МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБДОДАТКУ ДОШКИ ОГОЛОШЕНЬ З СИСТЕМОЮ КЕРУВАННЯ НА ОСНОВІ SPRING FRAMEWORK ТА HIBERNATE»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.Д. Ткаченко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ткаченку Вадиму Денисовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебдодатку дошки оголошень з системою керування на основі Spring Framework та Hibernate

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження та аналіз предметної області.
2. Проєктування вебзастосунку.
3. Розробка вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.04.2024	
2.	Збір вихідних даних.	24.05.2024	
3.	Обробка методичних та теоретичних	28.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	11.09.2024	
5.	Розробка третього розділу.	04.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ В.Д. Ткаченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебдодатку дошки оголошень з системою керування на основі Spring Framework та Hibernate»: 105 с., 81 рис., 1 табл., 7 джерел, 6 додатків.

БУТСТРАП, ВЕБЗАСТОСУНОК, КАПЧА, СПРІНГ ДАТА, СПРІНГ ФРЕЙМВОРК, ТАЙМЛІФ, ХІБЕРНЕЙТ.

Об’єкт дослідження – вебзастосунки, які призначено для обміну інформацією між користувачами.

Мета роботи: дослідження створення вебдодатків із застосуванням фреймворків Spring Framework та Hibernate, а також взаємодії серверної частини з компонентами фреймворків.

Методи дослідження – методи збору та аналізу вимог до вебзастосунків, методи моделювання, проєктування та конструювання.

У кваліфікаційній роботі проведено аналіз предметної області, огляд існуючих систем для роботи з оголошеннями та блогами, а також описано технології, що використовуються для створення вебзастосунків. На основі отриманих теоретичних відомостей розроблено багатофункціональний вебзастосунок для розміщення, пошуку та взаємодії з оголошеннями та блогами. Результатом виконання роботи став вебзастосунок, створений із використанням Java, Spring Framework, MySQL, Thymeleaf та інших сучасних технологій для забезпечення безпеки, зручності та функціональності.

SUMMARY

Master's qualifying paper "Development of a Bulletin Board Web Application with Management System Based on Spring Framework and Hibernate": 105 pages, 81 figures, 1 table, 7 references, 6 supplements.

BOOTSTRAP, WEB APPLICATION, CAPTCHA, SPRING DATA, SPRING FRAMEWORK, THYMELEAF, HIBERNATE.

The object of the study is web applications designed to exchange information between users.

The aim of the study is creation of web applications using the Spring Framework and Hibernate frameworks, as well as the interaction of the server side with the components of the frameworks.

The methods of research are methods of collecting and analyzing requirements for web applications, methods of modeling, design and construction.

The qualification work analyzes the subject area, reviews existing systems for working with ads and blogs, and describes the technologies used to create web applications. Based on the theoretical information obtained, a multifunctional web application for posting, searching and interacting with ads and blogs was developed. The result of the work is a web application created using Java, Spring Framework, MySQL, Thymeleaf and other modern technologies to ensure security, convenience and functionality.

ЗМІСТ

Завдання на кваліфікаційну роботу……...2
Реферат 4
Summary 5
Вступ..... 8
Розділ 1 Аналіз архітектури сучасних вебзастосунків..... 10
1.1 Огляд мікросервісної та монолітної архітектури 10
1.1.1 Монолітна архітектура 10
1.1.2 Мікросервісна архітектура..... 12
1.1.3 Порівняльний аналіз мікросервісної та монолітної архітекутри 13
1.2 Шаблон MVC..... 15
1.3 Вибір архітектури та шаблону проєктування 18
Розділ 2 Технологічні аспекти реалізації проєкту 20
2.1 Фреймворк Spring як основа для побудови застосунку..... 20
2.2 Використання Thymeleaf для зображення динамічних даних..... 23
2.3 Роль Spring Data JPA та Hibernate у взаємодії з базою даних 25
2.4 Організація даних за допомогою MySQL..... 29
2.5 Застосування Bootstrap під час розробки інтерфейсу користувача 30
2.6 Схема даних та класова модель вебзастосунку 32
Розділ 3 Реалізація програмної системи 37
3.1 Реалізація структури даних..... 37
3.2 Реалізація бізнес-логіки..... 41
3.3 Реалізація конфігураційних класів..... 47
3.4 Реалізація контролерів..... 49
Висновки 60
Перелік посилань..... 61
Додаток А Класи для роботи з даними та сховищем даних 62
Додаток Б Сервісні класи 67

Додаток В Конфігураційні класи 76
Додаток Г Класи контролери 78
Додаток Д Html файли 85
Додаток Е Перечислення 105

ВСТУП

Розробка вебзастосунків стає все більш необхідною для вирішення численних завдань сучасних інформаційних систем. Ця робота спрямована на створення багатофункціонального вебзастосунка із використанням технологій Java, Spring, MySQL та Thymeleaf, а також з інтеграцією інструментів безпеки, таких як Spring Security. Розроблений застосунок призначений для ефективного розміщення, пошуку та взаємодії з оголошеннями та блогами, надаючи користувачам зручний інтерфейс і актуальні інструменти для обміну інформацією.

Актуальність. Створення застосунку зумовлено необхідністю надати користувачам надійний та швидкий інструмент для пошуку та розміщення оголошень про роботу, товари, послуги, а також можливість вести блоги та підписуватись на користувачів, що їх цікавлять. Вебзастосунок надає можливість користувачам розміщувати оголошення, вибирати їх категорії, фільтрувати за різними критеріями, такими як пошук/пропозиція роботи чи продаж/покупка. Додатково, важливою функціональністю є можливість додавати фотографії до оголошень, та блогів, що реалізовано через збереження зображень у базі даних у вигляді масиву байтів.

Новизна. Новизна роботи полягає у розробці вебзастосунку з використанням сучасних технологій та підходів, що дозволяє вирішувати актуальні проблеми користувачів. Особливу увагу приділено інтеграції функціоналу роботи із зображеннями, реалізації підписки з автоматичними сповіщеннями, забезпеченню безпеки та зручного пошуку. Отже, новизна роботи полягає не тільки у використанні сучасних технологій, але й у реалізації специфічних функціональних можливостей.

Практична значимість. Практична значимість роботи полягає у створенні інструменту, який дозволяє користувачам швидко і легко знаходити необхідну інформацію, будь то вакансії, блоги, товари чи послуги.

Можливість підписки на користувачів, що цікавлять, і отримання сповіщень електронною поштою спрощує взаємодію між користувачами. Розроблений функціонал також дозволяє використовувати застосунок як основу для подальших розробок. Окрім цього, розроблений застосунок може служити базою для подальших розробок та вдосконалень, адаптуючись під змінні потреби ринку.

Мета і завдання. Метою роботи є створення вебзастосунку для розміщення та перегляду оголошень, який би відповідав сучасним стандартам зручності та функціональності. Основні завдання включають:

- розробка інтерфейсу користувача для розміщення та перегляду оголошень;
- реалізація системи категоризації та фільтрації оголошень;
- забезпечення можливості додавання фотографій до оголошень;
- інтеграція бази даних для надійного зберігання інформації;
- забезпечення безпеки та аутентифікації користувачів.

Об’єкт і предмет дослідження. Об’єкт дослідження – вебзастосунки, які призначено для обміну інформацією між користувачами та включають різні платформи, сервіси, що дозволяють користувачам розміщувати, переглядати та взаємодіяти з оголошеннями, товарами чи послугами в режимі онлайн.

Предмет дослідження – архітектурні підходи та технології, які дозволяють реалізувати функціональність вебзастосунків, а також забезпечують необхідний рівень безпеки та продуктивності системи.

РОЗДІЛ 1 АНАЛІЗ АРХІТЕКТУРИ СУЧАСНИХ ВЕБЗАСТОСУНКІВ

1.1 Огляд мікросервісної та монолітної архітектури

У сучасній розробці програмного забезпечення вибір архітектурного підходу відіграє ключову роль у визначенні ефективності, масштабованості та стійкості додатків. Серед популярних підходів виокремлюються монолітна та мікросервісна архітектури.

1.1.1 Монолітна архітектура

Монолітна архітектура – це підхід до розробки програмного забезпечення, де весь застосунок будується як єдиний, нероздільний блок [1]. Усі компоненти та функціональності, такі як інтерфейс користувача, бізнес-логіка, доступ до бази даних, обробка запитів і т. ін., інтегровані в один великий кодовий блок (див. рис. 1.1).

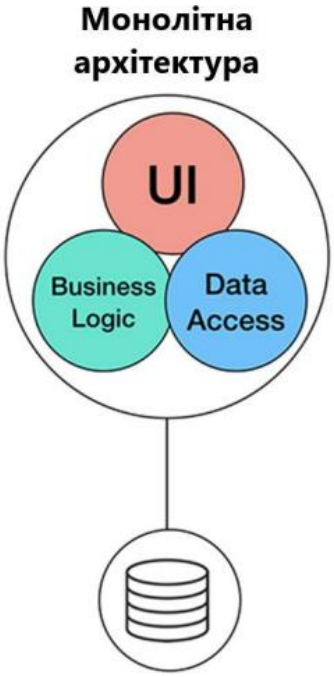


Рисунок 1.1 – Монолітна архітектура

Основна перевага монолітів полягає в їх легкості розробки на ранніх стадіях життєвого циклу розробки програмного забезпечення (SDLC). Оскільки компоненти монолітної системи тісно пов’язані, єдина кодова база спрощує документування, керування конфігурацією та моніторинг продуктивності додатків [1].

Деякі монолітні програми можуть мати кращу пропускну здатність і час відгуку, ніж модульні програми на основі мікросервісів, особливо якщо кількість потоків і користувачів невелика. Їх також може бути легше тестувати та налагоджувати, оскільки вони складаються з меншої кількості елементів і, отже, містять менше змінних і сценаріїв тестування [1].

Ще однією перевагою монолітів є спільне використання даних. Всі компоненти мають прямий доступ до єдиної бази даних, що дозволяє легко обмінюватися інформацією без необхідності в складних механізмах синхронізації.

Серед недоліків можна визначити гнучкість. Оскільки застосунки розвиваються та ускладнюються, стає ще важче правильно впроваджувати зміни без погіршення якості коду та продуктивності програми. Після кожного оновлення розробники повинні скомпілювати всю кодову базу та повторно розгорнути повну програму, а не лише оновлену частину [1].

Масштабування монолітних архітектур є складним завданням, оскільки весь застосунок масштабується як єдине ціле. В результаті, якщо ж додаткові ресурси потрібні лише для певних компонентів, це може призвести до нераціонального використання ресурсів.

Якщо застосунок включає багато функцій, то вся система може почати працювати повільніше, коли для виконання однієї з функцій знадобляться додаткові ресурси.

Отже, монолітна архітектура забезпечує простоту та ефективність, використовуючи єдину базу коду та спільні дані. Однак вона має явні проблеми з масштабованістю, розгортанням та обслуговуванням, а також обмежену технологічну гнучкість.

1.1.2 Мікросервісна архітектура

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, де застосунок складається з набору невеликих, незалежних сервісів, кожен з яких відповідає за конкретну функціональність (див. рис. 1.2). Кожен мікросервіс може бути розроблений, розгорнутий і масштабований незалежно від інших сервісів [2].

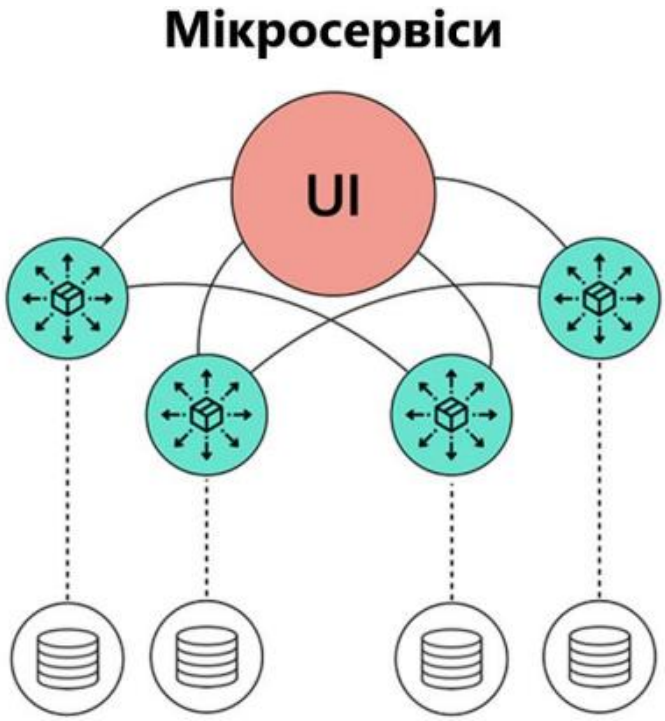


Рисунок 1.2 – Мікросервісна архітектура

Однією з ключових переваг цієї архітектури є масштабованість і гнучкість. Мікросервіси дозволяють незалежно масштабувати кожну службу для задоволення попиту на функцію програми, яку вона підтримує [2].

Для мікросервісів також характерні технологічна різноманітність і автономія. Оскільки кожен сервіс працює окремо, можна використовувати різні технології та фреймворки, за умови, що їхні інтерфейси не залежать від конкретних технологій (наприклад, використання HTTP API або вебсокетів). Мікросервіси формують динамічну екосистему, де кожен сервіс може розвиватися та вдосконалюватися у власному темпі.

Ще однією важливою перевагою є безперервна доставка та розгортання. Оскільки сервіси не залежать один від одного, їх можна розробляти, тестувати та розгортати окремо. Нові функції та оновлення можна впроваджувати без змін у всьому застосунку [2].

Одним із недоліків мікросервісної архітектури є складність управління розподіленою системою. Для координації взаємодії між сервісами, забезпечення узгодженості даних та обробки збоїв необхідне ретельне проєктування та впровадження. Оскільки мікросервіси потребують обміну даними між собою, виникає проблема забезпечення стійкості системи у випадку виходу з ладу одного з компонентів. Окрім цього, необхідно гарантувати функціонування інших сервісів у разі виникнення проблеми.

Таким чином, мікросервісна архітектура забезпечує масштабованість, технологічну різноманітність та безперервну доставку. Проте, це пов’язано з низкою труднощів, що виникають через складність управління розподіленими системами та взаємодією між сервісами.

1.1.3 Порівняльний аналіз мікросервісної та монолітної архітекутри

Для обґрунтування вибору архітектури був проведений порівняльний аналіз монолітної та мікросервісної моделі, результати наведені в таблиці 1.1.

Таблиця 1.1 – Опис відмінностей

Категорія	Монолітна	Мікросервісна
Проєктування	Єдина кодова база з кількома взаємозалежними функціями.	Незалежні програмні компоненти з автономними функціональними можливостями, які взаємодіють між собою через API.

Продовження табл. 1.1

Категорія	**Монолітна**	**Мікросервісна**
Розробка	Вимагає менше планування на початковому етапі, але складність розуміння та підтримки поступово зростає.	Вимагає більше планування на початковому етапі, але спрощує розуміння та підтримку завдяки ізольованим компонентам.
Розгортання	Весь застосунок розгорнута як єдине ціле.	Кожен мікросервіс є незалежним програмним об’єктом, що вимагає індивідуального контейнерного розгортання.
Налагодження	Весь шлях коду можна відстежувати в одному середовищі.	Потрібні складні інструменти налагодження, які вміють відстежувати обмін даними між кількома мікросервісами.
Модифікація	Невеликі зміни спричиняють великі ризики, оскільки торкаються всієї бази коду.	Можна змінювати окремі мікросервіси, не торкаючись застосунку в цілому.
Масштабування	Необхідно масштабувати весь проєкт, навіть якщо збільшиться навантаження лише на деякі функціональні області.	За потреби можна масштабувати окремі мікросервіси, що знижує загальні витрати на масштабування.

Популярні приклади застосування монолітної архітектури:

– застосунки для малого та середнього бізнесу: порівняно прості застосунки з обмеженим функціоналом та невеликими командами розробників;
– середовища з обмеженими ресурсами: монолітна архітектура може стати гарним рішенням для певних середовищ з обмеженими ресурсами інфраструктури або можливостями розгортання;
– застосунки, що виконують одну чітко визначену функцію без складної інтеграції або суттєвих вимог до масштабованості.

Популярні приклади застосування мікросервісної архітектури:

– застосунки з непостійними або непередбаченими робочими навантаженнями гарантовано виграють від використання мікросервісів;
– якщо для компонентів проєкту потрібні різні технології або фреймворки, гнучкість мікросервісів допоможе це реалізувати;
– мікросервіси відмінно узгоджуються з принципами безперервної інтеграції, доставки та розгортання: кожен сервіс можна розробляти, тестувати та розгортати окремо від інших.

Отже, для монолітної архітектури характерна простота, ефективність та середовище спільно використовуваних даних. З іншого боку, мікросервіси мають масштабованість, гнучкість і технологічну різноманітність.

1.2 Шаблон MVC

MVC (Model-View-Controller) – це архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення, передбачає розділення системи на три взаємопов’язані частини: модель даних, представлення (інтерфейс користувача) та контролер (див. рис. 1.3). Він застосовується для відокремлення даних (моделі) від

інтерфейсу користувача (представлення), забезпечуючи можливість внесення змін у інтерфейс користувача без впливу на роботу з даними та можливість змін в моделі даних без необхідності змінювати інтерфейс користувача.

Шаблон архітектури не суперечить ні монолітній, ні мікросервісній архітектурі. Це шаблон проєктування, який визначає структуру проєкту. Таким чином, є можливість мати як монолітний, так і мікросервісний застосунок, що використовує MVC для організації своєї архітектури.



Рисунок 1.3 – Шаблон MVC

Ключова концепція MVC полягає в тому, що будь-який застосунок з інтерфейсом користувача в першу чергу можна розчленувати на дві основні частини: модуль, що відповідає за реалізацію бізнес-логіки проєкту, і сам інтерфейс користувача.

У першому модулі реалізується основний функціонал проєкту, а другий модуль відповідає за реалізацію інтерфейсу користувача, включаючи відображення даних для користувача та логіку взаємодії з застосунку.

Після застосування такого підходу архітектура застосунку буде мати такий вигляд, як наведено на рисунку 1.4.



Рисунок 1.4 – Архітектура

Інтерфейс користувача, який був виділений в окремий модуль, ділиться на вигляд та контролер і виконує основне завдання – забезпечення взаємодії користувача з системою. Це означає, що в інтерфейсі існує всього дві функції:

– відображення та зручне відображення користувачеві інформації про систему;
– введення даних та команд користувача в систему (їх передача до системи).

Ці функції визначають, як інтерфейс користувача має бути розчленований на модулі.

У результаті архітектура системи виглядає так, як наведено на рисунку 1.5.



Рисунок 1.5 – Архітектура

Коли користувач вводить інформацію через контролер, він одночасно вносить зміни до моделі, а коли він отримує інформацію через елементи інтерфейсу (через Вигляд), він отримує інформацію про дані моделі.

Переваги застосування MVC особливо очевидні, коли користувачеві потрібно надавати одні й ті ж дані у різних формах, таких як таблиці, графіки або діаграми, використовуючи різні види. При цьому, без втручання в реалізацію вигляду, можна легко змінювати реакції на дії користувача, такі як натискання мишею на кнопці або введення даних.

1.3 Вибір архітектури та шаблону проєктування

Основною перевагою використання монолітної архітектури у застосунку є легке проєктування та розробка на початковому етапі. Оскільки додаток включає в себе кілька важливих функцій, таких як авторизація користувачів, управління зображеннями та підписками, єдина структура дозволяє їм тісно взаємодіяти в рамках єдиної кодової бази.

Монолітний підхід, попри обмеження у масштабованості та гнучкості, має перевагу у швидкості розгортання, налагодження та глобальному баченні проєкту. Використання мікросервісної архітектури, з її перевагами масштабування та ізоляції компонентів, вимагає більш складного планування. Таким чином, монолітна структура була обрана як найбільш раціональне рішення для вирішення завдань застосунку.

Окрім вибору монолітної архітектури, застосунок створювався за принципом патерну проєктування MVC, забезпечуючи чітке розділення логіки, представлення та управління даними. Цей підхід робить код структурованим і легшим в обслуговуванні, оскільки кожен компонент має чітко визначену роль. Модель відповідає за роботу з даними, включаючи їх збір, зберігання та обробку, через інтеграцію з сервісами та базами даних. Представлення відповідає за відображення інформації користувача,

використовуючи можливості шаблону Thymeleaf. У свою чергу, контролер керує обробкою запитів і взаємодією між користувачами та серверами, забезпечуючи логіку маршрутизації.

Застосунок складається з декількох ключових модулів, які забезпечують основні функції. Взаємодія між модулями покликана забезпечити сумісність модулів. Наприклад, модулі авторизації відповідають за вхід, вихід і реєстрацію користувачів, і ці процеси тісно пов’язані з модулями управління користувачами, що забезпечують зберігання і обробку персональних даних. Модулі оголошень і блогів реалізують їх створення, редагування, видалення та прив’язку до користувачів, зберігаючи цю інформацію в єдиній базі даних. Окремі компоненти обробки зображень дозволяють керувати файлами усіх основних функцій та інтегрувати їх у загальну структуру.

РОЗДІЛ 2 ТЕХНОЛОГІЧНІ АСПЕКТИ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1 Фреймворк Spring як основа для побудови застосунку

При реалізації вебзастосунків на мові програмування Java останнім часом найчастіше використовується фреймворк Spring, що обумовлено його архітектурою у вигляді контейнеру з впровадженням залежностей, який також включає зручні у використані модулі (див. рис. 2.1), що забезпечують доступ до бази даних, проксі, підтримують аспектно-орієнтоване програмування, забезпечують інфраструктуру MVC та інші.

У структурі фреймворку контейнер Spring відповідає за створення, конфігурацію та збирання об’єктів, відомих як bean-компоненти, а також за управління їхнім життєвим циклом. Щоб зібрати компоненти, контейнер використовує метадані конфігурації, які можуть бути у формі конфігурації XML або анотацій [1].

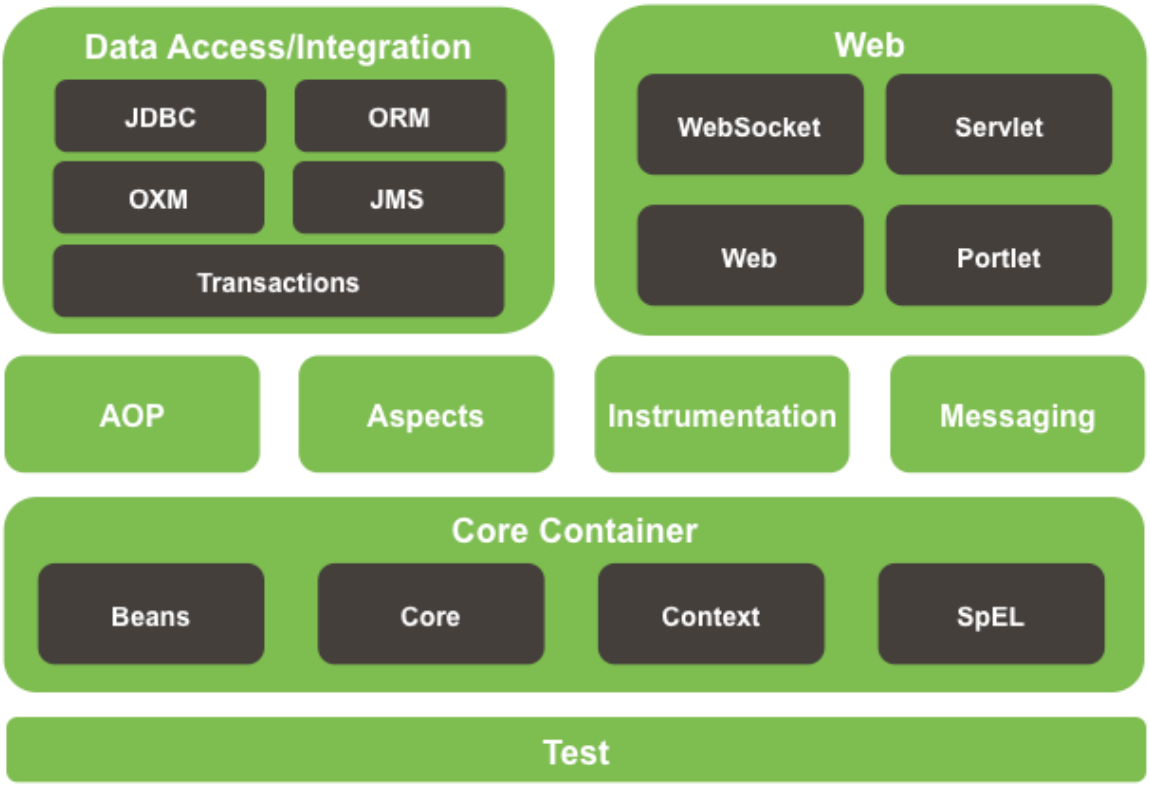


Рисунок 2.1 – Модулі Spring [3]

Основні функції застосунку були реалізовані з використанням можливостей Spring, що дозволило створити зручну, масштабовану та безпечну систему. Зокрема, функціонал входу та авторизації, захист сторінок від несанкціонованого доступу, підтвердження реєстрації через email та захист від CSRF-атак було реалізовано за допомогою Spring Security. Це стало можливим завдяки використанню анотацій для конфігурації безпеки. Реалізувати хешування паролів вдалося за допомогою компонента PasswordEncoder.

Spring Security – це структура, яка забезпечує аутентифікацію, авторизацію та захист від типових атак [4].

Завдяки компоненту MultipartFile, фреймворк забезпечує зручну роботу з файлами, що дозволяє реалізувати функцію завантаження зображень для оголошень і блогів. Зображення зберігаються у файловій системі, а посилання на них – в базі даних. Це спростило логіку зберігання медіафайлів.

Для обробки запитів користувачів та реалізації RESTAPI використовуються функції Spring MVC. Контролери додатку приймають HTTP-запити на додавання оголошень, редагування профілів та отримання списків оголошень. У методі контролера підтримуються як GET, так і POST запити, а параметри обробляються через анотації @RequestParam та @PathVariable.

IoC – це процес, за допомогою якого об’єкти визначають свої залежності (тобто інші об’єкти у взаємодії) через аргументи конструктора, параметри фабричного методу або властивості, встановлені після створення об’єктів. Spring-контейнер автоматично реалізує ці залежності при створенні контейнерів [3]. Механізм IoC реалізовано в декількох ключових компонентах програми.

Цей підхід використовується в контролерах. Наприклад, у будь-якому контролері залежність від сервісного класу реалізована через конструктор. Це дозволяє контролерам працювати з бізнес-логікою, визначеною в сервісі, не турбуючись про його створення та управління.

Серед переваг можна виділити використання Spring’ом принципів IoC та DI, що спрощує тестування, перевикористання компонентів та покращує підтримку модульності коду.

Ще однією з ключових переваг є автоматична конфігурація Spring Boot. Це дозволяє розгортати вебзастосунки швидше, з мінімальними витратами на налаштування, що спрощує процес розробки.

За допомогою Spring Security у застосунку реалізована можливість авторизації та аутентифікації користувачів. У фреймворку вже передбачено логіку логіну та логауту, що є ще одним плюсом у використанні. Налаштування Spring Security наведено у додатку В, на рисунку В.3.

Також, однією з ключових переваг є реалізація обробки HTTP-запитів у Spring MVC. Запит користувача з браузера надходить до DispatcherServlet, який маршрутизує цей запит до відповідного методу контролера, визначеного за допомогою анотацій @RequestMapping, @GetMapping або @PostMapping, контролери наведено у додатку Г.

До недоліків можна віднести ресурсомісткість, так як застосунки, побудовані на Spring, часто займають більше ресурсів, особливо при неправильній конфігурації. Наприклад, Spring Boot може завантажити непотрібні залежності.

Використання Spring накладає певні обмеження, оскільки застосунок стає сильно залежним від «екосистеми». Це може ускладнити міграцію на іншу технологію, особливо якщо система вже містить велику кількість спеціалізованих рішень на базі Spring.

Так як Spring підтримує багато залежностей та конфігурацій, налагодження проблем може ускладнитись. Автоматична конфігурація Spring Boot може спричинити неправильну поведінку.

Використання Spring у застосунку обумовлене здатністю вирішувати завдання розробки завдяки широким вбудованим можливостям. Цей фреймворк допомагає реалізувати у застосунку важливі функції, а саме – реєстрація та аутентифікація, взаємодія між компонентами, а також обробка та валідація даних.

За допомогою Spring Security реалізована основна безпека застосунку. Інтеграція reCAPTCHA та валідація даних користувачів – важливий аспект надійності системи.

Використання Spring Data JPA значно спрощує роботу з базою даних. З її допомогою, можливо легко керувати записами, оновлювати та видаляти дані, оброблювати пов’язані сутності.

Spring робить застосунок більш масштабованим та зручним для підтримки. Завдяки його модульності можна додавати нові функції та покращення, не торкаючись основних частин коду.

2.2 Використання Thymeleaf для зображення динамічних даних

Thymeleaf – це сучасний шаблонізатор Java на стороні сервера, який дозволяє створювати природні HTML-шаблони, зручні для розробки та попереднього перегляду прямо в браузері. У застосунку Thymeleaf інтегровано зі Spring MVC, що забезпечує ефективну взаємодію між контролерами, сервісами та користувацьким інтерфейсом [5] (див. рис. 2.2).

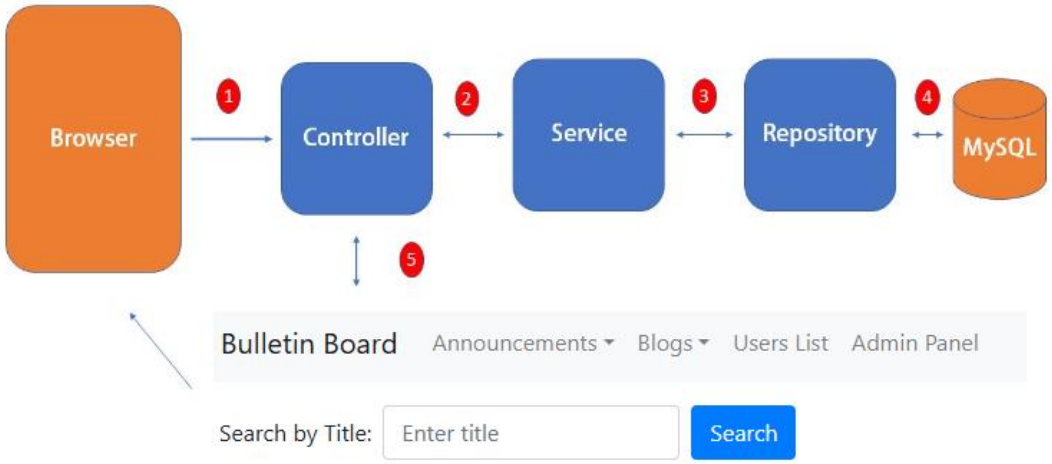


Рисунок 2.2 – Зв’язок між Thymeleaf та іншими компонентами системи

Основною метою використання Thymeleaf у додатку є відображення користувацьких інтерфейсів та динамічного вмісту сторінок. Цього вдалося

досягти завдяки інтеграції Thymeleaf з Spring MVC. Використовуючи вираз «${}», можна передавати дані від контролера до об’єкта моделі, такі як заголовок оголошення, його опис, категорію та фотографії, без потреби у кодуванні HTML.

Однією з ключових функцій, реалізованих за допомогою Thymeleaf, є створення форми входу та реєстрації, а також форми для додавання оголошень і блогів.

Крім того, Thymeleaf відіграє особливу роль у реалізації функцій пошуку та фільтрації. Поля введення пошуку та випадаючі списки підключаються до сервера за допомогою Spring MVC, а результати динамічно виводяться у список за допомогою циклу th:each.

Інший приклад використання Thymeleaf у застосунку – контроль доступу до функцій в залежності від ролі або стану користувача. Наприклад, використовуючи атрибути th:if і th:unless, можна відображати кнопки «Редагувати» або «Видалити» тільки для авторів оголошень або блогів.

Завдяки підтримці валідації Spring у Thymeleaf, якщо дані введені неправильно, користувач побачить повідомлення про помилку безпосередньо на сторінці. Наприклад, при заповненні форми реєстрації, якщо дані не відповідають правилам валідації, відображається помилка, пов’язана з недійсним паролем, існуючим нікнеймом або помилка валідації captcha.

Thymeleaf підтримує інтеграцію з URL-адресами застосунків, реалізовану за допомогою атрибутів th:href і th:action. Це гарантує, що посилання на такі сторінки, як деталі списку, редагування профілю або реєстрація, будуть побудовані правильно. Thymeleaf дозволяє створювати гнучкі, динамічні та зручні користувацькі інтерфейси, які інтегруються з внутрішньою логікою та забезпечують високу якість роботи з додатком.

До переваг використання Thymeleaf можна віднести можливість роботи з динамічними даними, циклами або умовами напряму у шаблонах. Наприклад, у створюваному застосунку можна відобразити список усіх існуючих оголошень.

Оскільки Thymeleaf створювався з урахуванням Spring, його можна легко інтегрувати та налаштувати у застосунок, що є одним із плюсів його використання у застосунку.

Також, Thymeleaf дозволяє створювати шаблони, які виглядають як звичайний HTML, завдяки чому можна побачити сторінку ще до впровадження даних.

Ще однією перевагою є автоматичне екранування виведення даних, що допомагає захиститись від впровадження шкідливого коду, тобто XSS-атаки.

Серед недоліків можна зазначити те, що кожен шаблон оброблюється на сервері, що може збільшити навантаження на сервер.

Незважаючи на свою популярність у Java-розробці, Thymeleaf погано підходить для інших мов або фреймворків.

Таким чином, Thymeleaf може створювати гнучкі, динамічні та зручні користувацькі інтерфейси, які інтегруються з логікою бекенду та забезпечують високу якість роботи з додатком.

Використання Thymeleaf у застосунку обумовлено тим що, він ідеально підходить для створення динамічних вебінтерфейсів у застосунках на Spring. Цей шаблонизатор інтегрується з Spring Boot і спрощує створення багатофункціональних HTML сторінок, які можна оновлювати, не торкаючись серверної частини. Цей інструмент особливо корисний для застосунку, адже за його допомогою можна відображати користувачу динамічні дані, наприклад, список оголошень, профілі користувачів, повідомлення про помилки тощо.

2.3 Роль Spring Data JPA та Hibernate у взаємодії з базою даних

Hibernate використовується для реалізації роботи з базами даних, спрощення доступу до даних та управління ними за рахунок високорівневої абстракції. Ця технологія дозволяє інтегрувати об’єктно-реляційне

відображення (ORM) в логіку програми, уникаючи створення великої кількості SQL-запитів.

Hibernate, як рішення ORM, фактично являє собою рівнем доступу до даних додатків Java і реляційною базою даних, як можна побачити на рисунку 2.3. Програма Java використовує Hibernate API для завантаження, зберігання, запиту даних свого домену [6].

Під час запуску програми Hibernate перевіряє відповідність між об’єктом коду та таблицею бази даних. Наприклад, при створенні нового поля в класі оголошення, Hibernate оновить структуру таблиці, що допомагає при розробці.

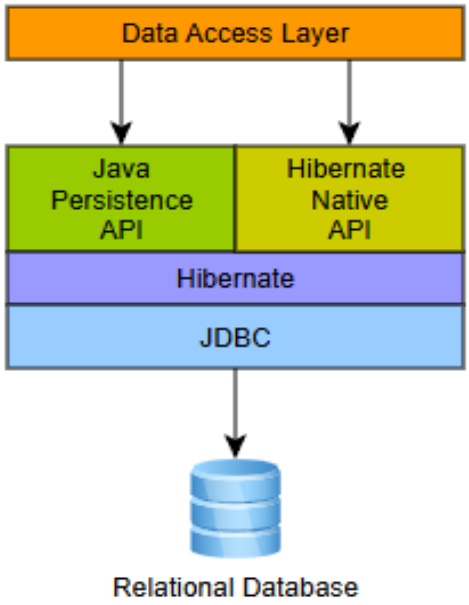


Рисунок 2.3 – Структура Hibernate [6]

Spring Data JPA – це популярна структура для доступу до даних, яка допомагає писати логіку доступу до даних/персистентності з меншим кодом. Spring Data JPA – це рівень абстракції над JPA, який зменшує обсяг коду, який ми пишемо, і може використовуватися для ефективного написання логіки за допомогою репозиторіїв на основі JPA [7] (див. рис. 2.4).

Spring Data JPA реалізує функціонал створення, читання, оновлення та видалення (CRUD) об’єктів в базі даних, за допомогою цього, у застосунку реалізован функціонал усіх CRUD-операцій для оголошень, блогів та

користувачів. Наприклад, оголошення, блоги, користувачі та їх підписки представлені як Java-об’єкти (сутності), що зберігаються в базі даних і автоматично конвертуються Hibernate.

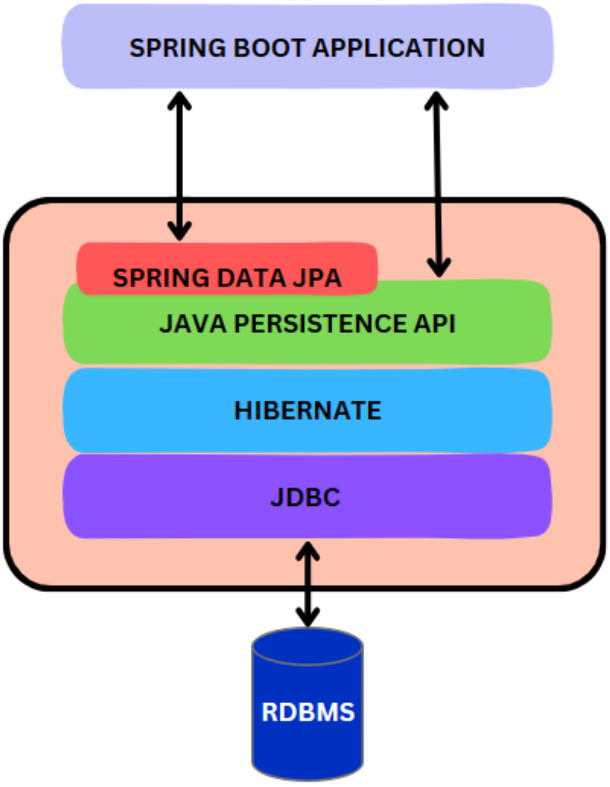


Рисунок 2.4 – Структура Spring Data JPA [7]

Однією з найважливіших особливостей Spring Data JPA є підтримка побудови запитів з використанням ключових слів у назвах методів. Це дозволяє створювати складні запити, такі як метод findByCategory(String category) в репозиторії оголошень, який використовується для реалізації сортування.

Серед переваг можна виділити те, що Spring Data JPA надає вже готові методи для стандартних операцій створення, читання, оновлення та видалення (CRUD). Тобто не потрібно писати SQL для базових операцій – усе це забезпечує фреймворк через базові методи.

Замість ручного написання запитів можна створити метод з інтуїтивно зрозумілою назвою і Spring Data JPA самостійно згенерує відповідний SQL-запит.

Ще однією з переваг є те, що Spring Data JPA може інтегрувати інші модулі Spring, такі як Spring Security, для роботи з даними користувача та відображення результатів запитів у вебінтерфейсі.

Завдяки готовим репозиторіям Spring Data JPA зменшує обсяг шаблонного коду.

До недоліків можна віднести потребу конкретної реалізації JPA. Тому, у разі переходу на іншу реалізацію JPA або ORM-підхід, можуть виникнути проблеми з сумісністю.

Spring Data JPA чудово підходить для базових CRUD-операцій, але для складніших запитів іноді потрібно використовувати нативні SQL-запити або переносити частину логіки на рівень сервісів, що може знизити ефективність.

Іноді автоматична генерація запитів призводить до того, що Spring Data JPA робить більше запитів, ніж потрібно, що може погано впливати на продуктивність. Наприклад, при неправильній налаштуванні зв’язків @OneToMany або @ManyToOne може виникнути «проблема N+1».

Використання Spring Data у проєкті полегшує роботу з базою даних, роблячи її набагато зручнішою та гнучкішою. У застосунку, в якому потрібно оброблювати складні сутності, такі як користувачі, анонси, підписки та блоги, Spring Data допомагає спростити взаємодію з базою даних за рахунок автоматизації основних операцій.

Spring Data надає стандартний набір методів та підтримує можливість написання нових методів за допомогою гнучких іменованих запитів. Це дозволяє зосередитись на бізнес-логіці та функціоналі, а не на рутинному коді для роботи з базою даних. Крім того, Spring Data JPA автоматично адаптує роботу з сутностями, стежить за їх станом та забезпечує оптимальне керування транзакціями.

Таким чином, використання Spring Data допомагає значно знизити обсяг коду, робить його більш читаним та підтримуваним, а також спрощує управління даними, що критично важливо у додатках з активною роботою з базою даних, як у застосунку дипломній роботі.

2.4 Організація даних за допомогою MySQL

MySQL використовується як основна система управління базами даних для зберігання, організації та доступу до даних, що використовуються у вебзастосунку. Ця технологія відповідає за надійне зберігання інформації про користувачів, оголошень, блогів, підписок та інших об’єктів застосунку.

Реляційна модель MySQL дозволяє ефективно працювати з великими обсягами структурованих даних. Кожен об’єкт застосунку (користувачі, оголошення, блоги тощо) зіставляється з окремою таблицею бази, а зв’язки між ними встановлюються за допомогою первинних і зовнішніх ключів.

Індексування первинних ключів та інших полів дозволяє отримувати необхідні дані з мінімальними затримками. Наприклад, таблиця користувачів має зв’язок з таблицею оголошень і може зберігати всі оголошення, створені конкретним користувачем, і легко отримувати ці дані при необхідності.

MySQL також зберігає дані аутентифікації користувачів. Всі паролі шифруються за допомогою Spring Security, а MySQL забезпечує їх надійне збереження у зашифрованому вигляді, дозволяючи додатку використовувати ці дані для перевірки користувача під час входу.

Запити до бази даних, зроблені за допомогою Spring Data JPA, автоматично перетворюються на SQL-запити.

Серед переваг MySQL можна виділити здатність оброблювати великі обсяги даних, забезпечуючи стабільну продуктивність навіть під навантаженням, що добре впливає на застосунок в якому потрібно керувати користувачами, зображеннями, оголошеннями та багатьом іншим.

Також, MySQL добре інтегрується з багатьма мовами програмування (Java, Python тощо) і платформами. Багато популярних фреймворків та інструментів пропонують готові бібліотеки для інтеграції з MySQL, наприклад у Spring – це MySQL Connector.

Ще однією перевагою є підтримка масштабування як у вертикальному напрямку (на потужнішому обладнанні), так і в горизонтальному (розподіл бази даних між кількома серверами).

Крім того, MySQL надає інструменти для керування транзакціями. Наприклад, якщо ж під час реєстрації користувача або додавання нового оголошення виникне збій, MySQL відкатить зміни.

До недоліків можна віднести погану продуктивність з великим обсягом завдань, які виконуються одночасно. Чим більше завдань виконується, тим помітніші простої та сповільнення.

Оскільки, у проєкті є безліч сутностей, між якими потрібно встановлювати взаємозв’язки та забезпечувати постійне збереження даних, використання MySQL обумовлюється завдяки точній системи управління відносинами між записами, що дозволяє швидко створювати зв’язки між сутностями (наприклад, пов’язувати користувачів з їх оголошеннями чи підписками).

2.5 Застосування Bootstrap під час розробки інтерфейсу користувача

У застосунку Bootstrap використовується для створення адаптивного та зручного інтерфейсу користувача. Ця технологія забезпечує ефективне відображення вебсторінок на різних пристроях і екранах, зберігаючи зручність.

Щоб використовувати фреймворк, необхідно підключити його до сторінки. У застосунку підключення до базового шаблону відбувається через теги link та script, що забезпечує доступ до інструментів фреймворку, а саме: колонкова система (сітка Bootstrap), класи та компоненти.

Наприклад, щоб створити кнопку на сторінці за допомогою Bootstrap, достатньо до посилання або елементу button додати лише кілька класів.

Bootstrap складається з:

– інструментів для створення макета: у застосунку, на великому екрані оголошення упорядковані у три стовпці (col-md-4), а на мобільних пристроях автоматично переходять у вертикальний список;

– класів для стилізації контенту: відображення зображень у блогах реалізовано за допомогою класу img-fluid, який забезпечує адаптацію зображень по ширині контейнера;
– готових компонентів: кнопок, форм, горизонтальних і вертикальних, навігаційних панелей, списків, що випадають, модальних вікон, підказок та ін;
– утилітних класів: утилітні класи дозволяють швидко налаштувати відступи (mt-3, mb-4), вирівнювання тексту (text-left, text-right) та зміну кольору елементів (bg-light, text-muted).

Система сітки Bootstrap використовується для упорядкуванні контенту на сторінках. Наприклад, оголошення та блоги відображаються у вигляді карток, вирівняних у рядок, зберігаючи чітке та зрозуміле представлення інформації незалежно від розміру екрана.

Серед переваг можна виділити можливість швидкого введення нового функціоналу на сайт та надання його користувачам. Таким чином, це вирішує одну із проблем циклу «ідея – функціонал – верстка».

Крім того, Bootstrap надає багато готових компонентів – таких як навігаційні панелі (головне меню створено за допомогою компонента navbar), кнопки, картки (використовуються для відображення оголошень) та форми (наприклад форма реєстрації включає класи form-group і form-control).

Оскільки, у вебпросторі існує багато різних браузерів: Chrome, Firefox, Edge, Opera, Safari, тощо, кожен з яких оброблює HTML і CSS по-різному, через що верстальникам необхідно приводити стандартні елементи до єдиного вигляду, Bootstrap враховує ці відмінності та оптимізує компоненти таким чином, щоб мінімізувати різницю між відмінностями браузерів, що є плюсом.

Одна з ключових переваг Bootstrap – його мобільна адаптивність із коробки. Проєкт автоматично підлаштовується під екрани різних розмірів. Наприклад, сторінка перегляду списку оголошень використовує колонки (col-md-4), які на великих екранах відображають три елементи в ряд, а на мобільних пристроях контент автоматично стає вертикальним списком.

Bootstrap також дозволяє підтримувати проєкт на довгостроковій основі завдяки єдиному стилю. Якщо ж потрібно змінити зовнішній вигляд, це можна зробити змінивши основні змінні Bootstrap у файлах налаштувань.

Серед недоліків можна зазначити те, що фреймворк Bootstrap дає як набір класів, так і інтерактивні компоненти. Для їх роботи використовується кілька JavaScript-бібліотек, які збільшують вагу файлів, що підключаються на сторінці, що може сповільнити застосунок.

Стандартні компоненти накладають обмеження на дизайн. Ці рамки виражені в тому, що компоненти можна розширювати тільки вглиб, вставляючи одні компоненти всередину інших. Модифікація компонентів вимагатиме знань у верстці, оскільки навіть незначні зміни необхідно перевірити у браузерах та на різних дозволах. Дизайн стандартних компонентів Bootstrap лаконічний і є прикладом для дизайнера.

Використання Bootstrap у проєкті робить відчутний внесок у покращення інтерфейсу та загальної візуальної привабливості застосунку. Bootstrap допомагає забезпечити єдиний стиль інтерфейсу незалежно від типу пристрою, на якому його переглядають.

2.6 Схема даних та класова модель вебзастосунку

Діаграма класів представляє архітектуру програми, показуючи взаємодію між її компонентами, сутностями та сервісами. Вона демонструє, як реалізуються основні функції та як дані проходять через різні рівні системи.

Користувач, оголошення, блог і зображення – основні об’єкти системи (див. рис. 2.5). Користувачі пов’язані з оголошеннями і блогами, а самі блоги та оголошення пов’язані з зображеннями. Ці зв’язки показують, що кожен користувач може створювати блоги або оголошення і завантажувати в них медіафайли.

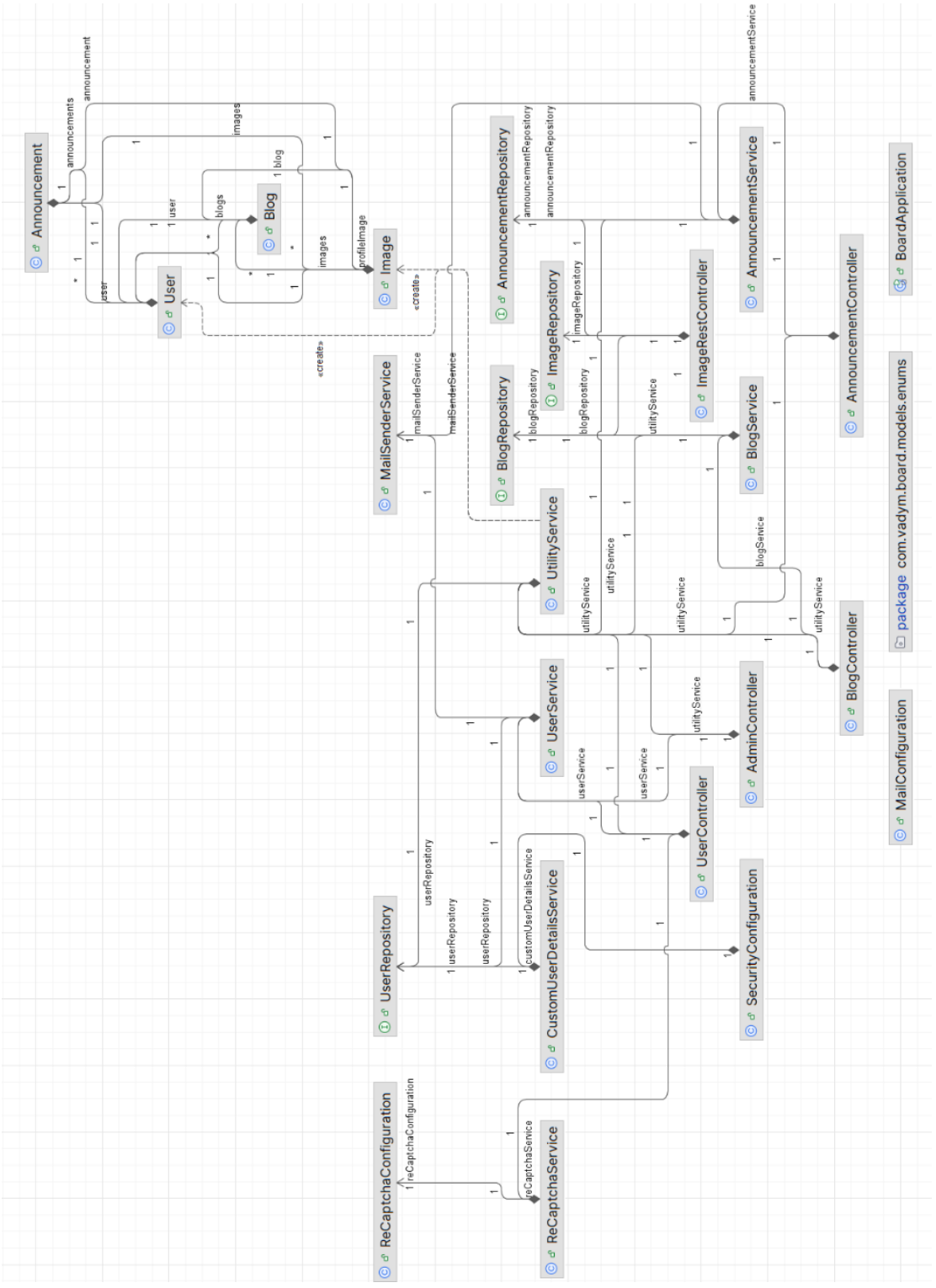


Рисунок 2.5 – Діаграма класів

За роботу з даними відповідають репозиторії користувачів, оголошень, блогів та зображень. Ці компоненти інтегровані через Spring Data JPA і можуть працювати з даними на високому рівні абстракції.

Сервісні класи UserService, AnnouncementService, BlogService та UtilityService виконують бізнес-логіку. UserService керує реєстрацією, авторизацією та взаємодією з користувачами. AnnouncementService та BlogService забезпечують логіку для створення, редагування та видалення контенту, тоді як UtilityService виступає як звичайний допоміжний компонент, наприклад, для конвертації валюти.

Безпеку застосунку реалізовано за допомогою компонентів SecurityConfiguration та CustomUserDetailsService, які забезпечують авторизацію та аутентифікацію. Ці компоненти інтегровані з UserRepository і використовують дані користувачів для перевірки облікових записів.

Контролери UserController, AdminController, AnnouncementController, BlogController, ImageRestController керують взаємодією користувача у системі. Ці класи приймають запити, обробляють їх через сервіси та повертають відповіді за допомогою Thymeleaf.

Крім того, застосунок використовує MailSenderService, який відповідає за надсилання сповіщень електронною поштою, наприклад для підтвердження реєстрації або сповіщення користувачів.

Компоненти ReCaptchaService та ReCaptchaConfiguration інтегрують захист від ботів у формі реєстрації та логіну.

Вся архітектура програми має визначений організаційний розподіл обов’язків між компонентами, що полегшує обслуговування та розширення функціональних можливостей, оскільки кожна частина системи виконує завдання та взаємодіє з іншими частинами через зрозумілий інтерфейс.

Діаграма бази даних показує структуру сховища даних застосунку та зв’язок між сутностями (див. рис. 2.6). Основні елементи архітектури включають користувачів, оголошення, блоги та таблиці зображень, які представляють ключові елементи системи.

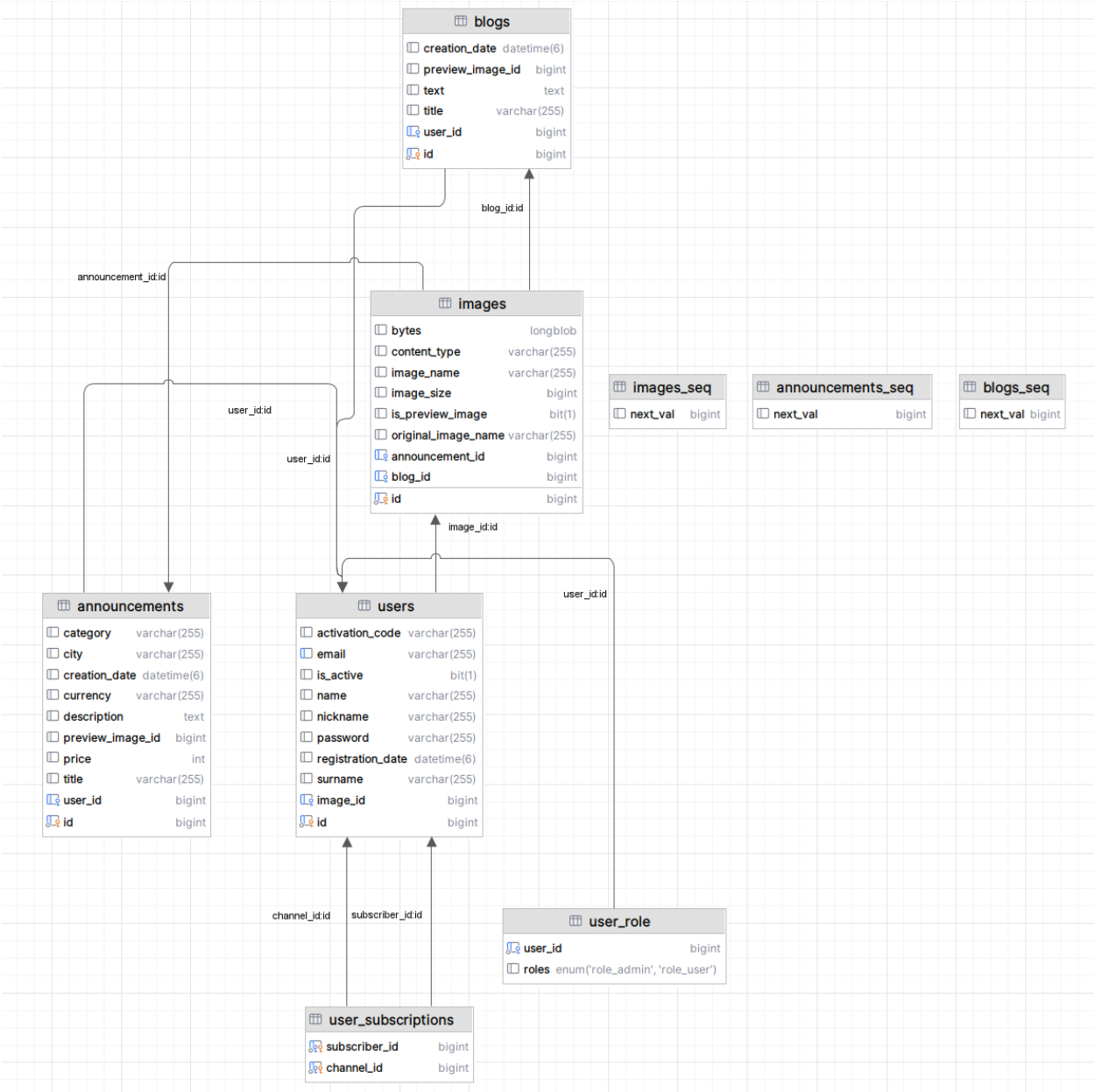


Рисунок 2.6 – Діаграма бази даних

У таблиці users зберігається інформація про зареєстрованих користувачів. Таблиця user_role пов’язує користувачів з роллю admin або user, що дає користувачам відповідні права.

Таблиці announcements та blogs зберігають контент, створений користувачами. Кожен запис у цих таблицях пов’язаний з певним користувачем через поле user_id. В цих таблицях зберігаються такі дані, як заголовок, текст, дата публікації та посилання на зображення в полі preview_image_id. Це поле пов’язане з таблицею зображень і зберігає метадані зображення та двійкові дані.

Таблиця images зберігає усі завантажені зображення. Вона містить поля для назв файлів, типу контенту, розмірів зображення та інформацію стосовно того, чи є зображення прев'ю. Кожне зображення може бути пов'язане з блогом або оголошенням за допомогою зовнішніх ключів – blog_id та announcement_id відповідно.

Щоб реалізувати підписки між користувачами, використовується таблиця user_subscriptions, яка з'єднує підписників і канали через поля subscriber_id і channel_id. В результаті користувачі можуть підписатися один на одного, щоб отримувати сповіщення про нові публікації.

Таблиці images_seq, announcements_seq і blogs_seq являють собою послідовність, яка генерує унікальний ідентифікатор для відповідних таблиць. Це важливо для застосунків, що працюють під високим навантаженням і паралельних операцій.

Вся структура бази даних розроблена з урахуванням нормалізації, щоб мінімізувати дублювання даних і забезпечити цілісність даних. Зв'язки між таблицями реалізуються за допомогою зовнішніх ключів, що дає змогу ефективно виконувати складні запити, які об'єднують дані з кількох сутностей.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Реалізація структури даних

У додатку А, рисунок А.1 представлений **клас Announcement**, який являє собою модель оголошення у застосунку і містить дані про кожне оголошення, такі як: унікальний ідентифікатор, що генерується автоматично (id), заголовок оголошення (title), категорія (category), ціна (price), валюта (currency), місто (city), опис (description) для зберігання великої кількості тексту, автоматична дата створення об’єкту (creationDate), ідентифікатор прев’ю зображення (previewImageId) та список зображень, пов’язаних з оголошеннями (images).

Список зображень пов’язаний з об’єктом зображення за допомогою каскадної та відкладеної ініціалізації. Поле user зберігає інформацію про користувача, який створив оголошення і посилається на відповідну сутність. CascadeType.REFRESH використовується для оновлення інформації за необхідності, а FetchType.LAZY для відкладеного завантаження.

Клас містить метод init(), позначений анотацією @PrePersist, який автоматично встановлює дату і час створення оголошення на поточну перед збереженням оголошення в базі даних. Метод addAnnouncementImage() використовується для додавання зображень до оголошення. При додаванні зображення створюється посилання на відповідне оголошення.

Анотації бібліотеки Lombok @Getter, @Setter, @AllArgsConstructor і @NoArgsConstructor використовуються для генерації гетерів, сетерів і конструктора з усіма параметрами відповідно.

У додатку А, рисунок А.2 представлений **клас Image**, який являє собою модель зображення у застосунку, яке може бути пов’язане як з оголошеннями (Announcement), так і з блогами (Blog) і містить дані про кожне оголошення, такі як: унікальний ідентифікатор зображення

генерується автоматично (id); також, клас містить поле для зберігання розміру зображення в байтах (imageSize), системну назву файлу (imageName), оригінальну назву зображення (originalImageName), типи вмісту MIME, що визначають формат зображення (contentType), і булеве поле, що вказує, чи є зображення прев'ю оголошення або блогу (isPreviewImage).

Зображення зберігається в базі даних в байтовому масиві з використанням типу LongBLOB, що дозволяє зберігати великі за розміром файли. Зображення може бути пов'язане з оголошеннями або блогами. Обидва поля жадібно завантажуються (FetchType.EAGER), що означає, що вони миттєво отримуються при зверненні до сутності зображення.

Класи позначені анотаціями @Entity і @Table (name = «images»), які вказують, що вони є сутностями JPA і відповідають таблицям у базі даних. Це дозволяє керувати зображеннями в контексті оголошень та роботи з блогами.

У додатку А, рисунок А.3 представлений **клас User**, який являє собою модель користувача у застосунку та реалізує інтерфейс UserDetails для інтеграції із системою Spring Security, надаючи методи для керування аутентифікацією та авторизацією. Поля класу: унікальний ідентифікатор користувача, який генерується автоматично (id), ім'я (name) та прізвище (surname) користувача, унікальний нікнейм (nickname), адресу електронної пошти (email), яка слугує унікальним іменем користувача для аутентифікації, та код активації, що використовується для підтвердження реєстрації (activationCode).

Пароль користувача зберігається у відповідному полі (password), а дата реєстрації (registrationDate) автоматично встановлюється під час створення об'єкта. Поле isActive визначає, чи є користувач активним.

Ролі користувача, такі як ROLE_ADMIN або ROLE_USER, зберігаються у таблиці user_role і визначають права доступу. Користувач також має списки оголошень та блогів, які він створив.

Клас реалізує логіку підписок між користувачами через колекції subscribers і subscriptions. Поле subscribers містить підписників користувача, тоді як subscriptions відображає користувачів, на яких підписаний даний користувач.

Метод класу isAdmin() перевіряє, чи має користувач права адміністратора. Метод getAuthorities() повертає роль користувача для авторизації в Spring Security. Метод getUsername() повертає адресу електронної пошти як ім’я користувача, а isEnabled() перевіряє, чи активований користувач.

Цей клас дозволяє зберігати та оброблювати всю необхідну інформацію про користувача, включаючи аутентифікацію, авторизацію, профільну інформацію та підписки.

У додатку А, рисунок А.4 представлений **клас Blog**, який являє собою модель блогу у застосунку і містить дані про кожен блог, такі як: унікальний ідентифікатор блогу (id), поле для зберігання назви блогу (title), основного тексту, який може включати довгі записи (text), а також дату створення, яка автоматично встановлюється перед збереженням у базу даних (creationDate).

Поле previewImageId містить ідентифікатор зображення, що використовується як прев’ю зображення блогу. Поле images містить список зображень, пов’язаних з блогом. Зображення зв’язані із сутністю Image з використанням каскадного завантаження (CascadeType.ALL) та відкладеної ініціалізації (FetchType.LAZY).

Користувач, який створив блог, встановлюється у полі user, яке пов’язане із сутністю User.

Клас має метод addBlogImage(), який дозволяє додавати зображення до блогу та встановлює зв’язок між блогом і доданим зображенням.

Цей клас дозволяє зберігати всі необхідні дані, пов’язані з блогами, включаючи текст, зображення, і інформацію про автора.

У додатку А, рисунок А.5 представлений **інтерфейс ImageRepository**, який успадковується від JpaRepository, що дозволяє використовувати

стандартні CRUD (створення, читання, оновлення, видалення) операції для сутності Image.

У додатку А, рисунок А.6 представлений **інтерфейс UserRepository**, який успадковується від JpaRepository що надає стандартні методи для роботи з сутністю User. У цьому інтерфейсі визначені методи, які наведено нижче.

Метод findByEmail дозволяє знайти користувача за адресою електронної пошти. Він використовується переважно для перевірки унікальності електронної пошти під час реєстрації.

Інший метод, findByActivationCode, допомагає знайти користувача за його кодом активації, який необхідний для активації облікового запису після реєстрації.

Для пошуку за нікнеймом використовується метод findByNickname, який використовується для перевірки нікнейму на унікальність.

Більш гнучкий метод findByNicknameContainingIgnoreCase шукає всіх користувачів, чий нік містить заданий текст у будь-якому регістрі. Цей метод зручний для реалізації функцій пошуку за частиною імені.

У додатку А, рисунок А.7 представлений **інтерфейс BlogRepository**, який успадковується від JpaRepository що надає стандартні методи для роботи з сутністю Blog. У цьому інтерфейсі реалізовано один метод: List<Blog> findByTitleContainingIgnoreCase(String title), який знаходить усі блоги, в яких поле title містить підрядок, переданий у параметрі title. Цей метод використовується у застосунку для пошуку блогів за назвою без урахування регістру.

У додатку А, рисунок А.8 представлений **інтерфейс AnnouncementRepository**, який успадковується від JpaRepository що надає стандартні методи для роботи з сутністю Announcement. У цьому інтерфейсі визначені методи, які наведено нижче.

Метод findByTitleContainingIgnoreCaseAndCategory(String title, String category) знаходить усі оголошення, в яких поле title містить переданий

підрядок без урахування регістру, одночасно з полем category. Цей метод використовується для пошуку оголошень по назві з урахуванням категорії.

List<Announcement> findByTitleContainingIgnoreCase(String title) аналогічен попередньому методу, але використовується для пошуку без урахування категорії.

Метод List<Announcement> findByCategory(String category) знаходить усі оголошення, в яких поле category збігається з переданим значенням. Цей метод використовується для сортування.

У застосунку сутності та репозиторії є основним засобом роботи з даними, які забезпечують структуру та доступ до бази даних. Їх спільна робота дозволяє спростити взаємодію даних з бізнес-логікою.

Класи сутності являють собою моделі даних, які описують структуру таблиць в базі даних за допомогою анотацій @Entity і @Table. Кожна сутність містить поле, що представляє відповідний стовпець таблиці. За допомогою таких класів, розробники можуть керувати даними за допомогою звичайних об’єктів Java.

Інтерфейси репозиторіїв позбавляють від необхідності писати SQL-код вручну. Важливу роль відіграє взаємозв’язок між сутностями та репозиторіями: репозиторій знає, як працювати з конкретною сутністю, що дозволяє організувати доступ до даних через репозиторії.

3.2 Реалізація бізнес-логіки

Компоненти, які позначені анотацією @Service, вказують, що вони зберігають бізнес-логіку. Крім того, ця анотація використовується на рівні обслуговування, вона не має іншого спеціального використання.

У додатку Б, на рисунках Б.1 та Б.2 представлено сервісний **клас AnnouncementService**. Далі розглянемо методи.

Метод filterByTitle фільтрує оголошення за заголовком. Він розбиває

заданий заголовок на ключові слова і шукає по кожному з них. Якщо заголовок не вказано, метод повертає всі доступні оголошення.

Інший метод filterOnCategoryPage дозволяє шукати оголошення за заголовком у певній категорії. Якщо не вказано ні заголовка, ні категорії, метод повертає всі оголошення, що належать до попередньо обраної категорії.

Для сортування оголошень використовується метод sortAnnouncements. Він дозволяє сортувати оголошення за ціною або датою, в залежності від обраного порядку – від найнижчої до найвищої або навпаки.

Додавання нового оголошення здійснюється за допомогою методу addAnnouncement. При цьому враховується інформація про користувача, оголошення та пов’язані з ним зображення. Якщо завантажено одне або декілька зображень, вони обробляються і одне з них встановлюється в якості прев’ю. За бажанням користувача система також може надсилати повідомлення підписникам про появу нового оголошення.

Редагування оголошень реалізовано за допомогою методу updateAnnouncement. Цей метод перевіряє кожен переданий параметр і, якщо значення не є порожнім, оновлює відповідне поле оголошення.

Для видалення оголошень використовується метод deleteAnnouncement, який видаляє оголошення за його ідентифікатором.

Якщо необхідно отримати конкретне оголошення, це можна зробити за допомогою методу getAnnouncementById.

Оголошення також можна переглядати за категорією, використовуючи метод getAnnouncementsByCategory, або ж отримати список усіх доступних оголошень через getAllAnnouncements.

У додатку Б, на рисунку Б.3 представлено сервісний **клас CustomUserDetailsService**, який реалізує інтерфейс UserDetailsService. Це потрібно для того щоб Spring Security міг порівняти дані аутентифікації, введені користувачем, з даними, що зберігаються у БД.

За допомогою метода loadByUsername(), знаходиться користувач у базі даних за полем email. А Якщо користувача не знайдено, метод викидує

виключення UsernameNotFoundException. Якщо ж користувача знайдено, метод обертає його в об’єкт UserDetails, який Spring Security використовує для перевірки даних.

У додатку Б, на рисунку Б.4 представлено сервісний **клас ReCaptchaService**, який потрібен для корректної роботи reCAPTCHA.

Метод boolean validateRecaptcha(String response) приймає response, який містить токен reCAPTCHA, відправлений з клієнтської сторони. За допомогою String.format, формується URL для перевірки, куди підставляється секретний ключ та response. Потім, за допомогою мапи створюється request з параметром response, який необхіден для POST запросу. Після цього, за допомогою методу RestTemplate restTemplate.postForObject() надсилається HTTP-запит до Google reCAPTCHA API. Якщо ключ “success” в результаті запиту має значення true, то це означає що reCAPTCHA пройдено.

У додатку Б, на рисунку Б.5 представлено сервісний **клас MailSenderService**, який відповідає за відправку повідомлень. За допомогою цього сервісу реалізована логіка відправки коду активації після реєстрації та повідомлення підписників про нові оголошення або блоги.

Поле javaMailSender – Основний компонент для надсилання електронної пошти.

Поле username – Змінна, яка зберігає ім’я відправника. За допомогою анотації @Value можна ввести значення полів в компонентах, керованих Spring, і її можна застосовувати на рівні параметрів поля або конструктора/методу. Саме значення знаходиться у конфігураційному файлі.

Метод sendMessage(String sendTo, String subject, String message) відправляє e-mail на вказану адресу (sendTo) із заданою темою (subject) та текстом повідомлення (message).

За допомогою методу sendActivationCode(User user) надсилається лист з активаційним кодом. Використовуючи String.format, метод формує лист з ім’ям користувача та кодом активації, який присвоюється після реєстрації.

Після цього, сформоване повідомлення відправляєтеся на пошту, яку користувач вказав під час реєстрації.

Метод notifySubscribers(Announcement announcement) повідомляє всіх підписників користувача про нове оголошення. Метод отримує список підписників користувача, для кожного з котрих формується персональне повідомлення.

У додатку Б, на рисунках Б.6 та Б.7 представлено сервісний **клас UtilityService**, який надає методи для різних операцій у застосунку, включаючи обробку зображень, конвертацію валют, а також встановлення прев'ю-зображень для оголошень та блогів.

Поле userRepository використовується для отримання користувача за Principal. Principal – це користувач, який наразі ввійшов у систему. Отримуємо його через контекст безпеки, який прив'язаний до поточного потоку, і тому він також прив'язаний до поточного запиту та його сеансу.

Метод currencyConversion(int price, String currency) конвертує ціни з USD в інші валюти. Якщо ж валюту не вказано, повертає ціну без змін.

Метод handleImageFiles(Announcement announcement, MultipartFile imageFile1, MultipartFile imageFile2, MultipartFile imageFile3) обробляє та додає зображення до оголошення. Перевіряє, наявність файлу, і якщо файл присутній – конвертує його в об'єкт Image за допомогою методу imageConversion. Перше зображення встановлюється як превью, решта додається як звичайні зображення.

HandleBlogImageFiles(Blog blog, MultipartFile imageFile1, MultipartFile imageFile2, MultipartFile imageFile3) схожий на handleImageFiles, але працює з об'єктом Blog.

За допомогою imageConversion(MultipartFile imageFile) MultipartFile перетворюється на об'єкт типу Image. Потім встановлюються властивості зображення: розмір, ім'я, вихідне ім'я, тип контенту та байтові дані.

Метод updatePreviewImage(Announcement announcement) проходить по всім зображенням оголошення і шукає те, що позначене як прев'ю і вставновлює його як прев'ю для оголошення.

UpdateBlogPreviewImage(Blog blog) аналогічний updatePreviewImage, але для об’єкта Blog.

Метод getUserByPrincipal(Principal principal) повертає об’єкт користувача Principal. Якщо ж Principal дорівнює null, повертає пустий об’єкт User.

У додатку Б, на рисунках Б.8 та Б.9 представлено сервісний **клас UserService**, який надає бізнес-логіку для керування користувачами у застосунку.

Поле passwordEncoder відповідає за хешування паролів перед збереженням їх у базі даних.

Метод createUser(User user) відповідає за створення нового користувача. Спочатку метод перевіряє дані користувача у бд з введеними даними нового користувача, і якщо такі існують – метод поверне статус enum EMAIL_EXISTS або NICKNAME_EXISTS. Потім, новому користувачу присвоюється роль ROLE_USER, генерується код активації, який надсилається на пошту користувача, хешутся пароль. У разі, успішної реєстрації, метод поверне статус SUCCESS.

За допомогою методу banAndUnbanUser(Long id) можна активувати або заблокувати користувача за його ID, змінюючи статус активності. Права на «бан» та розбан користувачів доступні лише адміністраторам.

UserChangeRole(User user, Map<String, String> form) дозволяє змінювати ролі користувача, ґрунтуючись на ролях, зазначених у формі.

Методи subscribe(User userByPrincipal, User user) та unSubscribe(User userByPrincipal, User user) реалізують підписку та відписку від іншого користувача. Методи оновлюють список підписників та зберігають зміни у базі даних.

FindSubscriptions(String type, String nickname, User user) використовується для пошуку користувачів за типом зв’язку з поточним користувачем, тобто якщо type=subscriptions, метод буде шукати серед

підписок користувача, а якщо type= subscribers, метод буду шукати серед підписників. Якщо ж вказано nickname, шукає користувачів, які містять цей нікнейм.

Метод updateProfile(User user, String nickname, String name, String surname, String email, String password) оновлює профіль користувача. Якщо email змінився, метод сгенерує новий активаційний код, який надішле на нову пошту та деактивує користувача, поки той не активує свій акаунт. Також хешує та зберігає новий пароль, якщо він змінився.

ActivateUser(String code) активує користувача за активаційним кодом. Якщо ж код є вірним, видаляє його з бази даних, встановлюючи користувача як активного.

Метод getUsersByNickname(String nickname) шукає користувачів за їх нікнеймом, а метод getAllUsers() повертає список усіх користувачів.

У додатку Б, на рисунок Б.10 представлено сервісний **клас BlogService**, який надає функціонал для роботи з блогами.

Метод List<Blog> filterBlogs(String title) фільтрує блоги за назвою, враховуючи ключові слова.

AddBlog(Principal principal, Blog blog, MultipartFile imageFile1, MultipartFile imageFile2, MultipartFile imageFile3) дозволяє створювати новий блог, включаючи необхідні атрибути, такі як заголовок, контент, зображення ті іншу, зберігаючи усе у БД.

Метод updateBlog(Blog blog, String title, String text, MultipartFile imageFile1, MultipartFile imageFile2, MultipartFile imageFile3) надає можливість оновлювати існуючий блог. Дозволяє змінити його вміст, додати або видалити зображення.

Методи List<Blog> getAllBlogs() та Blog getBlogById(Long id) – геттери, за допомогою яких можна отримати список усіх блогів або конкретний блог за id відповідно.

За допомогою deleteBlog(Long id) можна видаляти блоги.

3.3 Реалізація конфігураційних класів

@Configuration – це анотація на рівні класу, яка вказує на те, що об’єкт є джерелом визначень компонентів. Класи @Configuration оголошують компоненти за допомогою анотованих методів @Bean. Виклики методів @Bean у класах @Configuration також можна використовувати для визначення залежностей між компонентами.

У додатку В, на рисунку В.1, представлено конфігураційний **клас MailConfiguration**, який налаштовує поштовий сервер для надсилання електронної пошти.

Поле host зберігає адресу поштового сервера. У застосунку це smtp.gmail.com.

Поле username містить ім’я користувача за допомогою якого проходить аутентифікація на поштовому сервері.

Поле password зберігає пароль, який необхідний для аутентифікація на поштовому сервері.

Поле port: використовується для з’єднання SMTP.

Метод getMailSender() створює об’єкт JavaMailSenderImpl, який надає функціональність для листів. Після цього, метод конфігурує цей об’єкт:

- setHost() встановлює хост поштового сервера;
- setUsername() задає пошту через яку будуть відправлятись повідомлення;
- setPassword() встановлює пароль до пошти;
- setPort() задає порт підключення.

Після налаштування, через getJavaMailProperties() включається TLS для безпеки передачі даних через поштовий сервер.

Наприкінці метод повертає налаштований екземпляр JavaMailSenderImpl, який буде використовуватися для надсилання листів у застосунку.

У додатку В, на рисунку В.2, представлено конфігураційний **клас ReCaptchaConfiguration**, який необхідний для інтеграції Google

reCAPTCHA.

Поля siteKey та secretKey містять ключі reCAPTCHA. siteKey використовується на клієнтській стороні для відображення reCAPTCHA, а secretKey використовується на серверній стороні для перевірки відповіді клієнта.

Поле RECAPTCHA_VERIFY_URL зберігає URL, яка використовується для надсилання запиту до API Google reCAPTCHA. Вона містить два параметри: secret та response (відповідь від клієнта).

Для перевірки відповіді клієнта на сервері потрібно надіслати запит на https://www.google.com/recaptcha/api/siteverify, передавши secretKey та response. Це використовується для валідації того, що введений код reCAPTCHA був дійсним.

У додатку В, на рисунку В.3, представлено конфігураційний **клас SecurityConfiguration**, який налаштовує параметри безпеки для Spring Boot.

Анотація @EnableWebSecurity вмикає підтримку веббезпеки Spring Security, активуючи конфігурації безпеки.

@EnableMethodSecurity вмикає анотації безпеки на рівні методу, наприклад, @PreAuthorize або @Secured.

Поле customUserDetailsService потрібне для завантаження даних користувача з бази даних для подальшої аутентифікації.

Метод filterChain(HttpSecurity http) налаштовує захист HTTP, вказуючи, які URL-адреси, дозволено переглянути не аутентифікувавшись за допомогою permitAll(). Тобто у застосунку URL /", "/images/**", "/registration", "/details/**", "/filter/**", "/activate/*", "/photos/**", можна переглянути не залогінівшись, а для перегляду усіх інших адрес(.anyRequest()), потрібна аутентифікація(.authenticated()).

.formLogin у методі налаштовує сторінку логіну, вказуючи її адресу(.loginPage("/login")) і дозволяє переглянути цю сторінку усім(.permitAll()).

.logout налаштовує сторінку вже для логауту, вказуючи адресу, на яку перейде користувач у разі виходу з акаунту(.logoutSuccessUrl("/")).

Наприкінці, return http.build() завершує налаштування та створює компонент SecurityFilterChain.

Метод authenticationProvider() створює провайдер для аутентифікації та встановлює customUserDetailsService для отримання інформації про користувача (ім’я користувача, пароль, ролі і т.д.).

Компонент passwordEncoder() потрібен для кодування та декодування паролів.

3.4 Реалізація контролерів

Spring MVC надає модель програмування на основі анотацій, де компоненти @Controller і @RestController використовують анотації для вираження зіставлення запитів, введення запитів, обробки винятків тощо (рис. 3.1).

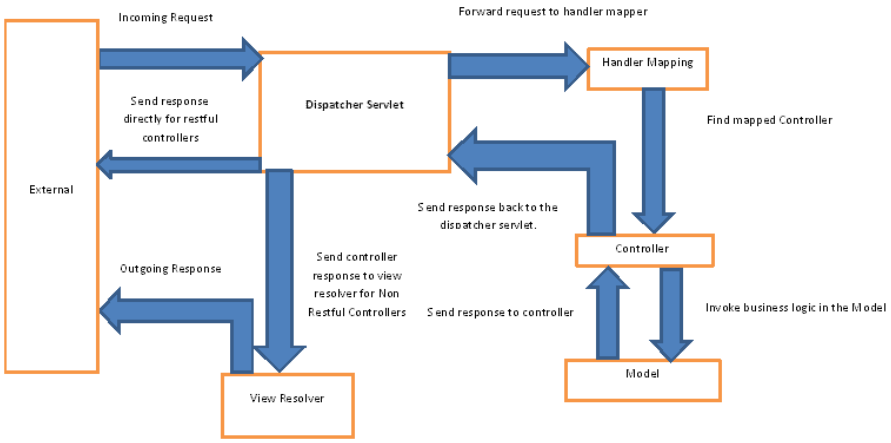


Рисунок 3.1 – Принцип дії контролера у Spring

У додатку Г, на рисунках Г.1 та Г.2, зображено **контролер AnnouncementController** для керування оголошеннями.

Мапінг @GetMapping("/") Відображає головну сторінку з оголошеннями, де можна фільтрувати оголошення за назвою та сортувати їх.

@GetMapping("/details/{id}"): Відображає докладну інформацію про конкретне оголошення, яке знаходиться за допомогою id. Метод

announcementDetails завантажує деталі оголошення та поточного користувача потім відображає сторінку announcement-details.

За допомогою @GetMapping("/{announcement}/edit") можна перейти на сторінку редагування оголошення. Доступ до сторінки обмежений за допомогою @PreAuthorize. Ця анотація перевіряє, чи є користувач власником оголошення, яке він хоче відрегувати.

@PostMapping("/{announcement}/edit") оброблює запит на редагування оголошення.

Мапінг @GetMapping("/filter/{category}") фільтрує оголошення за вказаною категорією та сортує їх в межах категорії, а якщо передано параметр sort. Потім повертає відфільтровані та відсортовані оголошення на сторінку announcement-category.

@GetMapping("/add") відображає форму для створення нового оголошення.

Запит @PostMapping("/add") в свою чергу вже оброблює створення нового оголошення. Він приймає до трьох зображень, об’єкт announcement і Principal.

За допомогою @PostMapping("/delete/{id}") можна видалити оголошення за його id.

Розглянемо шаблони, з якими працює AnnouncementController.

На рисунку 3.2 зображено головну сторінку, повний код шаблону наведено у додатку Д, на рисунку Д.15.

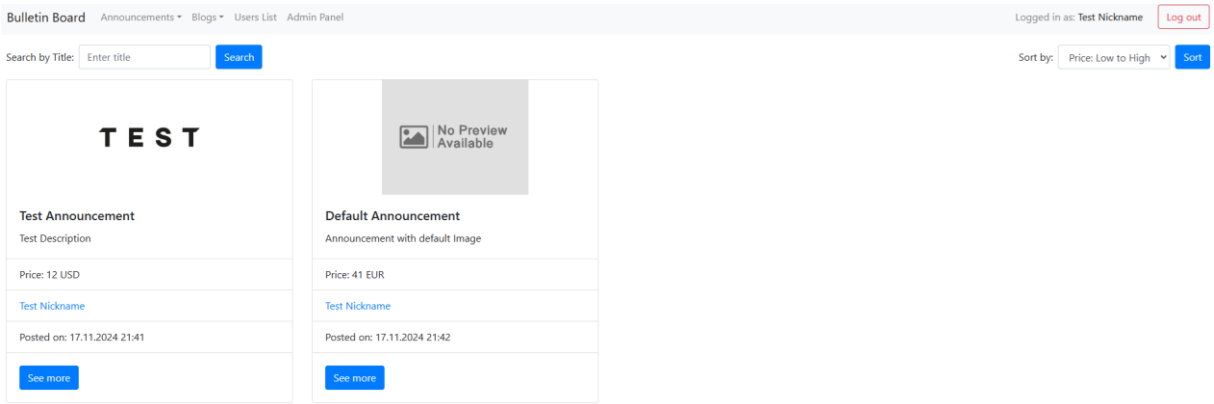


Рисунок 3.2 – Main-page.html

На рисунку 3.3 зображено сторінку з детальним описом оголошення, повний код шаблону наведено у додатку Д, на рисунку Д.7.

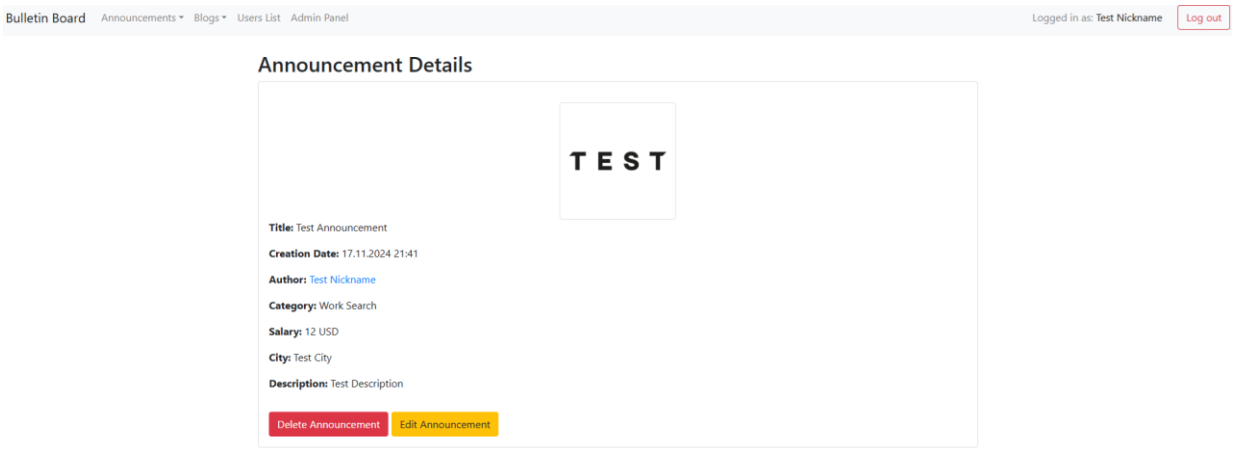


Рисунок 3.3 – Announcement-details.html

На рисунку 3.4 зображено сторінку редагування оголошень, повний код шаблону наведено у додатку Д, на рисунках Д.8 та Д.9.

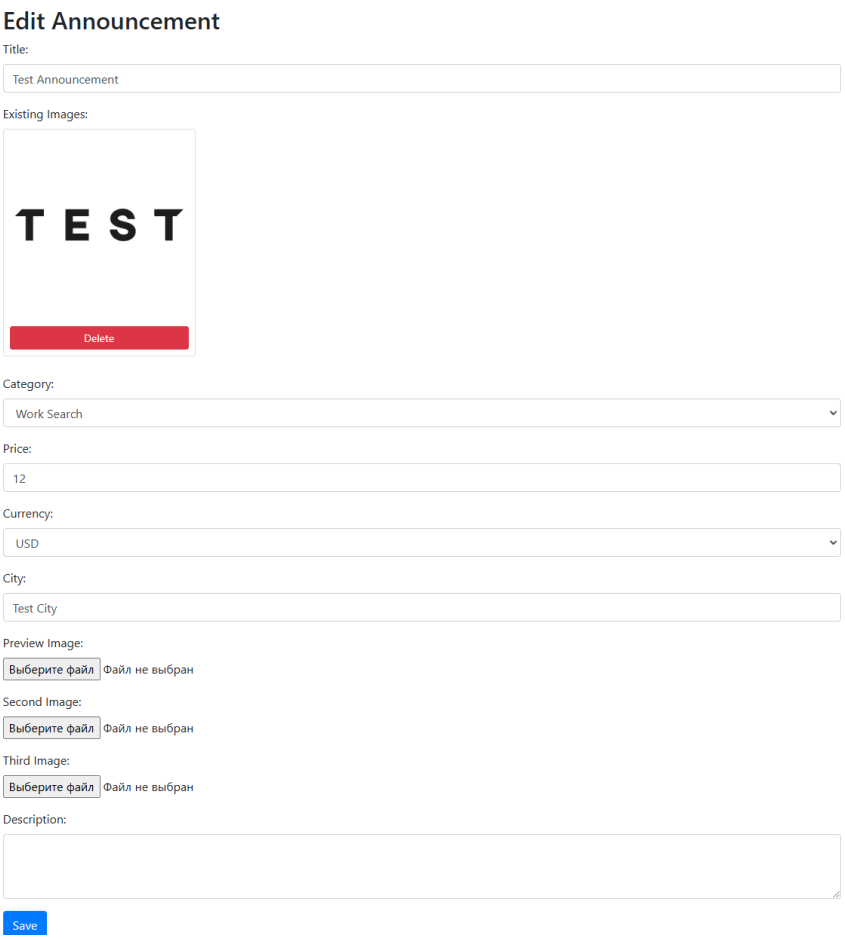


Рисунок 3.4 – Announcement-edit.html

На рисунку 3.5 зображено сторінку з відфільтрованими за категорією оголошення, повний код шаблону наведено у додатку Д, на рисунках Д.5 та Д.6.

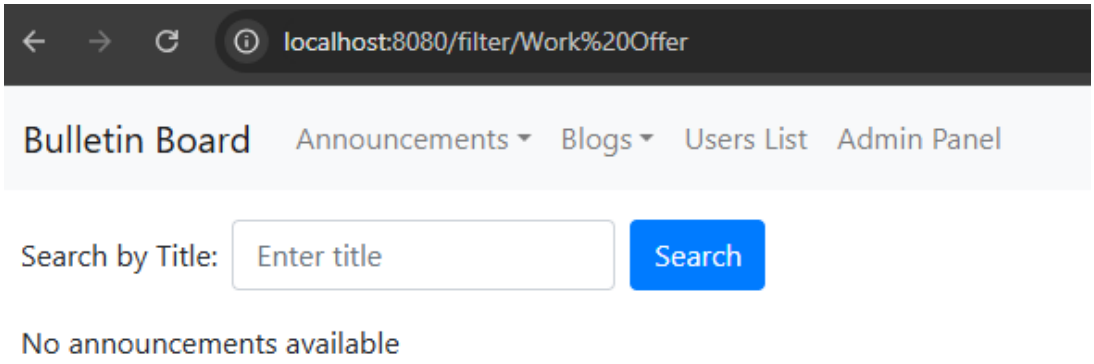


Рисунок 3.5 – Announcement-category.html

На рисунку 3.6 зображено сторінку додавання нових оголошень, повний код шаблону наведено у додатку Д, на рисунках Д.1 та Д.2.

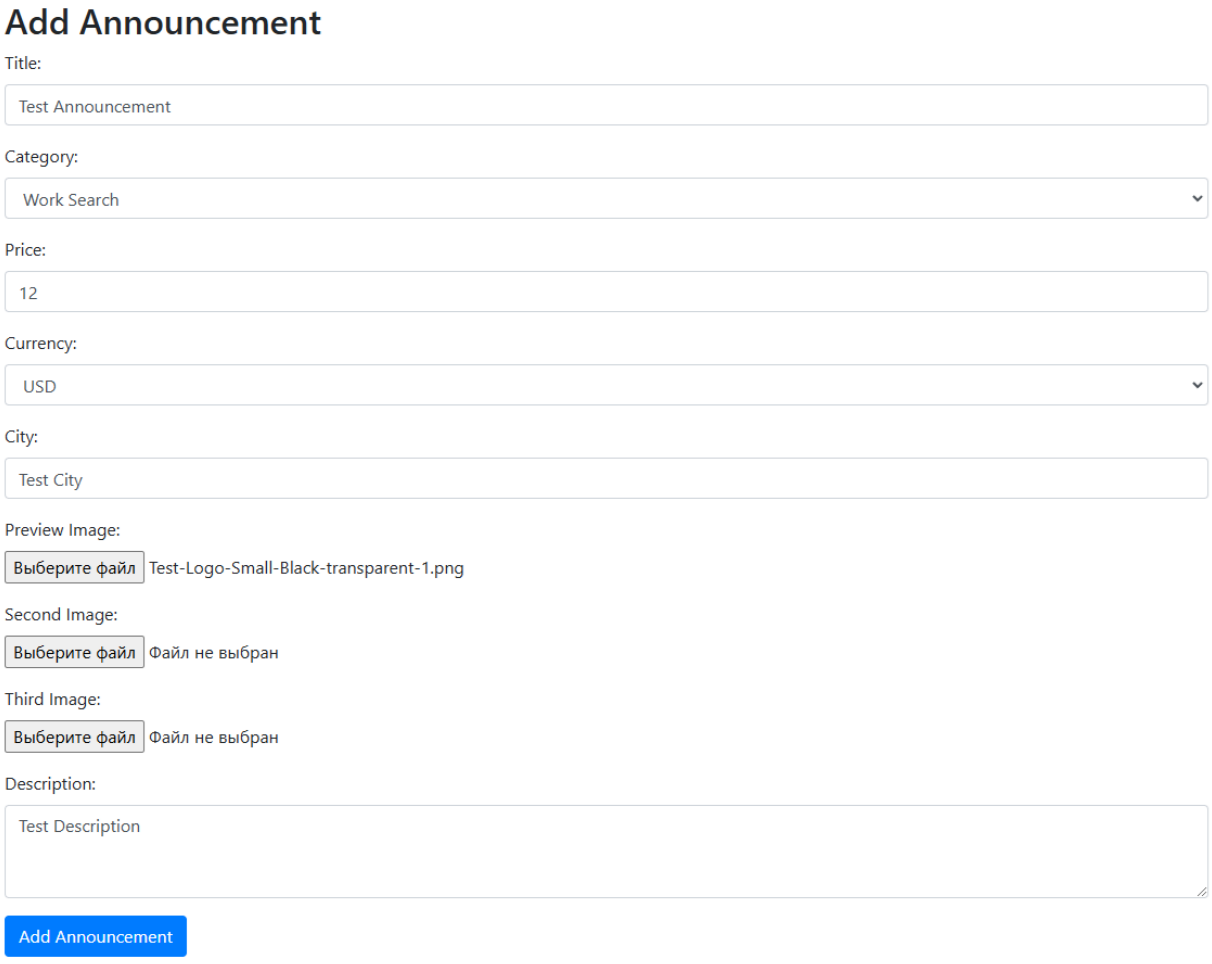


Рисунок 3.6 – Add-announcement.html

У додатку Г, на рисунку Г.3 представлено **AdminController**, який призначений для користувачів з роллю admin.

Мапінг @GetMapping("/admin") відображає панель адміністратора, на якій відображаються усі користувачі.

@PostMapping("/user/ban/{id}") оброблює запити POST для блокування або розблокування користувачів за id. Якщо ж адміністратор вирішить забанити користувача, метод banAndUnbanUser перенаправить його на URL /admin.

За допомогою @GetMapping("/user/edit/{user}") відображається сторінка зміни ролей користувача.

@PostMapping("/user/edit") в свою чергу вже оброблює запити POST для оновлення ролей користувача. Він приймає userId і передає його у userService для оновлення ролей.

Розглянемо шаблони, з якими працює AdminController.

На рисунку 3.7 зображено панель адміністратора, повний код шаблону наведено у додатку Д, на рисунках Д.3 та Д.4.

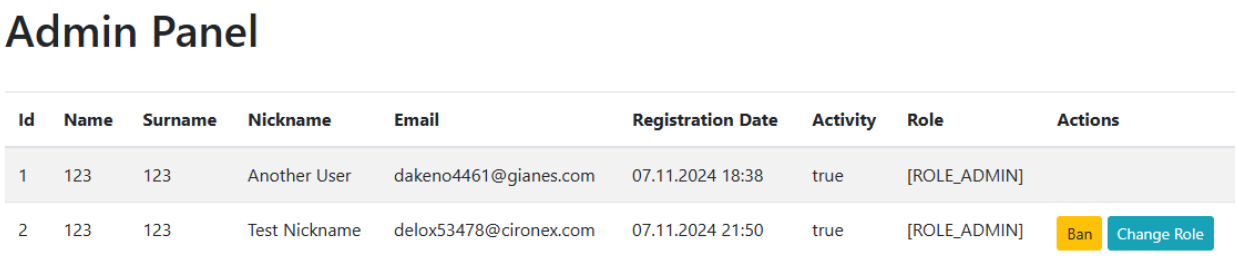


Рисунок 3.7 – Admin-page.html

На рисунку 3.8 зображено сторінку зміни ролей, повний код шаблону наведено у додатку Д, на рисунку Д.17.

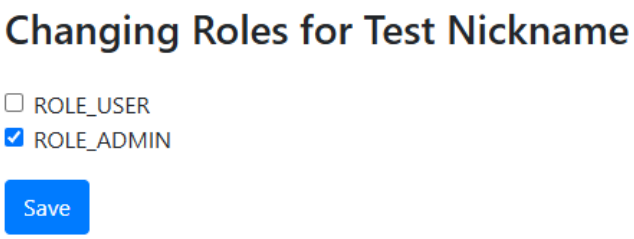


Рисунок 3.8 – User-change-role.html

У додатку Г, на рисунку Г.4 представлено **ImageRestController**, який відповідає за обробку запитів для керування зображеннями, пов’язаними з оголошеннями та блогами.

@GetMapping("/images/{id}") знаходить зображення в imageRepository і, якщо воно існує, створює ResponseEntity з даними зображення(filename, contentType, contentLenghth). А саме зображення передається у тілі відповіді як потік даних типу InputStreamResource.

Метод deleteImage() використовується для видалення зображення, пов’язаного з оголошенням. Метод перевіряє чи наявне зображення за вказаним id, якщо ж воно є, то метод отримує пов’язане оголошення та видаляє зображення зі списку зображень цього оголошення. А якщо зображення було встановлено як прев’ю, то метод встановить поле previewImageId як null. Після цього, зберігає оновлене оголошення та повертає ResponseEntity.ok().

Метод deleteBlogImage() також використовується для видалення зображення, аналогічно deleteImage(), але для сутності blog.

У додатку Г, на рисунках Г.5 та Г.6 представлено **UserController**, який відповідає за обробку запитів, пов’язаних з користувачами, наприклад, реєстрація, аутентифікаця, керування профілем та багато іншого.

Запити @GetMapping("/login") та @GetMapping("/registration") повертають сторінки логіну та реєстрації відповідно.

@PostMapping("/registration") відповідає за реєстрацію нового користувача з перевіркою reCAPTCHA. Якщо перевірка reCAPTCHA не пройдена, запит поверне повідомлення про помилку на сторінку реєстрації. Якщо ж реєстрація є успішною, перенаправляє користувача на сторінку логіна.

Запит @GetMapping("/user/{user}") відображає сторінку «профілю» користувача. На якій можна знайти його оголошення, кількість аідписників та підписок.

За допомогою запитів @GetMapping("/subscribe/{user}") та @GetMapping("/unsubscribe/{user}") можна підписатись або навпаки

відписатись від конкретного користувача.

Запит @GetMapping("/{type}/{user}/list") в залежності від типу відобразить кількість підписок або підписників користувача.

@GetMapping("/users/list") відображає список усіх користувачів. Також, на сторінці є можливість пошуку користувачів за нікнймом, якщо він вказан.

@GetMapping("/user/{user}/edit") повертає сторінку на якій можно редагувати акаунт. А @PostMapping("/user/{user}/edit") відповідає вже за обробку форми редагування. За допомогою анотації @PreAuthorize("#user.email.equals(authentication.principal.email)") можна обмежити доступ всім користувачам окрім власника профілю.

@GetMapping("/activate/{code}") відповідає за активацію акаунта користувача за кодом активації.

Розглянемо шаблони, з якими працює UserController.

На рисунку 3.9 зображено сторінку логіну, повний код шаблону наведено у додатку Д, на рисунку Д.13.

На рисунку 3.10 зображено сторінку реєстрації користувачів, повний код шаблону наведено у додатку Д, на рисунку Д.14.

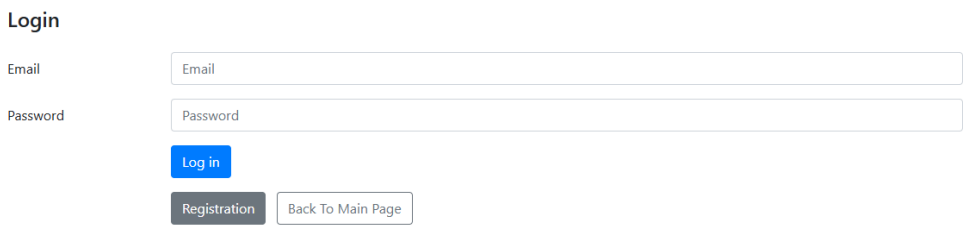


Рисунок 3.9 – Login-page.html

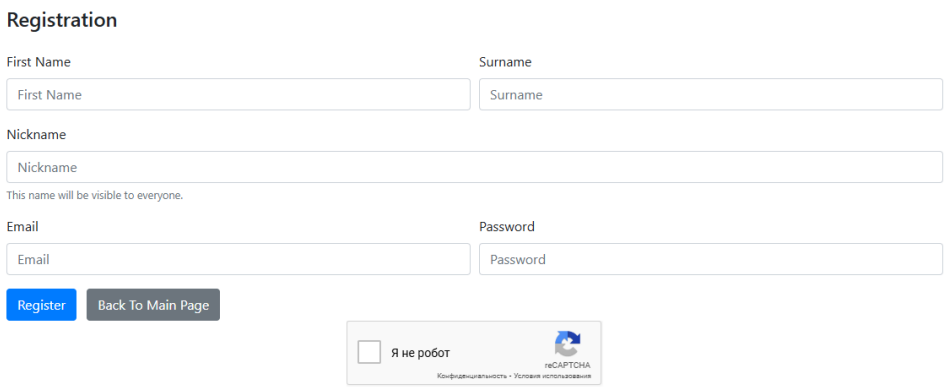


Рисунок 3.10 – Registration-page.html

На рисунку 3.11 зображено сторінку профілю користувача, повний код шаблону наведено у додатку Д, на рисунках Д.20 та Д.21.

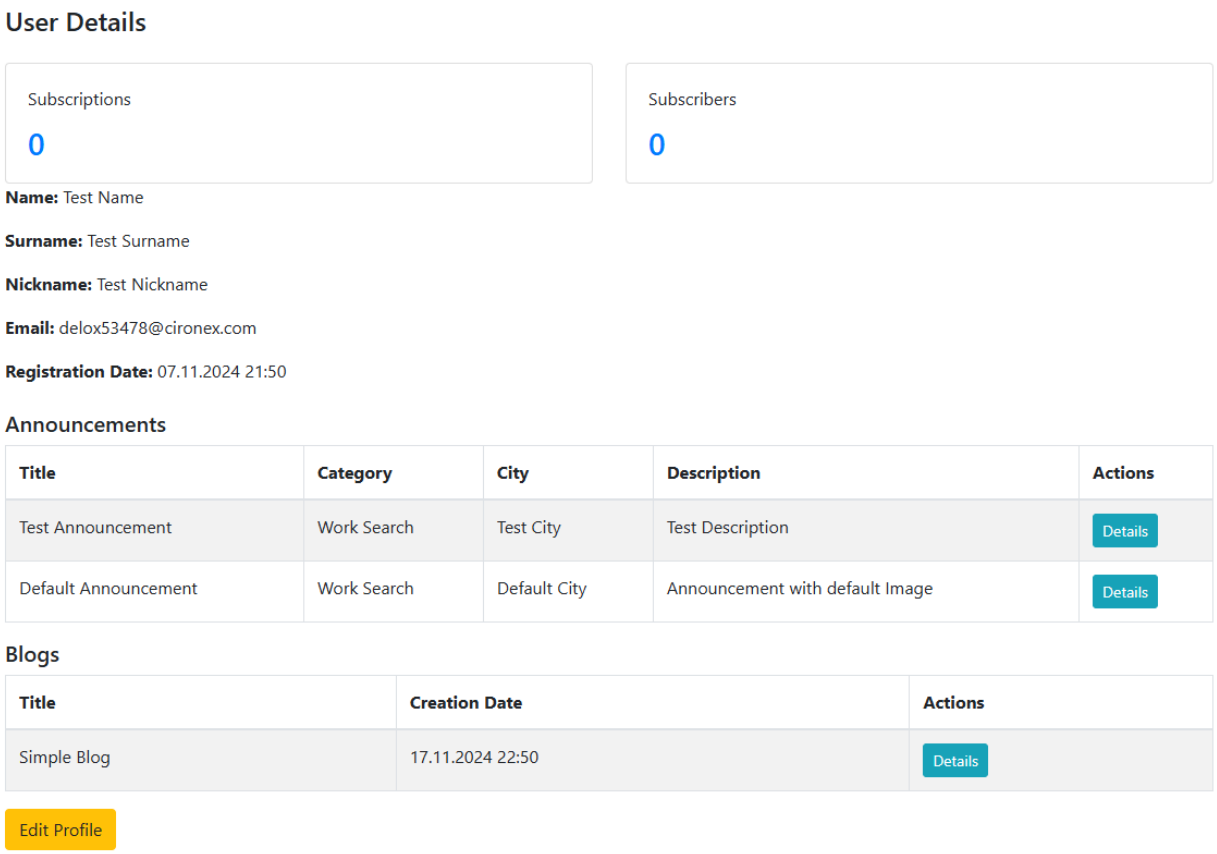


Рисунок 3.11 – User-details.html

На рисунку 3.12 зображено сторінку, яка в залежності від type відображає кількість підписників або підписок, код шаблону наведено у додатку Д, на рисунку Д.16.

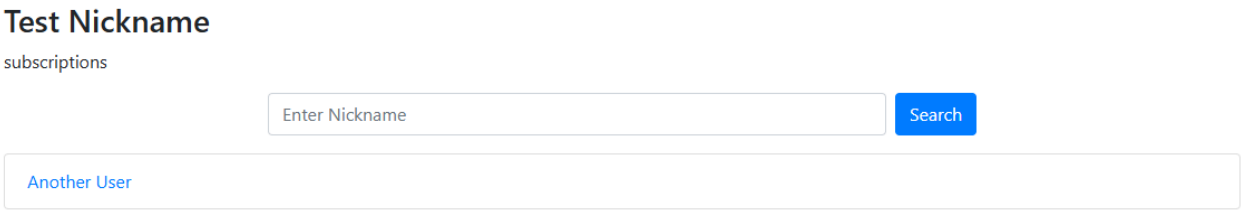


Рисунок 3.12 – Subscriptions-list.html

На рисунку 3.13 зображено сторінку з усіма зареєстрованими користувачами, код шаблону наведено у додатку Д, на рисунку Д.19.

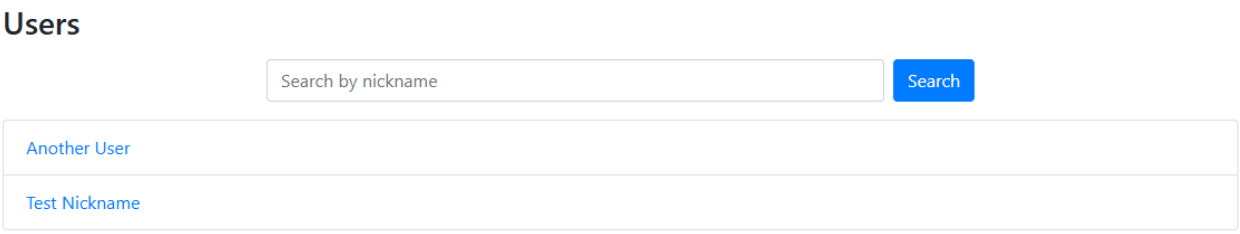


Рисунок 3.13 – User-list.html

На рисунку 3.14 зображено форму редагування акаунту, код шаблону наведено у додатку Д, на рисунку Д.18.



Рисунок 3.14 – User-edit.html

У додатку Г, на рисунках Г.7 представлено **BlogController**, який відповідає за керування блогами, включаючи створення, редагування, перегляд та видалення.

@GetMapping("/blogs") відображає сторінку з усіма блогами. На сторінці є можливість пошуку блогів за назвою.

За допомогою @GetMapping("/blog/details/{id}") користувач може редагувати блоги, які сам запостив.

Запит @GetMapping("/addBlog") відображає сторінку з формою для створення нового блогу, а @PostMapping("/addBlog") в свою ж чергу відповідає за створення нового блогу.

@GetMapping("/blog/{blog}/edit") відображає сторінку для редагування блогу, з обмеженим за допомогою анотації @PreAuthorize доступом. @PostMapping("/blog/{blog}/edit") відповідає за саме редагування. Він приймає параметри для оновлення та передає їх у сервісний клас.

@PostMapping("/blog/delete/{id}") видаляє блог за id.

Розглянемо шаблони, з якими працює BlogController.

На рисунку 3.15 зображено сторінку з списком усіх блогів, код шаблону наведено у додатку Д, на рисунку Д.22.

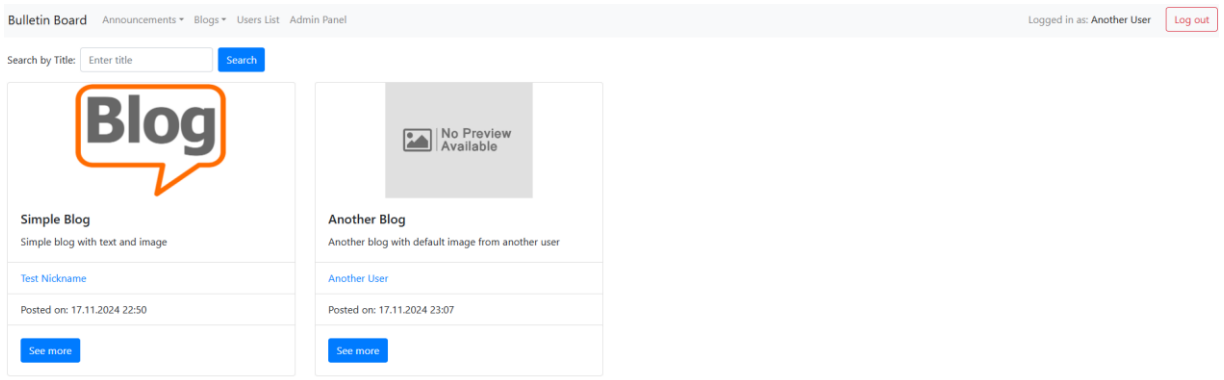


Рисунок 3.15 – Blogs-page.html

На рисунку 3.16 зображено сторінку з інформацією про конкретний блог, код шаблону наведено у додатку Д, на рисунку Д.11.



Рисунок 3.16 – Blog-details.html

На рисунку 3.17 зображено сторінку для створення нового блогу, код шаблону наведено у додатку Д, на рисунку Д.10.

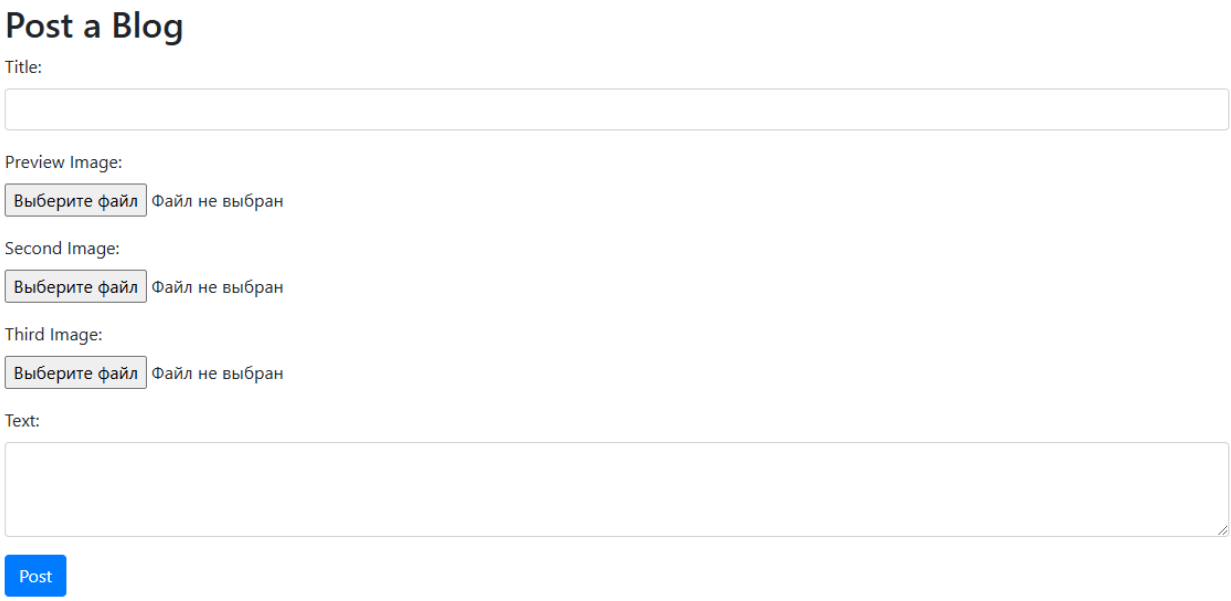


Рисунок 3.17 – Add-blog.html

На рисунку 3.18 зображено сторінку для редагування блогу, код шаблону наведено у додатку Д, на рисунку Д.12.

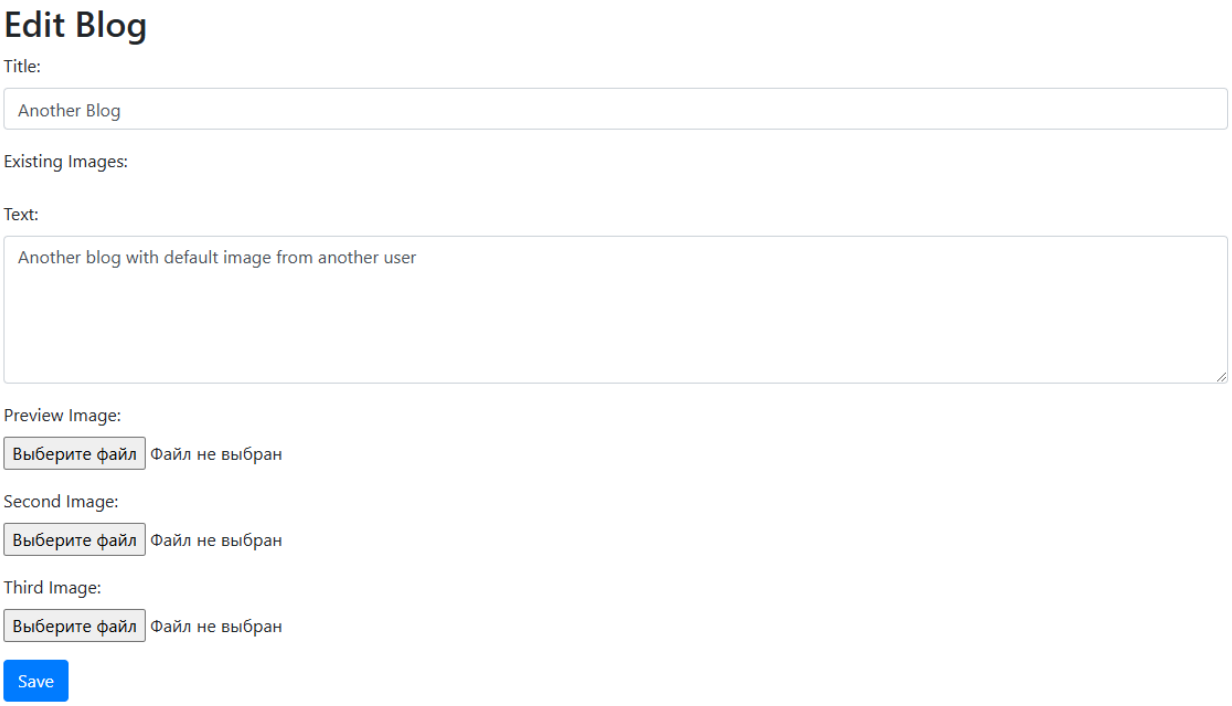


Рисунок 3.18 – Blog-edit.html

ВИСНОВКИ

Завдяки сучасним технологіям вдалося реалізувати визначені функції.

Розробка зручного інтерфейсу, за допомогою якого користувач може розміщувати оголошення та блоги з зображеннями вдалося реалізувати за допомогою Bootstrap та Thymeleaf. Також, у застосунку були реалізовані методи пошуку та фільтраціїї, за допомогою яких користувач може швидко знаходити інтересуючі його блоги або оголошення.

Сам по собі Bootstrap дуже сильний інструмент, за допомогою якого вдалося створити інтуїтивно зрозумілий інтерфейс як для нових так і досвідчених користувачів. Але використання Bootstrap у застосунку сильно обмежує візуальну кастомізацію.

Інтегрувати базу даних для збереження інформації вдалося за допомогою MySQL. Завдяки цій базі даних, додаток отримав перевагу у вигляді гнучкості та масштабованості, що сильно спрощує інтеграцію нових функцій у майбутньому.

MySQL добре підходить для цього застосунку, але для великих навантажень і значного розширення кількості користувачів можуть виникнути обмеження, які доведеться вирішувати через додаткові оптимізації або зміну бази даних.

Забезпечити безпеку та аутентифікацію користувачів стало можливим за допомогою Spring Security.

Проєкт реалізовано із застосуванням найкращих практик у сфері веброзробки, що забезпечило високу якість і функціональність.

ПЕРЕЛІК ПОСИЛАНЬ

1. Rahul Awati, Ivy Wigmore. What is monolithic architecture in software? *TechTarget*. URL: https://www.techtarget.com/whatis/definition/monolithic-architecture (дата звернення: 11.09.2024).

2. What are Microservices? *Amazon*. URL: https://aws.amazon.com/microservices/?nc1=h_ls (дата звернення: 13.09.2024).

3. Introduction to the Spring Framework. *Spring.io*. URL: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html (дата звернення: 15.09.2024).

4. Spring Security. *Spring.io*. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення: 16.09.2024).

5. Tutorial: Using Thymeleaf. *Thymeleaf*. URL: https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html#contexts (дата звернення: 18.09.2024).

6. Hibernate ORM User Guide. *JBoss*. URL: https://docs.jboss.org/hibernate/orm/6.3/userguide/html_single/Hibernate_User_Guide.html#architecture (дата звернення: 22.09.2024).

7. Ayush Singh. Spring Data JPA and JPQL. *Medium*. URL: https://medium.com/@aprayush20/spring-data-jpa-with-spring-boot-9e3b261e4cc7 (дата звернення: 23.09.2024).

ДОДАТОК А

Класи для роботи з даними та сховищем даних

```
@Getter  29 usages  Tkachenko Vadym *
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "announcements")
public class Announcement {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;
    @Column(name = "title")
    private String title;
    @Column(name = "category")
    private String category;
    @Column(name = "price")
    private int price;
    @Column(name = "currency")
    private String currency;
    @Column(name = "city")
    private String city;
    @Column(name = "description", columnDefinition = "text")
    private String description;
    @Column(name = "creation_Date")
    private LocalDateTime creationDate;
    @Column(name = "preview_Image_Id")
    private Long previewImageId;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "announcement")
    private List<Image> images = new ArrayList<>();
    @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.LAZY)
    @JoinColumn
    private User user;

    @PrePersist  Tkachenko Vadym
    private void init() {
        creationDate = LocalDateTime.now();
    }

    public void addAnnouncementImage(Image image) {  3 usages  Tkachenko Vadym
        image.setAnnouncement(this);
        images.add(image);
    }
}
```

Рисунок А.1 – Сутність Announcement

```java
@Getter  26 usages  Tkachenko Vadym *
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "images")
public class Image {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;
    @Column(name = "image_Size")
    private Long imageSize;
    @Column(name = "image_Name")
    private String imageName;
    @Column(name = "original_Image_Name")
    private String originalImageName;
    @Column(name = "content_Type")
    private String contentType;
    @Column(name = "is_Preview_Image")
    private boolean isPreviewImage;
    @Column(name = "bytes", columnDefinition = "LONGBLOB")
    private byte[] bytes;
    @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.EAGER)
    private Announcement announcement;
    @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.EAGER)
    private Blog blog;
}
```

Рисунок А.2 – Сутність Image

```java
@Getter  46 usages  Tkachenko Vadym +1 *
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;
    @Column(name = "name")
    private String name;
    @Column(name = "surname")
    private String surname;
    @Column(name = "nickname")
    private String nickname;
    @Column(name = "email", unique = true)
    private String email;
    @Column(name = "activation_Code")
    private String activationCode;
    @Column(name = "password")
    private String password;
    @Column(name = "registration_Date")
    private LocalDateTime registrationDate;
    @Column(name = "is_Active")
    private boolean isActive;
    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "image_id")
    private Image profileImage;
    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles = new HashSet<>();
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
    private List<Announcement> announcements = new ArrayList<>();
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
    private List<Blog> blogs = new ArrayList<>();
    @ManyToMany
    @JoinTable(name = "user_subscriptions", joinColumns = {@JoinColumn(name = "channel_id")},
            inverseJoinColumns = {@JoinColumn(name = "subscriber_id")})
    private Set<User> subscribers = new HashSet<>();
    @ManyToMany
    @JoinTable(name = "user_subscriptions", joinColumns = {@JoinColumn(name = "subscriber_id")},
            inverseJoinColumns = {@JoinColumn(name = "channel_id")})
    private Set<User> subscriptions = new HashSet<>();

    @PrePersist  Tkachenko Vadym
    private void init() { registrationDate = LocalDateTime.now(); }

    public boolean isAdmin() { return roles.contains(Role.ROLE_ADMIN); }

    @Override  Tkachenko Vadym
    public Collection<? extends GrantedAuthority> getAuthorities() { return roles; }

    @Override  Tkachenko Vadym
    public String getUsername() { return email; }

    @Override  Tkachenko Vadym
    public boolean isAccountNonExpired() { return true; }

    @Override  9 usages  Tkachenko Vadym
    public boolean isAccountNonLocked() { return true; }

    @Override  9 usages  Tkachenko Vadym
    public boolean isCredentialsNonExpired() { return true; }

    @Override  Tkachenko Vadym
    public boolean isEnabled() { return isActive; }
}
```

Рисунок А.3 – Сутність User

```java
@Getter  20 usages  new *
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "blogs")
public class Blog {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;
    @Column(name = "title")
    private String title;
    @Column(name = "text", columnDefinition = "text")
    private String text;
    @Column(name = "creation_Date")
    private LocalDateTime creationDate;
    @Column(name = "preview_Image_Id")
    private Long previewImageId;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "blog")
    private List<Image> images = new ArrayList<>();
    @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.LAZY)
    @JoinColumn
    private User user;

    @PrePersist  new *
    private void init() {
        creationDate = LocalDateTime.now();
    }

    public void addBlogImage(Image image) {  3 usages  new *
        image.setBlog(this);
        images.add(image);
    }
}
```

Рисунок А.4 – Сутність Blog

```java
import com.vadym.board.models.Image;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ImageRepository extends JpaRepository<Image, Long> {  2 usages  Tkachenko Vadym *
}
```

Рисунок А.5 – Репозиторій ImageRepository

```
import com.vadym.board.models.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface UserRepository extends JpaRepository<User, Long> {  6 usages  Tkachenko Vadym +1
    User findByEmail(String email);  3 usages  Tkachenko Vadym
    User findByActivationCode(String code);  1 usage  Tkachenko Vadym
    User findByNickname(String nickname);  1 usage  Kabratigr\Vadidya
    List<User> findByNicknameContainingIgnoreCase(String nickname);  1 usage  Kabratigr\Vadidya
}
```

Рисунок А.6 – Репозиторій UserRepository

```
import com.vadym.board.models.Blog;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface BlogRepository extends JpaRepository<Blog, Long> {  4 usages  new *
    List<Blog> findByTitleContainingIgnoreCase(String title);  1 usage  new *
}
```

Рисунок А.7 – Репозиторій BlogRepository

```
import com.vadym.board.models.Announcement;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface AnnouncementRepository extends JpaRepository<Announcement, Long> {  4 usages  Tkachenko Vadym
    List<Announcement> findByTitleContainingIgnoreCaseAndCategory(String title, String category);  1 usage  Tkachenko Vadym
    List<Announcement> findByTitleContainingIgnoreCase(String title);  1 usage  Tkachenko Vadym
    List<Announcement> findByCategory(String category);  2 usages  Tkachenko Vadym
}
```

Рисунок А.8 – Репозиторій AnnouncementRepository

ДОДАТОК Б

Сервісні класи

```
@Service  2 usages  Tkachenko Vadym +1 *
@RequiredArgsConstructor
public class AnnouncementService {

    private final AnnouncementRepository announcementRepository;

    private final UtilityService utilityService;

    private final MailSenderService mailSenderService;

    public List<Announcement> filterByTitle(String title) {  1 usage  Tkachenko Vadym
        if (title != null && !title.isEmpty()) {
            List<String> searchByKeyWords = Arrays.asList(title.split( regex: "\\s+"));
            return searchByKeyWords.stream()  Stream<String>
                    .map(keyword -> announcementRepository.findByTitleContainingIgnoreCase(title))  Stream<List<...>>
                    .flatMap(List::stream)  Stream<Announcement>
                    .collect(Collectors.toList());
        }
        return announcementRepository.findAll();
    }

    public List<Announcement> filterOnCategoryPage(String title, String category) {  1 usage  Tkachenko Vadym
        if (!title.isEmpty() && !category.isEmpty()) {
            List<String> searchByKeyWords = Arrays.asList(title.split( regex: "\\s+"));
            return searchByKeyWords.stream()  Stream<String>
                    .map(keyword -> announcementRepository.findByTitleContainingIgnoreCaseAndCategory(title, category))  Stream<List<...>>
                    .flatMap(List::stream)  Stream<Announcement>
                    .collect(Collectors.toList());
        }
        return announcementRepository.findByCategory(category);
    }

    public List<Announcement> sortAnnouncements(List<Announcement> announcements, String sortOrder) {  2 usages  Tkachenko Vadym
        announcements.sort((announcement1, announcement2) -> {
            if (sortOrder.equals("priceAscending") || sortOrder.equals("priceDescending")) {
                double price1 = utilityService.currencyConversion(announcement1.getPrice(), announcement1.getCurrency());
                double price2 = utilityService.currencyConversion(announcement2.getPrice(), announcement2.getCurrency());
                return sortOrder.equals("priceAscending") ? Double.compare(price1, price2) : Double.compare(price2, price1);
            } else if (sortOrder.equals("dateAscending") || sortOrder.equals("dateDescending")) {
                LocalDateTime dateTime1 = announcement1.getCreationDate();
                LocalDateTime dateTime2 = announcement2.getCreationDate();
                return sortOrder.equals("dateAscending") ? dateTime1.compareTo(dateTime2) : dateTime2.compareTo(dateTime1);
            }
            return 0;
        });
        return announcements;
    }
```

Рисунок Б.1 – Сервісний клас AnnouncementService

```
public void addAnnouncement(Principal principal,  1 usage  ± Tkachenko Vadym +1 *
                            Announcement announcement,
                            MultipartFile imageFile1,
                            MultipartFile imageFile2,
                            MultipartFile imageFile3,
                            boolean notifySubscribers) throws IOException {
    announcement.setUser(utilityService.getUserByPrincipal(principal));
    utilityService.handleImageFiles(announcement, imageFile1, imageFile2, imageFile3);
    Announcement announcementFromDb = announcementRepository.save(announcement);
    utilityService.updatePreviewImage(announcementFromDb);
    announcementRepository.save(announcement);
    if (notifySubscribers) {
        mailSenderService.notifySubscribers(announcementFromDb);
    }
}

public void updateAnnouncement(Announcement announcement, String title, String category, int price, String currency,  1 usage  ± Tkachenko Vadym +1 *
                               String city, String description,  MultipartFile imageFile1, MultipartFile imageFile2,
                               MultipartFile imageFile3) throws IOException {
    if (!announcement.getTitle().equals(title) && !StringUtils.isEmpty(title)) {
        announcement.setTitle(title);
    }
    if (!announcement.getCategory().equals(category) && !StringUtils.isEmpty(category)) {
        announcement.setCategory(category);
    }
    if (announcement.getPrice() != price) {
        announcement.setPrice(price);
    }
    if (!announcement.getCurrency().equals(currency) && !StringUtils.isEmpty(currency)) {
        announcement.setCurrency(currency);
    }
    if (!announcement.getCity().equals(city) && !StringUtils.isEmpty(city)) {
        announcement.setCity(city);
    }
    if (!announcement.getDescription().equals(description) && !StringUtils.isEmpty(description)) {
        announcement.setDescription(description);
    }
    utilityService.handleImageFiles(announcement, imageFile1, imageFile2, imageFile3);
    announcementRepository.save(announcement);
    utilityService.updatePreviewImage(announcement);
    announcementRepository.save(announcement);
}

public void deleteAnnouncement(Long id) {  1 usage  ± Tkachenko Vadym
    announcementRepository.deleteById(id);
}

public Announcement getAnnouncementById(Long id) {  2 usages  ± Tkachenko Vadym
    return announcementRepository.findById(id).orElse( other: null);
}

public List<Announcement> getAnnouncementsByCategory(String category) {  2 usages  ± Tkachenko Vadym
    return announcementRepository.findByCategory(category);
}

public List<Announcement> getAllAnnouncements() {  1 usage  ± Tkachenko Vadym
    return announcementRepository.findAll();
}
```

Рисунок Б.2 – Сервісний клас AnnouncementService

```java
@Service  Tkachenko Vadym
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override  Tkachenko Vadym
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email);
        if (user == null) {
            throw new UsernameNotFoundException("User with " + email + " not found");
        }
        return user;
    }
}
```

Рисунок Б.3 – Сервісний клас CustomUserDetailsService

```java
@Service  2 usages  Kabratigr\Vadidya
@RequiredArgsConstructor
public class ReCaptchaService {

    private final ReCaptchaConfiguration reCaptchaConfiguration;

    public boolean validateRecaptcha(String response) {  1 usage  Kabratigr\Vadidya
        String url = String.format(reCaptchaConfiguration.getRECAPTCHA_VERIFY_URL(), reCaptchaConfiguration.getSecretKey(), response);
        RestTemplate restTemplate = new RestTemplate();
        Map<String, String> request = new HashMap<>();
        request.put("response", response);
        Map<String, Object> result = restTemplate.postForObject(url, request, Map.class);
        return Boolean.TRUE.equals(result.get("success"));
    }
}
```

Рисунок Б.4 – Сервісний клас ReCaptchaService

```java
@Service  2 usages  Tkachenko Vadym +1
@RequiredArgsConstructor
public class MailSenderService {

    private final JavaMailSender javaMailSender;

    @Value("${spring.mail.username}")
    private String username;

    public void sendMessage(String sendTo, String subject, String message) {  2 usages  Tkachenko Vadym
        SimpleMailMessage mailMessage = new SimpleMailMessage();
        mailMessage.setFrom(username);
        mailMessage.setTo(sendTo);
        mailMessage.setSubject(subject);
        mailMessage.setText(message);
        javaMailSender.send(mailMessage);
    }

    public void sendActivationCode(User user) {  2 usages  Tkachenko Vadym
        if (!StringUtils.isEmpty(user.getEmail())) {
            String message = String.format(
                    "Hello, %s! \n" +
                            "To activate your account, visit next link: http://localhost:8080/activate/%s",
                    user.getName(),
                    user.getActivationCode()
            );
            sendMessage(user.getEmail(),  subject: "Activation Code", message);
        }
    }

    public void notifySubscribers(Announcement announcement) {  1 usage  Kabratigr\Vadidya
        User announcingUser = announcement.getUser();
        Set<User> subscribers = announcingUser.getSubscribers();

        for (User subscriber : subscribers) {
            String subject = "New Announcement from " + announcingUser.getName() + " " + announcingUser.getSurname();
            String message = String.format(
                    "Hello, %s!\n\n%s has added a new announcement: \n\nVisit http://localhost:8080 to view it.",
                    subscriber.getName(),
                    announcingUser.getName()
            );
            sendMessage(subscriber.getEmail(), subject, message);
        }
    }
}
```

Рисунок Б.5 – Сервісний клас MailSenderService

```java
@Service  10 usages  Tkachenko Vadym *
@RequiredArgsConstructor
public class UtilityService {

    private final UserRepository userRepository;

    public double currencyConversion(int price, String currency) {  2 usages  Tkachenko Vadym
        return switch (currency) {
            case "EUR" -> price * 1.06;
            case "UAH" -> price * 0.025;
            default -> price;
        };
    }

    public void handleImageFiles(Announcement announcement,  2 usages  Tkachenko Vadym
                                 MultipartFile imageFile1,
                                 MultipartFile imageFile2,
                                 MultipartFile imageFile3) throws IOException {
        if (imageFile1.getSize() != 0) {
            Image image1 = imageConversion(imageFile1);
            image1.setPreviewImage(true);
            announcement.addAnnouncementImage(image1);
        }
        if (imageFile2.getSize() != 0) {
            Image image2 = imageConversion(imageFile2);
            announcement.addAnnouncementImage(image2);
        }
        if (imageFile3.getSize() != 0) {
            Image image3 = imageConversion(imageFile3);
            announcement.addAnnouncementImage(image3);
        }
    }

    public void handleBlogImageFiles(Blog blog,  2 usages  new *
                                     MultipartFile imageFile1,
                                     MultipartFile imageFile2,
                                     MultipartFile imageFile3) throws IOException {
        if (imageFile1.getSize() != 0) {
            Image image1 = imageConversion(imageFile1);
            image1.setPreviewImage(true);
            blog.addBlogImage(image1);
        }
        if (imageFile2.getSize() != 0) {
            Image image2 = imageConversion(imageFile2);
            blog.addBlogImage(image2);
        }
        if (imageFile3.getSize() != 0) {
            Image image3 = imageConversion(imageFile3);
            blog.addBlogImage(image3);
        }
    }
```

Рисунок Б.6 – Сервісний клас UtilityService

```
    public Image imageConversion(MultipartFile imageFile) throws IOException {  6 usages   Tkachenko Vadym
        Image image = new Image();
        image.setImageSize(imageFile.getSize());
        image.setImageName(imageFile.getName());
        image.setOriginalImageName(imageFile.getOriginalFilename());
        image.setContentType(imageFile.getContentType());
        image.setBytes(imageFile.getBytes());
        return image;
    }

    public void updatePreviewImage(Announcement announcement) {  2 usages   new *
        for (Image image : announcement.getImages()) {
            if (image.isPreviewImage()) {
                announcement.setPreviewImageId(image.getId());
            }
        }
    }

    public void updateBlogPreviewImage(Blog blog) {  2 usages   new *
        for (Image image : blog.getImages()) {
            if (image.isPreviewImage()) {
                blog.setPreviewImageId(image.getId());
            }
        }
    }

    public User getUserByPrincipal(Principal principal) {  19 usages   Tkachenko Vadym
        if (principal == null) {
            return new User();
        }
        return userRepository.findByEmail(principal.getName());
    }
}
```

Рисунок Б.7 – Сервісний клас UtilityService

```java
@Service  4 usages  Tkachenko Vadym +1
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    private final PasswordEncoder passwordEncoder;

    private final MailSenderService mailSenderService;

    public UserStatus createUser(User user) {  1 usage  Tkachenko Vadym +1
        if (userRepository.findByEmail(user.getEmail()) != null) return UserStatus.EMAIL_EXISTS;
        if (userRepository.findByNickname(user.getNickname()) != null) return UserStatus.NICKNAME_EXISTS;
        user.setActive(false);
        user.getRoles().add(Role.ROLE_ADMIN);
        user.setActivationCode(UUID.randomUUID().toString());
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        mailSenderService.sendActivationCode(user);
        return UserStatus.SUCCESS;
    }

    public void banAndUnbanUser(Long id) {  1 usage  Tkachenko Vadym
        User user = userRepository.findById(id).orElse( other: null);
        if (user != null) {
            user.setActive(!user.isActive());
        }
        userRepository.save(user);
    }

    public void userChangeRole(User user, Map<String, String> form) {  1 usage  Tkachenko Vadym
        Set<String> roles = Arrays.stream(Role.values()) Stream<Role>
                .map(Role::name) Stream<String>
                .collect(Collectors.toSet());
        user.getRoles().clear();
        for (String role : form.keySet()) {
            if (roles.contains(role)) {
                user.getRoles().add(Role.valueOf(role));
            }
        }
        userRepository.save(user);
    }

    public void subscribe(User userByPrincipal, User user) {  1 usage  Kabratigr\Vadidya
        user.getSubscribers().add(userByPrincipal);
        userRepository.save(user);
    }

    public void unSubscribe(User userByPrincipal, User user) {  1 usage  Kabratigr\Vadidya
        user.getSubscribers().remove(userByPrincipal);
        userRepository.save(user);
    }
```

Рисунок Б.8 – Сервісний клас UserService

```java
public List<User> findSubscriptions(String type, String nickname, User user) {  1 usage  ± Kabratigr\Vadidya
    if (nickname != null && !nickname.isEmpty()) {
        List<String> searchByKeyWords = Arrays.asList(nickname.split( regex: "\\s+"));
        Stream<User> userStream;

        if ("subscriptions".equals(type)) {
            userStream = user.getSubscriptions().stream();
        } else if ("subscribers".equals(type)) {
            userStream = user.getSubscribers().stream();
        } else {
            return Collections.emptyList();
        }
        return userStream
                .filter(u -> searchByKeyWords.stream().anyMatch(keyword ->
                        u.getNickname().toLowerCase().contains(keyword.toLowerCase())))
                .collect(Collectors.toList());
    }
    return "subscriptions".equals(type) ? new ArrayList<>(user.getSubscriptions()) : new ArrayList<>(user.getSubscribers());
}

public void updateProfile(User user, String nickname, String name, String surname, String email, String password) {  1 usage  ± Tkachenko Vadym +1
    if (!user.getName().equals(name) && !StringUtils.isEmpty(name)) {
        user.setName(name);
    }
    if (!user.getSurname().equals(surname) && !StringUtils.isEmpty(surname)) {
        user.setSurname(surname);
    }
    if (!user.getNickname().equals(nickname) && !StringUtils.isEmpty(nickname)) {
        user.setNickname(nickname);
    }
    if (!user.getEmail().equals(email) && !StringUtils.isEmpty(email)) {
        user.setEmail(email);
        user.setActivationCode(UUID.randomUUID().toString());
        user.setActive(false);
        mailSenderService.sendActivationCode(user);
    }
    if (!user.getPassword().equals(passwordEncoder.encode(password)) && !StringUtils.isEmpty(password)) {
        user.setPassword(passwordEncoder.encode(password));
    }
    userRepository.save(user);
}

public void activateUser(String code) {  1 usage  ± Tkachenko Vadym
    User user = userRepository.findByActivationCode(code);
    if (user != null) {
        user.setActivationCode(null);
        user.setActive(true);
        userRepository.save(user);
    }
}

public List<User> getUsersByNickname(String nickname) {  1 usage  ± Kabratigr\Vadidya
    if (nickname != null && !nickname.isEmpty()) {
        List<String> searchByKeyWords = Arrays.asList(nickname.split( regex: "\\s+"));
        return searchByKeyWords.stream()  Stream<String>
                .map(keyword -> userRepository.findByNicknameContainingIgnoreCase(nickname))  Stream<List<...>>
                .flatMap(List::stream)  Stream<User>
                .collect(Collectors.toList());
    }
    return userRepository.findAll();
}

public List<User> getAllUsers() {  1 usage  ± Tkachenko Vadym
    return userRepository.findAll();
}
```

Рисунок Б.9 – Сервісний клас UserService

```java
@Service  2 usages  new *
@RequiredArgsConstructor
public class BlogService {

    private final BlogRepository blogRepository;

    private final UtilityService utilityService;

    public List<Blog> filterBlogs(String title) {  1 usage  new *
        if (title != null && !title.isEmpty()) {
            List<String> searchByKeywords = Arrays.asList(title.split( regex: "\\s+"));
            return searchByKeywords.stream()  Stream<String>
                    .map(keyword -> blogRepository.findByTitleContainingIgnoreCase(title))  Stream<List<...>>
                    .flatMap(List::stream)  Stream<Blog>
                    .collect(Collectors.toList());
        }
        return blogRepository.findAll();
    }

    public void addBlog(Principal principal, Blog blog,  1 usage  new *
                        MultipartFile imageFile1,
                        MultipartFile imageFile2,
                        MultipartFile imageFile3) throws IOException {
        blog.setUser(utilityService.getUserByPrincipal(principal));
        utilityService.handleBlogImageFiles(blog, imageFile1, imageFile2, imageFile3);
        blogRepository.save(blog);
        utilityService.updateBlogPreviewImage(blog);
        blogRepository.save(blog);
    }

    public void updateBlog(Blog blog,  1 usage  new *
                           String title, String text,
                           MultipartFile imageFile1,
                           MultipartFile imageFile2,
                           MultipartFile imageFile3) throws IOException {
        if (!blog.getTitle().equals(title) && !StringUtils.isEmpty(title)) {
            blog.setTitle(title);
        }
        if (!blog.getText().equals(text) && !StringUtils.isEmpty(text)) {
            blog.setText(text);
        }
        utilityService.handleBlogImageFiles(blog, imageFile1, imageFile2, imageFile3);
        blogRepository.save(blog);
        utilityService.updateBlogPreviewImage(blog);
        blogRepository.save(blog);
    }

    public List<Blog> getAllBlogs() {  1 usage  new *
        return blogRepository.findAll();
    }

    public Blog getBlogById(Long id) {  2 usages  new *
        return blogRepository.findById(id).orElse( other: null);
    }

    public void deleteBlog(Long id) {  1 usage  new *
        blogRepository.deleteById(id);
    }
}
```

Рисунок Б.10 – Сервісний клас BlogService

ДОДАТОК В

Конфігураційні класи

```
@Configuration  Tkachenko Vadym
public class MailConfiguration {

    @Value("${spring.mail.host}")
    private String host;

    @Value("${spring.mail.username}")
    private String username;

    @Value("${spring.mail.password}")
    private String password;

    @Value("${spring.mail.port}")
    private int port;

    @Bean  Tkachenko Vadym
    public JavaMailSender getMailSender() {
        JavaMailSenderImpl sender = new JavaMailSenderImpl();
        sender.setHost(host);
        sender.setUsername(username);
        sender.setPassword(password);
        sender.setPort(port);
        Properties properties = sender.getJavaMailProperties();
        properties.put("mail.smtp.starttls.enable", "true");
        return sender;
    }
}
```

Рисунок В.1 – Конфігураційний клас MailConfiguration

```
@Getter  Kabratigr\Vadidya
@Setter
@Configuration
public class ReCaptchaConfiguration {

    @Value("${google.recaptcha.key.site}")
    private String siteKey;

    @Value("${google.recaptcha.key.secret}")
    private String secretKey;

    private final String RECAPTCHA_VERIFY_URL = "https://www.google.com/recaptcha/api/siteverify?secret=%s&response=%s";
}
```

Рисунок В.2 – Конфігураційний клас ReCaptchaConfiguration

```java
@Configuration  Tkachenko Vadym *
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {

    private final CustomUserDetailsService customUserDetailsService;

    @Bean  Tkachenko Vadym *
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests((requests) -> requests
                        .requestMatchers("/", "/images/**", "/registration", "/details/**",
                                "/filter/**", "/activate/*", "/photos/**") AuthorizedUrl
                        .permitAll() AuthorizationManagerRequestMat...
                        .anyRequest().authenticated()
                )
                .formLogin((form) -> form
                        .loginPage("/login")
                        .permitAll()
                )
                .logout(logout -> logout
                        .logoutSuccessUrl("/")
                        .permitAll()
                );
        return http.build();
    }

    @Bean  Tkachenko Vadym
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(customUserDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

    @Bean  Tkachenko Vadym
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
}
```

Рисунок В.3 – Конфігураційний клас SecurityConfiguration

ДОДАТОК Г

Класи контролери

```
@Controller  Tkachenko Vadym +1
@RequiredArgsConstructor
public class AnnouncementController {

    private final AnnouncementService announcementService;

    private final UtilityService utilityService;

    @GetMapping("/")  Tkachenko Vadym
    public String announcements(@RequestParam(name = "title", required = false) String title,
                                @RequestParam(name = "sort", required = false) String sort,
                                Model model, Principal principal) {
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        if (sort != null && !sort.isEmpty()) {
            model.addAttribute( attributeName: "announcements", announcementService.sortAnnouncements(
                    announcementService.getAllAnnouncements(), sort));
        } else {
            model.addAttribute( attributeName: "announcements", announcementService.filterByTitle(title));
        }
        return "main-page";
    }

    @GetMapping("/details/{id}")  Tkachenko Vadym
    public String announcementDetails(@PathVariable Long id, Model model, Principal principal) {
        model.addAttribute( attributeName: "announcement", announcementService.getAnnouncementById(id));
        model.addAttribute( attributeName: "images", announcementService.getAnnouncementById(id).getImages());
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "announcement-details";
    }

    @PreAuthorize("#announcement.user.email.equals(authentication.principal.email)")  Kabratigr\Vadidya +1
    @GetMapping("/{announcement}/edit")
    public String editAnnouncementForm(@PathVariable(name = "announcement") Announcement announcement, Model model, Principal principal) {
        model.addAttribute( attributeName: "announcement", announcement);
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "announcement-edit";
    }

    @PreAuthorize("#announcement.user.email.equals(authentication.principal.email)")  Tkachenko Vadym
    @PostMapping("/{announcement}/edit")
    public String editAnnouncement(@RequestParam(name = "title", required = false) String title,
                                   @RequestParam(name = "category", required = false) String category,
                                   @RequestParam(name = "price", required = false) int price,
                                   @RequestParam(name = "currency", required = false) String currency,
                                   @RequestParam(name = "city", required = false) String city,
                                   @RequestParam(name = "description", required = false) String description,
                                   @RequestParam(name = "imageFile1") MultipartFile imageFile1,
                                   @RequestParam(name = "imageFile2") MultipartFile imageFile2,
                                   @RequestParam(name = "imageFile3") MultipartFile imageFile3,
                                   @PathVariable(name = "announcement") Announcement announcement) throws IOException {
        announcementService.updateAnnouncement(announcement, title, category, price, currency, city, description, imageFile1, imageFile2, imageFile3);
        return "redirect:/";
    }
```

Рисунок Г.1 – Контролер AnnouncementController

```
    @GetMapping("/filter/{category}")  Tkachenko Vadym
    public String announcementCategories(@PathVariable String category,
                                         @RequestParam(name = "sort", required = false) String sort,
                                         Model model, Principal principal) {
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        if (sort != null && !sort.isEmpty()) {
            model.addAttribute( attributeName: "announcements", announcementService.sortAnnouncements(
                    announcementService.getAnnouncementsByCategory(category), sort));
        } else {
            model.addAttribute( attributeName: "announcements", announcementService.getAnnouncementsByCategory(category));
        }
        return "announcement-category";
    }

    @PostMapping("/filter")  Tkachenko Vadym +1
    public String announcementsSearchOnCategoriesPage(@RequestParam(name = "title", required = false) String title,
                                                      @RequestParam(name = "category", required = false) String category,
                                                      Model model, Principal principal) {
        model.addAttribute( attributeName: "announcements", announcementService.filterOnCategoryPage(title, category));
        model.addAttribute( attributeName: "category", category);
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "announcement-category";
    }

    @GetMapping("/add")  Kabratigr\Vadidya +1
    public String addAnnouncementPage(Principal principal, Model model) {
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "add-announcement";
    }

    @PostMapping("/add")  Tkachenko Vadym +1
    public String addAnnouncement(@RequestParam(name = "imageFile1") MultipartFile imageFile1,
                                  @RequestParam(name = "imageFile2") MultipartFile imageFile2,
                                  @RequestParam(name = "imageFile3") MultipartFile imageFile3,
                                  @RequestParam(name = "notifySubscribers", defaultValue = "false") boolean notifySubscribers,
                                  Announcement announcement, Principal principal) throws IOException {
        announcementService.addAnnouncement(principal, announcement, imageFile1, imageFile2, imageFile3, notifySubscribers);
        return "redirect:/";
    }

    @PostMapping("/delete/{id}")  Tkachenko Vadym
    public String deleteAnnouncement(@PathVariable Long id) {
        announcementService.deleteAnnouncement(id);
        return "redirect:/";
    }
}
```

Рисунок Г.2 – Контролер AnnouncementController

```java
@Controller  Tkachenko Vadym
@PreAuthorize("hasAuthority('ROLE_ADMIN')")
@RequiredArgsConstructor
public class AdminController {

    private final UserService userService;

    private final UtilityService utilityService;

    @GetMapping("/admin")  Tkachenko Vadym
    public String adminPanel(Principal principal, Model model) {
        model.addAttribute( attributeName: "users", userService.getAllUsers());
        model.addAttribute( attributeName: "admin", utilityService.getUserByPrincipal(principal));
        return "admin-page";
    }

    @PostMapping("/user/ban/{id}")  Tkachenko Vadym
    public String banAndUnbanUser(@PathVariable(name = "id") Long id) {
        userService.banAndUnbanUser(id);
        return "redirect:/admin";
    }

    @GetMapping("/user/edit/{user}")  Tkachenko Vadym
    public String userChangeRole(@PathVariable(name = "user") User user, Model model) {
        model.addAttribute( attributeName: "user", user);
        model.addAttribute( attributeName: "roles", Role.values());
        return "user-change-role";
    }

    @PostMapping("/user/edit")  Tkachenko Vadym
    public String userChangeRole(@RequestParam(name = "userId") User user,
                                 @RequestParam Map<String, String> form) {
        userService.userChangeRole(user, form);
        return "redirect:/admin";
    }
}
```

Рисунок Г.3 – Контролер AdminController

```java
@RestController Kabratigr\Vadidya +1 *
@RequiredArgsConstructor
public class ImageRestController {

    private final ImageRepository imageRepository;

    private final AnnouncementRepository announcementRepository;

    private final BlogRepository blogRepository;

    @GetMapping("/images/{id}") Tkachenko Vadym
    private ResponseEntity<?> getImageById(@PathVariable Long id) {
        Image image = imageRepository.findById(id).orElse( other: null);
        return ResponseEntity.ok()
                .header( headerName: "fileName", image.getOriginalImageName())
                .contentType(MediaType.valueOf(image.getContentType()))
                .contentLength(image.getImageSize())
                .body(new InputStreamResource(new ByteArrayInputStream(image.getBytes())));
    }

    @DeleteMapping("/images/{id}") Kabratigr\Vadidya
    public ResponseEntity<?> deleteImage(@PathVariable Long id) {
        Optional<Image> imageOptional = imageRepository.findById(id);
        if (imageOptional.isPresent()) {
            Image image = imageOptional.get();
            Announcement announcement = image.getAnnouncement();
            announcement.getImages().remove(image);
            imageRepository.delete(image);
            if (image.isPreviewImage()) {
                announcement.setPreviewImageId(null);
            }
            announcementRepository.save(announcement);
            return ResponseEntity.ok().build();
        }
        return ResponseEntity.notFound().build();
    }

    @DeleteMapping("/blog/images/{id}") new *
    public ResponseEntity<?> deleteBlogImage(@PathVariable Long id) {
        Optional<Image> imageOptional = imageRepository.findById(id);
        if (imageOptional.isPresent()) {
            Image image = imageOptional.get();
            Blog blog = image.getBlog();
            blog.getImages().remove(image);
            imageRepository.delete(image);
            if (image.isPreviewImage()) {
                blog.setPreviewImageId(null);
            }
            blogRepository.save(blog);
            return ResponseEntity.ok().build();
        }
        return ResponseEntity.notFound().build();
    }
}
```

Рисунок Г.4 – Контролер ImageRestController

```java
@Controller  Kabratigr\Vadidya +1
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    private final UtilityService utilityService;

    private final ReCaptchaService reCaptchaService;

    @GetMapping("/login")  Tkachenko Vadym
    public String login() {
        return "login-page";
    }

    @GetMapping("/registration")  Tkachenko Vadym
    public String registration() {
        return "registration-page";
    }

    @PostMapping("/registration")  Kabratigr\Vadidya +1
    public String createUser(@RequestParam("g-recaptcha-response") String reCaptchaResponse,
            User user, Model model) {
        if (!reCaptchaService.validateRecaptcha(reCaptchaResponse)) {
            model.addAttribute( attributeName: "captchaError",  attributeValue: "Invalid Captcha Validation");
            return "registration-page";
        }
        UserStatus errorCode = userService.createUser(user);
        switch (errorCode) {
            case EMAIL_EXISTS:
                model.addAttribute( attributeName: "emailError",  attributeValue: "A user with " +
                        user.getEmail() + " already exists.");
                return "registration-page";
            case NICKNAME_EXISTS:
                model.addAttribute( attributeName: "nickNameError",  attributeValue: "A user with " +
                        user.getNickname() + " already exists.");
                return "registration-page";
            case SUCCESS: return "redirect:/login";
            default:
                model.addAttribute( attributeName: "unknownError",  attributeValue: "Unknown error");
                return "registration-page";
        }
    }

    @GetMapping("/user/{user}")  Tkachenko Vadym +1
    public String userProfile(@PathVariable(name = "user") User user, Principal principal, Model model) {
        model.addAttribute( attributeName: "user", user);
        model.addAttribute( attributeName: "announcements", user.getAnnouncements());
        model.addAttribute( attributeName: "numberOfSubscribers", user.getSubscribers().size());
        model.addAttribute( attributeName: "numberOfSubscriptions", user.getSubscriptions().size());
        model.addAttribute( attributeName: "isSubscriber", user.getSubscribers().contains(utilityService.getUserByPrincipal(principal)));
        model.addAttribute( attributeName: "authenticatedUser", utilityService.getUserByPrincipal(principal));
        return "user-details";
    }

    @GetMapping("/subscribe/{user}")  Kabratigr\Vadidya
    public String subscribe(@PathVariable(name = "user") User user, Principal principal) {
        userService.subscribe(utilityService.getUserByPrincipal(principal), user);
        return "redirect:/user/" + user.getId();
    }

    @GetMapping("/unsubscribe/{user}")  Kabratigr\Vadidya
    public String unSubscribe(@PathVariable(name = "user") User user, Principal principal) {
        userService.unSubscribe(utilityService.getUserByPrincipal(principal), user);
        return "redirect:/user/" + user.getId();
    }
```

Рисунок Г.5 – Контролер UserController

```
    @GetMapping("/{type}/{user}/list")  Kabratigr\Vadidya
    public String subscriptionsList(@PathVariable String type,
                                    @PathVariable User user,
                                    @RequestParam(name = "nickname", required = false) String nickname,
                                    Model model, Principal principal) {
        model.addAttribute( attributeName: "userChannel", user);
        model.addAttribute( attributeName: "type", type);
        model.addAttribute( attributeName: "authenticatedUser", utilityService.getUserByPrincipal(principal));
        if ("subscriptions".equals(type)) {
            model.addAttribute( attributeName: "userSubscriptions", user.getSubscriptions());
        } else {
            model.addAttribute( attributeName: "userSubscribers", user.getSubscribers());
        }
        model.addAttribute( attributeName: "userList", userService.findSubscriptions(type, nickname, user));
        return "subscriptions-list";
    }

    @GetMapping("/users/list")  Kabratigr\Vadidya
    public String usersList(@RequestParam(name = "nickname", required = false) String nickname,
                            Model model, Principal principal) {
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        model.addAttribute( attributeName: "allUsers", userService.getUsersByNickname(nickname));
        return "user-list";
    }

    @PreAuthorize("#user.email.equals(authentication.principal.email)")  Tkachenko Vadym
    @GetMapping("/user/{user}/edit")
    public String editUserProfileForm(@PathVariable(name = "user") User user, Model model) {
        model.addAttribute( attributeName: "user", user);
        return "user-edit";
    }

    @PreAuthorize("#user.email.equals(authentication.principal.email)")  Tkachenko Vadym +1
    @PostMapping("/user/{user}/edit")
    public String editUserProfile(@RequestParam(name = "name", required = false) String name,
                                  @RequestParam(name = "surname", required = false) String surname,
                                  @RequestParam(name = "nickname", required = false) String nickname,
                                  @RequestParam(name = "email", required = false) String email,
                                  @RequestParam(name = "password", required = false) String password,
                                  @PathVariable(name = "user") User user) {
        String userEmailBeforeUpdate = user.getEmail();
        userService.updateProfile(user, nickname, name, surname, email, password);
        if (!userEmailBeforeUpdate.equals(email)) {
            return "redirect:/login";
        }
        return "redirect:/user/{user}";
    }

    @GetMapping("/activate/{code}")  Tkachenko Vadym
    public String activateUser(@PathVariable String code) {
        userService.activateUser(code);
        return "redirect:/login";
    }
}
```

Рисунок Г.6 – Контролер UserController

```java
@Controller new *
@RequiredArgsConstructor
public class BlogController {

    private final BlogService blogService;

    private final UtilityService utilityService;

    @GetMapping("/blogs") new *
    public String blogs(@RequestParam(name = "title", required = false) String title, Model model, Principal principal) {
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        if (title != null && !title.isEmpty()) {
            model.addAttribute( attributeName: "blogs", blogService.filterBlogs(title));
        } else {
            model.addAttribute( attributeName: "blogs", blogService.getAllBlogs());
        }
        return "blogs-page";
    }

    @GetMapping("/blog/details/{id}") new *
    public String announcementDetails(@PathVariable Long id, Model model, Principal principal) {
        model.addAttribute( attributeName: "blog", blogService.getBlogById(id));
        model.addAttribute( attributeName: "images", blogService.getBlogById(id).getImages());
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "blog-details";
    }

    @GetMapping("/addBlog") new *
    public String addBlogPage(Principal principal, Model model) {
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "add-blog";
    }

    @PostMapping("/addBlog") new *
    public String addBlog(@RequestParam(name = "imageFile1") MultipartFile imageFile1,
                          @RequestParam(name = "imageFile2") MultipartFile imageFile2,
                          @RequestParam(name = "imageFile3") MultipartFile imageFile3,
                          Blog blog, Principal principal) throws IOException {
        blogService.addBlog(principal, blog, imageFile1, imageFile2, imageFile3);
        return "redirect:/blogs";
    }

    @PreAuthorize("#blog.user.email.equals(authentication.principal.email)") new *
    @GetMapping("/blog/{blog}/edit")
    public String editBlogPage(@PathVariable(name = "blog") Blog blog, Model model, Principal principal) {
        model.addAttribute( attributeName: "blog", blog);
        model.addAttribute( attributeName: "user", utilityService.getUserByPrincipal(principal));
        return "blog-edit";
    }

    @PreAuthorize("#blog.user.email.equals(authentication.principal.email)") new *
    @PostMapping("/blog/{blog}/edit")
    public String editBlog(@PathVariable(name = "blog") Blog blog,
                           @RequestParam(name = "title", required = false) String title,
                           @RequestParam(name = "text", required = false) String text,
                           @RequestParam(name = "imageFile1") MultipartFile imageFile1,
                           @RequestParam(name = "imageFile2") MultipartFile imageFile2,
                           @RequestParam(name = "imageFile3") MultipartFile imageFile3) throws IOException {
        blogService.updateBlog(blog, title, text, imageFile1, imageFile2, imageFile3);
        return "redirect:/blogs";
    }

    @PostMapping("/blog/delete/{id}") new *
    public String deleteBlog(@PathVariable Long id) {
        blogService.deleteBlog(id);
        return "redirect:/blogs";
    }
}
```

Рисунок Г.7 – Контролер BlogController

ДОДАТОК Д

Html файли

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <title>Add Announcement</title>
    <!-- Include Bootstrap CSS -->
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" th:href="@{/}">Bulletin Board</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarText"
            aria-controls="navbarText" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarText">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" id="announcementsDropdown" role="button"
                   data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                    Announcements
                </a>
                <div class="dropdown-menu" aria-labelledby="announcementsDropdown">
                    <a class="dropdown-item" th:href="@{'/filter/Work Offer'}">Work Offers</a>
                    <a class="dropdown-item" th:href="@{'/filter/Work Search'}">Work Searches</a>
                    <a class="dropdown-item" th:href="@{'/filter/Purchase'}">Purchases</a>
                    <a class="dropdown-item" th:href="@{'/filter/Sale'}">Sales</a>
                    <a class="dropdown-item" th:href="@{/add}">Add Announcement</a>
                </div>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/users/list}">Users List</a>
            </li>
            <li class="nav-item" th:if="${user.isAdmin()}">
                <a class="nav-link" th:href="@{/admin}">Admin Panel</a>
            </li>
        </ul>
        <span class="navbar-text mr-3" th:if="${user.email != null}">
            Logged in as: <a th:href="@{'/user/' + ${user.id}}" th:text="${user.name + ' ' + user.surname}"></a>
        </span>
        <a th:if="${user.email == null}" th:href="@{/login}" class="btn btn-outline-primary ml-2">Log in</a>
        <form th:if="${user.email != null}" th:action="@{/logout}" method="post" class="ml-2">
            <button type="submit" class="btn btn-outline-danger">Log out</button>
        </form>
    </div>
</nav>

<div class="container mt-4">
    <h2>Add Announcement</h2>
    <form th:action="@{/add}" method="post" enctype="multipart/form-data">
        <div class="form-group">
            <label for="title">Title:</label>
            <input type="text" class="form-control" id="title" name="title" required>
        </div>

        <div class="form-group">
            <label for="category">Category:</label>
            <select class="form-control" id="category" name="category" required>
                <option value="Work Search">Work Search</option>
                <option value="Work Offer">Work Offer</option>
                <option value="Sale">Sale</option>
                <option value="Purchase">Purchase</option>
            </select>
        </div>

        <div class="form-group">
            <label for="price">Price:</label>
            <input type="number" class="form-control" id="price" name="price" required>
        </div>
```

Рисунок Д.1 – Шаблон add-announcement.html

```
        <div class="form-group">
            <label for="currency">Currency:</label>
            <select class="form-control" id="currency" name="currency" required>
                <option value="USD">USD</option>
                <option value="EUR">EUR</option>
                <option value="UAH">UAH</option>
            </select>
        </div>

        <div class="form-group">
            <label for="city">City:</label>
            <input type="text" class="form-control" id="city" name="city" required>
        </div>

        <div class="form-group">
            <label for="file1">Preview Image:</label>
            <input type="file" class="form-control-file" id="file1" name="imageFile1">
        </div>

        <div class="form-group">
            <label for="file2">Second Image:</label>
            <input type="file" class="form-control-file" id="file2" name="imageFile2">
        </div>

        <div class="form-group">
            <label for="file3">Third Image:</label>
            <input type="file" class="form-control-file" id="file3" name="imageFile3">
        </div>

        <div class="form-group">
            <label for="description">Description:</label>
            <textarea class="form-control" id="description" name="description" rows="3"></textarea>
        </div>

        <div class="form-group" th:if="${user.subscribers.size() > 0}">
            <input type="checkbox" id="notifySubscribers" name="notifySubscribers">
            <label for="notifySubscribers">Notify all subscribers</label>
        </div>

        <button type="submit" class="btn btn-primary">Add Announcement</button>
    </form>
</div>

<!-- Include Bootstrap JS and dependencies -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Рисунок Д.2 – Шаблон add-announcement.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <title>Admin Panel</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css"
          integrity="sha384-WskhaSGFgHYWDcbwN70/dfYBj47jz9qbsMId/iRN3ewGhXQFZCSftd1LZCfmhktB" crossorigin="anonymous">
</head>
<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" th:href="@{/}">Bulletin Board</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarText"
            aria-controls="navbarText" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarText">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" id="announcementsDropdown" role="button"
                   data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">Announcements</a>
                <div class="dropdown-menu" aria-labelledby="announcementsDropdown">
                    <a class="dropdown-item" th:href="@{'/filter/Work Offer'}">Work Offers</a>
                    <a class="dropdown-item" th:href="@{'/filter/Work Search'}">Work Searches</a>
                    <a class="dropdown-item" th:href="@{'/filter/Purchase'}">Purchases</a>
                    <a class="dropdown-item" th:href="@{'/filter/Sale'}">Sales</a>
                    <a class="dropdown-item" th:href="@{/add}">Add Announcement</a>
                </div>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/users/list}">Users List</a>
            </li>
            <li class="nav-item" th:if="${admin.isAdmin()}">
                <a class="nav-link" th:href="@{/admin}">Admin Panel</a>
            </li>
        </ul>
        <span class="navbar-text mr-3" th:if="${admin.email != null}">
            Logged in as: <a th:href="@{'/user/' + ${admin.id}}" th:text="${admin.nickname}"></a>
        </span>
        <a th:if="${admin.email == null}" th:href="@{/login}" class="btn btn-outline-primary ml-2">Log in</a>
        <!-- Log out with confirmation modal trigger -->
        <button type="button" class="btn btn-outline-danger ml-2"
                th:if="${admin.email != null}" data-toggle="modal" data-target="#confirmLogoutModal">Log out</button>
    </div>
</nav>

<!-- Log out confirmation modal -->
<div class="modal fade" id="confirmLogoutModal" tabindex="-1"
     role="dialog" aria-labelledby="confirmLogoutModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="confirmLogoutModalLabel">Confirm Logout</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                Are you sure you want to log out?
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
                <form th:action="@{/logout}" method="post" class="d-inline">
                    <button type="submit" class="btn btn-danger">Log out</button>
                </form>
            </div>
        </div>
    </div>
</div>

<div class="container mt-5">
    <div class="d-flex justify-content-between align-items-center mb-4">
        <h1>Admin Panel</h1>
    </div>

    <table class="table table-striped">
        <thead>
        <tr th:if="${#lists.isEmpty(users)}">
            <th colspan="8" class="text-center">No users found</th>
        </tr>
        <tr th:unless="${#lists.isEmpty(users)}">
            <th>Id</th>
            <th>Name</th>
            <th>Surname</th>
            <th>Nickname</th>
            <th>Email</th>
            <th>Registration Date</th>
            <th>Activity</th>
            <th>Role</th>
            <th>Actions</th>
        </tr>
        </thead>
        <tbody>
```

Рисунок Д.3 – Шаблон admin-page.html

```
        <tbody>
        <tr th:each="user : ${users}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.name}"></td>
            <td th:text="${user.surname}"></td>
            <td th:text="${user.nickname}"></td>
            <td th:text="${user.email}"></td>
            <td th:text="${#temporals.format(user.registrationDate, 'dd.MM.yyyy HH:mm')}"></td>
            <td th:text="${user.isActive}"></td>
            <td th:text="${user.getRoles()}"></td>
            <td>
                <form th:if="${user.isActive() && admin.email != user.email}"
                      th:action="@{'/user/ban/' + ${user.id}}" method="post" class="d-inline">
                    <button type="submit" class="btn btn-warning btn-sm">Ban</button>
                </form>
                <form th:if="${!user.isActive() && admin.email != user.email}"
                      th:action="@{'/user/ban/' + ${user.id}}" method="post" class="d-inline">
                    <button type="submit" class="btn btn-success btn-sm">Unban</button>
                </form>
                <a th:if="${admin.email != user.email}"
                   th:href="@{'/user/edit/' + ${user.id}}" class="btn btn-info btn-sm">Change Role</a>
            </td>
        </tr>
        </tbody>
    </table>
</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T" crossorigin="anonymous"></script>

</body>
</html>
```

Рисунок Д.4 – Шаблон admin-page.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <title>Announcement Category</title>
    <!-- Add Bootstrap CSS here -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css"
          integrity="sha384-WskhaSGFgHYWDcbwN70/dfYBj47jz9qbsMId/iRN3ewGhXQFZCSftd1LZCfmhktB" crossorigin="anonymous">
</head>
<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" th:href="@{/}">Bulletin Board</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarText"
            aria-controls="navbarText" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarText">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" id="announcementsDropdown" role="button"
                   data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">Announcements</a>
                <div class="dropdown-menu" aria-labelledby="announcementsDropdown">
                    <a class="dropdown-item" th:href="@{'/filter/Work Offer'}">Work Offers</a>
                    <a class="dropdown-item" th:href="@{'/filter/Work Search'}">Work Searches</a>
                    <a class="dropdown-item" th:href="@{'/filter/Purchase'}">Purchases</a>
                    <a class="dropdown-item" th:href="@{'/filter/Sale'}">Sales</a>
                    <a class="dropdown-item" th:href="@{/add}">Add Announcement</a>
                </div>
            </li>
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" id="blogsDropdown" role="button"
                   data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">Blogs</a>
                <div class="dropdown-menu" aria-labelledby="blogsDropdown">
                    <a class="dropdown-item" th:href="@{/blogs}">All Blogs</a>
                    <a class="dropdown-item" th:href="@{/addBlog}">Add Blog</a>
                </div>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/users/list}">Users List</a>
            </li>
            <li class="nav-item" th:if="${user.isAdmin()}">
                <a class="nav-link" th:href="@{/admin}">Admin Panel</a>
            </li>
        </ul>
        <span class="navbar-text mr-3" th:if="${user.email != null}">
            Logged in as: <a th:href="@{'/user/' + ${user.id}}" th:text="${user.nickname}"></a>
        </span>
        <a th:if="${user.email == null}" th:href="@{/login}" class="btn btn-outline-primary ml-2">Log in</a>
        <!-- Log out with confirmation modal trigger -->
        <button type="button" class="btn btn-outline-danger ml-2"
                th:if="${user.email != null}" data-toggle="modal" data-target="#confirmLogoutModal">Log out</button>
    </div>
</nav>

<!-- Log out confirmation modal -->
<div class="modal fade" id="confirmLogoutModal" tabindex="-1" role="dialog" aria-labelledby="confirmLogoutModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="confirmLogoutModalLabel">Confirm Logout</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                Are you sure you want to log out?
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
                <form th:action="@{/logout}" method="post" class="d-inline">
                    <button type="submit" class="btn btn-danger">Log out</button>
                </form>
            </div>
        </div>
    </div>
</div>
```

Рисунок Д.5 – Шаблон announcement-category.html

```
<div class="container-fluid">
    <div class="row my-3">
        <div class="col-md-6">
            <form th:action="@{/filter}" method="post" class="form-inline">
                <label for="title" class="mr-2">Search by Title:</label>
                <input type="text" id="title" name="title" class="form-control mr-2" th:value="${title}" placeholder="Enter title"/>
                <input type="hidden" id="category" name="category" th:value="${category}"/>
                <button type="submit" class="btn btn-primary">Search</button>
            </form>
        </div>
        <div class="col-md-6">
            <form th:action="@{/filter/{category}(category=${category})}" method="get" class="form-inline float-md-right">
                <label for="sort" class="mr-2">Sort by:</label>
                <select id="sort" name="sort" class="form-control mr-2">
                    <option value="priceAscending">Price: Low to High</option>
                    <option value="priceDescending">Price: High to Low</option>
                    <option value="dateAscending">Date: Old to New</option>
                    <option value="dateDescending">Date: New to Old</option>
                </select>
                <button type="submit" class="btn btn-primary">Sort</button>
            </form>
        </div>
    </div>

    <div class="row">
        <div th:if="${#lists.isEmpty(announcements)}" class="col-12">
            <p>No announcements available</p>
        </div>

        <div th:each="announcement : ${announcements}" class="col-md-3 mb-4">
            <div class="card" style="width: 100%;">
                <img class="card-img-top"
                     th:if="${announcement.previewImageId != null}"
                     th:src="@{'/images/{id}'(id=${announcement.previewImageId})}"
                     alt="Announcement Image" style="max-height: 180px; object-fit: contain;">
                <img class="card-img-top"
                     th:if="${announcement.previewImageId == null}"
                     th:src="@{/photos/defaultPreviewImage.jpg}"
                     alt="Default Announcement Image" style="max-height: 180px; object-fit: contain;">
                <div class="card-body">
                    <h5 class="card-title" th:text="${announcement.title}"></h5>
                    <p class="card-text" th:text="${announcement.description}"></p>
                </div>
                <ul class="list-group list-group-flush">
                    <li class="list-group-item"
                        th:text="'Price: ' + ${announcement.price} + ' ' + ${announcement.currency}"></li>
                    <li class="list-group-item">
                        <a th:href="@{'/user/' + ${announcement.user.id}}"
                           th:text="${announcement.user.nickname}" style="text-decoration: none;"></a>
                    </li>
                    <li class="list-group-item" th:text="'Posted on: ' +
                    ${#temporals.format(announcement.creationDate, 'dd.MM.yyyy HH:mm')}"></li>
                </ul>
                <div class="card-body">
                    <a th:href="@{'/details/' + ${announcement.id}}" class="btn btn-primary">See more</a>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Add Bootstrap JS and dependencies here -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T" crossorigin="anonymous"></script>
</body>
</html>
```

Рисунок Д.6 – Шаблон announcement-category.html

```
<div class="container mt-4">
    <h2>Announcement Details</h2>
    <div class="card p-3">

        <div class="d-flex justify-content-center mt-3">
            <div class="d-flex flex-row gap-2" th:each="image : ${images}">
                <img th:src="@{/images/{id}(id=${image.id})}" class="rounded img-thumbnail" style="width: 180px; height: auto;" alt="Image"/>
            </div>
        </div>

        <p><strong>Title:</strong> <span th:text="${announcement.title}"></span></p>
        <p><strong>Creation Date:</strong> <span th:text="${#temporals.format(announcement.creationDate, 'dd.MM.yyyy HH:mm')}"></span></p>
        <p><strong>Author:</strong>
            <a th:href="@{'/user/' + ${announcement.user.id}}">
                <span th:text="${announcement.user.nickname}"></span>
            </a>
        </p>

        <p><strong>Category:</strong> <span th:text="${announcement.category}"></span></p>
        <p>
            <span th:if="${announcement.category == 'Work Offer' || announcement.category == 'Work Search'}">
                <strong>Salary:</strong> <span th:text="${announcement.price}"></span> <span th:text="${announcement.currency}"></span>
            </span>
            <span th:if="${announcement.category == 'Sale' || announcement.category == 'Purchase'}">
                <strong>Price:</strong> <span th:text="${announcement.price}"></span> <span th:text="${announcement.currency}"></span>
            </span>
        </p>
        <p><strong>City:</strong> <span th:text="${announcement.city}"></span></p>
        <p><strong>Description:</strong> <span th:text="${announcement.description}"></span></p>

        <div th:if="${announcement.user.email == user.email}" class="mt-3">
            <button type="button" class="btn btn-danger" data-toggle="modal" data-target="#confirmDeleteModal">
                Delete Announcement
            </button>
            <a th:href="@{'/' + ${announcement.id} + '/edit'}" class="btn btn-warning">Edit Announcement</a>
            <div class="modal fade" id="confirmDeleteModal" tabindex="-1" role="dialog" aria-labelledby="confirmDeleteModalLabel" aria-hidden="true">
                <div class="modal-dialog" role="document">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h5 class="modal-title" id="confirmDeleteModalLabel">Confirm Deletion</h5>
                            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                                <span aria-hidden="true">&times;</span>
                            </button>
                        </div>
                        <div class="modal-body">
                            Are you sure you want to delete this announcement? This action cannot be undone.
                        </div>
                        <div class="modal-footer">
                            <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
                            <button type="button" class="btn btn-danger" onclick="confirmDeletion()">Delete</button>
                        </div>
                    </div>
                </div>
            </div>

            <form id="deleteForm" th:action="@{'/delete/' + ${announcement.id}}" method="post" class="d-none">
                <input type="hidden" name="_csrf" th:value="${_csrf.token}" />
            </form>
        </div>
    </div>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
<script>
    function confirmDeletion() {  Show usages   Kabratigr\Vadidya
        document.getElementById("deleteForm").submit();
    }
</script>
</body>
</html>
```

Рисунок Д.7 – Шаблон announcement-details.html

```
<!-- Confirm Delete Image Modal -->
<div class="modal fade" id="confirmDeleteImageModal" tabindex="-1" role="dialog" aria-labelledby="confirmDeleteImageModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="confirmDeleteImageModalLabel">Confirm Delete</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                Are you sure you want to delete this image?
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
                <button type="button" class="btn btn-danger" onclick="deleteImage()">Delete</button>
            </div>
        </div>
    </div>
</div>

<div class="container mt-4">
    <h2>Edit Announcement</h2>
    <form th:action="@{'/' + ${announcement.id} + '/edit'}" method="post" enctype="multipart/form-data">
        <div class="form-group">
            <label for="title">Title:</label>
            <input type="text" class="form-control" id="title" name="title" th:value="*{announcement.title}" required>
        </div>

        <div class="form-group">
            <label>Existing Images:</label>
            <div class="row">
                <div class="col-md-3" th:each="image : ${announcement.images}">
                    <div class="card mb-2">
                        <img th:src="@{'/images/' + ${image.id}}" class="card-img-top" alt="Image">
                        <div class="card-body p-2">
                            <button type="button" class="btn btn-danger btn-sm btn-block"
                                    th:onclick="'showDeleteConfirmation(' + ${image.id} + ')'">Delete</button>
                        </div>
                    </div>
                </div>
            </div>
        </div>

        <div class="form-group">
            <label for="category">Category:</label>
            <select class="form-control" id="category" name="category" th:value="*{announcement.category}" required>
                <option value="Work Search">Work Search</option>
                <option value="Work Offer">Work Offer</option>
                <option value="Sale">Sale</option>
                <option value="Purchase">Purchase</option>
            </select>
        </div>

        <div class="form-group">
            <label for="price">Price:</label>
            <input type="number" class="form-control" id="price" name="price" th:value="*{announcement.price}" required>
        </div>

        <div class="form-group">
            <label for="currency">Currency:</label>
            <select class="form-control" id="currency" name="currency" th:value="*{announcement.currency}" required>
                <option value="USD">USD</option>
                <option value="EUR">EUR</option>
                <option value="UAH">UAH</option>
            </select>
        </div>

        <div class="form-group">
            <label for="city">City:</label>
            <input type="text" class="form-control" id="city" name="city" th:value="*{announcement.city}" required>
        </div>

        <div class="form-group">
            <label for="file1">Preview Image:</label>
            <input type="file" class="form-control-file" id="file1" name="imageFile1">
        </div>

        <div class="form-group">
            <label for="file2">Second Image:</label>
            <input type="file" class="form-control-file" id="file2" name="imageFile2">
        </div>
```

Рисунок Д.8 – Шаблон announcement-edit.html

```
        <div class="form-group">
            <label for="file3">Third Image:</label>
            <input type="file" class="form-control-file" id="file3" name="imageFile3">
        </div>

        <div class="form-group">
            <label for="description">Description:</label>
            <textarea class="form-control" id="description" name="description" th:value="*{announcement.description}" rows="3"></textarea>
        </div>

        <button type="submit" class="btn btn-primary">Save</button>
    </form>
</div>

<!-- Include Bootstrap JS and dependencies -->
<script>
    let imageIdToDelete = null;

    function showDeleteConfirmation(imageId) {  no usages  Kabratigr\Vadidya
        imageIdToDelete = imageId;
        $('#confirmDeleteImageModal').modal('show');
    }

    function deleteImage() {  Show usages  Kabratigr\Vadidya
        const csrfToken = document.querySelector('meta[name="_csrf"]').getAttribute('content');
        const csrfHeader = document.querySelector('meta[name="_csrf_header"]').getAttribute('content');

        fetch(`/images/${imageIdToDelete}`, {
            method: 'DELETE',
            headers: {
                [csrfHeader]: csrfToken
            }
        }).then(response => {
            $('#confirmDeleteImageModal').modal('hide');
            if (response.ok) {
                location.reload();
            } else {
                alert('Failed to delete image');
            }
        }).catch(error => {
            alert('Failed to delete image');
        });
    }
</script>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Рисунок Д.9 – Шаблон announcement-edit.html

```
<div class="container mt-4">
    <h2>Post a Blog</h2>
    <form th:action="@{/addBlog}" method="post" enctype="multipart/form-data">
        <div class="form-group">
            <label for="title">Title:</label>
            <input type="text" class="form-control" id="title" name="title" required>
        </div>

        <div class="form-group">
            <label for="file1">Preview Image:</label>
            <input type="file" class="form-control-file" id="file1" name="imageFile1">
        </div>

        <div class="form-group">
            <label for="file2">Second Image:</label>
            <input type="file" class="form-control-file" id="file2" name="imageFile2">
        </div>

        <div class="form-group">
            <label for="file3">Third Image:</label>
            <input type="file" class="form-control-file" id="file3" name="imageFile3">
        </div>

        <div class="form-group">
            <label for="text">Text:</label>
            <textarea class="form-control" id="text" name="text" rows="3"></textarea>
        </div>

        <button type="submit" class="btn btn-primary">Post</button>
    </form>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Рисунок Д.10 – Шаблон add-blog.html

```
<div class="container mt-4">
    <h2 th:text="${blog.title}">Blog Title</h2>
    <div class="card p-3">

        <!-- Images -->
        <div class="d-flex justify-content-center mt-3">
            <div class="d-flex flex-row gap-2" th:each="image : ${images}">
                <img th:src="@{'/images/{id}'(id=${image.id})}" class="rounded img-thumbnail" style="width: 180px; height: auto;" alt="Blog Image"/>
            </div>
        </div>

        <!-- Blog details -->
        <p><strong>Content:</strong> <span th:text="${blog.text}">Blog content</span></p>
        <p><strong>Author:</strong>
            <a th:href="@{'/user/' + ${blog.user.id}}">
                <span th:text="${blog.user.nickname}"></span>
            </a>
        </p>
        <p><strong>Posted on:</strong> <span th:text="${#temporals.format(blog.creationDate, 'dd.MM.yyyy HH:mm')}"></span></p>

        <!-- Actions for blog owner -->
        <div th:if="${blog.user.email == user.email}" class="mt-3">
            <button type="button" class="btn btn-danger" data-toggle="modal" data-target="#confirmDeleteModal">
                Delete Blog
            </button>
            <a th:href="@{'/blog/' + ${blog.id} + '/edit'}" class="btn btn-warning ml-2">Edit Blog</a>

            <!-- Delete confirmation modal -->
            <div class="modal fade" id="confirmDeleteModal" tabindex="-1" role="dialog" aria-labelledby="confirmDeleteModalLabel" aria-hidden="true">
                <div class="modal-dialog" role="document">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h5 class="modal-title" id="confirmDeleteModalLabel">Confirm Deletion</h5>
                            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                                <span aria-hidden="true">&times;</span>
                            </button>
                        </div>
                        <div class="modal-body">
                            Are you sure you want to delete this blog? This action cannot be undone.
                        </div>
                        <div class="modal-footer">
                            <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
                            <button type="button" class="btn btn-danger" onclick="confirmDeletion()">Delete</button>
                        </div>
                    </div>
                </div>
            </div>

            <!-- Hidden delete form -->
            <form id="deleteForm" th:action="@{'/blog/delete/' + ${blog.id}}" method="post" class="d-none">
                <input type="hidden" name="_csrf" th:value="${_csrf.token}" />
            </form>
        </div>
    </div>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
<script>
    function confirmDeletion() {  Show usages  new *
        document.getElementById("deleteForm").submit();
    }
</script>
</body>
</html>
```

Рисунок Д.11 – Шаблон blog-details.html

```
<div class="container mt-4">
    <h2>Edit Blog</h2>
    <form th:action="@{'/blog/{blog}/edit'(blog=${blog.id})}" th:object="${blog}" method="post" enctype="multipart/form-data">
        <div class="form-group">
            <label for="title">Title:</label>
            <input type="text" class="form-control" id="title" th:field="*{title}">
        </div>

        <div class="form-group">
            <label>Existing Images:</label>
            <div class="row">
                <div class="col-md-3" th:each="image : ${blog.images}">
                    <div class="card mb-2">
                        <img th:src="@{'/images/' + ${image.id}}" class="card-img-top" alt="Image">
                        <div class="card-body p-2">
                            <button type="button" class="btn btn-danger btn-sm btn-block"
                                    th:onclick="'showDeleteConfirmation(' + ${image.id} + ')'">Delete</button>
                        </div>
                    </div>
                </div>
            </div>
        </div>

        <div class="form-group">
            <label for="text">Text:</label>
            <textarea class="form-control" id="text" th:field="*{text}" rows="5"></textarea>
        </div>

        <div class="form-group">
            <label for="file1">Preview Image:</label>
            <input type="file" class="form-control-file" id="file1" name="imageFile1">
        </div>

        <div class="form-group">
            <label for="file2">Second Image:</label>
            <input type="file" class="form-control-file" id="file2" name="imageFile2">
        </div>

        <div class="form-group">
            <label for="file3">Third Image:</label>
            <input type="file" class="form-control-file" id="file3" name="imageFile3">
        </div>

        <button type="submit" class="btn btn-primary">Save</button>
    </form>
</div>

<!-- Include Bootstrap JS and dependencies -->
<script>
    let imageIdToDelete = null;

    function showDeleteConfirmation(imageId) {  no usages  new *
        imageIdToDelete = imageId;
        $('#confirmDeleteImageModal').modal('show');
    }

    function deleteImage() {  Show usages  new *
        const csrfToken = document.querySelector('meta[name="_csrf"]').getAttribute('content');
        const csrfHeader = document.querySelector('meta[name="_csrf_header"]').getAttribute('content');

        fetch(`/blog/images/${imageIdToDelete}`, {
            method: 'DELETE',
            headers: {
                [csrfHeader]: csrfToken
            }
        }).then(response => {
            $('#confirmDeleteImageModal').modal('hide');
            if (response.ok) {
                location.reload();
            } else {
                alert('Failed to delete image');
            }
        }).catch(error => {
            alert('Failed to delete image');
        });
    }
</script>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
```

Рисунок Д.12 – Шаблон blog-edit.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <title>Login Page</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css"
          integrity="sha384-WskhaSGFgHYWDcbwN70/dfYBj47jz9qbsMId/iRN3ewGhXQFZCSftd1LZCfmhktB" crossorigin="anonymous">
</head>
<body>
<div class="container mt-5">
    <h4 class="mb-4">Login</h4>
    <form th:action="@{/login}" method="post">
        <div class="form-group row">
            <label for="inputEmail" class="col-sm-2 col-form-label">Email</label>
            <div class="col-sm-10">
                <input type="email" class="form-control" id="inputEmail" name="username" placeholder="Email" required>
            </div>
        </div>
        <div class="form-group row">
            <label for="inputPassword" class="col-sm-2 col-form-label">Password</label>
            <div class="col-sm-10">
                <input type="password" class="form-control" id="inputPassword" name="password" placeholder="Password" required>
            </div>
        </div>
        <div class="form-group row">
            <div class="col-sm-10 offset-sm-2">
                <button type="submit" class="btn btn-primary">Log in</button>
            </div>
        </div>
    </form>

    <div class="form-group row">
        <div class="col-sm-10 offset-sm-2">
            <a th:href="@{/registration}" class="btn btn-secondary mr-2">Registration</a>
            <a th:href="@{/}" class="btn btn-outline-secondary">Back To Main Page</a>
        </div>
    </div>
</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T" crossorigin="anonymous"></script>
</body>
</html>
```

Рисунок Д.13 – Шаблон login-page.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <title>Registration Page</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css"
          integrity="sha384-WskhaSGFgHYWDcbwN70/dfYBj47jz9qbsMId/iRN3ewGhXQFZCSftd1LZCfmhktB" crossorigin="anonymous">
    <script src="https://www.google.com/recaptcha/api.js" async defer></script>
</head>
<body>
<div class="container mt-5">
    <h4 class="mb-4">Registration</h4>
    <form th:action="@{/registration}" method="post">
        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="firstname">First Name</label>
                <input type="text" class="form-control" id="firstname" name="name" placeholder="First Name" required>
            </div>
            <div class="form-group col-md-6">
                <label for="surname">Surname</label>
                <input type="text" class="form-control" id="surname" name="surname" placeholder="Surname" required>
            </div>
        </div>

        <div class="form-group">
            <label for="nickname">Nickname</label>
            <input type="text" class="form-control" id="nickname" name="nickname" placeholder="Nickname" required>
            <small class="form-text text-muted">This name will be visible to everyone.</small>
        </div>

        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="email">Email</label>
                <input type="email" class="form-control" id="email" name="email" placeholder="Email" required>
            </div>
            <div class="form-group col-md-6">
                <label for="password">Password</label>
                <input type="password" class="form-control" id="password" name="password" placeholder="Password" required>
            </div>
        </div>

        <!-- Error Message Display -->
        <p th:if="${emailError}" th:text="${emailError}" class="text-danger"></p>
        <p th:if="${nickNameError}" th:text="${nickNameError}" class="text-danger"></p>
        <p th:if="${unknownError}" th:text="${unknownError}" class="text-danger"></p>
        <p th:if="${captchaError}" th:text="${captchaError}" class="text-danger"></p>

        <button type="submit" class="btn btn-primary">Register</button>
        <a th:href="@{/}" class="btn btn-secondary ml-2">Back To Main Page</a>
        <div class="d-flex justify-content-center mb-3">
            <div class="g-recaptcha" th:attr="data-sitekey=${@reCaptchaConfiguration.getSiteKey()}"></div>
        </div>
    </form>
</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T" crossorigin="anonymous"></script>
</body>
</html>
```

Рисунок Д.14 – Шаблон registration-page.html

```
<div class="container-fluid">
    <div class="row my-3">
        <div class="col-md-6">
            <form th:action="@{/}" class="form-inline">
                <label for="title" class="mr-2">Search by Title:</label>
                <input type="text" id="title" name="title" class="form-control mr-2" th:value="${title}" placeholder="Enter title"/>
                <button type="submit" class="btn btn-primary">Search</button>
            </form>
        </div>
        <div class="col-md-6">
            <form th:action="@{/}" method="get" class="form-inline float-md-right">
                <label for="sort" class="mr-2">Sort by:</label>
                <select id="sort" name="sort" class="form-control mr-2">
                    <option value="priceAscending">Price: Low to High</option>
                    <option value="priceDescending">Price: High to Low</option>
                    <option value="dateAscending">Date: Old to New</option>
                    <option value="dateDescending">Date: New to Old</option>
                </select>
                <button type="submit" class="btn btn-primary">Sort</button>
            </form>
        </div>
    </div>

    <div class="row">
        <div th:if="${#lists.isEmpty(announcements)}" class="col-12">
            <p>No announcements available</p>
        </div>

        <div th:each="announcement : ${announcements}" class="col-md-3 mb-4">
            <div class="card" style="width: 100%;">
                <img class="card-img-top"
                     th:if="${announcement.previewImageId != null}"
                     th:src="@{'/images/{id}'(id=${announcement.previewImageId})}"
                     alt="Announcement Image" style="max-height: 180px; object-fit: contain;">
                <img class="card-img-top"
                     th:if="${announcement.previewImageId == null}"
                     th:src="@{/photos/defaultPreviewImage.jpg}"
                     alt="Default Announcement Image" style="max-height: 180px; object-fit: contain;">
                <div class="card-body">
                    <h5 class="card-title" th:text="${announcement.title}"></h5>
                    <p class="card-text" th:text="${announcement.description}"></p>
                </div>
                <ul class="list-group list-group-flush">
                    <li class="list-group-item" th:text="'Price: ' + ${announcement.price} + ' ' + ${announcement.currency}"></li>
                    <li class="list-group-item">
                        <a th:href="@{'/user/' + ${announcement.user.id}}" th:text="${announcement.user.nickname}" style="text-decoration: none;"></a>
                    </li>
                    <li class="list-group-item" th:text="'Posted on: ' + ${#temporals.format(announcement.creationDate, 'dd.MM.yyyy HH:mm')}"></li>
                </ul>
                <div class="card-body">
                    <a th:href="@{'/details/' + ${announcement.id}}" class="btn btn-primary">See more</a>
                </div>
            </div>
        </div>
    </div>
</div>
```

Рисунок Д.15 – Шаблон main-page.html

```
<form th:action="@{'/{type}/{userChannel}/list'(type=${type}, userChannel=${userChannel.id})}"
      method="get" class="form-inline justify-content-center my-3">
    <div class="container">
        <div class="row justify-content-center">
            <input type="text" name="nickname" class="form-control w-100 col-md-6 mr-2" placeholder="Enter Nickname"
                   th:value="${nickname}">
            <button type="submit" class="btn btn-primary">Search</button>
        </div>
    </div>
</form>

<div class="container mt-5">
    <h3 th:text="${userChannel.nickname}"></h3>
    <p th:text="${type}"></p>

    <ul class="list-group mt-3">
        <li class="list-group-item" th:each="user : ${userList}">
            <a th:href="@{'/user/' + ${user.id}}" th:text="${user.nickname}"></a>
        </li>

        <li class="list-group-item text-center" th:if="${#lists.isEmpty(userList)}">
            <span th:text="'No ' + ${type} + ' found'"></span>
        </li>
    </ul>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Рисунок Д.16 – Шаблон subscriptions-list.html

```
<div class="container mt-5">
    <h3 class="mb-4">Changing Roles for <span th:text="${user.nickname}"></span></h3>
    <form th:action="@{/user/edit}" method="post">
        <div class="form-group">
            <div th:each="role : ${roles}">
                <div class="form-check">
                    <input type="checkbox" class="form-check-input" th:name="${role}" th:checked="${user.roles.contains(role)}" id="role-${role}" />
                    <label class="form-check-label" th:for="'role-' + ${role}" th:text="${role}"></label>
                </div>
            </div>
        </div>
        <input type="hidden" th:value="${user.id}" name="userId" />
        <button type="submit" class="btn btn-primary">Save</button>
    </form>
</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T" crossorigin="anonymous"></script>
```

Рисунок Д.17 – Шаблон user-change-role.html

```
<div class="container mt-5">
    <h4 class="mb-4">Edit Profile</h4>
    <form th:action="@{'/user/' + ${user.id} + '/edit'}" method="post">
        <div class="form-group">
            <label for="name">Name</label>
            <input type="text" class="form-control" id="name" name="name" th:value="${user.name}" placeholder="Enter your name" required>
        </div>

        <div class="form-group">
            <label for="surname">Surname</label>
            <input type="text" class="form-control" id="surname" name="surname" th:value="${user.surname}" placeholder="Enter your surname" required>
        </div>

        <div class="form-group">
            <label for="nickname">Nickname</label>
            <input type="text" class="form-control" id="nickname" name="nickname" th:value="${user.nickname}" placeholder="Nickname" required>
            <small class="form-text text-muted">This name will be visible to everyone.</small>
        </div>

        <div class="form-group">
            <label for="email">Email</label>
            <input type="email" class="form-control" id="email" name="email" th:value="${user.email}" placeholder="name@example.com" required>
        </div>

        <div class="form-group">
            <label for="password">Password</label>
            <input type="password" class="form-control" id="password" name="password" placeholder="Enter a new password">
        </div>

        <button type="submit" class="btn btn-primary">Save</button>
    </form>
</div>
```

Рисунок Д.18 – Шаблон user-edit.html

```
<div class="container mt-5">
    <h3>Users</h3>
    <form th:action="@{/users/list}" method="get" class="form-inline justify-content-center my-3">
        <input type="text" name="nickname" class="form-control w-50 mr-2" placeholder="Search by nickname" th:value="${nickname}">
        <button type="submit" class="btn btn-primary">Search</button>
    </form>

    <ul class="list-group mt-3">
        <li class="list-group-item" th:each="user : ${allUsers}">
            <a th:href="@{'/user/' + ${user.id}}" th:text="${user.nickname}"></a>
        </li>

        <li class="list-group-item text-center" th:if="${#lists.isEmpty(allUsers)}">
            <span>No users found</span>
        </li>
    </ul>
</div>
```

Рисунок Д.19 – Шаблон user-list.html

```
<div class="container mt-5">
    <h4>User Details</h4>

    <!-- Subscriber and Subscription Counts with Links -->
    <div class="row mt-4">
        <div class="col">
            <div class="card">
                <div class="card-body">
                    <div class="card-title">Subscriptions</div>
                    <h3 class="card-text">
                        <a th:href="@{'/subscriptions/' + ${user.id} + '/list'}" th:text="${numberOfSubscriptions}"></a>
                    </h3>
                </div>
            </div>
        </div>
        <div class="col">
            <div class="card">
                <div class="card-body">
                    <div class="card-title">Subscribers</div>
                    <h3 class="card-text">
                        <a th:href="@{'/subscribers/' + ${user.id} + '/list'}" th:text="${numberOfSubscribers}"></a>
                    </h3>
                </div>
            </div>
        </div>
    </div>

    <!-- Subscribe/Unsubscribe Button -->
    <div class="mt-4" th:if="${authenticatedUser.email != user.email}">
        <a th:if="${isSubscriber}" th:href="@{'/unsubscribe/' + ${user.id}}" class="btn btn-info">Unsubscribe</a>
        <a th:unless="${isSubscriber}" th:href="@{'/subscribe/' + ${user.id}}" class="btn btn-info">Subscribe</a>
    </div>

    <div class="mb-4">
        <p><strong>Name:</strong> <span th:text="${user.name}"></span></p>
        <p><strong>Surname:</strong> <span th:text="${user.surname}"></span></p>
        <p><strong>Nickname:</strong> <span th:text="${user.nickname}"></span></p>
        <p>
            <strong>Email:</strong>
            <span th:if="${authenticatedUser.email == user.email}" th:text="${user.email}"></span>
            <span th:unless="${authenticatedUser.email == user.email}">***</span>
        </p>
        <p><strong>Registration Date:</strong> <span th:text="${#temporals.format(user.registrationDate, 'dd.MM.yyyy HH:mm')}"></span></p>
    </div>
```

Рисунок Д.20 – Шаблон user-details.html

```
<h5>Announcements</h5>
<table class="table table-striped table-bordered">
    <thead>
    <tr th:if="${#lists.isEmpty(announcements)}">
        <th colspan="5" class="text-center">User has no Announcements</th>
    </tr>
    <tr th:unless="${#lists.isEmpty(announcements)}">
        <th>Title</th>
        <th>Category</th>
        <th>City</th>
        <th>Description</th>
        <th>Actions</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="announcement : ${announcements}">
        <td th:text="${announcement.title}"></td>
        <td th:text="${announcement.category}"></td>
        <td th:text="${announcement.city}"></td>
        <td th:text="${announcement.description}"></td>
        <td>
            <form th:action="@{'/details/' + ${announcement.id}}" method="get">
                <button type="submit" class="btn btn-sm btn-info">Details</button>
            </form>
        </td>
    </tr>
    </tbody>
</table>

<!-- Blogs Section -->
<h5>Blogs</h5>
<table class="table table-striped table-bordered">
    <thead>
    <tr th:if="${#lists.isEmpty(blogs)}">
        <th colspan="4" class="text-center">User has no Blogs</th>
    </tr>
    <tr th:unless="${#lists.isEmpty(blogs)}">
        <th>Title</th>
        <th>Creation Date</th>
        <th>Actions</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="blog : ${blogs}">
        <td th:text="${blog.title}"></td>
        <td th:text="${#temporals.format(blog.creationDate, 'dd.MM.yyyy HH:mm')}"></td>
        <td>
            <form th:action="@{'/blog/details/' + ${blog.id}}" method="get">
                <button type="submit" class="btn btn-sm btn-info">Details</button>
            </form>
        </td>
    </tr>
    </tbody>
</table>

<a th:if="${authenticatedUser.email == user.email}" th:href="@{'/user/' + ${user.id} + '/edit'}" class="btn btn-warning">
    Edit Profile
</a>
```

Рисунок Д.21 – Шаблон user-details.html

```
<div class="container-fluid">

    <div class="row my-3">
        <div class="col-md-6">
            <form th:action="@{/blogs}" class="form-inline">
                <label for="title" class="mr-2">Search by Title:</label>
                <input type="text" id="title" name="title" class="form-control mr-2" th:value="${title}" placeholder="Enter title"/>
                <button type="submit" class="btn btn-primary">Search</button>
            </form>
        </div>
    </div>

    <div class="row">
        <div th:if="${#lists.isEmpty(blogs)}" class="col-12">
            <p>No blogs available</p>
        </div>

        <div th:each="blog : ${blogs}" class="col-md-3 mb-4">
            <div class="card" style="width: 100%;">
                <img class="card-img-top"
                     th:if="${blog.previewImageId != null}"
                     th:src="@{'/images/{id}'(id=${blog.previewImageId})}"
                     alt="Blog Image" style="max-height: 180px; object-fit: contain;">
                <img class="card-img-top"
                     th:if="${blog.previewImageId == null}"
                     th:src="@{/photos/defaultPreviewImage.jpg}"
                     alt="Default Blog Image" style="max-height: 180px; object-fit: contain;">
                <div class="card-body">
                    <h5 class="card-title" th:text="${blog.title}"></h5>
                    <p class="card-text" th:text="${blog.text}"></p>
                </div>
                <ul class="list-group list-group-flush">
                    <li class="list-group-item">
                        <a th:href="@{'/user/' + ${blog.user.id}}" th:text="${blog.user.nickname}"
                           style="text-decoration: none;"></a>
                    </li>
                    <li class="list-group-item"
                        th:text="'Posted on: ' + ${#temporals.format(blog.creationDate, 'dd.MM.yyyy HH:mm')}"></li>
                </ul>
                <div class="card-body">
                    <a th:href="@{'/blog/details/' + ${blog.id}}" class="btn btn-primary">See more</a>
                </div>
            </div>
        </div>
    </div>
</div>
```

Рисунок Д.22 – Шаблон blogs-page.html

ДОДАТОК Е

Перечислення

```
import org.springframework.security.core.GrantedAuthority;

public enum Role implements GrantedAuthority {  11 usages  Tkachenko Vadym
    ROLE_USER, ROLE_ADMIN;  no usages

    @Override  Tkachenko Vadym
    public String getAuthority() {
        return name();
    }
}
```

Рисунок Е.1 – Перечислення Role

```
public enum UserStatus {  7 usages  Kabratigr\Vadidya
    EMAIL_EXISTS, NICKNAME_EXISTS, SUCCESS;  2 usages
}
```

Рисунок Е.2 – Перечислення UserStatus