НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок "Персоналізований новинний агрегатор"

Виконав студент IV курсу, групи ІТ-04
(шифр групи)

Філіповський Даніїл Євгенович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., Писаренко А.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Філіповський Данііл Євгенович
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок "Персоналізований новинний агрегатор"
керівник проєкту Ліщук Катерина Ігорівна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27»травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст поясню вальної записки
1) Предпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання, аналіз системних вимог, розроблення функціональних вимог, розроблення нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: Архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів програмного забезпечення

3) Схема бази даних

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	13.03.2024	
3	Постановка та формалізація задачі	15.03.2024	
4	Розробка інформаційного забезпечення	23.03.2024	
5	Алгоритмізація задачі	29.03.2024	
6	Обґрунтування вибору використаних технічних засобів	09.04.2024	
7	Розробка програмного забезпечення	18.05.2024	
8	Налагодження програми	22.05.2024	
9	Виконання графічних документів	25.05.2024	
10	Оформлення пояснювальної записки	30.05.2024	
11	Подання ДП на попередній захист	03.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Даніїл ФІЛІПОВСЬКИЙ
(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 38 таблиць, 21 рисунків та 11 джерел – загалом 67 сторінки.

Дипломний проєкт присвячений розробці веб-застосунку для персоналізованої агрегації новин.

Мета – створення веб-застосунку для персоналізованої агрегації новин, який дозволить користувачам отримувати релевантний контент з різних джерел у зручному форматі.

Об'єкт дослідження: програмне забезпечення для агрегації новин.

Предмет дослідження: методи та засоби персоналізації новинного контенту у веб-застосунках.

У розділі «Передпроєктне обстеження предметної області» було проведено аналіз предметної області, окреслено мету розробки проекту, визначено основні цілі та завдання.

Розділ «Розроблення вимог до програмного забезпечення» присвячений розробці варіантів використання програмного забезпечення, аналізу системних вимог, формуванню нефункціональних та функціональних вимог.

Розділ «Конструювання та розроблення програмного забезпечення» описує архітектуру програмного забезпечення, обґрунтовує вибір засобів розробки, описує структури даних та розглядає питання безпеки даних.

Розділ «Аналіз якості та тестування програмного забезпечення» присвячений аналізу якості програмного забезпечення, проведенню тестувань та опису контрольних прикладів.

Розділ «Розгортання та супровід програмного забезпечення» описує процеси розгортання системи та її подальший супровід.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ПЕРСОНАЛІЗОВАНА АГРЕГАЦІЯ НОВИН, DOCKER, POSTGRESQL, REDIS, SPRING BOOT, VUE3, VUETIFY.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 38 tables, 21 figures, and 11 sources – in total 67 pages.

The diploma project is dedicated to the development of a web application for personalized news aggregation.

Purpose – to create a web application for personalized news aggregation, allowing users to receive relevant content from various sources in a convenient and customizable format.

Object of research: software for news aggregation.

Subject of research: methods and means of personalizing news content in web applications.

The section "Pre-project Survey of the Subject Area" analyzes the subject area, outlines the project's development goal, and defines the main objectives and tasks.

The section "Development of Software Requirements" is dedicated to developing use cases for the software, analyzing system requirements, and forming non-functional and functional requirements.

The section "Design and Development of Software" describes the software architecture, justifies the choice of development tools, details data structures, and addresses data security issues.

The section "Quality Analysis and Software Testing" focuses on analyzing the software quality, conducting tests, and describing control examples.

The section "Deployment and Maintenance of Software" describes the processes of system deployment and its subsequent maintenance.

KEYWORDS: WEB APPLICATION, PERSONALIZED NEWS AGGREGATION, DOCKER, POSTGRESQL, REDIS, SPRING BOOT, VUE3, VUETIFY.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-0421.045440.01.91	Технічне завдання	16	
3	А4	КПІ.ІТ-0421.045440.02.81	Пояснювальна записка	67	
4	А4	КПІ.ІТ-0421.045440.03.12	Текст програми	36	
5	А4	КПІ.ІТ-0421.045440.04.51	Програма та методика тестування	4	
6	А4	КПІ.ІТ-0421.045440.05.34	Керівництво користувача	12	
7	А3	КПІ.ІТ-0421.045440.06.99	Графічний матеріал	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-0421.045440.00.90		
Розроб.		Філіповський Д.Є.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К.І.					
Н.контр.		Родіонов П.Ю				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-04	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Вебзастосунок "Персоналізований новинний агрегатор"

Технічне завдання

КПІ.ІТ-0421.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Катерина ЛІЩУК

Нормоконтроль:

_____ Павло РОДІОНОВ

Виконавець:

_____ Даніїл ФІЛІПОВСЬКИЙ

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ 3
2 ПІДСТАВА ДЛЯ РОЗРОБКИ 4
3 ПРИЗНАЧЕННЯ РОЗРОБКИ 5
4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6
4.1 Вимоги до функціональних характеристик 6
4.2 Вимоги до надійності 11
4.3 Умови експлуатації 11
4.4 Вимоги до складу і параметрів технічних засобів 12
4.5 Вимоги до інформаційної та програмної сумісності 12
4.6 Вимоги до маркування та пакування 13
4.7 Вимоги до транспортування та зберігання 13
4.8 Спеціальні вимоги 13
5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ 14
6 СТАДІЇ І ЕТАПИ РОЗРОБКИ 15
7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 16

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для персоналізованої агрегації новин

Галузь застосування:

Наведене технічне завдання поширюється на розробку вебзастосунку для агрегування новин з різних джерел. Вебзастосунок призначений для збору, організації та відображення новин з різних веб-сайтів, що дозволяє користувачам отримувати актуальну інформацію в одному місці.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки вебзастосунку для агрегування новин є завдання на дипломне проектування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації процесу збору новин з різних джерел та їх подальшого відображення в зручному для користувача інтерфейсі. Метою розробки є підвищення ефективності доступу до новин за рахунок автоматизації процесів збору, організації та фільтрації новин.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу:

веб інтерфейс користувача для налаштування та управління новинним агрегатором.

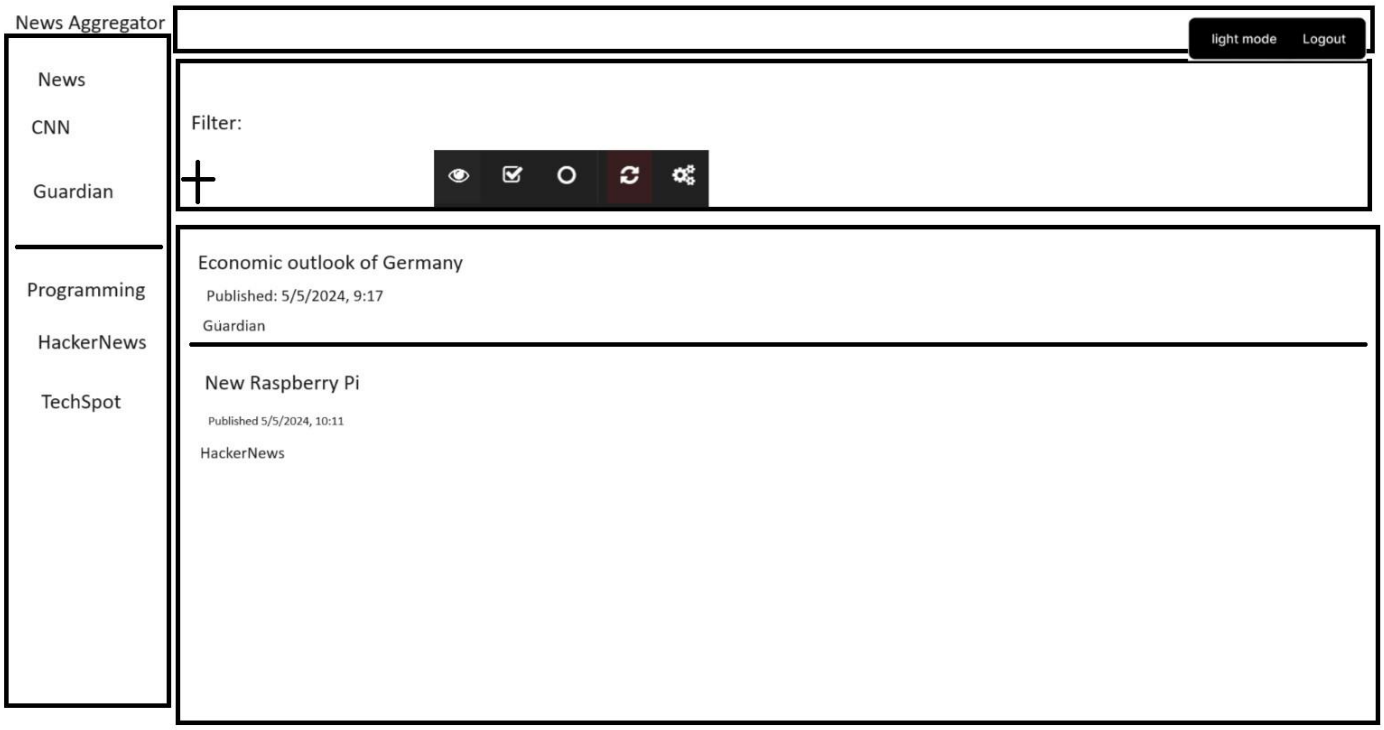


Рисунок 4.1 – Головний екран з новинною стрічкою

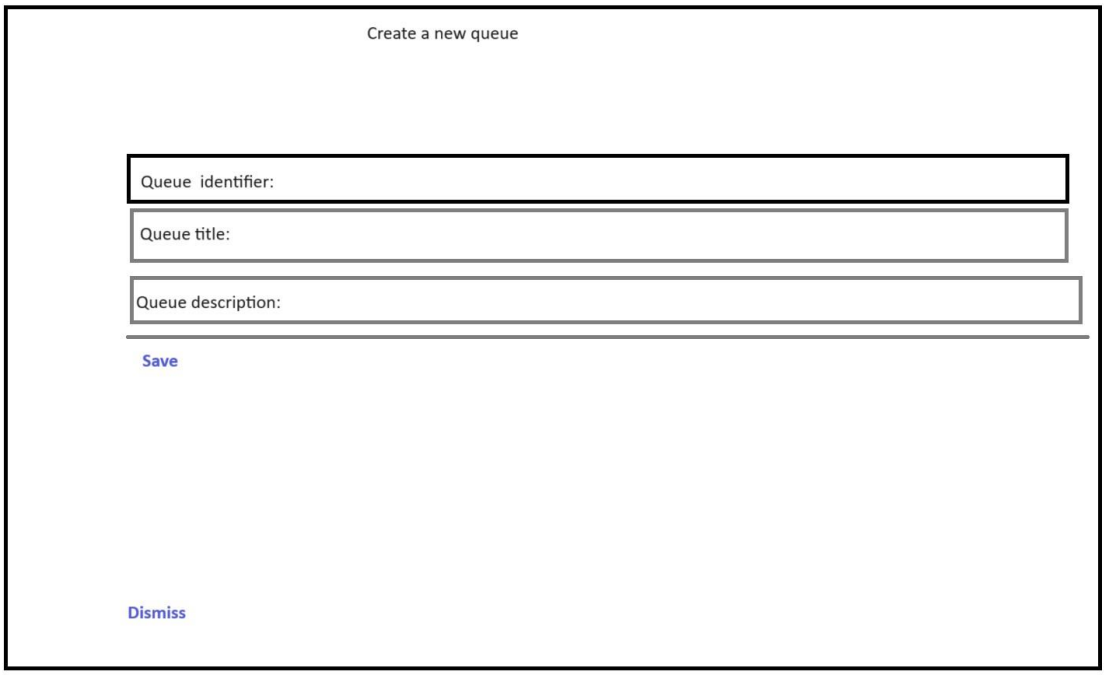


Рисунок 4.2 – Екран створення нової категорії

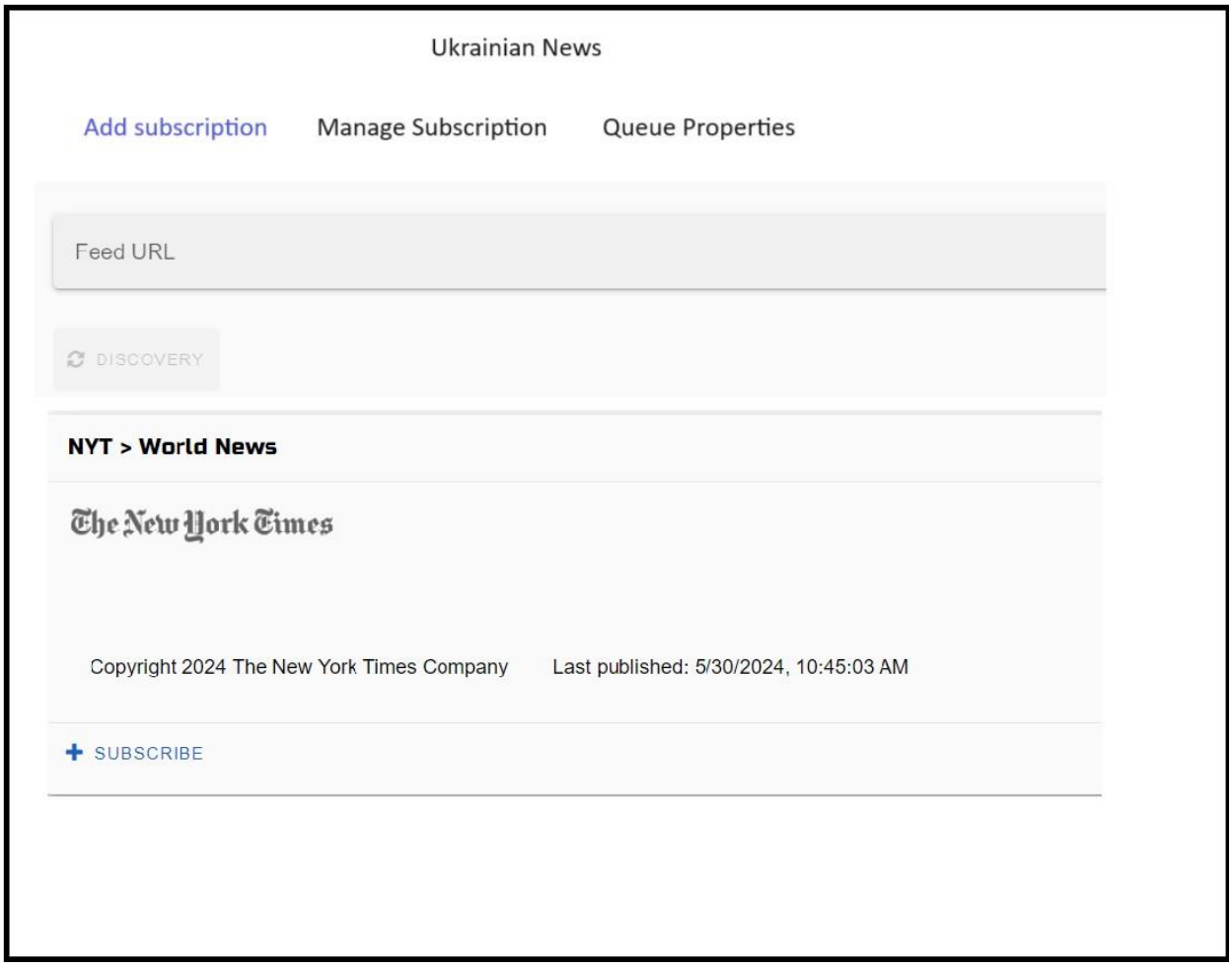


Рисунок 4.3 – Екран додавання нової підписки

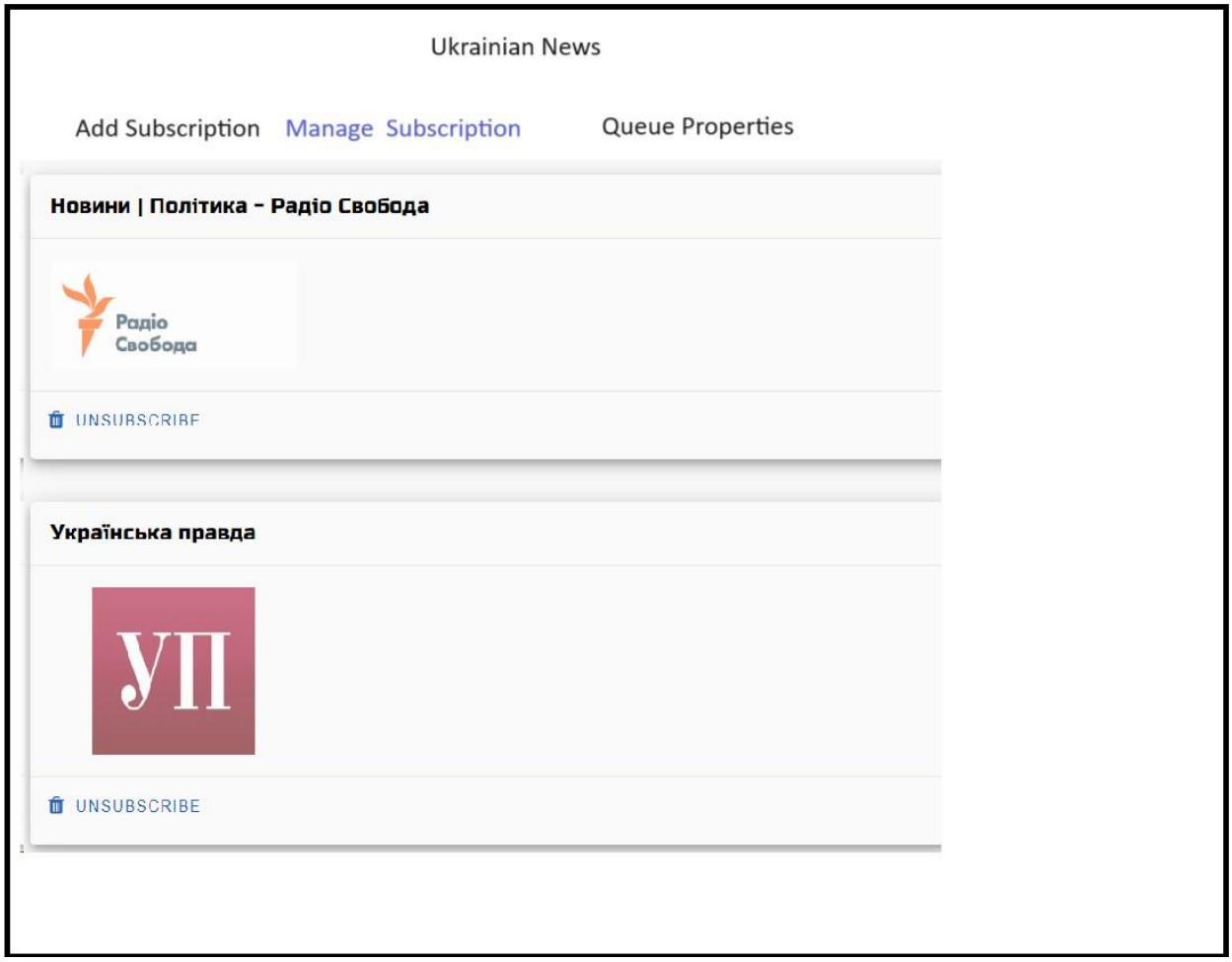


Рисунок 4.4 – Екран управління підписками

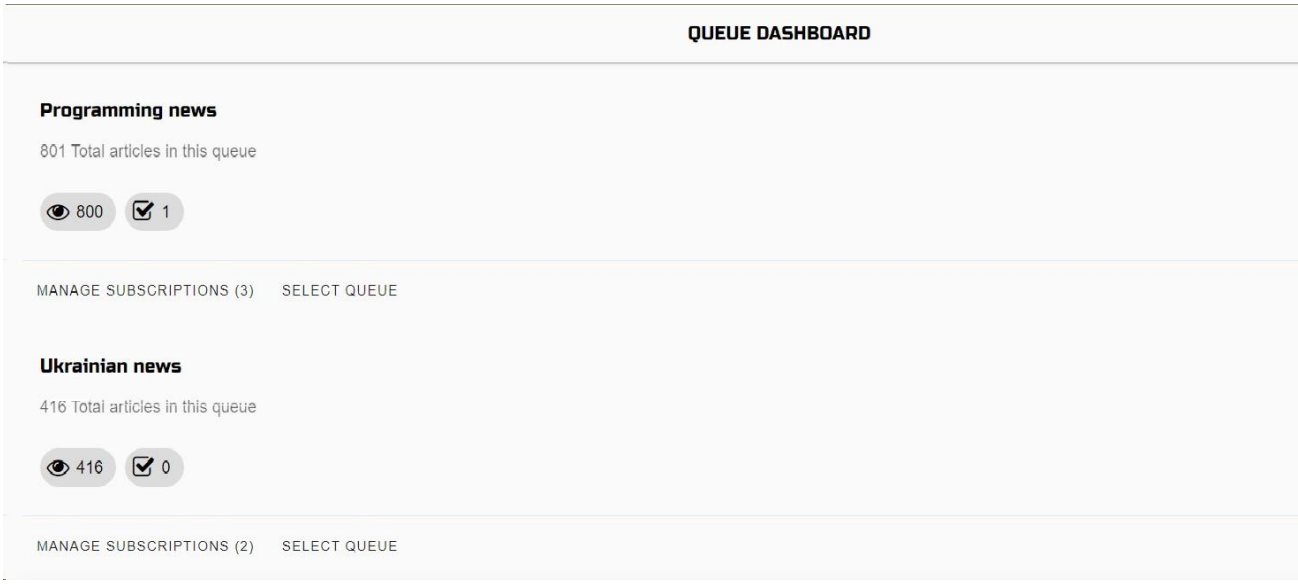


Рисунок 4.5 – Панель категорій

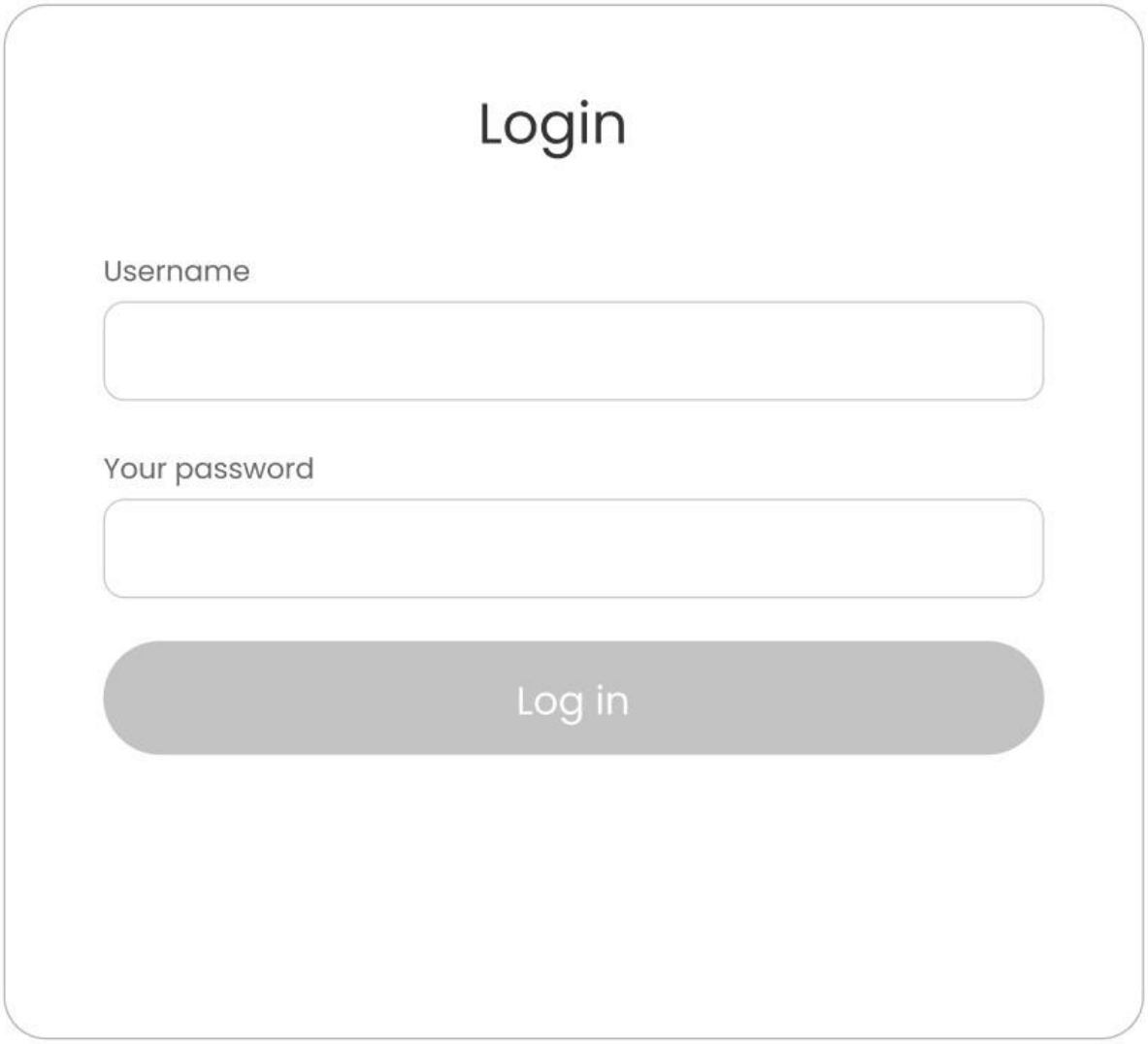


Рисунок 4.6 – Екран логіну користувача

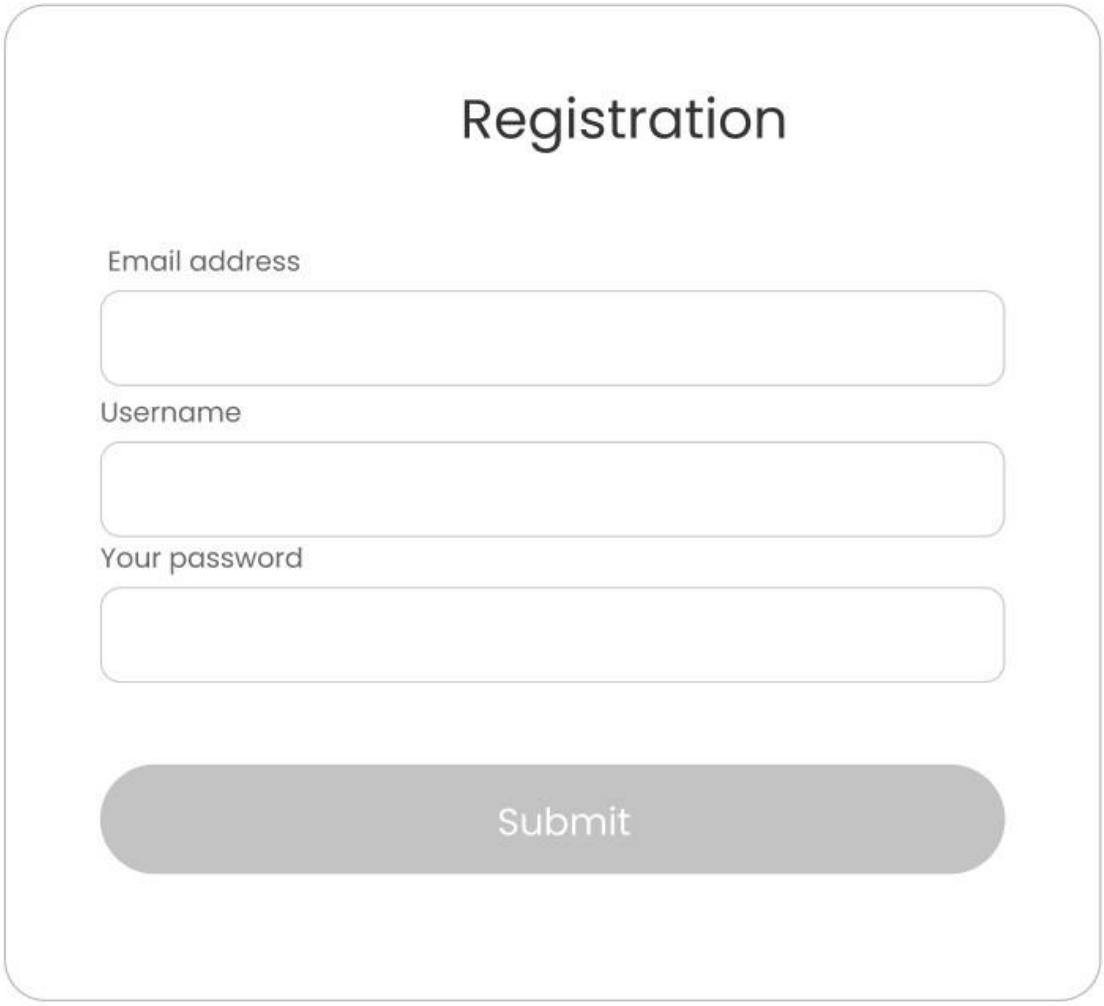


Рисунок 4.7 – Екран реєстрації користувача

4.1.2 Для користувача:

- перегляд новинної стрічки;
- можливість додавання нових підписок (RSS/ATOM);
- управління існуючими підписками (редагування та видалення);
- організація новин у черги;
- фільтрація новин за автором, датою, ключовими словами;
- пошук новин;
- відмічання новин як прочитані або прочитати пізніше;
- відкриття оригінальних статей у нових вкладках;

- ділення посиланнями на новини у соціальних мережах;
- персоналізація вигляду інтерфейсу (вибір теми);
- реєстрація та авторизація користувача.

4.1.3 Для адміністратора системи (якщо він передбачений):

Не передбачений

4.1.4 Додаткові вимоги:

- імплементація підходу інфраструктури як коду;
- забезпечити безперебійний доступ до додатку;
- забезпечити проведення RollingUpdate під час оновлення додатку;
- забезпечити зберігання образів додатку.

4.2 Вимоги до надійності

- передбачити контроль введення інформації;
- передбачити захист від некоректних дій користувач;
- забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.
Не висуваються

4.3.1 Вид обслуговування

Адміністрування записів користувачів.

4.3.2 Обслуговуючий персонал

Обслуговуючий персонал повинен мати базові навички роботи з комп'ютером та базові знання у веб-технологіях. Зокрема, адміністратор повинен вміти працювати з веб-браузером, системами керування підписками та фільтрацією новин.

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

- процесор: мінімум двоядерний 2 ГГц;
- оперативна пам'ять: Мінімум 4 ГБ;
- дисковий простір: Мінімум 20 ГБ;
- підключення до мережі Інтернет зі швидкістю не менше 20 Мбіт/с.

Рекомендована конфігурація технічних засобів

- процесор: чотириядерний 3 ГГц;
- оперативна пам'ять: 8 ГБ і більше;
- дисковий простір: 50 ГБ і більше;
- підключення до мережі Інтернет зі швидкістю від 100 Мбіт/с.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Windows (7, 8, 8.1, 10 чи пізніше), macOS або Unix.

4.5.1 Вимоги до вхідних даних

Відсутні.

4.5.2 Вимоги до вихідних даних

Відсутні.

4.5.3 Вимоги до мови розробки

Розробка програмного забезпечення повинна виконуватися на мовах програмування Java та JavaScript. Основний фреймворк для серверної частини - Spring Boot, для клієнтської частини - Vue.js.

4.5.4 Вимоги до середовища розробки

Розробка програмного забезпечення повинна виконуватися за допомогою середовищ розробки IntelliJ IDEA для серверної частини та WebStorm для клієнтської частини.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторія на платформі розміщення коду GitHub

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текс програми;
- програма та методика тестування;
- керівництво користувача.

перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

- перевірка збереження і обробки даних;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна класів програмного забезпечення;
- схема структурна варіантів використання;
- схема бази даних.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою роботи	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини роботи	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини роботи	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації роботи	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Вебзастосунок "Персоналізований новинний агрегатор"

КПІ.ІТ-0421.045440.02.81

Київ – 2024

ЗМІСТ

ВСТУП 5
1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 6
1.1 Аналіз предметної області 6
1.2 Аналіз існуючих рішень 8
1.2.1 Аналіз відомих програмних продуктів 8
1.3 Опис бізнес-процесів 11
1.4 Постановка задачі 14
Висновки до розділу 15
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 16
2.1 Варіанти використання програмного забезпечення 16
2.2 Аналіз системних вимог 25
2.3 Розроблення функціональних вимог 26
2.4 Розроблення нефункціональних вимог 29
Висновки до розділу 29
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 31
3.1 Архітектура програмного забезпечення 31
3.2 Обґрунтування вибору засобів розробки 38
3.3 Конструювання програмного забезпечення 41
3.4 Аналіз безпеки даних 50
4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 52
4.1 Аналіз якості ПЗ 52
4.2 Опис процесів тестування 52
4.3 Опис контрольного прикладу 58
Висновки до розділу 64
5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 66
5.1 Розгортання програмного забезпечення 66
5.2 Супровід програмного забезпечення 67

Висновки до розділу 68
ВИСНОВКИ 69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70
ДОДАТОК А ЗВІТ ПОДІБНОСТІ 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
SDK	–	Software development kit
IT	–	Інформаційні технології
ER	–	Entity-Relation diagram
ОС	–	Операційна система.
БД	–	База даних.
RSS	–	Rich Site Summary - збагачене зведення сайту
XML	–	Extensible Markup Language - Розширювана мова розмітки

ВСТУП

У сучасному світі інформація відіграє ключову роль, а доступ до якісних новинних ресурсів є життєво необхідним для багатьох людей. Проте, зважаючи на величезну кількість джерел інформації, стає складніше оперативно отримувати новини, які відповідають індивідуальним інтересам та потребам. Саме тому виникає потреба у розробці інструментів, що забезпечують зручний та персоналізований доступ до новинного контенту. Одним із найбільш перспективних рішень цієї проблеми є створення агрегаторів новин, які дозволяють збирати, фільтрувати та надавати користувачам інформацію з різних джерел в одному місці.

Актуальність розробки обумовлена необхідністю створення інструменту, що забезпечує користувачів можливістю отримувати персоналізовані новини швидко та зручно. З огляду на сучасні тенденції, розробка таких систем є важливою складовою інновацій в галузі інформаційних технологій. Світові гіганти, такі як Google, Microsoft та Apple, активно інвестують у розвиток новинних платформ, інтегруючи їх з іншими сервісами та додатками для підвищення зручності користувачів.

Сфера застосування розробленого веб-застосунку для персоналізованої агрегації новин є дуже широкою. Такий інструмент стане незамінним для індивідуальних користувачів, які бажають отримувати новини з різних джерел у зручному форматі, налаштованому відповідно до їхніх інтересів. Завдяки можливості налаштування, такий інструмент може забезпечити високу гнучкість та адаптивність до потреб різних користувачів.

Метою цієї роботи є створення веб-застосунку для персоналізованої агрегації новин, який дозволить користувачам отримувати релевантний контент з різних джерел у зручному та налаштовуваному форматі.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Персоналізовані агрегатори новин є програмними системами, що автоматизують процес збору, фільтрації та відображення новин з різних джерел відповідно до індивідуальних уподобань користувачів. Ці системи відіграють важливу роль у сучасному інформаційному просторі, де обсяг доступної інформації постійно зростає. Основна мета персоналізованих агрегаторів полягає в тому,пе щоб користувачі могли отримувати лише ті новини, які відповідають їхнім інтересам, що значно зменшує інформаційне перевантаження і підвищує ефективність споживання контенту.

Основні переваги персоналізованих агрегаторів новин:

– економія часу: користувачі отримують доступ до релевантної інформації без необхідності переглядати численні вебсайти;

– підвищення обізнаності: інформування про важливі події та новини, які можуть залишитися непоміченими в загальному потоці інформації;

– індивідуальний підхід: можливість налаштування стрічок новин відповідно до особистих інтересів та уподобань;

– відсутність реклами: багато персоналізованих агрегаторів новин не містять реклами, що забезпечує більш комфортний досвід користувача.

– пояснювальна записка;

– технічне завдання;

– текс програми;

– програма та методика тестування;

– керівництво користувача.

Однією з ключових технологій, що лежать в основі агрегаторів новин, є RSS. Це формат передачі даних в Інтернеті, представлений у вигляді збору та трансляції інформації з вебсайтів на спеціальне програмне забезпечення

користувачів. RSS дозволяє автоматично отримувати оновлення від різних вебсайтів та представляти їх у зручному форматі.

RSS є XML-форматом. Для запуску власного каналу вебмайстру необхідно створити спеціальний файл з розширенням .RSS або .XML і підключити його до свого сайту. Потім відвідувачі сайту можуть додати цей канал до свого RSS-рідера і таким чином підписатися на стрічку новин [1].

Переваги RSS включають:

– зручність: можливість упорядкування цікавого контенту з різних сайтів в одній стрічці;

– швидкість: повідомлення про новини надходять у рідери миттєво після їх публікації на сайтах-джерелах;

– безкоштовність: RSS – це повністю безкоштовний метод розповсюдження контенту;

– економія трафіку: RSS-фід постачає структурований текстовий контент з мінімумом медіа-контенту;

– відсутність реклами: важливою перевагою є відсутність реклами при передачі даних з RSS.

На сьогоднішній день існує велика кількість типів RSS-рідерів: вбудовані опції в браузерах, настільні та мобільні додатки, популярні онлайн-сервіси для читання RSS-каналів, а також можливість отримувати інформацію про новий веб-контент через e-mail розсилки.

На поточний момент основними недоліками існуючих рішень у сфері персоналізованих агрегаторів новин є:

– інформаційне перевантаження: незважаючи на персоналізацію, користувачі можуть все ще стикатися з великою кількістю нерелевантної інформації;

– безпека даних: загрози, пов'язані з безпекою персональних даних користувачів, що збираються та зберігаються системою;

– приватність: проблеми з приватністю даних користувачів, які можуть бути використані для цільової реклами або інших комерційних цілей.

Можливі шляхи покращення ситуації:

– покращення алгоритмів персоналізації: використання більш складних способів фільтрації для точнішої персоналізації новин;

– покращення безпеки: впровадження більш надійних механізмів захисту даних, таких як шифрування та анонімізація;

– фокус на приватність: забезпечення користувачів інструментами для контролю за своїми даними та можливістю налаштування рівня приватності.

1.2 Аналіз існуючих рішень

Проаналізуємо сучасні технічні рішення в цій сфері, які можуть сприяти створенню персоналізованого новинного агрегатора. Також розглянемо наявні програмні рішення, допоміжні інструменти та засоби розробки..

1.2.1 Аналіз відомих програмних продуктів

Feedly – це агрегатор новин, доступний для різних веб-браузерів та мобільних пристроїв на базі iOS та Android. Feedly дозволяє користувачам збирати новини з різних джерел, налаштовувати та ділитися контентом. Додаток має мінімалістичний дизайн і підтримує входження через обліковий запис Google. Основний функціонал включає можливість збереження статей, позначення їх як прочитаних, а також додавання нових джерел. Feedly синхронізується з соціальними мережами, такими як Facebook і Twitter, що дозволяє користувачам переглядати всі цікаві посилання в одному зручному інтерфейсі. Розширений доступ до функцій програми надається за умови платної підписки. (рисунок 1.1).

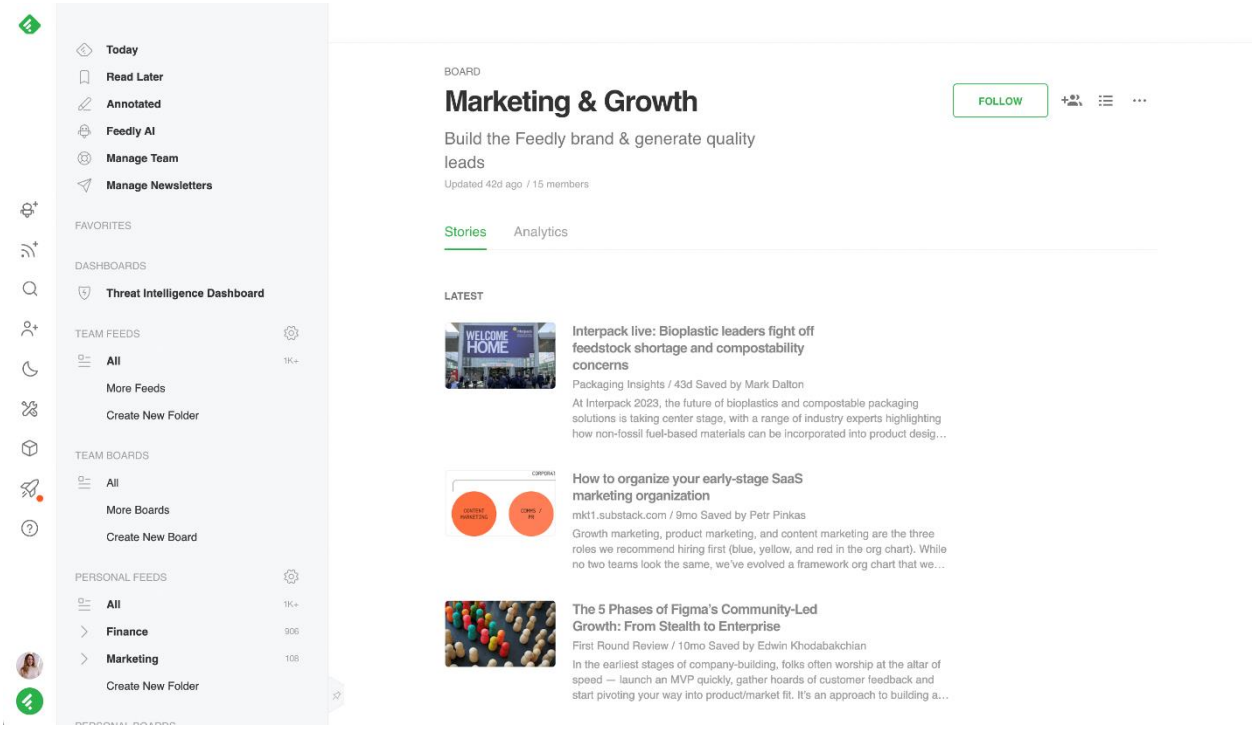


Рисунок 1.1 – Інтерфейс Feedly[2]

Inoreader пропонує широкий спектр функцій для збору та фільтрації новин. Цей сервіс підтримує RSS, Atom і JSON-фіди, а також інтеграцію з соціальними мережами та іншими сервісами. Inoreader має потужні інструменти для пошуку та організації контенту, що дозволяє користувачам ефективно управляти своїми підписками. Крім того, Inoreader надає можливість налаштування категорій та тегів для кращої організації новин. Хоча сервіс має платні плани з додатковими функціями, у безкоштовній версії відсутня реклама (рисунок 1.2).

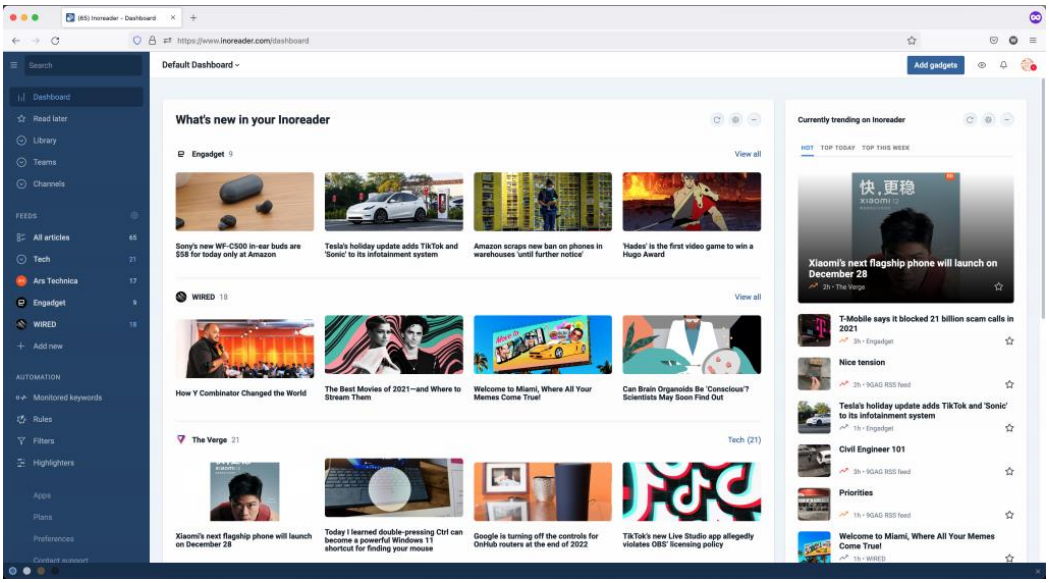


Рисунок 1.2 – Домашня сторінка Inoreader[3].

NewsBlur також є популярним агрегатором новин, який забезпечує збір новин з різних джерел і налаштування фільтрів та тегів[4]. NewsBlur дозволяє читати новини офлайн і підтримує спільноту користувачів для обговорення новин. Сервіс пропонує безкоштовну версію з обмеженим функціоналом і платну підписку для доступу до розширених можливостей.

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Агрегатор	Feedly	Inoreader	NewsBlur	Пояснення
Агрегація новин з різних джерел	Так	Так	Так	Так	Основна функція для всіх продуктів
Персоналізація контенту	Так	Так	Так	Так	Можливість налаштування стрічок новин
Інтеграція з іншими сервісами	Так	Так	Так	Ні	Підключення до інших сервісів для зручності
Підтримка командної роботи	Ні	Так	Так	Ні	Функція для колективного читання новин
Відсутність реклами	Так	Ні	У платній версії	У платній версії	Важлива перевага для комфортного користування

Продовження таблиці 1.1

Безкоштовна версія	Так	Так	Так	Так	Можливість використання продукту безкоштовно
Платна підписка	Ні	Так	Так	Так	Доступ до розширених функцій за плату

1.3 Опис бізнес-процесів

Для опису процесу реєстрації використовується BPMN модель (рисунок 1.3).

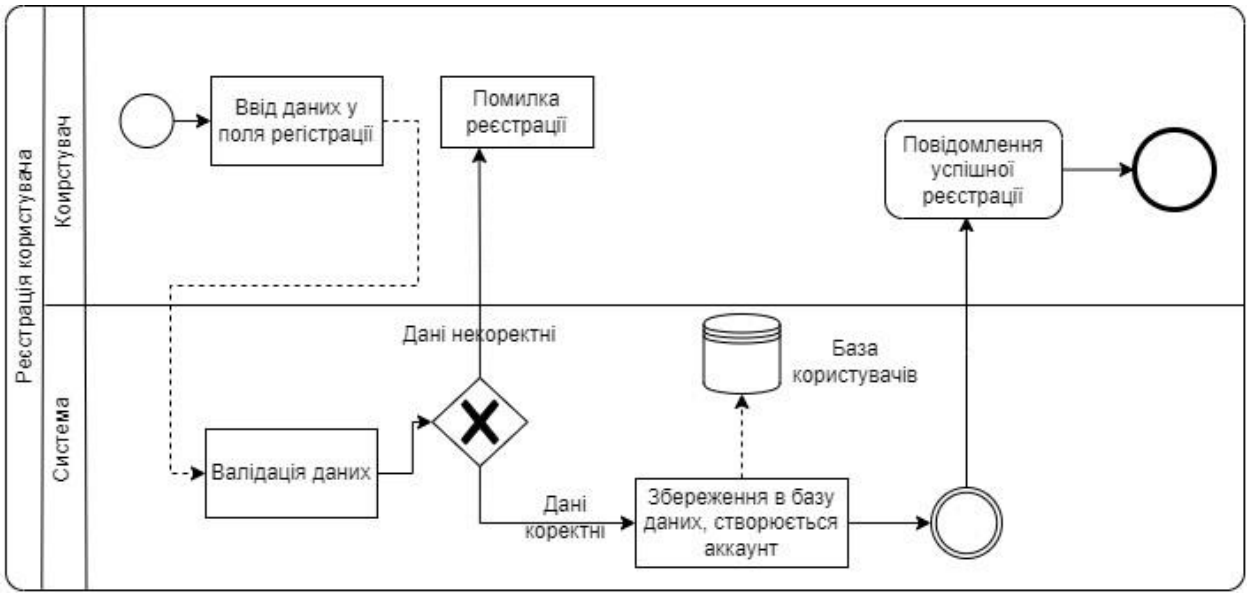


Рисунок 1.3 – Процес реєстрації користувача

Опис процесу створення облікового запису користувача:

– користувач переходить на сторінку реєстрації;

– користувач заповнює поля реєстраційної форми (електронна пошта, пароль, підтвердження пароля);

– система перевіряє коректність введених даних;

– якщо дані вірні, система зберігає їх у базі даних та створює обліковий запис;

– після успішної реєстрації користувач отримує повідомлення про успішну реєстрацію та перенаправляється на сторінку входу;

– якщо введені дані містять помилки, система виводить повідомлення про помилку реєстрації, і користувач має виправити помилки.

Для опису авторизації було використано BPMN модель (рис. 1.4).

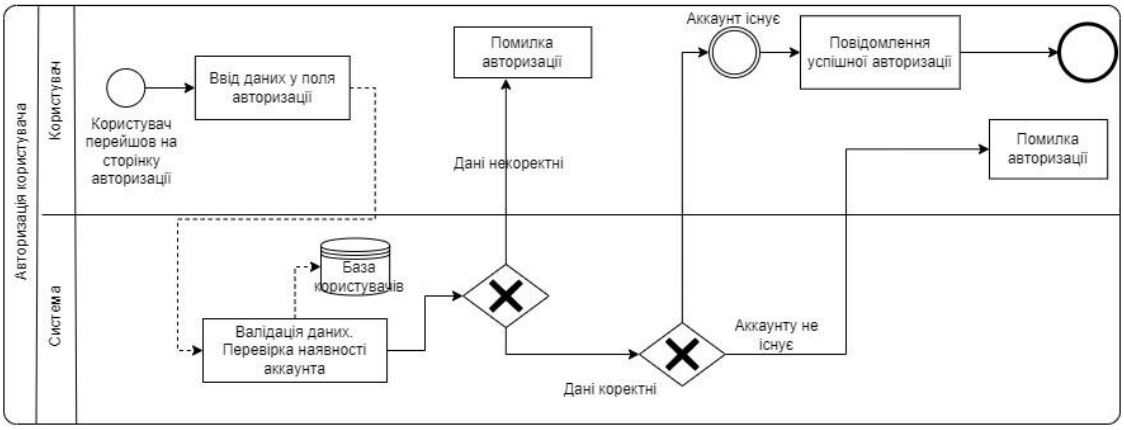


Рисунок 1.4 – Авторизація користувача

– користувач переходить на сторінку авторизації;

– вводить електронну пошту та пароль;

– система перевіряє правильність введених даних та наявність акаунта в базі даних;

– якщо дані коректні та акаунт існує, користувач отримує доступ до системи;

– у разі успішної авторизації користувач отримує повідомлення про успішну авторизацію;

– якщо введені дані некоректні або акаунт не існує, система виводить повідомлення про помилку авторизації.

Для опису оновлення новин було використано BPMN модель (рис. 1.5).

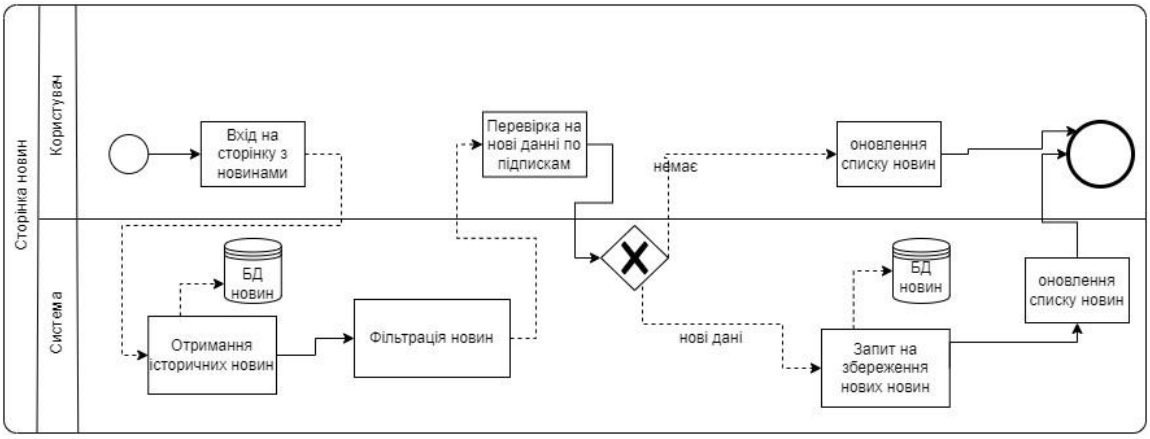


Рисунок 1.5 – Оновлення новин

– користувач входить на сторінку з новинами;

– система отримує історичні новини з бази даних;

– система перевіряє наявність нових даних по підпискам;

– якщо нові дані є, система оновлює список новин, зберігає нові дані в базу даних;

– якщо нових даних немає, система відображає історичні новини.

Для опису запиту на нову підписку було використано BPMN модель (рис. 1.6).

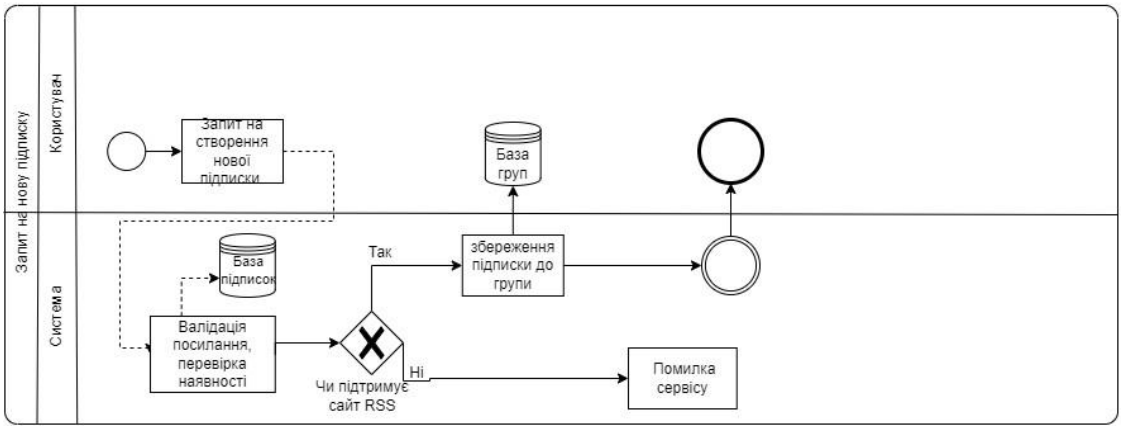


Рисунок 1.6 – Запит на нову підписку

– користувач створює запит на нову підписку, вводячи URL каналу;

– система перевіряє валідність посилання та наявність RSS-підтримки на сайті;

– якщо посилання коректне та підтримує RSS, система зберігає підписку в базу даних;

– у разі успішного додавання користувач отримує повідомлення про успішне створення підписки;

– якщо сайт не підтримує RSS або посилання некоректне, система виводить повідомлення про помилку сервісу.

1.4 Постановка задачі

Розробка призначена для автоматизації процесу збору новин з різних джерел та їх подальшого відображення в зручному для користувача інтерфейсі. Метою розробки є підвищення ефективності доступу до новин за рахунок автоматизації процесів збору, організації та фільтрації новин.

Основні задачі дипломного проекту включають:

– створення інтерфейсу (браузер): розробка сучасного, адаптивного інтерфейсу, який забезпечить зручне відображення та взаємодію з новинами на різних пристроях, включаючи комп'ютери, планшети та смартфони; інтерфейс повинен бути інтуїтивно зрозумілим і легко налаштовуваним під індивідуальні потреби користувачів;

– створення системи категорій: реалізація системи категорій для організації новин; користувачі повинні мати можливість сортувати новини за різними категоріями, що дозволить легко знаходити необхідну інформацію;

– створення системи підписок на RSS канали: розробка функціоналу для додавання, редагування та видалення підписок на RSS канали; система повинна автоматично завантажувати новини з підписаних каналів і додавати їх до стрічки новин користувача;

– створення системи фільтрації: реалізація функції фільтрації новин за різними критеріями, такими як дата публікації, джерело, категорія, автор та статус прочитаності; це дозволить користувачам швидко знаходити релевантний контент;

– створення системи управління постами: розробка системи управління постами, яка дозволить користувачам відмічати новини як прочитані, зберігати їх для подальшого прочитання, ділитися ними у соціальних мережах та відкривати оригінальні статті у нових вкладках.

Висновки до розділу

Основна мета проекту полягає у створенні веб-застосунку для персоналізованої агрегації новин, який забезпечить користувачам зручний, ефективний та безпечний доступ до релевантної інформації з різних джерел.

Додаток повинен забезпечити персоналізацію контенту, що дозволить користувачам налаштовувати свої стрічки новин відповідно до їх інтересів. Також було зазначено важливість створення зручного інтерфейсу користувача, який буде адаптивним та інтуїтивно зрозумілим, що забезпечить комфортне використання застосунку на різних пристроях, включаючи комп'ютери, планшети та смартфони.

Для забезпечення надійності та масштабованості системи передбачається розробка бекенд-системи, яка включатиме функції управління підписками, пошуку новин, додавання нових RSS-каналів та управління користувачами. Також важливим аспектом є реалізація фонових завдань, таких як оновлення стрічок новин та очищення старих записів, що забезпечить стабільну роботу застосунку.

Особлива увага приділяється питанням безпеки та приватності користувачів. Планується впровадження систем аутентифікації та авторизації користувачів, а також використання методів шифрування для захисту даних, що передаються між клієнтом та сервером. Дані користувачів не будуть використовуватися для цільової реклами, що забезпечить високий рівень приватності.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Основною функцією веб-застосунку є персоналізована агрегація новин з різних джерел, забезпечуючи користувачам зручний інтерфейс для керування своїми підписками та фільтрацією контенту. Веб агрегатор дозволяє користувачам організовувати новини за допомогою черг, додавати та видаляти підписки, а також фільтрувати та сортувати новини за різними критеріями.

Ось основні функціональні можливості системи:

– управління підписками: користувачі можуть додавати нові RSS/ATOM підписки, редагувати та видаляти існуючі підписки; після додавання підписки система автоматично завантажує нові пости з цих джерел;

– організація контенту: новини з підписок можуть бути організовані у черги, що дозволяє користувачам групувати та індексувати статті з пов'язаних джерел;

– фільтрація та пошук: користувачі можуть фільтрувати новини за різними критеріями, такими як категорії, автори, дати публікації та статус прочитаності; це забезпечує можливість швидко знаходити релевантний контент;

– інтерактивність: користувачі можуть відмічати новини як прочитані або такі, що будуть прочитані пізніше, відкривати оригінальні статті у нових вкладках та ділитися посиланнями на новини у соціальних мережах;

– персоналізація: система дозволяє користувачам налаштовувати вигляд інтерфейсу, включаючи вибір теми (світлої або темної), та створювати персоналізовані черги новин.

Головною функцією програмного забезпечення є отримання нових новин/статтей у персоналізовану стрічку, більше варіантів використання можна побачити на рисунку 2.1.

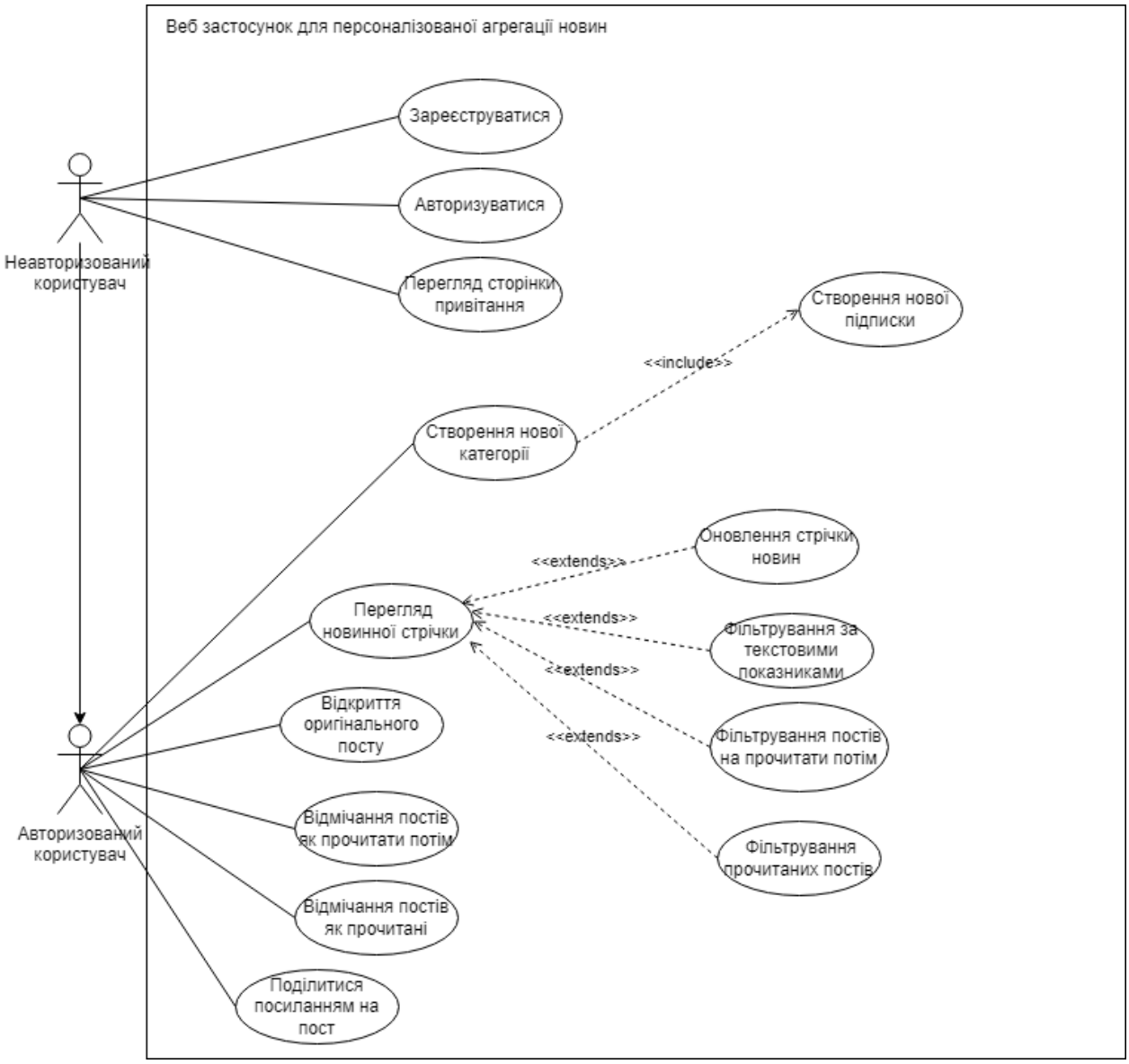


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.1 - 2.14 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі

Продовження таблиці 2.1

Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться дані: пошта користувача, пароль в системі. Після заповнення даних користувач натискає кнопку реєстрації. Після цього з’являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.
Extension	В випадку якщо користувач вже існує буде показана помилка
Post-Condition	Створення сторінки користувача, перехід на сторінку входу

Таблиця 2.2 - Варіант використання UC-2

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Вхід у систему
Actors	Користувач
Trigger	Користувач бажає авторизуватися
Pre-conditions	Наявний акаунт користувача
Flow of Events	Користувач переходить на сторінку авторизації. Вводить електронну пошту та пароль. Натискає кнопку "Login". Система перевіряє правильність даних та наявність акаунта. У разі успіху надає доступ до системи. У разі помилки виводить повідомлення про некоректні дані.
Extension	У випадку некоректних даних або відсутності акаунта, система виводить повідомлення про помилку авторизації.
Post-Condition	Вхід до системи

Таблиця 2.3 - Варіант використання UC-3

Use case name	Перегляд сторінки привітання
Use case ID	UC-03
Goals	Огляд можливостей системи
Actors	Гість
Trigger	Користувач відкриває веб-сайт
Pre-conditions	-
Flow of Events	Користувач відкриває веб-сайт. Система відображає сторінку привітання. Користувач обирає реєстрацію або авторизацію.
Extension	-
Post-Condition	Відображення сторінки привітання

Таблиця 2.4 - Варіант використання UC-4

Use case name	Створення нової черги новин
Use case ID	UC-04
Goals	Створення нової черги для організації контенту
Actors	Авторизований користувач
Trigger	Користувач бажає створити нову чергу
Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку створення нової черги. Вводить ідентифікатор, назву та опис черги. Натискає кнопку "Створити". Система створює нову чергу. Користувач отримує підтвердження.
Extension	У випадку некоретних даних система виводить повідомлення про помилку
Post-Condition	Створення нової черги

Таблиця 2.5 - Варіант використання UC-5

Use case name	Управління підписками
Use case ID	UC-05
Goals	Управління існуючими підписками
Actors	Авторизований користувач
Trigger	Користувач бажає керувати підписками
Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку управління підписками. Обирає підписку для редагування або видалення. Вносить зміни або видаляє підписку. Система оновлює інформацію про підписки. Користувач отримує підтвердження про успішне оновлення.
Extension	Якщо підписка не може бути знайдена або видалена, система виводить відповідне повідомлення
Post-Condition	Оновлення або видалення підписки

Таблиця 2.6 – Варіант використання UC-6

Use case name	Додавання нової підписки
Use case ID	UC-06
Goals	Додавання нового RSS-каналу до стрічки новин
Actors	Авторизований користувач
Trigger	Користувач бажає додати новий канал
Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку додавання нової підписки. Вводить URL RSS-каналу. Натискає кнопку "Discovery". Система перевіряє валідність URL і підтримку RSS каналів. У разі успіху користувач має можливість натиснути на "Subscribe" що би додати канал до підписок користувача.

Продовження таблиці 2.6

Extension	У випадку некоректного URL або неможливості отримання інформації по RSS система виводить повідомлення про помилку
Post-Condition	Додавання нового каналу

Таблиця 2.7 – Варіант використання UC-7

Use case name	Оновлення стрічки новин
Use case ID	UC-07
Goals	Отримання нових постів із підписаних каналів
Actors	Авторизований користувач
Trigger	Користувач бажає оновити стрічку новин
Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку новин і натискає кнопку Refresh. Система отримує нові пости з підписаних каналів. Відображає нові пости користувачу.
Extension	У випадку відсутності нових постів система виводить останні пости збережені в системі
Post-Condition	Оновлення стрічки новин

Таблиця 2.8 – Варіант використання UC-8

Use case name	Відмічання постів як прочитані
Use case ID	UC-08
Goals	Відмітка постів як прочитані
Actors	Авторизований користувач
Trigger	Користувач бажає відмітити пост як прочитаний
Pre-conditions	Авторизація

Продовження таблиці 2.8

Flow of Events	Користувач обирає пост у стрічці новин. Натискає кнопку Mark as Read. Система оновлює статус посту.
Extension	-
Post-Condition	Оновлення статусу посту

Таблиця 2.9 – Варіант використання UC-9

Use case name	Відмічання постів як прочитати потім
Use case ID	UC-09
Goals	Відмітка постів як прочитати потім
Actors	Авторизований користувач
Trigger	Користувач бажає відмітити пост як прочитати потім
Pre-conditions	Авторизація
Flow of Events	Користувач обирає пост у стрічці новин. Натискає кнопку Read later. Система оновлює статус посту.
Extension	-
Post-Condition	Оновлення статусу посту

Таблиця 2.10 – Варіант використання UC-10

Use case name	Відкриття оригінального посту
Use case ID	UC-10
Goals	Перейти до першоджерела
Actors	Авторизований користувач
Trigger	Користувач бажає відкрити оригінал посту
Pre-conditions	Авторизація

Продовження таблиці 2.10

Flow of Events	Користувач обирає пост у стрічці новин. Натискає кнопку Open original article. Система перенаправляє користувача до оригінального посту у новій вкладці.
Extension	-
Post-Condition	Відкриття оригінального посту

Таблиця 2.11 – Варіант використання UC-11

Use case name	Поділитися посиланнямм
Use case ID	UC-11
Goals	Поділитися посиланням на пост у соціальних мережах
Actors	Авторизований користувач
Trigger	Користувач бажає поділитися посиланням
Pre-conditions	Авторизація
Flow of Events	Користувач обирає пост у стрічці новин. Натискає кнопку Share a post та обирає соціальну мережу. Система генерує посилання та перенаправляє у соціальну мережу.
Extension	У випадку помилки отримання посилання посту система виводить повідомлення про помилку
Post-Condition	Поділитися посиланням у соціальній мережі

Таблиця 2.12 – Варіант використання UC-12

Use case name	Фільтрування прочитаних постів
Use case ID	UC-12
Goals	Фільтрування постів, щоб відобразити прочитані
Actors	Авторизований користувач
Trigger	Користувач бажає фільтрувати пости, відмічені як «Read»

Продовження таблиці 2.12

Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку новин. Обирає фільтр "Read". Система відображає лише пости, відмічені як "Read"
Extension	-
Post-Condition	Відображення постів відмічених як "Read"

Таблиця 2.13 - Варіант використання UC-13

Use case name	Фільтрування прочитаних постів
Use case ID	UC-13
Goals	Фільтрування постів, щоб відобразити відмічені для прочитання пізніше
Actors	Авторизований користувач
Trigger	Користувач бажає фільтрувати пости, відмічені як "Read later"
Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку новин. Обирає фільтр "Read later". Система відображає лише пости, відмічені як "Read later"
Extension	-
Post-Condition	Відображення постів відмічених як "Read later"

Таблиця 2.14 - Варіант використання UC-14

Use case name	Фільтрування за показниками
Use case ID	UC-14
Goals	Фільтрування постів за текстовими показниками
Actors	Авторизований користувач

Продовження таблиці 2.14

Trigger	Користувач бажає задати правила фільтрації для відображення релевантних постів
Pre-conditions	Авторизація
Flow of Events	Користувач переходить на сторінку новин. Вводить текстові або тег параметри для фільтрування. Система відображає пости відповідно до параметрів фільтрування.
Extension	-
Post-Condition	Відображення постів відповідно до параметрів фільтрування

2.2 Аналіз системних вимог

Вимоги до серверного обладнання:

– процесор: мінімум двоядерний 2 ГГц, рекомендовано чотириядерний 3 ГГц;

– оперативна пам'ять: мінімум 4 ГБ, рекомендовано 8 ГБ і більше;

– дисковий простір: мінімум 20 ГБ, рекомендовано 50 ГБ і більше;

– мережеве підключення: мінімум 20 Мбіт/с, рекомендовано 100 Мбіт/с.

Вимоги до програмного забезпечення серверу:

– середовище виконання: Java Development Kit (JDK) 11 або вище;

– база даних: мінімум PostgreSQL 10, рекомендовано PostgreSQL 12 або новіша версія;

– веб-сервер: мінімум Apache Tomcat 9, рекомендовано Apache Tomcat 10.

Вимоги до клієнтського обладнання:

– процесор: мінімум двоядерний 1.6 ГГц, рекомендовано чотириядерний 2.4 ГГц;

– оперативна пам'ять: мінімум 2 ГБ, рекомендовано 4 ГБ і більше;

– дисковий простір: мінімум 500 МБ, рекомендовано 1 ГБ і більше;

– мережеве підключення: мінімум 5 Мбіт/с, рекомендовано 20 Мбіт/с і більше.

Вимоги до програмного забезпечення клієнта:

– операційна система: мінімум Windows 10, macOS Mojave, Linux;

– браузер: мінімум Google Chrome 80, Mozilla Firefox 75, Microsoft Edge 80.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.15 наведено загальну модель вимог, а в таблиці 2.16 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.3.

Таблиця 2.15 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Перегляд сторінки привітання	FR-1	Низький	Низький
2	Реєстрація користувача	FR-2	Високий	Високий
3	Авторизація користувача	FR-3	Високий	Високий
4	Вихід із акаунту	FR-4	Середній	Низький
5	Додавання нової підписки	FR-5	Високий	Середній
6	Управління підписками	FR-6	Високий	Середній
7	Завантаження нових постів	FR-7	Високий	Високий
8	Відмічання постів як прочитані	FR-8	Середній	Низький
9	Відмічання постів як прочитати потім	FR-9	Середній	Низький
10	Відкриття оригіналу посту	FR-10	Середній	Низький

Продовження таблиці 2.15

11	Поділитися посиланням	FR-11	Середній	Низький
12	Фільтрування прочитаних постів	FR-12	Середній	Низький
13	Фільтрування постів на прочитати потім	FR-13	Середній	Низький
14	Фільтрування за текстовими показниками	FR-14	Середній	Низький
16	Фільтрація по основним категоріям	FR-15	Середній	Низький

Таблиця 2.16 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Неавторизований користувач бачить веб-сторінку привітання. На цій сторінці є можливість зареєструватися або авторизуватися.
FR-2	Реєстрація користувача. Гості можуть створити новий обліковий запис, заповнивши форму реєстрації з електронною поштою, паролем та підтвердженням пароля.
FR-4	Користувачі можуть увійти до системи, використовуючи свої облікові дані (електронна пошта та пароль). Після успішної авторизації користувач перенаправляється на головну сторінку.
FR-5	Вихід із акаунту. Користувач може вийти з системи, натиснувши кнопку "Вийти". Після виходу сесія користувача завершується.
FR-6	Додавання нової підписки. Користувачі можуть додавати нові RSS/ATOM підписки, вводячи URL-адресу підписки. Система перевіряє валідність URL та додає підписку до списку користувача.

Продовження таблиці 2.16

FR-6	Додавання нової підписки. Користувачі можуть додавати нові RSS/ATOM підписки, вводячи URL-адресу підписки. Система перевіряє валідність URL та додає підписку до списку користувача.
FR-7	Управління підписками. Користувачі можуть редагувати та видаляти свої підписки. Кожна підписка може бути налаштована індивідуально.
FR-8	Завантаження нових постів. Система автоматично завантажує нові пости з підписаних каналів та відображає їх у стрічці новин користувача.
FR-9	Відмічання постів як прочитані. Користувачі можуть відмічати пости як прочитані, змінюючи їх статус у системі.
FR-10	Відмічання постів як прочитати потім. Користувачі можуть відмічати пости як такі, що будуть прочитані пізніше, змінюючи їх статус у системі.
FR-11	Відкриття оригіналу посту. Користувачі можуть відкривати оригінал посту у новій вкладці браузера.
FR-12	Поділитися посиланням. Користувачі можуть ділитися посиланнями на пости у соціальних мережах або через інші засоби комунікації.
FR-13	Фільтрування прочитаних постів. Користувачі можуть фільтрувати стрічку новин, відображаючи лише прочитані пости.
FR-14	Фільтрування постів на прочитати потім. Користувачі можуть фільтрувати стрічку новин, відображаючи лише пости, відмічені як такі, що будуть прочитані пізніше.
FR-15	Фільтрування за текстовими показниками. Користувачі можуть фільтрувати стрічку новин за різними текстовими параметрами, такими як ключові слова або категорії.

Продовження таблиці 2.16

FR-16	Фільтрація по основним категоріям. Користувачі можуть фільтрувати новини за основними категоріями, такими як subscriptionId, feed, category, title, author, published, updated, contents, та status.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15
UC-01	x	x													
UC-02			x												
UC-03	x														
UC-04					x										
UC-05						x									
UC-06					x		x								
UC-07							x								
UC-08								x							
UC-09									x						
UC-10										x					
UC-11											x				
UC-12												x			x
UC-13													x		x
UC-14														x	x

Рисунок 2.2 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Продуктивність: Система повинна забезпечувати час відгуку не більше 2 секунд для більшості операцій користувача, включаючи додавання нових підписок, завантаження нових постів та фільтрацію новин. Система повинна обробляти щонайменше 1000 запитів на хвилину без значного зниження продуктивності.

Зручність використання: Інтерфейс користувача має бути інтуїтивно зрозумілим і простим у використанні, з мінімальною кількістю кроків для виконання основних операцій.

Висновки до розділу

У цьому розділі було детально розглянуто варіанти використання програмного забезпечення, його функціональні та нефункціональні вимоги.

По-перше, було описано головні функціональні можливості веб-застосунку. Основними з них є управління підписками, організація контенту за допомогою черг, фільтрація та пошук новин за різними критеріями, а

також інтерактивні можливості для користувачів, такі як відмічання новин, відкриття оригінальних статей і поділ посиланнями у соціальних мережах. Персоналізація інтерфейсу користувача також була відзначена як ключова функція.

Було розглянуто 14 варіантів використання ,які охоплюють основні сценарії взаємодії користувача із системою. Ці варіанти включають реєстрацію та авторизацію користувачів, створення нових черг, додавання та управління підписками, оновлення стрічки новин, фільтрацію контенту та інші дії, що забезпечують ефективне використання платформи. Кожен варіант використання детально описаний із зазначенням цілей, акторів, тригерів, попередніх умов, послідовності дій, розширень та постумов.

Також був проведений аналіз системних вимог, які включають апаратні та програмні вимоги як до серверного, так і до клієнтського обладнання. Цей аналіз допоможе забезпечити належну продуктивність та стабільність системи.

Описано 16 функціональних вимог, які включають такі важливі функції, як перегляд сторінки привітання, реєстрація та авторизація користувачів, додавання нових підписок, управління підписками, завантаження нових постів, фільтрація новин за різними критеріями тощо. Матриця трасування вимог була створена для відображення взаємозв'язків між функціональними вимогами та варіантами використання.

Також були визначені основні нефункціональні вимоги, які охоплюють продуктивність, масштабованість, безпеку, зручність використання, надійність і тестованість системи. Ці вимоги забезпечують, що система буде не тільки функціональною, але й зручною, швидкою, безпечною та надійною у використанні.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Новинний агрегатор має багатошарову архітектуру, що дозволяє розділити різні аспекти системи на окремі компоненти для полегшення розробки, підтримки та масштабування.

Система складається з клієнтської частини та серверної частини, модуля обробки даних та бази даних.

Клієнтська частина реалізована за допомогою Vue3 та Vuetify, забезпечуючи сучасний інтерфейс користувача для взаємодії зі стрічками новин. Для підвищення продуктивності та користувацького досвіду реалізовано повнотекстовий пошук.

Серверна частина побудована на Spring Boot і забезпечує REST API для взаємодії з клієнтською частиною та іншими компонентами. Контролери серверної частини відповідають за управління користувачами, стрічками новин та підписками. Наприклад, StagingPostController управляє операціями з постами, QueueDefinitionController обробляє черги новин, а FeedDiscoveryController займається виявленням нових стрічок новин.

Модуль обробки даних виконує періодичні завдання, такі як імпорт нових постів з RSS стрічок та очищення застарілих даних. Ці завдання реалізовані за допомогою планувальника задач Spring, який забезпечує регулярне оновлення контенту. Кожні декілька секунд модуль обробки даних перевіряє наявність нових постів у підписаних RSS стрічках і додає їх у базу даних.

База даних використовує PostgreSQL для зберігання даних користувачів, підписок, постів та інших метаданих. Для зберігання даних було створено кілька таблиць, включаючи users, queue_definitions, subscription_definitions, staging_posts та інші. Ця структура дозволяє ефективно управляти даними користувачів та стрічок новин.

Управління користувачами здійснюється через систему авторизації, де облікові записи користувачів зберігаються в базі даних. Користувачі реєструються та авторизуються за допомогою стандартної схеми аутентифікації. Під час логіну перевіряються облікові дані користувача, а потім генеруються JWT токени для забезпечення безпеки сесій. Це дозволяє забезпечити безпечний доступ до ресурсів та захистити дані користувачів.

Додаток не інтегрується напряму з іншими системами, але використовує RSS канали для отримання контенту. RSS канали забезпечують стандартний протокол для отримання новин та оновлень з різних джерел. Тобто якщо ресурс підтримує RSS або Atom то ми зможемо без проблем отримувати дані. Це дозволяє автоматично імпортувати новий контент і відображати його користувачам.

Безпека досягається через використання JWT токенів для аутентифікації користувачів. Масштабованість забезпечується багатошаровою архітектурою, що дозволяє легко масштабувати окремі компоненти, такі як серверну частину та базу даних. Продуктивність покращується за рахунок використання Redis для кешування, що зменшує навантаження на базу даних. Висока доступність системи забезпечується завдяки використанню надійних технологій, таких як Spring Boot та PostgreSQL.

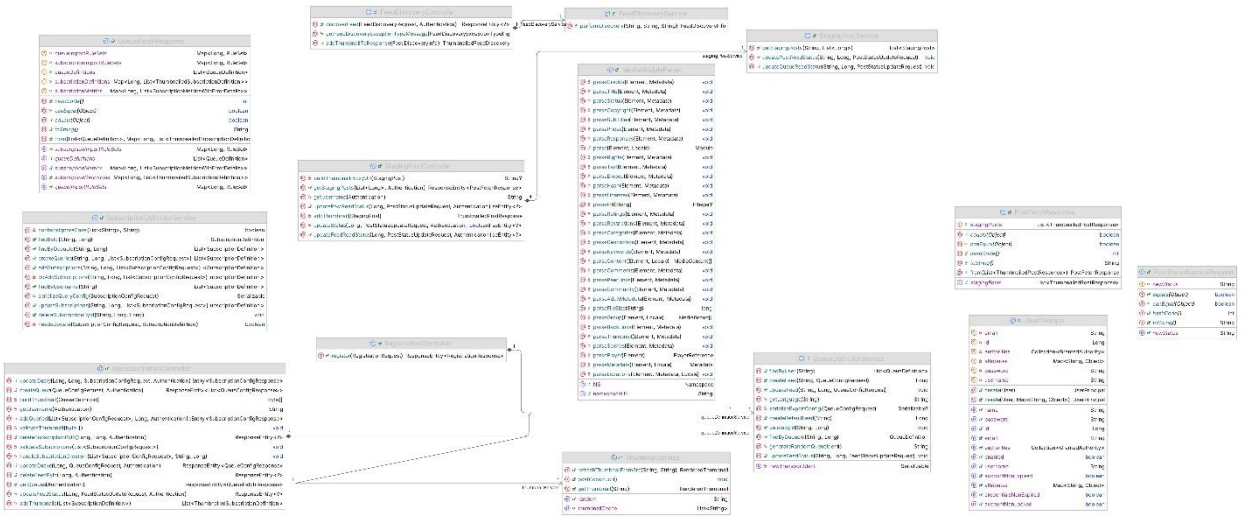


Рисунок 3.1 – Діаграма класів.

У таблиці 3.2 наведений опис класів, наведених на рисунку 3.1.

Таблиця 3.2 – Опис призначення класів

Назва класу	Опис класу
FeedDiscoveryController	Відповідає за виявлення нових каналів і додавання мініатюр до відповіді.
FeedDiscoveryService	Виконує виявлення каналів за заданими параметрами.
MediaModuleParser	Парсер для медіамодулів, який обробляє різні типи метаданих і медіа вмісту.
PostFetchResponse	Клас для формування відповіді на запит отримання постів, включаючи інформацію про мініатюри постів.
PostStatusUpdateRequest	Клас для запитів на оновлення статусу постів.
QueueDefinitionController	Відповідає за створення, оновлення і видалення черг, а також за додавання запитів і оновлення статусу черг.
QueueDefinitionService	Сервіс для обробки бізнес-логіки, пов’язаної з чергами, включаючи створення, оновлення і видалення черг.
QueueFetchResponse	Клас для формування відповіді на запит отримання черг, включаючи інформацію про підписки і метрики черг.
RegistrationController	Відповідає за управління процесами реєстрації і верифікації користувачів.
StagingPostController	Відповідає за управління чернетками постів, включаючи оновлення статусу постів і отримання чернеток.

Продовження таблиці 3.2

StagingPostService	Сервіс для обробки чернеток постів, включаючи оновлення статусу постів і отримання чернеток для користувачів.
SubscriptionDefinitionService	Сервіс для управління підписками, включаючи створення, оновлення і видалення підписок.
ThumbnailService	Відповідає за обробку мініатюр, включаючи отримання і оновлення мініатюр з джерел.
AuthService	Сервіс для обробки логін запитів

Таблиця 3.3 – Опис призначення методів

Назва методу	Призначення методу	Вхідні дані	Вихідні дані
getQueues	Отримання списку черг для користувача, включаючи інформацію про підписки та метрики черг.	Параметри аутентифікації	Список черг
createQueue	Створення нової черги для організації новин. Метод обробляє запит користувача на створення нової черги та додає її до системи.	Запит на створення черги, параметри аутентифікації	Список черг
addQueries	Додавання нових підписок до існуючої черги. Метод перевіряє валідність підписок та додає їх до черги користувача.	Список запитів, ID черги, параметри аутентифікації	Список підписок

Продовження таблиці 3.3

updateQueue	Оновлення існуючої черги за її ID. Метод обробляє запит на оновлення інформації черги, включаючи назву, опис та інші параметри.	ID черги, запит на оновлення, параметри аутентифікації	Оновлена черга
updateQuery	Оновлення підписки за її ID. Метод обробляє запит на оновлення інформації підписки, такої як URL, заголовок, зображення тощо.	ID черги, ID підписки, запит на оновлення, параметри аутентифікації	Оновлена підписка
updateFeedStatus	Оновлення статусу каналу. Метод змінює статус каналу, наприклад, на активний або архівний.	ID каналу, запит на оновлення статусу, параметри аутентифікації	Підтвердження оновлення
deleteFeedById	Видалення черги за її ID. Метод обробляє запит користувача на видалення черги та видаляє її з системи.	ID черги, параметри аутентифікації	Підтвердження видалення
deleteSubscriptionById	Видалення підписки за її ID. Метод обробляє запит на видалення підписки з черги користувача.	ID черги, ID підписки, параметри аутентифікації	Підтвердження видалення

Продовження таблиці 3.3

discoverFeed	Виявлення каналу за наданим URL. Метод перевіряє URL на наявність каналу та повертає інформацію про знайдений каналу.	Запит на виявлення каналу, параметри аутентифікації	Інформація про фід
performDiscovery	Виконання процесу виявлення фіду. Метод здійснює аналіз наданого URL для виявлення фіду, використовуючи вказані параметри аутентифікації.	URL, ім'я користувача, пароль	Інформація про виявлення фіду
findById	Знаходження підписки за її ID. Метод повертає інформацію про конкретну підписку, пов'язану з користувачем.	Ім'я користувача, ID	Інформація про підписку
findByUsername	Знаходження всіх підписок для користувача. Метод повертає список усіх підписок, зареєстрованих для даного користувача.	Ім'я користувача	Список підписок
findByQueueId	Знаходження підписок в черзі за її ID. Метод повертає список усіх підписок, пов'язаних з конкретною чергою користувача.	Ім'я користувача, ID черги	Список підписок

Продовження таблиці 3.3

updateSubscriptions	Оновлення підписок в черзі за заданими параметрами. Метод обробляє запити на оновлення інформації підписок в межах конкретної черги.	Ім'я користувача, ID черги, список запитів	Оновлені підписки
addSubscriptions	Додавання нових підписок до черги. Метод перевіряє валідність підписок та додає їх до черги користувача.	Ім'я користувача, ID черги, список запитів	Список доданих підписок
deleteSubscriptionById	Видалення підписки з черги за її ID. Метод обробляє запит користувача на видалення підписки з черги.	Ім'я користувача, ID черги, ID підписки	Підтвердження видалення
register	Реєстрація нового користувача в системі. Метод обробляє запит на реєстрацію, створює новий обліковий запис користувача та генерує верифікаційний токен.	Запит на реєстрацію	Підтвердження реєстрації
createFeed	Створення нової черги для користувача. Метод обробляє запит на створення нової черги та додає її до системи.	Ім'я користувача, запит на створення черги	ID створеної черги

Для реалізації додатку було обрано технологічний стек, який включає Vue3, Vuetify, Spring Boot, PostgreSQL, Redis, Spring Scheduler, JWT та Maven.

Vue3 та Vuetify були обрані для клієнтської частини завдяки їх можливостям створювати сучасні, інтуїтивно зрозумілі інтерфейси користувача, що підтримують адаптивний дизайн.

Spring Boot використовується для серверної частини, забезпечуючи потужні інструменти для розробки REST API та управління бізнес-логікою, а також легку інтеграцію з іншими компонентами.

PostgreSQL була вибрана як база даних за її надійність, масштабованість та підтримку ACID транзакцій, що гарантує цілісність даних.

Redis використовується для кешування, що підвищує продуктивність системи шляхом зменшення навантаження на базу даних.

Spring Scheduler відповідає за періодичне виконання завдань, таких як оновлення контенту, що дозволяє підтримувати актуальність даних.

JWT забезпечують безпеку користувацьких сесій через надійний механізм аутентифікації та авторизації.

Maven використовується для управління збіркою проекту, що спрощує керування залежностями та налаштування проекту.

Обраний технологічний стек дозволяє створити ефективну, масштабовану та легко підтримувану систему для агрегування та відображення новин.

3.2 Обґрунтування вибору засобів розробки

Для розробки додатку було використано кілька допоміжних програмних засобів, які сприяли ефективній реалізації проекту. Оскільки існує багато аналогів для кожного з вибраних інструментів, важливо розглянути їхні особливості та зробити порівняльний аналіз, щоб обґрунтувати наш вибір.

Для клієнтської частини ми обрали Vue3 у поєднанні з Vuetify. Vue3 пропонує зручну та потужну архітектуру для створення інтерфейсів користувача з високою продуктивністю. Vuetify, як набір компонентів для Vue, дозволяє швидко створювати елегантні інтерфейси з готовими UI-елементами. Інші популярні фреймворки, такі як React або Angular, також могли бути використані, але Vue3 був обраний за його простоту, легкість навчання та високу продуктивність.

Redis використовується для кешування, що дозволяє підвищити продуктивність додатку, зменшуючи навантаження на базу даних. Інші системи кешування, такі як Memcached, також могли бути розглянуті, але Redis пропонує більш багатий набір функцій, включаючи підтримку складніших типів даних і можливість виконання атомарних операцій[5].

Допоміжні бібліотеки:

– ROME
– Lunr.js

ROME була обрана для роботи з RSS та Atom стрічками завдяки її простоті у використанні та потужним можливостям парсингу[6]. Вона дозволяє легко інтегруватися з різними форматами стрічок і має потужні засоби для генерації та читання RSS/Atom. Альтернативи, такі як Feedparser, можуть бути ефективними для інших мов програмування, але ROME забезпечує більшу гнучкість і є добре інтегрованою з Java-екосистемою, що робить її ідеальним вибором для нашого проекту.

Lunr.js використовується для реалізації повнотекстового пошуку на клієнтській стороні. Він обраний за свою легкість і ефективність. Lunr.js дозволяє індексувати документи і здійснювати пошук за текстом без необхідності в додаткових серверних ресурсах[7]. Альтернативи, такі як Elasticsearch або Solr, могли б бути використані для серверного пошуку, але вони більш складні у налаштуванні та вимагають додаткових серверних ресурсів. Lunr.js забезпечує достатню функціональність для нашого випадку, при цьому залишаючись легкою бібліотекою, що працює безпосередньо в

браузері, що дозволяє знизити навантаження на сервер і забезпечити швидкий пошук для користувачів.

Для розробки серверної частини ми використовуємо IntelliJ IDEA Ultimate, яке є потужним середовищем розробки для Java [8]. IntelliJ IDEA забезпечує інтелектуальне автозавершення коду, рефакторинг, відлагодження та інтеграцію з системами керування версіями, що значно підвищує продуктивність розробників. Також вона підтримує генерацію класових діаграм прямо з коду, що дуже полегшує процес документації. Як альтернатива могла бути використана Eclipse або NetBeans, які також є популярними середовищами розробки для Java. Проте IntelliJ IDEA надає більш зручний інтерфейс та потужніші інструменти для підвищення ефективності розробки.

Для клієнтської частини ми використовуємо WebStorm, яке є ідеальним середовищем розробки для JavaScript та Vue.js. WebStorm надає інтеграцію з популярними фреймворками, такими як Vue.js, React та Angular, а також підтримує різні інструменти для відлагодження та тестування коду[9]. Як альтернатива могла бути використана Visual Studio Code, яка також є популярним вибором серед веб-розробників.

3.3 Конструювання програмного забезпечення

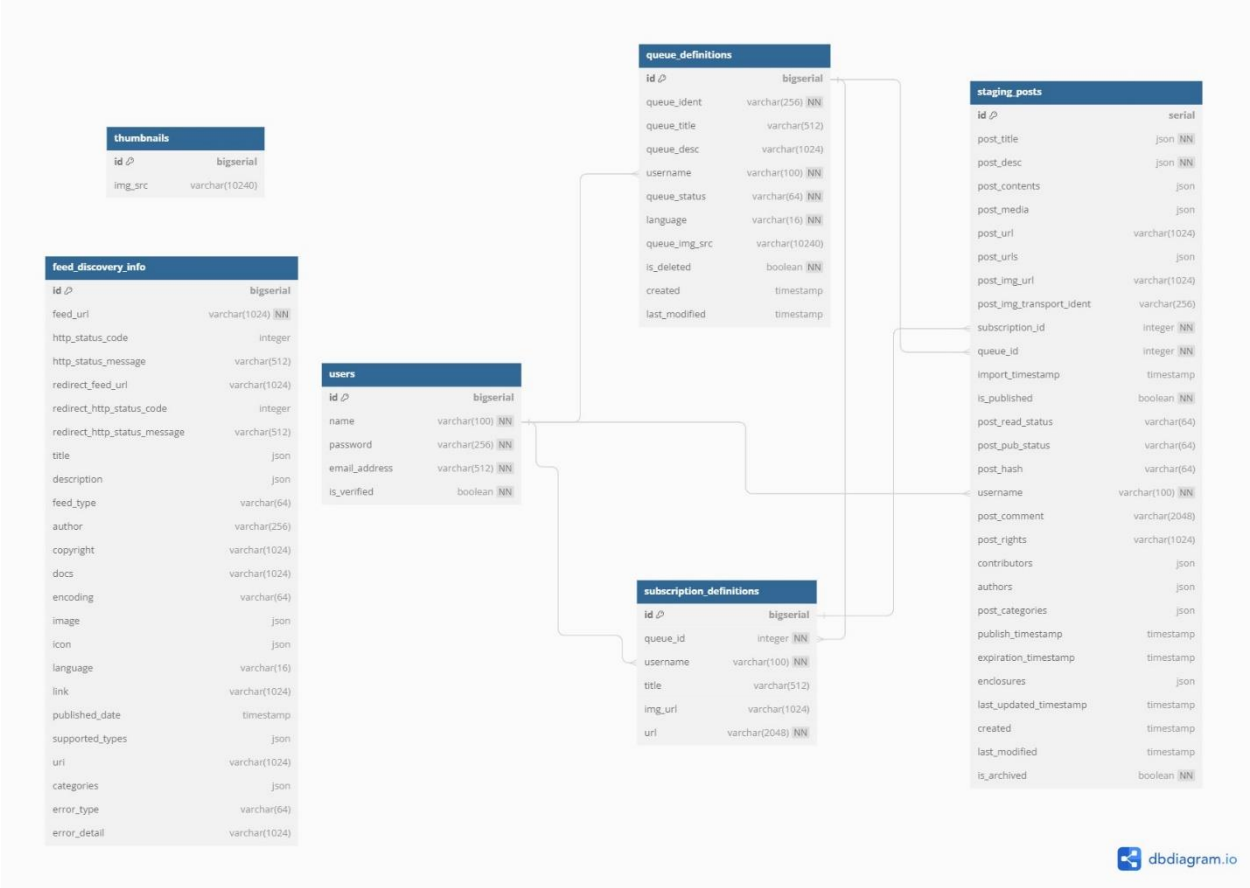


Рисунок 3.3 – ER діаграма сутностей

Система управління базами даних, що використовується, - це Postgres. База даних сервера призначена для зберігання інформації про користувачів та їх підписки. Опис таблиць бази даних представлено у таблицях 3.4 – 3.11, а модель бази даних показана на рисунку 3.3.

Таблиця users використовується для зберігання даних про користувачів системи. Вона містить інформацію про ідентифікаційний номер користувача, його ім’я, хеш пароля, електронну пошту та інші атрибути, пов’язані з автентифікацією та верифікацією користувачів.

Таблиця 3.4 – Опис таблиці user

Назва поля	Тип даних	Опис
id	bigserial	Ідентифікаційний номер користувача

Продовження таблиці 3.4

name	varchar(100)	Унікальне ім’я користувача
password	varchar(256)	Хеш пароля користувача
email_address	varchar(512)	Унікальна електронна пошта користувача
auth_claim	varchar(256)	Твердження аутентифікації
verification_claim	varchar(256)	Твердження для верифікації
is_verified	boolean	Статус верифікації користувача

Таблиця queue_definitions використовується для зберігання інформації про черги новин, створені користувачами. Вона містить дані про ідентифікатор черги, її назву, опис, статус

Таблиця 3.5 – Опис таблиці queue_definitions

Назва поля	Тип даних	Опис
id	bigserial	Ідентифікаційний номер черги
queue_ident	varchar(256)	Унікальний ідентифікатор черги
queue_title	varchar(512)	Назва черги
queue_desc	varchar(1024)	Опис черги
username	varchar(100)	Ім’я користувача, який створив чергу
queue_status	varchar(64)	Статус черги
language	varchar(16)	Мова черги
queue_img_src	varchar(10240)	Джерело зображення черги

Продовження таблиці 3.5

is_deleted	boolean	Статус видалення черги
created	timestamp with time zone	Час створення черги
last_modified	timestamp with time zone	Час останньої модифікації черги

Таблиця 3.6 – Опис таблиці api_keys

Назва поля	Тип даних	Опис
Id	bigserial	Ідентифікаційний номер ключа API
user_id	integer	Посилання на користувача
api_key	varchar(512)	Ключ API
api_secret	varchar(512)	Секрет API
application_id	varchar(32)	Ідентифікатор застосунку

Таблиця subscription_definitions використовується для зберігання підписок на новини. Вона містить дані про ідентифікатор підписки, пов'язану чергу, ім'я користувача, назву підписки, URL, тип запиту та розклад імпорту.

Таблиця 3.7 – Опис таблиці subscription_definitions

Назва поля	Тип даних	Опис
id	bigserial	Ідентифікаційний номер підписки
queue_id	integer	Посилання на чергу
username	varchar(100)	Ім’я користувача
title	varchar(512)	Назва підписки

Продовження таблиці 3.7

img_url	varchar(1024)	URL зображення підписки
url	varchar(2048)	URL підписки
query_type	varchar(64)	Тип запиту
import_schedule	varchar(32)	Розклад імпорту
query_config	json	Конфігурація запиту

Таблиця staging_posts використовується для зберігання нових постів, які ще не були опубліковані. Вона містить дані про заголовок посту, опис, URL, статус публікації та інші метадані, що стосуються посту.

Таблиця 3.8 – Опис таблиці staging_posts

Id	serial	Ідентифікаційний номер посту
post_title	json	Заголовок посту
post_desc	json	Опис посту
post_contents	json	Вміст посту
post_media	json	Медіафайли посту
post_url	varchar(1024)	URL посту
post_urls	json	Список URL посту
post_img_url	varchar(1024)	URL зображення посту
post_img_transport_ident	varchar(256)	Ідентифікатор транспорту зображення
subscription_id	integer	Посилання на підписку
queue_id	integer	Посилання на чергу

Продовження таблиці 3.8

import_timestamp	timestamp with time zone	Час імпорту
is_published	boolean	Статус публікації
post_read_status	varchar(64)	Статус прочитання посту
post_pub_status	varchar(64)	Статус публікації посту
post_hash	varchar(64)	Хеш посту
username	varchar(100)	Ім'я користувача
post_comment	varchar(2048)	Коментар до посту
post_rights	varchar(1024)	Права на пост
contributors	json	Автори посту
authors	json	Автори посту
post_categories	json	Категорії посту
publish_timestamp	timestamp with time zone	Час публікації посту
expiration_timestamp	timestamp with time zone	Час закінчення терміну дії посту
enclosures	json	Додатки до посту
last_updated_timestamp	timestamp with time zone	Час останнього оновлення посту
created	timestamp with time zone	Час створення посту

Продовження таблиці 3.8

last_modified	timestamp with time zone	Час останньої модифікації посту
is_archived	boolean	Статус архівування посту

Таблиця feed_discovery_info використовується для зберігання інформації про виявлені RSS стрічки. Ця таблиця містить дані про URL стрічок, HTTP статуси, метадані стрічок та інші важливі параметри, що допомагають у процесі виявлення та аналізу стрічок новин.

Таблиця 3.9 – Опис таблиці feed_discovery_info

id	bigserial	Ідентифікаційний номер інформації про стрічку
feed_url	varchar(1024)	URL стрічки
http_status_code	integer	HTTP статус код
http_status_message	varchar(512)	HTTP статус повідомлення
redirect_feed_url	varchar(1024)	URL перенаправлення
redirect_http_status_code	integer	HTTP статус код перенаправлення
redirect_http_status_message	varchar(512)	HTTP статус повідомлення перенаправлення
title	json	Заголовок стрічки
description	json	Опис стрічки
feed_type	varchar(64)	Тип стрічки
author	varchar(256)	Автор стрічки

Продовження таблиці 3.9

copyright	varchar(1024)	Правовласник стрічки
docs	varchar(1024)	Документація стрічки
encoding	varchar(64)	Кодування стрічки
image	json	Зображення стрічки
icon	json	Іконка стрічки
language	varchar(16)	Мова стрічки
link	varchar(1024)	Посилання на стрічку
published_date	timestamp with time zone	Дата публікації стрічки
supported_types	json	Підтримувані типи стрічки
uri	varchar(1024)	URI стрічки
categories	json	Категорії стрічки
error_type	varchar(64)	Тип помилки
error_detail	varchar(1024)	Деталі помилки
id	bigserial	Ідентифікаційний номер зображення
img_src	varchar(10240)	Джерело зображення

Інформація про утиліти, бібліотеки та інше стороннє програмне забезпечення, що застосовується в процесі розробки, представлена в таблиці 3.10.

Таблиця 3.10 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	IntelliJ IDEA	Основне середовище розробки програмного забезпечення для серверної частини проекту. Воно забезпечує потужні інструменти для розробки на Java, включаючи підтримку Spring Boot та інтеграцію з системами керування версіями.
2	WebStorm	Основне середовище розробки для клієнтської частини проекту. Забезпечує зручні інструменти для розробки на JavaScript та Vue.js, включаючи підтримку різних плагінів та інструментів для відлагодження та тестування.
3	pgAdmin	Графічний інтерфейс для управління PostgreSQL базами даних. Використовується для адміністрування бази даних, виконання SQL-запитів та перегляду даних.
4	Redis Desktop Manager	Інструмент для управління Redis базами даних. Використовується для перегляду та адміністрування даних у Redis, що забезпечує кешування у проекті.
5	Git	Система керування версіями яка використовується для відстеження змін у коді, спільної роботи над проектом та керування версіями програмного забезпечення.
6	Docker	Платформа для автоматизації розгортання додатків у контейнерах. Використовується для створення, запуску та керування контейнерами, що забезпечує ізольоване середовище для запуску компонентів проекту [10].
7	Docker Compose	Інструмент для визначення та запуску багатоконтейнерних Docker додатків. Використовується для спрощення конфігурації та запуску всіх необхідних сервісів у проекті за допомогою єдиного YAML-файлу[11].

Таблиця 3.11 – Опис бібліотек

№ п/п	Назва утиліти	Опис застосування
1	Spring Boot	Використовується для створення серверної частини додатку, забезпечуючи просте налаштування та високу продуктивність. Інтегрується з іншими компонентами та бібліотеками для створення потужних REST API.
2	ROME	Використовується для роботи з RSS та Atom стрічками, забезпечуючи парсинг та генерацію стрічок. Дозволяє легко інтегрувати різні формати стрічок у проект.
3	Lunr.js	Використовується для реалізації повнотекстового пошуку на клієнтській стороні. Забезпечує швидкий і ефективний пошук без навантаження на сервер.
4	Vue3	Використовується для створення клієнтської частини додатку. Забезпечує високу продуктивність та зручність у розробці інтерфейсів користувача.
5	Vuetify	Набір компонентів для Vue.js, який дозволяє швидко створювати елегантні інтерфейси з готовими UI-елементами.
6	PostgreSQL	Використовується як основна база даних. Забезпечує надійне та масштабоване зберігання даних, підтримку складних SQL-запитів та ACID транзакцій.
7	Redis	Використовується для кешування даних, що дозволяє зменшити навантаження на базу даних та підвищити продуктивність додатку.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Однією з головних вразливостей є ненадійна аутентифікація, яка може дозволити зловмисникам отримати несанкціонований доступ до системи. Для захисту використовується JWT (JSON Web Tokens), що забезпечує безпечну аутентифікацію та авторизацію. Токени зберігаються у HTTP-only cookies для захисту від XSS атак.

Ще однією важливою вразливістю є SQL ін'єкції, які дозволяють зловмисникам використовувати вразливості у SQL запитах для виконання шкідливих команд. Захист від цього забезпечується використанням параметризованих запитів та ORM (Object-Relational Mapping). Spring Data JPA забезпечує безпечне управління запитами до бази даних, що мінімізує ризик SQL ін'єкцій.

Ненадійне зберігання паролів також є серйозною вразливістю, яка може призвести до компрометації облікових записів користувачів. Для захисту паролі зберігаються у хешованому вигляді за допомогою алгоритмів, таких як bcrypt. Хешування паролів здійснюється за допомогою Spring Security, що забезпечує їх надійне зберігання.

Таким чином, забезпечення безпеки даних у додатку включає комплекс заходів, спрямованих на захист від потенційних вразливостей та забезпечення цілісності, конфіденційності та доступності даних. Використання сучасних методів захисту та регулярне оновлення програмного забезпечення дозволяє мінімізувати ризики та забезпечити надійну роботу системи.

Висновки до розділу

У даному розділі було розглянуто архітектуру програмного забезпечення новинного агрегатора, обґрунтовано вибір засобів розробки, описано процес конструювання програмного забезпечення та проведено аналіз безпеки даних.

Архітектура програмного забезпечення додатку побудована на багатошаровій архітектурі, що включає клієнтську частину, серверну частину

(API), та базу даних. Клієнтська частина реалізована за допомогою Vue3 та Vuetify, що забезпечує зручний та сучасний інтерфейс користувача.

Вибір засобів розробки був обґрунтований їх ефективністю та продуктивністю. Java та JavaScript були обрані як основні мови програмування, оскільки вони забезпечують високу надійність та підтримку широкого спектра інструментів. Для середовища розробки було використано IntelliJ IDEA та WebStorm, які забезпечують потужні можливості для роботи з відповідними мовами програмування. Використання Spring Boot, ROME, Lunr.js, Vue3, Vuetify, PostgreSQL та Redis дозволило створити ефективну та легко масштабовану систему.

Було описано структури даних, що використовуються у проекті, зокрема таблиці users, queue_definitions, api_keys, subscription_definitions, staging_posts, thumbnails та feed_discovery_info. Кожна з цих таблиць відіграє важливу роль у зберіганні та управлінні даними, необхідними для функціонування системи.

Аналіз безпеки даних включав розгляд можливих вразливостей програмного забезпечення та заходів для їх усунення.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox, …);

– перевірка сумісності застосунку з різними операційними системами (Windows, Linux, …);

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

Метриками для оцінки якості ПЗ обрано наступні:

– швидкість виконання операцій – вимірюється час, необхідний для виконання основних операцій системи, таких як завантаження новин, фільтрація та пошук;

– надійність – оцінюється стабільність роботи системи та здатність працювати без збоїв протягом тривалого часу;

– юзабіліті – оцінюється зручність використання інтерфейсу, його інтуїтивність та доступність для користувачів;

– сумісність – оцінюється здатність системи працювати на різних платформах та в різних браузерах.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було проведено мануальне тестування програмного забезпечення, опис відповідних тестів наведено в таблицях 4.1 – 4.13.

Таблиця 4.1 – Тест 1.1 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Назва акаунту, пароль
Опис проведення тесту	У відповідні поля вводяться коректна назва акаунту, який був зареєстрований у системі, та правильний пароль. Після цього натискається кнопка входу.
Очікуваний результат	Авторизація проходить успішно, користувач перенаправляється на головну сторінку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Додавання субкатегорії

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці новин.
Вхідні дані	Назва субкатегорії
Опис проведення тесту	Користувач натискає символ на панелі управління і відкривається вспливаюче вікно. Користувач вводить дані для субкатегорії: назву і опис. Після цього користувач натискає кнопку додавання і вікно закривається автоматично
Очікуваний результат	Субкатегорія успішно додається до списку існуючих субкатегорій користувача.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Видалення субкатегорії

Початковий стан системи	Користувач авторизований і знаходиться на сторінці управління субкатегоріями.
Вхідні дані	Обрана субкатегорія
Опис проведення тесту	Користувач обирає субкатегорію для видалення і натискає кнопку видалення.

Продовження таблиці 4.3

Очікуваний результат	Субкатегорія успішно видаляється зі списку субкатегорій користувача.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Додавання підписки

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці
Вхідні дані	URL новинного ресурсу, обрана субкатегорія
Опис проведення тесту	Користувач переходить до субкатегорії, переходить до розділу додавання нових підписок. Користувач вводить URL новинного ресурсу та натискає кнопку перевірки. Сервер виконує перевірку URL та робить запит до ресурсу щоби перевірити чи підтримує ресурс RSS передачу даних. Виводиться інформація про ресурс і при позитивній відповіді користувач може натиснути кнопку додати.
Очікуваний результат	Підписка успішно додається до обраної субкатегорії.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Видалення підписки

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці
Вхідні дані	URL новинного ресурсу, обрана субкатегорія
Опис проведення тесту	Користувач переходить до субкатегорії, переходить до розділу управління підписками підписками. Користувач обирає підписку і натискає кнопку видалення.
Очікуваний результат	Підписка успішно видаляється зі списку підписок користувача.

Продовження таблиці 4.5

Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Оновлення списку новин

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці з новинами
Вхідні дані	Кнопка оновлення новин
Опис проведення тесту	Користувач натискає кнопку оновлення новин.
Очікуваний результат	Список новин оновлюється і відображає нові повідомлення.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Пошук новин

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці з новинами
Вхідні дані	Пошуковий запит
Опис проведення тесту	У пошукове поле вводиться запит, після чого список новин оновлюється включаючи новини які відповідають запиту.
Очікуваний результат	Відображаються результати пошуку, що відповідають запиту.

Таблиця 4.8 – Тест 1.8 Відмітка новин як прочитаних

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці з новинами
Вхідні дані	Обрані новини
Опис проведення тесту	Користувач обирає новини і натискає кнопку відмітки як прочитаних.

Продовження таблиці 4.8

Очікуваний результат	Відмічені новини змінюють свій статус на "прочитані" і в залежності від заданої фільтрації, або залишаються видимі або не відображаються допоки користувач не змінить фільтри.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Відмітка новин як прочитати потім

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці з новинами
Вхідні дані	Обрані новини
Опис проведення тесту	Користувач обирає новини і натискає кнопку відмітки як прочитати потім.
Очікуваний результат	Відмічені новини змінюють свій статус на "прочитати потім" і в залежності від заданої фільтрації, або залишаються видимі або не відображаються допоки користувач не змінить фільтри..
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Сортування прочитаних новин

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці з новинами
Вхідні дані	Обрані новини
Опис проведення тесту	Користувач вмикає або вимикає сортування прочитаних новин.
Очікуваний результат	Відображаються або ховаються новини, що відповідають заданим параметрам.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Сортування новин прочитати потім

Початковий стан системи	Користувач авторизований і знаходиться на головній сторінці з новинами
Вхідні дані	Обрані новини
Опис проведення тесту	Користувач вмикає або вимикає сортування новин на прочитати потім.
Очікуваний результат	Відображаються новини, що відповідають заданим параметрам сортування.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Відкриття оригінального посту

Початковий стан системи	Користувач авторизований і знаходиться на сторінці новин.
Вхідні дані	Обраний пост
Опис проведення тесту	Користувач обирає пост і натискає кнопку щоб перейти до першоджерела
Очікуваний результат	Система перенаправляє користувача до оригінального посту у новій вкладці.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Поділитися посиланням

Початковий стан системи	Користувач авторизований і знаходиться на сторінці новин.
Вхідні дані	Обраний пост, обрана соціальна мережа
Опис проведення тесту	Користувач обирає пост і натискає кнопку щоби поділитися посиланням, обирає соціальну мережу.

Продовження таблиці 4.13

Очікуваний результат	Система генерує посилання та перенаправляє у обрану соціальну мережу.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Користувач має змогу за допомогою навігаційної панелі нагорі сторінки авторизуватись у додаток. Якщо користувач ще не має акаунту то він може створити новий акаунт натиснувши на кнопку “Create an account”. При натисканні на кнопку користувач перейде на сторінку реєстрації де потрібно ввести дані для нового акаунту(рисунок 4.1)

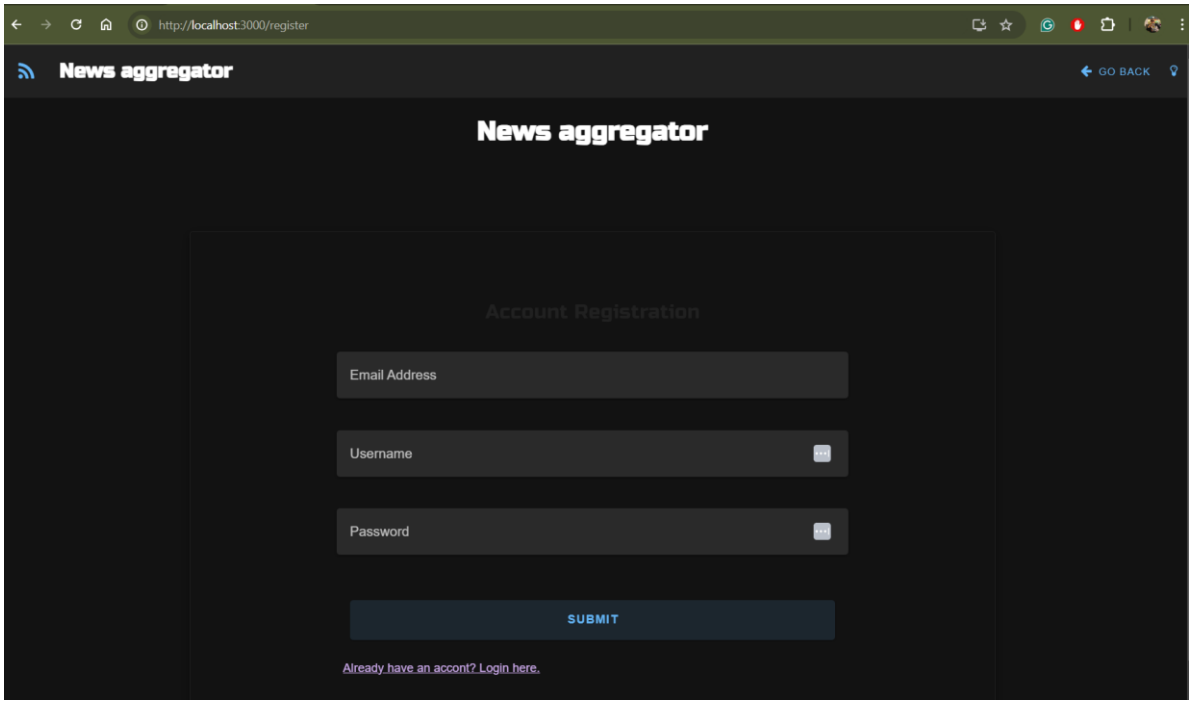


Рисунок 4.1 – Сторінка реєстрації

Після успішної реєстрації користувач перенаправляється на сторінку авторизації (рисунок 4.2).

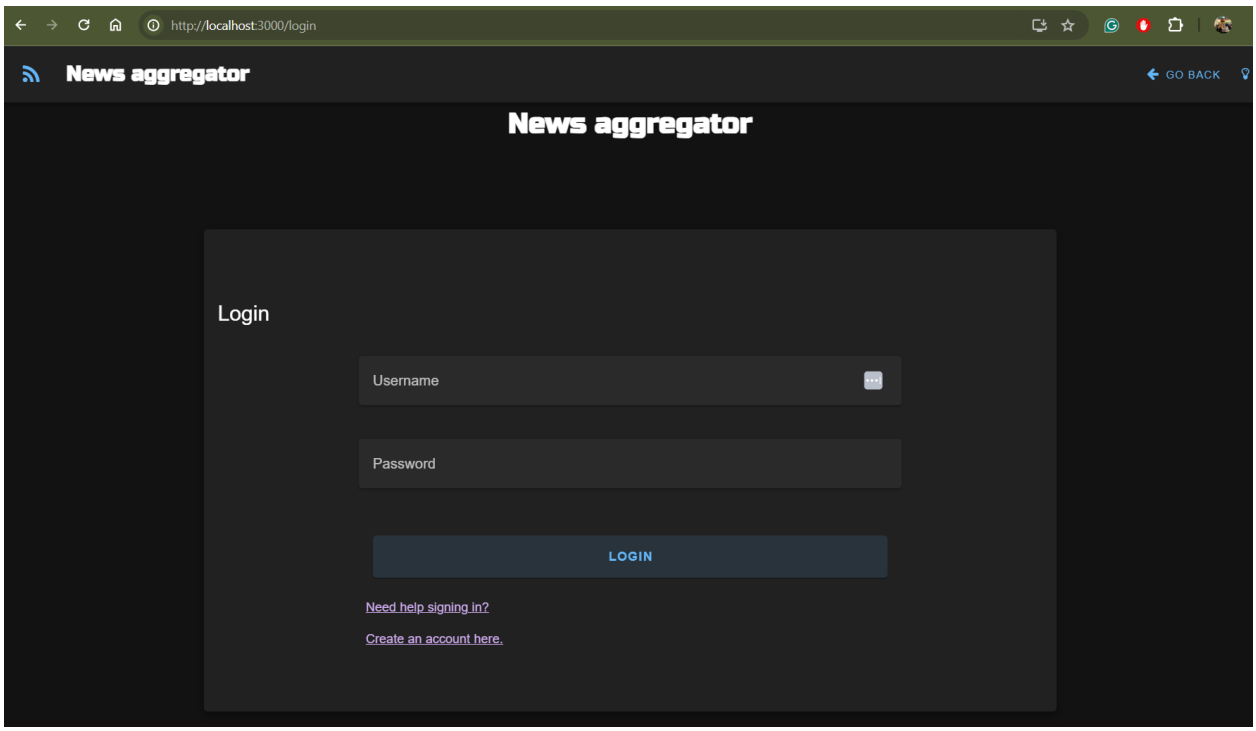


Рисунок 4.2 – Сторінка авторизації

Після успішної авторизації користувач переходить на головну сторінку додатку (рисунок 4.3).

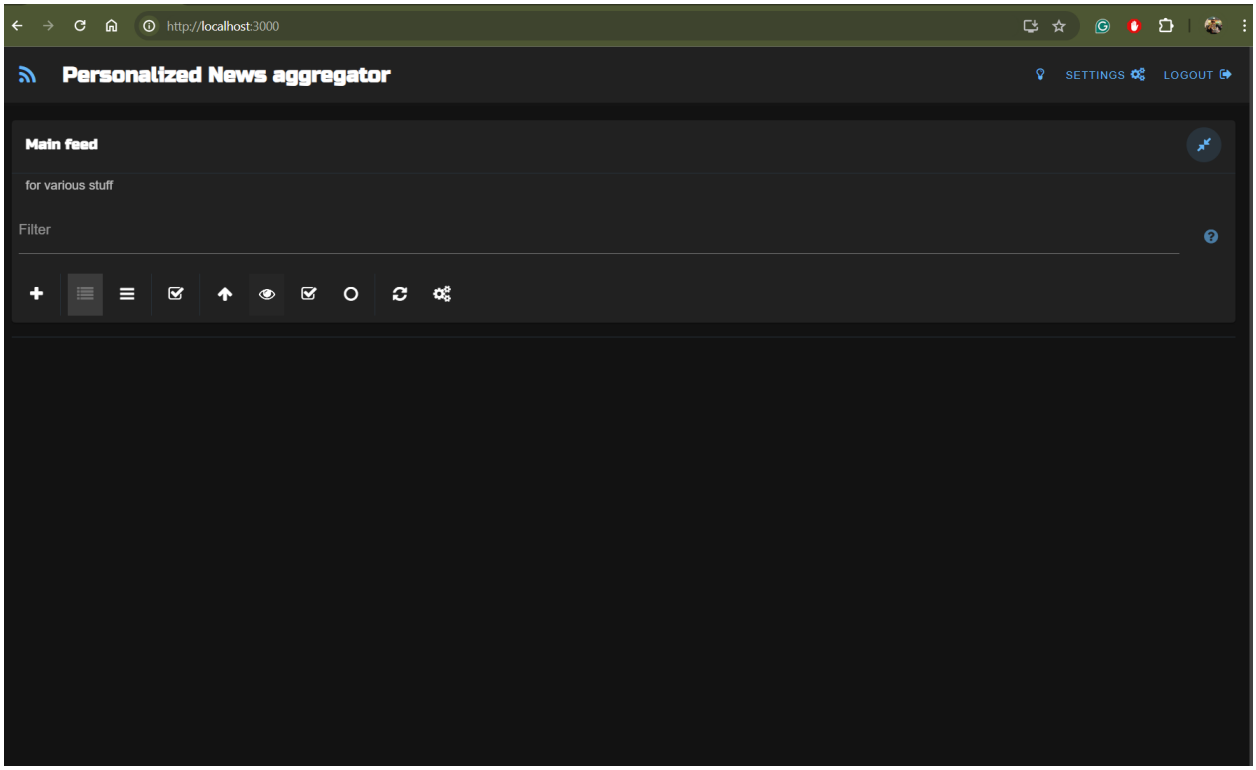


Рисунок 4.3 – Головна сторінка новин

Перед тим як додати нові підписки на новини нам треба додати нові категорії новин, для цього в інтерфейсі треба натиснути на кнопку плюс, після цього відкривається діалогове вікно для створення нової категорії (рисунок 4.4).

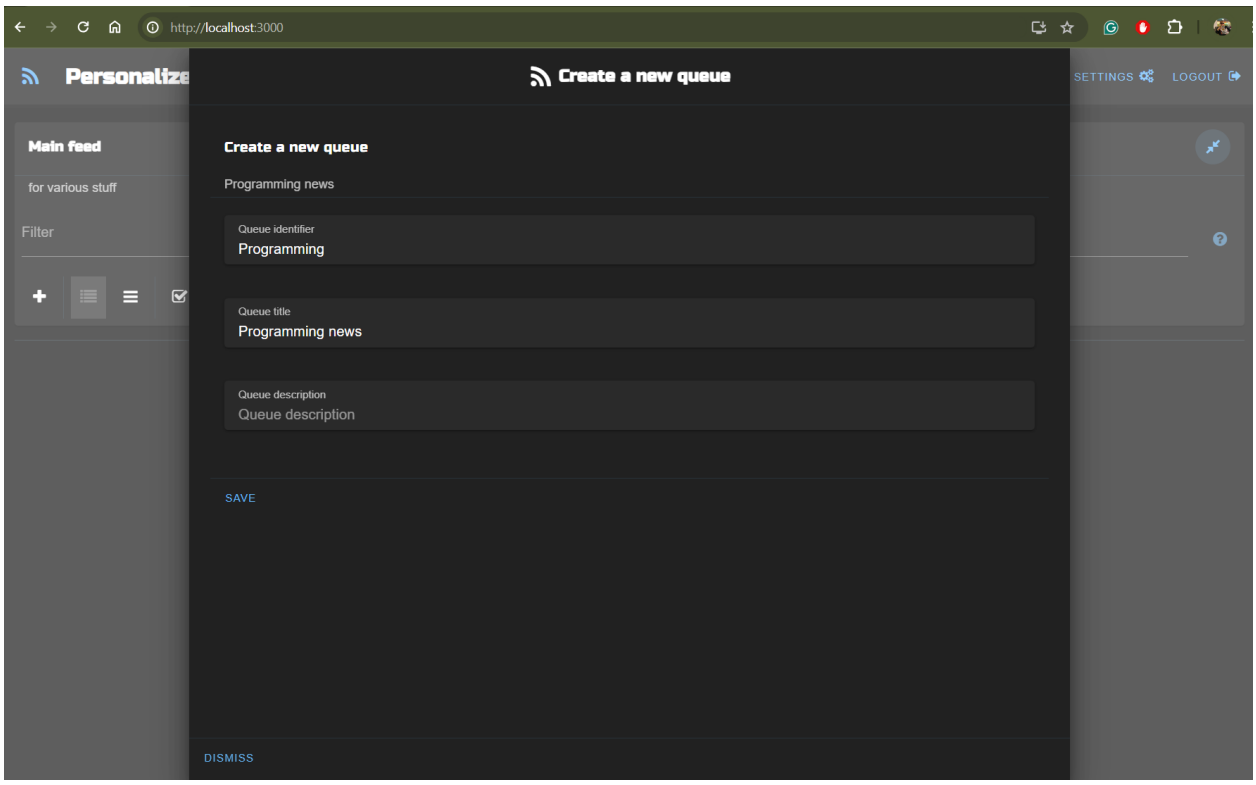


Рисунок 4.4 – Діалогове вікно створення категорії

Після створення нової категорії її можна побачити у списку всіх категорій натиснувши на кнопку згори зліва в інтерфейсі (рисунок 4.5).

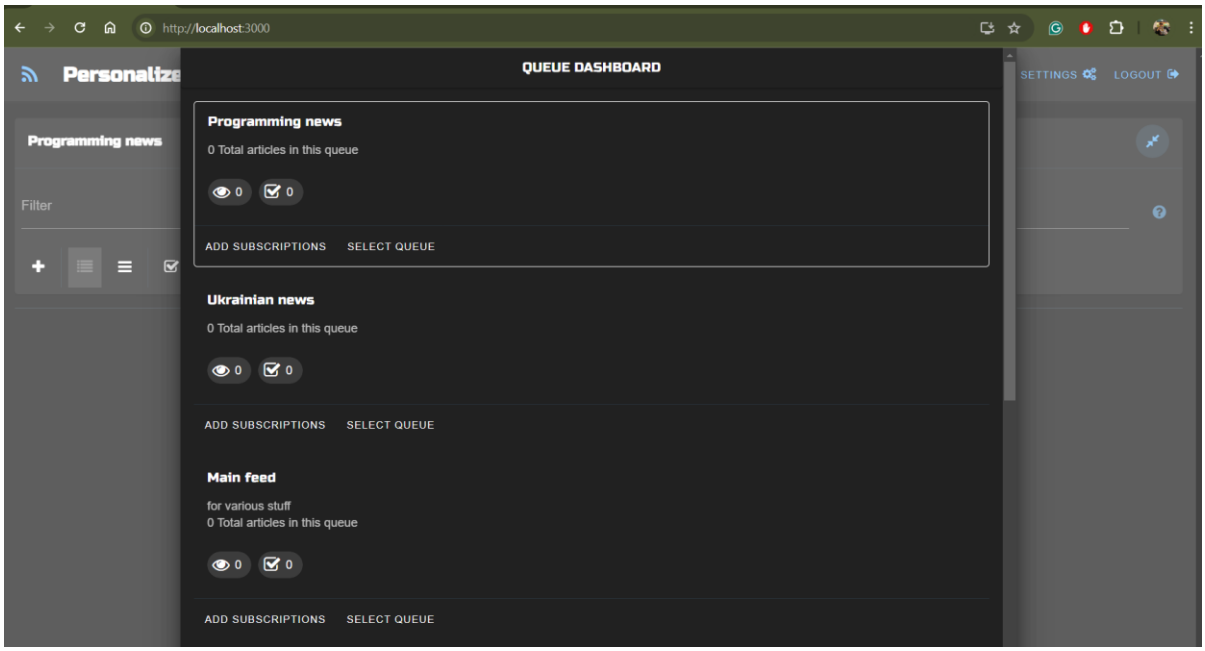


Рисунок 4.5 – Список існуючих категорій

Після цього можна натиснути натиснути на кнопку “Add subscriptions” щоби перейти до створення нової підписки на новину у цій категорії (рисунок 4.6).

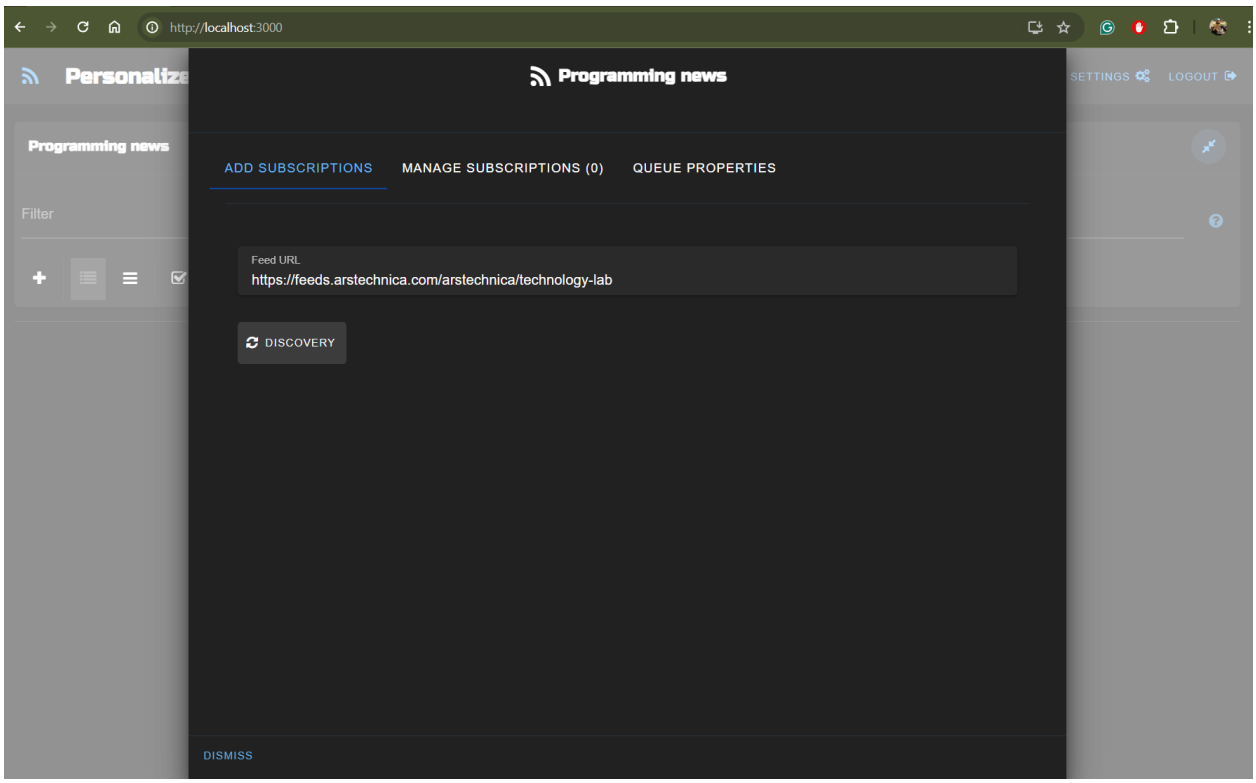


Рисунок 4.6 – Діалогове вікно створення нової підписки у категорії

Перед тим як додати підписку до категорії користувач натискає на кнопку “Discovery” після чого сервер робить запит до першоджерела щоби перевірити валідність посилання та перевірити чи є підтримка передачі даних по RSS, і тільки після цього користувач може зробити нову підписку (рисунок 4.7).

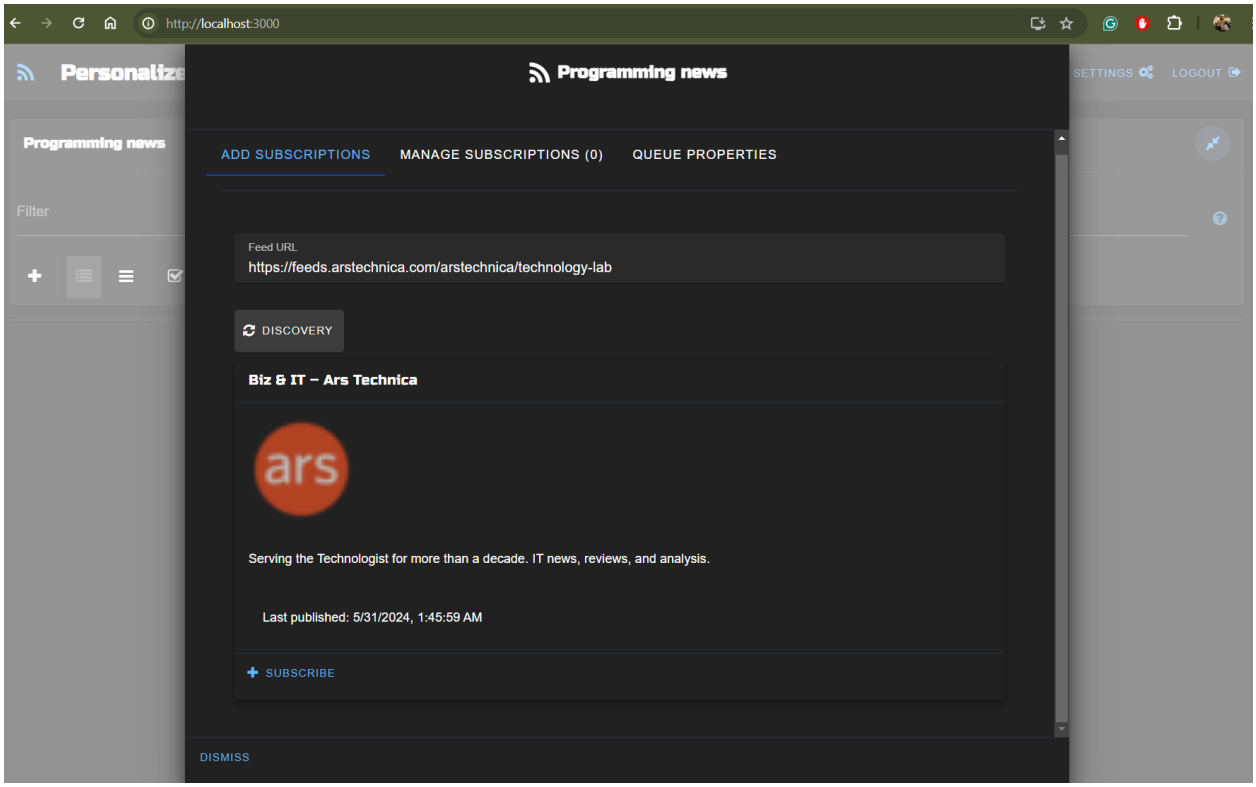


Рисунок 4.7 – Діалогове вікно створення нової підписки у категорії

Після цього відбувається оновлення головної сторінки новин і тепер користувач бачить нові новини з першоджерела (рисунок 4.8)

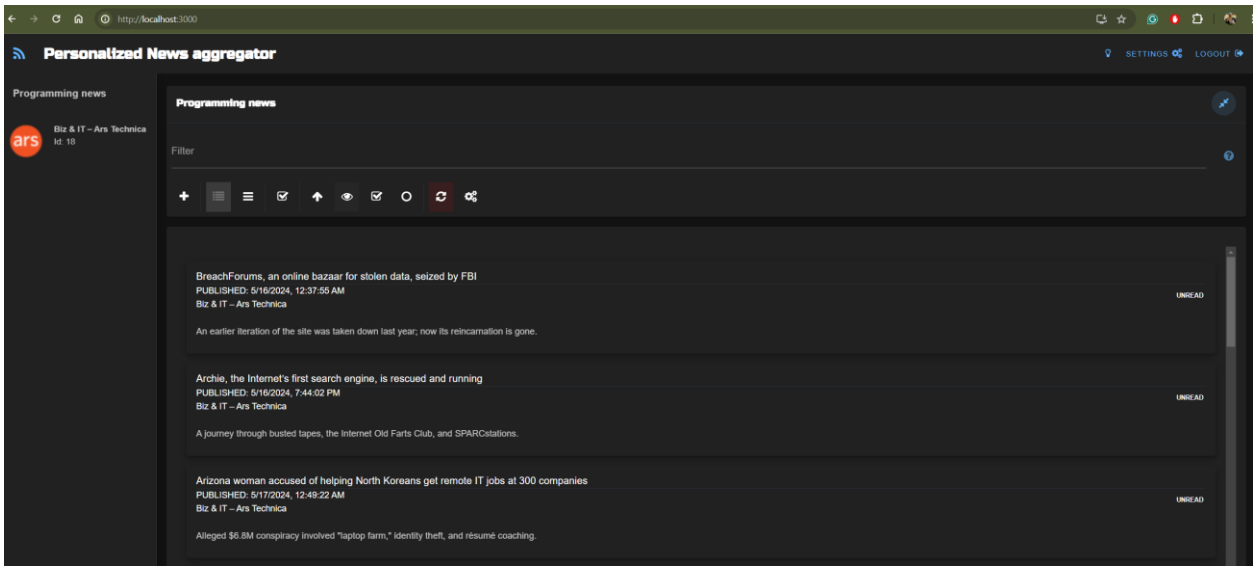


Рисунок 4.8 – Оновлена головна сторінка новин

При натисканні на новини відкривається діалогове вікно для детального читання новини включаючи зображення, текст, посилання в тексті, посилання на коментарі до новини у першоджерелі, та інші метадані. Можливо переходити від читання одної новини до іншої за допомогою стрілочок, можна відмічати новини як прочитані або прочитати потім, можна перейти до першоджерела або отримати поділитися посиланням у соцмережах (рисунок 4.9).

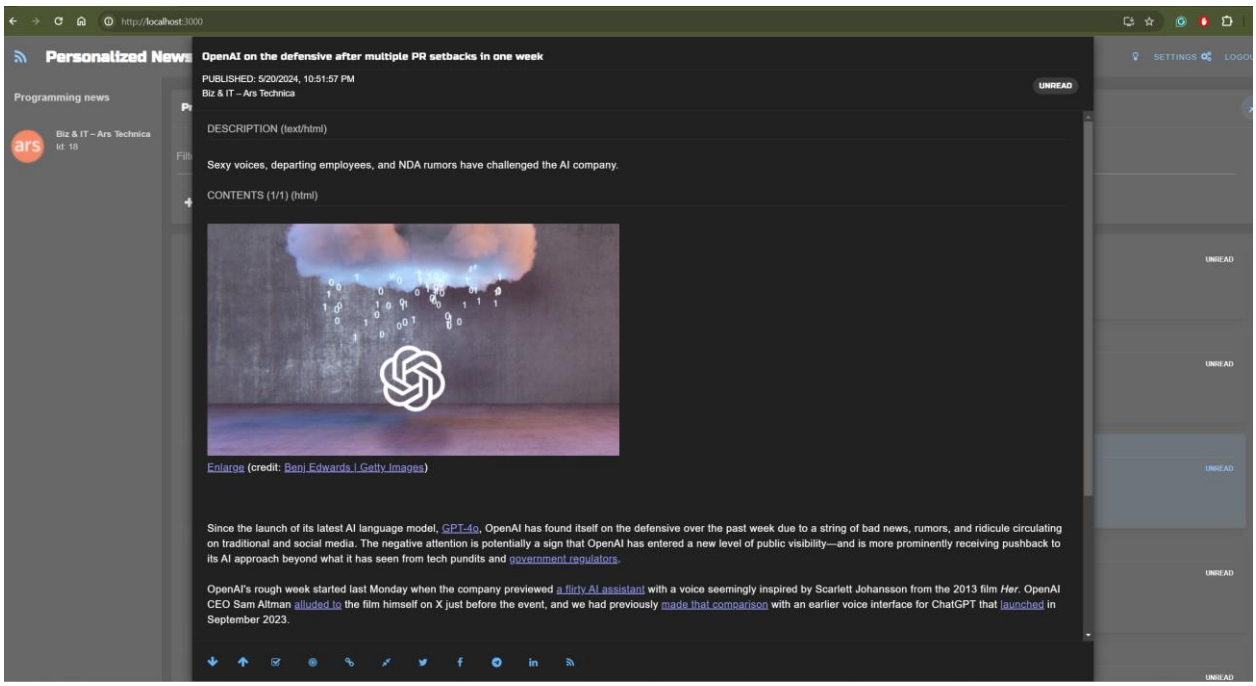


Рисунок 4.9 – Діалогове вікно читання новин

При відмічанні новин як прочитані або прочитати потім вони зникають з списку новин допоки користувач не буде фільтрувати прочитані новини (рисунок 4.10) або прочитати потім (рисунок 4.11).

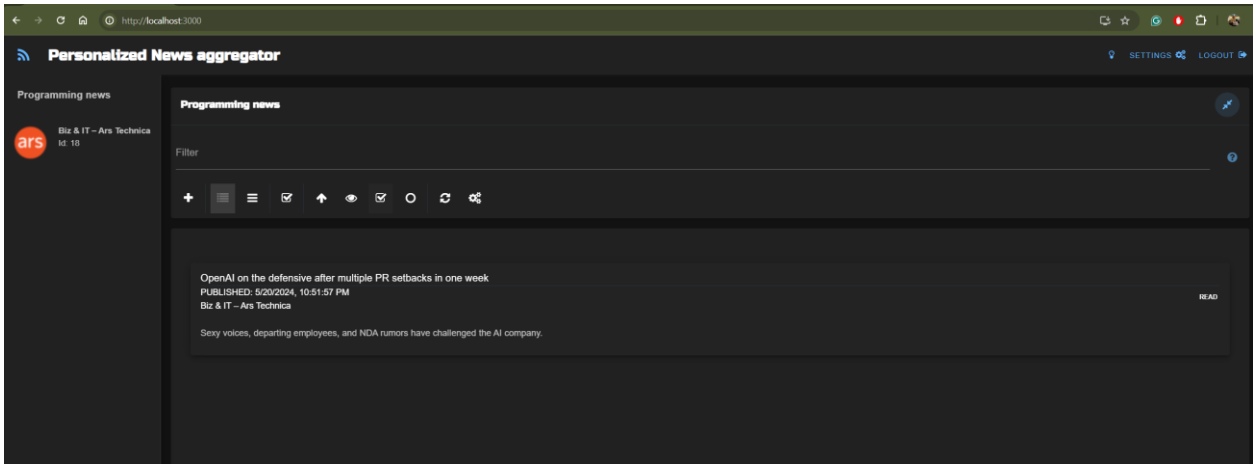


Рисунок 4.10 – Фільтрування прочитаних новин

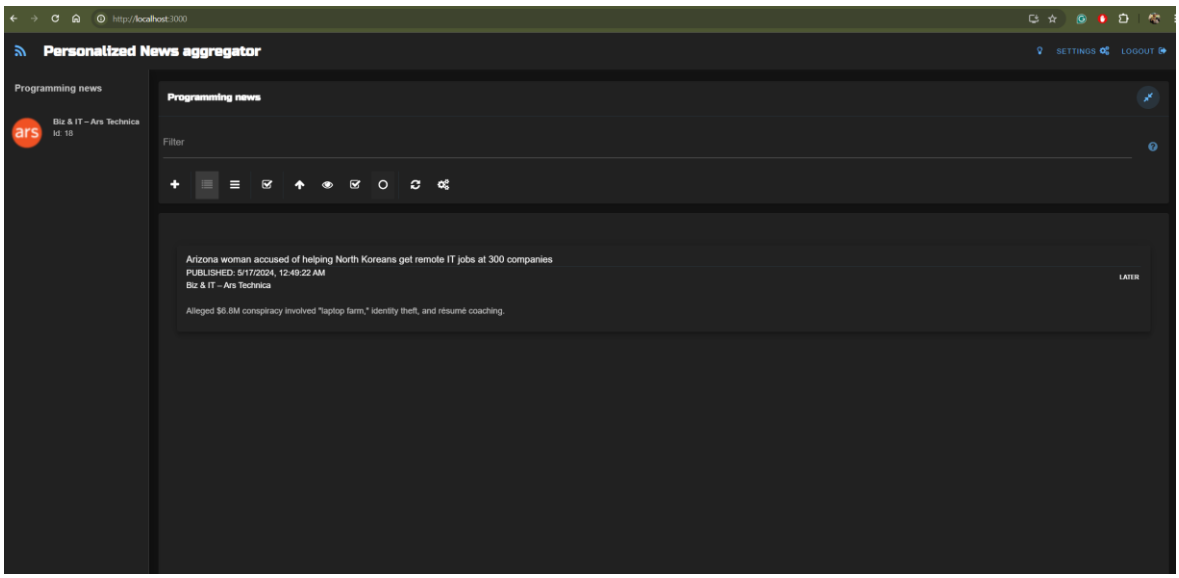


Рисунок 4.11 – Фільтрування новин на прочитати потім

Для пошуку новин серед декількох підписок у категорії можна використовувати поле фільтрації для пошуку по заданому фільтру (рисунок 4.12).

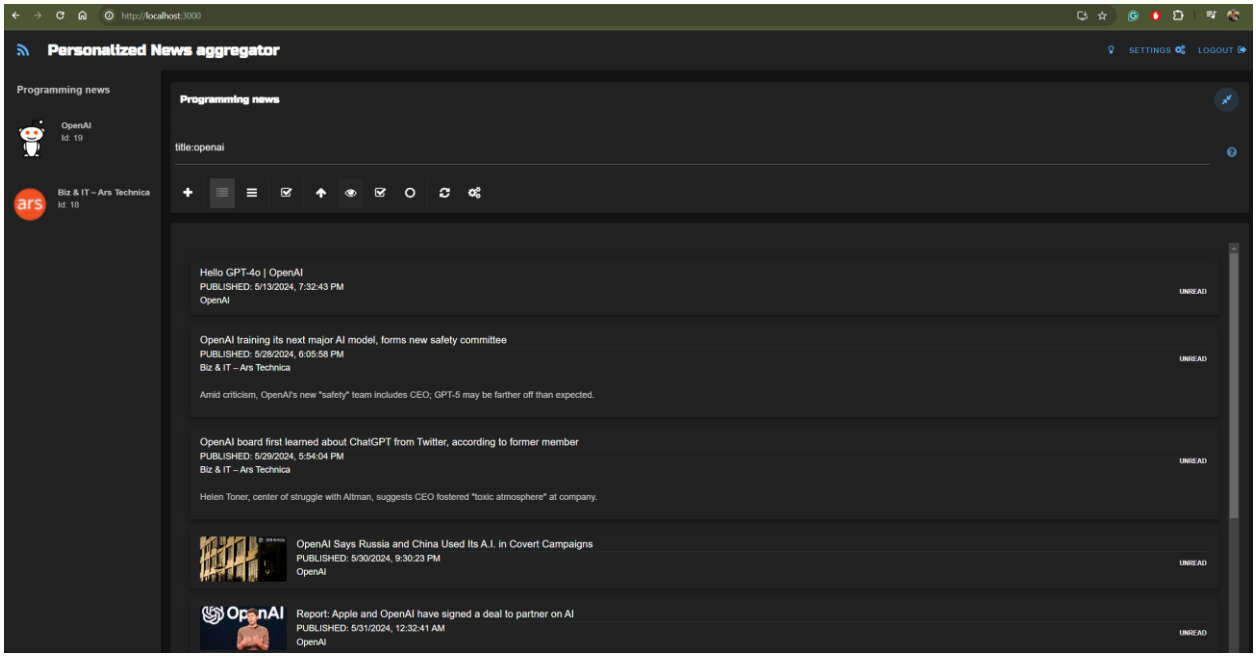


Рисунок 4.12 – Пошук новин за заданим запитом

Висновки до розділу

У даному розділі було проведено аналіз якості та тестування веб-застосунку. Було визначено мету тестування та описані метрики якості, такі як швидкість виконання операцій, надійність, зручність використання (юзабіліті) та сумісність.

Результати тестування підтвердили коректність роботи основних функцій застосунку, його стабільність та сумісність з різними веб-браузерами та операційними системами.

Були описані процеси тестування, які включали реєстрацію та авторизацію користувачів, додавання та видалення підписок, пошук новин, фільтрацію та сортування новин, а також взаємодію з новинними постами. Проведені тести підтвердили відповідність програмного забезпечення заданим вимогам.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Процес розгортання програмного забезпечення персоналізованого новинного агрегатора включає кілька основних етапів, які забезпечують належну конфігурацію системи та її готовність до використання. Основними компонентами системи є база даних PostgreSQL, кеш Redis та основний застосунок, що працює на базі Docker-контейнерів.

Після встановлення Docker та Docker Compose, ми використовуємо Docker Compose файл, який вже містить конфігурацію всіх необхідних сервісів. У нашому випадку це база даних PostgreSQL, кеш Redis, основний застосунок та допоміжні сервіси. У конфігураційному файлі зазначені образи контейнерів, порти, що використовуються, змінні середовища та залежності між сервісами.

Розпочинаємо з налаштування бази даних PostgreSQL. У Docker Compose файлі визначається сервіс для бази даних, який використовує офіційний образ PostgreSQL. Налаштовуються змінні середовища для встановлення користувача та пароля бази даних, а також визначаються порти, що будуть використані для з'єднання з базою даних. Після налаштування сервісу PostgreSQL переходимо до конфігурації кеша Redis. Для цього використовується офіційний образ Redis, який дозволяє швидко налаштувати кешування даних. Вказуються порти для підключення до Redis та задаються необхідні параметри, такі як пароль для доступу до кешу.

Наступним кроком є налаштування основного застосунку. У конфігураційному файлі Docker Compose визначається сервіс для основного застосунку, який базується на образі завантаженому до Docker Hub. Встановлюються змінні середовища для налаштування з'єднання з базою даних PostgreSQL та кешем Redis, а також задаються порти, що будуть використані для взаємодії з користувачами. Крім того, налаштовуються

додаткові параметри, такі як URL-адреси для аутентифікації та поштових серверів.

Особливу увагу приділяємо налаштуванню залежностей між сервісами. Важливо, щоб кожен сервіс запускався у правильному порядку та міг взаємодіяти з іншими компонентами системи. Для цього у Docker Compose файлі задаються залежності між сервісами за допомогою параметра "depends_on". Наприклад, основний застосунок залежить від бази даних та кешу, тому вони мають бути запущені раніше.

Після завершення налаштування всіх сервісів, здійснюється запуск контейнерів. Для цього використовується команда "docker-compose up", яка завантажує необхідні образи, створює контейнери та запускає їх. У процесі запуску Docker Compose виконує налаштування мережі між контейнерами та забезпечує їхню взаємодію відповідно до заданих конфігурацій.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Коли розробник завершує роботу над певним компонентом та вносить зміни до репозиторію, ці зміни автоматично тригерять запуск CI/CD пайплайну у GitHub Actions. Цей пайплайн забезпечує автоматизоване тестування та збірку програмного забезпечення.

Застосунок компілюється та пакується у Docker-образи, які потім завантажуються у Docker Hub або інше сховище контейнерів. Це забезпечує централізоване зберігання образів та їх доступність для розгортання на будь-якому сервері або у хмарному середовищі.

Для користувачів нової версії застосунку забезпечується простий процес оновлення. Вони можуть отримати нову версію програмного забезпечення, завантаживши нові образи контейнерів та розгорнувши їх за допомогою Docker Compose. Користувачам не потрібно вручну налаштовувати кожен компонент, оскільки всі необхідні конфігурації вже вказані у Docker Compose файлі.

Висновки до розділу

У цьому розділі було детально розглянуто процес розгортання та супроводу програмного забезпечення персоналізованого новинного агрегатора. Спочатку описано основні етапи розгортання системи, включаючи налаштування бази даних PostgreSQL, кешу Redis та основного застосунку за допомогою Docker та Docker Compose. Було наголошено на важливості правильного налаштування залежностей між сервісами та забезпеченні їх взаємодії через конфігураційний файл Docker Compose.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано, розроблено та протестовано веб-застосунок для персоналізованої агрегації новин. Метою даного проекту було створення зручного, ефективного та безпечного засобу для отримання релевантного контенту з різних джерел.

У першому розділі визначено мету розробки проекту та окреслено основні цілі та завдання. Було підкреслено необхідність створення інтуїтивно зрозумілого інтерфейсу користувача, а також важливість забезпечення надійності та масштабованості системи.

У другому розділі детально проаналізовано функціональні та нефункціональні вимоги до програмного забезпечення. Описано основні функціональні можливості веб-застосунку, такі як управління підписками, організація контенту, пошук та фільтрація новин.

У третьому розділі описано архітектуру програмного забезпечення, обґрунтовано вибір засобів розробки та детально розглянуто процес конструювання.

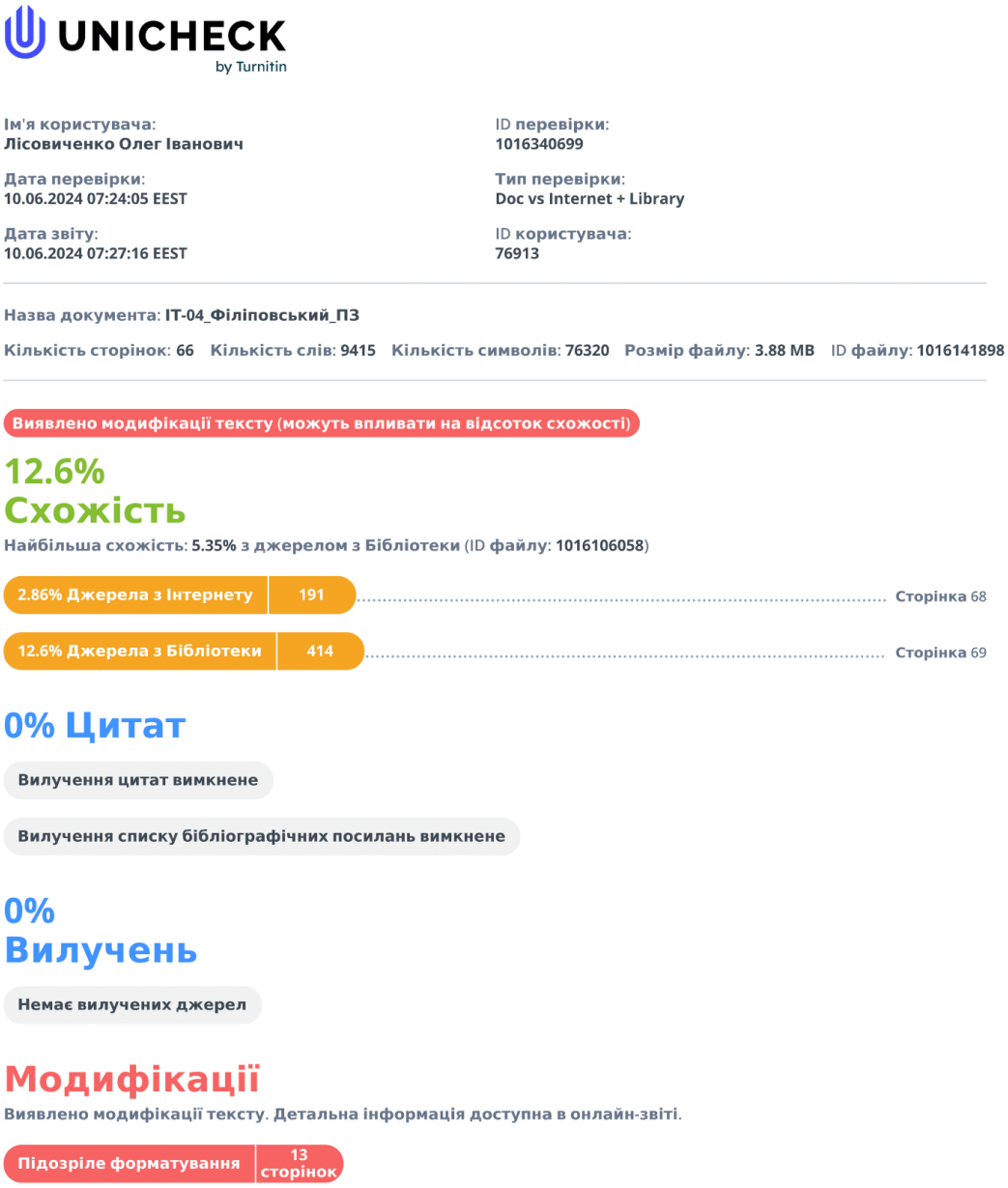
Четвертий розділ присвячено аналізу якості програмного забезпечення та результатам тестування. Було визначено мету тестування та описано метрики якості, такі як швидкість виконання операцій, надійність, зручність використання та сумісність. Проведене мануальне тестування підтвердило коректність роботи основних функцій, стабільність та сумісність застосунку з різними веб-браузерами та операційними системами.

У п’ятому розділі детально розглянуто процес розгортання та супроводу програмного забезпечення. Описано кроки розгортання, налаштування середовища та забезпечення належної роботи всіх компонентів системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Технологія RSS [Електронний ресурс] // Apix-Drive. – Режим доступу до ресурсу: https://apix-drive.com/ua/blog/useful/shcho-take-rss.
2) Feedly News Reader: take control of your newsfeed [Електронний ресурс] // Feedly. – Режим доступу до ресурсу: https://feedly.com/news-reader.
3) Inoreader – Build your own newsfeed [Електронний ресурс] // Inoreader. – Режим доступу до ресурсу: https://www.inoreader.com/uk.
4) GitHub - samuelclay/newsblur [Електронний ресурс] // Newsblur. – Режим доступу до ресурсу: https://github.com/samuelclay/NewsBlur.
5) Redis Docs [Електронний ресурс] // Redis - The Real-time Data Platform. – Режим доступу до ресурсу: https://redis.io/docs/latest.
6) ROME tools documentation [Електронний ресурс] // Rome tools. – Режим доступу до ресурсу: https://rometools.github.io/rome/index.html.
7) LunrJS Docs: home [Електронний ресурс] // Lunr: A bit like Solr, but much smaller and not as bright. – Режим доступу до ресурсу: https://lunrjs.com/docs/index.html.
8) Getting started with IntelliJ IDEA [Електронний ресурс] // JetBrains. – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/getting-started.html.
9) Getting started with WebStorm [Електронний ресурс] // JetBrains. – Режим доступу до ресурсу: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html.
10) Docker Home [Електронний ресурс] // Docker Documentation. – Режим доступу до ресурсу: https://docs.docker.com/.
11) Docker Compose overview [Електронний ресурс] // Docker Documentation. – Режим доступу до ресурсу: https://docs.docker.com/compose/.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016340699

Дата перевірки:
10.06.2024 07:24:05 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
10.06.2024 07:27:16 EEST

ID користувача:
76913

Назва документа: ІТ-04_Філіповський_ПЗ

Кількість сторінок: 66 Кількість слів: 9415 Кількість символів: 76320 Розмір файлу: 3.88 MB ID файлу: 1016141898

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

12.6%
Схожість

Найбільша схожість: 5.35% з джерелом з Бібліотеки (ID файлу: 1016106058)

2.86% Джерела з Інтернету 191 ..... Сторінка 68

12.6% Джерела з Бібліотеки 414 ..... Сторінка 69

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 13 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Вебзастосунок "Персоналізований новинний агрегатор"

Текст програми

КПІ.ІТ-0421.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Даніїл ФІЛІПОВСЬКИЙ

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/onestep87/news-aggregator

Файл AuthenticationController.java
Реалізація функціональної вимоги логіну

```
package main_app;


import main_app.audit.AuthClaimException;
import main_app.audit.AuthProviderException;
import main_app.auth.AuthService;
import main_app.user.LocalUserService;
import main_app.model.request.LoginRequest;
import main_app.model.response.LoginResponse;
import main_app.model.TokenType;
import main_app.user.UserRoles;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.*;
import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;


import static org.apache.commons.lang3.StringUtils.EMPTY;
import static org.springframework.http.ResponseEntity.ok;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;

/**
 * This controller handles authentication functionality (i.e., login,
logout).
 *
 * user registration, and email validation are handled elsewhere.
 *
 * The 'currentUser' call is used by the front-end to determine if the user
 * has a valid refresh token.
 *
 * The 'authenticate' call is used to setup a logged-in session.  This call
also adds a
 * refresh token cookie to the response, so that the user remains 'logged in'
as long as
 * the cookie persists.
 *
 * The 'deauthenticate' call is used to log out.  Calling this method
finalized the auth claim
 * on the user object, so that the further attempts to validate already-
generated tokens will fail.
 */
@Controller
@Validated
class AuthenticationController {

    @Autowired
    AuthService authService;

    @Autowired
    AuthenticationManager authenticationManager;
```

```
    @Autowired
    LocalUserService localUserService;
    //
    // auth check
    //
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @RequestMapping(value = "/currentuser", method = GET)
    public ResponseEntity<LoginResponse> getCurrentUser(Authentication
authentication) throws AuthClaimException, DataAccessException {
        UserDetails userDetails = (UserDetails) authentication.getDetails();
        String username = userDetails.getUsername();
        LoginResponse authenticationResponse = buildAuthenticationResponse(
                authService.generateAuthToken(username).authToken,
                username,

userDetails.getAuthorities().contains(UserRoles.SUBSCRIBER_AUTHORITY)
        );

        return ok(authenticationResponse);
    }
    //
    // login (open access)
    //
    @RequestMapping(value = "/authenticate", method = POST)
    @Transactional
    public ResponseEntity<LoginResponse> createAuthenticationToken(@Valid
@RequestBody LoginRequest loginRequest, HttpServletResponse response)
            throws BadCredentialsException, UsernameNotFoundException,
AuthProviderException, AuthClaimException, DataAccessException {
        // extract username
        String username = loginRequest.getUsername();
        // validate the auth provider
        authService.requireAuthProvider(username, LOCAL);
        // add refresh token cookie to response
        String authClaim = authService.requireAuthClaim(username);
        // authenticate user
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(username,
loginRequest.getPassword()));
        // generate the auth token/build auth response
        authService.addTokenCookieToResponse(TokenType.APP_AUTH_REFRESH,
username, authClaim, response);
        LoginResponse authenticationResponse = buildAuthenticationResponse(
                authService.generateAuthToken(username).authToken,
                username,

localUserService.loadUserByUsername(username).getAuthorities().contains(UserR
oles.SUBSCRIBER_AUTHORITY)
        );

        log.info("Login succeeded for username={}", username);

        return ok(authenticationResponse);
    }

    private LoginResponse buildAuthenticationResponse(String authToken,
String username, boolean hasSubscription) {
        LoginResponse loginResponse;
        loginResponse = LoginResponse.from(authToken, username,
hasSubscription);

        return loginResponse;
    }
    //
```

```
    // logout (open access)
    //
    @RequestMapping(value = "/deauthenticate", method = GET)
    @Transactional
    public ResponseEntity<String> deauthenticate(Authentication
authentication) throws DataAccessException, DataUpdateException {
        if (authentication != null) {
            UserDetails userDetails = (UserDetails)
authentication.getDetails();
            String username = userDetails.getUsername();
            // finalize the auth claim
            authService.finalizeAuthClaim(username);
            log.info("Finalized auth claim for username={}", username);
        }

        return ok(EMPTY);
    }
}
```

Файл LoginView.vue

Реалізація функціональної вимоги логіну

```
 <template>
  <v-app>
    <!-- pre-auth app bar -->
    <v-app-bar app location="top" :scroll-behavior="'elevate'">
      <template #title>
        <span class="newsaggregator-rss"> News Aggregator </span>
      </template>
      <template #prepend>
        <v-app-bar-nav-icon icon="fa-rss" :aria-
label="$t('newsaggregatorRssLogo')" />
      </template>
      <v-toolbar-items>
        <GoBack />
        <DisplayModeButton />
      </v-toolbar-items>
    </v-app-bar>

    <!-- pre-auth main -->
    <v-main>
      <BannerPanel :show-auth="false" />

      <AuthPanel
        v-show="!isAuthenticated"
        :server-message="roAuthServerMessage"
        :is-loading="roLoginIsLoading"
        @login="login"
      />
    </v-main>
  </v-app>
</template>

<script>
import { inject } from 'vue';
import GoBack from '@/components/generic/GoBack.vue';
import DisplayModeButton from '@/components/generic/DisplayModeButton.vue';
import BannerPanel from '@/components/banner-panel/BannerPanel.vue';
import AuthPanel from '@/components/auth-panel/AuthPanel.vue';
import { useAuth } from '@/composable/useAuth.js';

export default {
  name: 'LoginView',
  components: {
    GoBack,
```

```
    DisplayModeButton,
    BannerPanel,
    AuthPanel,
  },
  setup() {
    const isAuthenticated = inject('isAuthenticated');
    const { roAuthServerMessage, roLoginIsLoading, login } = useAuth();

    return {
      isAuthenticated,
      roAuthServerMessage,
      roLoginIsLoading,
      login,
    };
  },
};
</script>

<style scoped>
.newsaggregator-rss {
  font-family: 'Russo One', system-ui, sans-serif;
  font-weight: bold;
  font-size: larger;
}
</style>
```

Файл RegistrationController.java

Реалізація функціональної вимоги реєстрації

```
package main_app;

import main_app.auth.AuthService;
import main_app.feed.QueueDefinitionService;
import main_app.model.AppToken;
import main_app.model.request.RegistrationRequest;
import main_app.token.TokenService;
import main_app.user.LocalUserService;
import org.apache.commons.lang3.time.StopWatch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import static org.springframework.http.ResponseEntity.ok;
import static org.springframework.web.bind.annotation.RequestMethod.*;

@Controller
@Validated
public class RegistrationController {


    @Autowired
    private LocalUserService userService;

    @Autowired
    private AuthService authService;

    @Autowired
    private TokenService tokenService;

    @Autowired
    private QueueDefinitionService queueDefinitionService;

    @RequestMapping(path = "/register", method = POST)
    @Transactional
    public ResponseEntity<RegistrationResponse> register(@Valid @RequestBody
```

```
RegistrationRequest registrationRequest) throws RegistrationException {
        String username = registrationRequest.getUsername();
        String email = registrationRequest.getEmail();
        String password = registrationRequest.getPassword();
        log.info("Registering user with username={}, email={}", username,
email);
        StopWatch timer = StopWatch.createStarted();
        userService.registerUser(username, email, password);
        authService.finalizeVerificationClaim(username);
        authService.finalizeAuthClaim(username);
        queueDefinitionService.createDefaultFeed(username);
        AppToken verificationToken =
authService.generateVerificationToken(username);
        timer.stop();
        appLogService.logUserRegistration(username, timer);
        return ok(new RegistrationResponse(username, password));
    }
}
```

Файл RegistrationRequestView.vue

Реалізація функціональної вимоги реєстрації користувача

```
 <template>
  <v-app>
    <v-app-bar app location="top" :scroll-behavior="'elevate'">
      <template #title>
        <span class="newsaggregator-rss"> News Aggregator </span>
      </template>
      <template #prepend>
        <v-app-bar-nav-icon icon="fa-rss" :aria-
label="$t('newsaggregatorRssLogo')" />
      </template>
      <v-toolbar-items>
        <GoBack />
        <DisplayModeButton />
      </v-toolbar-items>
    </v-app-bar>

    <v-main>
      <BannerPanel :show-auth="false" />
      <RegistrationRequestPanel
        :is-loading="registrationIsLoading"
        :server-message="serverMessage"
        @submit-registration="submitRegistration"
      />
      <FooterPanel app />
    </v-main>
  </v-app>
</template>

<script>
import { inject, ref } from 'vue';
import { useI18n } from 'vue-i18n';
import { useRouter } from 'vue-router';
import BannerPanel from '@/components/banner-panel/BannerPanel.vue';
import GoBack from '@/components/generic/GoBack.vue';
import DisplayModeButton from '@/components/generic/DisplayModeButton.vue';
import RegistrationRequestPanel from '@/components/registration-
panel/RegistrationRequestPanel.vue';
import FooterPanel from '@/components/footer-panel/FooterPanel.vue';

export default {
  name: 'RegistrationRequestView',
```

```
  components: {
    BannerPanel,
    GoBack,
    DisplayModeButton,
    RegistrationRequestPanel,
    FooterPanel,
  },
  setup() {
    const auth = inject('auth');
    const router = useRouter();
    const { t } = useI18n();

    const registrationIsLoading = ref(false);
    const serverMessage = ref(null);

    function submitRegistration(registrationRequest) {
      const { username, email, password, userType } = registrationRequest;
      serverMessage.value = null;
      if (!username || !email || !password) {
        serverMessage.value = t('registrationRequirements');
        return;
      }

      registrationIsLoading.value = true;
      auth
        .registerWithSupplied(username, email, password, userType)
        .then((response) => {
          auth
            .loginWithSupplied(response.username, response.password, false)
            .then(() => {
              router.push('/');
            })
            .catch((error) => {
              serverMessage.value = error;
            })
            .finally(() => {
              registrationIsLoading.value = false;
            });
        })
        .catch((error) => {
          serverMessage.value = error;
        })
        .finally(() => {
          registrationIsLoading.value = false;
        });
    }

    return {
      auth,
      router,
      t,
      registrationIsLoading,
      serverMessage,
      submitRegistration,
    };
  },
};
</script>

<style scoped>
.newsaggregator-rss {
  font-family: 'Russo One', system-ui, sans-serif;
  font-weight: bold;
  font-size: larger;
```

```
}
</style>
```

Файл ControlPanel.vue

Реалізація функціональної вимоги виходу з акаунту

```
<template>
  <v-toolbar>
    <v-toolbar-items class="overflow-auto flex-row-reverse" style="justify-
content: space-around;align-content: space-around;">
      <LogoutButton v-if="isAuthenticated" @logout="$emit('logout')" />
      <v-btn v-if="isAuthenticated" v-show="showNotificationWarning"
:title="$t('pleaseEnableNotifications')" icon="fa-bell" :size="buttonSize" />
      <SettingsButton v-if="isAuthenticated"
@showSettings="$emit('showSettings')" />
      <DisplayModeButton />
    </v-toolbar-items>
  </v-toolbar>
</template>

<script>
import DisplayModeButton from '@/components/generic/DisplayModeButton.vue';
import LogoutButton from '@/components/control-panel/LogoutButton.vue';
import SettingsButton from '@/components/control-panel/SettingsButton.vue';
import buttonSizeMixin from '@/mixins/buttonSizeMixin';

export default {
  name: 'ControlPanel',
  components: {
    LogoutButton,
    SettingsButton,
    DisplayModeButton,
  },
  mixins: [buttonSizeMixin],
  props: {
    showSettingsPanel: { type: Boolean, default: false },
    showHelpPanel: { type: Boolean, default: false },
    showNotificationWarning: { type: Boolean, default: false },
    isAuthenticated: { type: Boolean, default: false },
  },
  emits: ['logout', 'showSettings'],
};
</script>
```

Файл SubscriptionDefinitionService.java

Реалізація функціональної вимоги створення та управління підписками

```
package main_app.query;

import com.google.gson.JsonObject;
import main_app.model.request.SubscriptionConfigRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.io.Serializable;
import java.util.*;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Collectors.toMap;
import static org.apache.commons.collections4.CollectionUtils.isNotEmpty;
import static org.apache.commons.lang3.StringUtils.isBlank;
import static org.apache.commons.lang3.StringUtils.isNotBlank;

@Service
public class SubscriptionDefinitionService {

    @Autowired
    private SubscriptionDefinitionDao subscriptionDefinitionDao;
```

```
    public SubscriptionDefinition findById(String username, Long id) throws
DataAccessException {
        return subscriptionDefinitionDao.findById(username, id);
    }

    public List<SubscriptionDefinition> findByUsername(String username)
throws DataAccessException {
        return subscriptionDefinitionDao.findByUsername(username);
    }

    public List<SubscriptionDefinition> findByQueueId(String username, Long
id) throws DataAccessException {
        return subscriptionDefinitionDao.findByQueueId(username, id);
    }

    public List<SubscriptionDefinition> updateSubscriptions(String username,
Long queueId, List<SubscriptionConfigRequest> subscriptions) throws
DataAccessException {
        Map<Long, SubscriptionDefinition> currentSubscriptions =
subscriptionDefinitionDao.findByQueueId(username, queueId, RSS)
                .stream()
                .collect(toMap(SubscriptionDefinition::getId, v -> v));

        List<SubscriptionDefinition> updatedSubscriptions = new
ArrayList<>();

        if (isNotEmpty(subscriptions)) {
            for (SubscriptionConfigRequest request : subscriptions) {
                if (request.getId() != null) {
                    SubscriptionDefinition subscription =
currentSubscriptions.get(request.getId());
                    if (subscription != null && needsUpdate(request,
subscription)) {
                        subscription.setTitle(request.getTitle());
                        subscription.setImgUrl(request.getImgUrl());
                        subscription.setUrl(request.getUrl());

subscription.setQueryConfig(isNotBlank(request.getUsername()) ||
isNotBlank(request.getPassword()) ? serializeQueryConfig(request) : null);
                        updatedSubscriptions.add(subscription);
                    }
                }
            }
            if (isNotEmpty(updatedSubscriptions)) {

subscriptionDefinitionDao.updateSubscriptions(updatedSubscriptions.stream()
                        .map(sub -> new Object[]{
                                sub.getTitle(),
                                sub.getImgUrl(),
                                sub.getUrl(),
                                sub.getQueryType(),
                                sub.getImportSchedule(),
                                sub.getQueryConfig(),
                                sub.getId()
                        })
                        .collect(toList()));
            }
        }

        return updatedSubscriptions;
    }

    public List<SubscriptionDefinition> addSubscriptions(String username,
```

```
Long queueId, List<SubscriptionConfigRequest> subscriptions) throws
DataAccessException, DataUpdateException, DataConflictException {
        return isNotEmpty(subscriptions) ? doAddSubscriptions(username,
queueId, subscriptions) : new ArrayList<>();
    }

    private List<SubscriptionDefinition> doAddSubscriptions(String username,
Long queueId, List<SubscriptionConfigRequest> subscriptions) throws
DataAccessException, DataUpdateException, DataConflictException {
        List<SubscriptionDefinition> additions = new ArrayList<>();
        for (SubscriptionConfigRequest request : subscriptions) {
            if (isBlank(request.getUrl())) {
                log.warn("Subscription request missing URL, skipping...");
                continue;
            }
            SubscriptionDefinition newSubscription =
SubscriptionDefinition.from(
                    queueId, username, request.getTitle(),
request.getImgUrl(), request.getUrl(), RSS, "A",
serializeQueryConfig(request));
            additions.add(newSubscription);
        }
        List<Long> subscriptionIds =
subscriptionDefinitionDao.add(additions);
        return subscriptionDefinitionDao.findByIds(username,
subscriptionIds);
    }

    public List<SubscriptionDefinition> createQueries(String username, Long
queueId, List<SubscriptionConfigRequest> subscriptions) throws
DataAccessException, DataUpdateException, DataConflictException {
        List<SubscriptionDefinition> createdSubscriptions = new
ArrayList<>();
        if (isNotEmpty(subscriptions)) {
            List<SubscriptionDefinition> toImport = new ArrayList<>();
            List<String> existingSubscriptionUrls =
subscriptionDefinitionDao.getSubscriptionUrlsByUsername(username);

            for (SubscriptionConfigRequest request : subscriptions) {
                if (containsIgnoreCase(existingSubscriptionUrls,
request.getUrl())) {
                    log.debug("Subscription already defined for user, url={},
username={}", request.getUrl(), username);
                    continue;
                }
                SubscriptionDefinition newSubscription =
SubscriptionDefinition.from(
                        queueId, username, request.getTitle(),
request.getImgUrl(), request.getUrl(), RSS, "A",
serializeQueryConfig(request));
                toImport.add(newSubscription);
            }
            if (isNotEmpty(toImport)) {
                List<Long> subscriptionIds =
subscriptionDefinitionDao.add(toImport);
                toImport = subscriptionDefinitionDao.findByIds(username,
subscriptionIds);
                createdSubscriptions.addAll(toImport);
            }
        }

        return createdSubscriptions;
    }
```

```
    public void deleteSubscriptionById(String username, Long queueId, Long
subscriptionId) throws DataAccessException, DataUpdateException {
        subscriptionDefinitionDao.deleteById(username, queueId,
subscriptionId);
    }

    private boolean needsUpdate(SubscriptionConfigRequest request,
SubscriptionDefinition subscription) {
        return !Objects.equals(subscription.getTitle(), request.getTitle())
||
                !Objects.equals(subscription.getImgUrl(),
request.getImgUrl()) ||
                !Objects.equals(subscription.getUrl(), request.getUrl()) ||
                !Objects.equals(serializeQueryConfig(request),
subscription.getQueryConfig());
    }

    private Serializable serializeQueryConfig(SubscriptionConfigRequest
request) {
        JsonObject jsonObject = new JsonObject();
        jsonObject.addProperty("username", request.getUsername());
        jsonObject.addProperty("password", request.getPassword());
        return jsonObject.toString();
    }

    private static boolean containsIgnoreCase(List<String> list, String str)
{
        return isNotEmpty(list) && list.stream().anyMatch(s ->
s.equalsIgnoreCase(str));
    }
}
```

Файл QueueConfigPanel.vue

Реалізація функціональної вимоги додавання і управління підписками

```
<template>
  <v-card>
    <v-card-title class="text-center" :class="pa4r">
      <h3 class="queue-settings">
        <v-icon icon="fa-rss" />
        {{ queueId ? queueTitle : $t('createANewQueue') }}
      </h3>
    </v-card-title>
    <v-card-subtitle v-if="queueId" class="text-center" :class="pa4r">
      {{ queueDescription }}
    </v-card-subtitle>
    <v-divider />
    <v-card-text>
      <!-- tab panel -->
      <v-tabs v-if="queueId" v-model="selectedTab">
        <v-tab key="ADD_SUBSCRIPTION" value="ADD_SUBSCRIPTION">
          {{ $t('addSubscriptions') }}
        </v-tab>
        <v-tab key="MANAGE_SUBSCRIPTIONS" value="MANAGE_SUBSCRIPTIONS">
          {{ $t('manageSubscriptions', { ct: subscriptions ?
subscriptions.length : 0 }) }}
        </v-tab>
        <v-tab key="QUEUE_PROPERTIES" value="QUEUE_PROPERTIES">
          {{ $t('queueProperties') }}
        </v-tab>
      </v-tabs>
      <!-- config window -->
      <v-window v-model="selectedTab">
        <v-window-item key="ADD_SUBSCRIPTION" value="ADD_SUBSCRIPTION">
          <v-sheet align="left" justify="center">
            <v-card :class="ma4r" elevation="0">
```

```
              <v-divider />
              <v-card-text v-show="showSubscriptions" border="top" icon="fa-
question-circle"></v-card-text>
              <v-card-text v-show="showAddSubscription">
                <v-text-field
                    v-model="newSubscription.url"
                    style="min-width: 128px;"
                    variant="solo-filled"
                    type="text"
                    autofocus
                    :label="$t('feedUrl')"
                    :placeholder="$t('feedUrl')"
                />
                <v-btn-group class="my-2 d-flex flex-wrap">
                  <v-btn
                      variant="tonal"
                      :size="buttonSize"
                      prepend-icon="fa-refresh"
                      :loading="discoveryIsLoading"
                      :disabled="!newSubscription.url"
                      :title="$t('discovery')"
                      :text="$t('discovery')"
                      @click="doDiscovery(newSubscription)"
                  />
                </v-btn-group>
                <v-text-field
                    v-if="showNewSubscriptionAuthConfig"
                    v-model="newSubscription.username"
                    class="my-2"
                    variant="solo-filled"
                    type="text"
                    :label="$t('username')"
                    :placeholder="$t('username')"
                    :disabled="!newSubscription.url"
                />
                <v-text-field
                    v-if="showNewSubscriptionAuthConfig"
                    v-model="newSubscription.password"
                    variant="solo-filled"
                    class="my-2"
                    type="password"
                    :label="$t('password')"
                    :placeholder="$t('password')"
                    :disabled="!newSubscription.url"
                />
                <SubscriptionInfo
                    v-if="newSubscription.discoveryUrl &&
!newSubscription.error"
                    elevation="6"
                    :info="newSubscription"
                >
                  <template #additional>
                    <v-btn
                        v-
if="alreadySubscribed(newSubscription.discoveryUrl)"
                        :size="buttonSize"
                        :disabled="true"
                        prepend-icon="fa-plus"
                        :title="$t('alreadySubscribed')"
                        :text="$t('alreadySubscribed')"
                    />
                    <v-btn
                        v-else
                        :size="buttonSize"
```

```
                        prepend-icon="fa-plus"
                        :title="$t('subscribe')"
                        :text="$t('subscribe')"
                        @click="addSubscription"
                      />
                    </template>
                  </SubscriptionInfo>
                  <v-alert v-if="newSubscription.error" closable type="error">
                    {{ newSubscription.error }}
                  </v-alert>
                </v-card-text>
              </v-card>
            </v-sheet>
          </v-window-item>
          <v-window-item key="MANAGE_SUBSCRIPTIONS"
value="MANAGE_SUBSCRIPTIONS">
            <v-sheet align="left" justify="center">
              <v-card v-if="subscriptions && subscriptions.length > 0"
:class="ma4r" elevation="0">
                <v-divider />
                <v-card-text
                    v-for="(subscription, idx) in subscriptions"
                    v-show="showSubscriptions"
                    :key="idx"
                >
                  <SubscriptionInfo elevation="6" :info="subscription" :filter-
support="false">
                    <template #additional>
                      <v-btn
                          :size="buttonSize"
                          prepend-icon="fa-trash"
                          :title="$t('unsubscribe')"
                          :text="$t('unsubscribe')"
                          @click="deleteSubscription(subscription.id)"
                      />
                    </template>
                  </SubscriptionInfo>
                </v-card-text>
              </v-card>
              <v-dialog v-model="showAuthConfig">
                <v-card>
                  <v-card-title :class="pa4r">
                    {{ $t('updateAuth', { subscriptionName:
subscriptionToUpdate.title }) }}
                  </v-card-title>
                  <v-divider />
                  <v-card-text>
                    <div :class="mb4r">
                      {{ $t('credentialsUseMessage') }}
                    </div>
                    <v-text-field
                        v-model="subscriptionToUpdate.username"
                        variant="solo-filled"
                        type="text"
                        :label="$t('username')"
                        :placeholder="$t('username')"
                    />
                    <v-text-field
                        v-model="subscriptionToUpdate.password"
                        variant="solo-filled"
                        type="password"
                        :label="$t('password')"
                        :placeholder="$t('password')"
                    />
```

```
                </v-card-text>
                <v-card-actions>
                  <v-btn
                      :size="buttonSize"
                      prepend-icon="fa-save"
                      :text="$t('update')"
                      @click="updateSubscriptionAuth(subscriptionToUpdate)"
                  />
                  <v-btn
                      :size="buttonSize"
                      :text="$t('cancel')"
                      @click="showAuthConfig = false"
                  />
                </v-card-actions>
              </v-card>
            </v-dialog>
          </v-sheet>
        </v-window-item>
        <v-window-item key="QUEUE_PROPERTIES" value="QUEUE_PROPERTIES">
          <v-sheet>
            <v-card variant="flat">
              <v-card-title :class="pa4r">
                {{ queueId ? $t('updateQueueSettings') :
$t('createANewQueue') }}
              </v-card-title>
              <v-card-subtitle v-if="queueTitle">
                {{ queueTitle }}
              </v-card-subtitle>
              <v-divider class="my-1" />
              <v-card-text>
                <v-text-field
                    v-model="queueIdent"
                    :class="mb4r"
                    :label="$t('queueIdentifier')"
                    :hint="$t('queueIdentifierHint')"
                    :required="true"
                    :placeholder="$t('queueIdentifier')"
                    persistent-placeholder
                    variant="solo-filled"
                />
                <v-text-field
                    v-model="queueTitle"
                    :class="mb4r"
                    :label="$t('queueTitle')"
                    :hint="$t('queueTitleHint')"
                    :required="false"
                    :placeholder="$t('queueTitle')"
                    persistent-placeholder
                    variant="solo-filled"
                />
                <v-text-field
                    v-model="queueDescription"
                    :class="mb4r"
                    :label="$t('queueDescription')"
                    :hint="$t('queueDescriptionHint')"
                    :required="false"
                    :placeholder="$t('queueDescription')"
                    persistent-placeholder
                    variant="solo-filled"
                />
              </v-card-text>
              <v-divider />
              <v-card-actions>
                <v-btn
```

```
                    v-if="!queueId"
                    :size="buttonSize"
                    :text="$t('save')"
                    @click="save"
                />
                <v-btn
                    v-if="queueId"
                    :size="buttonSize"
                    :text="$t('update')"
                    @click="update"
                />
              </v-card-actions>
            </v-card>
          </v-sheet>
        </v-window-item>
      </v-window>
    </v-card-text>
    <v-divider />
    <v-card-actions>
      <v-btn
          :size="buttonSize"
          :text="$t('dismiss')"
          @click="$emit('dismiss')"
      />
    </v-card-actions>
  </v-card>
</template>

<script>
import { inject, computed, ref, reactive, onMounted } from 'vue';
import { useI18n } from 'vue-i18n';
import { useNotifications } from '@/composable/useNotifications';
import { useQueues } from '@/composable/useQueues';
import SubscriptionInfo from './SubscriptionInfo.vue';
import spacingMixin from '@/mixins/spacingMixin';
import buttonSizeMixin from '@/mixins/buttonSizeMixin';

export default {
  name: "QueueConfigPanel",
  components: {
    SubscriptionInfo,
  },
  mixins: [spacingMixin, buttonSizeMixin],
  props: {
    baseUrl: { type: String, required: true },
  },
  emits: [
    "dismiss",
    "save",
    "update",
    "addSubscription",
    "deleteSubscription",
    "updateSubscriptionAuth",
  ],
  setup(props, { emit }) {
    const auth = inject('auth');
    const { shouldShowAlert, dismissAlert, handleServerError,
setLastServerMessage } = useNotifications();
    const { queueStore } = useQueues(props);
    const { t } = useI18n();

    const tabModel = computed(() => {
      let arr = [];
      if (queueId.value) {
```

```
        arr.push({ name: "MANAGE_SUBSCRIPTIONS", description:
t('manageSubscriptions', { ct: subscriptions ? subscriptions.length : 0 }),
icon: "feed" });
        arr.push({ name: "QUEUE_PROPERTIES", description:
t('queueProperties'), icon: "list" });
      }
      return arr;
    });

    const queueId = ref(queueStore.queueUnderConfig.id);
    const queueIdent = ref(queueStore.queueUnderConfig.ident);
    const queueTitle = ref(queueStore.queueUnderConfig.title);
    const queueDescription = ref(queueStore.queueUnderConfig.description);
    const queueGenerator = ref(queueStore.queueUnderConfig.generator);
    const queueCopyright = ref(queueStore.queueUnderConfig.copyright);
    const queueLanguage = ref(queueStore.queueUnderConfig.language);
    const subscriptions =
reactive(queueStore.queueUnderConfig.subscriptions);
    const showAddSubscription = ref(true);
    const showSubscriptions = ref(true);
    const newSubscription = ref({});
    const discoveryIsLoading = ref(false);
    const subscriptionToUpdate = ref(null);
    const showNewSubscriptionAuthConfig = ref(false);
    const showAuthConfig = ref(false);
    const selectedTab = ref(null);

    onMounted(() => {
      selectedTab.value = queueId.value ? 'MANAGE_SUBSCRIPTIONS' :
'QUEUE_PROPERTIES';
    });

    function alreadySubscribed(url) {
      if (url && subscriptions) {
        const normalizedUrl = url.trim().toLowerCase();
        return subscriptions.some(sub => sub.url &&
sub.url.trim().toLowerCase() === normalizedUrl);
      }
      return false;
    }

    function save() {
      const saveObj = {
        ident: queueIdent.value,
        title: queueTitle.value,
        description: queueDescription.value,
        generator: queueGenerator.value,
        copyright: queueCopyright.value,
        language: queueLanguage.value,
      };
      emit('save', saveObj);
    }

    function update() {
      const saveObj = {
        id: queueId.value,
        ident: queueIdent.value,
        title: queueTitle.value,
        description: queueDescription.value,
        generator: queueGenerator.value,
        copyright: queueCopyright.value,
        language: queueLanguage.value,
      };
      emit('update', saveObj);
```

```
    }

    function addSubscription() {
      newSubscription.value.discoveryUrl = null;
      emit('addSubscription', newSubscription.value);
    }

    function deleteSubscription(id) {
      emit('deleteSubscription', id);
    }

    function updateSubscriptionAuth(source) {
      emit('updateSubscriptionAuth', source);
    }

    function doDiscovery(subscription) {
      if (!subscription.url) return;
      subscription.error = null;
      discoveryIsLoading.value = true;
      auth.getTokenSilently().then(token => {
        const controller = new AbortController();
        const requestOptions = {
          method: 'POST',
          headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${token}`
          },
          credentials: 'include',
          body: JSON.stringify({
            url: subscription.url,
            username: subscription.username,
            password: subscription.password,
          }),
          signal: controller.signal
        };
        const timeoutId = setTimeout(() => controller.abort(), 45000);
        fetch(`${props.baseUrl}/discovery`, requestOptions)
            .then(response => response.json())
            .then(data => {
              if (data.error) {
                subscription.error = data.error;
              } else {
                Object.assign(subscription, data, { discoveryUrl:
data.feedUrl });
              }
            })
            .catch(error => {
              subscription.error = error.cause ? error.cause.details :
error.message;
            })
            .finally(() => {
              discoveryIsLoading.value = false;
              clearTimeout(timeoutId);
            });
      }).catch(error => {
        handleServerError(error);
        discoveryIsLoading.value = false;
      });
    }

    function configAuth(subscription) {
      subscriptionToUpdate.value = { ...subscription };
      showAuthConfig.value = true;
    }
```

```
    return {
      auth,
      shouldShowAlert,
      dismissAlert,
      handleServerError,
      setLastServerMessage,
      tabModel,
      queueId,
      queueIdent,
      queueTitle,
      queueDescription,
      queueGenerator,
      queueCopyright,
      queueLanguage,
      subscriptions,
      showAddSubscription,
      showSubscriptions,
      newSubscription,
      discoveryIsLoading,
      subscriptionToUpdate,
      showNewSubscriptionAuthConfig,
      showAuthConfig,
      selectedTab,
      configAuth,
      doDiscovery,
      updateSubscriptionAuth,
      deleteSubscription,
      addSubscription,
      update,
      save,
      alreadySubscribed,
    };
  },
};
</script>

<style scoped>
.queue-settings {
  font-family: "Russo One", system-ui, sans-serif;
  font-weight: bold;
  font-size: larger;
}
</style>
```

Файл QueueDefinitionController.java

Реалізація функціональної вимоги створення підкатегорій

```
package main_app;

import main_app.audit.AppLogService;
import main_app.feed.QueueDefinitionService;
import main_app.model.request.FeedStatusUpdateRequest;
import main_app.model.request.QueueConfigRequest;
import main_app.model.request.SubscriptionConfigRequest;
import main_app.model.response.*;
import main_app.proxy.ProxyService;
import main_app.resolution.FeedResolutionService;
import main_app.thumbnail.ThumbnailService;
import org.apache.commons.lang3.time.StopWatch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;

import java.util.*;
import java.util.stream.Collectors;

import static org.springframework.http.ResponseEntity.ok;

@RestController
@Validated
public class QueueDefinitionController {

    @Autowired
    private AppLogService appLogService;

    @Autowired
    private QueueDefinitionService queueDefinitionService;

    @Autowired
    private ThumbnailService thumbnailService;

    @Autowired
    private ProxyService proxyService;

    @Autowired
    private FeedResolutionService feedResolutionService;

    @Autowired
    private Validator validator;

    @Value("${newsaggregator.thumbnail.size}")
    private int thumbnailSize;

    @GetMapping("/queues")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    public ResponseEntity<QueueFetchResponse> getQueues(Authentication
authentication) throws DataAccessException {
        String username = getUsername(authentication);
        log.debug("getFeeds for user={}", username);
        StopWatch stopWatch = StopWatch.createStarted();

        List<SubscriptionDefinition> allSubscriptionDefinitions =
subscriptionDefinitionService.findByUsername(username);
        Map<Long, List<ThumbnailedSubscriptionDefinition>>
subscriptionDefinitionsByQueueId = allSubscriptionDefinitions.stream()

.collect(Collectors.groupingBy(SubscriptionDefinition::getQueueId,
Collectors.mapping(this::addThumbnail, Collectors.toList())));

        List<SubscriptionMetricsWithErrorDetails> allSubscriptionMetrics =
subscriptionMetricsService.findByUsername(username).stream()
                .map(q -> SubscriptionMetricsWithErrorDetails.from(q,
getQueryExceptionTypeMessage(q.getErrorType())))
                .collect(Collectors.toList());
        Map<Long, List<SubscriptionMetricsWithErrorDetails>>
metricsBySubscriptionDefinitionId = allSubscriptionMetrics.stream()

.collect(Collectors.groupingBy(SubscriptionMetricsWithErrorDetails::getSubscr
iptionId));

        List<QueueDefinition> queueDefinitions =
queueDefinitionService.findByUser(username);
        Map<Long, RuleSet> queueImportRuleSets =
ruleSetService.findQueueImportRulesSetsByUsername(username);
```

```
        Map<Long, RuleSet> subscriptionImportRuleSets =
ruleSetService.findSubscriptionImportRuleSetsByUsername(username);

        stopWatch.stop();
        appLogService.logFeedFetch(username, stopWatch,
queueDefinitions.size(), subscriptionDefinitionsByQueueId.size());

        return ok(QueueFetchResponse.from(queueDefinitions,
subscriptionDefinitionsByQueueId, metricsBySubscriptionDefinitionId,
queueImportRuleSets, subscriptionImportRuleSets));
    }

    @PostMapping("/queues/")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    public ResponseEntity<List<QueueConfigResponse>> createQueue(@RequestBody
@Valid QueueConfigRequest queueConfigRequest, Authentication authentication)
throws DataAccessException, DataUpdateException, DataConflictException {
        String username = getUsername(authentication);
        log.debug("createFeed adding queue for user={}", username);
        StopWatch stopWatch = StopWatch.createStarted();
        Long queueId = queueDefinitionService.createFeed(username,
queueConfigRequest);

        handleSubscriptionCreation(queueConfigRequest.getSubscriptions(),
username, queueId);

        QueueDefinition createdQueue =
queueDefinitionService.findByQueueId(username, queueId);
        List<ThumbnailedSubscriptionDefinition> subscriptionDefinitions =
addThumbnails(subscriptionDefinitionService.findByQueueId(username,
queueId));
        List<QueueConfigResponse> queueConfigResponse =
List.of(QueueConfigResponse.from(createdQueue, subscriptionDefinitions,
buildThumbnail(createdQueue)));

        stopWatch.stop();
        appLogService.logFeedCreate(username, stopWatch, queueConfigRequest,
createdQueue.getId());

        return ok(queueConfigResponse);
    }

    @PostMapping("/queues/{queueId}/subscriptions/")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    public ResponseEntity<SubscriptionConfigResponse> addQueries(@RequestBody
List<@Valid SubscriptionConfigRequest> subscriptions, @PathVariable Long
queueId, Authentication authentication) {
        String username = getUsername(authentication);
        log.debug("addQueries adding {} queries for user={}, queueId={}",
subscriptions.size(), username, queueId);
        StopWatch stopWatch = StopWatch.createStarted();

        validateSubscriptions(subscriptions);
        SubscriptionConfigResponse subscriptionConfigResponse = null;

        try {
            ImmutableMap<String, FeedDiscoveryInfo> discoveryCache =
feedResolutionService.resolveIfNecessary(subscriptions);
            List<SubscriptionDefinition> createdQueries =
subscriptionDefinitionService.createQueries(username, queueId,
subscriptions);
            importFromCache(createdQueries, discoveryCache);

            subscriptionConfigResponse =
```

```
SubscriptionConfigResponse.from(addThumbnails(createdQueries));
            stopWatch.stop();
            appLogService.logAddQueries(username, stopWatch,
createdQueries.size());
        } catch (Exception e) {
            log.warn("Query initial import failed due to: {}",
e.getMessage());
        }

        return ok(subscriptionConfigResponse);
    }

    @PutMapping("/queues/{id}")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @Transactional
    public ResponseEntity<QueueConfigResponse>
updateQueue(@PathVariable("id") Long id, @Valid @RequestBody
QueueConfigRequest queueConfigRequest, Authentication authentication) throws
DataAccessException, DataUpdateException, DataConflictException {
        String username = getUsername(authentication);
        log.debug("updateFeed for user={}, queueId={}", username, id);
        StopWatch stopWatch = StopWatch.createStarted();

        queueDefinitionService.updateFeed(username, id, queueConfigRequest);
        QueueDefinition queueDefinition =
queueDefinitionService.findByQueueId(username, id);
        List<ThumbnailedSubscriptionDefinition> subscriptionDefinitions =
addThumbnails(subscriptionDefinitionService.findByQueueId(username, id));
        byte[] thumbnail = buildThumbnail(queueDefinition);

        stopWatch.stop();
        appLogService.logFeedUpdate(username, stopWatch, id);

        return ok(QueueConfigResponse.from(queueDefinition,
subscriptionDefinitions, thumbnail));
    }

    @PutMapping("/queues/{queueId}/subscriptions/{subscriptionId}")
    public ResponseEntity<SubscriptionConfigResponse>
updateQuery(@PathVariable("queueId") Long queueId,
@PathVariable("subscriptionId") Long subscriptionId, @Valid @RequestBody
SubscriptionConfigRequest subscription, Authentication authentication) {
        String username = getUsername(authentication);
        log.debug("updateQuery updating queueId={}, subscriptionId={} for
user={}", queueId, subscriptionId, username);
        StopWatch stopWatch = StopWatch.createStarted();
        SubscriptionConfigResponse subscriptionConfigResponse = null;

        try {
            List<SubscriptionConfigRequest> firstPartition =
List.of(subscription);
            ImmutableMap<String, FeedDiscoveryInfo> discoveryCache =
feedResolutionService.resolveIfNecessary(firstPartition);
            List<SubscriptionDefinition> toImport =
subscriptionDefinitionService.updateSubscriptions(username, queueId,
firstPartition);
            importFromCache(toImport, discoveryCache);

            SubscriptionDefinition subscriptionDefinition =
subscriptionDefinitionService.findById(username, subscriptionId);
            List<SubscriptionDefinition> updatedSubscriptions =
List.of(subscriptionDefinition);
            subscriptionConfigResponse =
SubscriptionConfigResponse.from(addThumbnails(updatedSubscriptions));
```

```
            stopWatch.stop();
            appLogService.logUpdateSubscriptions(username, stopWatch,
updatedSubscriptions.size());
        } catch (Exception e) {
            log.warn("Query initial import failed due to: {}",
e.getMessage());
        }

        return ok(subscriptionConfigResponse);
    }

    @PutMapping("/queues/status/{id}")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @Transactional
    public ResponseEntity<?> updateFeedStatus(@PathVariable("id") Long id,
@Valid @RequestBody FeedStatusUpdateRequest feedStatusUpdateRequest,
Authentication authentication) throws DataAccessException,
DataUpdateException {
        String username = getUsername(authentication);
        log.debug("updateFeedStatus for user={}, postId={},
feedStatusUpdateRequest={}", username, id, feedStatusUpdateRequest);
        StopWatch stopWatch = StopWatch.createStarted();

        queueDefinitionService.updateFeedStatus(username, id,
feedStatusUpdateRequest);
        stopWatch.stop();
        appLogService.logFeedStatusUpdate(username, stopWatch, id,
feedStatusUpdateRequest, 1);

        return ok().body(buildResponseMessage("Successfully updated feed Id "
+ id));
    }


    @DeleteMapping("/queues/{id}")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @Transactional
    public ResponseEntity<?> deleteFeedById(@PathVariable Long id,
Authentication authentication) throws DataAccessException,
DataUpdateException {
        String username = getUsername(authentication);
        log.debug("deleteFeedById for user={}", username);
        StopWatch stopWatch = StopWatch.createStarted();

        queueDefinitionService.deleteById(username, id);
        stopWatch.stop();
        appLogService.logFeedDelete(username, stopWatch, 1);

        return ok().body(buildResponseMessage("Deleted feed Id " + id));
    }

    @DeleteMapping("/queues/{queueId}/subscriptions/{subscriptionId}")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @Transactional
    public ResponseEntity<?> deleteSubscriptionById(@PathVariable Long
queueId, @PathVariable Long subscriptionId, Authentication authentication)
throws DataAccessException, DataUpdateException {
        String username = getUsername(authentication);
        log.debug("deleteQueryById for user={}, queueId={},
subscriptionId={}", username, queueId, subscriptionId);
        StopWatch stopWatch = StopWatch.createStarted();

        subscriptionDefinitionService.deleteSubscriptionById(username,
```

```
queueId, subscriptionId);
        stopWatch.stop();
        appLogService.logQueryDelete(username, stopWatch, 1);

        return ok().body(buildResponseMessage("Deleted query Id " +
subscriptionId + ", feed Id " + queueId));
    }


    private String getUsername(Authentication authentication) {
        return ((UserDetails) authentication.getDetails()).getUsername();
    }

    private List<ThumbnailedSubscriptionDefinition>
addThumbnails(List<SubscriptionDefinition> subscriptionDefinitions) {
        return
subscriptionDefinitions.stream().map(this::addThumbnail).collect(Collectors.t
oList());
    }

    private byte[] buildThumbnail(QueueDefinition f) throws
DataAccessException {
        return Optional.ofNullable(f.getQueueImgSrc())
                .filter(src -> !src.isBlank())
                .map(src -> {
                    String transportIdent = f.getQueueImgTransportIdent();
                    return
Optional.ofNullable(thumbnailService.getThumbnail(transportIdent)).map(Render
edThumbnail::getImage).orElseGet(() ->
thumbnailService.refreshThumbnailFromSrc(transportIdent, src).getImage());
                })
                .orElse(null);
    }

    private void validateThumbnail(byte[] image) {
        if (image == null) {
            throw new ValidationException("This format isn't supported.");
        }
    }

    private void handleSubscriptionCreation(List<SubscriptionConfigRequest>
subscriptions, String username, Long queueId) throws DataAccessException,
DataUpdateException, DataConflictException {
        if (subscriptions != null && !subscriptions.isEmpty()) {
            try {
                ImmutableMap<String, FeedDiscoveryInfo> discoveryCache =
feedResolutionService.resolveIfNecessary(subscriptions);
                List<SubscriptionDefinition> createdQueries =
subscriptionDefinitionService.createQueries(username, queueId,
subscriptions);
                importFromCache(createdQueries, discoveryCache);
            } catch (DataAccessException | DataUpdateException |
DataConflictException e) {
                log.warn("Feed initial import failed due to: {}",
e.getMessage());
            }
        }
    }

    private void validateSubscriptions(List<SubscriptionConfigRequest>
subscriptions) {
        for (SubscriptionConfigRequest subscription : subscriptions) {
            Set<ConstraintViolation<SubscriptionConfigRequest>>
constraintViolations = validator.validate(subscription);
```

```
            if (!constraintViolations.isEmpty()) {
                throw new
SubscriptionValidationException(constraintViolations);
            }
        }
    }


    public static class SubscriptionValidationException extends
ValidationException {

SubscriptionValidationException(Set<ConstraintViolation<SubscriptionConfigReq
uest>> constraintViolations) {

super(constraintViolations.stream().map(ConstraintViolation::getMessage).coll
ect(Collectors.joining("; ")));
        }
    }
}
```

Файл StagingPostController.java

Реалізація функціональної вимоги завантаження нових постів

```
package main_app;

import main_app.audit.AppLogService;
import main_app.model.request.PostStatusUpdateRequest;
import main_app.model.response.PostFetchResponse;
import main_app.model.response.ThumbnailedPostResponse;
import main_app.post.StagingPostService;
import main_app.proxy.ProxyService;
import org.apache.commons.lang3.time.StopWatch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.stream.Collectors;

@RestController
@Validated
public class StagingPostController {

    @Autowired
    private AppLogService appLogService;

    @Autowired
    private StagingPostService stagingPostService;

    @Autowired
    private ProxyService proxyService;

    @GetMapping("/staging")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    public ResponseEntity<PostFetchResponse>
getStagingPosts(@RequestParam(required = false) List<Long> queueIds,
Authentication authentication) throws DataAccessException {
        String username = getUsername(authentication);
        log.debug("getStagingPosts for user={}, queueIds={}", username,
isEmpty(queueIds) ? "all" : queueIds);
        StopWatch stopWatch = StopWatch.createStarted();

        List<ThumbnailedPostResponse> stagingPosts =
```

```
proxyService.secureStagingPosts(stagingPostService.getStagingPosts(username,
queueIds))
                .stream()
                .map(this::addThumbnail)
                .collect(Collectors.toList());

        stopWatch.stop();
        appLogService.logStagingPostFetch(username, stopWatch, queueIds !=
null ? queueIds.size() : 0, stagingPosts.size());
        return ok(PostFetchResponse.from(stagingPosts));
    }

    @PutMapping("/staging/read-status/post/{id}")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @Transactional
    public ResponseEntity<?> updatePostReadStatus(@PathVariable Long id,
@Valid @RequestBody PostStatusUpdateRequest postStatusUpdateRequest,
Authentication authentication) throws DataAccessException,
DataUpdateException {
        return updateStatus(id, postStatusUpdateRequest, authentication,
true);
    }

    @PutMapping("/staging/read-status/queue/{id}")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    @Transactional
    public ResponseEntity<?> updateFeedReadStatus(@PathVariable Long id,
@Valid @RequestBody PostStatusUpdateRequest postStatusUpdateRequest,
Authentication authentication) throws DataAccessException,
DataUpdateException {
        return updateStatus(id, postStatusUpdateRequest, authentication,
false);
    }

    private ResponseEntity<?> updateStatus(Long id, PostStatusUpdateRequest
postStatusUpdateRequest, Authentication authentication, boolean isPost)
throws DataAccessException, DataUpdateException {
        String username = getUsername(authentication);
        log.debug("updateStatus for user={}, id={}, request={}", username,
id, postStatusUpdateRequest);
        StopWatch stopWatch = StopWatch.createStarted();

        if (isPost) {
            stagingPostService.updatePostReadStatus(username, id,
postStatusUpdateRequest);
            appLogService.logStagingPostReadStatusUpdate(username, stopWatch,
id, postStatusUpdateRequest, 1);
        } else {
            stagingPostService.updateQueueReadStatus(username, id,
postStatusUpdateRequest);
            appLogService.logFeedReadStatusUpdate(username, stopWatch, id,
postStatusUpdateRequest, 1);
        }

        stopWatch.stop();
        return ok().body(buildResponseMessage("Successfully updated " +
(isPost ? "post" : "feed") + " Id " + id));
    }

    private ThumbnailedPostResponse addThumbnail(StagingPost s) {
        String imageProxyUrl = buildThumbnailProxyUrl(s);
        return ThumbnailedPostResponse.from(s, imageProxyUrl);
    }
```

```
    private String buildThumbnailProxyUrl(StagingPost s) {
        return isNotBlank(s.getPostImgUrl()) ?
proxyService.rewriteImageUrl(s.getPostImgUrl(), s.getPostUrl()) : null;
    }

    private String getUsername(Authentication authentication) {
        return ((UserDetails) authentication.getDetails()).getUsername();
    }
}
```

Файл Postcard.vue

Реалізація функціональної вимоги читання постів і відмічання їх як прочитані або прочитати потім

```
<template>
  <v-card elevation="6" :class="{ 'post-read': post.isRead }">
    <!-- title + thumbnail -->
    <v-card-title
      class="d-flex flex-row flex-auto flex-wrap align-start overflow-auto
gap-1"
      :class="[pa4r, collapsible ? 'clickable' : '']"
      style="white-space: normal"
      @click="$event => { if (collapsible) { showFullPost = !showFullPost; }
}"
    >
      <v-img
        v-if="post.postImgSrc"
        class="post-thumbnail rounded"
        :src="post.postImgSrc"
        :alt="$t('postThumbnailImage')"
        width="140"
        max-height="140"
        max-width="140"
      />
      <div class="d-flex flex-column flex-auto flex-grow-1">
        <div
          v-if="isHtmlContent(post.postTitle)"
          v-dompurify-html="post.postTitle.value"
          class="post-html-frame"
          frameborder="0"
        />
        <div v-else class="post-text-frame">
          {{ post.postTitle.value }}
        </div>
        <v-divider class="my-1" />
        <div class="d-flex flex-row flex-wrap justify-space-between">
          <div>
            <div
              v-if="post.lastUpdatedTimestamp && post.lastUpdatedTimestamp
!== post.publishTimestamp"
              class="d-flex flex-grow-1 justify-start align-center text-
subtitle-2"
              style="font-weight: bold"
            >
              {{ $t("updatedColon") }}
              {{ formatTimestamp(post.lastUpdatedTimestamp) }}
            </div>
            <div class="d-flex flex-grow-1 justify-start align-center text-
subtitle-2">
              {{ $t("publishedColon") }}
              {{ formatTimestamp(post.publishTimestamp) }}
            </div>
            <div class="d-flex flex-grow-1 justify-start align-center text-
subtitle-2">
              {{ post.importerDesc }}
```

```
            </div>
          </div>
          <v-chip-group>
            <v-chip :size="buttonSize">
              {{ formatStatus(post.postReadStatus) }}
            </v-chip>
          </v-chip-group>
        </div>
      </div>
    </v-card-title>
    <!-- divider -->
    <v-divider v-if="showFullPost" />
    <!-- body -->
    <v-card-text v-if="showFullPost">
      <!-- description -->
      <div v-if="post.postDesc" class="overflow-auto">
        <v-label>{{ $t("description") }} ({{ post.postDesc.type }})</v-label>
        <v-divider class="my-1" />
        <div v-if="isHtmlContent(post.postDesc)" v-dompurify-
html="post.postDesc.value" class="post-html-frame" frameborder="0" />
        <div v-else class="post-text-frame">
          {{ post.postDesc.value }}
        </div>
      </div>
      <!-- contents -->
      <div v-if="post.postContents" class="overflow-auto">
        <div v-for="(c, idx) in post.postContents" :key="c">
          <v-label>
            {{ $t("contentsNofM", { n: idx + 1, m: post.postContents.length
}) }} ({{ c.type }})
          </v-label>
          <v-divider class="my-1" />
          <div v-if="isHtmlContent(c)" v-dompurify-html="c.value"
class="post-html-frame" />
          <div v-else class="post-text-frame">
            {{ c.value }}
          </div>
        </div>
      </div>
      <!-- divider -->
      <v-divider v-if="post.postMedia" />
      <!-- post media -->
      <PostMedia v-if="post.postMedia" ref="postMedia" class="ma-2"
:media="post.postMedia" @playEnclosure="$emit('playEnclosure', $event)" />
      <!-- divider -->
      <v-divider v-if="post.postITunes" />
      <!-- post itunes -->
      <PostITunes v-if="post.postITunes" class="ma-2" :i-
tunes="post.postITunes" />
      <!-- post enclosures -->
      <v-sheet v-if="post.enclosures">
        <PostEnclosure
          v-for="enclosure in post.enclosures"
          :key="enclosure"
          class="ma-2"
          :enclosure="enclosure"
          @playEnclosure="$emit('playEnclosure', $event)"
        />
      </v-sheet>
      <!-- post urls, i.e., 'other links' (hidden w/no details) -->
      <div v-if="post.postUrls">
        <v-label>{{ $t("links") }}</v-label>
        <div v-for="postUrl of post.postUrls" :key="postUrl" class="post-
other-link">
```

```
          <a :class="post.isRead ? 'subdued-link' : 'link'"
:href="postUrl.href" target="_blank">
            <v-icon icon="fa-link" />
          </a>
          {{
            (postUrl.title ? postUrl.title : postUrl.type) +
            (postUrl.rel ? " (" + postUrl.rel + ")" : "") +
            (postUrl.hreflang ? " (" + postUrl.hreflang + ")" : "")
          }}
        </div>
      </div>
      <!-- post comment (hidden w/no details) -->
      <div v-if="post.postComment">
        <v-label>{{ $t("postComments") }}</v-label>
        <div>{{ post.postComment }}</div>
      </div>
      <!-- post authors -->
      <div v-if="post.authors">
        <v-label>{{ post.authors.length > 1 ? $t("authors") : $t("author")
}}</v-label>
        <div v-for="author of post.authors" :key="author">
          {{
            (author.name ? author.name : author.email) +
            (author.email ? " (" + author.email + ")" : "")
          }}
        </div>
      </div>
      <!-- post contributors -->
      <div v-if="post.contributors">
        <v-label>{{ $t("contributors") }}</v-label>
        <div v-for="contributor of post.contributors" :key="contributor">
          {{
            (contributor.name ? contributor.name : contributor.email) +
            (contributor.name ? " (" + contributor.email + ")" : "")
          }}
        </div>
      </div>
      <!-- post rights (hidden w/no details) -->
      <div v-if="post.postRights">
        <div>{{ post.postRights }}</div>
      </div>
    </v-card-text>
    <!-- divider -->
    <v-divider />
    <!-- actions -->
    <v-card-actions class="d-flex flex-wrap" style="justify-content: start">
      <slot name="additional" />
      <!-- toggle read status button -->
      <v-btn
        :size="buttonSize"
        :title="post.isRead ? $t('markPostAsUnread') : $t('markPostAsRead')"
        :aria-label="$t('markPostAsUnread')"
        :prepend-icon="post.isRead ? 'fa-eye' : 'fa-check-square-o'"
        @click.stop="togglePostReadStatus"
      />
      <!-- toggle read-later status button -->
      <v-btn
        :size="buttonSize"
        :title="post.isReadLater ? $t('unmarkPostAsReadLater') :
$t('markPostAsReadLater')"
        :aria-label="$t('markPostAsReadLater')"
        :prepend-icon="post.isReadLater ? 'fa-circle-o' : 'fa-bullseye'"
        @click.stop="togglePostReadLaterStatus"
      />
```

```
      <!-- link button -->
      <v-btn
        :size="buttonSize"
        :title="$t('openOriginalArticle')"
        :aria-label="$t('openOriginalArticle')"
        :prepend-icon="'fa-link'"
        @click.stop="$emit('openPostUrl')"
      />
      <!-- toggle sharing options button -->
      <v-btn
        :size="buttonSize"
        :title="$t('showPostSharing')"
        :aria-label="$t('showPostSharing')"
        :prepend-icon="showPostSharing ? 'fa-compress' : 'fa-share-alt'"
        @click.stop="showPostSharing = !showPostSharing"
      />
      <!-- post sharing buttons (the actual sharing options) -->
      <v-btn
        v-for="sharingOption in sharingOptions"
        v-show="showPostSharing"
        :key="sharingOption"
        :size="buttonSize"
        :title="$t('shareWith_' + sharingOption.name)"
        :aria-label="$t('shareWith_' + sharingOption.name + '_ariaLabel')"
        :prepend-icon="'fa-' + sharingOption.icon"
        @click.stop="$emit('share', { sharingOption, post })"
      />
    </v-card-actions>
  </v-card>
</template>

<script>
import { ref } from 'vue';

import { useTimestamp } from '@/composable/useTimestamp.js';
import { usePosts } from '@/composable/usePosts.js';
import { useQueues } from '@/composable/useQueues.js';

import PostEnclosure from "./PostEnclosure.vue";
import PostMedia from "./PostMedia.vue";
import PostITunes from "./PostITunes.vue";
import buttonSizeMixin from '@/mixins/buttonSizeMixin';
import spacingMixin from '@/mixins/spacingMixin';

export default {
  name: "PostCard",
  components: {
    PostEnclosure,
    PostMedia,
    PostITunes,
  },
  mixins: [buttonSizeMixin, spacingMixin],
  props: {
    baseUrl: { type: String, required: true },
    post: { type: Object, required: true },
    sharingOptions: { type: Array, default: null },
    collapsible: { type: Boolean, default: true },
    collapsed: { type: Boolean, default: false },
  },
  emits: [
    "playEnclosure",
    "openPostUrl",
    "share",
    "updatePostReadStatus",
```

```
  ],
  setup(props, { emit }) {
    const { formatTimestamp } = useTimestamp();
    const { formatStatus } = usePosts();
    const { queueStore } = useQueues(props);

    const showPostCategories = ref(false);
    const showPostSharing = ref(false);
    const showFullPost = ref(!props.collapsed);
    const showFirstEnclosure = ref(false);

    function isHtmlContent(contentObj) {
      return contentObj != null && contentObj.type != null &&
contentObj.type.toLowerCase().indexOf("html") >= 0;
    }

    function togglePostReadStatus() {
      updatePostReadStatus(props.post.isRead ? "UNREAD" : "READ",
"togglePostReadStatus");
    }

    function togglePostReadLaterStatus() {
      updatePostReadStatus(props.post.isReadLater ? "UNREAD" : "READ_LATER",
"togglePostReadLaterStatus");
    }

    function updatePostReadStatus(newStatus, successSignal) {
      emit('updatePostReadStatus', { id: props.post.id, newStatus,
originator: successSignal });
    }

    return {
      formatTimestamp,
      formatStatus,
      queueStore,
      showPostCategories,
      showPostSharing,
      showFullPost,
      showFirstEnclosure,
      isHtmlContent,
      togglePostReadStatus,
      togglePostReadLaterStatus,
    };
  },
};
</script>

<style scoped>
.post-thumbnail {
  background-color: transparent;
}

.post-read {
  mix-blend-mode: luminosity;
}
</style>
```

Файл FeedDiscoveryController.java

Реалізація додавання нового джерела новин

```
package main_app;

import main_app.audit.AppLogService;
import main_app.discovery.FeedDiscoveryService;
import main_app.model.error.UpstreamErrorDetails;
```

```
import main_app.model.request.FeedDiscoveryRequest;
import main_app.proxy.ProxyService;
import main_app.resolution.FeedResolutionService;
import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.time.StopWatch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import java.util.Date;

import static org.apache.commons.lang3.StringUtils.isNotBlank;
import static org.springframework.http.ResponseEntity.badRequest;
import static org.springframework.http.ResponseEntity.ok;

@RestController
@Validated
public class FeedDiscoveryController {

    @Autowired
    AppLogService appLogService;

    @Autowired
    FeedDiscoveryService feedDiscoveryService;

    @Autowired
    FeedResolutionService feedResolutionService;

    @Autowired
    ProxyService proxyService;

    /**
     * Perform feed discovery on behalf of a user.  The feed discovery
request consists of a URL, username, and password.
     * This method invokes the feed resolution and discovery services to
gather information about the feed at the given URL.
     * The supplied username and password are provided if the remote system
requests authentication.
     *
     */
    @PostMapping("/discovery")
    @Secured({UserRoles.UNVERIFIED_ROLE})
    public ResponseEntity<?> discoverFeed(@Valid @RequestBody
FeedDiscoveryRequest feedDiscoveryRequest, Authentication authentication) {
        UserDetails userDetails = (UserDetails) authentication.getDetails();
        String username = userDetails.getUsername();
        log.info("discoverFeed for user={}, feedDiscoveryRequest={}",
username, feedDiscoveryRequest);
        StopWatch stopWatch = StopWatch.createStarted();
        String discoveryUrl = feedDiscoveryRequest.getUrl();
        String discoveryUsername = feedDiscoveryRequest.getUsername();
        String discoveryPassword = feedDiscoveryRequest.getPassword();
        try {
            // perform discovery
            FeedDiscoveryInfo feedDiscoveryInfo =
feedDiscoveryService.performDiscovery(discoveryUrl, discoveryUsername,
discoveryPassword);
            // assemble the response
```

```
            ThumbnailedFeedDiscovery thumbnailedFeedDiscoveryResponse =
addThumbnailToResponse(feedDiscoveryInfo);
            stopWatch.stop();
            appLogService.logFeedDiscovery(username, stopWatch,
discoveryUrl);
            return ok(thumbnailedFeedDiscoveryResponse);
        } catch (FeedDiscoveryException e) {
            if (e.exceptionType == PARSING_FEED_EXCEPTION) {
                try {
                    String resolvedUrl =
feedResolutionService.performResolution(discoveryUrl);
                    if (isNotBlank(resolvedUrl) &&
!StringUtils.equals(resolvedUrl, discoveryUrl)) {
                        FeedDiscoveryInfo feedDiscoveryInfo =
feedDiscoveryService.performDiscovery(resolvedUrl, discoveryUsername,
discoveryPassword);
                        ThumbnailedFeedDiscovery
thumbnailedFeedDiscoveryResponse = addThumbnailToResponse(feedDiscoveryInfo);
                        stopWatch.stop();
                        appLogService.logFeedDiscovery(username, stopWatch,
resolvedUrl);
                        // (the resolved path is different from the original
path, and has been discovered successfully)
                        return ok(thumbnailedFeedDiscoveryResponse);
                    }
                    // (the resolved path is the same as the original path,
which has error-ed out)
                    return badRequest().body(
                            new UpstreamErrorDetails(new Date(),
getFeedDiscoveryExceptionTypeMessage(e.exceptionType), e.httpStatusCode,
e.httpStatusMessage, e.redirectUrl, e.redirectHttpStatusCode,
e.redirectHttpStatusMessage));
                } catch (FeedDiscoveryException e1) {
                    // (the resolved path is different from the original
path, but has error-ed out)
                    return badRequest().body(
                            new UpstreamErrorDetails(new Date(),
getFeedDiscoveryExceptionTypeMessage(e1.exceptionType), e1.httpStatusCode,
e1.httpStatusMessage, e1.redirectUrl, e1.redirectHttpStatusCode,
e1.redirectHttpStatusMessage));
                }
            }
            // (the original path error-ed out due to a reason other than
feed resolution)
            return badRequest().body(
                    new UpstreamErrorDetails(new Date(),
getFeedDiscoveryExceptionTypeMessage(e.exceptionType), e.httpStatusCode,
e.httpStatusMessage, e.redirectUrl, e.redirectHttpStatusCode,
e.redirectHttpStatusMessage));
        }
    }

    private static String
getFeedDiscoveryExceptionTypeMessage(FeedDiscoveryExceptionType
exceptionType) {
        return switch (exceptionType) {
            case FILE_NOT_FOUND_EXCEPTION -> "We weren't able to locate a
feed at the URL you provided.";
            case SSL_HANDSHAKE_EXCEPTION -> "We're unable to reach this URL
due to a problem with the remote SSL.  Try another protocol, or resolve the
issue on the remote system.";
            case UNKNOWN_HOST_EXCEPTION -> "We're unable to resolve the
hostname in the URL you provided.";
            case SOCKET_TIMEOUT_EXCEPTION -> "The remote system seems to have
```

```
timed out; you might want to try to discover this feed later.";
            case SOCKET_EXCEPTION -> "We encountered a problem reading
network data from the URL you provided.";
            case CONNECT_EXCEPTION -> "We were unable to connect to the
remote system at the URL you provided.";
            case PARSING_FEED_EXCEPTION -> "The feed at the URL you provided
has syntax issues that prevent us being able to read it properly.";
            case ILLEGAL_ARGUMENT_EXCEPTION -> "Sorry, we're not able to read
the feed at the URL you provided.";
            case IO_EXCEPTION, OTHER -> "Something horrible happened while
reading the feed at the URL you provided.";
            case UNSECURE_REDIRECT -> "This feed is unsecure, and has been
redirected so we've opted not to follow it.";
            case TOO_MANY_REDIRECTS -> "This feed is being redirected too
many times.";
            case HTTP_CLIENT_ERROR, HTTP_SERVER_ERROR -> "We encountered an
HTTP error fetching the feed at the URL you provided.";
        };
    }

    // utility methods

    public ThumbnailedFeedDiscovery addThumbnailToResponse(FeedDiscoveryInfo
feedDiscoveryInfo) {
        //
        FeedDiscoveryImageInfo imageInfo = feedDiscoveryInfo.getImage();
        proxyService.secureFeedDiscoveryImageInfo(imageInfo);
        //
        FeedDiscoveryImageInfo iconInfo = feedDiscoveryInfo.getIcon();
        proxyService.secureFeedDiscoveryImageInfo(iconInfo);
        //
        List<StagingPost> sampleEntries =
feedDiscoveryInfo.getSampleEntries();
        proxyService.secureStagingPosts(sampleEntries);
        //
        return ThumbnailedFeedDiscovery.from(
                feedDiscoveryInfo,
                imageInfo,
                iconInfo,
                null // TODO: remove this
        );
    }
}
```

Файл QueueFilter.vue

Реалізація функціональної вимоги фільтрування постів

```
<template>
  <div>
    <!-- queue filter input -->
    <v-text-field
      id="queue-filter"
      v-model="queueStore.articleListFilter"
      :placeholder="$t('filter')"
      :aria-label="$t('filter')"
      variant="underlined"
      bg-color="transparent"
      @input="$event => queueStore.setArticleListFilter($event.target.value)"
      @click:append="showFilterHelp = !showFilterHelp"
    >
      <template #append>
        <!-- help buton -->
        <v-btn
          :size="buttonSize"
          :title="$t('showFilterHelp')"
          :aria-label="$t('showFilterHelp')"
```

```
          icon="fa-question-circle"
          variant="plain"
          @click="showFilterHelp = !showFilterHelp"
        />
      </template>
    </v-text-field>
    <!-- filter help card -->
    <v-dialog
      v-model="showFilterHelp"
      fullscreen
      scrollable
    >
      <QueueFilterHelp @dismiss="showFilterHelp = false" />
    </v-dialog>
  </div>
</template>

<script>
import { ref } from 'vue';

import { useQueues } from '@/composable/useQueues';

import QueueFilterHelp from './QueueFilterHelp.vue';
import buttonSizeMixin from '@/mixins/buttonSizeMixin';

export default {
  name: "QueueFilter",
  components: {
    QueueFilterHelp,
  },
  mixins: [buttonSizeMixin],
  props: {
    baseUrl: { type: String, required: true },
  },
  setup(props) {
    const { queueStore } = useQueues(props);

    const showFilterHelp = ref(false);

    return {
      queueStore,
      showFilterHelp,
    }
  },
}
</script>

filteredArticleList: (state) => {
  let results = [];
  if (state.articleList) {
    // if a lunr query expression is specified ..
    // then retrieve the preliminary results from lunrjs
    if (state.articleListFilter) {
      let lunrLambda = () =>
state.articleList.index.search(state.articleListFilter);
      try {
        let lunrResults = lunrLambda.apply();
        if (lunrResults) {
          lunrResults = lunrResults.map((item) => parseInt(item.ref));
          results = state.articleList.values.filter((item) => {
            return lunrResults.includes(item.id);
          });
        }
      } catch (error) {
```

```
        if (error instanceof lunr.QueryParseError) {
          // console.debug("lunrjs search query exception due to: " +
JSON.stringify(error));
        } else {
          console.error(error);
          throw error;
        }
      }
    } else {
      // (otherwise the preliminary results are the entire queue)
      results = state.articleList.values;
    }
    //
    // filter and sort the result
    //
    if (results) {
      results = results.filter((r) => {
        if (!state.showUnreadPosts && !r.isRead && !r.isReadLater) {
          return false;
        }
        if (!state.showReadPosts && r.isRead) {
          return false;
        }
        if (!state.showReadLaterPosts && r.isReadLater) {
          return false;
        }
        return true;
      });
      sortQueue(results, state.articleListSortOrder);
    } else {
      results = [];
    }
  }
  return results;
}
```

Файл ThumbnailService.java

Реалізація сервісу для отримання мініатюр зображень для підписок

```
package main_app.thumbnail;


import jakarta.annotation.PostConstruct;
import org.apache.commons.lang3.time.StopWatch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.List;

import static org.apache.commons.codec.binary.Base64.decodeBase64;
import static org.apache.commons.collections4.CollectionUtils.size;
import static org.apache.commons.lang3.RandomUtils.nextInt;
import static org.apache.commons.lang3.StringUtils.isBlank;

@Service
public class ThumbnailService {

    @Autowired
    ErrorLogService errorLogService;

    @Autowired
    RenderedThumbnailDao renderedThumbnailDao; // for redis interaction
```

```
    public RenderedThumbnail getThumbnail(String transportIdent) throws
DataAccessException {
        if (isBlank(transportIdent)) {
            return null;
        }
        log.debug("Attempting to locate thumbnail at transportIdent={}",
transportIdent);
        RenderedThumbnail thumbnail =
renderedThumbnailDao.findThumbnailByTransportIdent(transportIdent);
        if (thumbnail != null) {
            log.debug("Thumbnail located at transportIdent={}",
transportIdent);
        } else {
            log.debug("Unable to locate thumbnail at transportIdent={}",
transportIdent);
        }

        return thumbnail;
    }

    public RenderedThumbnail refreshThumbnailFromSrc(String transportIdent,
String feedImgSrc) throws DataAccessException {
        byte[] imageBytes = decodeBase64(feedImgSrc);
        if (imageBytes == null) {
            log.error("Failed to decode image with imgSrc={} @
transportIdent={} due to unknown format", feedImgSrc, transportIdent);
        }
        RenderedThumbnail thumbnail = RenderedThumbnail.from(transportIdent,
imageBytes);
        renderedThumbnailDao.putThumbnailAtTransportIdent(transportIdent,
thumbnail);

        return thumbnail;
    }

    //
    //
    //

    @Autowired
    ThumbnailDao thumbnailDao;

    @PostConstruct
    public void postConstruct() {
        try {
            StopWatch stopWatch = StopWatch.createStarted();
            List<String> cache = getThumbnailCache();
            stopWatch.stop();
            log.info("Thumbnail cache initialized: size={}, startTime={},
endTime={}, duration={}",
                    size(cache), stopWatch.getStartTime(),
stopWatch.getStopTime(), stopWatch.getTime());
        } catch (DataAccessException e) {
            errorLogService.logDataAccessException("sys", new Date(), e);
        }
    }

    public String getRandom() throws DataAccessException {
        List<String> cache = getThumbnailCache();
        int randIdx = nextInt(0, cache.size());
        return cache.get(randIdx);
    }

    @Cacheable(value = "thumbnailCache")
```

```
    List<String> getThumbnailCache() throws DataAccessException {
        return thumbnailDao.findAll();
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Вебзастосунок "Персоналізований новинний агрегатор"
Програма та методика тестування
КПІ.ІТ-0421.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Данііл ФІЛІПОВСЬКИЙ

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ 3
2 МЕТА ТЕСТУВАННЯ 4
3 МЕТОДИ ТЕСТУВАННЯ 5
4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ 5

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є вебзастосунок "Персоналізований новинний агрегатор".

Програмне забезпечення розроблене для роботи у веб-браузерах і сумісне з наступними платформами:

– операційні системи: Windows, macOS, Linux;

– веб-браузери: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera.

Клієнтську частину програмного забезпечення протестовано у веббраузері Google Chrome на ПК з операційною системою Windows 11

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження і обробки даних;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– функціональне тестування: цей метод полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній. Ми перевірили основні функції застосунку, такі як реєстрація та авторизація користувачів, додавання та видалення підписок, пошук новин, фільтрація та сортування новин;

– системне тестування: ми перевірили весь застосунок в цілому, включаючи серверну частину, клієнтську частину, базу даних та інтеграцію з зовнішніми сервісами, такими як RSS-канали. Це тестування дозволило оцінити, як різні компоненти взаємодіють один з одним і чи немає проблем у їхній сумісності;

– мануальне тестування: ми провели ручне тестування, де вручну виконувалися тести на основні функції застосунку;

– тестування "чорної скриньки": тестувалися функції застосунку,не заглядаючи в його внутрішню структуру. Це дозволило перевірити, чи правильно обробляються вхідні дані і чи видаються коректні результати;

– тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;

– тестування "сірої скриньки": цей метод був використаний для тестування критичних функцій та алгоритмів, таких як обробка RSS-каналів та пошук новин;

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування (End-to-end, E2E) та написанням unit-тестів, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування інтерфейсу користувача;
- тестування зручності використання;
- тестування навантаження.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Вебзастосунок "Персоналізований новинний агрегатор"

Керівництво користувача

КПІ.ІТ-0421.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Данііл ФІЛІПОВСЬКИЙ

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ 3
2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ..... 3
2.1 Системні вимоги для коректної роботи..... 4
2.2 Завантаження застосунку..... 4
2.3 Перевірка коректної роботи..... 4
3 ВИКОНАННЯ ПРОГРАМИ 4

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Веб-застосунок "Персоналізований новинний агрегатор" призначений для збирання, фільтрації та надання користувачам новинного контенту з різних джерел у зручному та налаштованому форматі. Він дозволяє користувачам створювати облікові записи, підписуватися на різні новинні ресурси, організовувати підписки за категоріями, шукати та фільтрувати новини, а також взаємодіяти з новинними постами (наприклад, відмічати як прочитані або ділитися ними).

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних мінімальних вимог:

– ПК зі встановленим браузером Google Chrome, Mozilla Firefox, Microsoft Edge, або Opera останніх версій;

– об’єм оперативної пам'яті (ОЗП): 4 Гб;

– наявність доступу до Інтернету;

– процесор сімейства Intel Core i3.

Для оптимальної роботи даного застосунку також необхідне виконання наступних рекомендованих вимог:

– об’єм оперативної пам'яті (ОЗП): 16 Гб;

– процесор сімейства Intel Core i5;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

2.2 Завантаження застосунку

Застосунок розгортається локально за допомогою Docker-контейнерів. Для цього використовується Docker Compose файл, що містить конфігурацію необхідних сервісів

2.3 Перевірка коректної роботи

Після перевірки коректної роботи застосунку необхідно виконати наступні дії:

– після запуску контейнерів, відкрити веб-браузер і перейти за URL-адресою: http://localhost:3000;

– переконатися, що завантажується сторінка привітання;

– виконати реєстрацію нового користувача або авторизацію існуючого користувача;

– перевірити доступність основних функцій: додавання та управління підписками, пошук та фільтрація новин, взаємодія з новинними постами;

– відсутність повідомлень про помилку свідчить про коректну роботу програми.

3 ВИКОНАННЯ ПРОГРАМИ

При переході за URL-адресою додатка у веб-браузері користувачу буде відображено головну сторінку (рисунок 3.1).

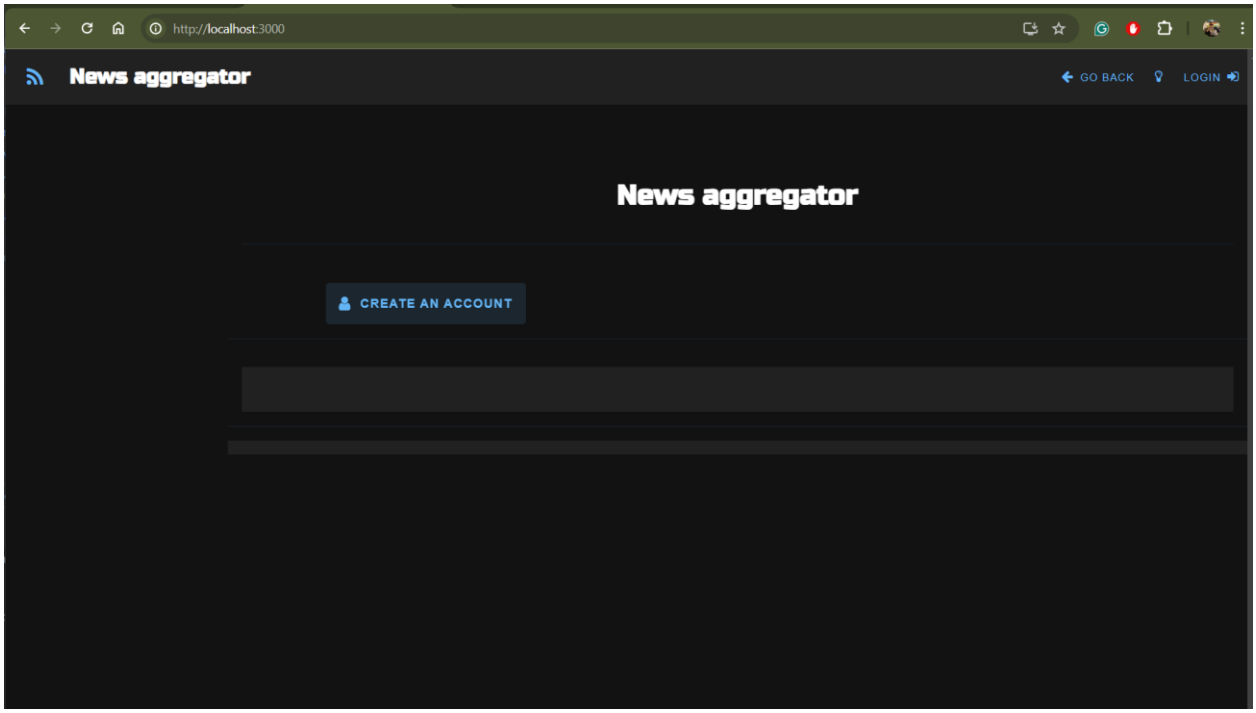


Рисунок 3.1 – Початкова сторінка додатку

Користувач має змогу за допомогою навігаційної панелі нагорі сторінки авторизуватись у додаток. Якщо користувач ще не має акаунту то він може створити новий акаунт натиснувши на кнопку “Create an account”. При натисканні на кнопку користувач перейде на сторінку реєстрації де потрібно ввести дані для нового акаунту(рисунок 3.2)

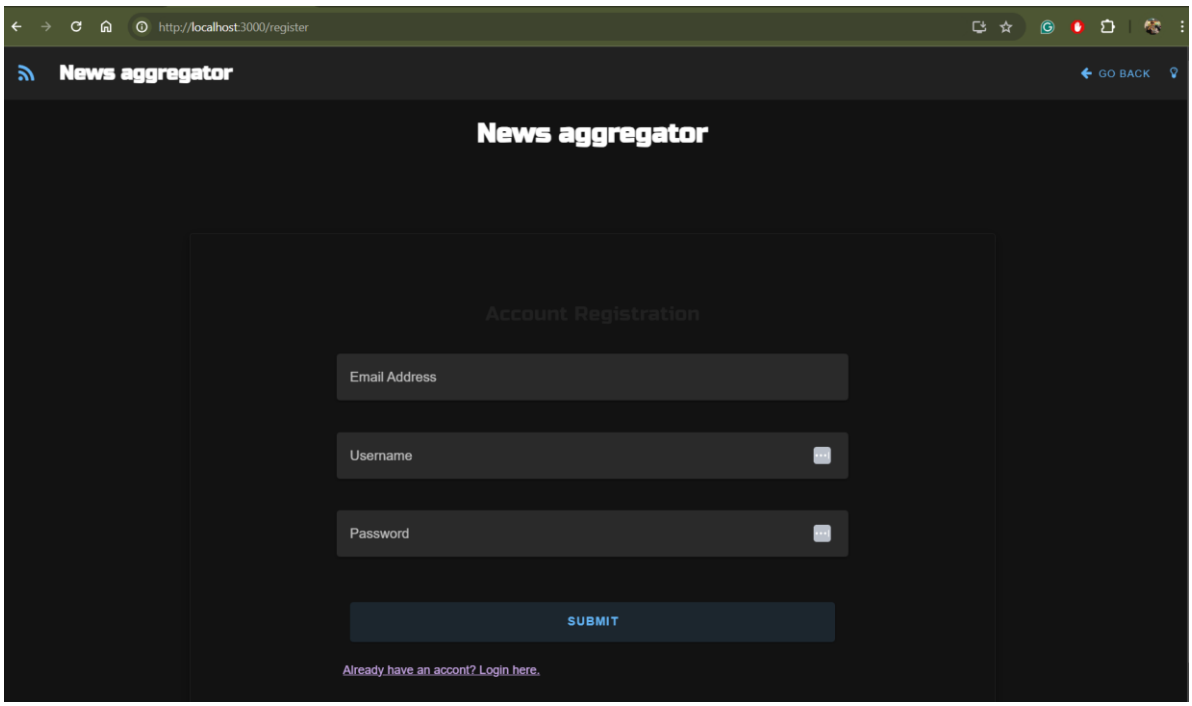


Рисунок 3.2 – Сторінка реєстрації

Після успішного реєстрації користувач переходить на сторінку авторизації(рисунок 3.3).

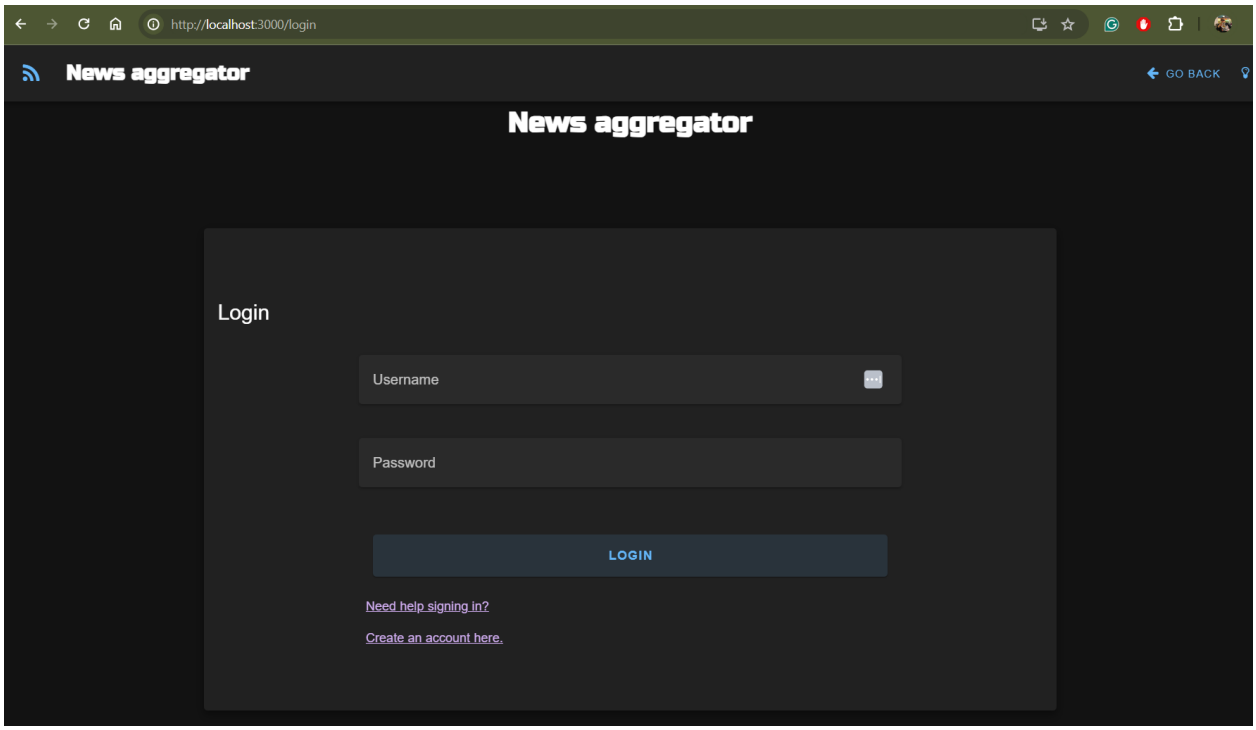


Рисунок 3.3 – Сторінка авторизації

Після успішної авторизації користувач переходить на головну сторінку додатку (рисунок3.4).

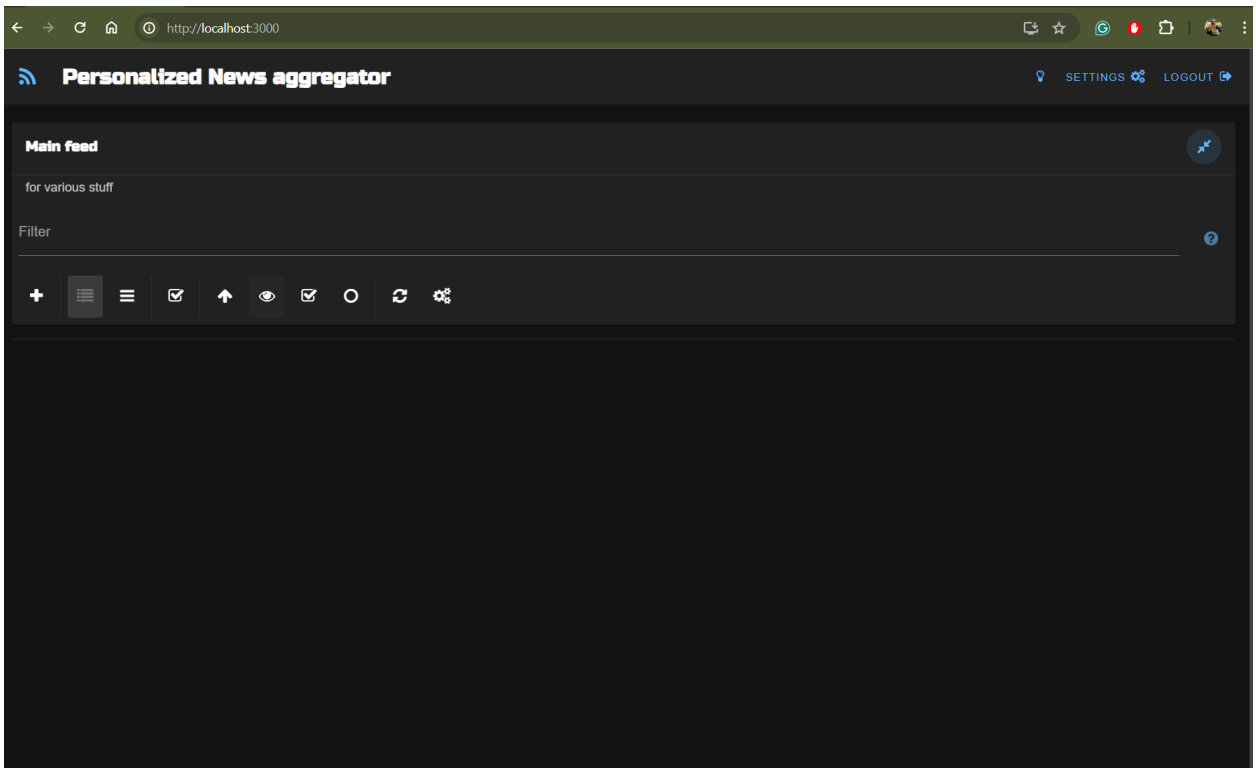


Рисунок 3.4 – Головна сторінка новин

Перед тим як додати нові підписки на новини нам треба додати нові категорії новин, для цього в інтерфейсі треба натиснути на кнопку плюс, після цього відкривається діалогове вікно для створення нової категорії (рисунок 3.5).

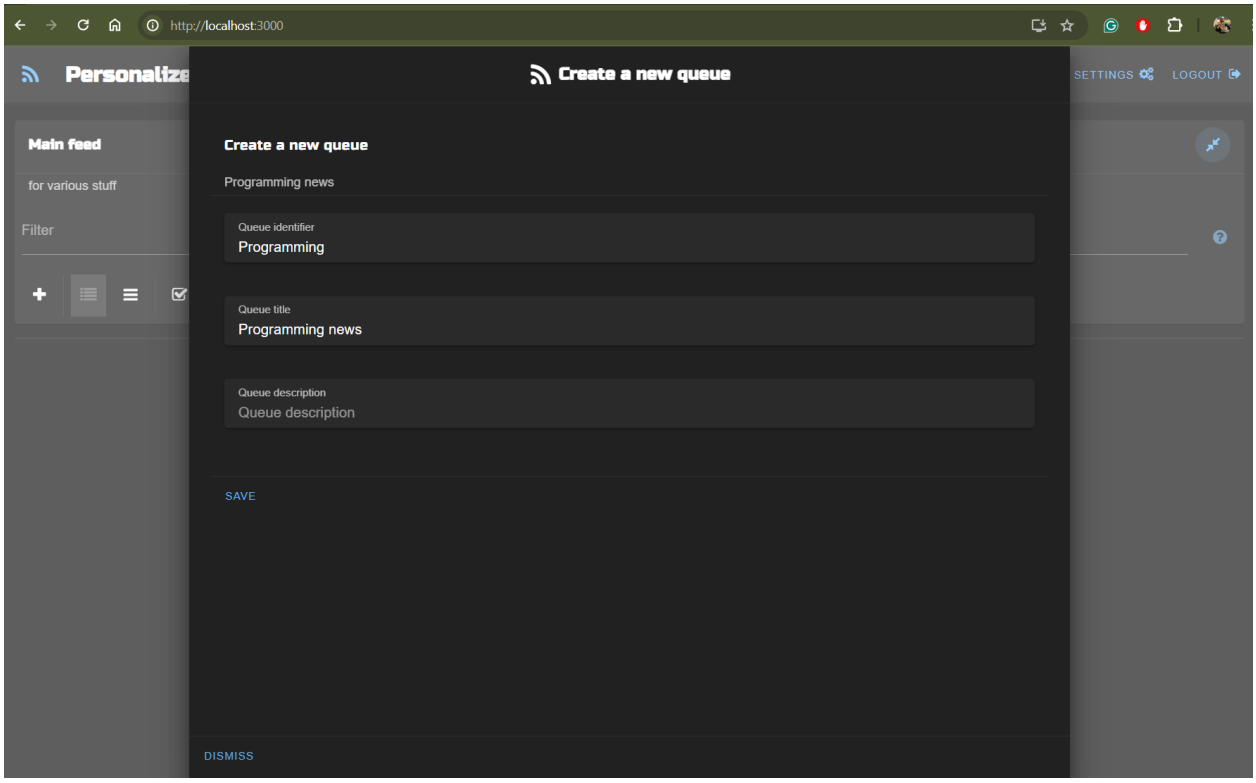


Рисунок 3.5 – Діалогове вікно створення категорії

Після створення нової категорії її можна побачити у списку всіх категорій натиснувши на кнопку згори зліва в інтерфейсі (рисунок 3.6).

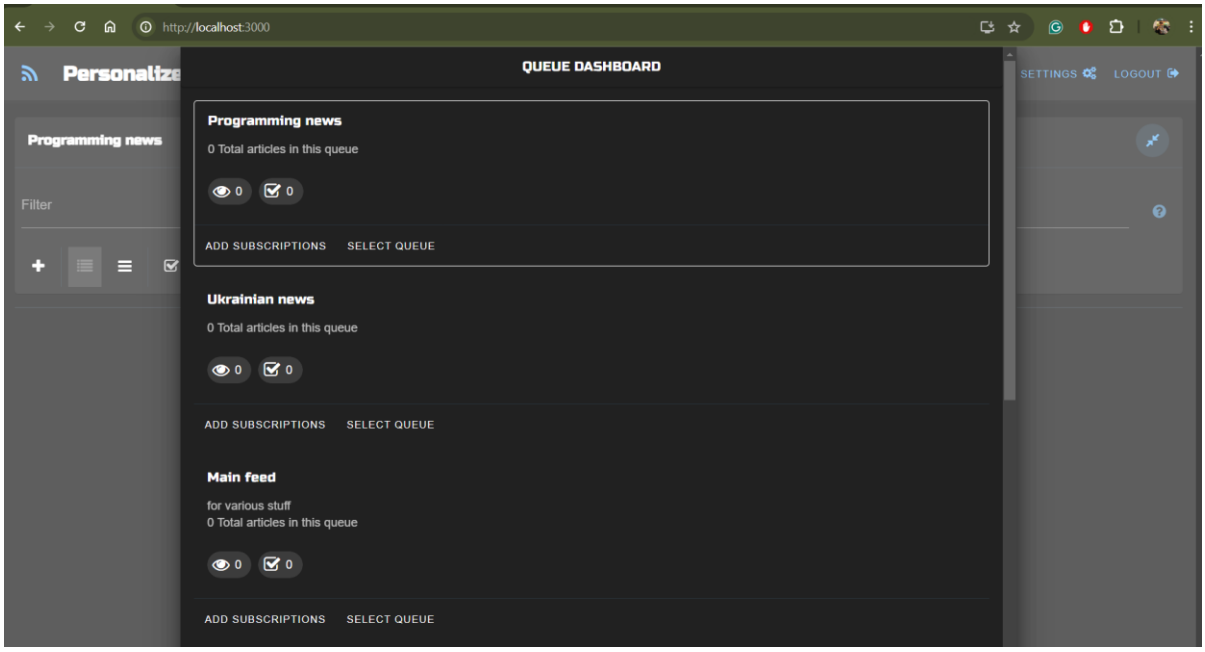


Рисунок 3.6 – Список існуючих категорій

Після цього можна натиснути натиснути на кнопку “Add subscriptions” щоби перейти до створення нової підписки на новину у цій категорії (рисунок 3.7).

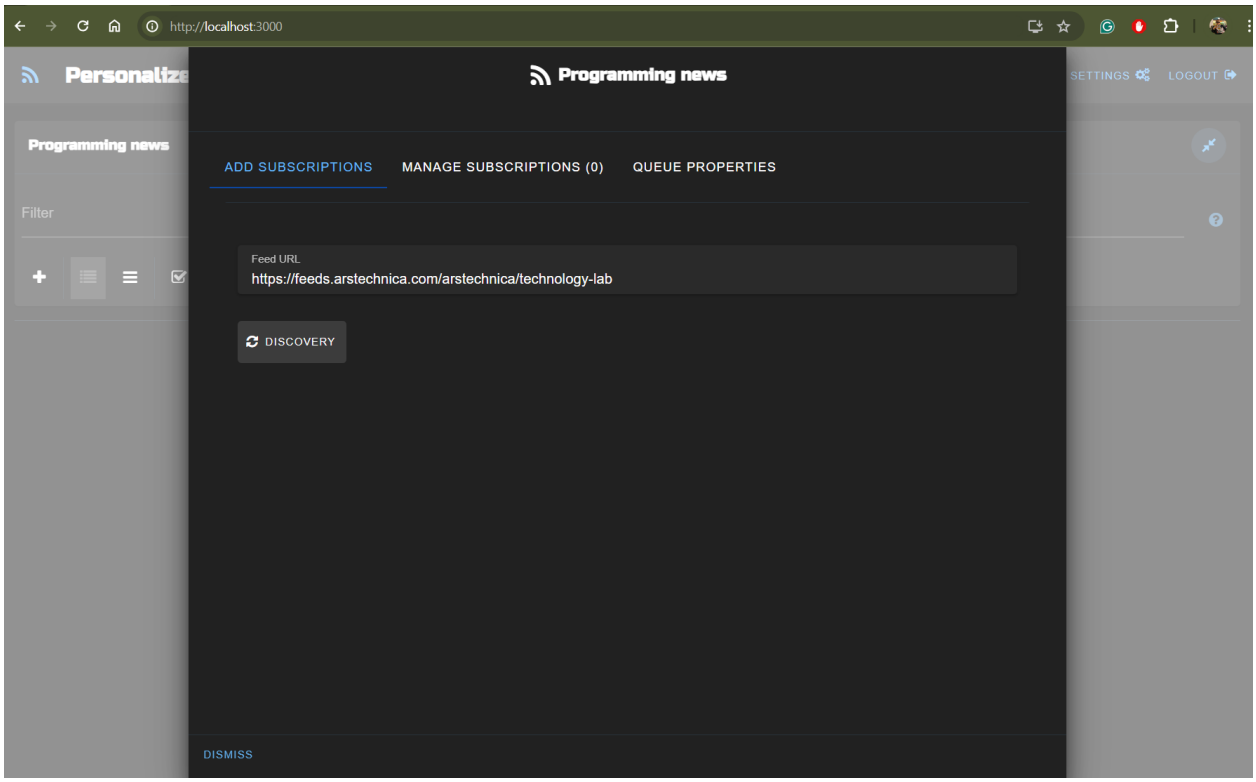


Рисунок 3.6 – Діалогове вікно створення нової підписки у категорії

Перед тим як додати підписку до категорії користувач натискає на кнопку “Discovery” після чого сервер робить запит до першоджерела щоби

перевірити валідність посилання та перевірити чи є підтримка передачі даних по RSS, і тільки після цього користувач може зробити нову підписку (рисунок 3.7).

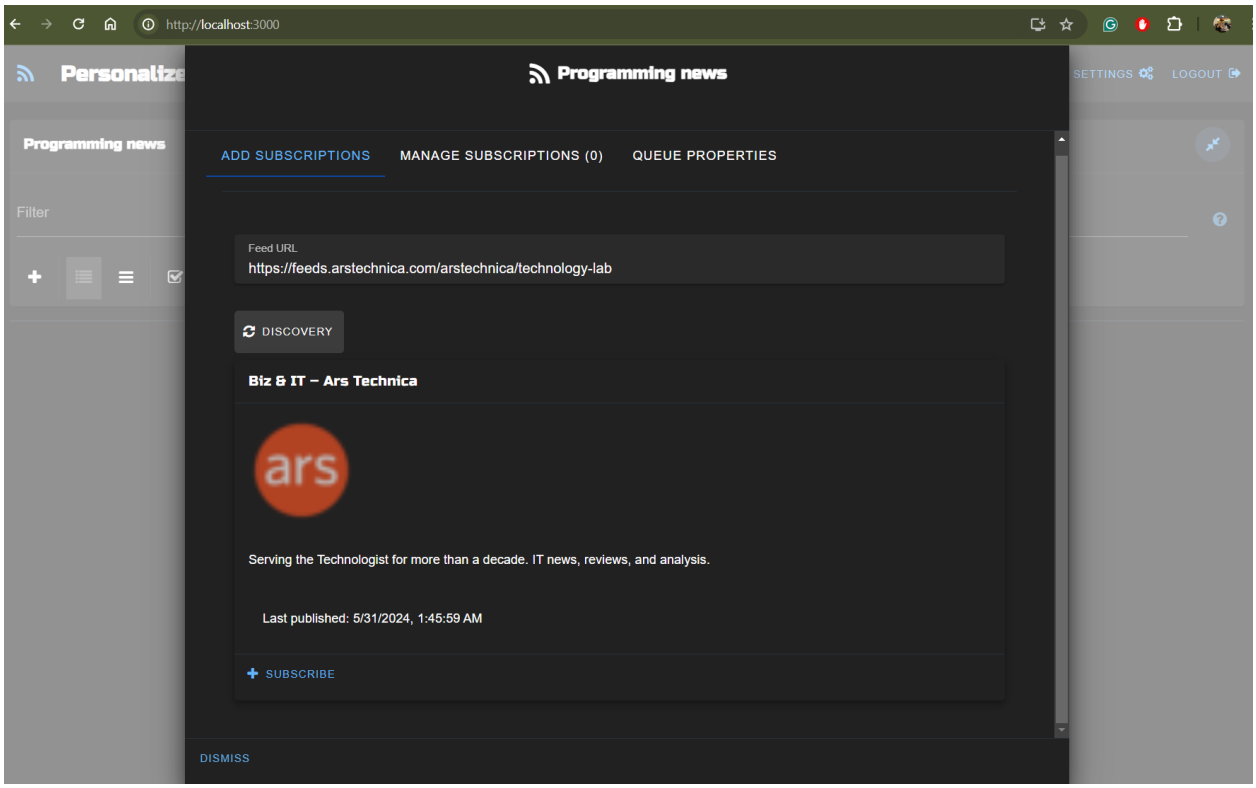


Рисунок 3.7 – Діалогове вікно створення нової підписки у категорії

Після цього відбувається оновлення головної сторінки новин і тепер користувач бачить нові новини з першоджерела (рисунок 3.8)

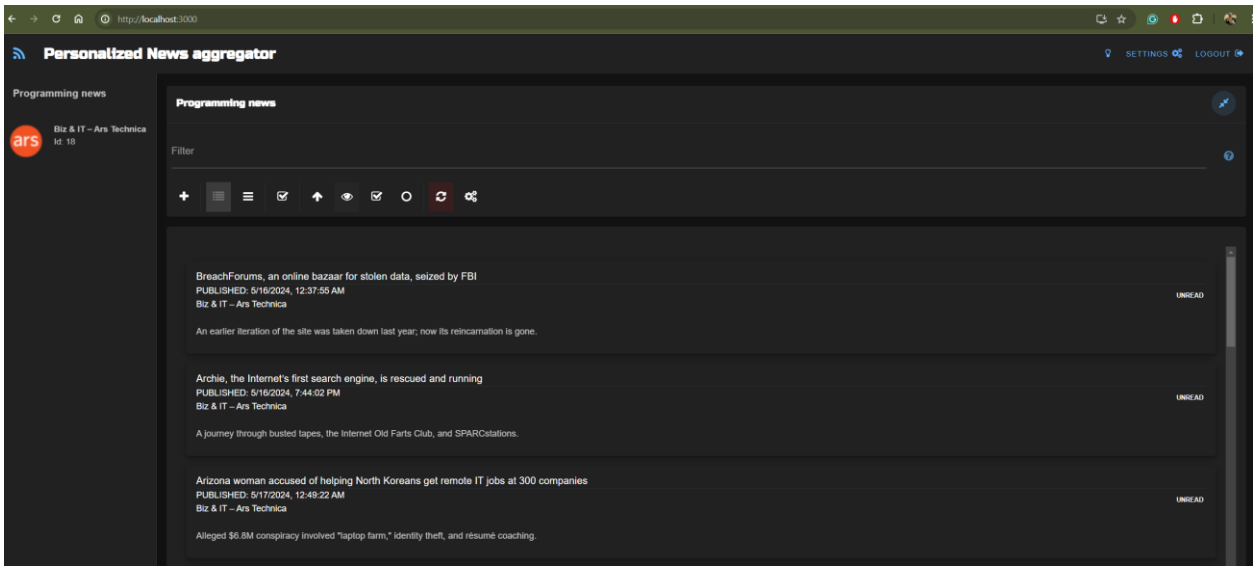


Рисунок 3.8 – Оновлена головна сторінка новин

При натисканні на новини відкривається діалогове вікно для детального читання новини включаючи зображення, текст, посилання в тексті,

посилання на коментарі до новини у першоджерелі, та інші метадані. Можливо переходити від читання одної новини до іншої за допомогою стрілочок, можна відмічати новини як прочитані або прочитати потім, можна перейти до першоджерела або отримати поділитися посиланням у соцмережах (рисунок 3.9).

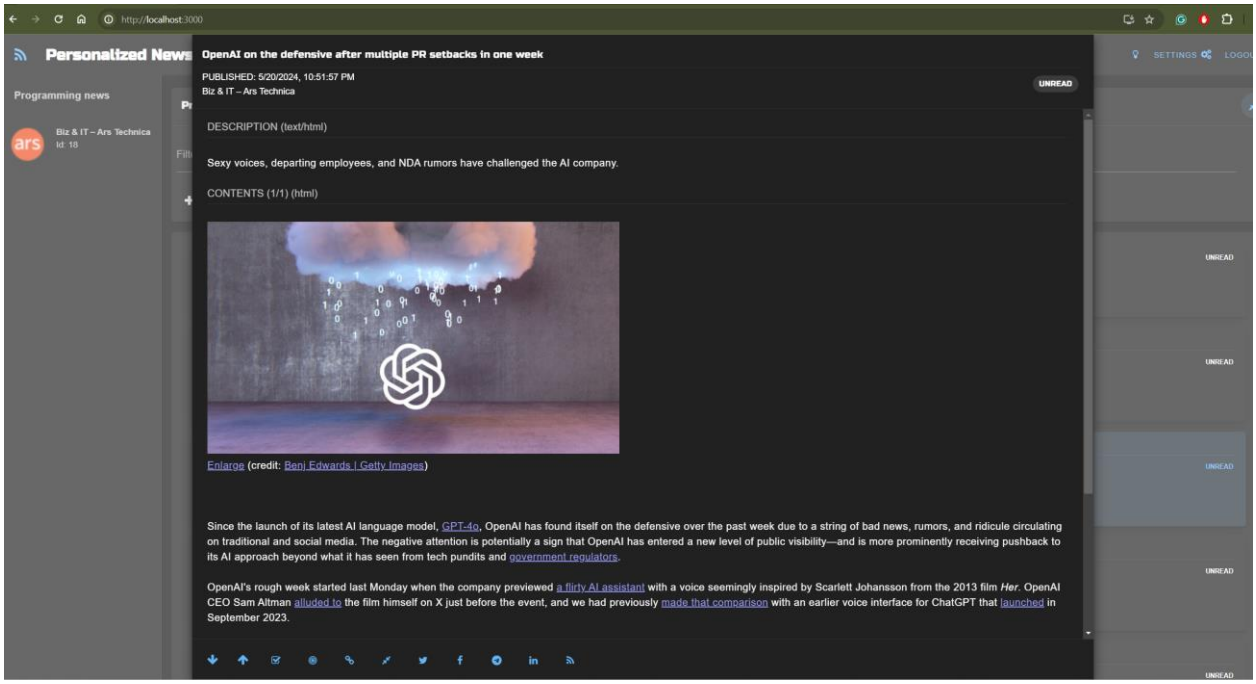


Рисунок 3.9 – Діалогове вікно читання новин

При відмічанні новин як прочитані або прочитати потім вони зникають з списку новин допоки користувач не буде фільтрувати прочитані новини (рисунок 3.10) або прочитати потім (рисунок 3.11).

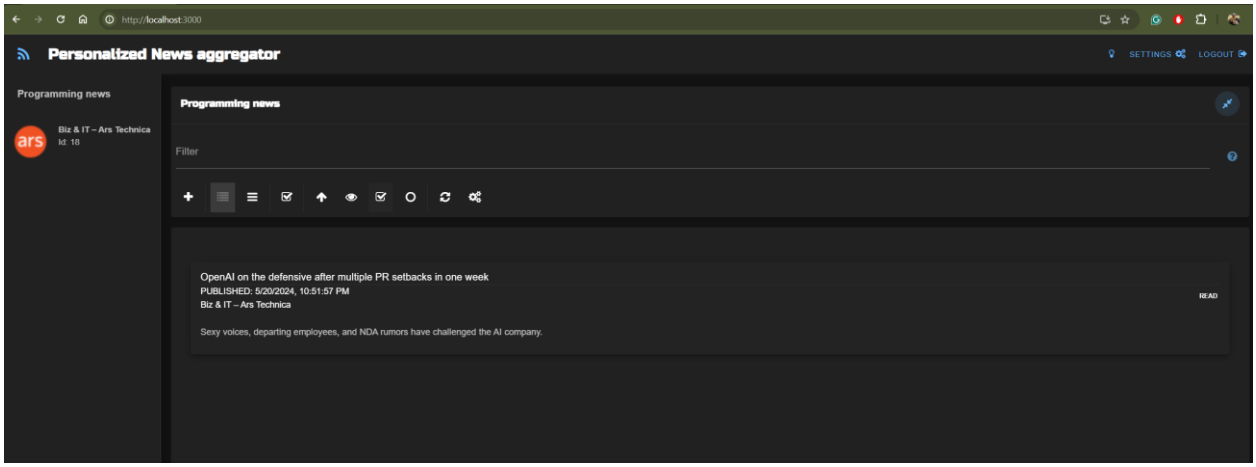


Рисунок 3.10 – Фільтрування прочитаних новин

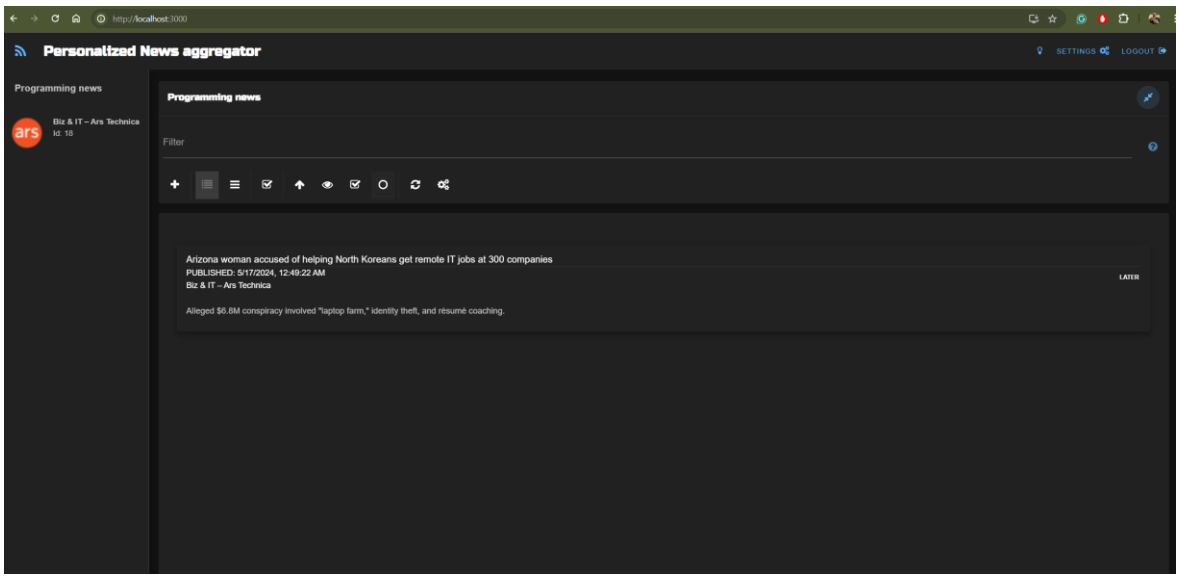


Рисунок 3.11 – Фільтрування новин на прочитати потім

Для пошуку новин серед декількох підписок у категорії можна використовувати поле фільтрації для пошуку по заданому фільтру (рисунок 3.12).

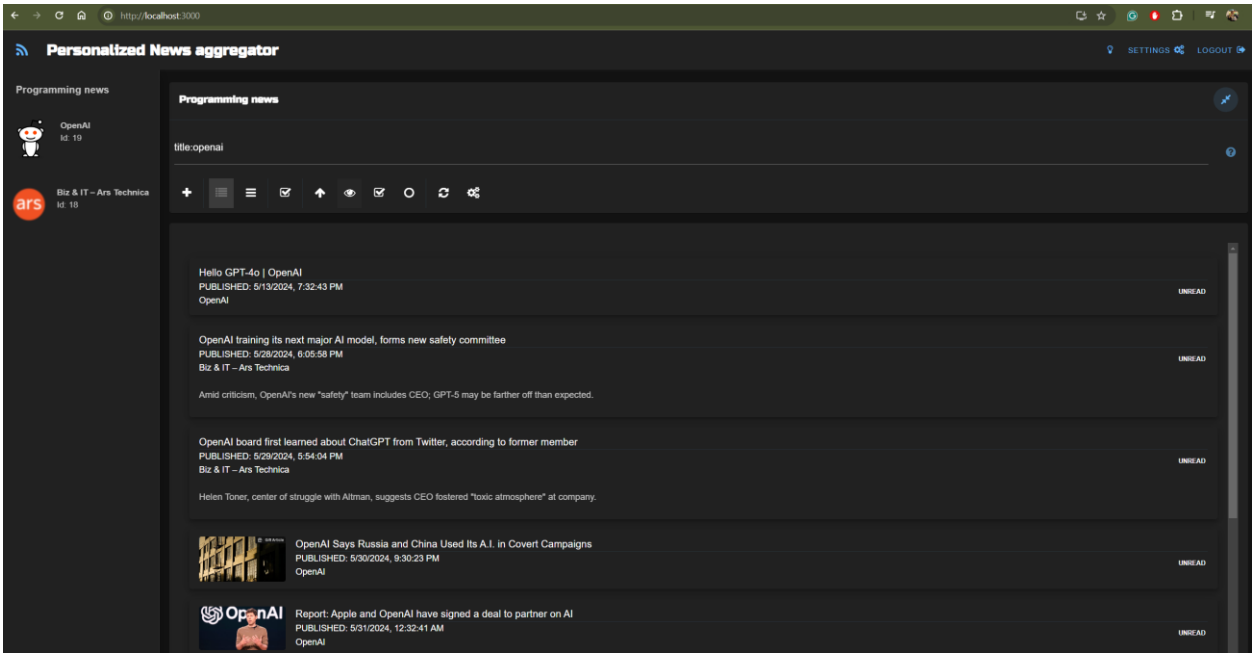


Рисунок 3.12 – Пошук новин за заданим запитом

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Вебзастосунок "Персоналізований новинний агрегатор"

Графічний матеріал

КПІ.ІТ-0421.045440.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Катерина ЛІЩУК

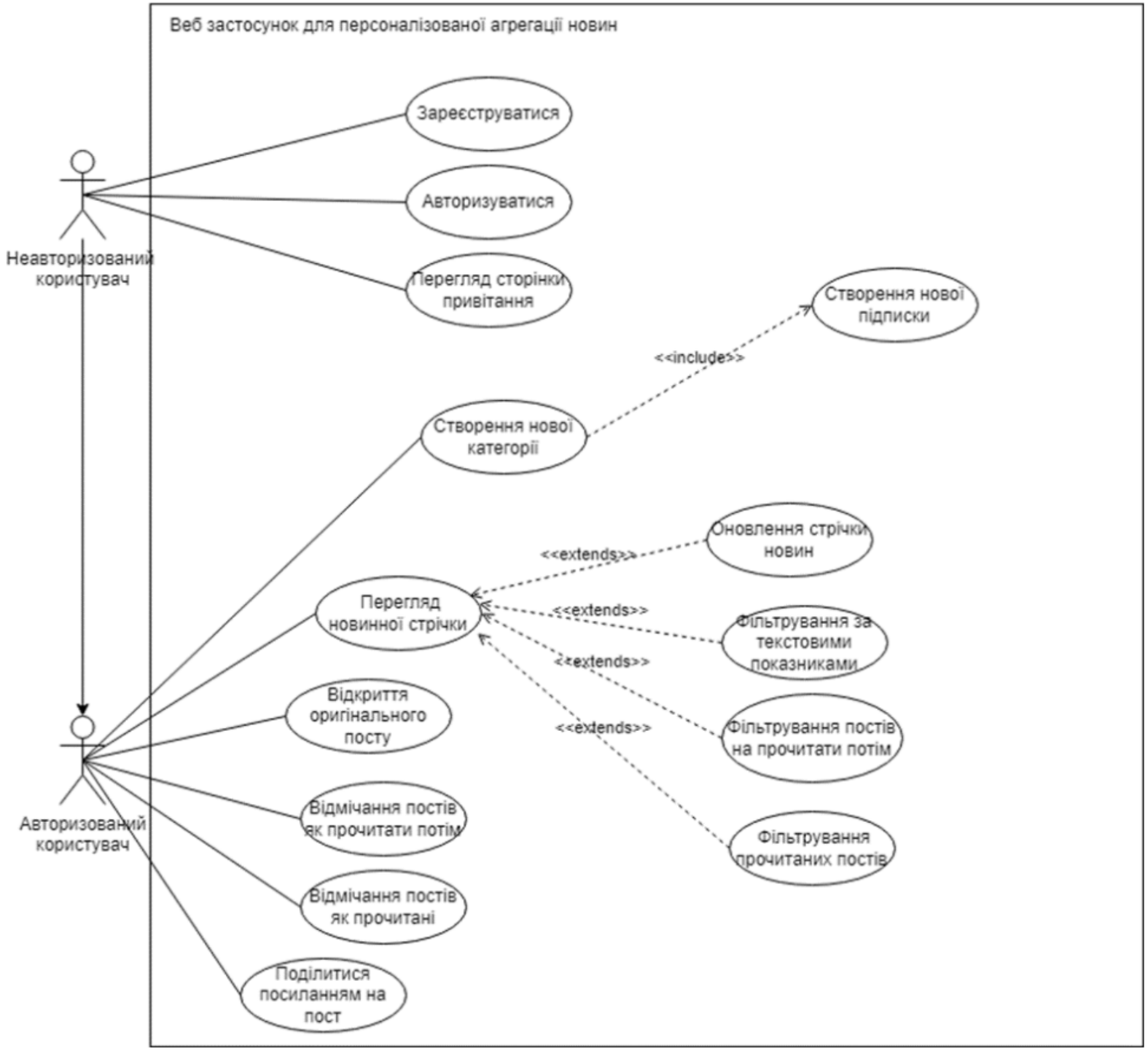
Нормоконтроль:

_____ Павло РОДІОНОВ

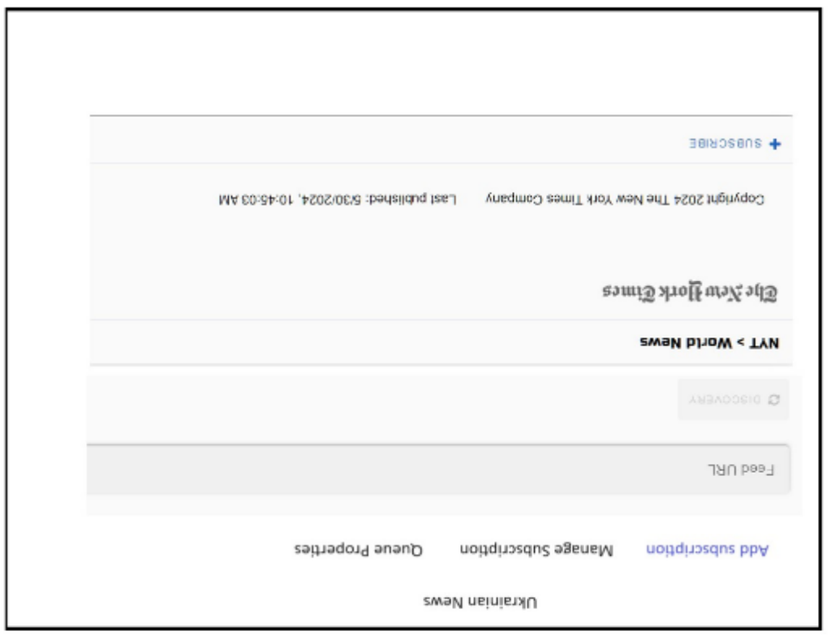
Виконавець:

_____ Даніїл ФІЛІПОВСЬКИЙ

Київ – 2024



					КПІ.ІТ-0421.045440.06.99 ССВ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Філіповський Д.Є			Схема структурна варіантів використань
Перевірив		Ліщук К.І.			Лит. / Маса / Масштаб
Т. контр.					Аркуш / Аркушів
					Вебзастосунок Персоналізований новинний агрегатор
Н. контр.		Родіонов П.Ю			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04
Затвердив		Жаріков Е.В.			



Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-0421.045440.06.99 КЕ		
Розробив		Філіповський Д.Є.					
Перевірив		Лищук К.І.					
Т. контр.							
Н. контр.		Радіонов П.Ю.					
Затвердив		Жаріков Е.В.					
					Кресленик витягу екранних форм	Літ.	Маса	Масштаб
					Відомість Перспективних новинних агрегаторів	КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІТ-04		
						Аркуш	Аркушів	

Зм.	Арк.	№ докум.	Підп.	Дата

Розробив	Філіповський Д.Є.	
Перевірив	Лішук К.І.	
Т. контр.		
Н. контр.	Радіонов П.Ю.	
Затвердив	Жаріков Е.В.	

КПІ.ІТ-0421.045440.06.99 СБД

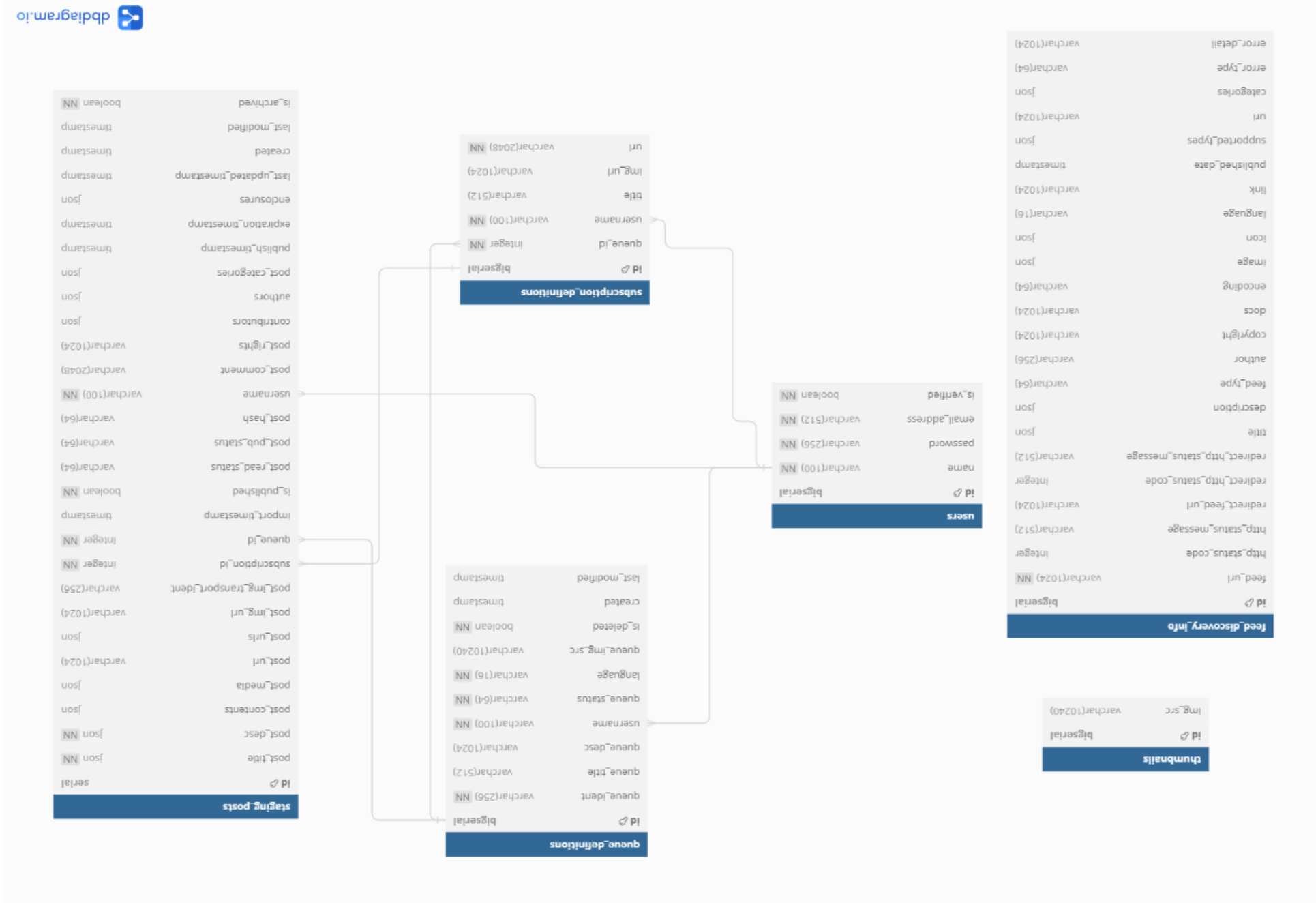
Схема бази даних

Літ.	Маса	Масштаб

Аркуш	Аркушів

Відомість Перспективних новинний агрегатор

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-04



КПІ.ІТ-0421.045440.06.99 СБД