Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Вороненка Святослава Вадимовича*
(ПІБ)

академічної групи *122-17-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка інформаційної системи для організації тематичних колективних блогів*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*доц. Касьяненко Л.В*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-17-2* *Вороненка Святослава Вадимовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інформаційної системи для організації тематичних колективних блогів*

затверджена наказом ректора НТУ «ДП» від *07.06.2021 р* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Спірінцев В.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Вороненко С.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 90 с., 38 рис., 0 табл., 3 дод., 20 джерел.

Об'єкт розробки: тематичні колективні блоги.

Мета кваліфікаційної роботи: Розробка інформаційної системи для організації тематичних колективних блогів.

У вступі розглядається сучасний стан проблеми, конкретизується мета кваліфікаційної роботи, актуальність та галузь її застосування, уточнюється постановка завдання.

У першому розділі розповідається про предметну галузь, визначено актуальність розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі порівнювалися наявні рішення, буда обрана платформа для розробки, виконано проєктування і розробка програми, описана робота програми, алгоритм і структура її функціонування, визначено вхідні і вихідні дані, охарактеризовані параметри технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні сайту, що надає можливість переглядати, створювати та оцінювати блоги.

Актуальність інформаційної системи для огранізації тематичних колективних блогів визначається великим попитом на актуальні новини за різною тематикою.

Список ключових слів: БЛОГ, ВЕБ-ДОДАТОК, ПУБЛІКАЦІЯ, ПРОГРАМА, БРАУЗЕР, СИСТЕМА, ЕОМ, ІНФОРМАЦІЙНА СИСТЕМА.

ABSTRACT

Explanatory note: 90 pages, 38 figures, 0 tables, 3 appendices, 20 sources.

Object of development: thematic collective blogs.

The purpose of the qualification work: Development of an information system for organization of thematic collective blogs.

The introduction considers the current state of the problem, specifies the purpose of the qualification work, relevance and scope of its application, clarifies the task.

The first section tells about the subject area, determines the relevance of development, formulates the problem, specifies the requirements for software implementation, technology and software.

The second section compares the available solutions, selects a platform for development, designed and developed the program, describes the program, algorithm and structure of its operation, identifies input and output data, describes the parameters of hardware.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical value is to create a site that allows you to view, create and evaluate blogs.

The relevance of an information system for organization of thematic collective blogs is determined by the high demand for current news on various topics.

Keywords: BLOG, WEB APPLICATION, PUBLICATION, PROGRAM, BROWSER, SYSTEM, COMPUTER, INFORMATION SYSTEM.

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ 10
1.1 Загальні відомості з предметної галузі 10
1.2. Призначення розробки та галузь застосування 19
1.3. Підстава для розробки 21
1.4. Постановка завдання 21
1.5. Вимоги до програми або програмного виробу 22
1.5.1. Вимоги до функціональних характеристик 22
1.5.2. Вимоги до інформаційної безпеки 22
1.5.3. Вимоги до складу та параметрів технічних засобів 22
1.5.4 Вимоги до інформаційної та програмної сумісності 23
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ 24
2.1. Функціональне призначення програми 24
2.2. Опис застосованих математичних методів 25
2.3. Опис використаних технологій та мов програмування 25
2.4. Опис структури програми та алгоритмів її функціонування 31
2.5. Обґрунтування та організація вхідних та вихідних даних програми 41
2.6. Опис роботи розробленого програмного продукту 42
2.6.1. Використані технічні засоби 42
2.6.2. Використані програмні засоби 42
2.6.3. Виклик та завантаження програми 44
2.6.4. Опис інтерфейсу користувача 45
РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ 56
3.1. Визначення трудомісткості розробки програмного забезпечення 56
3.2. Розрахунок витрат на створення програми 60

ВИСНОВКИ.....63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....64
ДОДАТОК А. КОД ПРОГРАМИ 66
ДОДАТОК Б. ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ.....89
ДОДАТОК В. ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ.....90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - програмне забезпечення;

ІТ - інформаційні технологии;

CSRF - Cross-site request forgery;

CSS - Cascading Style Sheets;

JS - JavaScript;

СКБД - Система керування базами даних;

SQL - structured query language;

AOP - Aspect-Oriented Programming;

MVC - Model-View-Controller;

HTTP - HyperText Transfer Protocol;

REST - Representational State Transfer;

JPA - Java Persistence API;

API - application programming interface;

БД - База данных;

ORM - Object-Relational Mapping;

XML - eXtensible Markup Language;

DTO - Data Transfer Object;

CRUD - create, read, update, delete;

ЕОМ - Електронна обчислювальна машина.

ВСТУП

Розроблена інформаційна система призначена для застосування у блогосфері.

В наш час розповсюдження інформації шляхом друку у газетах, транслюванням на радіо або телевізорах вже застаріле і іноді інформація може буду не актуальною, адже могло пройти багато часу з моменту події. На заміну цього приходять блоги, які дозволяють розповсюджувати в інтернеті корисну інформацію, новини, події у світі не тільки завдяки професійним журналістам, телекомпаніям або радіостанціям, а і звичайним людям, які могли стати свідками якоїсь події, чи тими, хто зіткнувся з якоюсь проблемою и хоче поділитися рішенням цього, або відомою людиною, яка ділиться моментами зі свого життя. У сучасному світі блоги набирають все більшої популярності, адже кожен з нас може почати вести блог на будь-яку тему и стати відомим у цій сфері, незалежно від віку, статі або раси. Вони дуже поширені і відрізняються не лише темою и мовою, на якій вони написані, а і навіть типом та функціями блога. Існування таких типів блогів як відеоблог може повністю замінити телевізійні трансляції, текстовий блог може замінити газети та журнали, а подкаст і блогкастинг - радіо. Перегляд блогів набагато зручніше старих способів, адже користувач не повинен чекати поки новини розпочняться, або газета вийде, він здатен у будь-який час отримати те, що він захоче, без особливих зусиль та очікувань. Блоги - це сучасний, зручний та швидкий спосіб отримання інформації в наш час.

У час інформаційних технологій майже усі люди використовують інтернет і для отримання інформації та відповіді на свої запитання вони звертаються до інтернет ресурсів, які в свою чергу можуть бути блогами. Записи там розташовані по порядку від самих нових. Кожен зареєстрований користувач може залишити свій запис або коментар по якій-небудь темі, котрий може допомогти отримати відповідь або проінформувати про щось.

Було прийнято рішення розробити систему для організації тематичних блогів. Даний веб додаток дозволить кожному бажаючому вести свої тематичні блоги і залишати до них коментарі. Він буде актуальний через те, що інформація - це дуже важливий ресурс, який потрібен людям і на це завжди буде попит, а за рахунок різноманітності кожен знайде щось для себе.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

Блог (англ. blog, від web log - мережевий журнал) - веб-сайт, головний зміст якого - доданні користувачами записи. Вони можуть містити у собі відгуки (коментарі) відвідувачів, тому є можливість спілкування з блогером під його блогом, це є великою перевагою над електронною поштою чи чатом. Блогери - це люди, які ведуть блоги.

класифікація блогів:

- По змісту - тематичний чи загальний.
- По автору чи авторам:
  - Особистий - блог, у якому записи ведуться власником свого блога.
  - "Примарний" - блог, у якому записи веде невідома суспільству персона.
  - Колективний чи соціальний - блог, який ведеться групою осіб за правилами, які встановлює власник чи модератор.
  - Корпоративний - блог, у якому записи ведуться співробітниками організації.
- По тематичному напрямку:
  - Комп'ютери, ПЗ – блоги, які посвячені сфері ІТ і всьому, що з нею пов'язано.
  - Політика - блоги, у яких розповідається про політичну діяльність, зазвичай такі блоги ведуть політичні лідери чи представники політичних партій.
  - Подорожі - блоги, у яких автори розповідають про поїздки у інші країни, дають поради, як проводити час у тій чи іншій країні.

- Освіта – блоги, у яких автори розповідають о темі освіти. У них обговорюють процес навчання та будь-які інші теми пов'язані з освітою.
- Мода – блоги, у яких люди розповідають про тренди і новинки у світі моди. Зазвичай такі блоги ведуть професійні стилісти та дизайнери, чи ті, хто захоплюється модою.
- Музика – блоги, які посвячений музиці, певному музичному напрямку, музикальному жанру чи стилю.
- Побут – блоги, у яких говорять про відносини між людьми, психології та про домашнє господарство – про те, що пов'язано з побутом та особистим життям.
- Спорт – блоги, у яких розповідають про спортивні досягнення, події і схожі с цим теми.
- Кіно – блоги, у яких люди пишуть рецензії та критику до різних кінофільмів.
- Здоров'я – блоги, які охоплюють різноманітні теми, які пов`язані із здоров'ям людини та його проблемами, про харчування, дієти, хвороби та про слідкування за своїм організмом.
- Бізнес – блоги, у яких розповідають про заробіток, інвестування та бінарні опціони.

- По наявності/виду мультимедії:
  - Текстовий блог – блог, у якому основний зміст це текст.
  - Фотоблог – блог, у якому основний зміст це фотографії.
  - Музичний блог – блог, у якому основний зміст це музичні файли.
  - Подкаст і блогкастинг – блог, у якому основний зміст надиктовується та подається у вигляді аудіофайлів.
  - Відеоблог – блог, основним змістом якого є відеофайли.
- По особливостям контенту:

- Контентний блог – блог, у якому публікується первісний текст автора.
- Мікроблог – блог, у якому публікуються короткі щоденні новини з життя блогера.
- Цитатний блог – блог, у якому переважно зміст складається з цитат інших блогів.
- Сплоґ – спам-блог.

Також блоги можна поділити на різні їх функції.

Комунікативна:

– Найвідоміша і найчастіша функція. Мета цієї функції – спілкування з друзями чи рідними та знайомства з новими людьми. Такий стиль можна помітити у таких відомих сайтів, як Фейсбук, Вконтакте, Живой Журнал і багато інших .

Самопрезентація:

– Така функція найчастіше зустрічається на особистих сайтах чи персональних сторінках. Зміст таких блогів повністю про автора, його захопленнях, твореннях чи професійних здібностях. Цей вид функції доволі часто використовують спеціалісти у будь-яких сферах діяльності, презентуючи себе та свої послуги.

Розвага:

– Дуже проста і популярна функція. Її мета дати читачу відпочити та розслабитися у вільний час. Зміст таких блогів – жарти, смішні картинки, розповіді та відеоролики. Користується попитом у молоді, людей з великою кількістю вільного часу чи з обмеженнями в інших засобах розваги. Такі блоги є величезним джерелом розважального контенту.

Мемуари:

– Функцію мемуарів використовують як щоденник, у яких роблять будь-які записи для того, щоб не забути про подробиці тих чи інших подій зі свого життя.

Психотерапія:

– Дуже не звичайна функція блога, але її доволі часто використовують. Такі блоги зазвичай для того щоб виплеснути свої емоції, сказати про свій стан, поскаржитися на життя та знайти рішення та підтримку зі сторони інших людей.

Найвідоміші тематичні блоги: SITE-UA, Українська Правда, Укрінформ, DTF, Хабр, SlashDot, CSS-Tricks. Існує десятки інших блогів і більшість з них мають такі характеристики:

– Дозволяють створювати свої публікації.
– Дозволяють коментувати та оцінювати публікації авторів, тим самим просувати їх в списки кращих.
– Дозволяють переглядати публікації за різною тематикою.
– Заохочують створення популярних та цікавих публікацій і активних блогерів.

Більшість блогів мають бали, які відображають наскільки ви успішний блогер і наскільки подобається людям ваша творчість. Їх люди ставлять під публікаціями та коментарями блогерів, але вони можуть бути і негативними.

Серед українських блогів можна виділити лише декілька дійсно непоганих, але і в них є ряд проблем, яких можна позбавитися і зробити краще.

Потрібно аналізувати проблеми сучасних тематичних блогів, для цього треба розглянути існуючі аналоги і визначити їх переваги та недоліки. Для аналізу були узяті такі проекти як SITE-UA і Хабр.

SITE-UA

Site-UA – це блог-платформа для публікацій записів та обговорень українською мовою.

Вона представляє собою українськомовний веб-сайт в форматі системи тематичних колективних блогів з елементами новинного сайту, створений для публікації новин та статей. Контент на сайті створюють не тільки користувачі, а і адміністрація сайту.

На сайті автори поділяються на три групи:

- Топ-блогери.
- Клуб.
- Sandbox.

Топ-блогери – автори, які декілька разів потрапляли у топ «Кращих статей», який публікується адміністрацією сайту. Кількість таких авторів близько 50 осіб. Якщо топ-блогер стає рідше потрапляти у топ, то він переходить у нову групу «Клуб». Статті топ-блогерів автоматично потрапляють у розділ «Вибір редакції» (рис. 1.1).

Клуб – автори, які хоча б один раз потрапляли у тижневий топ. Статті таких авторів публікуються у розділі «Клуб» і мають шанс потрапити у розділ «Вибір редакції». Крім того, їх видно на головній сторінці.

Sandbox – автори, які тільки почали писати свої статті і не мають жодної топової статті, але також мають шанс потрапити в «Вибір редакції». На користувачів з групою Sandbox діє спам-фільтр, він забороняє публікувати автору статті частіше ніж раз в 8 годин.

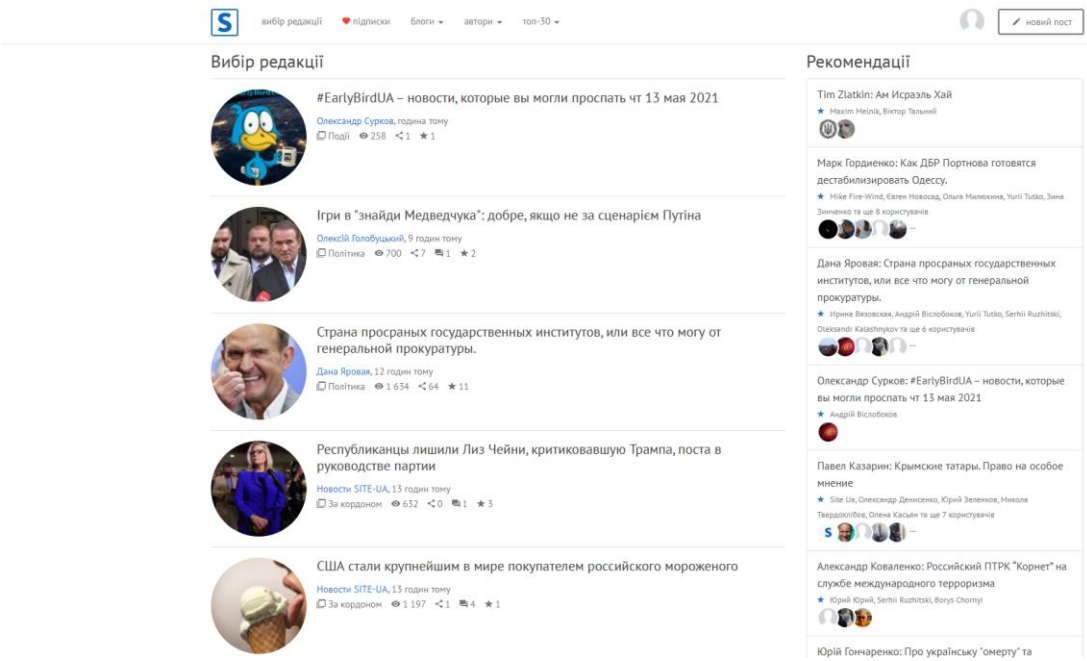

Рис. 1.1. «Вибір редакції» сайту Site-UA

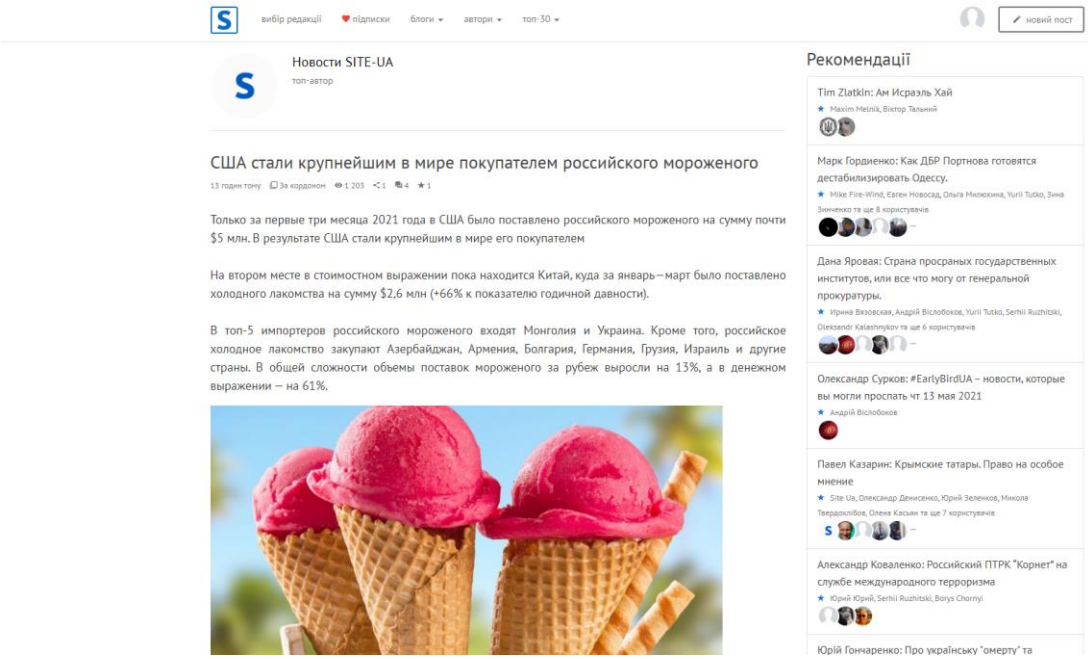

Рис. 1.2. Приклад сторінки з записом на сайті Site-UA

Кожну публікацію, приклад якої зображений на рис. 1.2., можна прокоментувати і поставити рекомендацію, але це доступно тільки тим, хто

авторизувався. Перегляди, коментарі та рекомендації роблять статтю та автора відомими і він починає отримувати новий статус.

Переваги:

- Безкоштовний доступ до усього контенту.
- У кожного є можливість стати відомим автором і потрапити в «Вибір редакції».

Недоліки:

- Реєстрація доступна лише тільки через соціальну мережу Facebook.
- Відсутність негативною оцінки, для більш точного розуміння якості публікації.
- Спам-блок для авторів з групою Sandbox, який заважає новим користувачам ділитися контентом.

На відміну від Site-UA розроблювана система буде кращою, унаслідувавши усі переваги і позбувшись недоліків, які були помічні у сайту. Покращення будуть зроблені за рахунок:

- додавання можливості реєстрації не тільки за допомогою одної соціальної мережі, а ще будуть і інші, до того ж користувач матиме змогу зареєструватися без жодних соціальних мереж.
- Кожну публікацію зареєстрований користувач зможе оцінити як позитивно, так і негативно. Це буде давати більше розуміння про якість публікації.
- Нові автори не будуть обмежені у створені публікацій, таке рішення допоможе новим користувачам швидше просуватися у топ.

Хабр

Хабр є одним з найвідоміших інформаційних сайтів у сфері IT.

«Він представляє собою російськомовний веб-сайт в форматі системи тематичних колективних блогів (іменованих хабами) (рис. 1.4) з елементами

новинного сайту, створений для публікації новин, аналітичних статей, думок, пов'язаних з інформаційними технологіями, бізнесом і інтернетом [1]». Контент (рис. 1.3) на сайті формується завдяки користувачам, які пишуть персональні або колективні блоги.

Система оцінки блогів або коментарів виконана у вигляді «Карми».

«Карма» - «основний рейтинг користувача або організації, дозволяє користувачам висловлювати ставлення один до одного. Змінювати карму можуть користувачі з «кармою» більше 5, при цьому зміна «карми» анонімна і не регулюється формальними критеріями. Високий рейтинг дозволяє створювати і писати в колективні блоги, змінювати «карму» іншим, додавати і писати про компанії. Також наявність значної «позитивної карми» необхідно для можливості відправки запрошень (інвайт). Негативний рейтинг накладає обмеження по частоті публікацій і коментарів. [2]». Підтримувати баланс допомагають карма і рейтинг. Карма ілюструє ставлення спільноти до користувача, а рейтинг показує його внесок у спільноту.

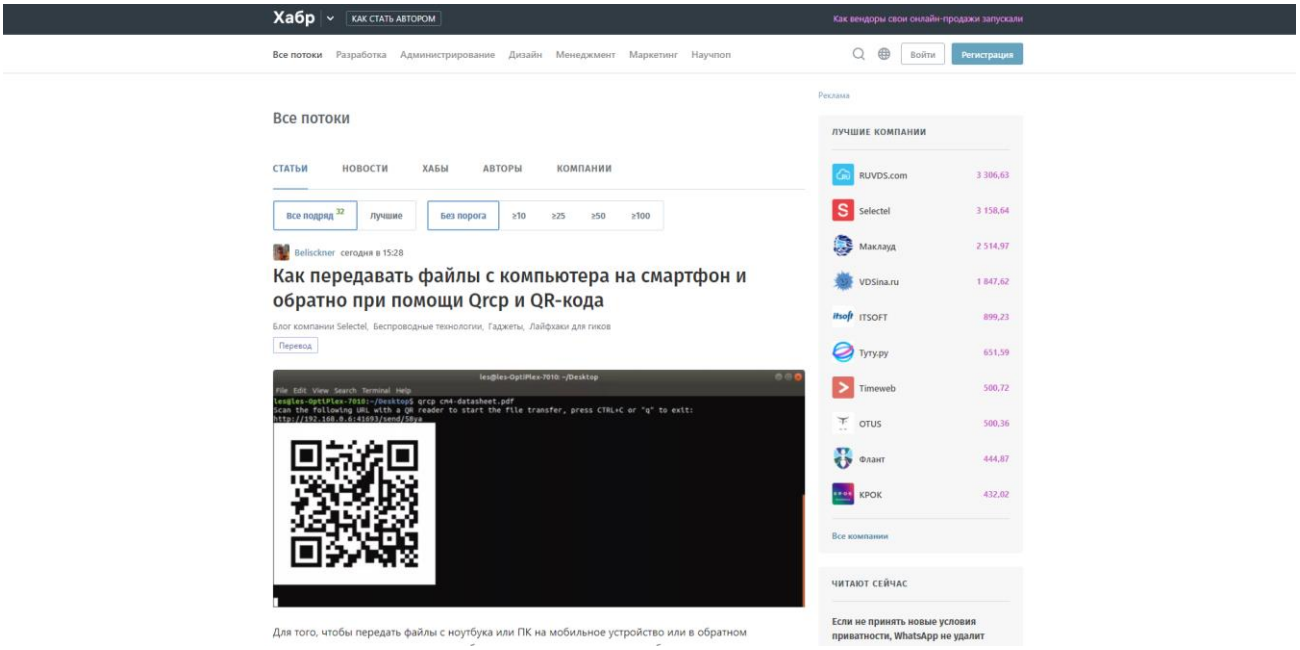

Рис. 1.3. Головна сторінка сайту Хабр

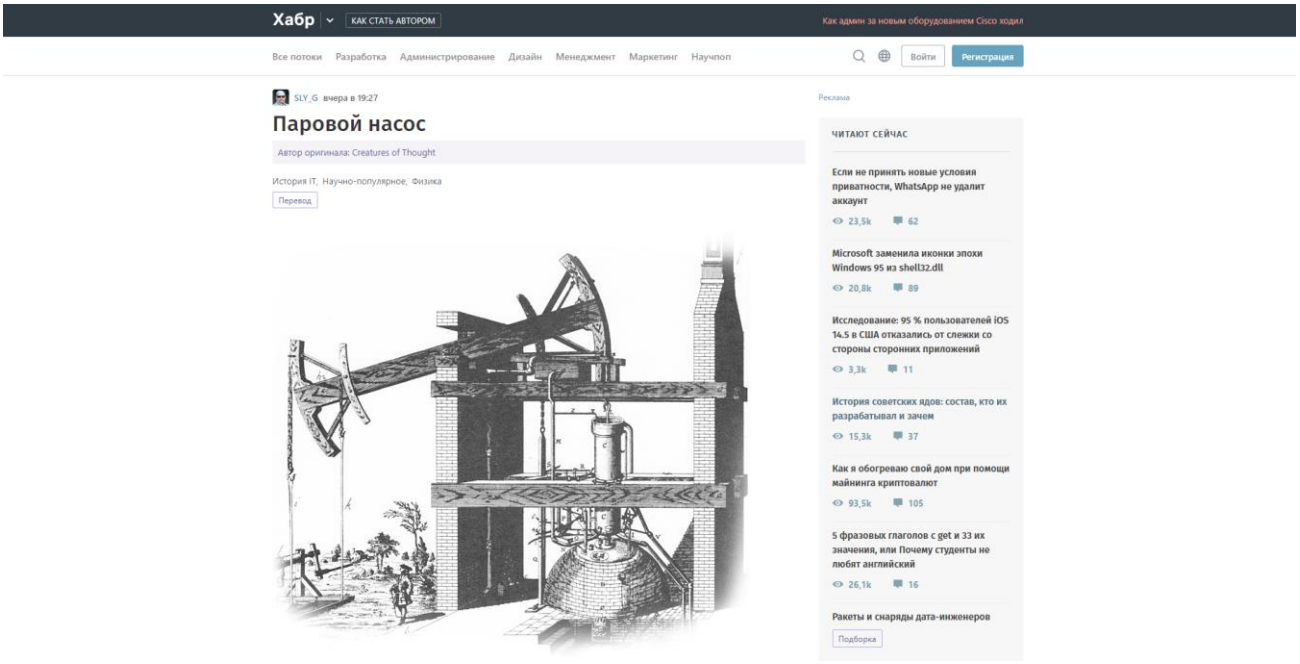

Рис. 1.4. Приклад сторінки з записом на сайті Хабр

До того ж, «На Хабрі усі рівні. Це активне, саморегулююче співтовариство, яке пропускає тільки якісний контент: цікаві пости плюсують і активно коментують, нецікаві - мінусують і залишають без уваги. спільноту [3]». Хоча рівність пропадає, якщо у користувача карма уходить в мінус і у нього зникає можливість швидко відповідати на коментарі та робити нові записи, це сильно заважає автору донести свої думки і створює перешкоду між співрозмовниками.

Переваги:

– Безкоштовний доступ до усього контенту.
– Матеріальна підтримка авторів за публікації, якщо їх добре оцінили читачі.
– Можливість повністю оцінити публікацію, за рахунок позитивної і негативної оцінки, та коментарів.

Недоліки:

– Обмеження спілкування та можливості коментувати у разі негативної карми, яку могли видати інші користувачі у разі незгоди з будь-чим. Це

дуже серйозний недолік, через який більшість людей перестають робити публікації на цьому сайті.

Для уникнення такої проблеми, розроблювана система не дозволить користувачам самим обмежувати інших в можливостях робити публікації і залишати коментарі. Замість цього користувачі можуть залишати скаргу, тоді модератор чи адміністратор будуть вирішувати питання.

1.2. Призначення розробки та галузь застосування

Даний веб-додаток має використовуватися людьми для того, щоб вони могли ділитися думками, розробками, новинами та обговорювати це.

Призначення розробки – створювання інформаційної системи для організації тематичних колективних блогів, яка дозволить створювати кожному користувачу свої записи, редагувати і видаляти їх. Також у кожного авторизованого є можливість оцінювати і коментувати публікації інших.

Зареєструвавшись, користувач має доступ до свого кабінету, головної сторінки, будь-якої публікації і створювання своїх публікацій. У особистому кабінеті можна побачити налаштування, де міняється ім'я або пароль, свої публікації, коментарі та користувачів, на котрих він підписався. На головній сторінці будуть відображені найкращі за неділю записи, список по популярності за різний проміжок часу, пошук по назвам публікацій або тегам, та список тем. Під записами інших є можливість залишати коментарі, ставити їм оцінку та самій публікації. У не зареєстрованого користувача є можливість тільки переглядати записи, без можливості створення або коментування та оцінювання інших.

В наш час інформаційних технологій обмін інформацією дуже затребуваний, тому розроблена система може бути корисною і буде використовуватися будь-ким. Читачі будуть спроможні знайти цікаві або корисні статті та публікації на будь-яку тему для вирішення своїх питань, чи для того, що дізнатися щось нове

та обговорити це. Автори, починаючи з новачків, закінчуючи спеціалістами у своїй сфері, можуть ділитися своїми думками та розробками, це допоможе їм розвиватися у цьому напрямку, або обговорити з зацікавленими у цій області людьми.

Враховуючи усю інформацію, була розроблена структура системи, яка відображена на рисунку 1.5.

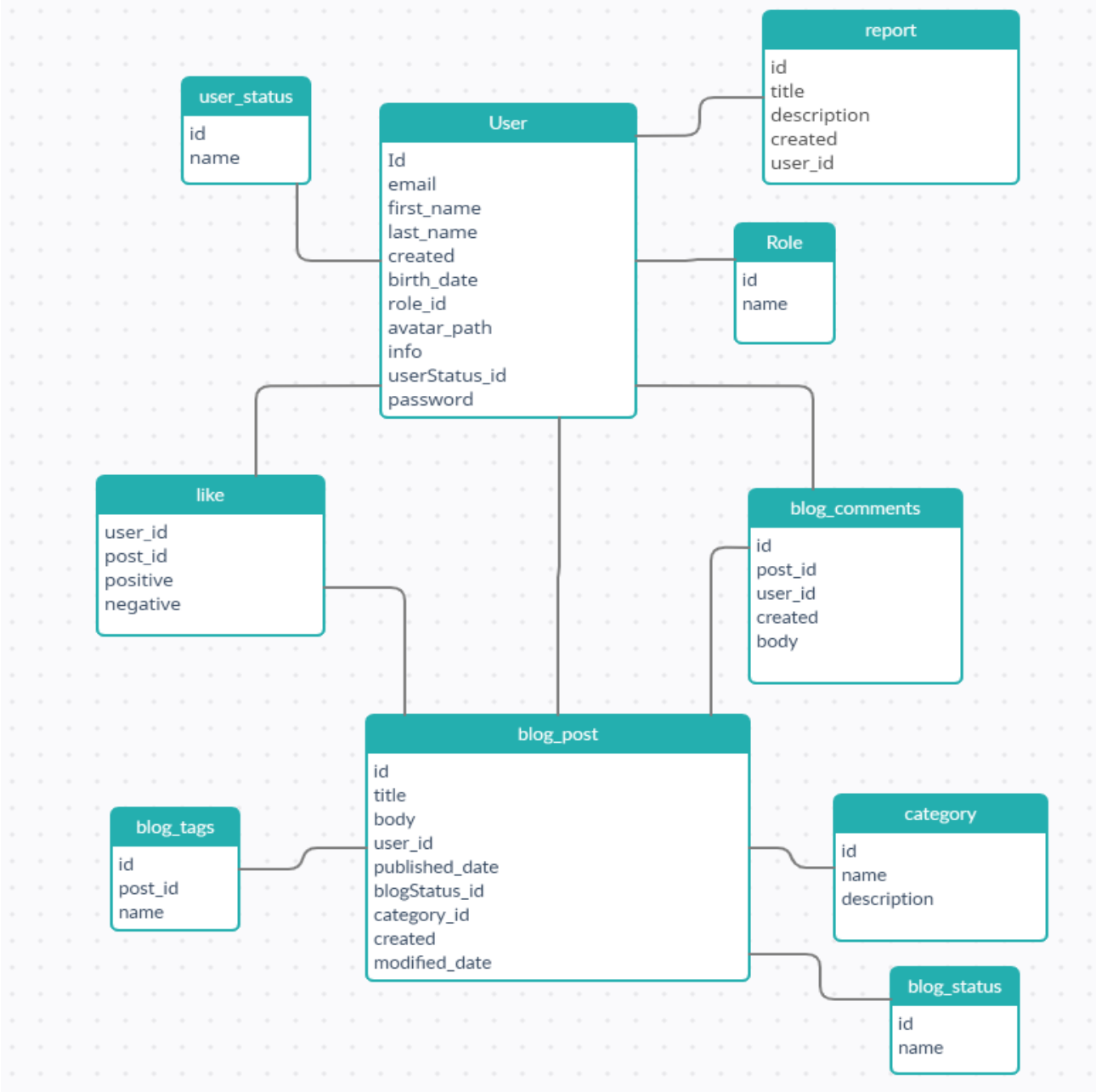

Рис. 1.5. Структура систем

1.3. Підстава для розробки

Відповідно до ОПП, згідно начального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (дипломний проект).

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

– ОПП за спеціальністю 122 «Комп'ютерні науки»;
– навчальний план та графік навчального процесу;
– наказ ректора Національного технічного університету «Дніпровська політехніка» № 317-с від 07.06. 2021 р;
– завдання на кваліфікаційну роботу на тему «Розробка інформаційної системи для організації тематичних колективних блогів».

1.4. Постановка завдання

Завданням кваліфікаційної роботи є розробка інформаційної системи для організації тематичних колективних блогів, яка дозволить зареєстрованим користувачам створювати свої записи на бажану тематику, коментувати та спілкуватися з іншими під будь-якими публікаціями. Основні можливості розробленої системи мають бути:

– Створювання публікацій будь-яким користувачем.
– Редагування публікацій автором чи адміністрацією сайту.
– Можливість перегляду публікацій.
– Коментування та оцінка будь-яких публікацій.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Веб-додаток повинен мати можливість реалізувати наступні функції:

– Додавання, редагування і видалення інформації в БД системи.
– Перегляд записів.
– Додавання коментарів та оцінки під записами.
– Отримання інформації про записи та самого автора.
– Відправка скарги на публікацію чи коментар.

Користувачі системи матимуть різні рівні доступу. Рівні доступу:

– Користувач.
– Модератор.
– Адміністратор.

1.5.2. Вимоги до інформаційної безпеки

Безпека веб-додатку буде реалізована за допомогою наступних вимог:

– Поділ користувачів на ролі і обмеження на можливості за допомогою них.
– Шифрування паролів у базі даних.
– Захист і надійність запитів за допомогою CSRF-токена.
– Валідація введених даних.
– Виведення коректних повідомлень у разі появи помилки.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для роботи з веб-застосунком користувачу потрібно мати систему, яка буде спроможна запустити і працювати з сучасними браузерами. Тому параметри ЕОМ повинні бути наступні:

– ЦП [CPU]: Intel Pentium 4/Athlon 64
– Диск [HDD/SSD]: 350 МБ
– Оперативна пам'ять [RAM]: 512 МБ
– Відеоадаптер [GPU]: 3D адаптер nVidia, Intel, AMD/ATI
– Відеопам’ять [VRAM]: 64 МБ

1.5.4 Вимоги до інформаційної та програмної сумісності

Для роботи інформаційної системи необхідно мати пристрій з наступними параметрами:

– Будь-яка з операційних систем: Windows, Linux, Mac OS, Android, IOS.
– Веб браузер: Google Chrome, Microsoft Edge, Mozilla Firefox, Safari.
– Доступ до мережі Інтернет.

Інформаційна система реалізована за допомогою мови програмування Java з використанням фреймворку Spring Framework та бібліотек. Інтерфейс для користувачів був створений за допомогою FreeMarker, CSS, JS. Для роботи і розробки системи необхідно мати наступні програми:

– Java EE Development Kit 8.
– Intellij IDEA/Eclipse.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення програми

Створена інформаційна система для організації тематичних колективних блогів має бути веб-застосунком, який дозволить будь-яким зареєстрованим користувачам робити свої публікації на бажану тему. Не зареєстрований користувач має змогу лише дивитися публікації.

Основні функції додатку:

- Авторизація та реєстрація користувача.
- Створення своїх публікацій на будь-яку доступну тему з використанням великої кількості інструментів для її оформлення та можливість редагування.
- Можливість поставити позитивну або негативну оцінку.
- Залишення коментарів під публікаціями.
- Залишення скарги у разі неприпустимого контенту або забороненого контенту.
- Відображення усіх публікацій, їх оцінки та коментарі, залишені іншими користувачами.
- Можливість підписатися на улюблених користувачів.
- Відображення особистої сторінки з можливістю редагувати свої дані та продивлятися сторінки інших.

2.2. Опис застосованих математичних методів

Особливості предметної галузі інформаційної системи не передбачає використання математичних методів, які могли бути застосовані для розробки веб-додатку створення колективних тематичних блогів.

2.3. Опис використаних технологій та мов програмування

Даний веб-додаток був розроблений у середовищі для розробки IntelliJ IDEA за допомогою таких технологій та мов програмування:

- Java (Основна мова програмування).
- JavaScript (Мова програмування).
- PostgreSQL (СУБД).
- Spring Boot (Основний фреймворк).
- Spring Security (Фреймворк для безпеки програми).
- Spring WEB (Фреймворк для створення веб-додатків).
- Spring Data JPA (Фреймворк для запитів у БД).
- FreeMarker (Шаблонізатор для створення HTML).
- CSS (Мова для створення зовнішнього виду сторінки).
- Liquibase (Система для керування БД).
- Gradle (Інструмент для збірки програми).

Java

Java – строго типізована об'єктно-орієнтована мова програмування. Особливістю цієї мови є те, що розроблені програми перетворюються у байт-код, котрий у свою чергу інтерпретується віртуальною машиною для будь-якої системи. Така особливість робить мову Java мультиплатформною і дозволяє

запускати програми у будь-якому середовищі, де є віртуальна машина Java. JVM – це програма, яка виконує код шляхом його перетворенням у байт-код.

JavaScript

JavaScript - це мова програмування, яка в основному використовується веб-браузерами для створення динамічного та інтерактивного інтерфейсу користувача. Більшість функцій та додатків, які роблять Інтернет необхідним для сучасного життя, створюються за допомогою JavaScript. Цю мову використовують разом з HTML та CSS для створення веб-додатків та веб-сторінок.

PostrgeSQL

PostrgeSQL – об’єктно-реляційна СКБД, котра використовую мову SQL. Ця мова призначена для управління даними, які знаходяться у базі даних та використовуються для користування цими даними. Реляційна база зберігає дані у таблицях, які пов’язані між собою. Вміст таблиці – стовпці та рядки. Кожний стовпець, їх ще називають атрибутами, визначає тип даних, а рядок – це повноцінний елемент бази даних у таблиці, який складається з набору стовпців. Стовпці, крім типу даних, також визначають чи є цей атрибут первинним або зовнішнім ключем, завдяки котрим таблиці пов’язані між собою.

Первинний ключ використовують як посилання для зв’язку з іншими таблицями. Такий ключ унікальний для кожної таблиці і може бути тільки один. Він може бути простим і складовим, різниця лише у тому, що простий застосовується лише до одного атрибуту, а складовий може визначатися за кількома одночасно.

Зовнішній ключ використовують для зв’язку між таблицями за допомогою посилань. Ці ключі застосовують до атрибутів таблиці і вони вказують на первинний ключ з іншої таблиці. Приклад роботи ключів зображений на рис. 2.1.

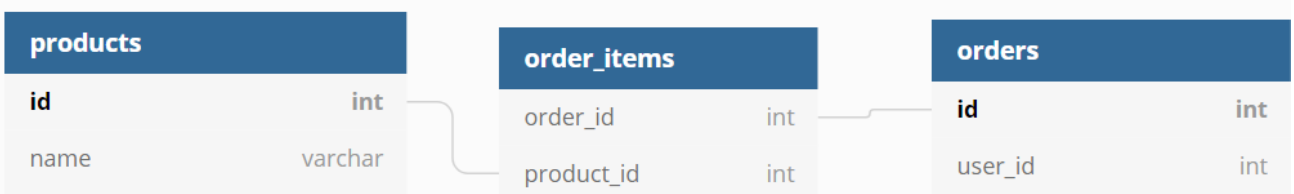

Рис. 2.1. Приклад використання ключів

Spring Boot

Spring Boot – це нащадок Spring Framework, який є удосконаленою версією. Особливістю Spring Boot є спрощення та прискорення розробки програм, за рахунок готових конфігурацій та програмних компонентів.

Основою Spring Framework є контейнер інверсії керування, який керує життєвим циклом об’єктів Java за допомогою рефлексії. Саме інверсія керування робить цей фреймворк настільки зручним і ефективним при розробці програм, адже цей принцип ООП зменшує кількість зв’язків компонентів програми між собою і дозволяє також досить легко розширювати можливості системи. Spring Framework також особливий за рахунок того, що він є набором або колекцією невеликих фреймворків. Вони можуть працювати як одночасно, так і незалежно один від одного.

Приклади таких фреймворків:

– Spring AOP Framework.
– Spring MVC Framework.
– Spring Security Framework.

Spring Security

Spring Secury - це фреймворк, який забезпечує аутентифікацію та авторизацію. Його обирають для захисту в усіх програмах на Java, які використовуються як веб застосунки. Особливістю цього фреймворку є не лише надійність, а і швидка та легка конфігурація з можливістю без зайвих зусиль розширити функціонал.

Spring WEB

Spring WEB – це фреймворк для створення веб-додатків або REST сервісів. Він працює за допомогою DispatcherServlet, який приймає усі HTTP-запити до нашої програми і опрацьовує їх. Приклад роботи зображений на рис. 2.2.

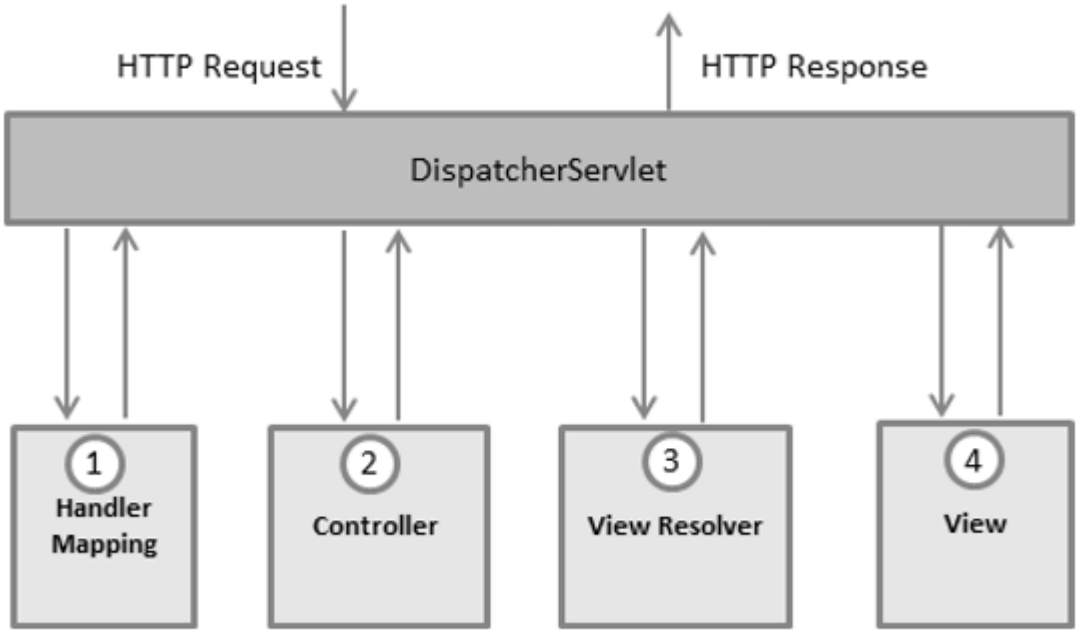

Рис. 2.2. Приклад роботи DispatcherServlet

Liquibase

Liquibase – це система, яка керує базою даних. Особливість і головна мета цієї системи – можливість повністю редагувати архітектуру вашої бази за допомогою файлів (changelogs), у яких знаходяться чейнджсети (changesets). Саме ці чейнджсети мають опис як повинна змінитися база. Кожний раз, коли запускається програма, Liquibase провіряє чи є вже такий чейнджсет у базі чи ні, якщо його немає, то його зміни застосовуються. Важлива особливість у тому, що коли чейнджсет застосував свої зміни, то його не можна більше редагувати, таким

чином ми маємо набор чейнджлогів (рис. 2.3), у яких бачимо історію змін бази даних.

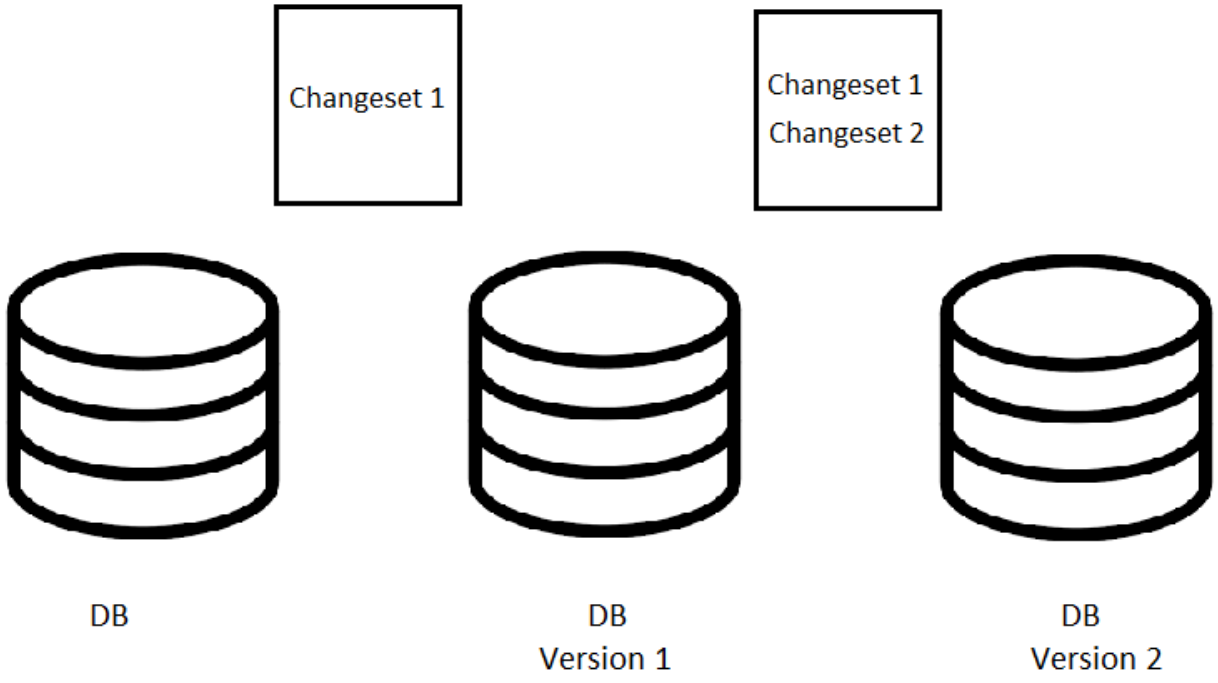

Рис. 2.3. Приклад роботи changesets

Spring Data JPA

Java Persistence API (JPA) – це специфікація Java EE, система для управління і збереження Java об’єктів у таблицях реляційних БД, ORM. Spring Data JPA – це покращена версія JPA, яка прискорює розробку програм, що пов’язані з базами даних, за рахунок вже готового функціоналу для роботи з об’єктами і відсутності великої кількості шаблонного коду для простих і постійних запитів до бази, але можливість самостійно писати запити залишилась.

FreeMarker

FreeMarker – це шаблонізатор і бібліотека Java для створення текстових документів, наприклад як HTML або XML. Він здатний створити документ з даними, які приходять з серверу шлях підстановки цих даних у документ замість спеціальних виразів, це зображено на рис. 2.4. До того ж FreeMarker здатен не

лише підставляти дані, а і виконувати прості логічні операції та працювати з колекціями об’єктів, розташовуючи будь яку їх кількість замість невеликих виразів.

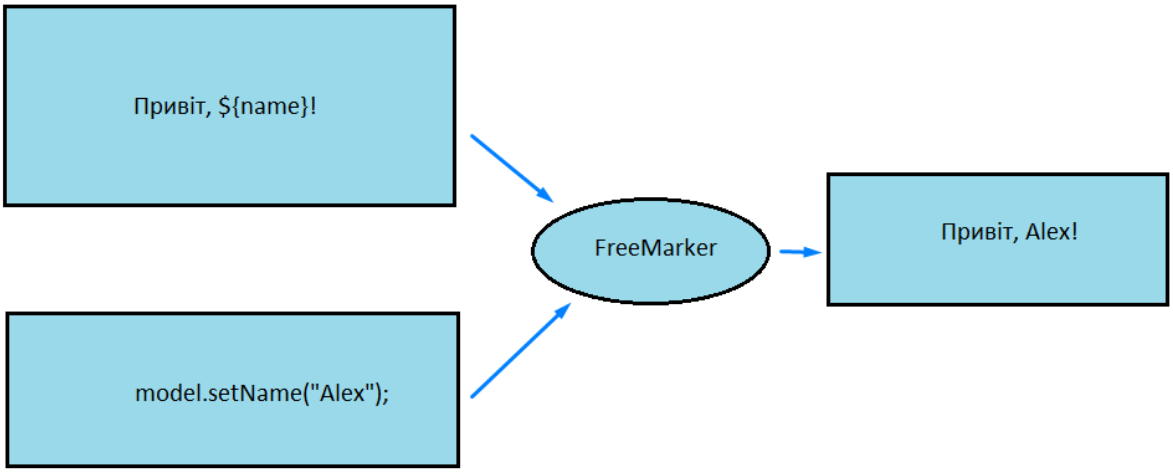

Рис 2.4. Приклад роботи FreeMarker

CSS

CSS – це мова для створення зовнішнього виду сторінки. Її використовують для того, щоб відокремлювати ту частину коду, яка відповідає за зовнішній вид, від самої веб-сторінки. CSS здатен змінити шрифти, верстку і будь-які елементи, що бачить користувач. Завдяки селекторам можна визначати якому елементу буде застосований стиль.

Gradle

Gradle - це сучасний інструмент автоматизації, який використовується при розробці програмного забезпечення для автоматизації побудови проектів. Інструмент популярний для створення будь-якого програмного забезпечення та великих проектів. Gradle включає плюси Ant і Maven і приборкає мінуси обох.

2.4. Опис структури програми та алгоритмів її функціонування

Зовнішній вид логічної структури веб-додатку виглядає наступним чином:

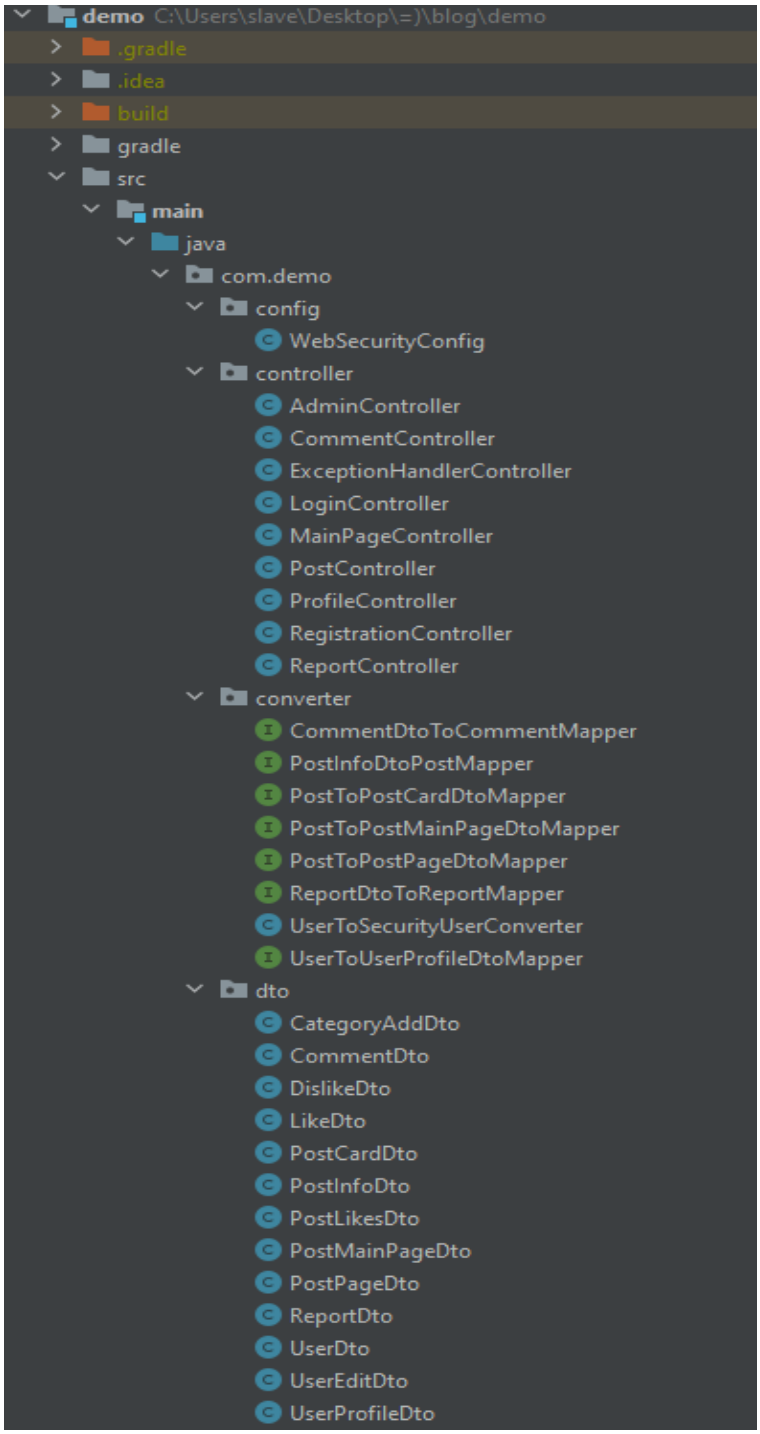

Рис. 2.5. Пакети: config, controller, converter, dto

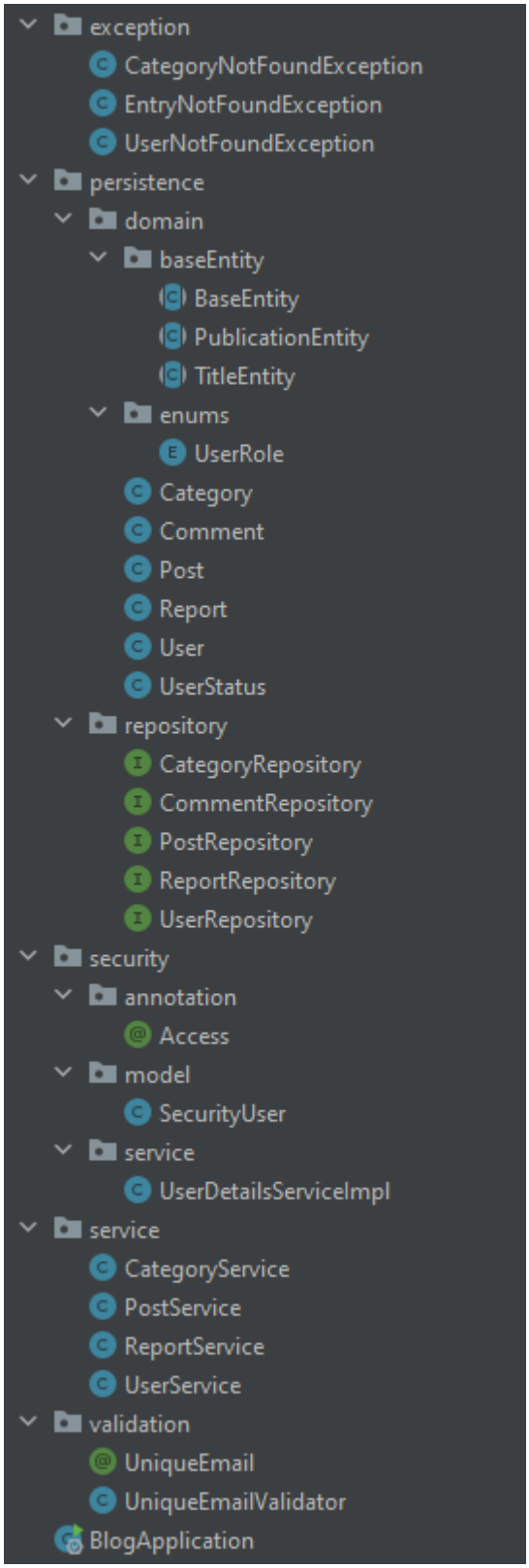

Рис. 2.6. Пакети: exception, persistence, security, service, validation

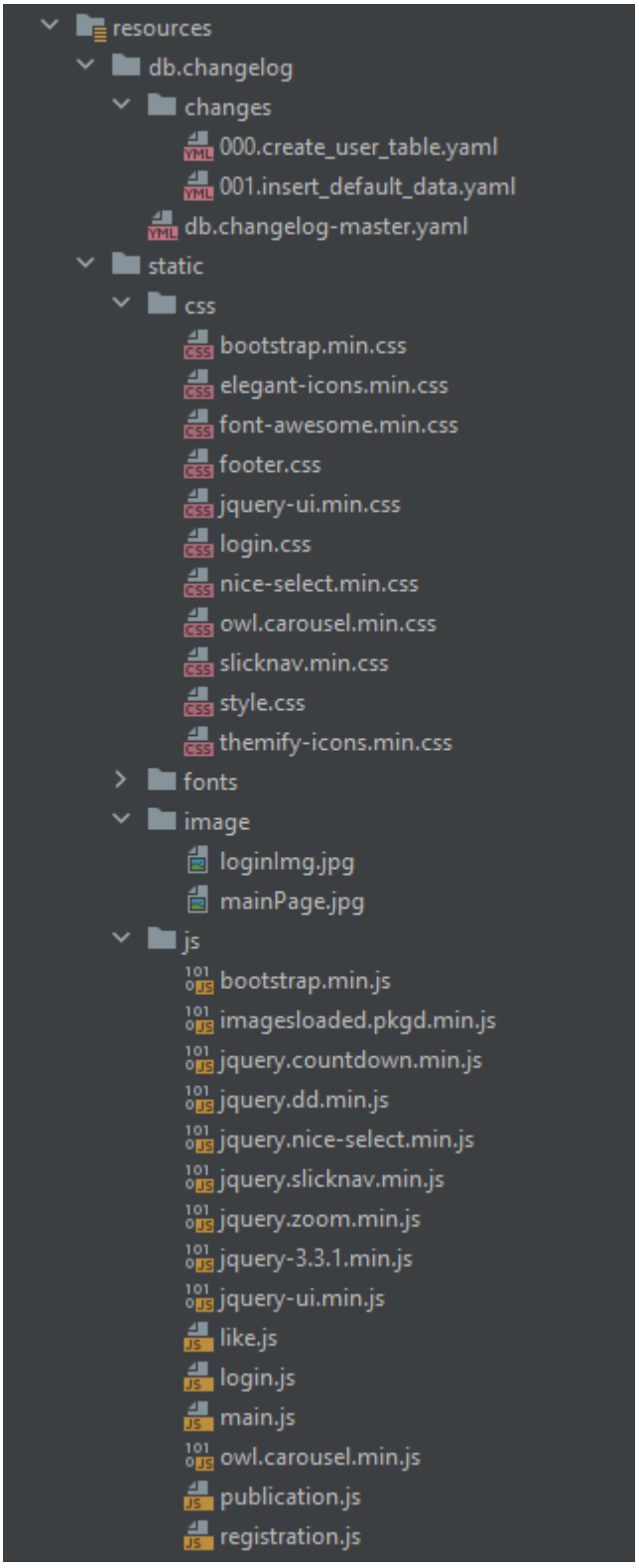

Рис. 2.7. Пакети: db.changelog, static

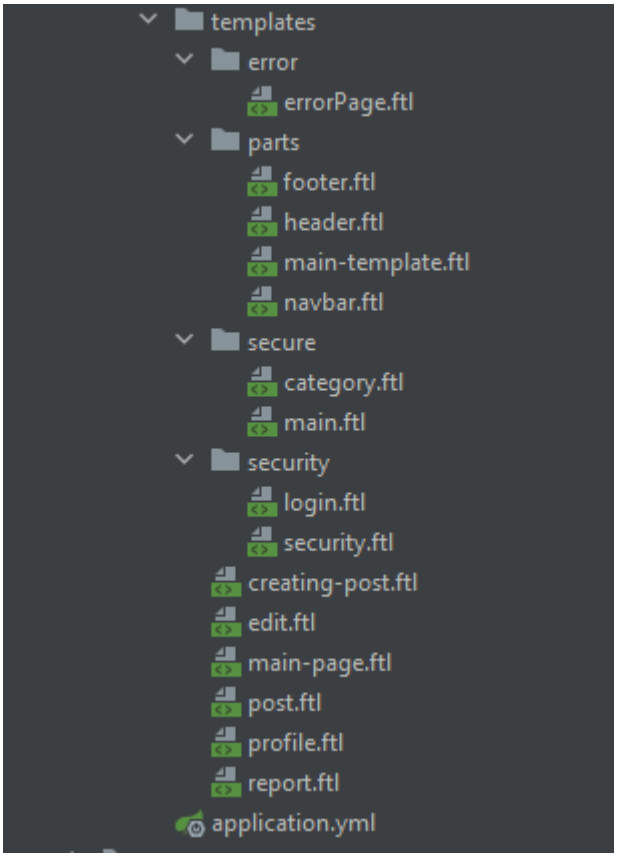

Рис. 2.8. Пакет tempates

Структуру можна поділити на дві основні частини, які знаходяться у наступних пакетах:

– java;
– resources.

У пакеті Java знаходиться вся backend частина, яка логічно поділена на окремі компоненти:

– config;
– controller;
– converter;
– dto;
– exception;
– persistence;

– security;
– service;
– validation.

У пакет config знаходяться конфігураційні файли програми. За рахунок використання Spring Boot замість простого Spring Framework кількість конфігураційних файлів значно менша, тому єдиний конфігураційний файл відповідає за шифровку паролів та аутентифікацію і авторизацію користувачів.

Пакет controller містить класи, у яких є інформація за те, яку адресу він обробляє і які дані повинен повернути або обробити.

Пакет converter допоміжний, класи та інтерфейси якого дозволяють перетворювати звичайні об’єкти у DTO класи та навпаки.

У пакеті dto знаходяться Data Transfer Object класи, головна задача яких – перенесення даних між шарами програми і виділення лише основних полів.

Пакет exception містить класи, які потрібні для того, щоб у разі якоїсь помилки програми або отриманих некоректних даних, програма не припиняла свою роботу і правильно відреагувала на ці події. Класи цього пакету використовуються у місцях, де може виникнути помилка або з’явитися некоректні дані, тоді вони активізуються, повідомляють про це, і спеціальний клас, ExceptionHandlerController, який оброблює усі помилки, повідомляє про це користувача.

Пакет persistence містить ще два пакета: domain та repository. Перший відповідає за сутності, які пов’язані з таблицями бази даних та їх зв’язки між собою. Repository містить інтерфейси, що дозволяють отримати доступ до бази за рахунок цих сутностей, але якщо подивитися у них, то більшість буде пустими. Це особливість Spring Data JPA, яка надає можливість користуватися основними CRUD операціями для управління базою даних, але без реалізації з нашої сторони, проте завжди є можливість створити свій метод і явно вказати який запит у базу він повинен зробити.

Пакет security містить спеціальні класи для Spring Security, які допомагають ідентифікувати користувача у системі. Також мається розроблена анотація для того, щоб обмежувати доступ к адресам в залежності від ролі користувача.

Пакет service відповідає за всю бізнес логіку програми. Цей шар програми один з найважливіших і використовується майже при кожному запиті до системи.

У пакеті validation знаходяться валідатори, які перевіряють ті дані на які ми поставимо перевірку.

Класс BlogApplication – головний клас програми, який містить лише один метод main, з якого починається робота веб-додатку.

Пакет resources містить у собі конфігураційний файл для підключення до бази даних та пакети з frontend частиною, структура яких наступна:

– db.changelog;
– static;
– templates.

db.changelog потрібен для Liquibase, котрий перевіряє кількість чейнджсетів і у разі, якщо з’явилися нові – застосовує їх до бази даних.

Пакети static та tempaltes відповідають за frontend. У пакеті static знаходяться усі CSS та JS файли, до того ж там зберігаються такі допоміжні файли як картинки або шрифти, а пакет templates містить у собі шаблони для HTML сторінок.

application.yml – файл, у якому знаходиться підключення до бази даних, конфігурація для неї і допоміжні налаштування для шаблонізатора FreeMarker.

Алгоритм роботи веб-додатку зображений на рис. 2.9.

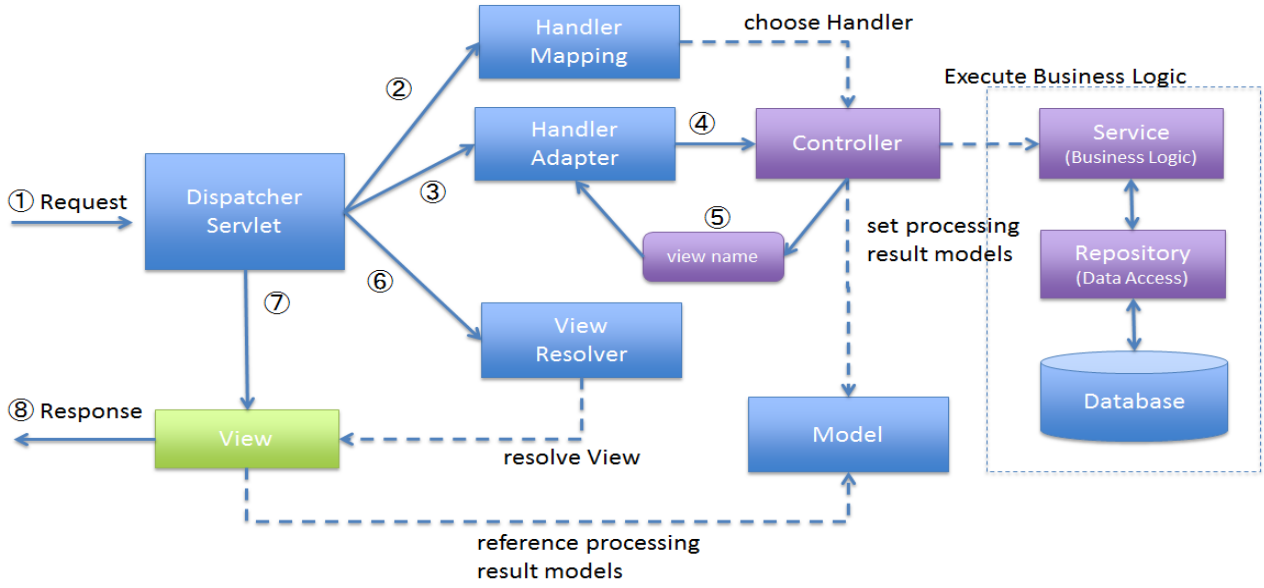

Рис 2.9. Алгоритм роботи системи

Будь-який запит, що адресується до нашої програми, потрапляє до Dispatcher Servlet. Він визначає контролер, який повинен обробити запит. Далі контролер викликає сервіс, якщо це необхідно, у котрому обробляються дані, що прийшли з контролера. Сервіс, рівень бізнес логіки, оброблює дані що прийшли з контролера і викликає репозиторій. Рівень репозиторія містить у собі інтерфейси для роботи з базою даних, але кожен інтерфейс працює і робить запити у рамках однієї сутності для кожного репозиторія, тому сервіс, як правило, має набір таких репозиторіїв.

Коли контролер отримав усі дані, які запитував у сервісів, їх кладуть у модель під різними ідентифікаторами. Модель необхідна для передачі даних у шаблон для створення веб сторінки. Після формування моделі контролер відправляє інформацію про назву документа, який потрібно відобразити користувачу, тоді і починається формування сторінки с даними з моделі.

Всю інформацію програма зберігаю у реляційній базі даних PostgreSQL структура зображена на рис. 2.10.

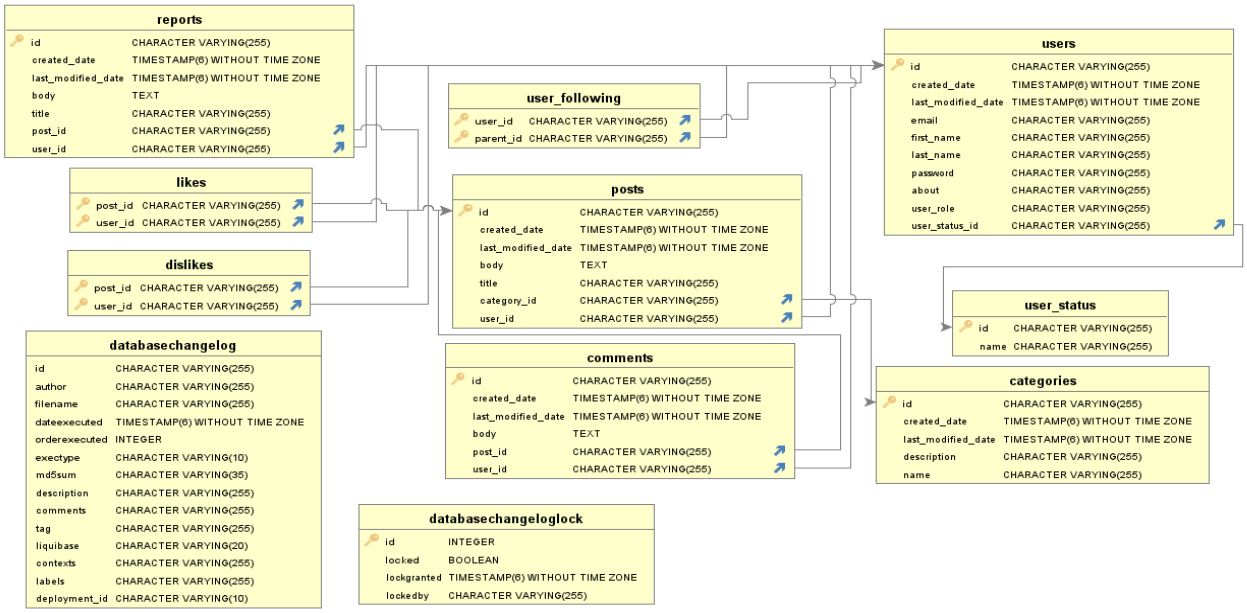

Рис 2.10. База даних

Основна таблиця у БД – «posts» (рис. 2.11). Вона зберігає інформацію о публікації і використовується у інших таблицях як зовнішній ключ.

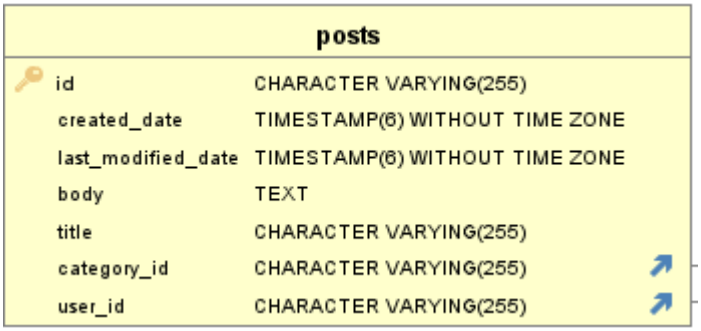

Рис 2.11. Таблиця «posts»

Таблиця «users» (рис. 2.12) зберігає особисту інформацію про користувача, його роль та статус.

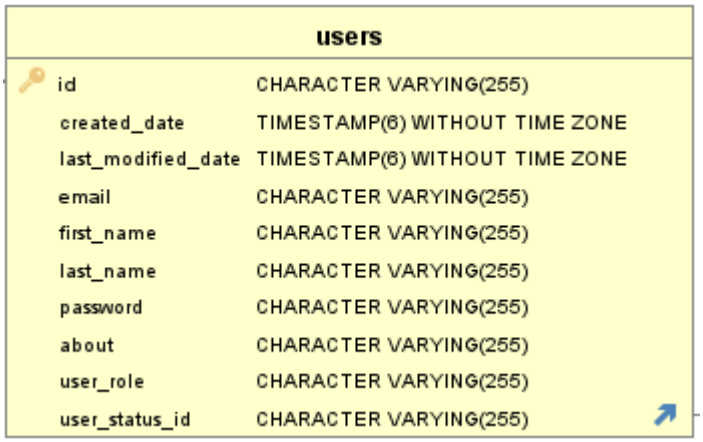

Рис. 2.12. Таблиця «users»

Таблиця «user_status» (рис. 2.13) потрібна для визначення статусу користувача, наприклад заблокований користувач або ні.

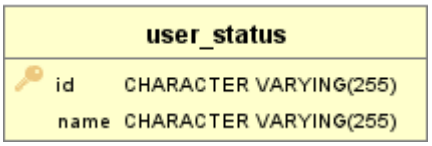

Рис. 2.13. Таблиця «user_status»

Категорії для сортування з їх описом зберігаються у «categories» (рис. 2.14)

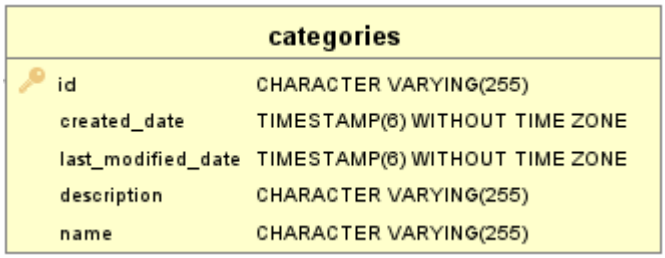

Рис. 2.14. Таблиця «categories»

Для збереження інформації о підписках створена таблиця «user_following» (рис. 2.15)

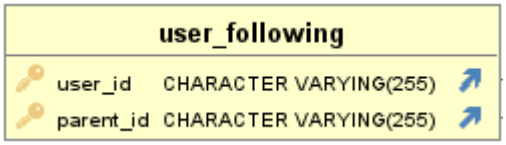

Рис. 2.15. Таблиця «user_following»

Дві таблиці «likes» (рис. 2.16) та «dislikes» (рис. 2.17) необхідні для визначення оцінки, яку користувач поставив під публікацією. На програмному рівні зроблено так, щоб один користувач не мав змогу знаходитися в обох таблицях одночасно, тому його знаходження у таблиці визначає його оцінку.

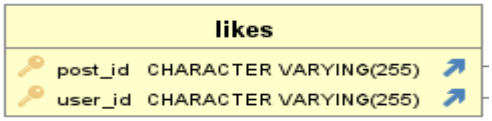

Рис 2.16. Таблиця «likes»

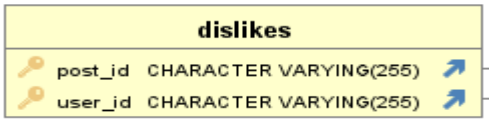

Рис 2.17. Таблиця «dislikes»

Таблиця «comments» (рис. 2.18) потрібна для того, щоб зв'язати коментарі з публікацією та користувачем .

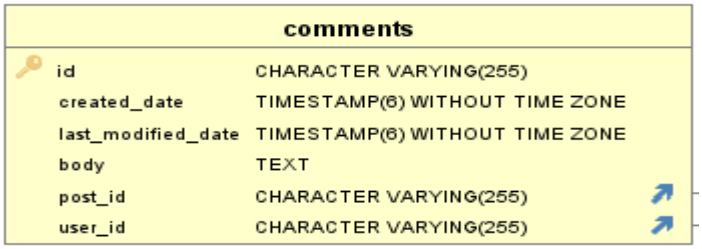

Рис 2.18. Таблиця «comments»

Для зберігання скарг на публікації створена таблиця «reports» (рис. 2.19), основний зміст якої – поле з заголовком та змістом.

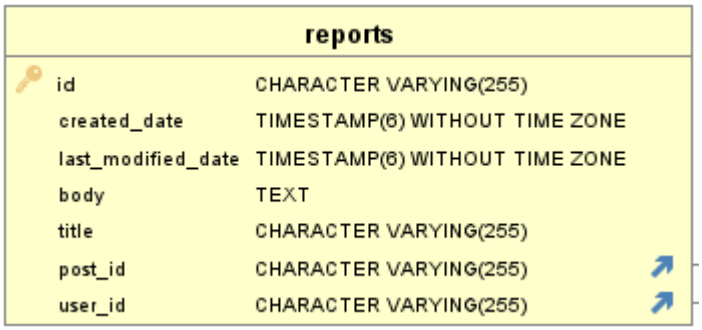

Рис 2.19. Таблиця «reports»

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Згідно функціоналу, який представлений у веб-додатку, вхідні та вихідні дані програми наступні.

Вхідні дані поділяються на два види:

– при авторизації або реєстрації;

– при створенні публікації;

При авторизації або реєстрації логін, що є поштою, валідується на те, щоб він дійсно був поштою, за рахунок спеціального шаблона, а пароль шифрується за допомогою BCryptPasswordEncoder.

При створенні публікації вхідними даними є:

– Заголовок публікації.

– Зміст.

– Категорія, яка пропонується у випадаючому списку.

Вихідними виступають дані, які приходять з БД по запиту користувача.

– Список публікацій.

– Повна інформація по публікації.

– Коментарі.
– Інформація про користувача.

2.6. Опис роботи розробленого програмного продукту

2.6.1. Використані технічні засоби

Для роботи з інформаційною системою, користувачу потрібно мати можливість працювати з сучасними браузерами, тому мінімальні характеристики ЕОМ повинні бути наступні:

– ЦП [CPU]: Intel Pentium 4/Athlon 64
– Диск [HDD/SSD]: 350 МБ
– Оперативна пам'ять [RAM]: 512 МБ
– Відеоадаптер [GPU]: 3D адаптер nVidia, Intel, AMD/ATI
– Відеопам’ять [VRAM]: 64 МБ

2.6.2. Використані програмні засоби

Для розробки веб-застосунку було використане наступне програмне забезпечення:

– Intellij IDEA.
– pgAdmin.
– Git, GitLab.

Intellij IDEA

Intellij IDEA – це інтегроване середовище розробки (IDE), що у багатьох випадках використовують для Java. Особливістю є те, що це середовище призначене для мов JVM і підтримує розробку на таких мовах як:

– Java.
– Kotlin.
– Scala.
– Groovy.

pgAdmin

pgAdmin – це дуже зручний інструмент з графічним інтерфейсом для керування СКБД PostgreSQL. Особливість у тому, що цей інструмент – вебдодаток, тому при збережені будь-якої інформації у БД, вона відправиться на сервер і її побачать інші користувачі, які підключені до БД.

Git та GitLab

При розробці програми використовувався Git. Це система для керування версіями файлів і можливості зрочної командної роботи. Принцип роботи зображен на рисунку 2.11. Кожне збереження файлів у Git називається коммітами і у нього є свій ідентифікатор. З таких коммітів створюється гілка, яка відображує історії змін проекту. Репозиторій може бути з великої кількості гілок, які у результаті зливаються в одну, основну гілку. Такий принцип роботи дуже ефективний, особливо при роботі в команді, коли розробники можуть паралельно роботи свої задачі, не заважаючи один одному, а в кінці отримати результат спільних зусиль.

GitLab – це сервіс для розміщення git-репозиторіїв. Він дозволяє зручно користуватися git, відстежувати помилки, аналізувати код та об’єднувати команди розробників. Приклад гілок зображений на рис. 2.20.

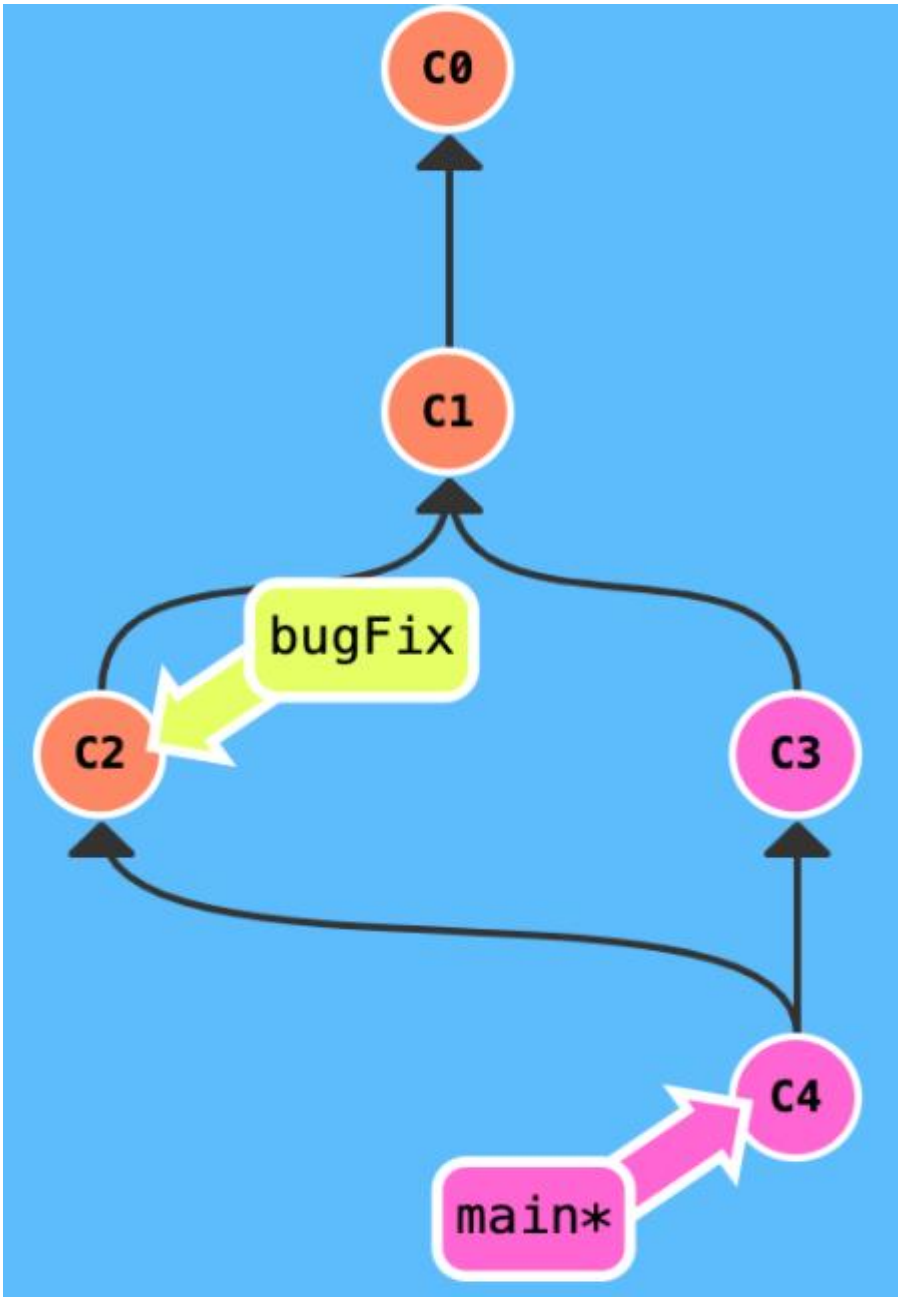

Рис 2.20. Система гілок і коммитів

2.6.3. Виклик та завантаження програми

Для користування інформаційною системою необхідно мати ЕОМ з мінімальними характеристиками, які потребують сучасні браузери та доступ до

інтернету. Отримати доступ і користуватися можна буде перейшовши по адресі вебдодатка у браузері без додаткових установок.

2.6.4. Опис інтерфейсу користувача

Робота з веб-додатком починається зі сторінки логіна, який видно на рис. 2.21.

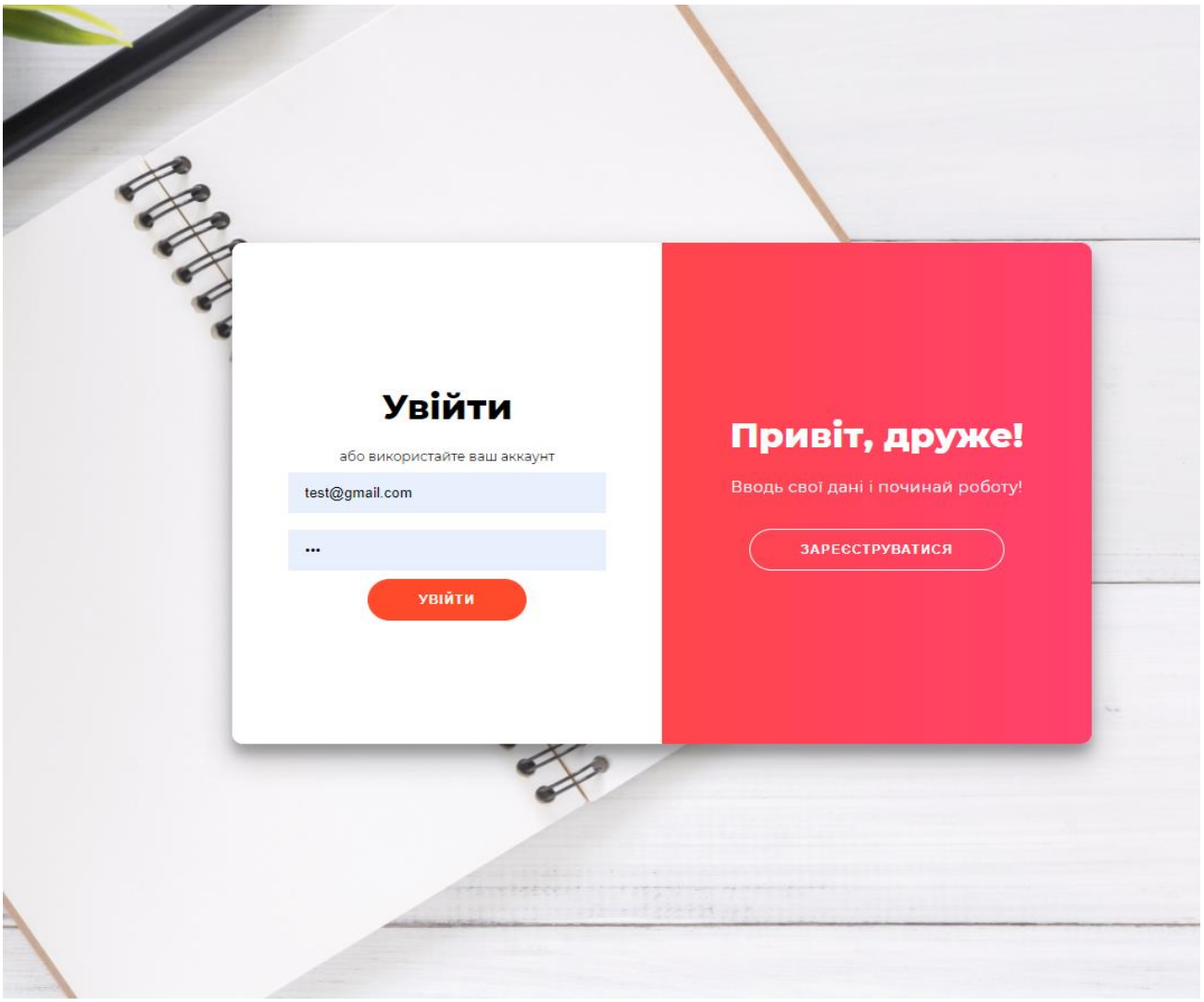

Рис 2.21. Сторінка логіна

У разі, якщо користувач не має свого аккаунта, то він може створити новий натиснувши кнопку “зареєструватися” і тоді буде доступна форма для реєстрації (рис. 2.22).

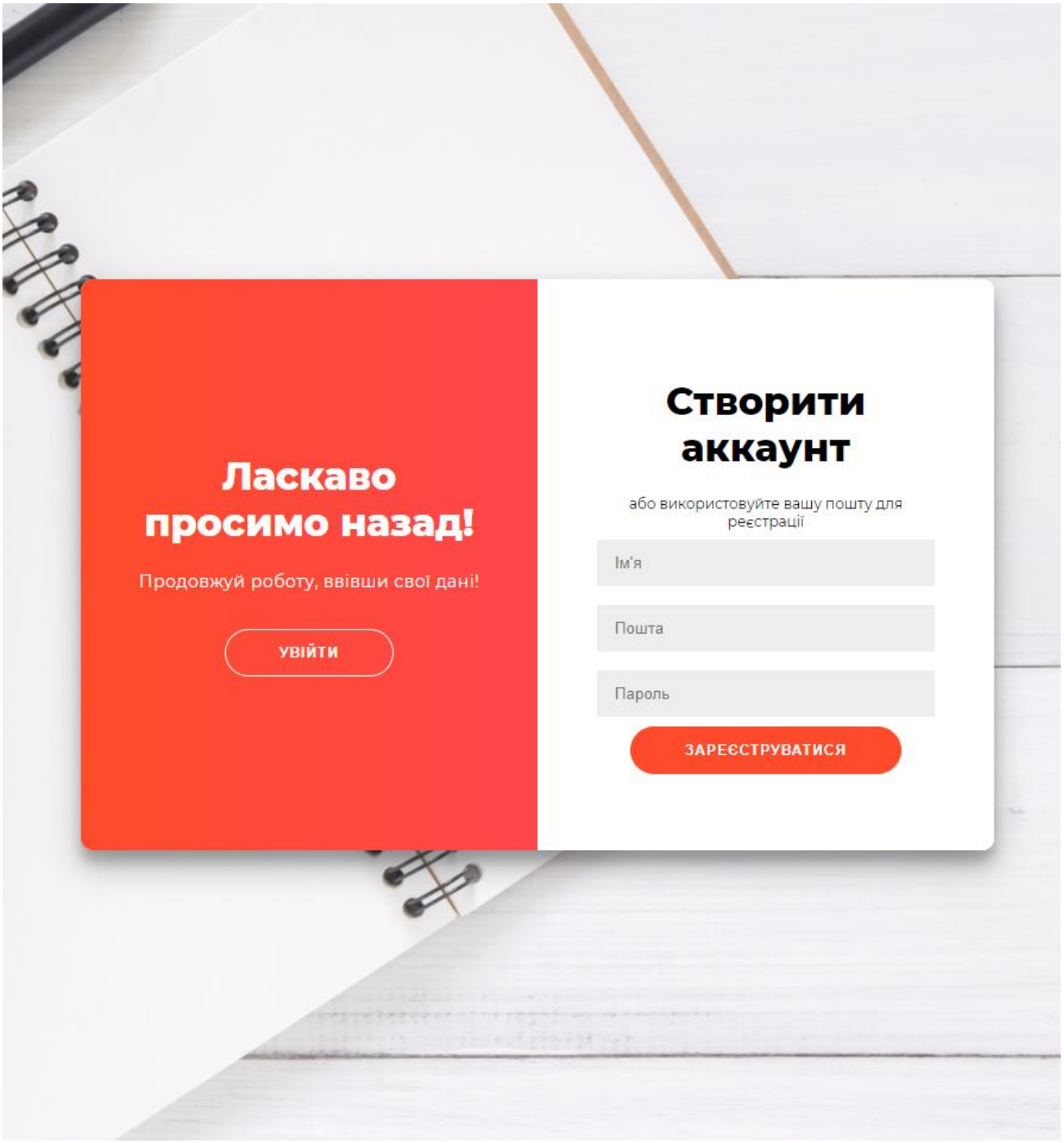

Рис. 2.22. Форма реєстрації

Після авторизації користувач потрапляє на домашню сторінку сайт, де він може бачити публікації інших людей (рис. 2.23), фільтри по категоріям (рис. 2.24.), має можливість перейти до створення публікації, або потрапити до особистого кабінету.

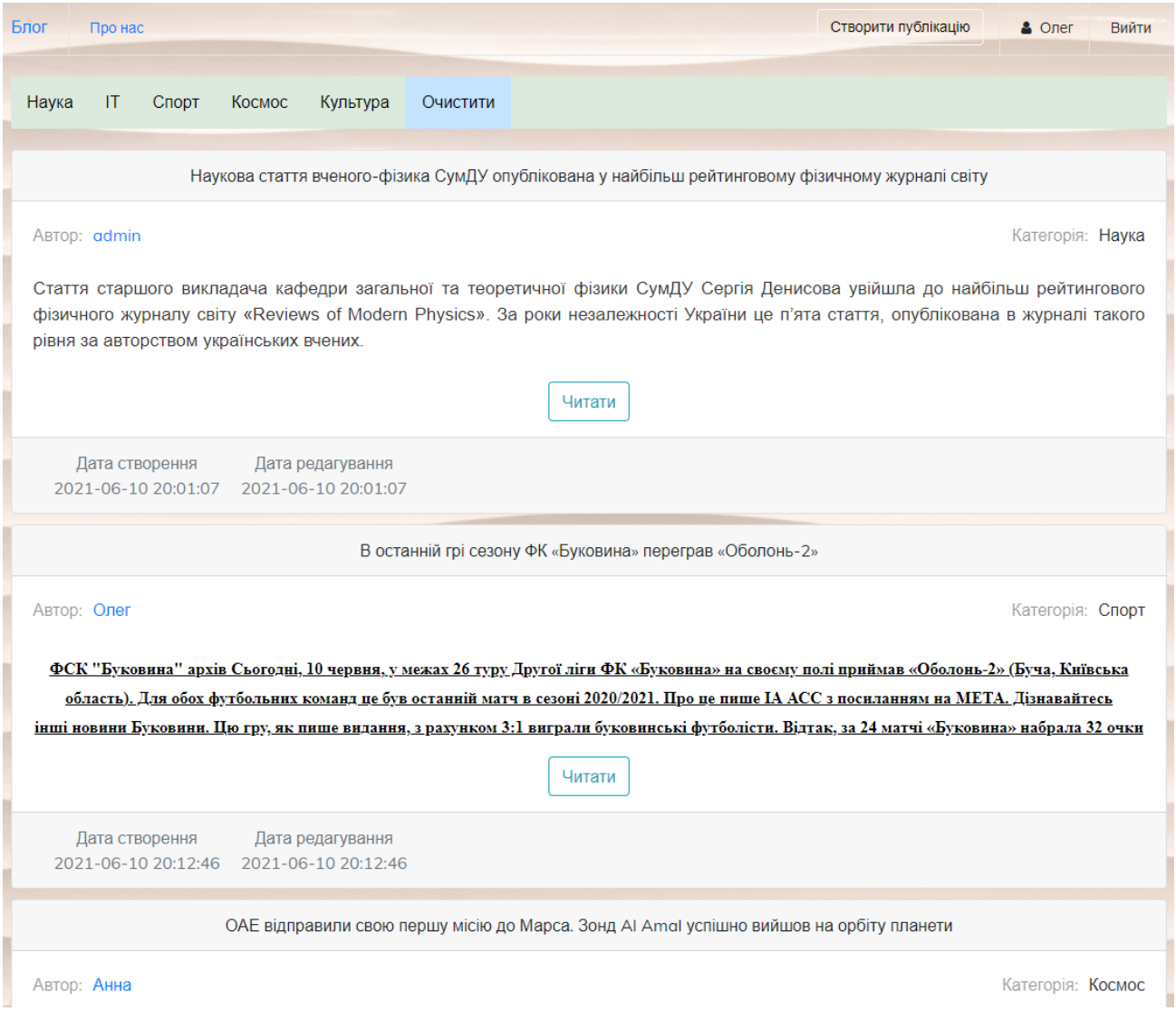

Рис. 2.23. Домашня сторінка сайту

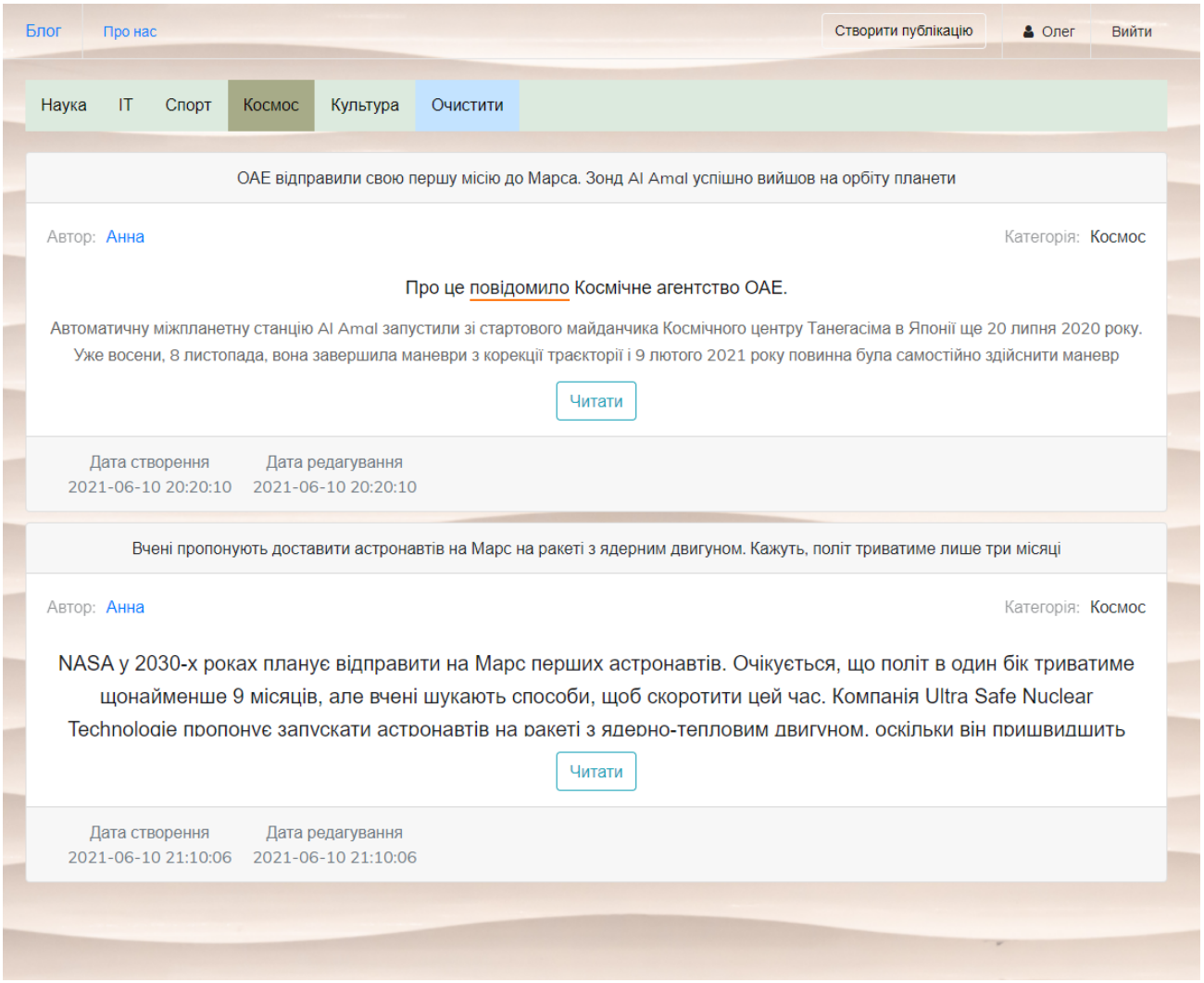

Рис. 2.24. Фільтрація по категорії «Космос»

На рис. 2.25. можна побачити зовнішній вид публікації, коли ми перейшли на неї через кнопку «читати». На цій сторінці користувачі мають змогу повністю продивлятися публікації, поскаржитися адміністрації, у разі порушень будь-яких правил, залишити оцінку (рис. 2.27.) і написати коментар (рис. 2.26.).

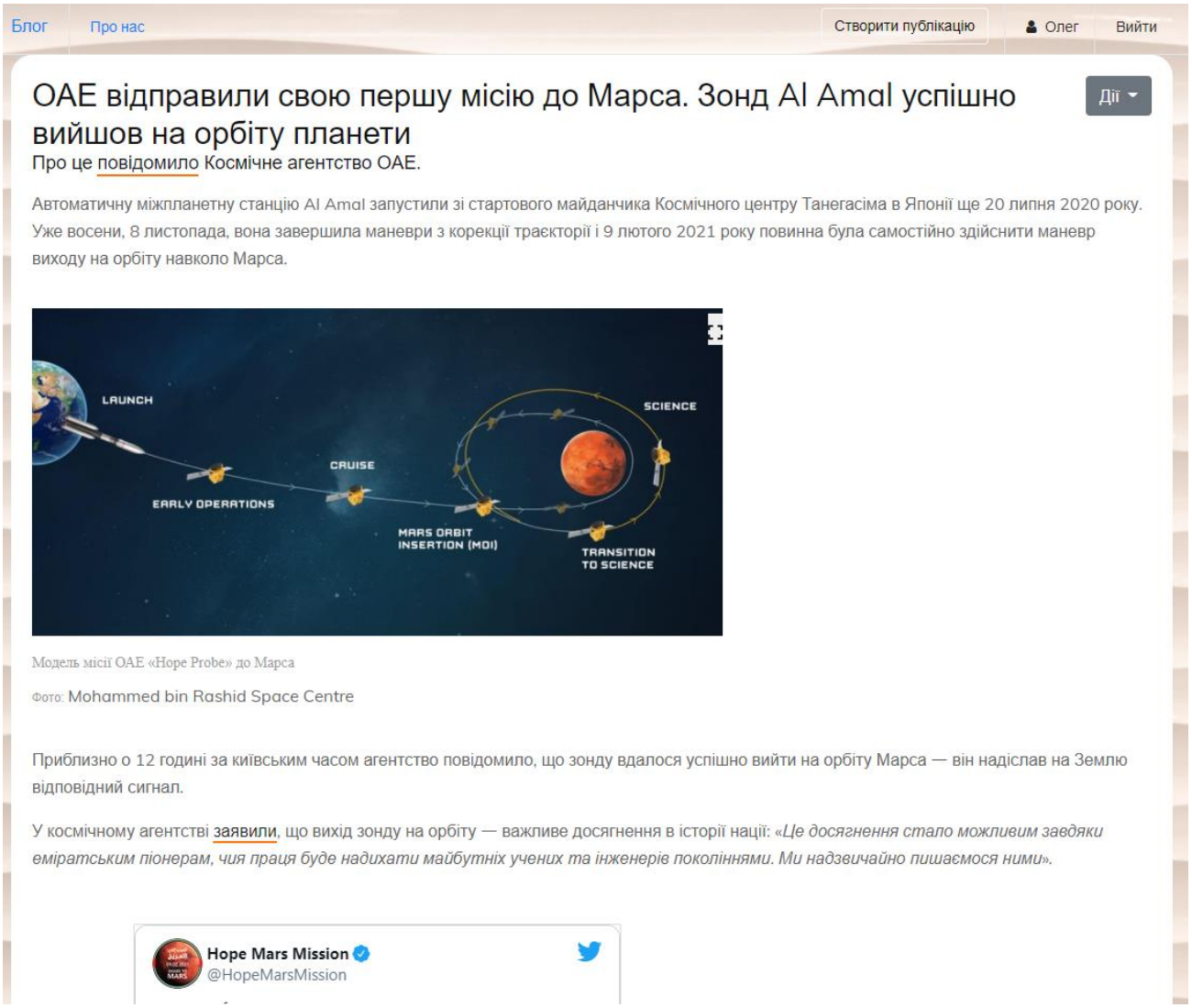

Рис. 2.25. Приклад публікації користувача

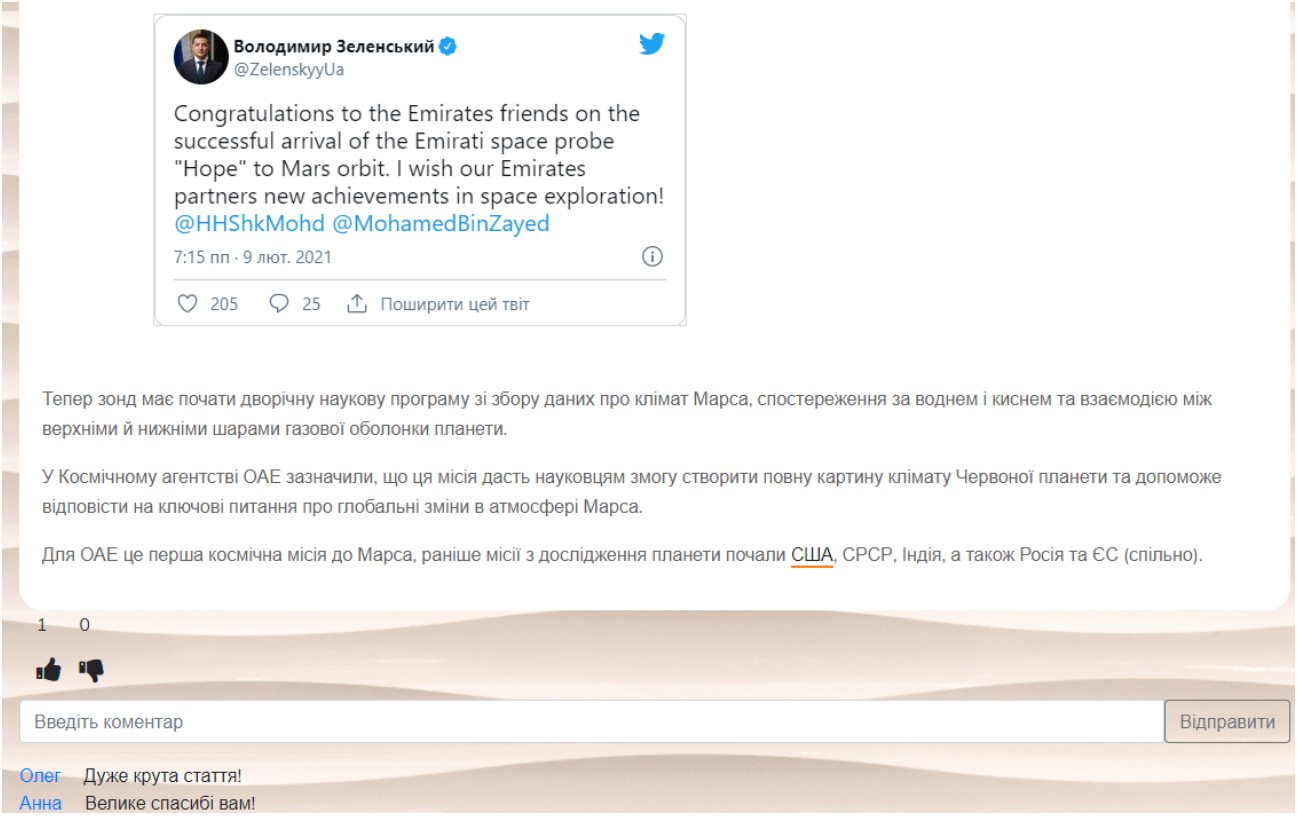

Рис. 2.26. Коментарі під публікацією

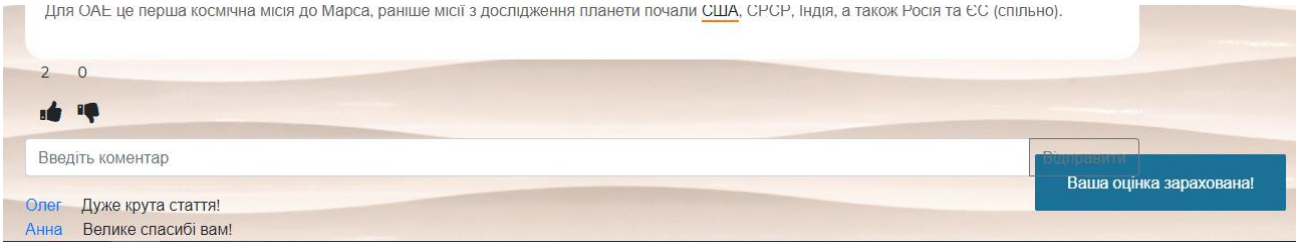

Рис. 2.27. Оцінювання публікації

Для відправки скарги потрібно натиснути на кнопку «Дії» та обрати пункт «Поскаржитися», тоді користувача буде переадресовано на сторінку для оформлення скарги, яка зображена на рис 2.28.

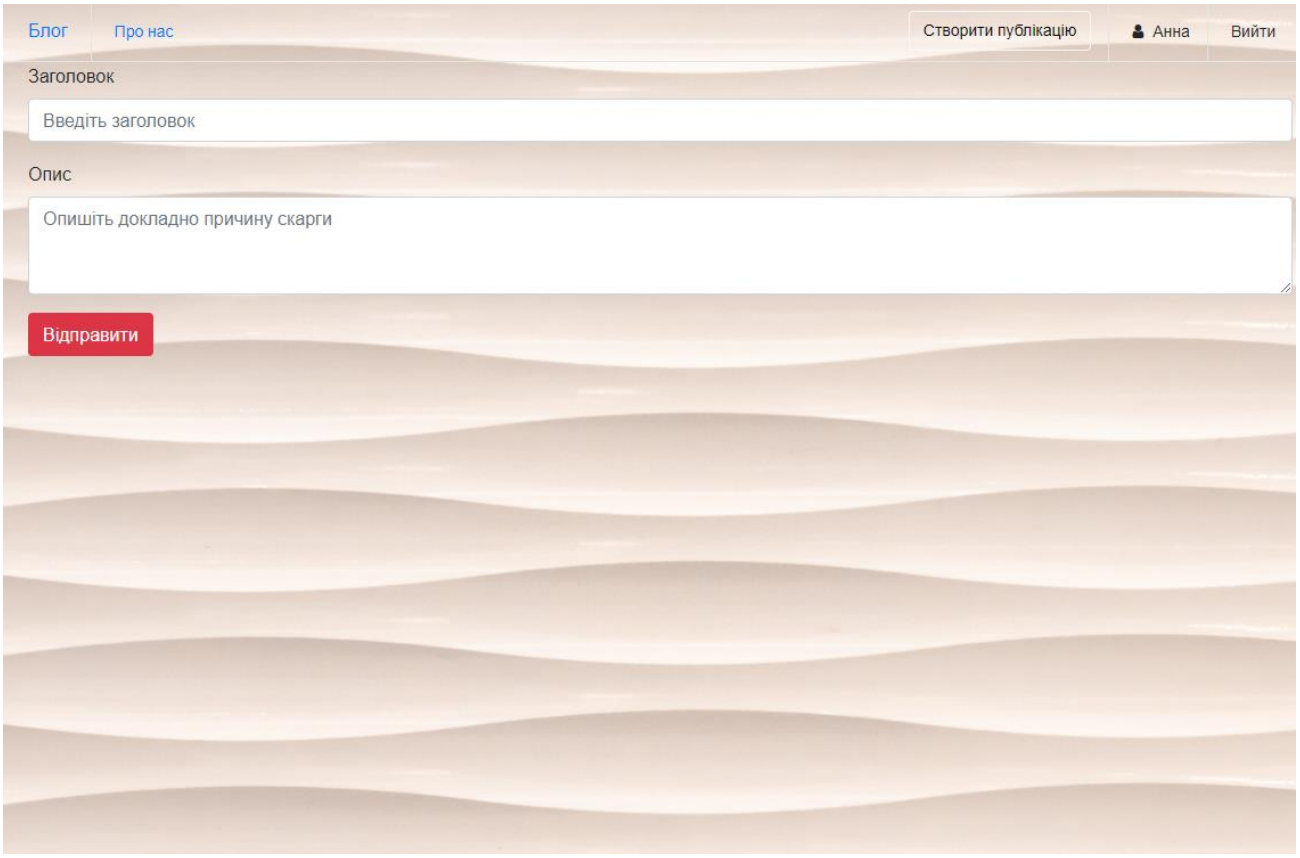

Рис. 2.28. Оформлення скарги на публікацію

Потрапивши на сторінку для створення публікацій перед автором з’являються великі можливості для форматування тексту, які видно на рис. 2.29. Після написання заголовку знаходиться повністю змінюване поле для створення публікації з великою кількістю інструментів. Список інструментів для форматування тексту:

– Стиль. Він задає тип тексту, наприклад можна зробити великий заголовок, або виділити и позначити частину тексту як програмний код.
– Товщина.
– Підкреслення.
– Інструмент для очищення форматування.
– Фарбування тексту та палітра кольорів.
– Два види списків.

– Вирівнювання.
– Таблиці.
– Посилання.
– Додавання картинок через посилання або напряму зі свого пристрою.
– Режим коду, коли видно які теги і стилі застосовуються до тексту.
– Вікно з підказками і гарячими клавішами.

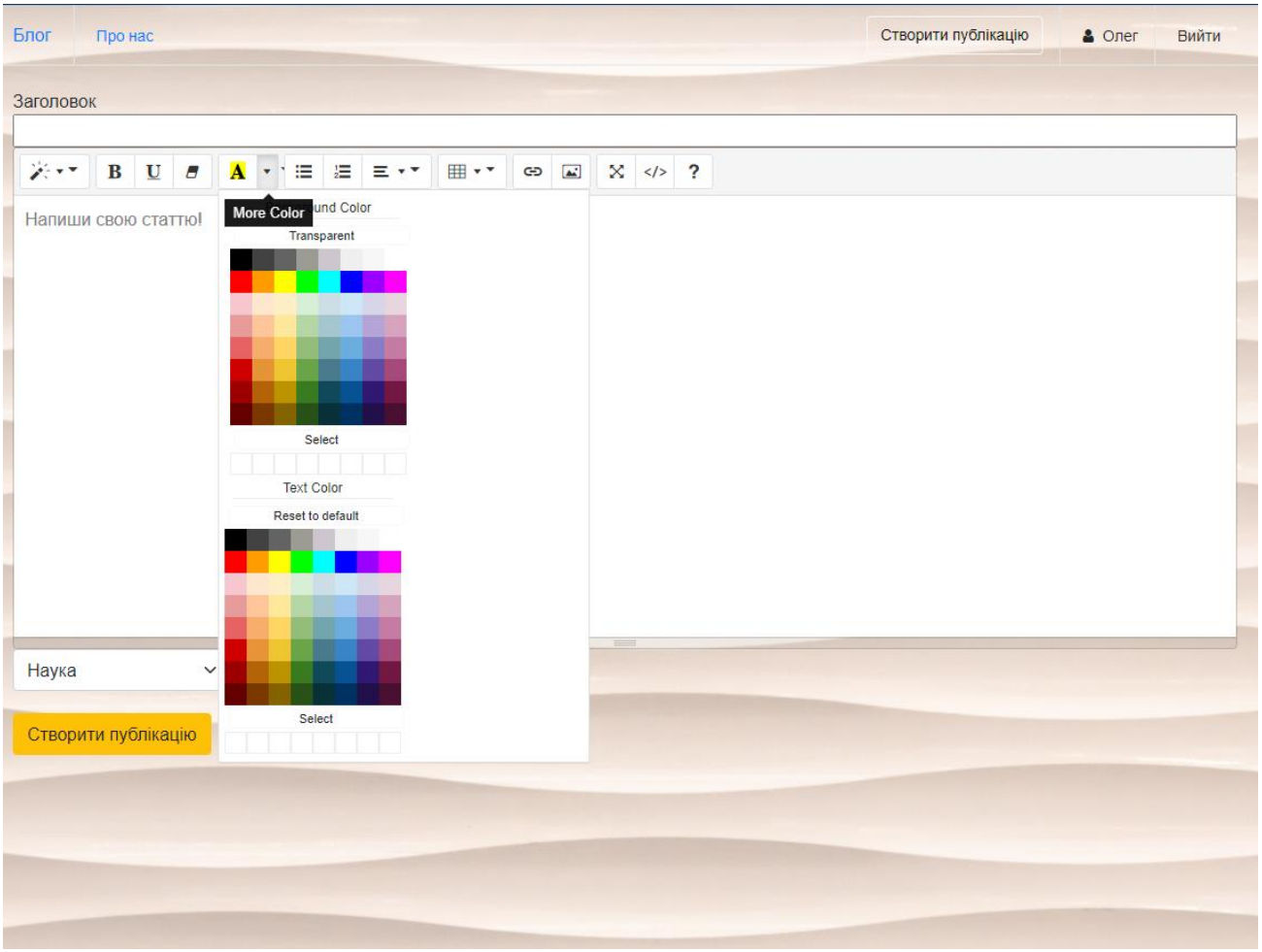

Рис. 2.29. Сторінка створення публікації

Після написання публікації, необхідно обрати категорії, яка розташована під полем для форматування тексту (рис. 2.30).

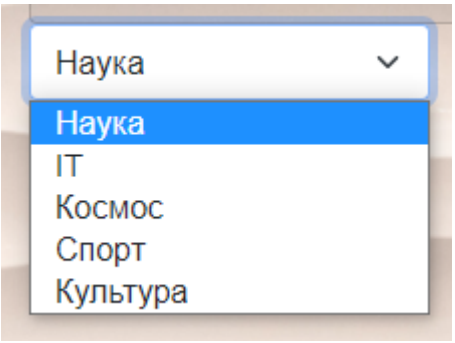

Рис. 2.30. Список категорій

У кожного користувача є особиста сторінка (рис. 2.31), де він може побачити свою інформацію, свої публікації, кількість підписок та підписників і має змогу перейти до редагування (рис. 2.32) особистої інформації.

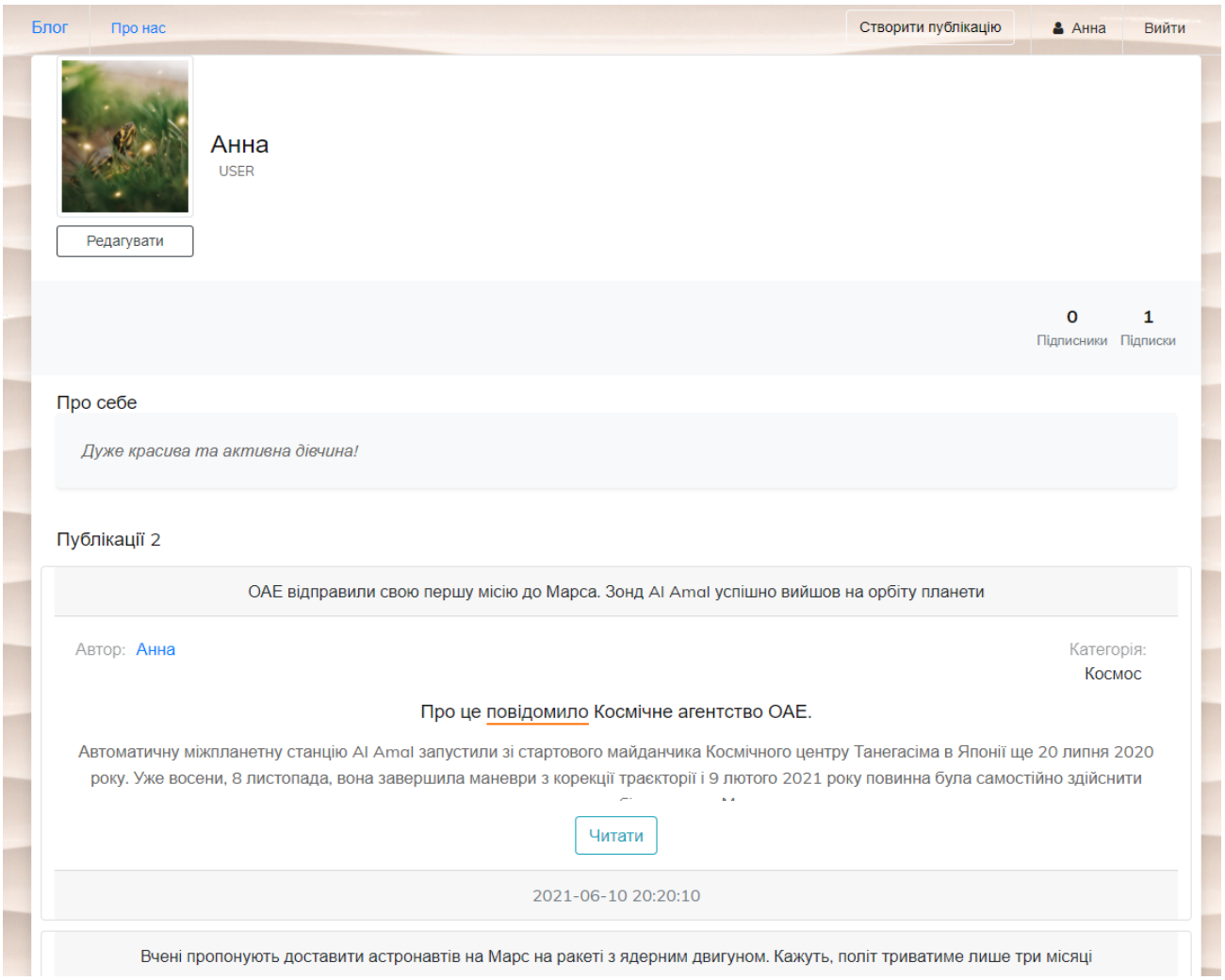

Рис. 2.31. Сторінка користувача

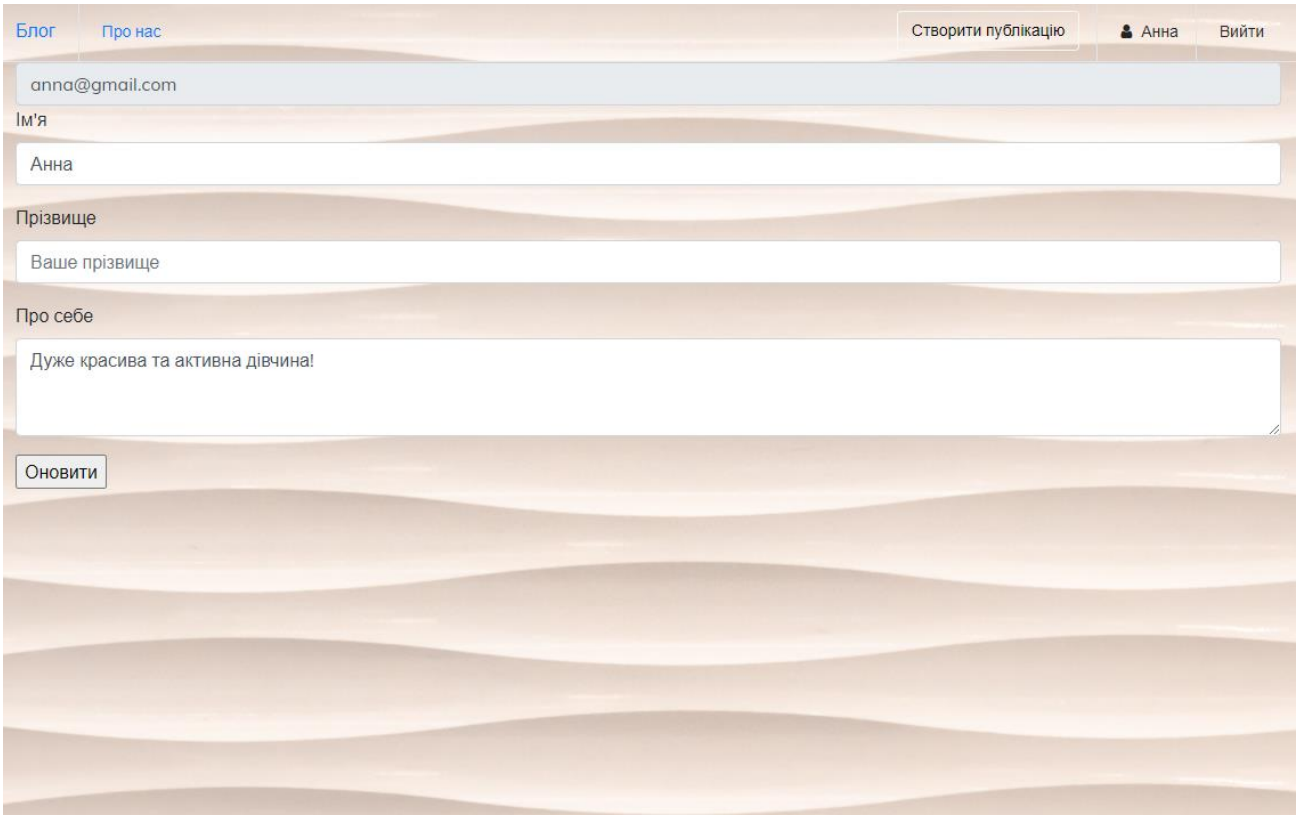

Рис. 2.32. Форма для редагування особистої інформації

Продивляючись публікації, користувач бачить ім’я автора і може перейти на його сторінку, де у нього є можливість підписатися (рис. 2.33) на автора або відписатися.

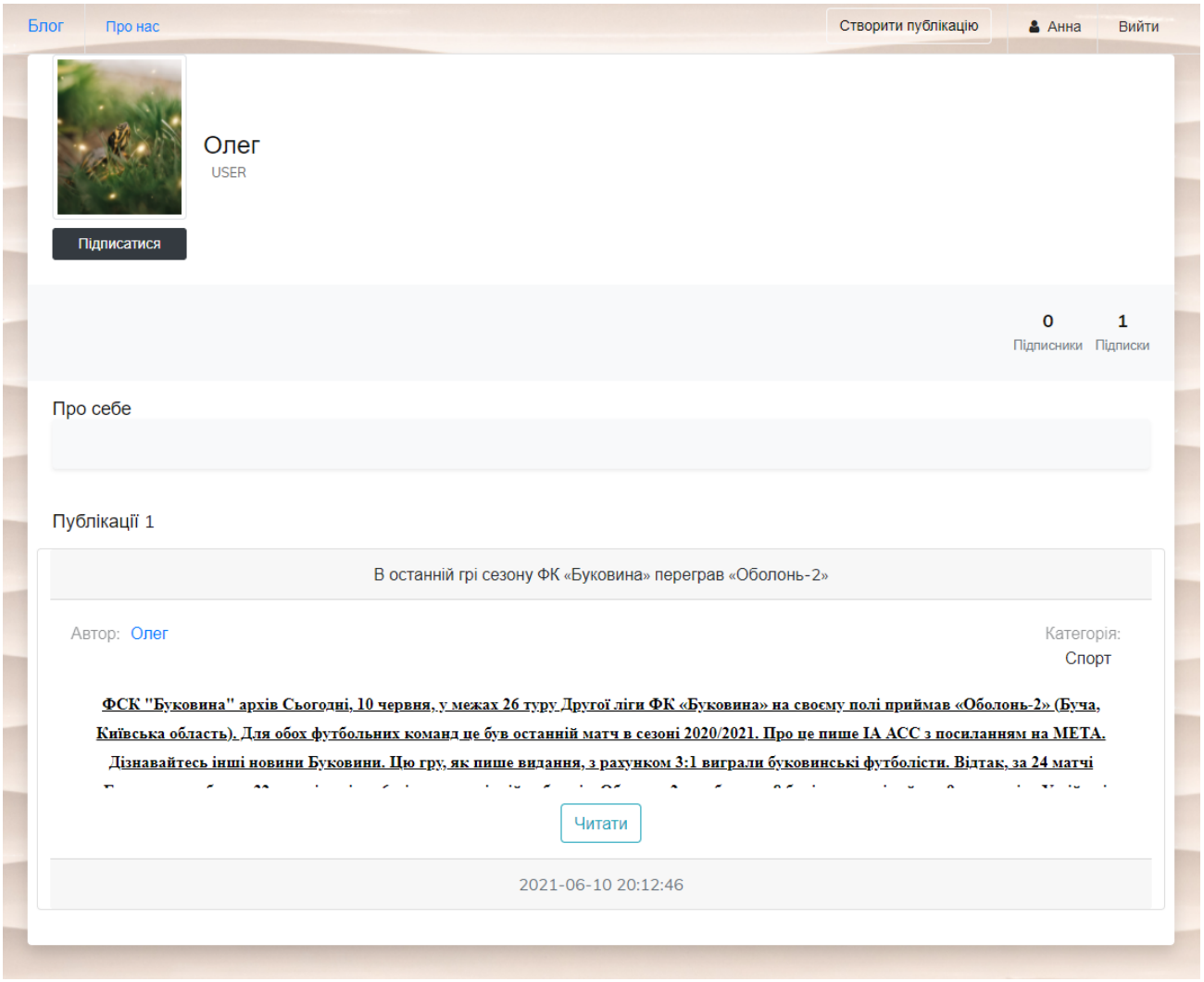

Рис. 2.33. Підписка на іншого користувача

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

Вихідні дані розробки програмного забезпечення:

а) передбачуване число операторів – 1218;

б) коефіцієнт складності програми – 1,41;

в) коефіцієнт кореляції програми в ході її розробки – 0,052;

г) середня годинна заробітна плата програміста, грн/год – 102;

д) коефіцієнт кваліфікації програміста, обумовлений від стажу – 0,8;

е) вартість машино-години ЕОМ, грн/год – 6.

3.1. Визначення трудомісткості розробки програмного забезпечення

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 60 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі,

t_a – витрати праці на розробку блок-схеми алгоритму,

t_n – витрати праці на програмування по готовій блок-схемі,

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ,

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C(1 + p), \text{ людино-годин,} \qquad (3.2)$$

де q – передбачуване число операторів,

C – коефіцієнт складності програми,

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1218 \cdot 1{,}41 \cdot (1 + 0{,}052) = 1806{,}68$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,} \qquad (3.3)$$

де B, яке дорівнює 1,4, – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k, яке дорівнює 0,8, – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності.

$$t_u = \frac{1806{,}68 \cdot 1{,}3}{80 \cdot 0{,}8} = 36{,}69 \text{ , людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25) \cdot k}; \quad (3.4)$$

$$t_a = \frac{1806,68}{23 \cdot 0,8} = 98,18 \text{ , людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25) \cdot k}, \quad (3.5)$$

$$t_n = \frac{1806.68}{22 \cdot 0,8} = 102,65 \text{ , людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k}; \quad (3.6)$$

$$t_{отл} = \frac{1806,68}{4 \cdot 0,8} = 564,58 \text{ , людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}4 \cdot t_{отл}; \tag{3.7}$$

$t_{отл}^{к} = 1{,}4 \cdot 564{,}58 = 790{,}41$, людино-годин.

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, \tag{3.8}$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15..20) \cdot k}; \tag{3.9}$$

$t_{др} = \frac{1806{,}68}{18 \cdot 0{,}8} = 125{,}46$, людино-годин.

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 \cdot t_{д}; \quad (3.10)$$

$t_{до} = 0,75 \cdot 125,46 = 94,1$, людино-годин.

$t_{д} = 125,46 + 94,1 = 219,56$, людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

$t_{д} = 60 + 36,69 + 98,18 + 102,65 + 564,58 + 219,56 = 1081,66$, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1081,66 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного для налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн}, \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин,

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1081{,}66 \cdot 102 = 110329{,}32, \text{ грн.}$$

$З_{МВ}$ – вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{МЧ}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 564{,}5 \cdot 6 = 3387{,}5, \text{ грн,}$$

$$К_{ПО} = 110329{,}3 + 3387{,}5 = 113716{,}8, \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес,} \quad (3.14)$$

де B_k – число виконавців,

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні Fp = 176 годин).

$$T = \frac{1081,66}{1 \cdot 176} \approx 6,14 \text{ міс.}$$

Висновки: час розробки даного програмного забезпечення складає 1081,66 людино-годин. Таким чином, очікувана тривалість розробки складе 6,14 місяця при 40 годинному робочому тижні (місячний фонд робочого часу 176 годин), а витрати на створення програмного забезпечення складатимуть 113716,8 грн.

ВИСНОВКИ

Метою кваліфікаційної роботи була розробка інформаційної системи, яка відкриє можливості для кожного стати відомим та захоплюючим автором, писати статті у різних категоріях і обговорювати цікаві теми з можливістю підписатися на того, хто до вподоби.

В результаті виконання даної кваліфікаційної роботи була розроблена інформаційна система для організації тематичних колективних блогів. Розробка такого веб-додатку дуже корисна і актуальна, особливо в нашій країні, коли поступово уся інформація переходить у інтернет. Використання блогів прискорить отримання та поширення інформації за рахунок зручності та простоти використання.

Дана система використовує язик програмування Java з використанням Spring Boot фреймворку та сучасних технологій і бібліотек.

В порівняні з існуючими подібними блогами створена система позбулась усіх недоліків і перейняла переваги. Це буде сприяти розвитку даної сфери у нашій країні в майбутньому.

Веб-додаток має простий та зрозумілий інтерфейс для користування. Кожен бажаючий може не просто читати записи інших людей, дізнаватися багато нового та цікавого, але і сам стати автором и писати захоплюючі статті та вести дискусію у коментарях.

Для створення цього проекту потрібно було 1801 людино-годин з приблизною тривалістю розробки в 6,14 місяця при 40 годинному робочому тижні, враховуючи, що місячний фонд робочого часу 176 годин, а витрати на створення інформаційної системи складатимуть 113716 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про Хабр [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%B1%D1%80. дата звернення: 11.05.2021.
2. Хабр "Карма" [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/docs/help/karma/. дата звернення: 11.05.2021.
3. Хабр про співтовариство [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/about/ дата звернення: 13.05.2021.
4. Horstmann C. Core Java Volume I – Fundamentals / Cay Horstmann., 2018. – p.1040.
5. Bloch J. Effective Java / Joshua Bloch., 2017. – p.416.
6. Walls C. Spring in Action / Craig Walls., 2018. – p.520.
7. Про Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-security. дата звернення: 22.05.2021.
8. Duckett J. HTML and CSS : Design and Build Websites / Jon Duckett. Robson E. Head First HTML and CSS: A Learner’s Guide to Creating Standards-Based Web Pages / E. Robson, E. Freeman., 2012. – p.768.
9. Database Concepts / D.Kroenke, D. Auer, S. Vandenberg, R. Yoder., 2019. – p.552.
10. Introduction to Java Spring Boot: Learning By Coding / [A. Henley, D. Wolf, A. Ankomah та ін.]., 2019. – p.132.
11. Spring MVC та Dispatcher Servlet [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/336816/. дата звернення: 23.05.2021.

12. Ansari H. Learn Spring Boot: Designed for Java developers to understand and develop production-ready spring applications with minimum configurations. / H. Ansari., 2021. – p.626.

13. JavaScript. Подробное руководство / Дэвид Флэнаган, 2012. – 1080с.

14. Coronel C. Database Systems: Design, Implementation, & Management 13th Edition / Carlos Coronel., 2018. – p.816.

15. CSS in Depth / Keith J. Grant, 2018. – p.434.

16. Fields D. IntelliJ IDEA in Action: Covers IDEA v.5 / D. Fields, S. Saunders, E. Belyaev., 2006. – p.450.

17. PETERSON R. PHP AND MY-SQL A FULL BASICS & ADVANCED / ROBERT PETERSON., 2019. – p.1338.

18. Робота Git [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/541258/. дата звернення: 24.05.2021.

19. А. Хрусталев, А. Кириченко / «HTML5 + CSS3. Основы современного WEB-дизайна», 2018. – 352с.

20. Заробітна плата Java Junior [Електронний ресурс] – Режим доступу до ресурсу: https://jobs.dou.ua/salaries/#period=dec2020&city=Dnipro&title=Junior%20Software%20Engineer&language=Java&spec=&exp1=0&exp2=1. дата звернення: 27.05.2021.

ДОДАТОК А

КОД ПРОГРАМИ

WebSecurityConfig.java

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

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

   @Bean
   @Override
   public AuthenticationManager authenticationManagerBean() throws Exception {
      return super.authenticationManagerBean();
   }

   @Override
   public void configure(HttpSecurity http) throws Exception {
      http
            .authorizeRequests()
            .antMatchers("/image/**", "/css/*", "/fonts/*", "/scripts/*", "/js/*").permitAll()
            .antMatchers("/", "/home", "/registration", "/posts", "/posts/**").permitAll() // all endpoints for
unauthenticated users

            .anyRequest()
            .authenticated()

            .and()
            .formLogin()
            .loginPage("/login")
            .defaultSuccessUrl("/home")
            .permitAll()

            .and()
            .logout()
            .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
            .logoutSuccessUrl("/login")
            .permitAll();
   }
}
```

AdminController,java

```
@Controller
@RequiredArgsConstructor
@RequestMapping("secure")
@Access.Admin
```

```
public class AdminController {

    private final CategoryService categoryService;

    @GetMapping
    public String getAdminPage() {
        return "secure/main";
    }

    @GetMapping("category")
    public String getCategory(Model model) {
        model.addAttribute("categories", categoryService.findAll());
        return "secure/category";
    }

    @PostMapping("category")
    public String createCategory(@ModelAttribute CategoryAddDto categoryAddDto) {
        categoryService.addCategory(categoryAddDto);
        return "redirect:/secure/category";
    }
}
```

ExceptionHandlerController.java

```
@Slf4j
@ControllerAdvice
public class ExceptionHandlerController {

    public static final String STATUS = "status";
    public static final String ERROR_MSG = "errorMsg";
    public static final String ERROR_PAGE_PATH = "/error/errorPage";

    @ResponseStatus(HttpStatus.FORBIDDEN)
    @ExceptionHandler(AccessDeniedException.class)
    public String handleAccessDeniedException(Model model, AccessDeniedException exception) {
        log.warn(exception.getMessage(), exception);

        model.addAttribute(STATUS, "403");
        model.addAttribute(ERROR_MSG, "Access denied");
        return ERROR_PAGE_PATH;
    }

    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ExceptionHandler(EntryNotFoundException.class)
    public String handleEntryNotFoundException(Model model, EntryNotFoundException exception) {
        log.warn(exception.getMessage(), exception);

        model.addAttribute(STATUS, "404");
        model.addAttribute(ERROR_MSG, exception.getMessage());
        return ERROR_PAGE_PATH;
    }
```

```
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler(Exception.class)
    public String handleError(Exception exception, Model model) {
        log.error(exception.getMessage(), exception);

        ResponseStatus annotation = AnnotationUtils.findAnnotation(exception.getClass(), ResponseStatus.class);
        if (annotation != null) {
            model.addAttribute(STATUS, annotation.value().value());
            model.addAttribute(ERROR_MSG, annotation.value().getReasonPhrase());
        } else {
            model.addAttribute(STATUS, "500");
            model.addAttribute(ERROR_MSG, "Oops... something went wrong");
        }

        return ERROR_PAGE_PATH;
    }

}
```

LoginController,java

```
@Controller
public class LoginController {
    @GetMapping("login")
    public String login() {
        return "/security/login";
    }
}
```

MainPageController.java

```
@Controller
@RequiredArgsConstructor
@RequestMapping({"/", "home"})
public class MainPageController {

    private final PostService postService;

    @GetMapping
    public String getMainPage(Model model) {
        model.addAttribute("posts", postService.findAllPosts());
        return "main-page";
    }
}
```

PostController.java

```
@Controller
@RequiredArgsConstructor
@RequestMapping("posts")
public class PostController {

    private final PostService postService;
    private final CategoryService categoryService;
```

```
    @GetMapping("create")
    public String getPost(Model model) {
        model.addAttribute("categories", categoryService.findAll());
        return "creating-post";
    }

    @GetMapping("{postId}")
    public String getPostById(Model model, @PathVariable String postId){
        model.addAttribute("post", postService.findPostById(postId));
        return "post";
    }

    @GetMapping
    public String findAll(Model model,
                          @RequestParam String category){
        model.addAttribute("posts", postService.findAllPosts(category));
        return "main-page";
    }

    @GetMapping("{postId}/report")
    public String reportPost(Model model, @PathVariable String postId){
        model.addAttribute("postId", postId);
        return "report";
    }

    @PostMapping
    public String addPost(@ModelAttribute @Valid PostInfoDto postInfoDto, @AuthenticationPrincipal SecurityUser
user) {
        postService.addPost(postInfoDto, user.getUsername());
        return "redirect:/home";
    }

    @PostMapping("{postId}/comment")
    public String addComment(@PathVariable String postId, @ModelAttribute CommentDto commentDto,
@AuthenticationPrincipal SecurityUser user) {
        postService.addComment(postId, commentDto, user.getUsername());
        return String.format("redirect:/posts/%s", postId);
    }

    @PostMapping("{postId}/like")
    public ResponseEntity<PostLikesDto> addLike(@PathVariable String postId, @AuthenticationPrincipal
SecurityUser user) {
        return new ResponseEntity<>(postService.likePost(postId, user.getUsername()), HttpStatus.OK);
    }

    @PostMapping("{postId}/dislike")
    public ResponseEntity<PostLikesDto> addDislike(@PathVariable String postId, @AuthenticationPrincipal
SecurityUser user) {
        return new ResponseEntity<>(postService.dislikePost(postId, user.getUsername()), HttpStatus.OK);
    }
```

```
}
```

ProfileController.java

```
@Controller
@RequiredArgsConstructor
@RequestMapping("profile")
public class ProfileController {

    private final UserService userService;

    @GetMapping("{userId}")
    public String getUser(Model model, @PathVariable String userId, @AuthenticationPrincipal SecurityUser user) {
        model.addAttribute("userDto", userService.getUserForProfile(userId, user.getUsername()));
        return "profile";
    }

    @GetMapping("{userId}/edit")
    public String editUserPage(Model model, @PathVariable String userId, @AuthenticationPrincipal SecurityUser
user){
        model.addAttribute("userDto", userService.getUserForProfile(userId, user.getUsername()));
        return "edit";
    }

    @PostMapping("{userId}/edit")
    public String editUserProfile(@PathVariable String userId, @ModelAttribute UserEditDto userEditDto){
        userService.editProfile(userId, userEditDto);
        return String.format("redirect:/profile/%s", userId);
    }

    @PostMapping("{userId}/follow")
    public String follow(@PathVariable String userId, @AuthenticationPrincipal SecurityUser user){
        userService.followToUser(userId, user.getUsername());
        return String.format("redirect:/profile/%s", userId);
    }

    @PostMapping("{userId}/unfollow")
    public String unfollow(@PathVariable String userId, @AuthenticationPrincipal SecurityUser user){
        userService.unfollowFromUser(userId, user.getUsername());
        return String.format("redirect:/profile/%s", userId);
    }

}
```

RegistrationController.java

```
@Controller
@RequestMapping("/registration")
@RequiredArgsConstructor
public class RegistrationController {

    private final UserService userService;
```

```
    @PostMapping
    public String register(@ModelAttribute @Valid UserDto userDto) {
        userService.registerUser(userDto);
        return "redirect:/login";
    }

}
```

ReportController.java

```
@Controller
@RequiredArgsConstructor
@RequestMapping("reports")
public class ReportController {

    private final ReportService reportService;

    @PostMapping
    public String createReport(@ModelAttribute ReportDto reportDto, @AuthenticationPrincipal SecurityUser user){
        reportService.createReport(reportDto, user.getUsername());
        return "redirect:/home";
    }
}
```

CommentDtoToCommentMapper.java

```
@Mapper
public interface CommentDtoToCommentMapper {

    Comment commentDtoToComment(CommentDto commentDto);
}
```

PostInfoDtoPostMapper.java

```
@Mapper
public interface PostInfoDtoPostMapper {

    @Mapping(target = "category", ignore = true)
    Post postInfoDtoToPost(PostInfoDto postInfoDto);
}
```

PostToPostCardDtoMapper.java

```
@Mapper
public interface PostToPostCardDtoMapper {

    @Mappings({
            @Mapping(target = "category", source = "post.category.name"),
            @Mapping(target = "createdDate", expression = "java(post.getCreatedDate().toString().replace(\"T\", \"
\").substring(0, 19))"),
            @Mapping(target = "lastModifiedDate", expression = "java(post.getLastModifiedDate().toString().replace(\"T\",
\" \").substring(0, 19))"),
    })
    PostCardDto PostToPostCardDto(Post post);
}
```

PostToPostMainPageDtoMapper.java

```
@Mapper
public interface PostToPostMainPageDtoMapper {

   @Mappings({
         @Mapping(target = "category", expression = "java(post.getCategory().getName())"),
         @Mapping(target = "countLikes", expression = "java(post.getLikes().size())"),
         @Mapping(target = "countDislikes", expression = "java(post.getDislikes().size())"),
         @Mapping(target = "createdDate", expression = "java(post.getCreatedDate().toString().replace(\"T\", \"
\").substring(0, 19))"),
         @Mapping(target = "lastModifiedDate", expression = "java(post.getLastModifiedDate().toString().replace(\"T\",
\" \").substring(0, 19))")}
   )
   PostMainPageDto postToPostMainPageDto(Post post);
}
```

PostToPostPageDtoMapper.java

```
@Mapper
public interface PostToPostPageDtoMapper {

   @Mappings({
         @Mapping(target = "category", expression = "java(post.getCategory().getName())"),
         @Mapping(target = "countLikes", expression = "java(post.getLikes().size())"),
         @Mapping(target = "countDislikes", expression = "java(post.getDislikes().size())")}
   )
   PostPageDto postToPostDto(Post post);
}
```

ReportDtoToReportMapper.java

```
@Mapper
public interface ReportDtoToReportMapper {

   Report reportDtoToReport(ReportDto reportDto);
}
```

UserToSecurityUserConverter.java

```
@Component
public class UserToSecurityUserConverter implements Converter<User, SecurityUser> {
   @Override
   public SecurityUser convert(User user) {
      return SecurityUser.builder()
            .id(user.getId())
            .password(user.getPassword())
            .username(user.getEmail()) // email is username for Spring
            .firstName(user.getFirstName())
            .authorities(List.of(new SimpleGrantedAuthority(user.getUserRole().name())))
            .admin(user.getUserRole() == UserRole.ADMIN)
            .build();
   }
}
```

UserToUserProfileDtoMapper.java

```
@Mapper
public interface UserToUserProfileDtoMapper {

   @Mappings({
         @Mapping(target = "followersCount", ignore = true),
         @Mapping(target = "posts", ignore = true),
   })
   UserProfileDto userToUserProfileDto(User user);

}
```

CategoryAddDto.java

```
@Data
public class CategoryAddDto {

   private String name;
}
```

CommentDto.java

```
@Data
public class CommentDto {

   private String body;
}
```

DislikeDto.java

```
@Data
@AllArgsConstructor
public class DislikeDto {

   private Integer dislikes;
}
```

LikeDto,java

```
@Data
@AllArgsConstructor
public class LikeDto {

   private Integer likes;
}
```

PostCardDto.java

```
@Data
public class PostCardDto {

   private String id;

   private String title;

   private String body;
```

```
    private String category;

    private String createdDate;

    private String lastModifiedDate;
}
```

PostInfoDto.java

```
@Data
public class PostInfoDto {

    private String title;

    private String body;

    private String category;
}
```

PostLikesDto.java

```
@Data
public class PostLikesDto {

    private Integer likes;

    private Integer dislikes;
}
```

PostMainPageDto.java

```
@Data
public class PostMainPageDto {

    private String id;

    private String title;

    private String body;

    private String category;

    private Integer countLikes;

    private Integer countDislikes;

    private List<Comment> comments;

    private String createdDate;

    private String lastModifiedDate;

    private User user;
```

```
}
```

PostPageDto.java

```
@Data
public class PostPageDto {

    private String id;

    private String title;

    private String body;

    private String category;

    private Integer countLikes;

    private Integer countDislikes;

    private List<Comment> comments;

}
```

ReportDto.java

```
@Data
public class ReportDto {

    private String title;

    private String body;

    private String postId;
}
```

UserDto.java

```
@Data
public class UserDto {

    @Size(min = 3, message = "minimum 3 characters")
    private String firstName;

    @UniqueEmail
    @Pattern(
            regexp = "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$",
            message = "invalid email"
    )
    private String email;

    @Pattern(
            regexp = "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9]).{8,}$",
            message = "wrong password"
    )
```

```
    private String password;

}
```

UserEditDto.java

```
@Data
public class UserEditDto {

    private String firstName;

    private String lastName;

    private String about;

}
```

UserProfileDto.java

```
@Data
public class UserProfileDto {

    private String id;

    private String firstName;

    private String email;

    private String lastName;

    private String about;

    private String userRole;

    private boolean isFollowed;

    private int followersCount;

    private int followingsCount;

    private int postsCount;

    private List<PostCardDto> posts;
}
```

CategoryNotFoundException.java

```
public class CategoryNotFoundException extends EntryNotFoundException{

    public CategoryNotFoundException(String message) {
        super(message);
    }
}
```

EntryNotFoundException.java

```
public class EntryNotFoundException extends RuntimeException {
   public EntryNotFoundException(String message) {
      super(message);
   }
}
```

UserNotFoundException.java

```
public class UserNotFoundException extends EntryNotFoundException{
   public UserNotFoundException(String message) {
      super(message);
   }
}
```

BaseEntity.java

```
@Data
@NoArgsConstructor
@SuperBuilder
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
public abstract class BaseEntity {

   @Id
   @GeneratedValue(generator = "UUID")
   @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
   private String id;

   @Column(updatable = false)
   @CreatedDate
   private LocalDateTime createdDate;

   @LastModifiedDate
   private LocalDateTime lastModifiedDate;

   @Override
   public boolean equals(Object o) {
      if (this == o)
         return true;
      if (o == null)
         return false;
      if (getClass() != o.getClass())
         return false;
      BaseEntity other = (BaseEntity) o;
      return id != null && id.equals(other.getId());
   }

   @Override
   public int hashCode() {
      return 42;
   }
}
```

PublicationEntity.java

```
@Data
@SuperBuilder
@NoArgsConstructor
@MappedSuperclass
public abstract class PublicationEntity extends BaseEntity {

   @Type(type = "text")
   private String body;
}
```

TitleEntity.java

```
@Data
@SuperBuilder
@NoArgsConstructor
@MappedSuperclass
public abstract class TitleEntity extends PublicationEntity {

   private String title;
}
```

UserRole.java

```
public enum UserRole {
   USER,
   MODERATOR,
   ADMIN,
}
```

Category.java

```
@Entity
@Table(name = "categories")
@Getter
@Setter
@EqualsAndHashCode(callSuper = false)
public class Category extends BaseEntity {

   private String name;

   private String description;

   @OneToMany(mappedBy = "category")
   private List<Post> posts;
}
```

Comment.java

```
@Entity
@Table(name = "comments")
@Getter
@Setter
@EqualsAndHashCode(callSuper = false)
public class Comment extends PublicationEntity {
```

```
    @ManyToOne
    private User user;

    @ManyToOne
    private Post post;
}
```

Post.java

```
@Entity
@Table(name = "posts")
@Getter
@Setter
@EqualsAndHashCode(callSuper = false)
public class Post extends TitleEntity {

    @ManyToOne
    private User user;

    @ManyToOne
    private Category category;

    @ManyToMany
    @JoinTable(name = "likes",
            joinColumns = {@JoinColumn(name = "post_id")},
            inverseJoinColumns = {@JoinColumn(name = "user_id")})
    private Set<User> likes = new HashSet<>();

    @ManyToMany
    @JoinTable(name = "dislikes",
            joinColumns = {@JoinColumn(name = "post_id")},
            inverseJoinColumns = {@JoinColumn(name = "user_id")})
    private Set<User> dislikes = new HashSet<>();

    @OneToMany(mappedBy = "post")
    private List<Report> reports;

    @OneToMany(mappedBy = "post")
    private List<Comment> comments;

    public void like(User user) {
        dislikes.remove(user);
        if (likes.contains(user)) {
            likes.remove(user);
        } else {
            likes.add(user);
        }
    }

    public void dislike(User user) {
        likes.remove(user);
```

```
        if (dislikes.contains(user)) {
            dislikes.remove(user);
        } else {
            dislikes.add(user);
        }
    }
}
```

Report.java

```
@Entity
@Table(name = "reports")
@Getter
@Setter
@EqualsAndHashCode(callSuper = false)
public class Report extends TitleEntity {

    @ManyToOne
    private User user;

    @ManyToOne
    private Post post;
}
```

User.java

```
@Entity
@Table(name = "users")
@Getter
@Setter
@SuperBuilder
@NoArgsConstructor
public class User extends BaseEntity {

    private String firstName;

    private String lastName;

    private String password;

    private String about;

    @Column(unique = true)
    private String email;

    @Enumerated(EnumType.STRING)
    private UserRole userRole;

    @ManyToMany(mappedBy = "following", cascade = CascadeType.ALL)
    private Set<User> followers = new HashSet<User>();

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "UserFollowing",
```

```
            joinColumns = {@JoinColumn(name = "UserId")},
            inverseJoinColumns = {@JoinColumn(name = "ParentId")})
    private Set<User> following = new HashSet<User>();

    @ManyToMany(mappedBy = "likes")
    private Set<Post> likedPosts;

    @ManyToMany(mappedBy = "dislikes")
    private Set<Post> dislikedPosts;

    @ManyToOne
    private UserStatus userStatus;

    @OneToMany(mappedBy = "user")
    private List<Report> reports;

    @OneToMany(mappedBy = "user")
    private List<Comment> comments;

    @OneToMany(mappedBy = "user")
    private List<Post> posts;

    public void subscribe(User user) {
        this.getFollowing().add(user);
        user.getFollowers().add(this);
    }

    public void unsubscribe(User user) {
        this.getFollowing().remove(user);
        user.getFollowers().remove(this);
    }
}
```

UserStatus.java

```
@Entity
@Table(name = "user_status")
@Getter
@Setter
public class UserStatus {

    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
    private String id;

    private String name;
}

public interface CategoryRepository extends JpaRepository<Category, String> {

    Optional<Category> findByName(String name);
```

```
}

public interface CommentRepository extends JpaRepository<Comment, String> {
}

public interface PostRepository extends JpaRepository<Post, String> {

   @Query("select p from Post p where p.category.name = :category")
   List<Post> findAllWithParams(@Param("category") String category);
}

public interface ReportRepository extends JpaRepository<Report, String> {
}

public interface UserRepository extends JpaRepository<User, String> {

   Optional<User> findByEmail(String email);

   boolean existsByEmail(String email);
}

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Access {

   @Target({ElementType.METHOD, ElementType.TYPE})
   @Retention(RetentionPolicy.RUNTIME)
   @PreAuthorize("hasAuthority('ADMIN')")
   @interface Admin {
   }

   @Target({ElementType.METHOD, ElementType.TYPE})
   @Retention(RetentionPolicy.RUNTIME)
   @PreAuthorize("hasAuthority('MODERATOR')")
   @interface Moderetor {
   }

}
```

SecurityUser.java

```
@Data
@Builder
public class SecurityUser implements UserDetails {

   private final String id;
   private final String firstName;
   private final String username;
   private final String password;
   private final Collection<GrantedAuthority> authorities;
   private boolean admin;
```

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

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

UserDetailsServiceImpl.java

```
@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UserRepository userRepository;
    private final ConversionService conversionService;

    @Override
    public UserDetails loadUserByUsername(String username) {

        User user = userRepository.findByEmail(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found: " + username));

        return conversionService.convert(user, SecurityUser.class);
    }
}
```

CategoryService.java

```
@Component
@RequiredArgsConstructor
public class CategoryService {

    private final CategoryRepository categoryRepository;


    public Category findByName(String name) {
        return categoryRepository.findByName(name).orElseThrow(() -> new CategoryNotFoundException("Category not found"));
    }
```

```
    public List<Category> findAll() {
        return categoryRepository.findAll();
    }

    public void addCategory(CategoryAddDto categoryAddDto) {
        Category category = new Category();
        category.setName(categoryAddDto.getName());
        categoryRepository.save(category);
    }

}
```

PostService.java

```
@Service
@RequiredArgsConstructor
public class PostService {

    private final PostRepository postRepository;
    private final UserService userService;
    private final CommentRepository commentRepository;
    private final CategoryService categoryService;
    private final PostInfoDtoPostMapper postInfoDtoPostMapper;
    private final CommentDtoToCommentMapper commentDtoToCommentMapper;
    private final PostToPostPageDtoMapper postToPostPageDtoMapper;
    private final PostToPostMainPageDtoMapper postToPostMainPageDtoMapper;

    public List<Post> findAll() {
        return postRepository.findAll();
    }

    public List<Post> findAll(String category) {
        return postRepository.findAllWithParams(category);
    }

    public List<PostMainPageDto> findAllPosts() {
        return
findAll().stream().map(postToPostMainPageDtoMapper::postToPostMainPageDto).collect(Collectors.toList());
    }

    public List<PostMainPageDto> findAllPosts(String category) {
        return
findAll(category).stream().map(postToPostMainPageDtoMapper::postToPostMainPageDto).collect(Collectors.toList());
    }

    public Post findById(String id) {
        return postRepository.findById(id).orElseThrow(() -> new EntryNotFoundException("Post not found"));
    }

    public PostPageDto findPostById(String id) {
        Post byId = findById(id);
```

```
        return postToPostPageDtoMapper.postToPostDto(byId);
    }

    public void addPost(PostInfoDto postInfoDto, String email) {
        Post post = postInfoDtoPostMapper.postInfoDtoToPost(postInfoDto);
        post.setCategory(categoryService.findByName(postInfoDto.getCategory()));
        post.setUser(userService.findByEmail(email));
        postRepository.save(post);
    }

    @Transactional
    public void addComment(String postId, CommentDto commentDto, String userEmail) {
        Post post = findById(postId);
        User user = userService.findByEmail(userEmail);
        Comment comment = commentDtoToCommentMapper.commentDtoToComment(commentDto);
        comment.setUser(user);
        comment.setPost(post);
        post.getComments().add(comment);
        postRepository.save(post);
        commentRepository.save(comment);
    }

    @Transactional
    public PostLikesDto likePost(String postId, String userEmail) {
        Post post = findById(postId);
        post.like(userService.findByEmail(userEmail));
        PostLikesDto postLikesDto = new PostLikesDto();
        postLikesDto.setLikes(post.getLikes().size());
        postLikesDto.setDislikes(post.getDislikes().size());
        return postLikesDto;
    }

    @Transactional
    public PostLikesDto dislikePost(String postId, String userEmail) {
        Post post = findById(postId);
        post.dislike(userService.findByEmail(userEmail));
        PostLikesDto postLikesDto = new PostLikesDto();
        postLikesDto.setLikes(post.getLikes().size());
        postLikesDto.setDislikes(post.getDislikes().size());
        return postLikesDto;
    }

}
```

ReportService.java

```
@Service
@RequiredArgsConstructor
public class ReportService {

    private final UserService userService;
    private final ReportDtoToReportMapper reportDtoToReportMapper;
```

```
    private final ReportRepository reportRepository;
    private final PostService postService;

    public void createReport(ReportDto reportDto, String email){
        Report report = reportDtoToReportMapper.reportDtoToReport(reportDto);
        report.setUser(userService.findByEmail(email));
        report.setPost(postService.findById(reportDto.getPostId()));
        reportRepository.save(report);
    }
}
```

UserService.java

```
@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final UserToUserProfileDtoMapper userToUserProfileDtoMapper;
    private final PostToPostCardDtoMapper postToPostCardDtoMapper;

    @Transactional
    public void registerUser(UserDto userDto) {
        userRepository.save(User.builder()
                .email(userDto.getEmail())
                .firstName(userDto.getFirstName())
                .password(passwordEncoder.encode(userDto.getPassword()))
                .userRole(UserRole.USER)
                .build());
    }

    public User findByEmail(String email) {
        return userRepository.findByEmail(email).orElseThrow(() -> new UserNotFoundException("User not found"));
    }

    public User findById(String id) {
        return userRepository.findById(id).orElseThrow(() -> new UserNotFoundException("User not found"));
    }

    public UserProfileDto getUserForProfile(String id, String userEmail){
        User byId = findById(id);
        UserProfileDto userProfileDto = userToUserProfileDtoMapper.userToUserProfileDto(byId);
        userProfileDto.setFollowersCount(byId.getFollowers().size());
        userProfileDto.setFollowingsCount(byId.getFollowing().size());
        userProfileDto.setPostsCount(byId.getPosts().size());
        userProfileDto.setFollowed(byId.getFollowers().contains(findByEmail(userEmail)));
        userProfileDto.setPosts(
                byId.getPosts()
                        .stream()
                        .map(postToPostCardDtoMapper::PostToPostCardDto)
                        .collect(Collectors.toList()));
```

```
        return userProfileDto;
    }

    @Transactional
    public void followToUser(String userId, String email){
        User byEmail = findByEmail(email);
        byEmail.subscribe(findById(userId));
    }

    @Transactional
    public void unfollowFromUser(String userId, String email){
        User byEmail = findByEmail(email);
        byEmail.unsubscribe(findById(userId));
    }

    @Transactional
    public void editProfile(String userId, UserEditDto userEditDto){
        User byId = findById(userId);
        byId.setFirstName(userEditDto.getFirstName());
        byId.setLastName(userEditDto.getLastName());
        byId.setAbout(userEditDto.getAbout());
    }
}
```

UniqueEmail.java

```
@Documented
@Constraint(validatedBy = UniqueEmailValidator.class)
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface UniqueEmail {

    String message() default "email already taken, try another";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

}
```

UniqueEmailValidator.java

```
@Component
@RequiredArgsConstructor
public class UniqueEmailValidator implements ConstraintValidator<UniqueEmail, String> {

    private final UserRepository userRepository;

    @Override
    public boolean isValid(String email, ConstraintValidatorContext context) {
        return !userRepository.existsByEmail(email);
    }
}
```

BlogApplication.java

```
@EnableJpaAuditing
@SpringBootApplication
@RequiredArgsConstructor
public class BlogApplication {

	public static void main(String[] args) {
		SpringApplication.run(BlogApplication.class, args);
	}

}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФПЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Появнювальні документи	
Диплом_Вороненко.docx	Пояснювальна записка кваліфікаційної роботи. Документ Word.
Диплом_Вороненко.pdf	Пояснювальна записка кваліфікаційної роботи в форматі PDF.
Програма	
Blog.zip	Архів. Містить коди програми.
Презентація	
Презентація_Вороненко.pptx	Презентація кваліфікаційної роботи.