середовищі [9].

Одним із найбільш важливих етапів у життєвому циклі розробки програмного забезпечення (SDLC - Software Development Life Cycle) є обробка даних, яка включає дії, такі як об'єднання, очищення та фільтрація даних. Для реалізації цих операцій розробники зазвичай використовують програми керування базами даних, такі як MySQL або PostgreSQL, і працюють з кількома тимчасовими програмами, які використовуються на різних етапах SDLC.

За допомогою Docker, як платформи контейнеризації, розробники можуть запускати різні програми та процеси, не потрібно встановлювати та запускати їх в різних середовищах. Це дозволяє розробникам використовувати будь-яку СУБД, наприклад PostgreSQL, для виконання операцій обробки даних, завантажуючи відповідні Docker-образи з Docker Hub [10].

У ході розробки даного проєкту було вирішено використовувати Docker-образ СУБД PostgreSQL для зберігання та використання даних соціальної мережі.

PostgreSQL є відкритою об'єктно-реляційною системою управління базами даних, яка використовує мову SQL разом із широким спектром функцій, що дозволяють ефективно зберігати та масштабувати навантаження даних. Система базується на проекті POSTGRES, який був розроблений в Каліфорнійському університеті в Берклі в 1986 році, та пройшла понад 35 років активного розвитку.

PostgreSQL відзначається високою надійністю, цілісністю даних, широким спектром функцій та можливостей для розширення. Ця СУБД підтримує всі основні операційні системи та є ACID-сумісною з 2001 року. Для збільшення потужності PostgreSQL, доступні доповнення, такі як PostGIS - популярний розширювач геопросторових баз даних. PostgreSQL має віддану спільноту розробників з відкритим вихідним кодом, яка постійно працює над створенням інноваційних та ефективних рішень. З цією репутацією, не дивно, що PostgreSQL є першим вибором багатьох людей та організацій, які шукають реляційну СУБД з відкритим вихідним кодом.

Нижче наведено вичерпний перелік різноманітних можливостей, які можна знайти в PostgreSQL, і які додаються з кожним новим випуском [11]:

Типи даних

- Примітиви: Цілі, числові, рядкові, логічні
- Структуровані: дата/час, масив, діапазон/мультидіапазон, UUID
- Документні: JSON/JSONB, XML, Key-value (Hstore)
- Геометрія: точка, лінія, коло, багатокутник
- Кастомізовані типи

Цілісність даних

- УНІКАЛЬНІ, НЕ НУЛЬОВІ
- Первинні ключі
- Зовнішні ключі
- Обмеження виключення

- Явні блокування, рекомендаційні блокування

Паралелізм, продуктивність

- Індексування: B-дерево, багатостовпчикові, вирази, часткове
- Розширене індексування: GiST, SP-Gist, KNN Gist, GIN, BRIN, індекси покриття, фільтри Блума
- Складний планувальник/оптимізатор запитів, сканування лише за індексами, багатоколонкова статистика
- Транзакції, вкладені транзакції (через точки збереження)
- Багатоверсійний контроль паралелізму (MVCC)
- Розпаралелювання запитів на читання та побудова індексів B-дерева
- Розбиття таблиць на розділи
- Всі рівні ізоляції транзакцій, визначені в стандарті SQL, включаючи Serializable
- Компіляція виразів Just-in-time (JIT)

Надійність, аварійне відновлення

- Журналювання з випередженням запису (Write-ahead Logging, WAL)
- Реплікація: Асинхронна, Синхронна, Логічна
- Відновлення в момент часу (PITR), активні резерви
- Табличні простори

Безпека

- Аутентифікація: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, сертифікат та інші
- Надійна система контролю доступу
- Безпека на рівні стовпців і рядків
- Багатофакторна автентифікація за допомогою сертифікатів та додаткового методу

Розширюваність

- Збережені функції та процедури

- Процедурні мови: PL/pgSQL, Perl, Python та Tcl. Існують інші мови, доступні через розширення, наприклад, Java, JavaScript (V8), R, Lua та Rust
- Вирази шляхів SQL/JSON
- Зовнішні обгортки даних: підключайтеся до інших баз даних або потоків за допомогою стандартного інтерфейсу SQL
- Налаштовуваний інтерфейс зберігання таблиць
- Багато розширень, які забезпечують додаткову функціональність, включаючи PostGIS

Інтернаціоналізація, текстовий пошук

- Підтримка міжнародних наборів символів, наприклад, через зіставлення ICU
- Нечутливі до регістру та наголосу зіставлення
- Повнотекстовий пошук

Розглянемо, як було встановлено, налаштовано та запущено PostgreSQL на Docker.

Перед тим, як почати процес налаштування програми для Docker PostgreSQL Environment, потрібно завантажити та встановити Docker на комп'ютері. Можна завантажити програму з офіційного сайту Docker - https://www.docker.com/, натиснувши на опцію "Почати роботу" на сторінці привітання веб-сайту Docker. На сторінці завантаження потрібно вибрати версію Docker відповідно до специфікацій операційної системи і завантажити файл встановлення. Після завантаження файлу можна встановити Docker на комп'ютері, дотримуючись інструкцій з встановлення. Далі, можна увійти в Docker Hub, щоб отримати доступ до файлів Docker Image, які можна використовувати для запуску зовнішніх програм, таких як PostgreSQL.

Під час розробки даного проєкту використовується 64-бітна операційна система Windows 10 Pro, саме тому було завантажено та встановлено найновішу версію Docker для Windows.

Далі було завантажено образ Docker PostgreSQL та запущено контейнер за допомогою кількох дій у командному рядку, приклад показано на рисунку 10.

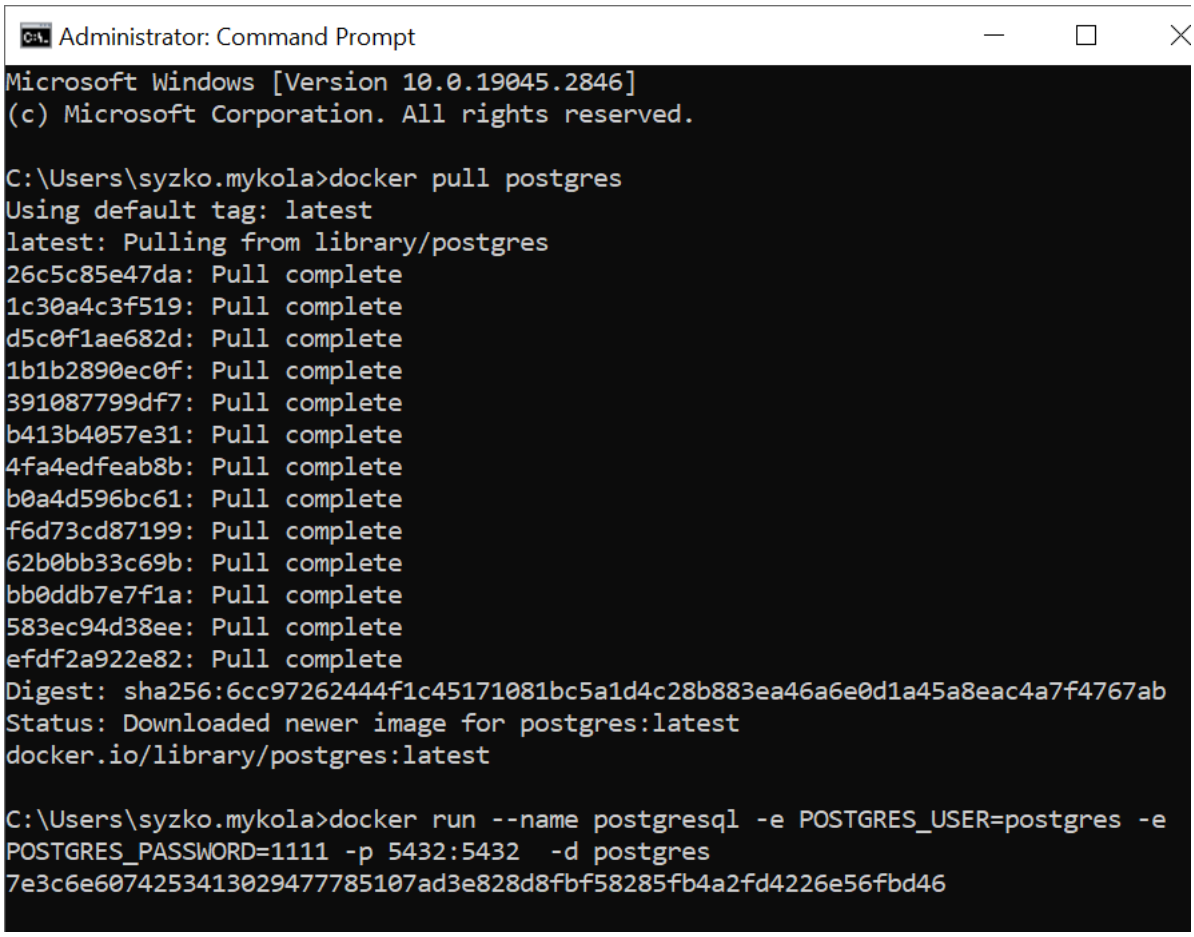

Рисунок 10 – Запуск PostgreSQL контейнера у Docker

Подальші дії з контейнером будуть виконуватись з допомогою Docker Desktop – показано на рисунку 11.

Для підключення до бази даних використовується DBeaver. DBeaver є безкоштовним універсальним інструментом для баз даних з відкритим кодом, призначеним для розробників та адміністраторів баз даних. Головна мета проекту - забезпечити зручність використання, для чого інтерфейс програми був ретельно розроблений та реалізований.

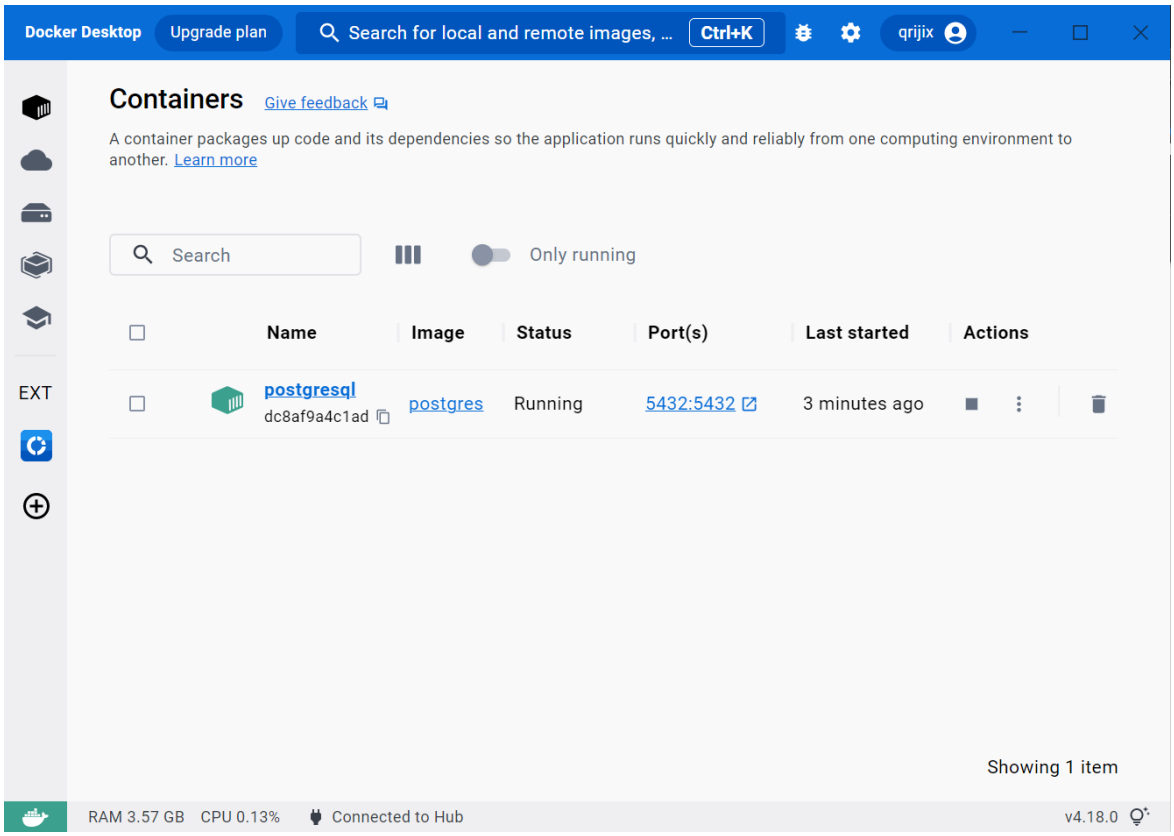

Рисунок 11 – Docker Desktop

DBeaver має відкритий код, є мультиплатформовим та заснованим на відкритому фреймворку, що дозволяє писати різні розширення (плагіни). Програма підтримує будь-яку базу даних, яка має драйвер JDBC, а також може обробляти будь-яке зовнішнє джерело даних, яке може мати або не мати драйвер JDBC. Для DBeaver існує набір плагінів для різних баз даних та різних утиліт керування базами даних, наприклад, ERD, передача даних, порівняння, експорт/імпорт даних, генерація імітаційних даних тощо. Крім того, програма має велику кількість функцій [12].

У базі даних, що використовується, є чотири таблиці:

1. Таблиця "user_entity": ця таблиця містить інформацію про користувачів, таку як їх ідентифікатор, ім'я, прізвище, вік, адреса, основна інформація про користувача, позиція, ідентифікатор аватара та ідентифікатор фонової картинки.

2. Таблиця "user_background_entity": ця таблиця містить дані розширення фонової картинки, назву, розмір та mime_type.
3. Таблиця "user_avatar_entity": ця таблиця містить дані розширення картинки для аватара, назву, розмір та mime_type.
4. Таблиця "post_entity": ця таблиця містить дані про публікації користувачів. У таблиці можуть є поля для зберігання ідентифікатора повідомлення, тексту та дати створення.

Схема бази даних зображена на рисунку 12.

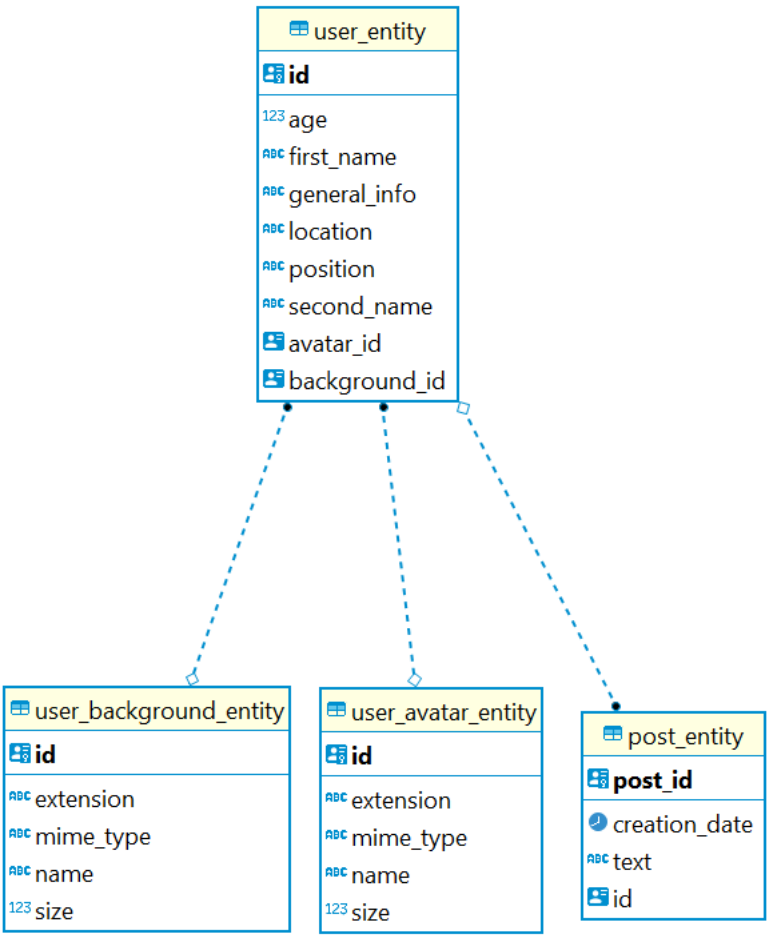

Рисунок 12 – Схема бази даних

4.4 Реалізація RSQL

У проєкті сервісу соціальної мережі для працевлаштування виникає необхідність реалізації пошуку із заданим фільтром. Прикладом можуть слугувати пошук користувачів за допомогою фільтрів, сортування користувачів, пошук вакансій за допомогою різних фільтрів та сортування.

Існує така мова запитів, яка дозволяє клієнтам здійснювати складні запити до REST-сервісу за допомогою простої рядкової форми – RSQL (RESTful Service Query Language). RSQL - це мова запитів для параметричної фільтрації записів у RESTful API. Вона заснована на FIQL (Feed Item Query Language) - синтаксисі, дружньому до URI, для вираження фільтрів по записах в атомарній стрічці. FIQL чудово підходить для використання в URI; в ньому немає небезпечних символів, тому кодування URL-адреси не потрібне. З іншого боку, синтаксис FIQL не дуже інтуїтивно зрозумілий, а кодування URL-адреси не завжди має велике значення, тому RSQL надає більш дружній синтаксис для логічних операторів і деяких операторів порівняння.

У роботі використовується RSQL парсер, автором якого є Jakub Jirutka, ентузіаст відкритого коду та розробник Alpine Linux [13].

Оператори порівняння, що реалізовані у цьому парсері:

- Дорівнює : «==»
- Не дорівнює: «!=»
- Менше ніж: «=lt=» або «<»
- Менше або дорівнює : «=le=» «або «<=»
- Більше ніж оператор : «=gt=» або «>»
- Більше або дорівнює : «=ge=» або «>=»
- В: «=in=»
- Не в : «=out=»

Також можливо розширити цей аналізатор своїми власними операторами.

У ході роботи було вирішено додати RSQL у сервіс соціальної мережі у зв’язку зі зручністю використання фільтрів та оптимізацією коду.

Розглянемо, як реалізовано у сервісі соціальної мережі пошук користувачів з фільтрацією. Фрагмент коду наведено на рисунку 13.

```
@Component
public abstract class EntityVisitor<T> implements RSQLVisitor<Specification<T>, Void> {
    @Override
    public Specification<T> visit(AndNode andNode, Void unused) { return null; }

    @Override
    public Specification<T> visit(OrNode orNode, Void unused) { return null; }

    @Override
    public Specification<T> visit(ComparisonNode comparisonNode, Void unused) {
        var operator = comparisonNode.getOperator().getSymbol();
        var fieldName = comparisonNode.getSelector();
        var argument = comparisonNode.getArguments().get(0);

        if (operator.equals("==")) {
            return new Specification<T>() {
                no usages
                @Override
                public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder criteriaBuilder) {
                    return criteriaBuilder.equal(root.get(fieldName), argument);
                }
            };
        }
```

Рисунок 13 – EntityVisitor

Клас EntityVisitor<T> є абстрактним класом, який реалізує інтерфейс RSQLVisitor<Specification<T>, Void>. Цей інтерфейс дозволяє перетворювати RSQL запити на Spring Data JPA специфікації, які можуть бути використані для пошуку даних в базі даних.

Клас містить реалізації методів інтерфейсу RSQLVisitor. Наприклад, метод visit(ComparisonNode comparisonNode, Void unused) визначає поведінку при обробці вузла порівняння RSQL запиту. За допомогою comparisonNode метод отримує інформацію про оператор порівняння, поле, яке порівнюється, та значення, з яким порівнюється поле.

У залежності від оператора порівняння метод повертає специфікацію, яка порівнює значення поля зі значенням, переданим в запиті. Метод visit, який розділяє rsql на fieldName, operator, argument, викликає певний метод criteriaBuilder

в залежності від оператору. Наприклад дано оператор «==», метод visit викличе criteriaBuilder.equal(fieldName, argument) і побудує specification, яку репозиторій від Hibernate і SringJpa використає для пошуку значень в БД. Це дозволяє використовувати RSQL запити для пошуку даних у базі даних за допомогою Spring Data JPA.

4.5 Завантаження зображень

Розглянемо метод для завантаження аватару користувача. Фрагмент коду з реалізацією наведено на рисунку 14.

Цей фрагмент коду є методом для завантаження аватара користувача на сервер. Тут використовується анотація @Transactional, щоб забезпечити транзакційність операцій з базою даних. У випадку exception або інших неполадок в роботі API анотація дозволяє повернути результат перед тим, як починає працювати метод. Метод отримує назву файлу, файловий об'єкт та ідентифікатор користувача. Починається метод з пошуку користувача за userId в базі даних. Якщо користувач не знайдений, генерується виключення ResourceNotFoundException. Далі перевіряється чи не порожній файл та чи не порожня назва файлу, якщо так, то викликається відповідний виняток.

Потім метод читає байти файлу та генерує унікальний ідентифікатор використовуючи клас UUID. За допомогою цього ідентифікатора метод створює директорію для збереження аватара, використовуючи метод createDirectory(). Крім того, створюється об'єкт класу MimetypesFileTypeMap, який використовується для визначення MIME-типу файлу. Далі створюється файл з аватаром, записуються байти файлу в цей файл та закривається потік.

Після цього створюється об'єкт класу UserAvatarEntity, який містить унікальний ідентифікатор, назву файлу, розмір файлу, розширення файлу та MIME-тип файлу. Цей об'єкт зв'язується з об'єктом користувача та зберігається в таблиці

бази даних аватарів користувачів. Крім того, зберігається зв'язок між користувачем та його аватаром в таблиці користувачів.

Якщо сталася помилка під час завантаження зображення або зберігання інформації в базі даних, генерується виключення ResourceFileUploadErrorException. Успішне завершення методу повертає об'єкт ResponseEntity зі статусом HTTP CREATED.

```
@Transactional
public ResponseEntity<HttpStatus> uploadUserAvatar(String name, MultipartFile file, UUID userId) {
    UserEntity user = usersRepository.findById(userId)
            .orElseThrow(
                    () -> new ResourceNotFoundException(USER_NOT_FOUND + userId)
            );

    if (file.isEmpty()) {
        throw new ResourceBadRequestException(FILE_IS_EMPTY);
    }
    if (name.isBlank() || name.isEmpty()) {
        throw new ResourceBadRequestException(FILE_NAME_IS_EMPTY);
    }

    try {
        byte[] bytes = file.getBytes();

        UUID uuid = UUID.randomUUID();
        var path = createDirectory(uuid, ROOT_PATH_AVATARS);
        var fileTypeMap = new MimetypesFileTypeMap();

        File avatar = new File(path,  child: uuid + FileUtils.getFileExtension(file));
        BufferedOutputStream stream = new BufferedOutputStream(new FileOutputStream(avatar));

        stream.write(bytes);
        stream.close();


        UserAvatarEntity userAvatarEntity = new UserAvatarEntity(
                uuid,
                name,
                file.getSize(),
                FileUtils.getFileExtension(file),
                fileTypeMap.getContentType(avatar));
        user.setUserAvatarEntity(userAvatarEntity);
        this.userAvatarRepository.save(userAvatarEntity);
        this.usersRepository.save(user);
        usersRepository.flush();


        return new ResponseEntity<>(HttpStatus.CREATED);
    } catch (Exception e) {
        throw new ResourceFileUploadErrorException(FILE_FAILED_UPLOAD, e);
    }
}
```

Рисунок 14 – Завантаження аватару користувача

4.7 Документація та тестування API

4.7.1 Swagger як інструмент документації та тестування

Swagger є відкритим фреймворком, який дозволяє створювати, відображати та тестувати RESTful API. Swagger забезпечує зручний інтерфейс, який дозволяє легко описувати, тестувати та документувати API. Swagger дозволяє описувати ресурси API за допомогою мови OpenAPI Specification (раніше відомої як Swagger Specification). Цей опис містить інформацію про ресурси, їх параметри, типи запитів, відповіді, коди стану та багато іншого.

Swagger також надає інтерфейс Swagger UI, який можна використовувати для тестування API. Інтерфейс Swagger UI забезпечує зручний спосіб для взаємодії з API, дозволяючи відображати та тестувати ресурси, їх параметри та відповіді.

Swagger допомагає створювати документацію для API. За допомогою Swagger можна створювати документацію в форматі HTML, PDF та JSON.

Узагальнюючи, Swagger - це потужний інструмент для тестування та документування RESTful API. Він забезпечує зручний інтерфейс для розробників, що дозволяє зосередитися на розробці функціоналу API, а не на його документації і тестуванні [16].

У ході роботи було згенеровано OpenAPI специфікацію сервісу у вигляді файлу з json форматом, на його основі сворено OpenAPI специфікацію у вигляді діаграми (Додаток Б).

Для прикладу розглянемо тестування POST запиту з фільтром з допомогою Swagger. Для проведення тесту було створено трьох користувачів віком 21, 22 та 23 роки та іменами Name 1, Name 2, Name 3 відповідно.

Тест 1: шукаємо всіх користувачів, що старші за 22 роки. Очікуваний результат: інформація про користувача Name 3.

Запит: age>22

Реальний результат: інформація про користувача Name 3 – зображено на рисунку 15.

Тест 2: шукаємо всіх користувачів, віком до 22 років включно. Очікуваний результат: інформація про користувачів Name 1 та Name 2.

Запит: age<=22

Відповідь: інформація про користувачів Name 1 та Name 2 у тому ж вигляді, що й у попередньому тесті.

Реальний результат: інформація про користувачів Name 1 та Name 2.

Отже, дані тести пройшли успішно. На основі цих та інших тестів з’ясовано, що у створеному сервісі пошук з фільтром працює належним чином, коректно.

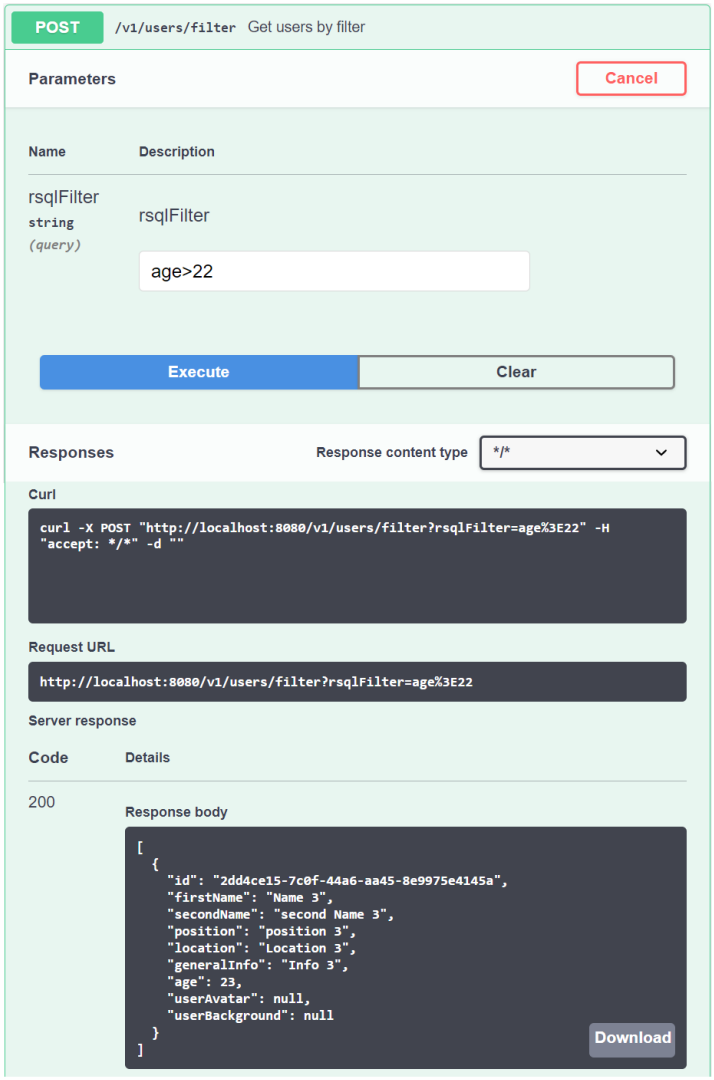

Рисунок 15 – Тест пошуку з фільтром

4.7.2 Використання Postman у тестуванні API

Postman - це інструмент для тестування API. Postman був розроблений у 2012 році розробником програмного забезпечення та підприємцем Абхінавом Астаною, щоб спростити розробку та тестування API. Цей інструмент можна використовувати для проектування, документування, перевірки, створення та зміни API.

Postman має функцію надсилання та перегляду запитів і відповідей за протоколом передачі гіпертексту (HTTP). Він має графічний інтерфейс користувача (GUI) і може використовуватися на таких платформах, як Linux, Windows і Mac. Він може створювати та надсилати різні HTTP-запити - POST, PUT, GET [16].

За допомогою Swagger неможливо надіслати файл у тілі запиту тривіальним чином, тому для тестування завантаження картинки для аватару або фонової картинки користувача використовуємо Postman.

Розглянемо приклад такого тесту. Параметри тестового запиту:

- name = avatarName2
- userId = 736f425f-9a22-4255-8a88-769a9a150a90

Тіло запиту – файл фотографії, який можна завантажити в Postman з допомогою графічного меню.

Postman формує такий запит POST: http://localhost:8080/v1/users/upload-background?name=avatarName2&userId=736f425f-9a22-4255-8a88-769a9a150a90, що містить введену нами інформацію.

Після цього надсилаємо запит і отримуємо у якості результату статус «201 Created», що свідчить про успішне завершення перевірки. На рисунку 16 виділено даний статус зеленим кольором. На основі цих та інших тестів з’ясовано, що у створеному сервісі маніпуляції із зображеннями для облікового запису функціонують коректно.

Отже, було розглянуто деякі аспекти реалізації сервісу та визначено, що створений сервіс відповідає вимогам та може бути використаним для подальшої розробки соціальної мережі.

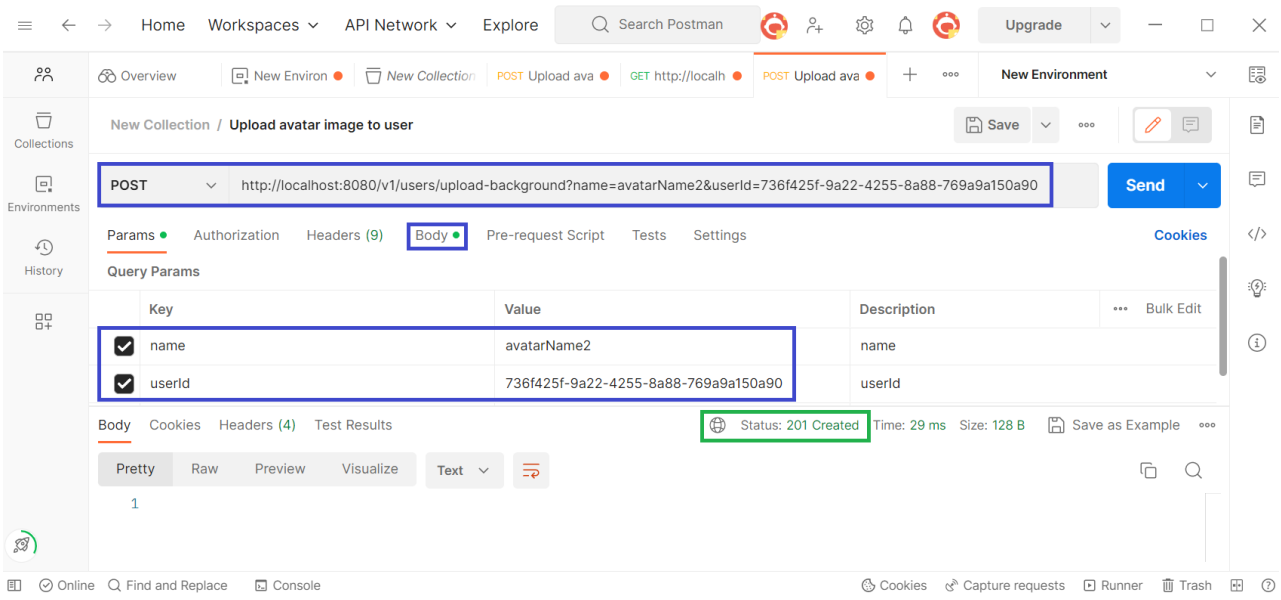

Рисунок 16 – Тест завантаження картинки для аватара

ВИСНОВКИ

Розвиток можливостей і зростання популярності соціальних мереж та месенджерів, зручність їх інформаційного спілкування суттєво змінили кадрову роботу на ринку праці, зокрема такий важливий її компонент як пошук місця працевлаштування. Проведені в кваліфікаційній роботі дослідження існуючих технологій та сервісів дозволили створити програмне забезпечення яке може допомогти більш ефективному, швидкому та зручному працевлаштуванню.

На основі проведеного дослідження використання соціальних мереж для кадрових цілей були визначені функціональні та нефункціональні вимоги створюваного програмного продукту, розроблено вимоги та змодельовано поведінку системи.

В роботі було досліджено зручні для використання в соціальних мережах сучасні архітектурні шаблони програмного забезпечення та на підставі аналізу обрано мову програмування Java, Spring Framework.

З використанням цієї мови створено та протестовано сервіс для соціальної мережі, в якому реалізовано побудова нових користувачів з персональними даними і аватаром, публікація постів і їх пошук з фільтром.

Дана робота може бути використана для створення нового програмного забезпечення зі зручним та ефективним інтерфейсом користувачів, що допоможе їм знайти потрібне місце працевлаштування, яке відповідає їхнім потребам та кваліфікації.

Зважаючи на тенденції швидкого розвитку соціальних мереж з’являтимуться потреби у створенні сервісів з новими, важливими для користувачів функціями, тому розпочаті у кваліфікаційній роботі дослідження варто продовжувати.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сизько М. Дослідження та розробка проєкту сервісно-орієнтованої соціальної мережі : матеріали XXI міжнар. наук.-прак. конф. "Шевченківська весна - 2023", м. Київ, 14 квітня 2023 р. С. 107.
2. Коваль Ю. В. Інформаційні мережі: навчальний посібник / Ю. В. Коваль, А. Б. Ставровський. – Київ, 2021. – 84 с.
3. DATAREPORTAL [Електронний ресурс]. Режим доступу до ресурсу: https://datareportal.com/.
4. Dennis Alan, Wixom Barbara, Tegarden David, Systems Analysis and Design: An Object-Oriented Approach with UML, 6th Edition: 5th Edition, Hoboken, 2020, 544p.
5. Martin Fowler, Uml Distilled: A Brief Guide To The Standard Object Modeling Language: 3d Edition, Pearson 2003, 179 p.
6. Andrew S. Tanenbaum, David J. Wetherall, Computer Networks 5th Edition, 2010. - 960 с.
7. REST API Tutorial [Електронний ресурс]. Режим доступу до ресурсу: https://restfulapi.net/.
8. IBM [Електронний ресурс]. Режим доступу до ресурсу: https://www.ibm.com/.
9. Eckel B. Thinking in Java 4th Edition: Pearson, 2006, 1150 p.
10. Horstmann C. S. Core Java Volume I – Fundamentals: 11th Edition, Prentice Hall 2018, 889 p.
11. docker docs [Електронний ресурс]. Режим доступу до ресурсу: https://docs.docker.com/.
12. HEVO [Електронний ресурс]. Режим доступу до ресурсу: https://hevodata.com/.

13. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. Режим доступу до ресурсу: https://www.postgresql.org/.
14. DBeaver Community [Електронний ресурс]. Режим доступу до ресурсу: https://dbeaver.io/.
15. Jakub Jirutka [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/.
16. Swagger API Platform - Fast API Development [Електронний ресурс]. Режим доступу до ресурсу: https://swagger.io/.
17. Postman API Platform [Електронний ресурс]. Режим доступу до ресурсу: https://www.postman.com/.

ДОДАТОК А

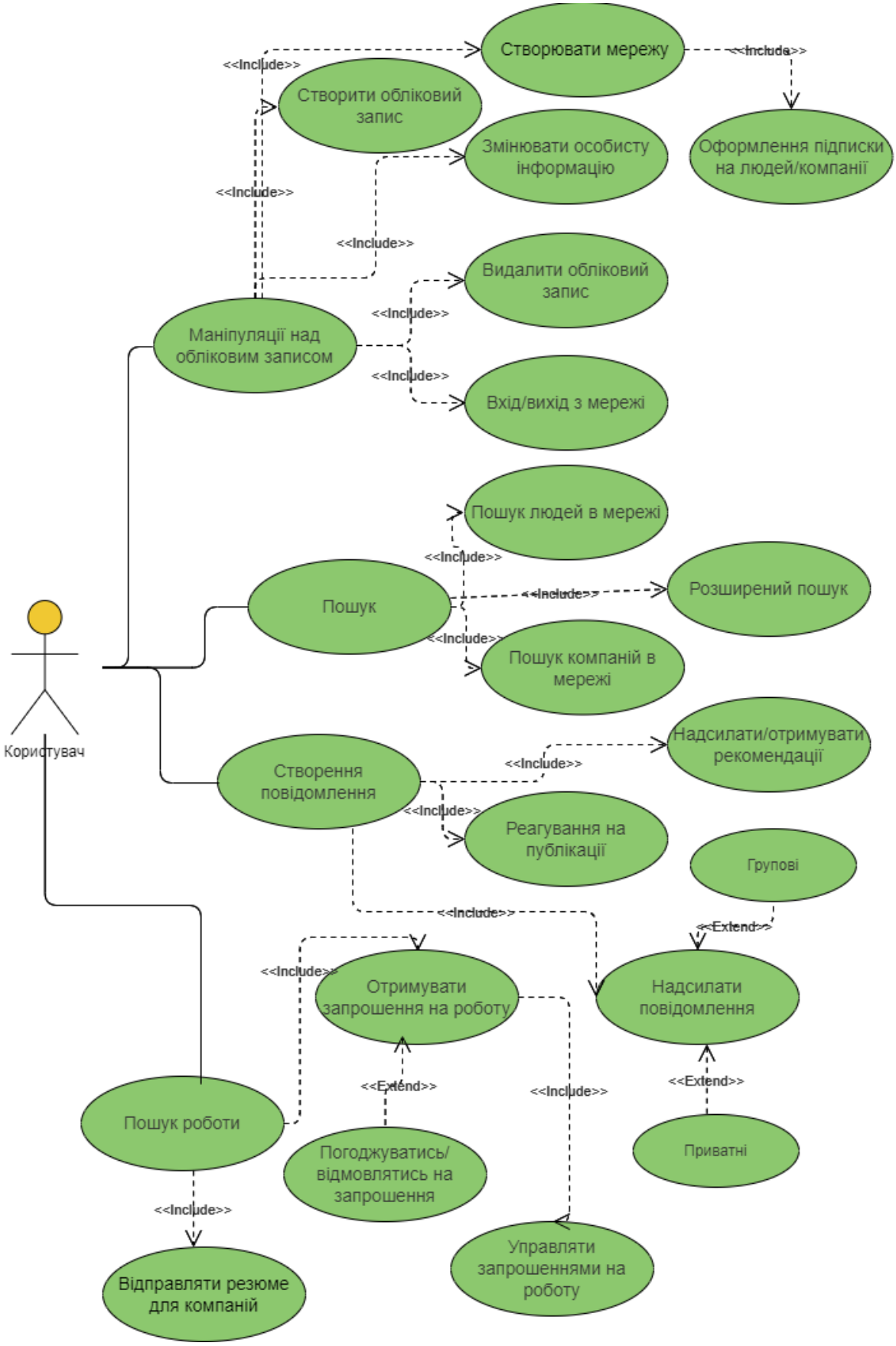

Рисунок А.1 – Use case діаграма соціальної мережі

ДОДАТОК Б

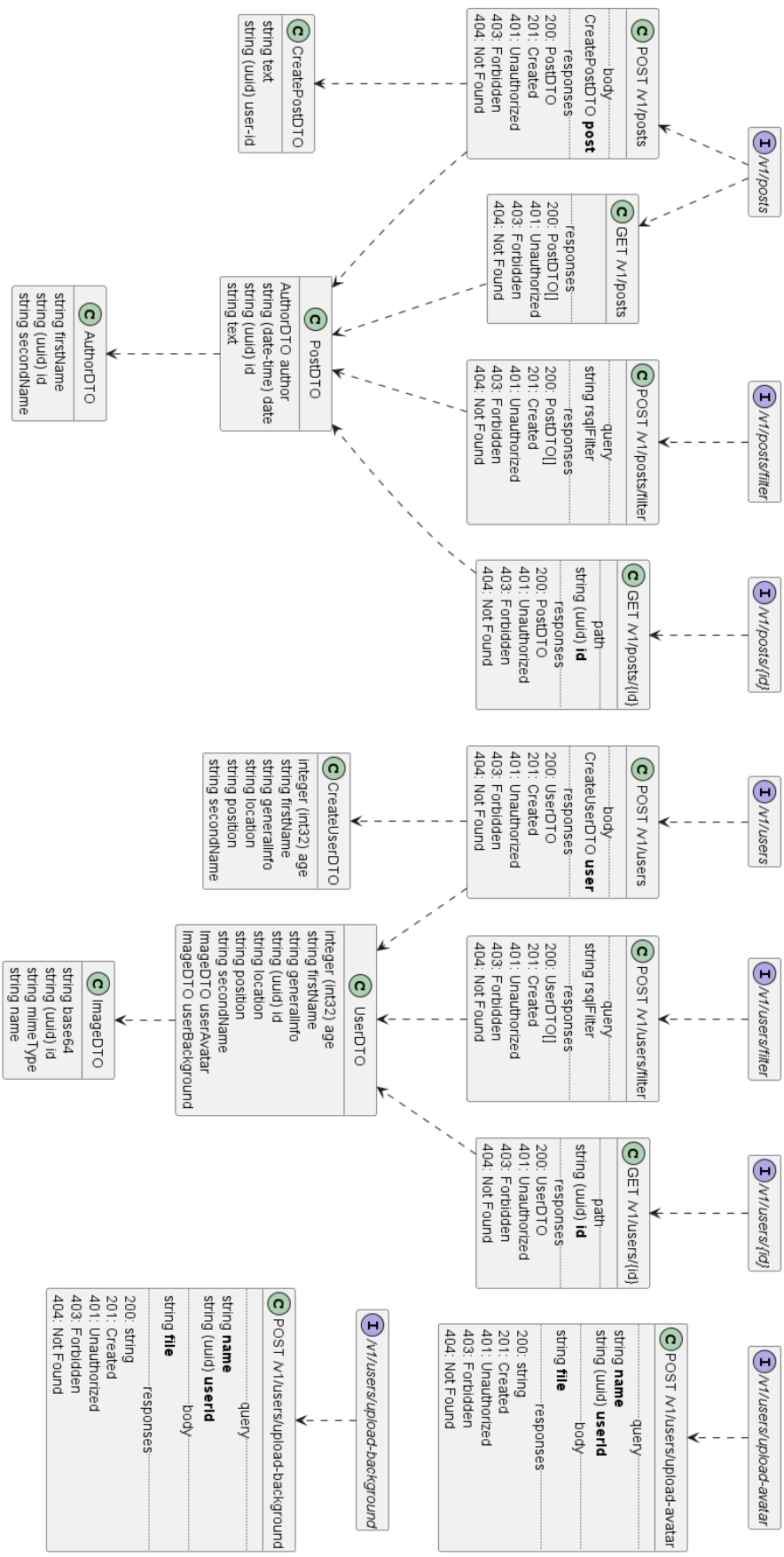

Рисунок Б.1 – OpenAPI специфікація