Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: _Розробка онлайн платформи для комунікації програмістів___

Виконав: студент__4___ курсу
групи __5ПІ-21б_____
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Свіца О. С.
(прізвище та ініціали)
Керівник:к.т.н., доц. каф. ПЗ Бабюк Н. П.
(прізвище та ініціали)
Рецензент:.к.т.н., доц. каф ОТ Колесник І.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпеченя»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Свіці Олександру Сергійовичу

1. Тема роботи – розробка онлайн платформи для комунікації програмістів.
Керівник роботи: Бабюк Наталя Петрівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.
2. Строк подання студентом роботи 2 червня 2025 р.
3. Вихідні дані до роботи: засоби створення та обробки запитів, методи реалізації програмних продуктів, інструменти створення онлайн платформ, мова програмування серверної частини – Golang, фреймворк для реалізації фронтенду – Angular.
4. Зміст текстової частини: аналіз актуальності платформ для комунікації програмістів, порівняльний аналіз аналогів, аналіз методів розв'язання поставлених завдань, аналіз функціональних вимог до застосунку, розробка візуальної моделі системи для комунікації програмістів, вибір архітектурного рішення для розробки платформи, розробка алгоритмів роботи системи, варіантний аналіз та обґрунтування вибору засобів реалізації застосунку, розробка модуля «Користувачі», розробка модуля «Чати», розробка модуля «Пости», тестування застосунку, розробка інструкції користувача, висновки.
5. Перелік ілюстративного матеріалу: титульний слайд; мета, об'єкт предмет дослідження, новизна і практичне значення роботи, актуальність, порівняльний аналіз аналогів, вибір мови програмування, модель діяльності системи, алгоритм створення нового користувача, алгоритм визначення доступних та видалених ресурсів, методи та засоби реалізації, демонстрація реєстрації в застосунку, демонстрація результату додавання поста, демонстрація

створення чату, демонстрація повідомлення в чаті, демонстрація авториза системі, демонстрація списку проєктів, результати роботи, апробаці публікація матеріалів бакалаврської кваліфікаційної роботи, кінцевий слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завданн прийняв
1-4	Бабюк Н. П., к.т.н., доц.. кафедри ПЗ	24.03.25 р.	02.06.25 р.

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Сформувати перелік функціональних вимог до системи.	25.03.25 – 31.03.25	Вик
2	Розробити модель системи та алгоритми роботи системи.	01.04.25 – 15.04.25	Вик
3	Реалізувати програмне забезпечення системи.	15.04.25 – 20.05.25	Вик
4	Протестувати розроблений продукт на відповідність вимогам та відсутність помилок.	21.05.25 – 02.06.25	Вик.

Студент О. Свіц **Свіца О. С.**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи **Бабюк Н. П.**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.04

Свіца О. С. Розробка онлайн платформи для комунікації програмістів: бакалаврська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 70 с.

На укр. мові. Бібліогр. : 25 назв.; рис. : 40; табл. : 2.

Під час виконання бакалаврської кваліфікаційної роботи розроблено онлайн платформу для комунікації програмістів. Система дозволяє покращити процес комунікації між програмістами. У ході роботи сформовано перелік функціональних вимог до системи, розроблено модель системи та алгоритми роботи системи, реалізовано програмне забезпечення системи. Протестовано розроблений продукт на відповідність вимогам та відсутність помилок.

Для розробки використано мову програмування Golang для серверної частини та Angular для фронтенду.

ABSTRACT

UDC 004.04

Svitsa O. S. Development of an Online Platform for Programmer Communication: bachelor's thesis in the specialty 121 Software Engineering, educational program – software engineering. Vinnytsia: VNTU, 2025. 70 p.

In Ukrainian. Language. Refs. : 25 titles; rice. : 40; Table. : 2.

During the completion of the bachelor's qualification work, an online platform for communication among programmers was developed. The system aims to improve the communication process between programmers. As part of the work, a list of functional requirements for the system was defined, a system model and operating algorithms were developed, and the system software was implemented. The developed product was tested for compliance with the requirements and for the absence of errors.

The development used the Go programming language for the backend and Angular for the frontend.

ЗМІСТ

ВСТУП

Розвиток сучасних технологій дозволяє покращити процес комунікації між користувачами. Завдяки таким інструментам як інтернет та мобільний зв'язок люди можуть обмінюватися інформацією з будь-якої точки світу. Постійне покращення засобів зв'язку та нові винаходи, наприклад оптоволокно, надають можливість швидко передавати дані на дуже великі відстані [1]. Це означає, що обмін такими об'ємними файлами як картинки чи відео скорочується.

Наявний прогрес у засобах створення програмного забезпечення. Високорівневі мови програмування, такі як Java, C# чи Go дозволяють реалізовувати застосунки абстрагуючись від технічних деталей [2]. Створення інтегрованого середовища розробки, яке включає редактор коду, дебагер, компілятор та інструменти тестування, підвищило продуктивність розробників. Винайдення системи управління версіями дозволило зберігати всі зміни коду та за необхідності повертатися до попередніх версій [3]. Наявність сучасних вебфреймовркв забезпечує швидке створення графічних інтерфейсів, які є зрозумілими для користувача.

Таким чином, сучасний стан технологій дозволяє розробникам задовольняти потреби користувачів у комунікації шляхом створенням різноманітних програмних продуктів, які дозволяють публікувати пости, писати коментарі, підписуватися на інших користувачів, обмінюватися повідомленнями.

Платформи, які існують на ринку та реалізують такий функціонал поділяються на дві категорії. Перша орієнтується на потреби дуже широкого кола користувачів. До цієї категорії відносяться такі платформи як X та Facebook. Вони забезпечують функціонал для комунікації, але не мають додаткових функцій необхідних для такої вузької категорії користувачів як програмісти. Друга категорія орієнтується на програмістів, але має суттєві недоліки в реалізації базових механізмів. До неї можна віднести StackOverflow.

Тому актуальною є розробка платформи для комунікації програмістів.

Метою роботи є покращення процесу комунікації між програмістами шляхом розробки онлайн платформи для комунікації програмістів, що міститиме функціонал, відсутній на інших платформах, але необхідний для зручної взаємодії користувачів. Розробка платформи включатиме застосування сучасних технологій та методів і засобів розробки програмного забезпечення.

Для розробки даної онлайн платформи необхідно виконати наступні завдання:

1. Сформувати перелік функціональних вимог до системи.
2. Розробити модель системи та алгоритми роботи системи.
3. Реалізувати програмне забезпечення системи.
4. Протестувати розроблений продукт на відповідність вимогам та відсутність помилок.

Об’єктом дослідження є методи та засоби розробки онлайн платформи для комунікації програмістів.

Предметом дослідження є онлайн платформа для комунікації, яка дозволяє публікувати пости, писати коментарі, обмінюватися повідомленнями в чатах.

Новизна роботи:

Вдосконалено метод комунікації між програмістами, які поєднують у собі сучасні технології для реалізації функціональних можливостей, як соціальних мереж, так і спеціалізованих каналів спілкування, які спрощують процес комунікації між програмістами.

Практичне значення полягає у використанні онлайн платформи реальними програмістами для знаходження інших користувачів та забезпечення швидкого обміну інформацією.

Апробація та публікація результатів роботи. Результати доповідалися на конференціях «Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)» та «Молодь в

науці: дослідження, проблеми, перспективи (МН-2025)» 2025 року та опубліковані в матеріалах конференцій [4, 5].

Під час виконання бакалаврської кваліфікаційної роботи використано 25 літературних джерела.

1 АНАЛІЗ АКТУАЛЬНОСТІ ПЛАТФОРМ ДЛЯ КОМУНІКАЦІЇ ПРОГРАМІСТІВ, ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ ТА ВИБІР МЕТОДІВ РОЗРОБКИ

1.1 Аналіз актуальності платформ для комунікації програмістів

Розвиток технологій безперервно змінює способи комунікації та кооперації між людьми. На даному етапі велику популярність набули платформи, що надають можливість користувачам спілкуватися онлайн за допомогою постів, коментарів та повідомлень. Важливу роль вони відіграють для користувачів, які професійно займаються проєктуванням та розробкою програмного забезпечення. Така популярність зумовлена багатьма чинниками:

1. Онлайн платформи забезпечують можливість спілкуватися з користувачами, що знаходяться в різних куточках світу, єдина вимога – наявність інтернет підключення. Така доступність дозволяє створювати команди з висококваліфікованих спеціалістів, які проживають в різних країнах, але можуть працювати над одним проєктом.

2. Сучасні технології дозволяють зробити процес доставки повідомлення надзвичайно швидким. Це дозволяє пришвидшити роботу над проєктом, що особливо важливо у ситуаціях, коли необхідно виправити помилки у вже працюючому коді.

3. Використання повідомлень різних форматів. До них можуть входити як текстові повідомлення так і файли, відео або картинки. Це дозволяє репрезентувати інформацію у максимально зручному та зрозумілому форматі. Наприклад, цей функціонал може бути використаний для публікацій блок-схем або UML діаграм.

4. Наявність чатів дозволяє взаємодіяти з користувачами декількох груп одночасно. Це є великою перевагою, коли виникає необхідність координації декількох команд для роботи над спільними задачами.

5. Онлайн платформи можуть інтегруватися з іншими інструментами, які програмісти використовують для своєї роботи.

Актуальність онлайн платформ для комунікації невпинно зростає [6]. Це є наслідком цілої низки факторів, що пов'язані з суттєвими змінами в технологіях, бізнес-процесах та в суспільстві в цілому. Основна причина – зміна робочих процесів в багатьох провідних компаніях світу. Ріст популярності віддаленого та гібридного формату роботи унеможливив організацію діяльності працівників без застосування відповідних інструментів. Важливу роль відіграє також глобалізація та економічна інтеграція. Внаслідок цих процесів компанії виходять на ринки різних країн, що ускладнює організацію діяльності їх працівників.

Зростання популярності платформ для комунікації також зумовлено безперервним розвитком сучасних технологій. Можна виокремити наступні основні етапи, на цьому шляху:

1. Виникнення інтернету. Важко переоцінити важливість цієї технології. З її допомогою користувачі можуть обмінюватися повідомленнями з мінімальними затримками у часі.

2. Створення персонального комп’ютера зробило інструменти для роботи з даними та їх обміном доступнішими для користувачів.

3. Розвиток смартфонів та планшетів ще більше спростив доступ до даних технологій. Написання повідомлень, здійснення дзвінків та публікація постів з телефону стало звичайною справою у сучасному світі.

4. Хмарні технології надають можливість зберігати та обробляти величезну кількість даних, що є надзвичайно важливим для роботи онлайн платформ. Крім того, хмарні провайдери дозволяють масштабувати програми, що забезпечує стабільність їх роботи навіть під час великих навантажень.

5. Розвиток штучного інтелекту лише посилює позиції платформ для комунікації. Він дозволяє створювати функціонал, реалізовувати який звичайними алгоритмами неможливо або недоцільно. Прикладами можуть слугувати: створення автоматичних відповідей, переклад тексту, фільтрація шкідливого контенту.

В подальшому функціонал даного застосунку може бути розширений для задоволення потреб користувачів інших професій.

1.2 Порівняльний аналіз аналогів

У даному розділі наведено порівняння платформи для комунікації з існуючими аналогами. Цей етап є невід'ємним у процесі створення програмного забезпечення. Він дозволить проаналізувати наш продукт, виявити його переваги та недоліки та зробити висновок чи дійсно він буде затребуваним на ринку.

Список платформ, що підлягають аналізу, включає Facebook, X, StackOverflow. Кожна з них має свої унікальні характеристики, що зробили її особливим продуктом, який користується попитом користувачів усього світу.

X [7] – соціальна мережа, яка дозволяє користувачам миттєво обмінюватися новинами шляхом надсилання повідомлень довжиною до 280 символів. Інтерфейс X наведено на рисунку 1.1.

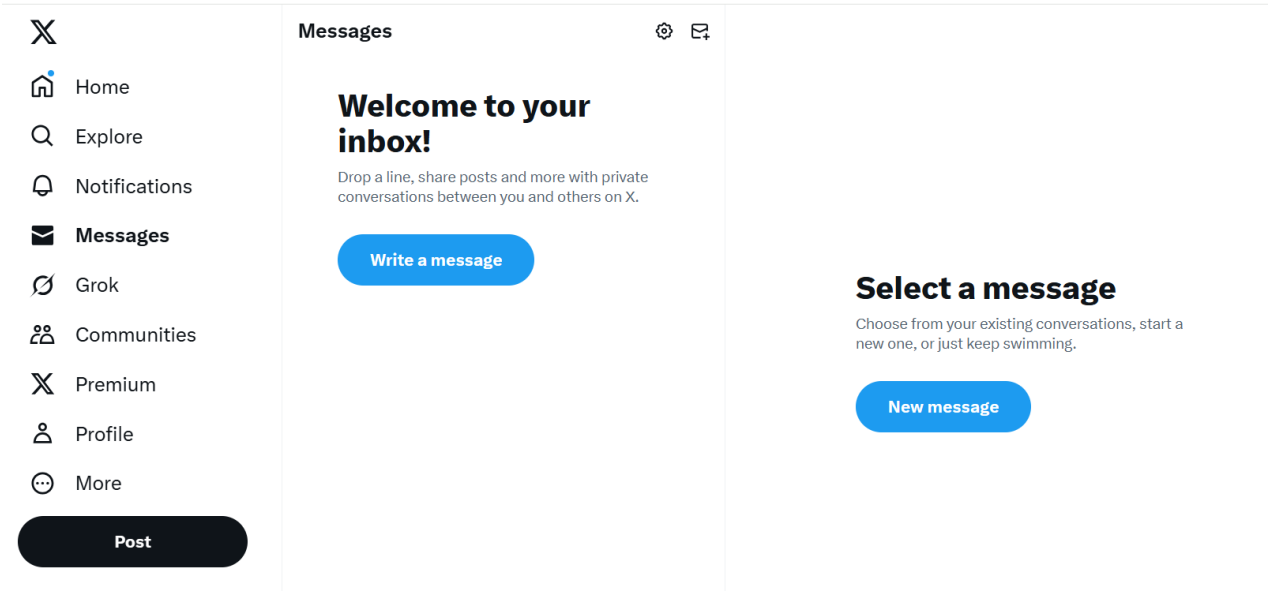

Рисунок 1.1 – Інтерфейс платформи X

Однією з основних функцій X є можливість публікувати пости. Цей функціонал надає можливість користувачу донести свою думку до широкої

аудиторії не обмежуючись конкретним географічним регіоном. Зазвичай він використовується для того, щоб поділитися досягненнями, подіями у житті або іншими новинами з усіма зацікавленими користувачами. Пости X також стають інструментом просування особистого бренду або бренду компанії. Ця функція особливо важлива для підприємств, що бажають підтримувати постійний зв’язок із своїми клієнтами та дуже швидко реагувати на будь-які зміни. Недоліком є обмеження на кількість символів у одному пості. Станом на березень 2025 року довжина повідомлення не має перевищувати 280 символів.

Наявність коментарів забезпечує комунікацію між автором поста та його читачами. Вони дозволяють висловити свою думку щодо прочитаного, що дозволяє автору зрозуміти як інші користувачі сприймають його пост. Наявність великої кількості коментарів свідчить про зацікавленість аудиторії в цій темі, тому такі пости легше знайти. Недоліком є аналогічне до постів обмеження: кількість символів не має перевищувати 280.

Наступною важливою функцією є комунікація в чатах. В той час як пост є одностороннім повідомленням та зазвичай націлений на доволі велику аудиторію, повідомлення, надіслане в чат, стосується лише його користувачів. Таким чином, можна створювати невеликі групи, залежно від інтересів або професійної діяльності і швидко та зручно обмінюватися у них повідомленнями. Перевагою також є відсутність ліміту на довжину повідомлення.

Ще однією важливою функцією є швидке знаходження користувачів за інтересами. Даний функціонал дозволяє знаходити людей, що поділяють погляди користувача, цікавляться однаковими речами або мають подібні захоплення. Це дозволяє швидко розширити своє коло знайомств та знаходити співбесідників на різноманітні теми. Особливо це важливо для спеціалістів, що шукають джерела інформації для поліпшення власних знань або готові стати ментором та поділитися знаннями та досвідом з іншими користувачами. Навіть досвідчені спеціалісти потребують допомоги при ознайомленні з новими

інструментами, або поради щодо того, який саме з існуючих підходів доцільно використати для розробки певного програмного продукта.

Facebook [8] – на сьогоднішній день найбільша соціальна мережа. Кількість її активних користувачів сягає 3 млрд [9]. Забезпечує обмін інформацією, фото та файлами між людьми, що знаходяться в будь-яких куточках світу.

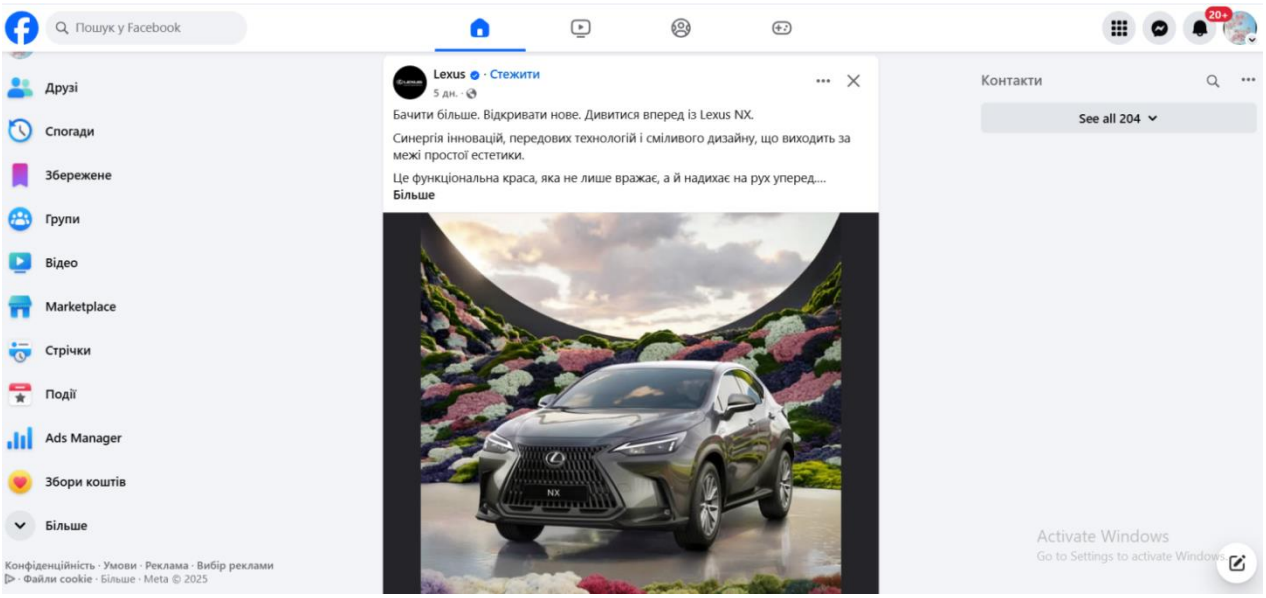

Рисунок 1.2 – Інтерфейс платформи Facebook

Платформа надає функціонал для публікації постів, що, враховуючи кількість користувачів застосунку, стає потужним інструментом для поширення власних ідей. Саме тому величезна кількість компаній, що мають на меті підтримувати зв’язок зі своїми користувачами мають профілі у Facebook. На відміну від X кількість символів у одному пості може сягати 63,206 [10]. Це дозволяє розміщувати набагато більше інформації, хоча найкращий розмір поста для більшого охоплення вважається до 250 символів.

Facebook також підтримує функцію коментарів. Перевагою є можливість надсилати коментарі довжиною до 8,000 символів, що перевищує ліміт, встановлений X [11]. Існує можливість комунікації в чатах. Недоліком є

перенаправлення користувача на іншу платформу – Messenger, що створює додаткові труднощі.

StackOverflow [12] – онлайн платформа, основною цільовою аудиторією якої є програмісти. Платформа дозволяє знайти відповіді на найрізноманітніші питання в сфері IT. Інтерфейс StackOverflow наведено на рисунку 1.3.

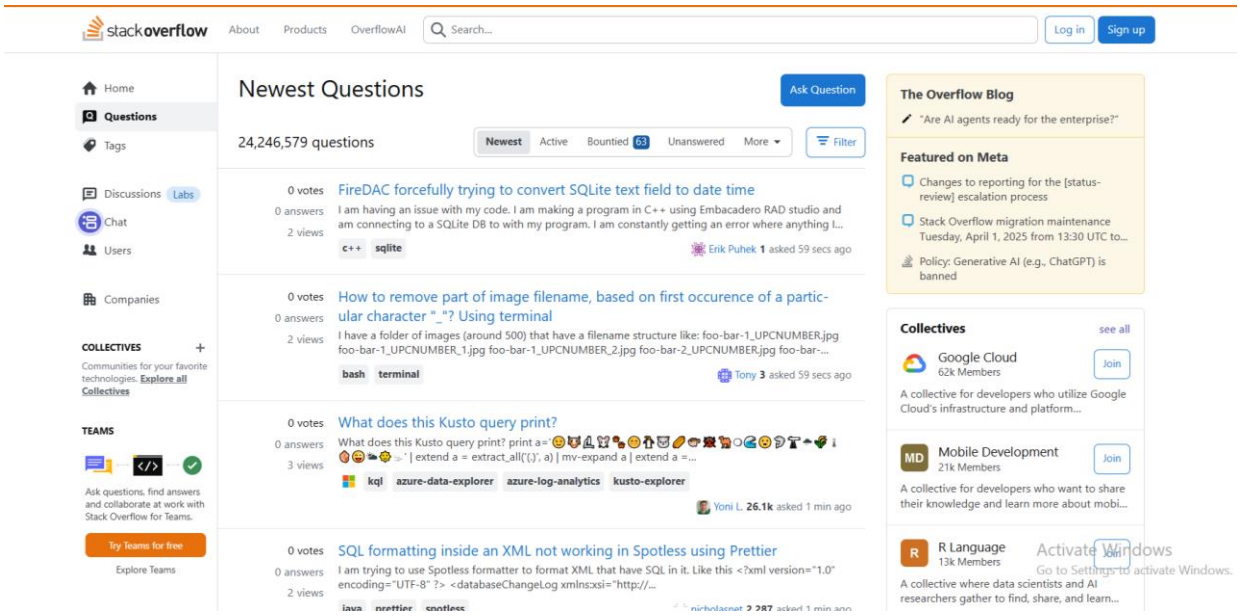

Рисунок 1.3 – Інтерфейс платформи StackOverflow

Найбільш популярною функцією є отримання допомоги від користувачів платформи у вирішенні завдань, пов'язаних з розробкою програмного забезпечення. Для цього потрібно опублікувати пост з запитанням, користувачі, які знатимуть на нього відповідь залишатимуть свої коментарі під цим постом. Коментарі, в яких будуть зазначені дієві поради щодо виконання завдання, отримуватимуть більший рейтинг від користувачів, тому в першу чергу варто скористатися ними. Цей функціонал є надзвичайно важливим, оскільки дозволяє розробникам з меншим досвідом скористатися допомогою професіоналів. Важливим елементом є звичне для програмістів підсвічення коду в повідомленнях. Такий механізм забезпечує максимальну схожість коду в повідомленні з кодом в IDE, де програмісти зазвичай працюють, що дозволяє

спростити сприйняття повідомлення з кодом та пришвидшити його опрацювання.

Наявна функція знаходження користувача. Це надзвичайно важливо, оскільки програміст зможе спитати поради конкретного, більш досвідченого користувача, але на жаль пошук здійснюється лише по імені.

Переваги та недоліки перелічених платформ зведено до таблиці 1.1

Таблиця 1.1 – Порівняльний аналіз аналогів

Характеристики	X	Facebook	StackOverflow	Платформа для комунікації
Публікація посту	+	+	+/-	+
Комунікація в чатах	+	+	+	+
Написання коментарів	+	+	+	+
Підсвітка коду	-	-	+	+
Оголошення про проєкти	-	-	-	+
Пошук користувачів	+	+	-	+
Загальна оцінка	70%	70%	60%	100%

В результаті порівняння виявлено, що такі платформи як X та Facebook приділяють належну увагу забезпеченню користувача усіма необхідними інструментами для зручної комунікації, але вони охоплюють велику аудиторію та не є спеціалізованими. Програмісти потребують додаткової функціональності. Одна з них – забезпечення підсвітки коду, опублікованого в повідомленні. Це зробить роботу з текстом набагато зручнішим та забезпечить швидкість роботи користувача.

StackOverflow, навпаки, орієнтується на програмістів, але не забезпечує основних інструментів для зручної комунікації. Користувач може публікувати пости на даній платформі лише у формі питань, що звужує коло потенційних тем, на які користувачі зазвичай пишуть на подібних платформах. Крім того, StackOverflow дозволяє знаходити користувача лише по імені, що ускладнює пошук нових знайомств у незнайомій сфері.

Також важливою потребою для програмістів є знаходження нових проєктів. Особливо це питання є актуальним для спеціалістів, що володіють необхідними знаннями та прагнуть здобути свій перший досвід роботи над складними проєктами. Для задоволення цієї потреби необхідно реалізувати функцію повідомлення про нові проєкти, до виконання яких можна долучитися.

В результаті проведення аналізу аналогів зроблено висновок, що наявні платформи мають як переваги, так і серйозні недоліки. Постає необхідність у розробці нового застосунку, який би задовольнив потреби користувача у цій сфері.

1.3 Аналіз методів розв’язання завдань

Цілі роботи можна досягнути декількома шляхами. Кожен з них надає особливі можливості, але також вимагає додаткових ресурсів або накладає певні обмеження на розробника.

Створення онлайн платформи для комунікації може бути здійснене на основі вже існуючих продуктів, які надають базовий функціонал для застосунку, з використанням конструкторів для вебсайтів та за допомогою сучасних вебфреймворків, таких як React, Angular та Vue.js на фронтенді та мови програмування для серверної частини.

До першої категорії можна віднести такі платформи як SendBird та ConstusFly. SendBird – платформа, що надає сайтам та мобільним застосункам функціонал з обміну повідомлень. Вона надає API, за допомогою якого можна надсилати, переглядати, змінювати повідомлення та створювати групи. При цьому платформа реалізує всю бізнес логіку застосунку, користувачу лише

потрібно правильно використовувати її API. Вся інформація зберігається на серверах платформи, що спрощує роботу розробника, оскільки не потрібно самому продумувати та реалізовувати збереження даних користувачів. Щоб скористатися функціоналом платформи необхідно зареєструватися на сайті компанії та створити новий застосунок. Далі, використовуючи отриманий API ключ, можна написати код, який використовує SendBird. Приклад запиту на створення нового чату продемонстрованого на сайті компанії наведено на рисунку 1.4 [13].

Рисунок 1.4 – Запит на створення нового чату

ContusFly – інструмент, що надає готові компоненти для програм, які дозволяють користувачам комунікувати між собою. Схожий на SendBird, але має певні переваги. Наприклад він дозволяє збереження інформації не лише на серверах компанії, але й на серверах розробників або третьої сторони.

По суті, наведені платформи виключають необхідність розробки серверної частини застосунку, оскільки програміст може скористатися вже готовими продуктами. Такий підхід має велику кількість переваг. Використання існуючих інструментів пришвидшує час розробки, відпадає необхідність у виконанні цілого ряду задач.

Одним з головних таких завдань є організація збереження інформації. Застосунки для комунікації працюють з великою кількістю інформації, яка може бути представленою у вигляді текстових повідомлень, картинок, медіа та інших файлів. При організації процесу збереження цієї інформації можуть виникнути складнощі. Саме тому передача такого завдання третій стороні пришвидшує та спрощує розробку.

Ще однією складністю з якою зазвичай зустрічаються розробники – це розгортання серверної інфраструктури. Користувачів платформи для комунікації може бути велика кількість, тому вимоги до серверів на яких вона буде працювати досить високі. Але, оскільки обробка та збереження інформації відбувається на іншій платформі, SendBird чи ContusFly, то й завдання з розгортання перекладаються на розробників цієї платформи.

Переваги цих продуктів суттєві, але їх використання несе за собою численні недоліки.

Найбільшим недоліком є залежність від постачальника. При створенні програмного забезпечення розробники віддають велику частку функціоналу на реалізацію сторонній компанії. Відповідно код застосунку буде дуже сильно залежати від стороннього інструмента, зміна його на інший може виявитися дуже складним процесом, що вимагатиме багато зусиль та часу.

Ще однією вадою є те, що такі інструменти як SendBird надають вже готовий перелік функціональних можливостей, але вони можуть не повністю задовольняти розробників застосунку. Особливо складною ситуація стає, якщо такий недолік виявлено вже під час розробки, коли більша частина коду написана, що робить заміну обраного інструменту набагато складнішою операцією.

Також важливо зазначити, що використання інструментів, які надають готовий функціонал, супроводжується використанням даних клієнтів третьою стороною. Важко переоцінити важливість даного питання. Якщо компанія, що постачає інструмент, використає ці дані некоректним чином, це може спричинити репутаційні збитки для тих, хто користується її послугами.

Категорія конструкторів для вебсайтів дуже широко представлена. Одним з найкращих її представників є WordPress. Цей інструмент дозволяє налаштовувати зовнішній вигляд вебсайтів, а велика кількість плагінів надають можливість підключити необхідний функціонал.

WordPress спрощує створення як серверної частини так і фронтенду. Це дозволяє знизити до мінімуму час реалізації проєкта. Але, на жаль, такий підхід має і низку недоліків.

WordPress є дуже популярною платформою, що використовується для побудови великої кількості сайтів. Саме тому вона є цінним об’єктом для кібератак. Крім того, сторонні плагіни, які використовуються в WordPress, можуть мати свої вразливості, що підвищує ймовірність успішної атаки.

Система плагінів є потужним інструментом WordPress, але такий підхід ускладнює розробку та підтримку програмного продукту. Плагіни необхідно постійно оновлювати, хоча б заради безпеки застосунку. Оновлення можуть бути несумісними з іншими плагінами, що призведе до конфліктів, які необхідно буде усувати.

Використання плагінів для обробки, збереження та передачі інформації про користувачів та їх повідомлень також збільшує ризик втрати або витоку цих даних. Щоб цього уникнути потрібно додатково перевіряти кожен плагін на те, наскільки він є безпечним.

Останнім варіантом створення онлайн платформи для комунікації програмістів є використання вебфреймворків для фронтенду та мови програмування для серверної частини. Такий підхід має низку переваг.

Найбільша перевага є повний контроль над функціоналом застосунку. На відміну від попередніх варіантів не потрібно використовувати плагіни або інші

існуючі продукти, які мають власні функціональні обмеження. Відкривається можливість реалізувати будь-які необхідні для кінцевого користувача функції, навіть якщо вони не відповідають вже існуючим шаблонам.

Ще однією важливою перевагою є повний контроль над даними. Вони ні в якій формі не передаються іншим сторонам. Весь процес обробки і збереження даних контролюється розробниками застосунку. Вони можуть імплементувати такі механізми захисту, які вважають за потрібне або які вимагаються законодавством регіонів у яких працюватиме цей застосунок.

Також важливо те, що програмісти самостійно можуть вибирати технології, за допомогою яких реалізовувати застосунок, здійснювати обробку та збереження даних. Для таких програм як онлайн платформа для комунікації це є надзвичайно важливо, оскільки не всі технології забезпечують необхідну швидкість та надійність.

Відсутність сторонніх інструментів може зекономити кошти, які були б оплачені за їх використання. Це особливо важливо для довгострокових проєктів, оскільки плата зазвичай знімається кожен місяць.

Повний контроль над проєктом поширюється також на його розгортання. У такому випадку команда сама обирає на яких серверах та яким способом розгортається створений програмний продукт. Переваги такого підходу дуже відчутні для високонавантажених систем, до яких можна віднести платформу для комунікації.

До недоліків можна віднести велику кількість деталей, яку потрібно знати, розуміти та реалізовувати на цьому рівні. Це вимагає вищого рівня кваліфікації та більше часу для виконання.

Отже, проаналізувавши різні способи створення онлайн платформи для комунікації програмістів, можна дійти висновку, що найкращим способом є використання останнього підходу, а саме: за допомогою сучасних вебфреймворків, таких як React, Angular та Vue.js на фронтенді та мови програмування для серверної частини.

Хоча такий підхід займає більше часу та вимагає глибших знань, він надає повну свободу у виборі технологій, дозволяє реалізовувати нестандартний функціонал та забезпечує повний контроль над даними користувачів.

1.4 Висновок

Перший розділ був присвячений аналізу актуальності розробки, пошуку аналогів, порівняння їх переваг та недоліків, аналізу методів реалізації функціоналу застосунку.

Під час аналізу актуальності розробки визначено основні причини популярності платформ для комунікації та зроблено висновок що кількість користувачів, зацікавлених у таких платформах лише зростатиме.

Для порівняння обрано такі платформи як: X, Facebook та StackOverflow. Вони є дуже якісними програмними продуктами, але мають недоліки. X та Facebook охоплюють велику аудиторію, але не забезпечують необхідний функціонал для більш вузьких аудиторій, такі як програмісти. StackOverflow приділяє належну увагу цій аудиторії, але не підтримує низку інших важливих функцій, наприклад знаходження користувача можливе лише за логіном.

Внаслідок аналізу методів реалізації програмного продукту зроблено висновок, що найкращим варіантом є використання вебфреймворку для фронтенду та мови програмування для серверної частини. Хоча такий підхід складніший та вимагає більше часу та вмінь, він забезпечує необхідну гнучкість та вільний вибір засобів для розгортання коду.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ, ВИБІР АРХІТЕКТУРИ

2.1 Аналіз функціональних вимог до застосунку

Першим етапом розробки програмного забезпечення є збір та аналіз вимог. Головною метою аналізу вимог є розуміння потреб користувачів [14]. Важливість аналізу функціональних вимог важко переоцінити, саме тоді формується перше уявлення про систему, регламентується що вона має робити, які очікування у користувачів щодо неї. Залежно від створюваної системи її складності та критичності, цей етап може займати більше або менше часу, бути більш або менш формалізованим, але він обов’язково присутній. Це пояснюється низкою переваг, які надає коректне виконання завдання збору та аналізу функціональних вимог.

Наявність списку добре сформульованих вимог дозволяє зрозуміти об’єм робіт над проєктом, здійснити якісне планування та розподіл завдань. Крім того, ретельно проаналізувавши документ вимог можна виокремити об’єкти з яких складається система, що є першим кроком до проєктування програмного забезпечення.

Аналіз вимог дозволяє отримати глибше розуміння потреб та очікувань замовника. Це надає можливість уникнути непорозумінь під час розробки програмного продукту [15].

Вимоги також відіграватимуть важливу роль під час етапу тестування застосунку, оскільки саме вони визначатимуть наскільки розробка задовольняє замовників та користувачів, а отже наскільки успішною або неуспішною вона є. Потреба у змінах та доопрацюванні також визначається на підставі сформованих вимог.

Тому для успішного виконання проєкту «Онлайн платформа для комунікації програмістів» система має включати наступний фнукціонал.

Користувач може зареєструватися в системі. Якщо він вже є зареєстрованим, він може скористатися функцією авторизації.

Користувач, за бажанням, може ввести дані про себе, які на його думку є релевантними. Дані можуть включати ім’я, прізвище, деякі факти з біографії та список інтересів.

Користувач може змінювати дані про себе. Система має унеможливлювати зміну даних користувача будь-якою іншою сторонньою особою.

Кожен користувач може викласти свої думки щодо певної теми у вигляді поста. Всі інші користувачі, що є підписаними на даного користувача матимуть можливість переглянути його.

Користувач може вносити зміни до свого вже опублікованого поста, а також видаляти його. Спроба зміни чи видалення поста стороннім користувачем має бути заблокована.

Кожен користувач, прочитавши пост, може його прокоментувати. Користувач-власник поста та інші можуть згодом переглянути коментарі. Також є можливість писати коментарі на коментарі.

Користувач може змінювати та видаляти написані ним коментарі. Спроба зміни чи видалення коментаря стороннім має бути заблокована.

Кожен користувач може створити чат та додати до нього інших користувачів. Таким чином створюється віртуальний простір, де за допомогою повідомлень можна обмінюватися думками та інформацією.

Користувач, який створив чат, може вносити до нього зміни. Зміни можуть стосуватися як загальної інформації про чат, так і користувачів, які мають до нього доступ. Зміни чату іншими користувачами мають бути відхилені.

Користувач може надіслати повідомлення в чат. За потреби, він може внести зміни до повідомлення або видалити його. Спроби зміни та видалення повідомлення користувачем, який не є автором поста не допускаються.

Програміст може мати необхідність поспілкуватися з іншим користувачем, який поділяє його інтереси. Для цього необхідна функція пошуку користувачів за даними, що він про себе опублікував. Знайшовши необхідного користувача програміст може підписатися, переглядати його пости або запросити його в нову групу.

Програміст може публікувати інформацію про новий проєкт. Користувачі що зацікавлені, можуть брати участь в реалізації даного проєкту. Інформація про проєкт має містити логін автора, назву, детальний опис, перелік технологій, які будуть використовуватися на цьому проєкті.

Автор проєкту може вносити зміни до інформації про проєкт або видаляти його. Зміни внесені будь-яким іншим користувачем мають бути відхилені.

2.2 Розробка візуальної моделі системи для комунікації програмістів

Моделювання є важливим етапом у процесі створення програмного забезпечення. Воно представляє систему, її структуру та поведінку, в спрощеному вигляді, що дозволяє всіх учасникам процесу розробки мати однакові та максимально точні уявлення про те, якою вона має бути. Якісна модель дозволить правильно підібрати архітектуру системи, а отже забезпечити гнучкість та масштабованість системи у майбутньому. Крім того, вона допомагає оцінити складність програми, що надзвичайно важливо при плануванні розробки та розрахунку часу та ресурсів, які вона займе. Більше того, модель може вводити офіційні терміни для об’єктів та процесів у системі, що робить її унікальним інструментом комунікації всередині команди розробників та між нею і замовниками чи користувачами.

Існують різні способи моделювання системи. Найпопулярнішим інструментом для їх відображення є UML діаграми. Модель системи візуалізовано у вигляді діаграми діяльності (рисунок 2.1).

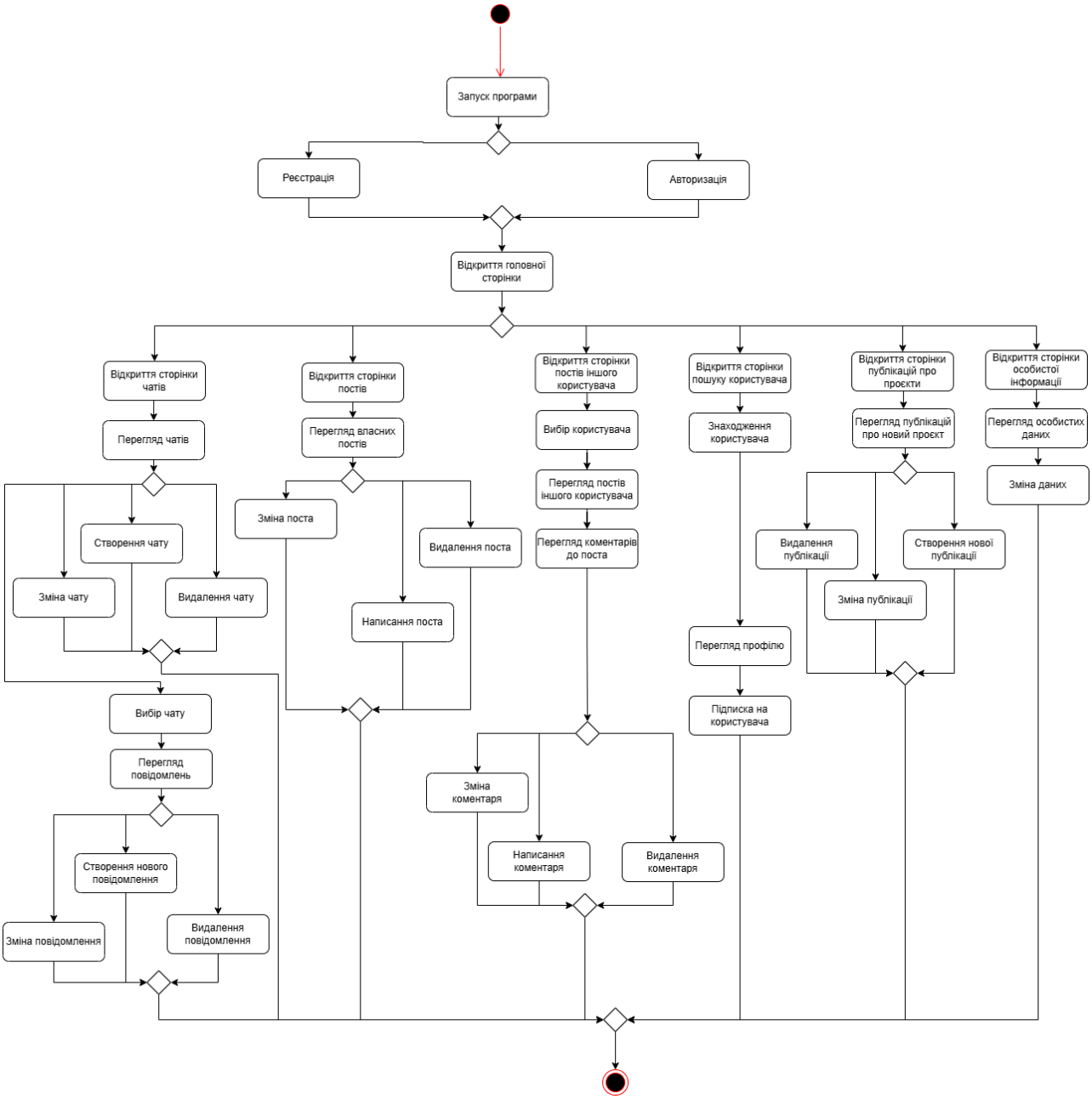

Рисунок 2.1 – UML модель діяльності системи

Можливі і інші моделі системи. Наприклад, структурна модель відображає компоненти, з яких складається програма. Діаграма станів вказує на те які стани система може приймати та події, які є причинами переходу між станами. Діаграма прецедентів показує які користувачі взаємодіють з програмним продуктом та які його функції використовують. Проте саме діаграма діяльності дозволяє зрозуміти які дії та в якій послідовності мають бути можливими в системі.

2.3 Вибір архітектурного рішення для розробки платформи

Для забезпечення зрозумілості та надійності системи, на етапі проєктування необхідно обрати її архітектуру. Коректно підібрана архітектура знизить витрати на розробку та підтримку проєкту. І навпаки, помилка може мати дуже серйозні наслідки і сповільнювати роботу на наступних етапах розробки програмного забезпечення.

По суті, архітектура вказує нам, на які компоненти необхідно розбити систему та як налагодити між ними зв’язок. Кожен компонент має своє завдання та об’єднує код, який його виконує. Для забезпечення надійності та повторного використання вже реалізованих функцій, код всередині компонента має мати мінімальні зв’язки з кодом інших компонентів. Таким чином можна досягти абстрагування компонента від решти програми та інкапсуляції його стану.

Існує багато підходів до реалізації архітектурни застосунків, які дозволяють створювати зрозуміле та структуроване програмне забезпечення. До них належать:

1. MVC.
2. Чиста архітектура.
3. Мікросервісна архітектура.
4. Архітектура, що базується на подіях.

Все в архітектурі програмного забезпечення є компромісом [16]. Це означає, що кожен підхід має свої переваги та недоліки. Кінцевий вибір залежить від системи, що знаходиться в розробці, її функціоналу, складності та важливості. Оскільки онлайн платформа для комунікації програмістів має бути надійною, стійкою до помилок, підтримувати роботу з великою кількістю користувачів, то найкращим вибором є мікросервісна архітектура.

Мікросервіси це сервіси, що розгортаються незалежно один від одного та створюються навколо бізнес-домену [17]. Це спонукає до розробки максимально незалежних один від одного компонентів з мінімальними зв’язками. Крім того, якщо на один з модулів припадає велика кількість

запитів, його можна масштабувати незалежно від інших. Також, при правильній реалізації, помилки, які виникатимуть в максимально незалежних модулях не матимуть суттєвий негативний вплив на всю систему.

Саме тому, для розробки онлайн платформи для програмістів обрано мікросервісну архітектуру.

2.4 Розробка алгоритмів роботи системи

Розробка алгоритмів є важливим етапом розробки програмного забезпечення. Алгоритми вказують чітку послідовність кроків, які має зробити програма для того, щоб отримати бажаний результат. Під час розробки алгоритму програміст абстрагується від деталей реалізації, які впливали б на нього під час кодування, та зосереджується на самому алгоритмі, а отже приділяє більше уваги його коректності. Це дозволяє знайти найкращі підходи для вирішення завдання, уникнути багатьох помилок та витрат додаткового часу та ресурсів. Крім того, запис найбільш важливих алгоритмів у вигляді блок-схем дозволяє покращити документування програмного продукту. Це особливо важливо, коли на проєкт залучають нових фахівців. Інколи проєкт може бути настільки великим та складним, що й розробникам якоїсь його частини необхідно витрачати багато часу для того, щоб зрозуміти як працюють інші компоненти проєкту.

Алгоритм наведений на рисунку 2.2 та 2.3 відповідає за створення нового користувача. На вхід функції передають такі значення як логін користувача, його ім’я, прізвище та пароль. Спочатку відбувається перевірка на наявність користувача з таким логіном. Якщо такий користувач існує в поле помилки встановлюється відповідне значення. Якщо ж обраний логін вільний, відбувається хешування пароля щоб зробити його збереження в базі даних більш безпечним. Наступним кроком створюється відповідний запис про нового користувача в базі даних. Після цього створюється токен, який разом з ідентифікатором користувача та полем помилки повертається з функції.

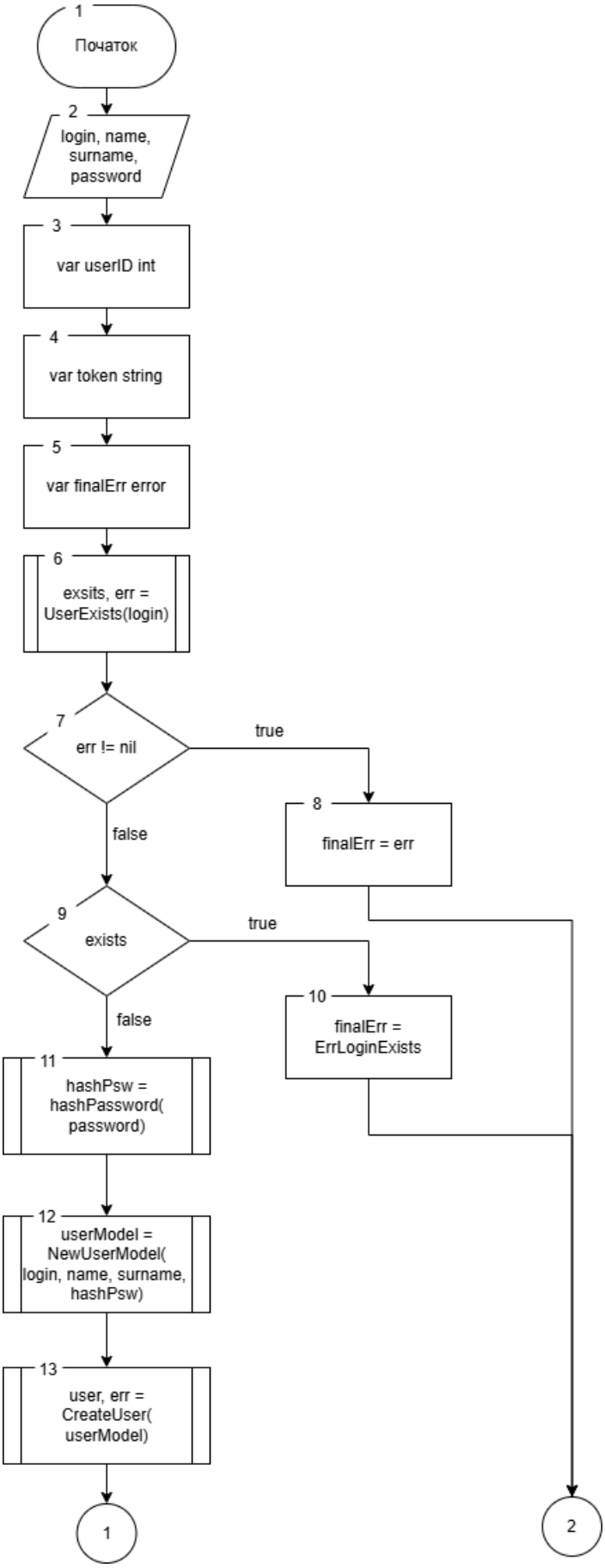

Рисунок 2.2 – Алгоритм створення нового користувача

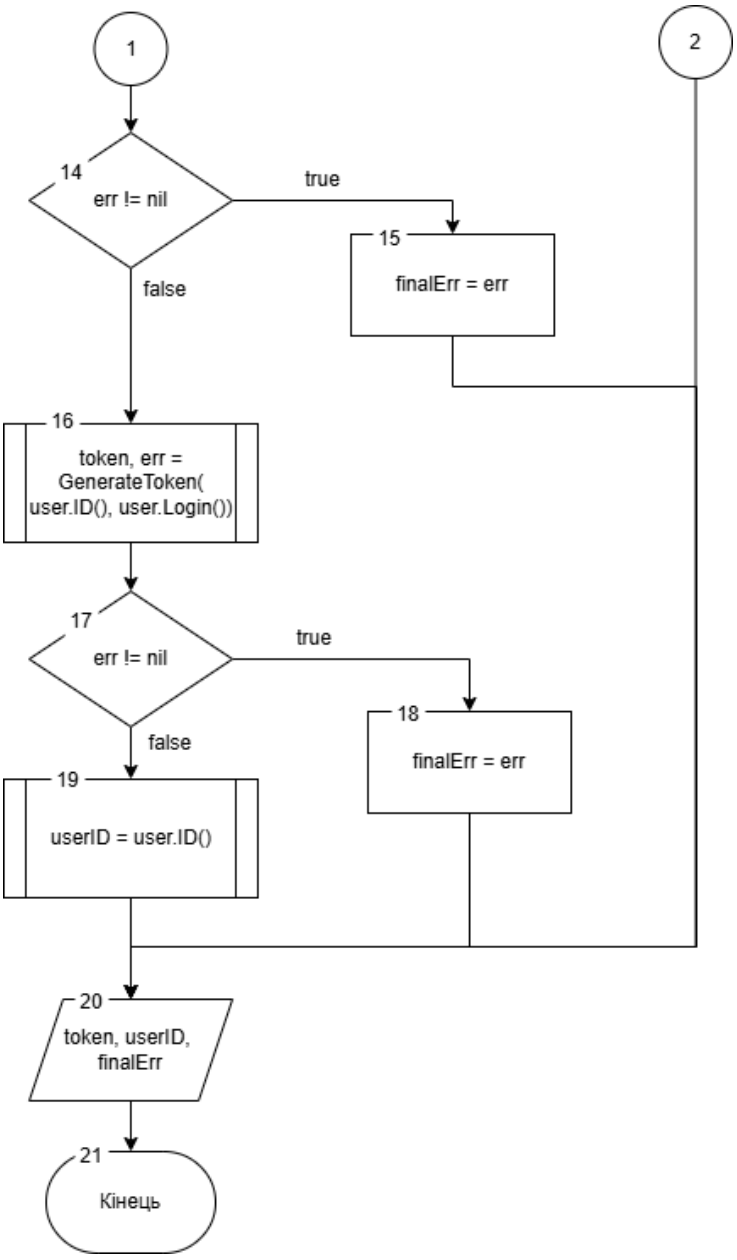

Рисунок 2.3 – Продовження алгоритму створення нового користувача

Останнім кроком стає створення нового токена. Він слугує ключем, який підтверджує, що користувач є дійсно зареєстрованим, і додається до кожного запиту, який фронтенд відправляє на сервер. Функція повертає такі значення: токен, ідентифікатор новоствореного користувача та помилку.

Наступним важливим алгоритмом є алгоритм визначення видалених та доступних ресурсів (рисунок 2.4). Автор поста, коментаря або повідомлення може за допомогою графічного інтерфейса користувача редагувати його, тому

на серверній частині необхідно зрозуміти які картинки чи файли було видалено, а які залишаються актуальними.

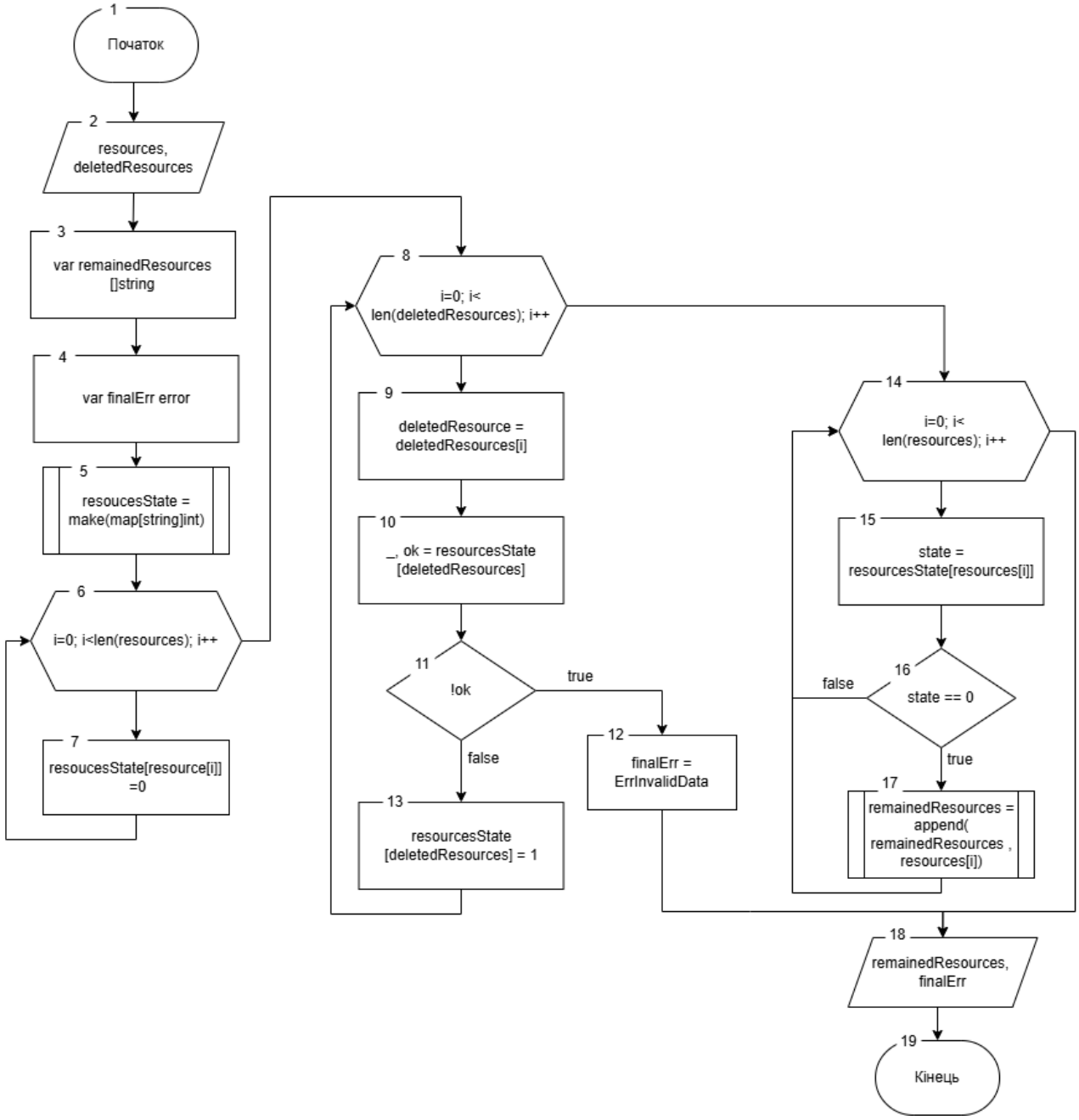

Рисунок 2.4 – Алгоритм визначення видалених та доступних ресурсів

Наведений алгоритм приймає на вхід списки всіх ресурсів та ресурсів, які було видалено. Кінцевим результатом є список ресурсів, що залишилися актуальними. Також відбувається перевірка, чи всі видалені ресурси належать

до одного поста. Для цього створюється хеш-таблиця, ключами до якої виступають назви файлів, а значенням їх стан. На початку виконання стан всіх файлів позначено як «Створений». Наступним кроком в циклі перебираються видалені файли. Якщо видалений файл є в хеш-таблиці, отже пост має такий файл і його стан переходить у «Видалений». Якщо ж файла в хеш-таблиці немає, це означає, що відбулася спроба видалення файлу, який немає відношення до поста, тому видається помилка. Далі ключі хеш-таблиці стан яких залишився «Створений» додаються до окремого списку, який вкінці повертається з функції.

2.5 Висновок

У цьому розділі визначено функціональні вимоги до застосунку, розроблено його модель, обрано архітектуру та проаналізовано основні алгоритми системи.

У функціональних вимогах застосунку наведено перелік можливостей, які система має надавати користувачу. Розроблено діаграму діяльності системи, що візуалізує потік подій, які відбуваються в застосунку. Наведено аргументи на користь вибору мікросервісної архітектури для онлайн платформи. Останній підрозділ, присвячений алгоритмам, описує алгоритми програми, такі як: алгоритм створення нового користувача та алгоритм визначення видалених та доступних ресурсів.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ОНЛАЙН ПЛАТФОРМИ ДЛЯ КОМУНІКАЦІЇ ПРОГРАМІСТІВ

3.1 Варіантний аналіз та обґрунтування вибору засобів реалізації застосунку

Для розробки серверної частини коду було обрано мову програмування Go. Оскільки слово «go» є поширеним в англійській мові, альтернативною назвою мови програмування є Golang. Вона була створеною компанією Google. Її творцями стали: Робер Ґрізмер, Роб Пайк та Кен Томпсон [18]. Вперше оприлюднений 2009 року. Golang широко використовується у хмарних обчисленнях, програмуванні серверів, створені інтерфейсів командного рядка та як інструмент для DevOps [19]. До компаній, які використовують цю мову програмування входять American Express, Meta, Google, Microsoft, Capital One, Cloudflare, Dropbox та багато інших підприємст [20]. За даними опитування платформи DOU [21] Golang лідирує серед мов, які розробники прагнуть вивчити наступного року (рисунок 3.1).

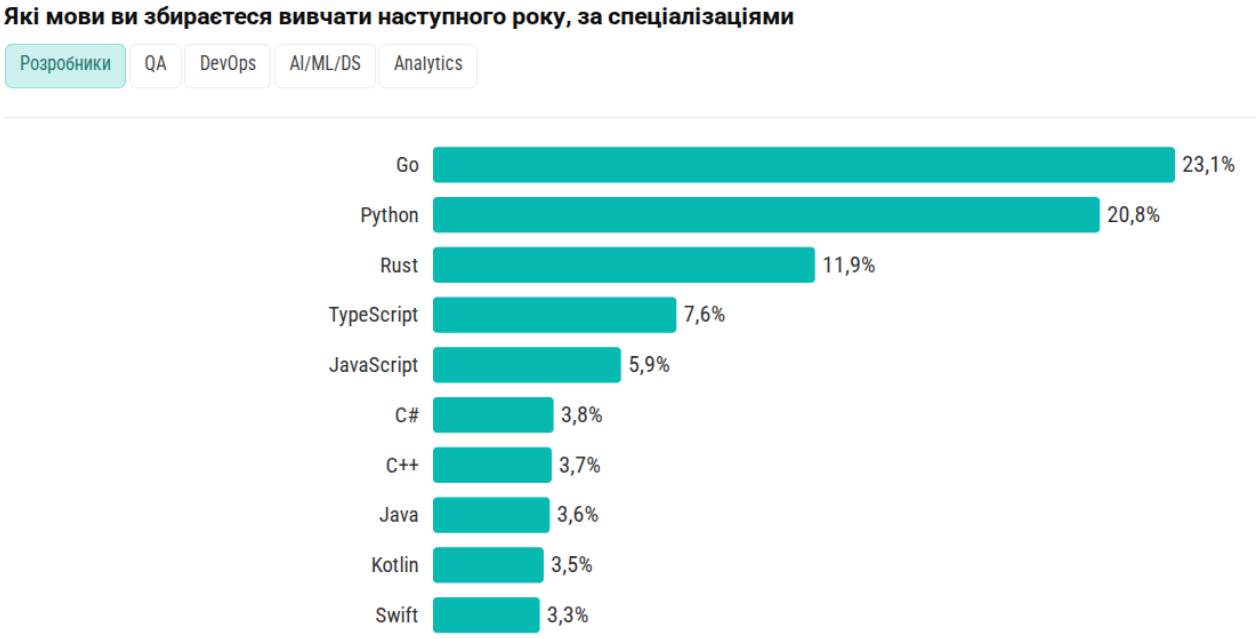

Рисунок 3.1 – Мови, які розробники збираються вивчити [21]

Високі позиції Golang займає і в індексі уподобання (рисунок 3.2). Це спеціальний індекс, який вказує на те, наскільки розробники задоволені тією чи іншою мовою програмування. Golang лише дещо поступається Rust та Kotlin.

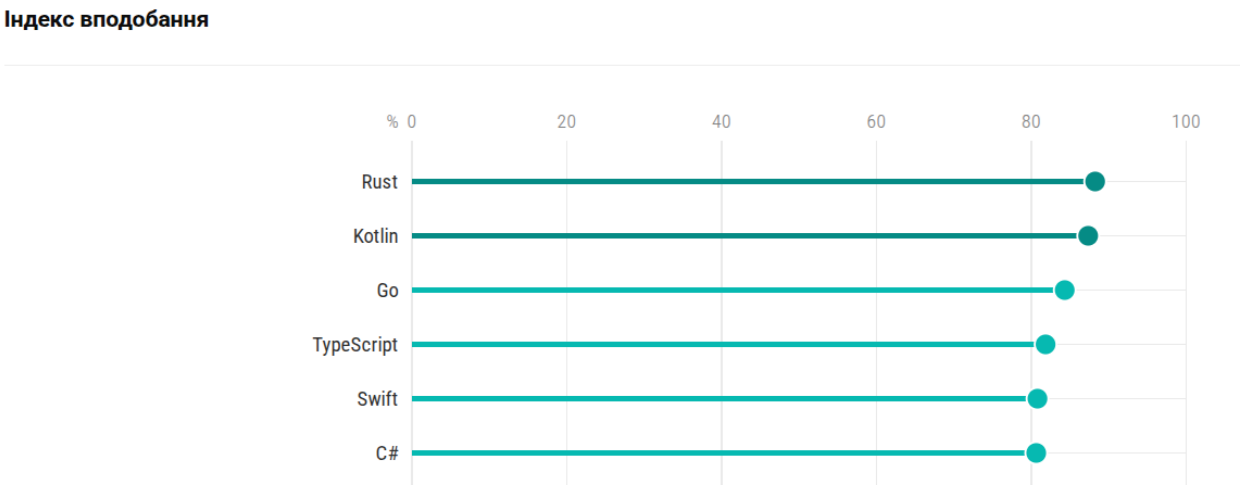

Рисунок 3.2 – Індекс вподобання мов програмування [21]

Ще одне цікаве питання, що входило в опитування DOU, було щодо вибору мови програмування для нового проєкту (рисунок 3.3). Незважаючи на порівняно молодий вік Golang зайняла п’яте місце обігнавши такі популярні мови як Java та PHP.

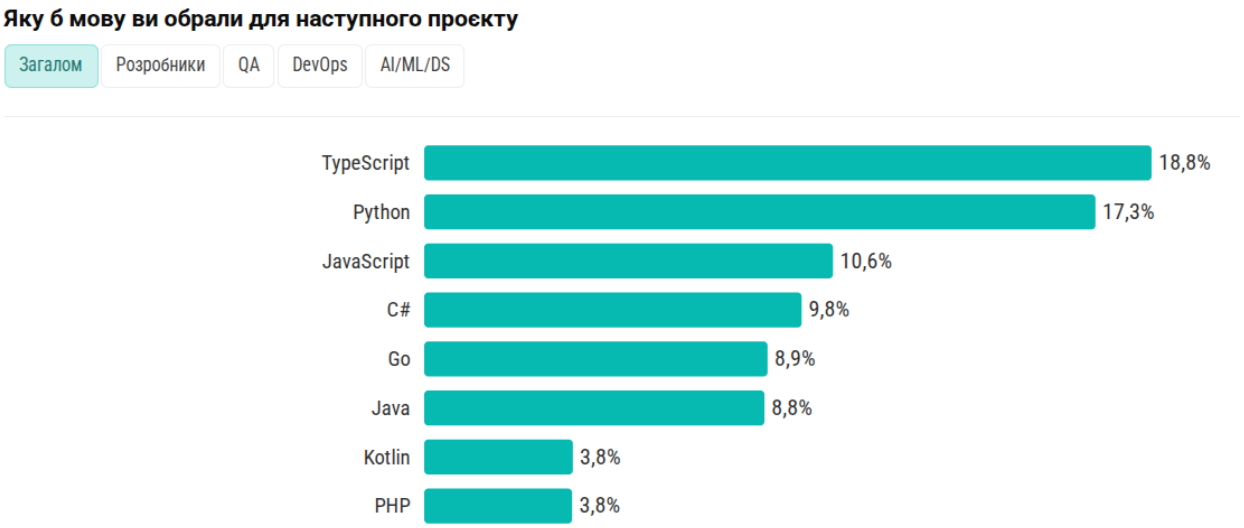

Рисунок 3.3 – Мови, які розробники збираються використати на новому проєкті [21]

Причиною такої популярності є низка особливостей цієї мови програмування. Golang є компільованою мовою програмування. Це означає, що перед виконанням вона компілюється в машинний код, тому під час виконання, вона не потребує додаткових інструментів, таких як інтерпретатор або віртуальна машина. Це дозволяє їй досягти високих показників у швидкості виконання, що робить можливим використання даної мови для програмування високонавантажених застосунків. Щоб краще зрозуміти, перевагу Golang у таблиці 3.1 наведено порівняння часу виконання наведених операцій між Go та Python [22].

Таблиця 3.1 – Порівняння часу виконання операцій з використанням Golang та Python

Найменування операції	Golang	Python	Одиниці вимірювання
Бінарний пошук	20.8	2442.1	нс/оп
Сортування бульбашкою	0.09	6.7	с/оп
Читання з файлу	5305	58359	нс/оп
Опрацювання HTTP запиту	0.07	1.26	мс/запит

З таблиці видно, що швидкість виконання операцій за допомогою Go набагато більша ніж якби для виконання цих операцій використовувався Python.

Ще однією перевагою цієї мови є статична типізація. Статична типізація це перевірка типів змінних під час компіляції, а не під час виконання. Такий підхід дозволяє компілятору перевірити відповідність типів ще до запуску програми та виявити багато помилок. Це покращує надійність коду, оскільки якби ці помилки виникали під час безпосереднього виконання, програма

починала б некоректно працювати або закінчила б своє виконання з помилкою. Особливо це важливо для складних проєктів, що мають велику кількість змінних, тому запам’ятати тип більшості з них надзвичайно складно. Крім того, статична типізація покращує сприйняття коду. Так як тип змінної явно вказаний, програміст, що ознайомлюється з кодом краще розуміє її призначення. Вагомим фактором є також покращення продуктивності. Оскільки компілятор знає тип змінних, він може визначити скільки пам’яті необхідно виділити для неї.

Наявність збірника сміття також є позитивною стороною Golang. Хоча його періодичний запуск дещо сповільнює виконання коду, він дозволяє спростити управління пам’яттю. Альтернативою цьому є ручне управління пам’яттю, реалізоване в C та C++. Хоча такий підхід забезпечує максимально можливу швидкість виконання, процес розробки ускладнюється, оскільки програміст має не забувати звільняти виділену пам’ять, інакше виникатиме витік пам’яті. Ще одним підходом до управління пам’яттю є система ownership в Rust. Система ownership це набір правил, які визначають, як Rust управляє пам’яттю [23]. Код під час компіляції перевіряється на відповідність цим правилам. Якщо будь-яке з них не дотримано, програма не скомпілюється. Коли ж код повністю відповідає вимогам мова програмування розуміє, коли необхідно видалити об’єкт з пам’яті, під час виконання без використання збірника сміття. Це усуває витоки пам’яті, наявні в C та C++, але слідування правилам Rust перетворює розробку програмного забезпечення на цій мові на дуже складний процес, що збільшує час та ресурси необхідні на реалізацію проєкту.

Важливою перевагою Golang є наявність вбудованого механізму конкурентного виконання програми. Це реалізується за допомогою goroutines. На відміну від концепції Threads, що використовується в Java або C#, виконанням конкурентних процесів управляє не операційна система, а сама мова програмування. Наслідком цього є низка переваг. Об’єм пам’яті необхідний для стеку goroutine сягає лише 2 кілобайт, і під час виконання

програми може збільшуватися [24]. Механізм конкуретності є вбудованим в стандартну бібліотеку Golang, і не потребує підключення сторонніх ресурсів, наприклад так як в Node.js. Комунікація між конкурентними процесами відбувається за допомогою каналів. Суть полягає в тому, що той самий канал передається в обидві goroutine, кожна з яких може здійснювати запис та читання даних з нього. Використання інших підходів, таких як робота зі змінними, що знаходяться у спільній пам’яті, не рекомендується, оскільки тоді виникає можливість одночасного доступу до тих самих змінних, що потребує вводу механізмів блокування.

Ключовою перевагою цієї мови програмування є простота. Вона виключає можливість розробити одну і ту ж саму річ безліччю різними способами. Це дозволяє створювати код максимально зрозумілий іншим розробникам. Go вводить суворі правила на розміщення дужок, використання імпортованих пакетів, використання створених змінних. Так, наприклад, неможливо створити змінну та не використати її, або імпортувати пакет без подальшого застосування. Результатом такого підходу є створення коду, який легко розробляти, розширювати та підтримувати.

Для програмування фронтенду обрано фреймворк Angular. Він дозволяє розбивати застосунок на окремі компоненти, які можна повторно використовувати. Крім того, на відміну від React, який покладається на додаткові бібліотеки, включає цілий набір інструментів, доступних «з коробки». Такі характеристики роблять Angular гарним інструментом для розробки складних систем.

Отже, використання мови програмування Golang зумовлено її простотою, швидкістю, статичною типізацією та наявністю механізмів конкурентного виконання. Angular обрано через можливість компонентної організації, та наявності великої кількості вбудованих інструментів.

3.2 Розробка модуля «Користувачі»

Модуль «Користувачі» є окремим мікросервісом системи, що забезпечує роботу функцій пов’язаних з обробкою даних та підписок користувачів системи. Для підвищення зрозумілості коду, цей модуль розділено на декілька частин. Компоненти transport та endpoint відповідають за прийом запиту та отримання з нього необхідних даних, service містить бізнес-логіку застосунку, repository працює з базою даних, в domain знаходяться об’єкти системи.

Найскладнішими елементами цієї системи є service та repository. Service приймає на вхід дані, які були отримані із запита користувача та виконує над ними дії необхідні для отримання кінцевого результату.

Функція CreateUser відповідає за створення нового користувача. Вона приймає на вхід інформацію про користувача, а саме: логін, ім’я, прізвище та пароль. Після чого вона перевіряє, чи існує вже користувач з таким логіном, якщо так то повертає повідомлення про помилку, якщо ж цей логін є новим, функція хешує пароль, щоб зробити його зберігання в базі даних безпечнішим та викликає функцію компонента repository CreateUser для занесення інформацію про користувача у базу даних. Далі генерується та повертається токен разом з ідентифікатором новоствореного користувача.

Функція Login викликається, коли користувач вже зареєстрований в системі та прагне авторизуватися. Приймає на вхід логін та пароль, введений користувачем. Для перевірки достовірності, отримує з бази даних хеш правильного пароля, хешує переданий пароль та звірює результати. Якщо вони співпадають, генерує токен, інакше повертає повідомлення про помилку. Функція повертає токен та ідентифікатор користувача.

Функція GetUser відповідає за отримання інформації про користувача. Вона приймає на вхід токен та ідентифікатор користувача, інформацію про якого треба знайти. Далі відбувається перевірка токену, якщо він дійсний, за допомогою функції GetUser компонента repository отримується інформація про користувача та повертається з функції.

Функція UpdateUser оновлює дані користувача. До інформації, яку можна оновити відносяться ім’я, прізвище, інтереси та біографічні дані. Функція приймає їх, а також токен на вхід. Далі відбувається перевірка токену, і якщо вона успішна оновлення даних.

Функція GetUsersByInfo повертає користувачів, які мають логін, інтереси або біографічні дані схожі на текст, який вона приймає на вхід.

Функція GetFollowedUsers отримує інформацію про всіх користувачів, на які підписаний автор запиту. Вона приймає на вхід токен, перевіряє його та отримує необхідну інформацію за допомогою GetFollowedUsers компонента repository.

Функція FollowUser підписує одного користувача на іншого. Вона приймає на вхід токен та ідентифікатор користувача, на якого треба підписатися. Відбувається перевірка токену та перевірка на наявність підписки, якщо токен недійсний або підписка вже існує повертається повідомлення з описом помилки. Інакше додається нова підписка.

Функція UsersInfo отримує на вхід перелік ідентифікаторів користувачів та за допомогою компонента repository повертає інформацію про кожного з них.

Програмний код описаних функцій наведено в додатку В.

3.3 Розробка модуля «Чати»

Модуль «Чати» також є окремим мікросервісом системи. Він відповідає за додавання, отримання та оновлення інформації про чати, в яких можуть переписуватися користувачі. Цей модуль також розділено на декілька частин для кращого розуміння коду. Він містить такі компоненти як: server, service, repository та domain. Оскільки даний модуль за складністю є простішим ніж модуль «Користувачі», за отримання даних з запиту відповідає лише компонент server. Компонент service абстрагується від таких речей, як запити та бази

даних, і містить бізнес-логіку модуля. Компонент repository відповідає за роботу з базою даних. Компонент domain містить об’єкти програми.

Найскладнішими елементами також є компоненти service та repository.

Функція CreateChat відповідає за створення чату, який буде використаний програмістами для комунікації. Вона приймає на вхід структуру, що має такі поля: токен користувача, який надіслав запит на створення чату, назву чату та перелік ідентифікаторів користувачів, які будуть його використовувати. Далі функція здійснює перевірку отриманого токена, якщо він недійсний, повертає повідомлення про помилку. Якщо ж з ним все гаразд, використовується функція CreateChat з компонента repository для створення нового відповідного запис в базі даних. Функція повертає на вихід об’єкт Chat, який містить всю інформацію про новостворений чат, включаючи його ідентифікатор, ім’я, інформацію про власника та інших користувачів.

Функція Chat відповідає за отримання всієї інформації про чат. А саме: ідентифікатор, назву, інформацію про власника та користувачів. Вона приймає на вхід токен та ідентифікатор чату. Наступним кроком відбувається перевірка токена. Якщо він не дійсний, повертається повідомлення про помилку. Інакше, за допомогою функції Chat компоненту repository з бази даних отримується інформація про чат. Але, на жаль, вона містить лише ідентифікатори власника та користувачів чату, в той час як нам потрібно повернути повну інформацію про них. Для цього, за допомогою функції UsersInfo, ініціюється запит до мікросервіса «Користувачі», який містить всю необхідну інформацію. Мікросервіс «Користувачі» приймає на вхід список ідентифікаторів програмістів та повертає список структур, які надають повнішу інформацію про них. Ця інформація додається до вже існуючої та повертається з функції.

Функція UpdateChat відповідає за оновлення чату. Вона приймає на вхід структуру, яка містить інформацію про токен користувача, який відправив запит, ідентифікатор чату, його назву та список ідентифікаторів його користувачів. Далі відбувається перевірка токену. Якщо він не дійсний повертається повідомлення про помилку. Якщо ж він дійсний, за допомогою

функції ChatOwner компонента repository отримується ідентифікатор власника чату. Наступним кроком порівнюється ідентифікатор отриманий з бази даних з ідентифікатором, який був отриманий з токену. Якщо вони співпадають, користувач, який ініціював зміну, дійсно є власником чату, тому операцію можна провести. Зміни в базі даних здійснюються за допомогою функції UpdateChat модуля repository.

Функція DeleteChat приймає на вхід токен та ідентифікатор чату, перевіряє, чи дійсно користувач, який ініціював запит є власником чату, і якщо це відповідає дійсності, видаляє чат з бази даних за допомогою функції DeleteChat компонента repository.

Функція UserChats відповідає за отримання інформації про всі чати користувача. Вона приймає на вхід токен та перевіряє його. Якщо він недійсний повертає повідомлення з описом помилки. Інакше за допомогою функції UserChats компонента repository отримує та повертає інформацію про всі чати, учасником яких є користувач-відправник запиту.

Програмний код описаних функцій наведено в додатку В.

3.4 Розробка модуля «Пости»

Модуль «Пости» відповідає за створення, оновлення, видалення та отримання інформації про пости користувачів, їх коментарі та повідомлення в чатах. Модуль також розбивається на декілька компонентів: transport, endpoint, service, repository та domain. Компоненти transport та endpoint відповідають за отримання даних з запиту користувача, service реалізує бізнес-логіку програми, repository відповідає за роботу з базою даних, в domain розміщено об’єкти модуля.

Service містить наступні функції для роботи з повідомленнями в чатах: ChatMessages, CreateChatMessage, UpdateChatMessage, DeleteChatMessage.

Функція ChatMessages відповідає за отримання інформації про повідомлення, які були опубліковані в певному чаті. Вона приймає на вхід токен та ідентифікатор чату, далі відбувається перевірка токену, якщо він не

дійсний, повертається помилка. Інакше, за допомогою функції ChatMessages компонента repository отримує інформацію про повідомлення, що знаходяться в чаті, ідентифікатор якого був переданий функції. Отримані дані містять інформацію про ідентифікатор чату, в якому створено повідомлення, ідентифікатор самого повідомлення, його текст, посилання на картинки чи файли. Дані про користувача, який створив повідомлення обмежуються лише його ідентифікатором. Далі викликається функція addUsersFullNames. Для того, щоб отримати всі необхідні дані про користувача вона використовує функцію UsersInfo модуля «Користувачі». UsersInfo приймає на вхід масив ідентифікаторів користувачів та повертає масив даних про них в тому самому порядку, в якому знаходились їх ідентифікатори. Останнім кроком функція addUsersFullNames поєднанує інформацію отриманої з компонента repository та модуля «Користувачі». Ці дані функція ChatMessages подає на вихід.

Функція CreateChatMessage створює нове повідомлення в чаті. Вона приймає на вхід структуру, яка містить інформацію про ідентифікатор чату, де створюється повідомлення, токен користувача, текст повідомлення, посилання на файли та картинки. Функція перевіряє токен, якщо він не дійсний, повертає помилку. Інакше за допомогою функцій Process опрацьовує отримані файли та картинки. Кожен з них зберігається у файловій системі та отримує унікальний ідентифікатор. Далі за допомогою функції CreateChatMessage компонента repository створюється відповідний запис в базі даних. Останнім кроком відбувається позначення повідомлення як прочитаного за допомогою функції UpdateReadMessages компонента repository.

Функція UpdateChatMessage оновлює повідомлення. Вона приймає на вхід структуру, яка містить такі поля: токен, ідентифікатор змінюваного повідомлення, новий текст, додані файли та картинки, списки видалених файлів та картинок. Наступним кроком за допомогою функції ChatMessage компонента repository отримуються збережені дані про повідомлення. Далі відбувається порівняння ідентифікатора користувача, який ініціював запит з іденифікаторм автора поста, якщо вони не співпадають, повертається помилка. Обробка змін

відбувається з використанням функції Update, яка приймає на вхід інформацію про всі зміни, які відбулися з повідомленням, оновлює текст повідомлення та його посилання, зберігає нові файли та видаляє непотрібні. Запис оновленої інформаці в базі даних відбувається за допомогою функції UpdateChatMessage компонента repository.

Функція DeleteChatMessage відповідає за видалення повідомлення з чату. Вона приймає на вхід токен користувача та ідентифікатор повідомлення. Відбувається перевірка токену на дійсність, якщо вона була не успішною, повертається повідомлення про помилку. Інакше функція перевіряє чи дійсно користувач, який ініціював запит є автором повідомлення. Це відбувається за допомогою функції ChatMessage компонента repository. Вона повертає інформацію про повідомлення, ідентифікатор автора повідомлення порівнюється з ідентифікатором користувача, який ініціював запит. Якщо вони не збігаються, користувач не має права видаляти це повідомлення, тому повертається повідомлення про помилку. Якщо ж ідентифікатори збігаються, повідомлення видаляється з бази даних за допомогою функції DeleteChatMessage компонента repository.

Для роботи з постами користувачів Service містить наступні функції: CreatePost, GetPosts, UpdatePost, DeletePost.

Функція CreatePost створює новий пост. Вона приймає на вхід структуру даних, яка містить токен користувача, текст поста, додані файли та картинки. Спершу відбувається перевірка токену, якщо він не дійсний повертається повідомлення про помилку. Інакше за допомогою функції Process обробляються додані файли та картинки. Вони зберігаються у файловій системі та отримують унікальні ідентифікатори. Після цього, інформація про новий пост з використанням функції CreatePost компонента repository зберігається в базі даних.

Функція GetPosts відповідає за отримання інформації про всі пости користувача. Вона приймає на вхід токен автора запиту та ідентифікатор користувача, пости якого необхідно отримати. Далі вона перевіряє, чи дійсний

отриманий токен, якщо ні, повертається повідомлення про помилку. Якщо ж токен дійсний, за допомогою функції UserPosts компонента repository вона отримує необхідну інформацію та повертає її.

Функція UpdatePost оновлює наявний пост. Вона отримує на вхід структуру даних, яка містить інформацію про токен, новий текст повідомлення, додані файли та картинки, а також список файлів та картинок, які були видалені. Далі відбувається перевірка токену, якщо він виявився недійсним, повертається повідомлення з інформацією про помилку. Якщо ж він дійсний, з бази даних отримується інформація про пост, зіставляється ідентифікатор користувача, який ініціював запит з ідентифікатором автора поста. Якщо вони не співпадають, користувач не має права вносити зміни в пост, тому повертається повідомлення з інформацією про помилку. Якщо ж ідентифікатори співпадають, за допомогою функції Update відбувається оновлення інформації про пост. Вона приймає на вхід новий текст поста, картинки, файли та список видалених картинок та файлів. Після цього функція змінює текст поста, обробляє нові файли і додає посилання на них до поста. Крім того видаляються файли, які більше не потрібні. Останнім кроком вносяться зміни в базу даних за допомогою функції UpdatePost компонента repository.

Функція DeletePost відповідає за видалення поста. Вона приймає на вхід токен користувача та ідентифікатор поста, який необхідно видалити. Здійснюється перевірка токену на дійсність, якщо вона не успішна повертається повідомлення з інформацією про помилку. Якщо ж токен дійсний, за допомогою функції GetPost компонента repository отримується інформація про пост. Після цього зіставляються ідентифікатор користувача, який ініціював запит та ідентифікатор автора поста, якщо вони не співпадають, автор запиту не має права видаляти пост, тому повертається помилка. Якщо операція порівняння успішно пройшла, пост видаляється з бази даних за допомогою функції DeletePost компонента repository.

Для роботи з коментарями Service містить наступні функції PostComments, CreatePostComment, DeletePostComment, UpdateComment, CommentComments, CreateCommentComment, DeleteCommentComment.

Функція PostComment відповідає за отримання всіх коментарів, які були написані на пост користувача. Вона приймає на вхід токен та ідентифікатор поста. Далі відбувається перевірка токену на дійсність, якщо вона пройшла не успішно, повертається повідомлення з помилкою. Якщо ж токен дійсний, наступним кроком, за допомогою функції PostComments компонента repository отримується інформація про коментарі заданого поста. На жаль, інформація про коментарі міститиме лише ідентифікатори тих, хто їх створив, хоча на вихід потрібно подати також ім’я та прізвище автора. Далі викликається функція addUsersFullNames, яка створює запит до компонента «Користувачі», який містить необхідну інформацію. Запит обробляє функція UsersInfo, яка приймає на вхід список ідентифікаторів користувачів та повертає список з інформацією про кожного користувача. Після цього, addUsersFullNames до кожного коментаря додає інформацію про його автора. На вихід PostComment повертає список з інформацією про кожен коментар на пост, ідентифікатор якого було передано у функцію.

Функція CreatePostComment створює новий коментар на пост. Вона приймає на вхід структуру, яка містить ідентифікатор поста, токен користувача, текст коментаря, додані катринки та файли. Відбувається перевірка токену, якщо він не дійсний, повертається повідомлення з описом помилки. Якщо ж він дійсний, файли та картинки оброблюються за допомогою функції Process. Кожен з них отримує унікальний ідентифікатор та зберігається у файловій системі. Функція Process повертає на вихід список ідентифікаторів оброблених файлів. Після цього, нова інформація вноситься до бази даних за допомогою функції CreatePostComment компонента repository.

Функція DeletePostComment видаляє інформацію про коментар, який був написаний під певним постом. Вона приймає на вхід токен користувача, ідентифікатор поста, під яким було написано коментар та ідентифікатор самого

коментаря. Першим кроком перевіряється дійсність токену. Якщо перевірка пройшла не успішно, повертається повідомлення про помилку. В іншому випадку отримується інформація про коментар за допомогою функції GetComment компонента repository. Після цього ідентифікатор автора коментаря порівнюється з ідентифікатором користувача, який ініціював запит. Якщо вони не співпадають, користувач не має права видаляти пост, тому повертається повідомлення про помилку. Якщо ж перевірка пройшла успішно, відбувається видалення коментаря за допомогою функції DeletePostComment компонента repository.

Функція UpdateComment оновлює коментар. Вона приймає на вхід структуру, яка містить токен користувача, ідентифікатор коментаря, новий текст, додані картинки та файли, а також список видалених картинок та файлів. Відбувається перевірка токену, якщо він не дійсний, повертається повідомлення про помилку. В іншому випадку отримується інформація про коментар за допомогою функції GetComment компонента repository. Після цього перевіряється, чи дійсно автор запиту є автором повідомлення, якщо це не так, він не має права змінювати пост, тому повертається повідомлення про помилку. Інакше відбувається оновлення даних коментаря. Нові файли та картинки отримують унікальні ідентифікатори та зберігаються у файловій системі. Елементи, які більше потрібні видаляються. Зміни, що були внесені в коментар зберігаються в базі даних за допомогою функції UpdateComment компонента repository. На вихід функція подає оновлену інформацію про коментар.

Функція CommentComments отримує інформацію про коментарі, що були написані у відповідь на інші коментарі. Вона приймає на вхід токен та ідентифікатор коментаря. Відбувається перевірка токену, після чого отримується потрібна інформація з бази даних за допомогою функції CommentComments компонента repository. Далі за допомогою функції addUsersFullNames з компонента «Користувачі» отримується додаткова інформація про авторів коментарів, яка додається до вже отриманої з

компонента repository. Функція повертає список коментарів до коментаря, ідентифікатор якого було отримано на вході.

Функція CreateCommentComment створює коментар на коментар користувача. Вона приймає на вхід структуру, яка містить ідентифікатор коментованого об’єкта, токен користувача, текст, додані картинки та файли. Токен перевіряється, якщо він не дійсний, повертається повідомлення про помилку. Якщо ж він дійсний, додані файли отримують унікальні ідентифікатори та зберігаються в файловій системі. Коментар створюється в базі даних за допомогою функції CreateCommentComment компонента repository. На вихід передається список новостворений коментар.

Функція DeleteCommentComment видаляє коментар, який був написаний на інший коментар. Вона приймає на вхід токен, ідентифікатор коментованого об’єкта та ідентифікатор коментаря, який необхідно видалити. Відбувається перевірка токену, якщо він не дійсний, повертається повідомлення про помилку. Якщо ж він дійсний, отримується інформація про коментар, якщо ідентифікатор його автора не співпадає з ідентифікатором користувача, який ініціював запит, повертається повідомлення про помилку, оскільки він не має права видаляти цей коментар. Якщо ідентифікатори співпадають, інформація про коментар видаляється за допомогою функції DeleteCommentComment компонента repository.

3.5 Висновок

Третій розділ присвячений вибору інструментів розробки та опису створених модулів «Користувачі», «Чати» та «Пости» онлайн застосунку для комунікації програмістів.

Для реалізації коду серверної частини було обрано мову програмування Golang. Вона є сучасним інструментом, який забезпечує належну швидкість та гнучкість розробки. Крім того, застосунки, розроблені цією мовою характеризуються високими показниками швидкості та продуктивності.

Модуль «Користувачі» є важливим компонентом онлайн платформи для комунікації програмістів. Він обробляє запити пов’язані з інформацією про користувача та його підписками. Сам модуль також ділиться на компоненти, кожен з яких відповідає за виконання певної роботи: отримання даних з запиту, виконання бізнес-логіки, робота з базою даних. Модуль «Чати» відповідає за додавання інформації про чат, її оновлення, видалення та отримання. Цей модуль також розділений на менші компоненти. Модуль «Пости» відповідає за створення, оновлення, видалення та повернення інформації про пости користувачів, їх коментарі та повідомлення в чатах.

4 ТЕСТУВАННЯ ОНЛАЙН ПЛАТФОРМИ ДЛЯ КОМУНІКАЦІЇ ПРОГРАМІСТІВ

4.1 Тестування застосунку

Тестування є невід'ємним етапом розробки програмного забезпечення. Воно необхідне тому, що код в переважній більшості пишеться розробниками, а вони можуть допускати помилки. Саме тому перед запуском продукту необхідно його ретельно протестувати [25]. Відповідно, метою тестування є виявлення помилок, допущених розробниками.

Спочатку необхідно перевірити, чи можливо зареєструватися на онлайн платформі. Для того, щоб це зробити необхідно заповнити форму наведену на рисунку 4.1. Вона містить такі поля як ім'я та прізвище користувача, логін та пароль. Також наявне поле для підтвердження пароля.

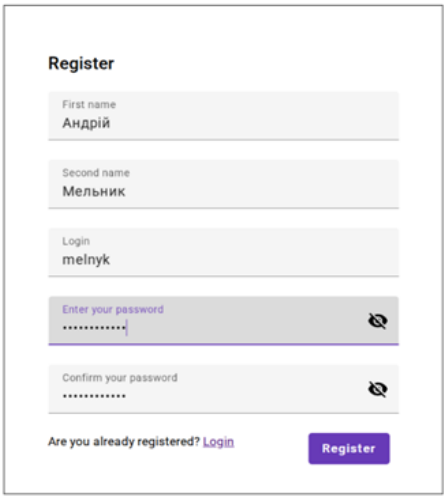

Рисунок 4.1 – Форма реєстрації нового користувача

Якщо логін ще не використовується, а пароль не є меншим 8 символів, користувач успішно реєструється в системі. Інакше відкривається вікно з описом помилки.

Наступним кроком варто перевірити можливість зміни інформації про себе. Для цього треба перейти на вкладку «Profile», натиснути олівець у правому верхньому куті, внести зміни в полях з інформацією про себе та натиснути кнопку «Update» (рисунок 4.2). Оновлення відбувається успішно і тепер профіль користувача має деякі дані щодо його біографії.

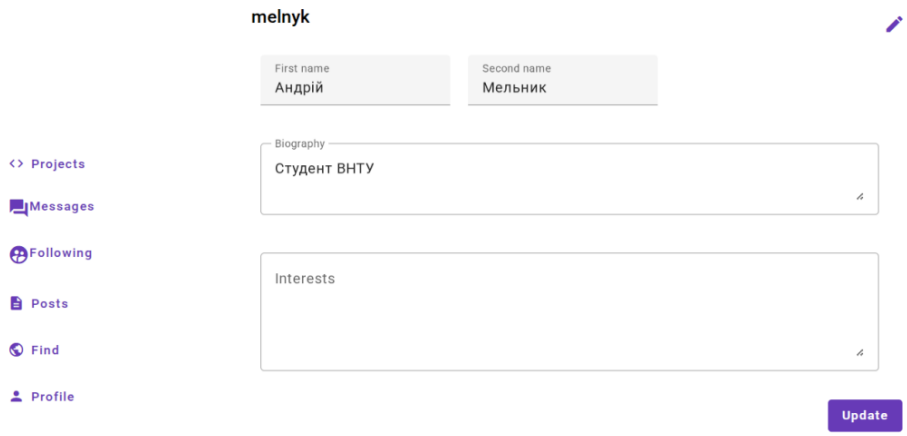

Рисунок 4.2 – Оновлення особистої інформації

Далі необхідно перевірити справність функції знаходження користувачів. Для цього треба перейти на вкладку «Find» та ввести в поле, що з’явилося текст, за яким буде здійснено пошук. На рисунку 4.3 продемонстровано пошук за початком абревіатури Вінницького національного технічного університету. Система реагує правильно, оскільки демонструє користувачів, які мають подібний текст в розповіді про себе, в інтересах або в логіні. Пошук одразу в кількох полях робить процес знаходження людей набагато простішим, що поділяють інтереси користувача. Це дозволяє швидко знайти співрозмовників на найрізноманітніші теми, включаючи такі, що цікавлять лише обмежене коло фахівців.

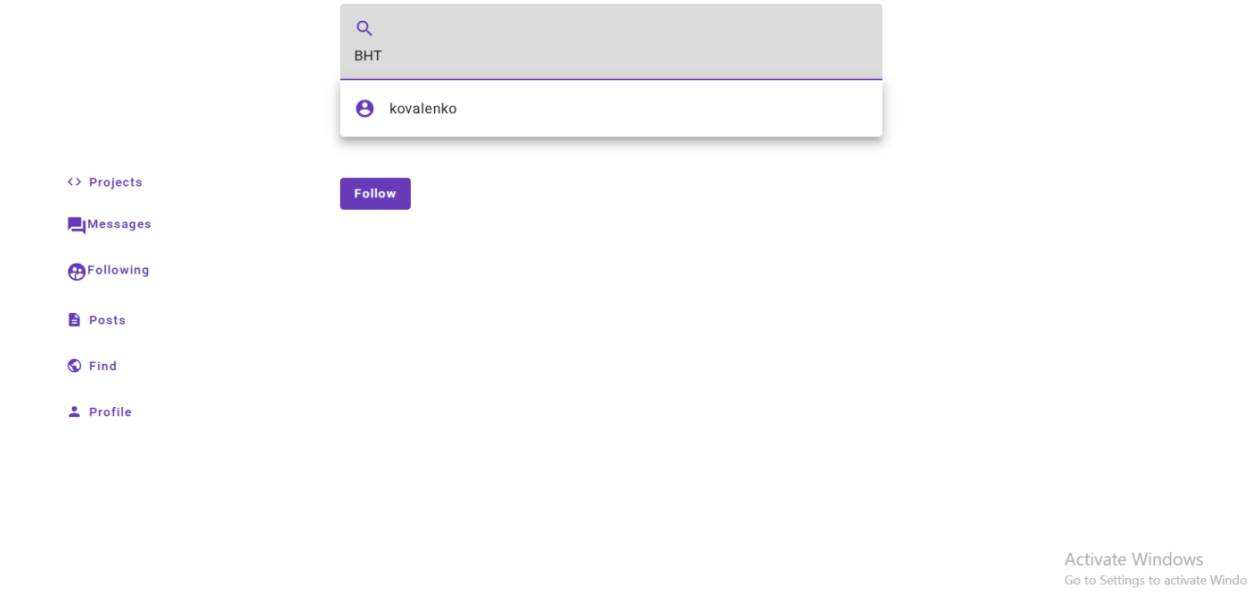

Рисунок 4.3 – Пошук користувачів

При натисканні на запропонованому користувачі, відкривається інформація про нього (рисунок 4.4). Можна пересвідчитися, що дана особа дійсно пов’язана з закладом, абревіатуру якого було введено.

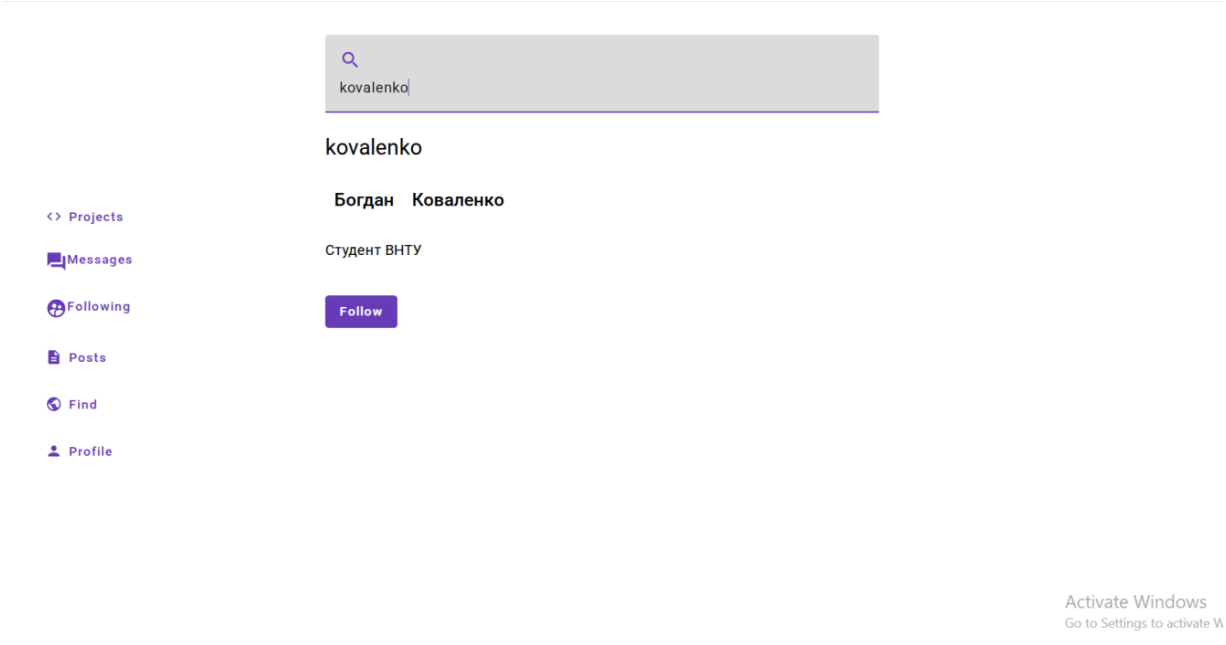

Рисунок 4.4 – Перегляд інформації про користувача

Для перевірки, чи може програміст підписатися на іншого користувача треба натиснути кнопку «Follow». Щоб пересвідчитися у результаті варто зайти на вкладку «Following» та натиснути на полі вводу. Програма виводить список з користувачами на яких є підписка (рисунок 4.5). Серед них є і нова підписка на Богдана Коваленко, отже функція підписки, успішно спрацювала.

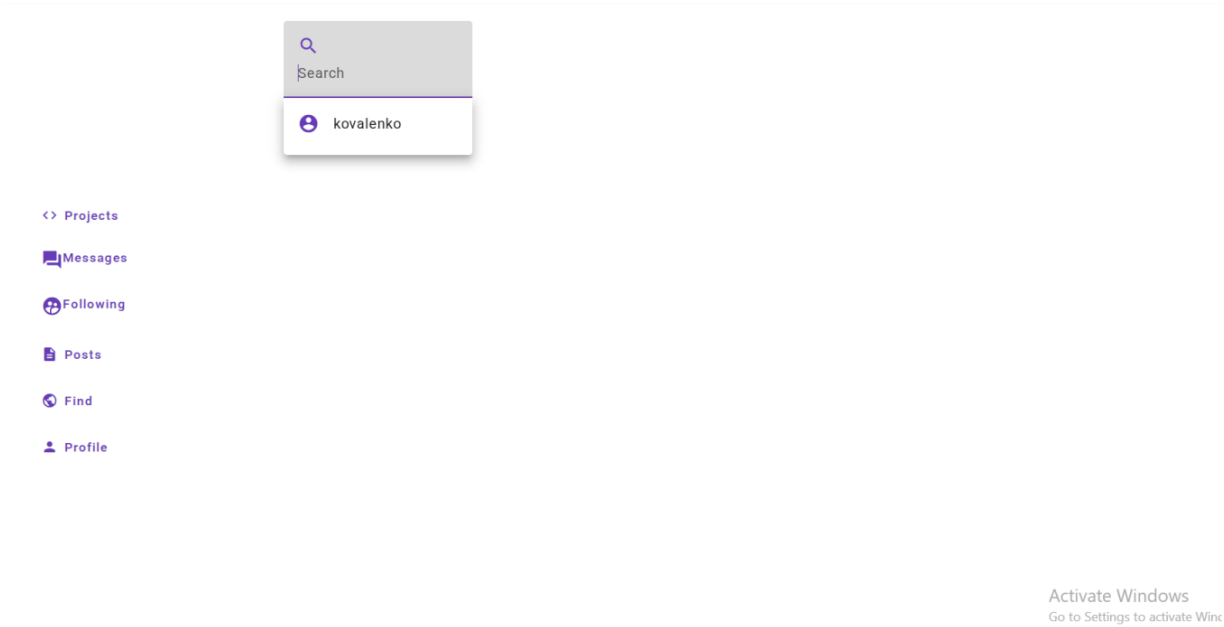

Рисунок 4.5 – Список підписок користувача

Важливим функціоналом платформи є створення нового поста. Щоб створити новий пост потрібно зайти на вкладку «Posts», натиснути кнопку «Write Post» та заповнити форму, що з’явилася. Інформація, яку можна додати в цю форму включає текст, код, картинки та інші файли. Приклад заповнення форми наведено на рисунку 4.6. Стрілочкою «-->» позначається початок коду. Програма аналізує текст поста та забезпечує підсвітку коду для того, щоб користувач, який буде з ним ознайомлюватися краще його зрозумів. Блок коду також можна закрити подібною стрілочкою «-->». Текст, що буде йти за нею більше не виділятиметься та буде схожий на текст перед стрілочкою.

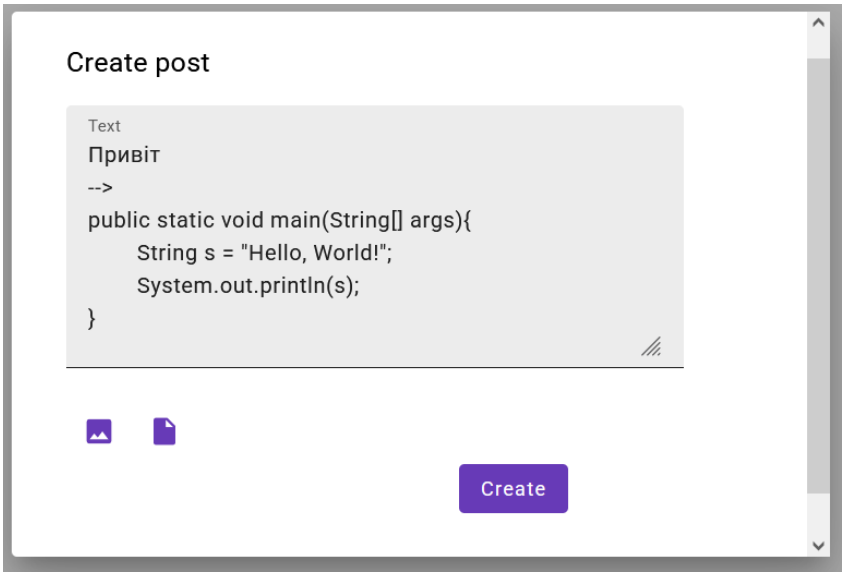

Рисунок 4.6 – Форма заповнення поста

Оскільки пост успішно додано, а текст, який містить код виділено відповідними кольорами (рисунок 4.7), зроблено висновок, що функція додавання посту працює успішно.

Рисунок 4.7 – Результат додавання нового поста

Цей пост створений користувачем Богданом Коваленком. Щоб протестувати, чи має Андрій Мельник можливість його переглянути необхідно

зайти на вкладку «Following» та серед списку запропонованих обрати логін Богдана. В результаті відображаються всі його пости (рисунок 4.8). Оскільки, система відображає нещодавно доданий пост, вона працює правильно.

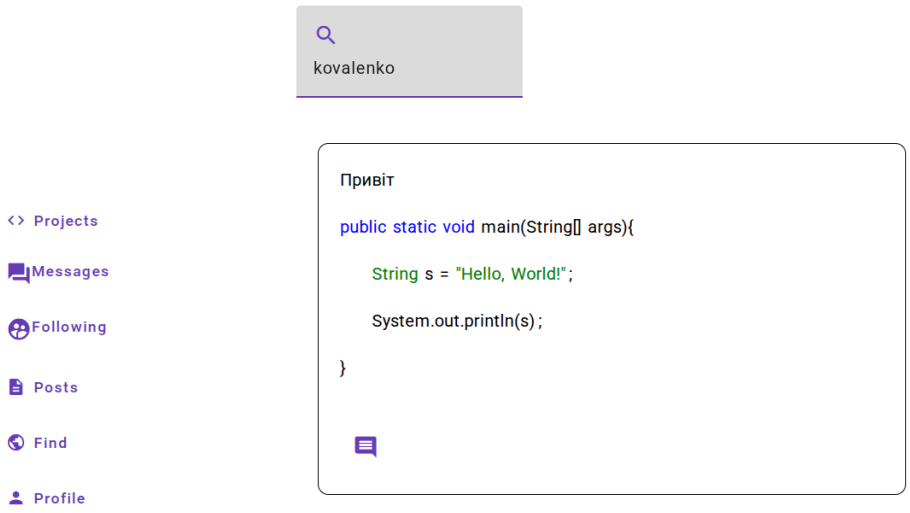

Рисунок 4.8 – Перегляд постів іншого користувача

Для перевірки функціоналу оновлення поста необхідно натиснути кнопку «Оновити». У вікні, що відкрилося (рисунок 4.9) потрібно внести зміни.

Рисунок 4.9 – Вікно оновлення поста

Після натискання кнопки «Update» програма вносить зміни до опублікованого посту (рисунок 4.10).

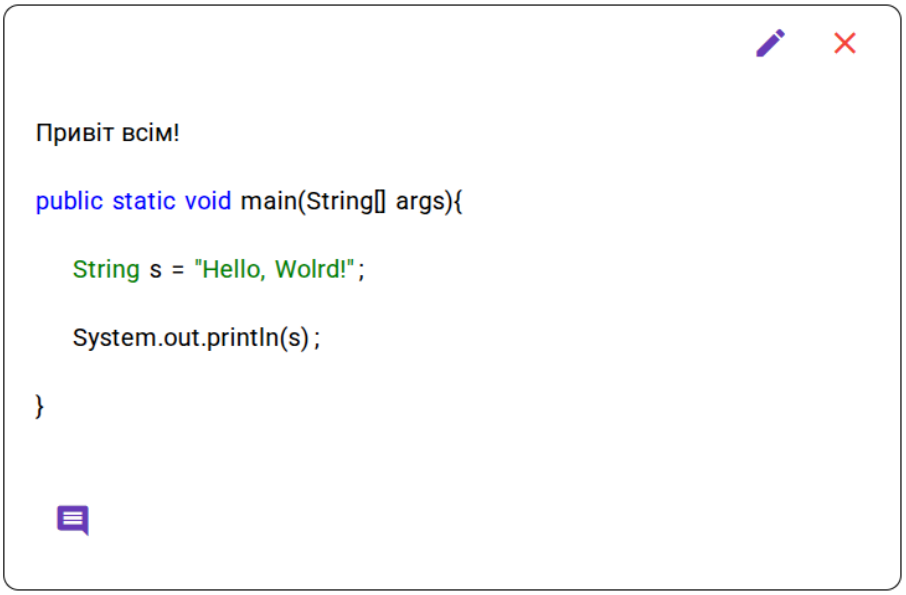

Рисунок 4.10 – Оновлений пост

Отже, даний функціонал оновлення користувачем існуючого поста працює коректно.

Щоб перевірити можливість видалення наявного поста потрібно натиснути кнопку «X» у правому верхньому куті посту. Після цього відповідний пост має бути видалений з системи (рисунок 4.11).

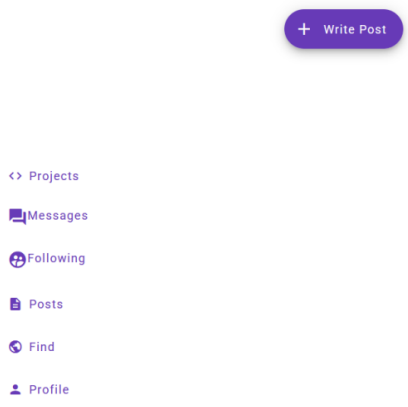

Рисунок 4.11 – Результат видалення поста

Оскільки видалений пост більше не з’являється у вкладці «Posts», можна зробити висновок, що функція видалення поста працює коректно. Так як користувач більше нічого не публікував, вкладка «Posts» не відображає жодного поста.

Далі необхідно протестувати можливість додавати коментарі до постів. Для цього треба натиснути кнопку «Коментарі» під постом та у вікні, що з’явилося натиснути «Write comment», заповнити форму та натиснути «Create». Після цього серед коментарів до поста має з’явиться новий, щойно створений. Результат виконання цих дій наведено на рисунку 4.12. Програма додає коментар, створений користувачем, отже система працює правильно. Онлайн платформа також вказує ім’я користувача, який написав коментар. Оскільки повідомлення написано користувачем, який зараз його переглядає, в полі автора вказано не повне ім’я, а лише написано «You».

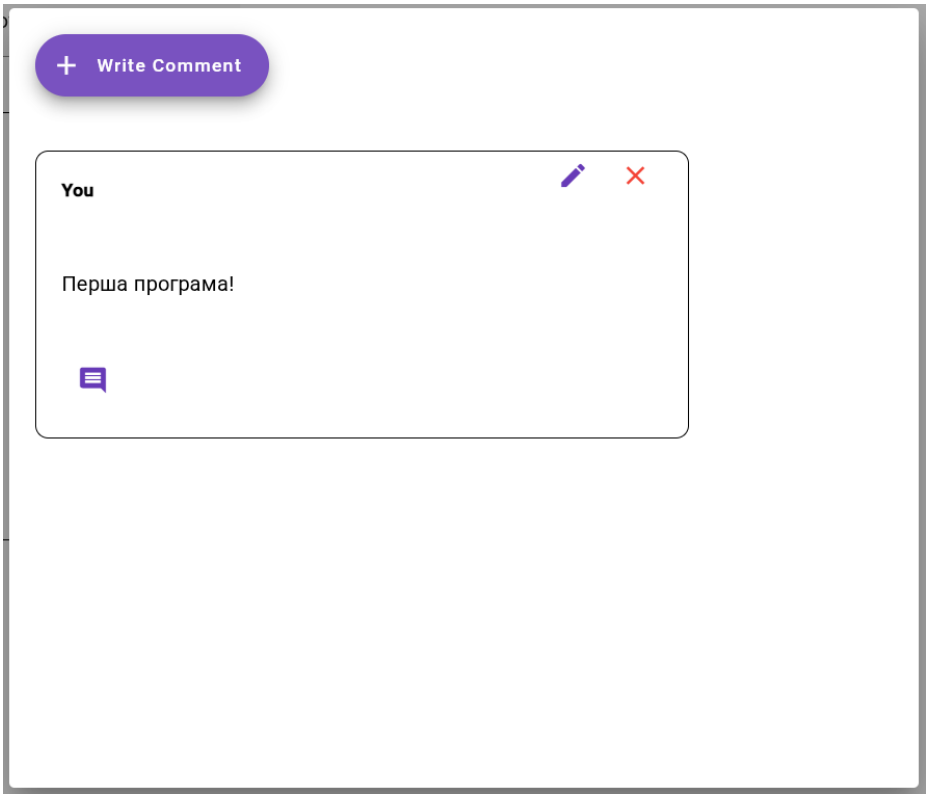

Рисунок 4.12 – Коментар до посту

Коментар також можна оновити натиснувши кнопку у вигляді олівця. Після цього з’явиться вікно з інформацією про коментар, яку можна редагувати (рисунок 4.13).

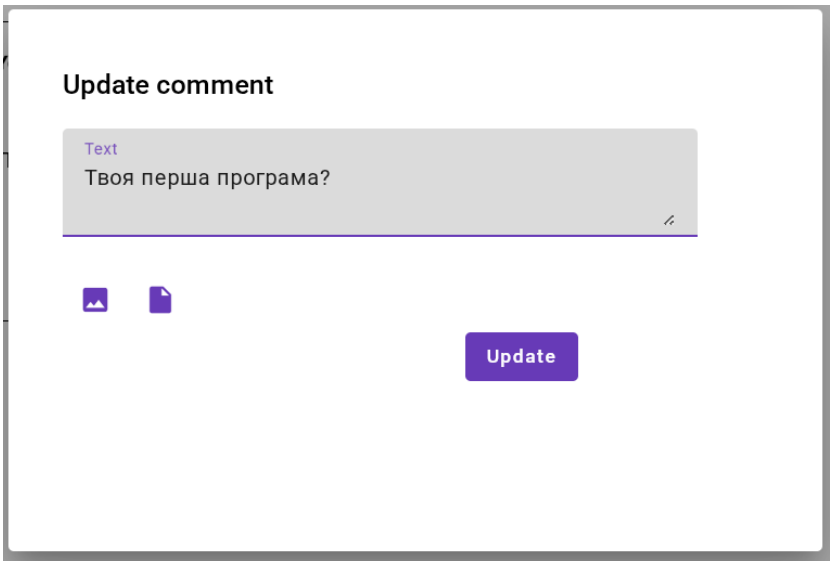

Рисунок 4.13 – Вікно оновлення коментаря

Після натискання кнопки «Update» інформація про коментар оновлюється (рисунок 4.14). Отже, функція оновлення коментарів працює правильно.

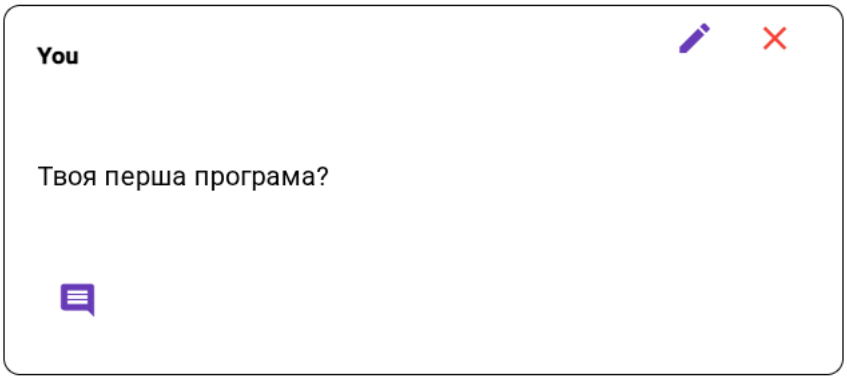

Рисунок 4.14 – Оновлений коментар

Для видалення коментаря потрібно натиснути кнопку «X» у верхньому правому куті. Після цього відповідний коментар буде видалено з системи.

Оскільки коментар більше з’являється (рисунок 4.15), програма успішно виконала функцію видалення.

Рисунок 4.15 – Результат видалення коментаря

Для перевірки на можливість комунікації в чатах треба зайти на вкладку «Messages». Натиснути кнопку «Add» та у вікні, що з’явилося ввести інформацію про чат: його ім’я та співрозмовників (рисунок 4.16).

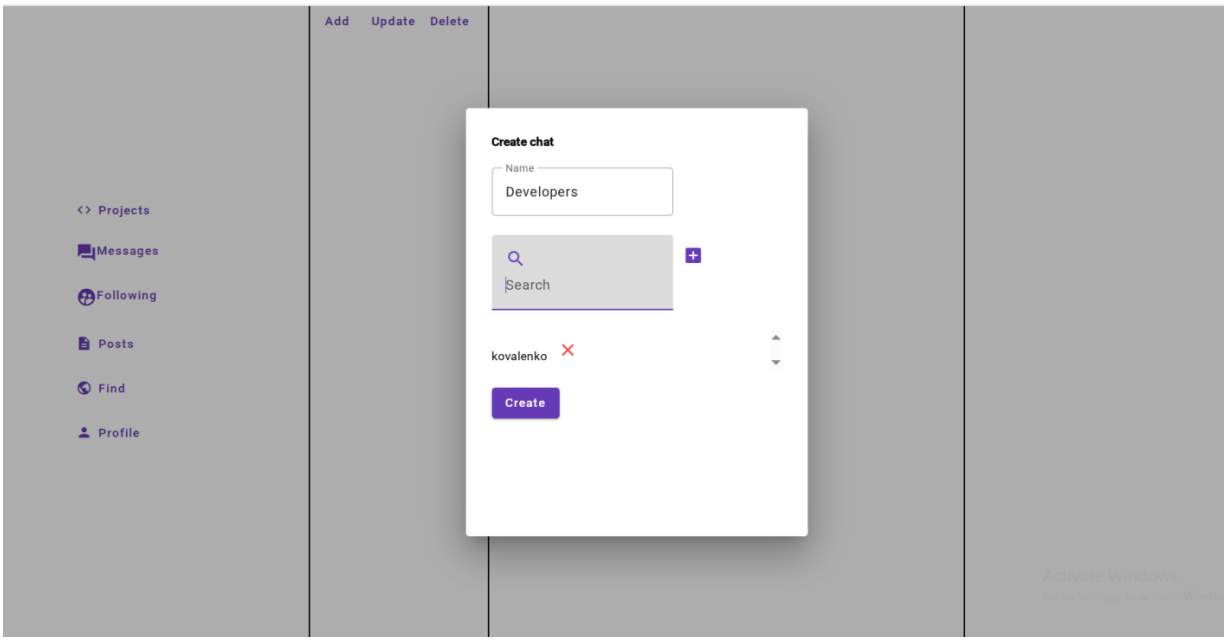

Рисунок 4.16 – Створення нового чату

Оскільки чат успішно додано, система спрацювала правильно. Далі необхідно вибрати чат та написати туди повідомлення. Це можна зробити натиснувши кнопку «Write Message» та заповнивши форму, що з’явилася (рисунок 4.17).

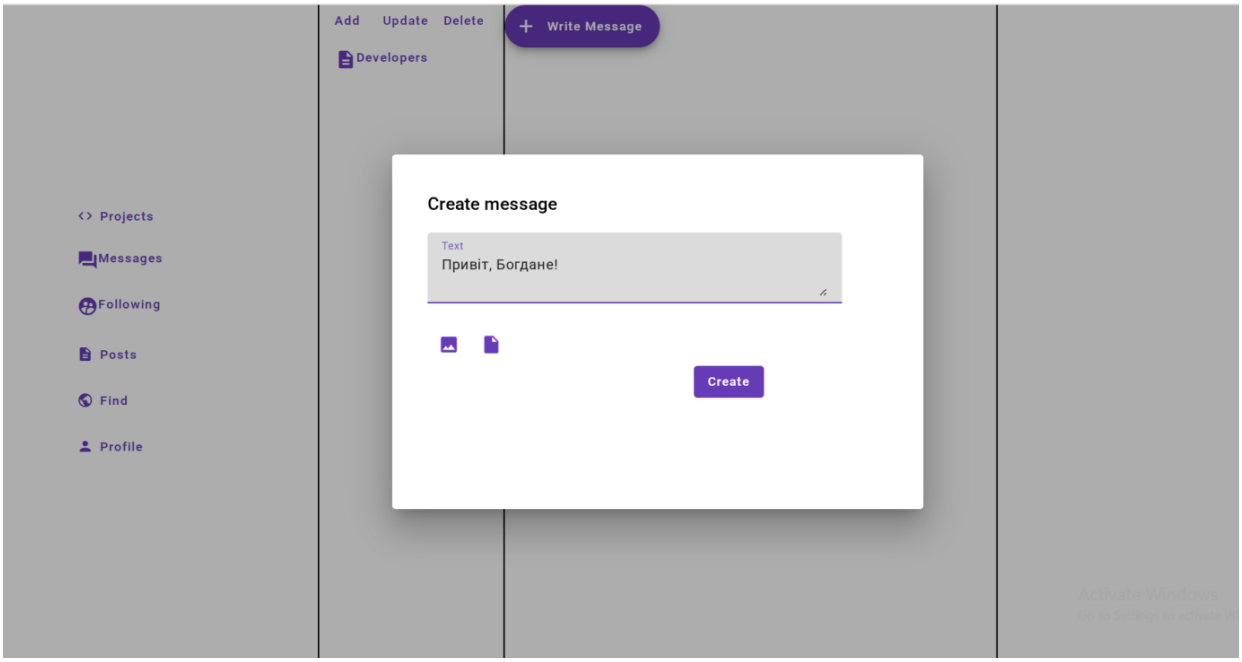

Рисунок 4.17 – Створення нового повідомлення

При успішному виконанні повідомлення додається в чат (рисунок 4.18).

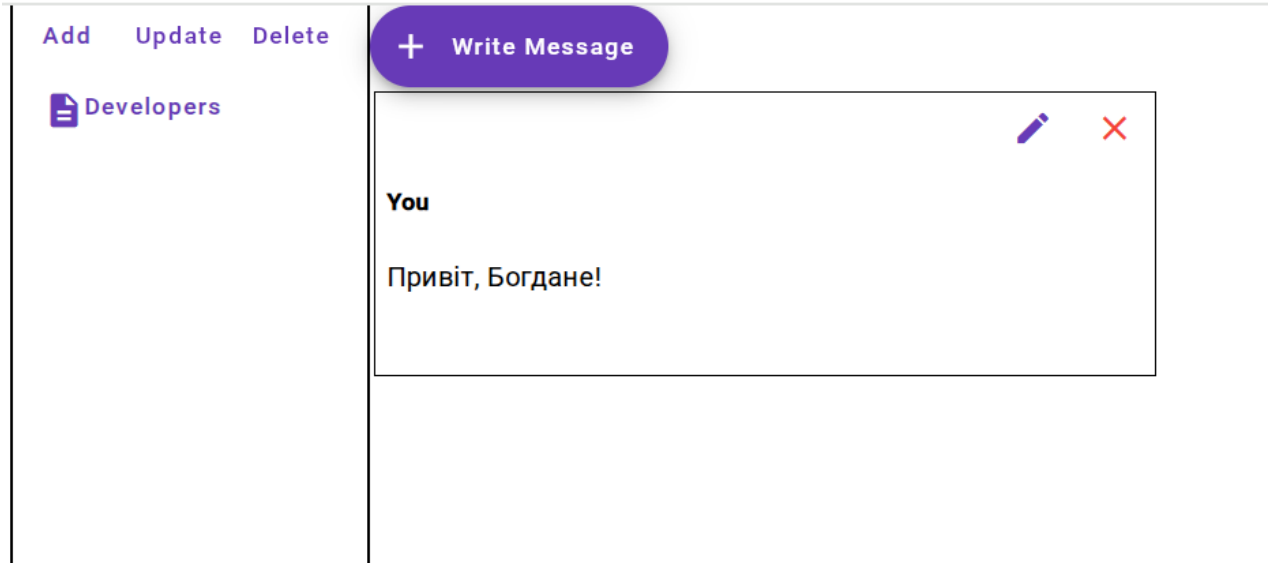

Рисунок 4.18 – Результат успішного виконання

Повідомлення можна оновити. Для цього потрібно натиснути кнопку, що нагадує олівець у верхньому правому кутку повідомлення. У вікні, що з’явилося (рисунок 4.19) внести потрібні зміни.

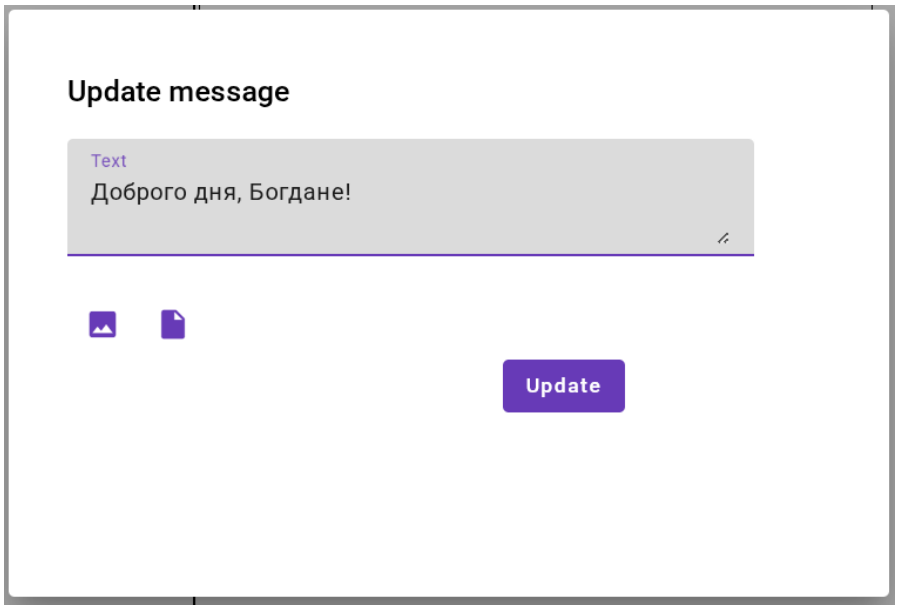

Рисунок 4.19 – Вікно оновлення повідомлення

Після натискання кнопки «Update» внесені зміни зберігаються у системі (рисунок 4.20). Отже, функціонал зміни повідомлення працює коректно.

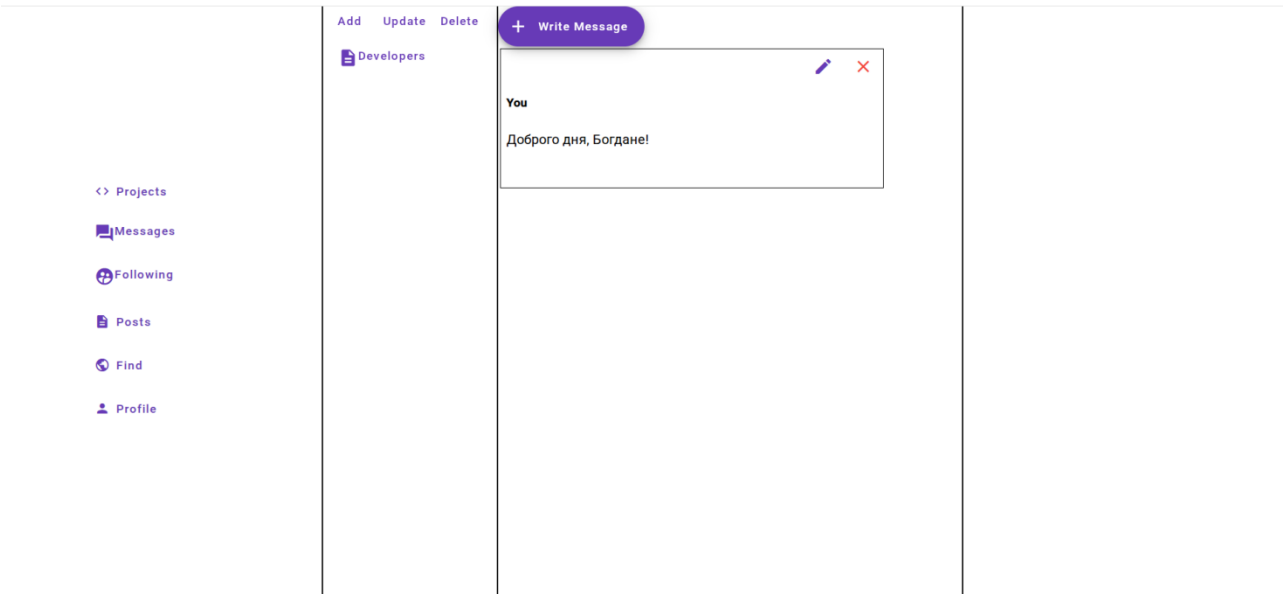

Рисунок 4.20 – Результат оновлення повідомлення

Для видалення повідомлення потрібно натиснути кнопку «X», що знаходиться поряд з кнопкою оновлення. Після цього відповідне повідомлення видаляється з чату (рисунок 4.21).

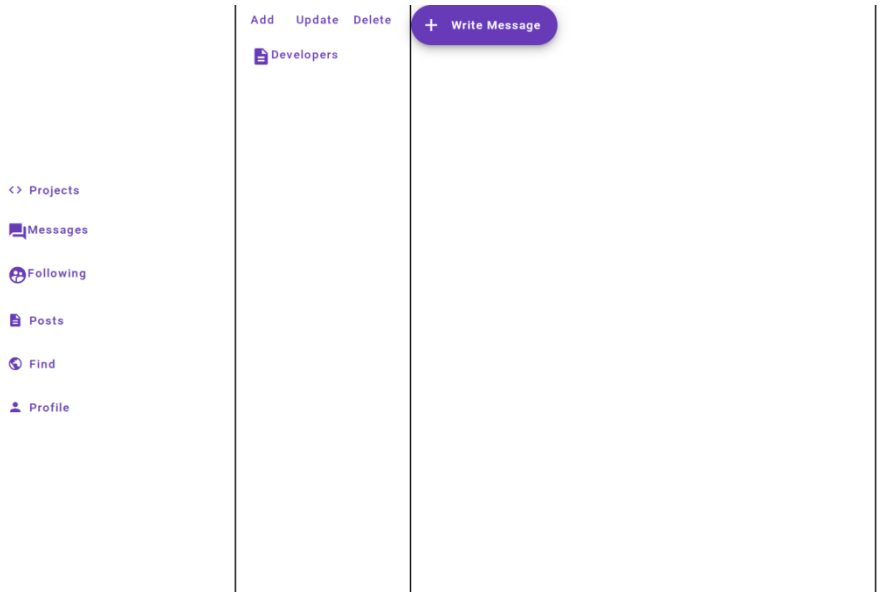

Рисунок 4.21 – Результат видалення повідомлення

Щоб оновити створений чат потрібно вибрати його та натиснути кнопку «Update», яка знаходиться над переліком чатів. З’явиться вікно, яке міститиме інформацію про чат (рисунок 4.22).

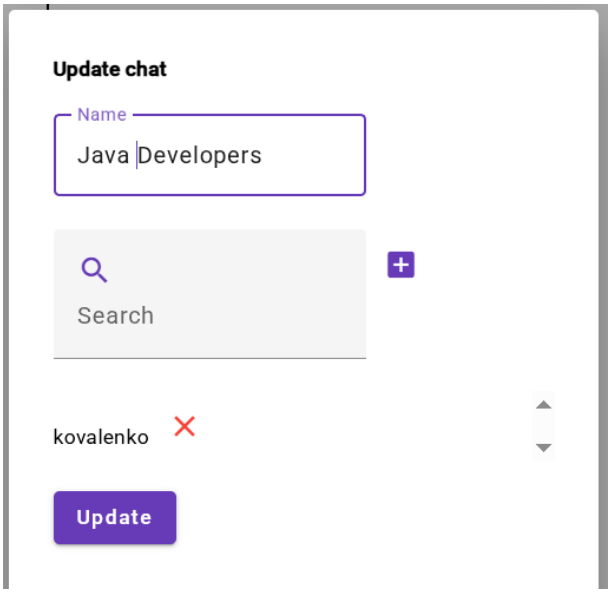

Рисунок 4.22 – Вікно оновлення чату

Після внесення змін, необхідно натиснути кнопку «Update». Оскільки програма зберігає всі зміни (рисунок 4.23), зроблено висновок, що функція оновлення чату працює успішно.

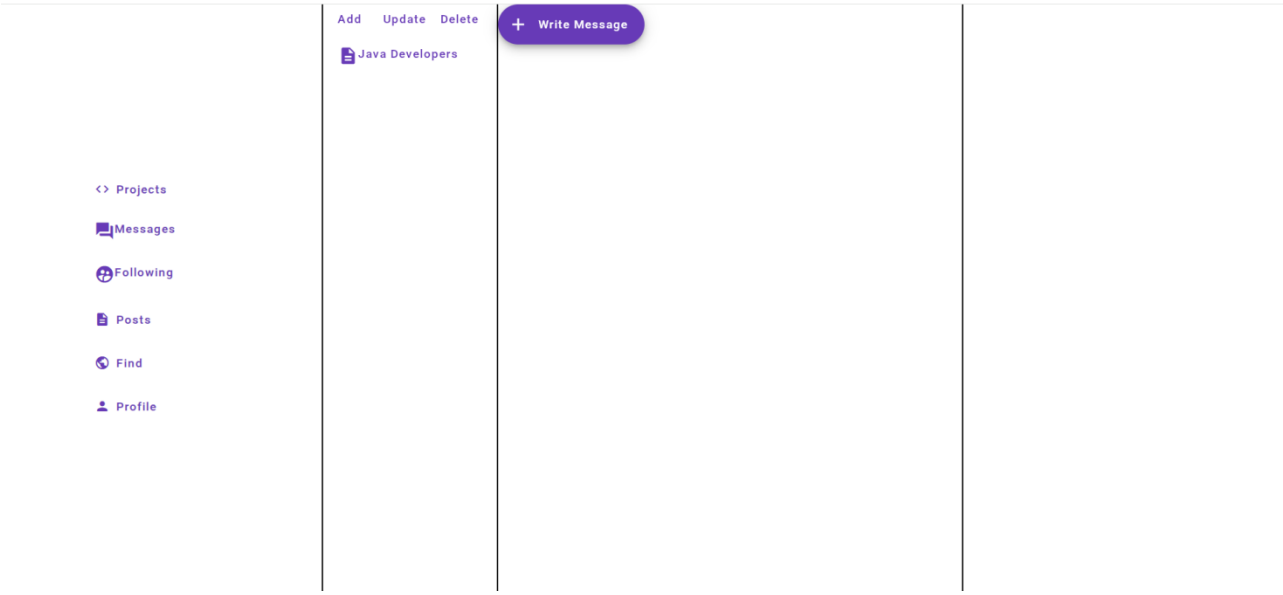

Рисунок 4.23 – Результат оновлення чату

Користувач також може видалити чат. Для цього потрібно вибрати його та натиснути кнопку «Delete», що знаходиться над списком чатів. Програма видаляє чат (рисунок 4.24), отже функція видалення чату працює правильно.

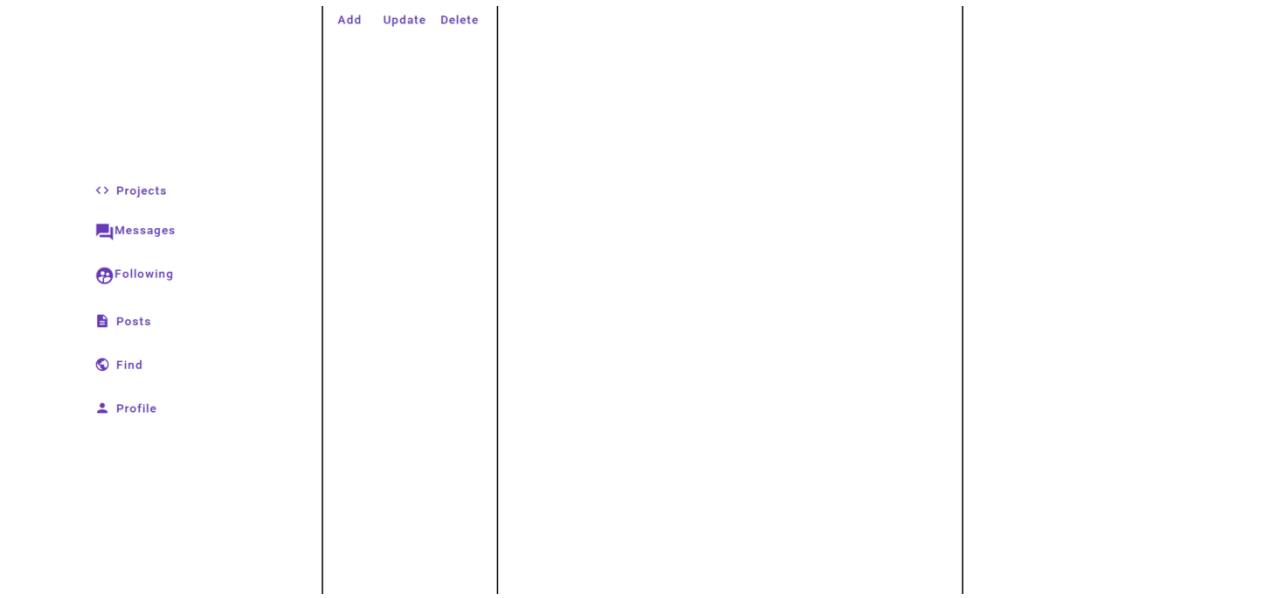

Рисунок 4.24 – Результат видалення чату

Вже зареєстровані користувачі можуть авторизуватися ввіши пароль та логін (рисунок 4.25).

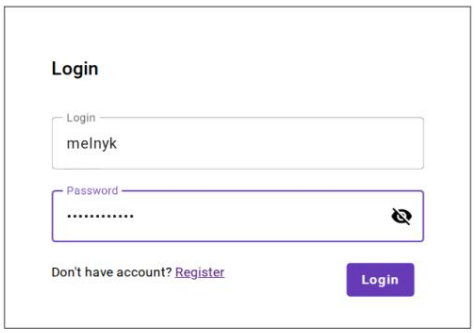

Рисунок 4.25 – Авторизація користувача

Після введення правильного логіну та пароля відкривається панель з закладками, тому можна зробити висновок, що функція авторизації працює справно.

Для перевірки можливості додавання нового проєкту необхідно перейти на вкладку «Projects», натиснути кнопку «Add Project» та заповнити форму, яка включає такі поля, як логін автора, назва проєкту, його опис та стек технологій, які на ньому використовуються. Така детальна інформація необхідна, щоб користувач зрозумів, чи підходить для нього цей проєкт. Оскільки після натискання кнопки «Add» інформація про новий проєкт додається до системи, функція додавання нових проєктів спрацювала успішно. Далі можна буде знайти новостворений проєкт серед низки інших. На рисунку 4.26 наведено вигляд списку проєктів, які наявні у системі.

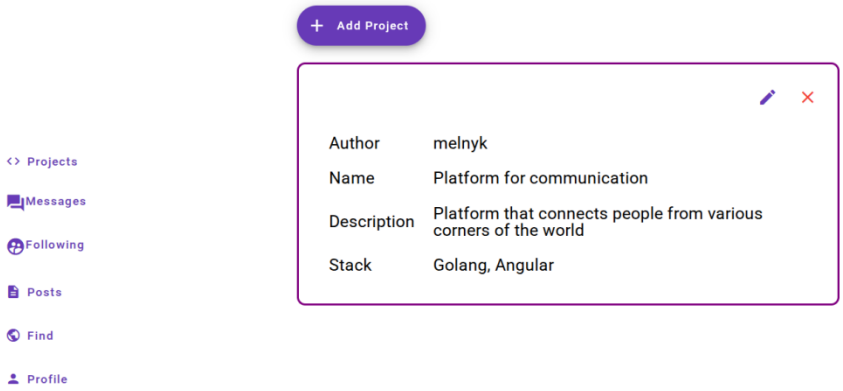

Рисунок 4.26 – Список наявних проєктів

Користувач може змінити опублікований проєкт. Для цього потрібно натиснути кнопку у вигляді олівця у верхньому правому куті інформації про проект. У вікні, що з’явилося (рисунок 4.27) необхідно внести потрібні зміни.

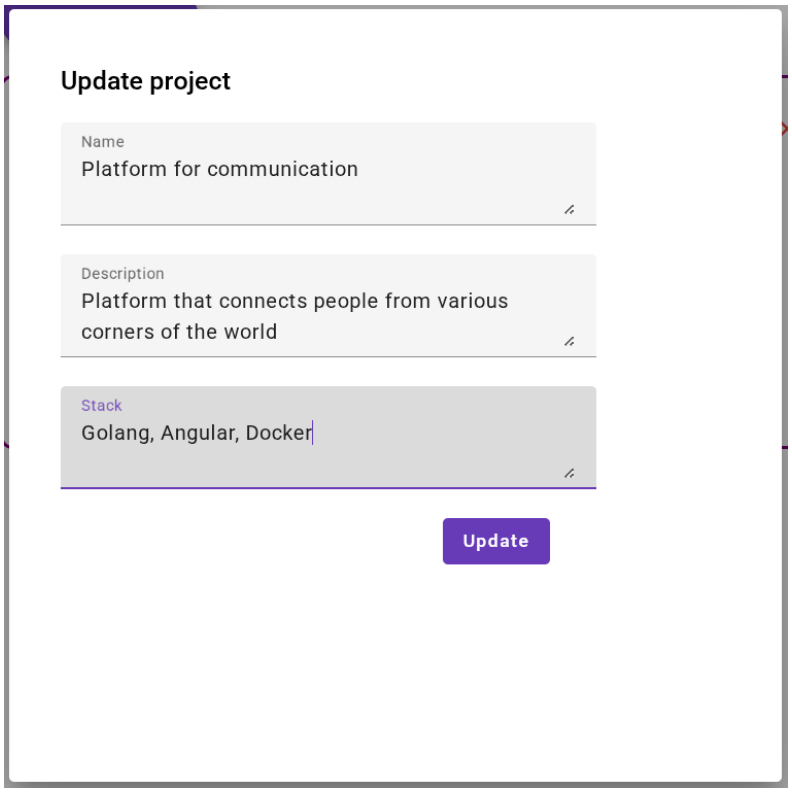

Рисунок 4.27 – Вікно зміни інформації про проект

Після цього потрібно натиснути кнопку «Update». Нова інформація відображається у вкладці «Projects» (рисунок 4.28), тому зроблено висновок, що операція зі зміни інформації про проект пройшла успішно.

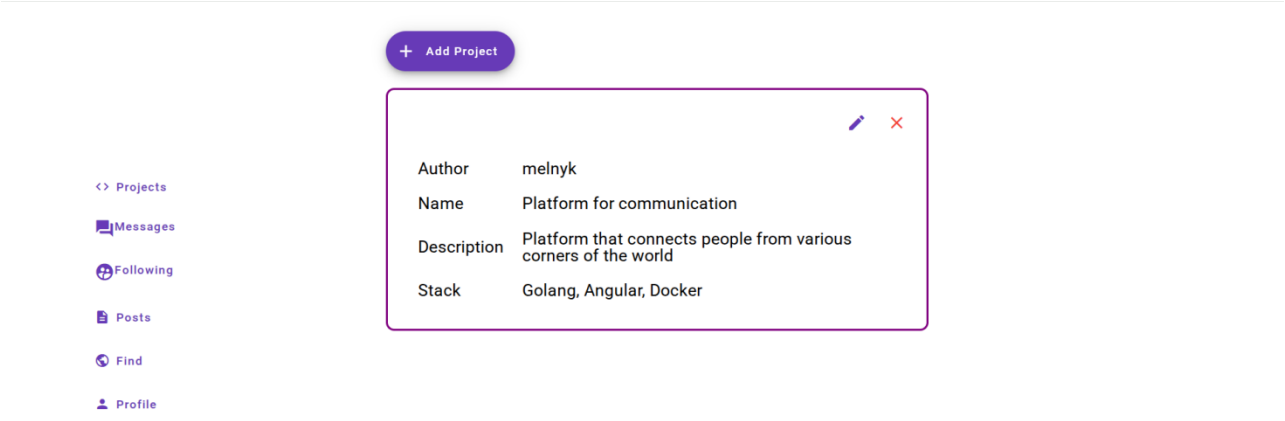

Рисунок 4.28 – Результат зміни інформації про проект

Для того, щоб видалити проєкт, потрібно натиснути кнопку «X» у верхньому правому куті інформації про нього. Програма видаляє відповідний проєкт (рисунок 4.29), отже функція видалення проєкту працює правильно.

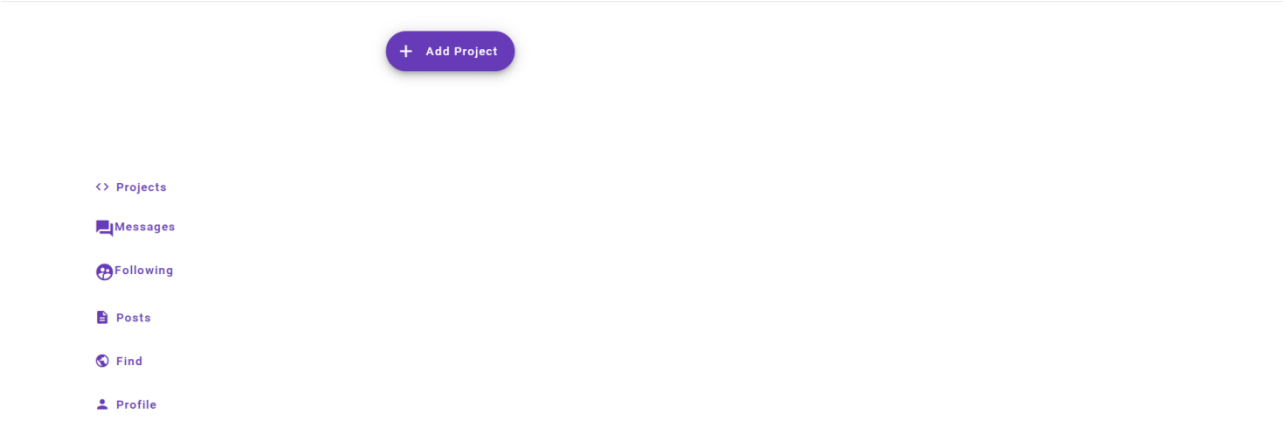

Рисунок 4.29 – Результат видалення інформації про проєкт

Оновлення та видалення постів, коментарів, повідомлень, чатів та проєктів може здійснюватися лише їх авторами.

Отже у ході тестування перевірено коректність роботи функцій системи та встановлено, що онлайн платформа для комунікації програмістів працює належним чином.

4.2 Розробка інструкції користувача

Система дозволяє виконати такі основні функції як: написання поста, оновлення поста, видалення поста, написання коментарів, оновлення та видалення коментарів, комунікація в чатах, оновлення та видалення повідомлень з чатів, оновлення та видалення чатів, знаходження користувачів, перегляд постів інших користувачів, реєстрація, авторизація, додавання нового проєкту, оновлення та видалення проєкту.

Для написання поста потрібно:

1. Обрати вкладку «Posts».
2. Натиснути кнопку «Write Post».
3. Ввести текст, та додати за необхідності картинки та файли.
4. Натиснути кнопку «Create».

Для зміни поста потрібно:

1. Обрати вкладку «Posts».
2. Вибрати пост.
3. Натиснути кнопку «Оновити» у вигляді олівця.
4. Ввести потрібні зміни.
5. Натиснути кнопку «Update».

Для видалення поста потрібно:

1. Обрати вкладку «Posts».
2. Вибрати пост.
3. Натиснути кнопку «X».

Для написання коментарів потрібно:

1. Обрати вкладку «Following».

2. Обрати користувача, пост якого треба прокоментувати.
3. Серед отриманих постів обрати необхідний.
4. Натиснути кнопку «Коментарі».
5. Натиснути кнопку «Write Comment»
6. Ввести текст, та додати за необхідності картинки та файли.
7. Натиснути кнопку «Create».

Для зміни коментаря потрібно:

1. Обрати вкладку «Following».
2. Обрати користувача, пост якого було прокоментовано.
3. Серед отриманих постів обрати необхідний.
4. Натиснути кнопку «Коментарі».
5. Вибрати коментар.
6. Натиснути кнопку «Оновити» у вигляді олівця.
7. Ввести потрібні зміни.
8. Натиснути кнопку «Update».

Для видалення коментаря потрібно:

1. Обрати вкладку «Following».
2. Обрати користувача, пост якого було прокоментовано.
3. Серед отриманих постів обрати необхідний.
4. Натиснути кнопку «Коментарі».
5. Вибрати коментар.
6. Натиснути кнопку «X».

Для комунікації в чатах потрібно:

1. Обрати вкладку «Messages».
2. Натиснути кнопку «Add».
3. Ввести назву кімнати та додати користувачів.
4. Натиснути кнопку «Create».
5. Натиснути кнопку «Write Message».
6. Ввести текст, та додати за необхідності картинки та файли.
7. Натиснути кнопку «Create».

Для того, щоб оновити повідомлення в чаті треба:

1. Обрати вкладку «Messages».
2. Обрати чат.
3. Обрати повідомлення.
4. Натиснути кнопку «Оновити» у вигляді олівця.
5. Ввести потрібні зміни.
6. Натиснути кнопку «Update».

Для того, щоб видалити повідомлення з чату потрібно:

1. Обрати вкладку «Messages».
2. Обрати чат.
3. Обрати повідомлення.
4. Натиснути кнопку «X».

Для того, щоб оновити інформацію про чат потрібно:

1. Обрати вкладку «Messages».
2. Обрати чат.
3. Натиснути кнопку «Update».
4. Ввести необхідні зміни.
5. Натиснути кнопку «Update».

Для того, щоб видалити чат потрібно:

1. Обрати вкладку «Messages».
2. Обрати чат.
3. Натиснути кнопку «Delete».

Для того, щоб знайти користувача треба:

1. Обрати вкладку «Find».
2. Ввести ключове слово.
3. Обрати з запропонованих профілів той, що відповідає його інтересам.
4. Натиснути кнопку «Follow».

Для перегляду постів інших користувачів потрібно:

1. Обрати вкладку «Following».
2. Обрати користувача, пост якого треба прочитати.

3. Ознайомитися з новим постом.

Для авторизації в системі потрібно:

1. У вікні авторизації ввести логін та пароль.
2. Натиснути кнопку «Login».

Для реєстрації в системі потрібно:

1. У вікні авторизації натиснути посилання «Register».
2. Заповнити форму, вказавши ім’я, прізвище, логін та пароль.
3. Натиснути кнопку «Register».

Для додавання нового проєкту потрібно:

1. Перейти на вкладку «Projects».
2. Натиснути кнопку «Add Project».
3. Заповнити форму, вказавши назву, опис та стек технологій проєкту.
4. Натиснути кнопку «Create».

Для оновлення проєкту потрібно:

1. Перейти на вкладку «Projects».
2. Обрати проєкт.
3. Натиснути кнопку «Оновити» у вигляді олівця.
4. Ввести потрібні зміни.
5. Натиснути кнопку «Update».

Для видалення проєкту потрібно:

1. Перейти на вкладку «Projects».
2. Обрати проєкт.
3. Натиснути кнопку «X».

Слідування наведеним інструкція забезпечить коректність використання онлайн платформи для комунікації програмістів та отримання очікуваних результатів.

4.3 Висновок

У четвертому розділі було протестовано застосунок та встановлено, що він функціонує належним чином. Сформовано інструкцію користувача, що

містить опис коректного використання таких функцій як: написання, оновлення та видалення постів, написання, оновлення та видалення коментарів, комунікація в чатах, оновлення та видалення чатів і повідомлень в них, знаходження користувачів, перегляд постів інших користувачів, реєстрація та авторизація, додавання, оновлення та видалення нового проєкту.

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи розроблено онлайн платформу для комунікації програмістів. Для розробки серверної частини використовувалась мова програмування Golang, для розробки графічного інтерфейсу користувача – фреймворк Angular.

Було доведено актуальність розробки, проведено порівняльний аналіз аналогів, обрано методи виконання поставлених завдань. На основі проведеного аналізу сформовано список функцій, які має реалізовувати платформа. Після цього розроблено модель системи та її алгоритми, обґрунтовано вибір архітектури. Наступним кроком став вибір інструментів програмування та безпосередня реалізація застосунку. Останнім етапом стало тестування застосунку та формування інструкцій для користувача.

Отже повністю виконано задачі, поставлені на початку роботи, які вимагали:

1. Сформувати перелік функціональних вимог до системи.
2. Розробити модель системи та алгоритми роботи системи.
3. Реалізувати програмне забезпечення системи.
4. Протестувати розроблений продукт на відповідність вимогам та відсутність помилок.

Система повністю реалізовує функції необхідні для користувача. Слідування інструкції користувача забезпечить швидке ознайомлення зі способами реєстрації, авторизації, публікування постів, коментування, написання повідомлень, додавання інформації про новий проєкт, зміни інформації про себе, знаходження нових користувачів та перегляду їх постів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Переваги оптоволоконного Інтернету порівняно зі звичайним мідним: Шлях до швидкості та надійності[Електронний ресурс] – Режим доступу до ресурсу: https://euronet.com.ua/83-perevagy-optovolokonnogo-internetu-porivniano-zi-zvychainym-midnym-shliakh-do-shvydkosti-ta-nadiinosti.html (дата звернення: 17.04.2025).

2. Мови програмування високого рівня[Електронний ресурс] – Режим доступу до ресурсу: https://viddaleno.com.ua/2024/11/14/movi-programuvannya-visokogo-rivnya/ (дата звернення: 17.04.2025).

3. What is version control? [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/git/tutorials/what-is-version-control (дата звернення: 17.04.2025).

4. Свіца О. С., Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025). Вибір архітектури онлайн платформи для комунікації програмістів, Вінниця, 2025.[Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24112 (дата звернення: 17.04.2025).

5. Свіца О. С., Молодь в науці: дослідження, проблеми, перспективи (МН-2025). Вибір мови програмування для реалізації онлайн платформи для комунікації програмістів, Вінниця, 2025.[Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25353 (дата звернення: 02.06.2025).

6. Тенденції розвитку медіа в сучасному світі. [Електронний ресурс] – Режим доступу до ресурсу: https://media-systems.ua/news/media_trends_2023 (дата звернення: 17.04.2025).

7. X[Електронний ресурс] – Режим доступу до ресурсу: https://x.com/home (дата звернення: 17.04.2025).

8. Facebook[Електронний ресурс] – Режим доступу до ресурсу: https://www.facebook.com/ (дата звернення: 17.04.2025).

9. Facebook users[Електронний ресурс] – Режим доступу до ресурсу: https://mezha.media/2023/07/28/facebook-users/ (дата звернення: 17.04.2025)

10. Social Media Posts Character Counts 2024[Електронний ресурс] – Режим доступу до ресурсу: https://www.upcontent.com/blog/post/social-media-posts-character-counts (дата звернення: 17.04.2025).

11. What Are Character Count Limits on Social Media & Email? [Електронний ресурс] – Режим доступу до ресурсу:https://capitalizemytitle.com/what-are-word-count-character-count-limits-on-various-social-media-platforms/ (дата звернення: 17.04.2025).

12. StackOverflow[Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/ (дата звернення: 17.04.2025).

13. Create an open channel[Електронний ресурс] – Режим доступу до ресурсу: https://sendbird.com/docs/chat/platform-api/v3/channel/creating-a-channel/create-an-open-channel#1-create-an-open-channel (дата звернення: 17.04.2025).

14. Що таке аналіз вимог? Процес і методи[Електронний ресурс] – Режим доступу до ресурсу: https://visuresolutions.com/uk/%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA-%D0%B7-%D0%B2%D1%96%D0%B4%D1%81%D1%82%D0%B5%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F-%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F-%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B0%D0%BC%D0%B8/%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7-%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3/ (дата звернення: 17.04.2025).

15. Аналіз та документація вимог проекту[Електронний ресурс] https://ua5.org/project/2045-analiz-ta-dokumentacziya-vymog-proektu.html (дата

звернення: 17.04.2025).

16. Richards M., Ford N. Fundamentals of Software. Boston : O'reilly, 2020. 400 p.

17. Newman S. Building microservices. 2nd ed. Boston : O'reilly, 2021. 586 p.

18. Who created the Go programming language? [Електронний ресурс] – Режим доступу до ресурсу: https://clouddevs.com/go/created-by/ (дата звернення: 17.04.2025).

19. Build simple, secure, scalable systems with Go[Електронний ресурс] – Режим доступу до ресурсу: https://go.dev/ (дата звернення: 17.04.2025)

20. Case Studies[Електронний ресурс] – Режим доступу до ресурсу: https://go.dev/solutions/case-studies (дата звернення: 17.04.2025)

21. Рейтинг мов програмування 2025[Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/articles/language-rating-2025/ (дата звернення: 17.04.2025).

22. Go vs Python For Web Development and Machine Learning Project[Електронний ресурс] – Режим доступу до ресурсу: https://www.bacancytechnology.com/blog/go-vs-python (дата звернення: 17.04.2025).

23. What Is Ownership? [Електронний ресурс] – Режим доступу до ресурсу: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html#what-is-ownership (дата звернення: 17.04.2025).

24. What is a goroutine? And what is their size? [Електронний ресурс] – Режим доступу до ресурсу: https://tpaschalis.me/goroutines-size/ (дата звернення: 17.04.2025).

25. Чому тестування необхідне? [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/chomu-testuvannya-neobhidne/ (дата звернення: 17.04.2025).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"_21_" _березня_____2025 р

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка онлайн платформи для комунікації програмістів» за спеціальністю 121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доц. каф. ПЗ Бабюк Н. П.
"_21_" ___березня_____2025 р.
Виконав:
студент гр.5ПІ-21б О.С. Свіца
"_21_" __березня_____2025

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка онлайн платформи для комунікації програмістів».

Галузь застосування – онлайн платформа

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Метою роботи є покращення процесу комунікації між програмістами шляхом розробки онлайн платформи для комунікації програмістів, що міститиме функціонал, відсутній на інших платформах, але необхідний для зручної взаємодії користувачів.

Призначення роботи – розробка засобів комунікації між програмістами.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Richards M., Ford N. Fundamentals of Software. Boston : O’reilly, 2020. 400 p.

2. Newman S. Building microservices. 2nd ed. Boston : O’reilly, 2021. 586 p.

3. Свіца О. С., Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025). Вибір архітектури онлайн платформи для комунікації програмістів, Вінниця, 2025.[Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24112 (дата звернення: 17.04.2025).

5. Технічні вимоги

Програма має бути реалізована у вигляді онлайн-платформи, її серверна частина має бути реалізована за допомогою мови програмування Golang, фронтенд створений за допомогою фреймворку Angular.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Сформувати перелік функціональних вимог до системи.	25.03.25 – 31.03.25
2	Розробити модель системи та алгоритми роботи системи.	01.04.25 – 15.04.25
3	Реалізувати програмне забезпечення системи.	15.04.25 – 20.05.25
4	Протестувати розроблений продукт на відповідність вимогам та відсутність помилок.	21.05.25 – 02.06.25

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка онлайн платформи для комунікації програмістів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібностітекстових запозичень, виявлених у роботі системою StrikePlagiarism 3,1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату,фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявленоознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник Бабюк Н. П. к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач Свіца О. С.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

```
package main

import (
	"social-media/internal/chats/repository"
	"social-media/internal/chats/server"
	"social-media/internal/chats/service"
	"social-media/internal/common/app"
	"social-media/internal/common/clients"
)

func main() {
	app.InitChatsService()

	repo := repository.New()

	authClient := clients.NewAuthClient()
	usersClient := clients.NewUsersClient()

	service := service.New(repo, authClient, usersClient)
	srv := server.New(service)
	srv.Run()
}

package server

import (
	"encoding/json"
	"net/http"
	"social-media/internal/chats/service"
	"social-media/internal/common"
	"social-media/internal/common/app/config"
	"social-media/internal/common/app/log"
	"social-media/internal/common/slice"
	"strconv"

	"github.com/gorilla/mux"
	"github.com/pkg/errors"
)

type Server interface {
	Run()
}

type server struct {
	router  *mux.Router
	service service.Service
}

func New(srv service.Service) Server {
	r := mux.NewRouter()
```

```
	s := &server{
		router:  r,
		service: srv,
	}
	r.Methods("GET").Path("/chats").HandlerFunc(s.getChats)
	r.Methods("POST").Path("/chats").HandlerFunc(s.createChat)
	r.Methods("GET").Path("/chats/{id}").HandlerFunc(s.getChat)
	r.Methods("PUT").Path("/chats/{id}").HandlerFunc(s.updateChat)
	r.Methods("DELETE").Path("/chats/{id}").HandlerFunc(s.deleteChat)

	return s
}

func (s *server) Run() {
	handler := common.CORS(s.router)

	err := http.ListenAndServeTLS(":"+config.App.ChatsService.Port, "server.crt",
"server.key", handler)
	if err != nil {
		log.Error(err)
		panic(err)
	}
}

func (s *server) getChats(w http.ResponseWriter, r *http.Request) {
	cookie, err := r.Cookie("token")
	if err != nil {
		writeError(w, common.ErrNoToken)
		return
	}

	token := cookie.Value
	chats, err := s.service.UserChats(token)
	if err != nil {
		writeError(w, common.ErrInternal)
		return
	}

	models := make([]BasicUserChat, 0, len(chats))
	for _, chat := range chats {
		model := BasicUserChat{
			ID:   chat.ID(),
			Name: chat.Name(),
		}
		models = append(models, model)
	}

	writeJSON(w, UserChatsResp(models))
}

func (s *server) createChat(w http.ResponseWriter, r *http.Request) {
```

```
	cookie, err := r.Cookie("token")
	if err != nil {
		writeError(w, common.ErrNoToken)
		return
	}

	if err := r.ParseMultipartForm(1 << 15); err != nil {
		writeError(w, common.ErrInvalidData)
		return
	}

	ids, err := slice.Convert(r.MultipartForm.Value["users_ids[]"], func(rawID string) (int,
error) {
		return strconv.Atoi(rawID)
	})
	if err != nil {
		writeError(w, common.ErrInvalidData)
		return
	}

	chat, err := s.service.CreateChat(service.CreateChatReq{
		Token:    cookie.Value,
		Name:     r.FormValue("name"),
		UsersIDs: ids,
	})
	if err != nil {
		writeError(w, err)
		return
	}

	writeJSON(w, chatToModel(chat))
}

func (s *server) getChat(w http.ResponseWriter, r *http.Request) {
	token, err := r.Cookie("token")
	if err != nil {
		writeError(w, common.ErrNoToken)
		return
	}

	params := mux.Vars(r)
	rawID, ok := params["id"]
	if !ok {
		writeError(w, common.ErrInvalidData)
		return
	}

	id, err := strconv.Atoi(rawID)
	if err != nil {
		writeError(w, common.ErrInvalidData)
		return
```

```
		}

		chat, err := s.service.Chat(token.Value, id)
		if err != nil {
			writeError(w, err)
			return
		}

		writeJSON(w, chatToModel(chat))
}

func (s *server) updateChat(w http.ResponseWriter, r *http.Request) {
		token, err := r.Cookie("token")
		if err != nil {
			writeError(w, common.ErrNoToken)
			return
		}

		if err := r.ParseMultipartForm(1 << 15); err != nil {
			writeError(w, common.ErrInvalidData)
			return
		}

		ids, err := slice.Convert(r.MultipartForm.Value["users_ids[]"], func(rawID string) (int,
error) {
			return strconv.Atoi(rawID)
		})
		if err != nil {
			writeError(w, common.ErrInvalidData)
			return
		}

		rawID, ok := mux.Vars(r)["id"]
		if !ok {
			writeError(w, common.ErrInvalidData)
			return
		}
		id, err := strconv.Atoi(rawID)
		if err != nil {
			writeError(w, common.ErrInvalidData)
			return
		}

		err = s.service.UpdateChat(service.UpdateChatReq{
			Token:    token.Value,
			ID:       id,
			Name:     r.FormValue("name"),
			UsersIDs: ids,
		})
		writeResponse(w, nil, err)
}
```

```
func (s *server) deleteChat(w http.ResponseWriter, r *http.Request) {
	token, err := r.Cookie("token")
	if err != nil {
		writeError(w, common.ErrNoToken)
		return
	}

	rawID, ok := mux.Vars(r)["id"]
	if !ok {
		writeError(w, common.ErrInvalidData)
		return
	}
	id, err := strconv.Atoi(rawID)
	if err != nil {
		writeError(w, common.ErrInvalidData)
		return
	}

	err = s.service.DeleteChat(token.Value, id)
	writeResponse(w, nil, err)
}

func writeResponse(w http.ResponseWriter, resp interface{}, err error) {
	if err != nil {
		writeError(w, err)
		return
	}
	writeJSON(w, resp)
}

func writeError(w http.ResponseWriter, err error) {
	code := 500
	msg := "Internal server error"
	if e, ok := err.(common.Error); ok {
		code = e.Code()
		msg = e.Message()
	}
	w.WriteHeader(code)

	_, err = w.Write([]byte(msg))
	if err != nil {
		log.Error(errors.WithStack(err))
	}
}

func writeJSON(w http.ResponseWriter, resp interface{}) {
	w.WriteHeader(200)
	err := json.NewEncoder(w).Encode(resp)
	if err != nil {
		log.Error(errors.WithStack(err))
```

```
	}
}

package server

import (
	"social-media/internal/chats/domain/chat"
	"social-media/internal/chats/domain/user"
)

type UserChatsResp []BasicUserChat

type BasicUserChat struct {
	ID   int    `json:"id"`
	Name string `json:"name"`
}

type UserModel struct {
	ID      int    `json:"id,omitempty"`
	Login   string `json:"login,omitempty"`
	Name    string `json:"name,omitempty"`
	Surname string `json:"surname,omitempty"`
}

func userToModel(u *user.User) UserModel {
	return UserModel{
		ID:      u.ID(),
		Login:   u.Login(),
		Name:    u.Name(),
		Surname: u.Surname(),
	}
}

type ChatModel struct {
	ID    int         `json:"id"`
	Name  string      `json:"name"`
	Owner UserModel   `json:"owner"`
	Users []UserModel `json:"users"`
}

func chatToModel(c *chat.Chat) ChatModel{
	users := make([]UserModel, 0, len(c.Users()))
	for _, user := range c.Users() {
		users = append(users, userToModel(user))
	}

	return ChatModel{
		ID: c.ID(),
		Name: c.Name(),
		Owner: userToModel(c.Owner()),
		Users: users,
```

```
	}
}

type CreateChatResp ChatModel

package service

import (
	"social-media/internal/chats/domain/chat"
	"social-media/internal/chats/domain/user"
	"social-media/internal/chats/repository"
	"social-media/internal/common"
	"social-media/internal/common/clients"
	"social-media/internal/common/slice"
)

type Service interface {
	UserChats(string) ([]*chat.Chat, error)
	CreateChat(CreateChatReq) (*chat.Chat, error)
	Chat(string, int) (*chat.Chat, error)
	UpdateChat(UpdateChatReq) error
	DeleteChat(string, int) error
}

type service struct {
	auth  clients.AuthClient
	users clients.UsersClient
	repo  repository.Repository
}

func New(repo repository.Repository, auth clients.AuthClient, users clients.UsersClient) Service {
	return &service{
		auth:  auth,
		users: users,
		repo:  repo,
	}
}

func (s *service) UserChats(token string) ([]*chat.Chat, error) {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return nil, err
	}

	return s.repo.UserChats(id)
}

func (s *service) CreateChat(req CreateChatReq) (*chat.Chat, error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		return nil, err
```

```
	}

	return s.repo.CreateChat(repository.CreateChatReq{
		Name:     req.Name,
		OwnerID:  id,
		UsersIDs: req.UsersIDs,
	})
}

func (s *service) Chat(token string, chatID int) (*chat.Chat, error) {
	_, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return nil, err
	}

	chat, err := s.repo.Chat(chatID)
	if err != nil {
		return nil, err
	}

	ids := slice.MustConvert(chat.Users(), func(u *user.User) int {
		return u.ID()
	})

	infos, err := s.users.UsersInfo(ids)
	if err != nil {
		return nil, err
	}

	for i, user := range chat.Users() {
		info := infos[i]
		user.AddInfo(info.Login, info.Name, info.Surname)
	}

	return chat, nil
}

func (s *service) UpdateChat(req UpdateChatReq) error {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		return err
	}

	ownerID, err := s.repo.ChatOwner(req.ID)
	if err != nil {
		return err
	}

	if ownerID != id {
		return common.ErrForbidden
	}
```

```
	users := slice.MustConvert(req.UsersIDs, func(id int) *user.User {
		return user.New(id)
	})
	c := chat.New(req.ID, req.Name, user.New(id), users)

	return s.repo.UpdateChat(c)
}

func (s *service) DeleteChat(token string, chatID int) error {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return err
	}

	ownerID, err := s.repo.ChatOwner(chatID)
	if err != nil {
		return err
	}

	if ownerID != id {
		return common.ErrForbidden
	}

	return s.repo.DeleteChat(chatID)
}

package service

type CreateChatReq struct {
	Token    string
	Name     string
	UsersIDs []int
}

type UpdateChatReq struct {
	Token    string
	ID       int
	Name     string
	UsersIDs []int
}

package repository

import (
	"bytes"
	"context"
	"database/sql"
	"fmt"
	"social-media/internal/chats/domain/chat"
	"social-media/internal/chats/domain/user"
```

```
	"social-media/internal/common"
	"social-media/internal/common/app/config"
	"social-media/internal/common/app/log"
	"social-media/internal/common/slice"
	"strconv"

	_ "github.com/lib/pq"
	"github.com/pkg/errors"
)

type Repository interface {
	UserChats(int) ([]*chat.Chat, error)
	Chat(int) (*chat.Chat, error)
	CreateChat(CreateChatReq) (*chat.Chat, error)
	UpdateChat(*chat.Chat) error
	ChatOwner(int) (int, error)
	DeleteChat(int) error
}

type repo struct {
	db *sql.DB
}

func New() Repository {
	user := config.App.PostgresDB.User
	password := config.App.PostgresDB.Password
	host := config.App.PostgresDB.Host
	port := config.App.PostgresDB.Port
	name := config.App.PostgresDB.Name
	url := fmt.Sprintf("postgres://%s:%s@%s:%s/%s?sslmode=disable", user, password, host,
port, name)
	db, err := sql.Open("postgres", url)
	if err != nil {
		log.Error(err)
		panic("Unable to connect to database")
	}
	return &repo{
		db: db,
	}
}

func (r *repo) UserChats(userID int) ([]*chat.Chat, error) {
	query := `SELECT chat_id, name FROM chats INNER JOIN
		(SELECT chat_id, user_id FROM chats_users WHERE user_id = $1) AS
user_chats
		ON chats.id = user_chats.chat_id;`
	rows, err := r.db.Query(query, userID)
	if err == sql.ErrNoRows {
		return nil, common.ErrNotFound
	}
	if err != nil {
```

```
                log.Error(errors.WithStack(err))
                return nil, common.ErrInternal
        }

        var chats []*chat.Chat
        for rows.Next() {
                var model ChatModel
                if err := rows.Scan(&model.ID, &model.Name); err != nil {
                        log.Error(errors.WithStack(err))
                        return nil, common.ErrInternal
                }
                c := chat.New(model.ID, model.Name, nil, nil)
                chats = append(chats, c)
        }

        return chats, nil
}

func (r *repo) Chat(id int) (*chat.Chat, error) {
        c, err := r.getChat(id)
        if err != nil {
                return nil, err
        }
        users, err := r.getChatUsers(id)
        if err != nil {
                return nil, err
        }

        return chat.New(c.ID(), c.Name(), c.Owner(), users), nil
}

func (r *repo) getChat(id int) (*chat.Chat, error) {
        query := `SELECT name, owner FROM chats WHERE id=$1`
        row := r.db.QueryRow(query, id)

        var ownerID int
        var name string
        if err := row.Scan(&name, &ownerID); err != nil {
                log.Error(errors.WithStack(err))
                return nil, common.ErrInternal
        }

        owner := user.New(ownerID)
        chat := chat.New(id, name, owner, nil)
        return chat, nil
}

func (r *repo) getChatUsers(chatID int) ([]*user.User, error) {
        query := `SELECT user_id FROM chats_users WHERE chat_id=$1`
        rows, err := r.db.Query(query, chatID)
        if err == sql.ErrNoRows {
```

```
			return nil, common.ErrNotFound
		}
		if err != nil {
			log.Error(errors.WithStack(err))
			return nil, common.ErrInternal
		}

		var users []*user.User
		for rows.Next() {
			var id int
			if err := rows.Scan(&id); err != nil {
				log.Error(errors.WithStack(err))
				return nil, common.ErrInternal
			}
			user := user.New(id)
			users = append(users, user)
		}
		return users, nil
}

func (r *repo) CreateChat(req CreateChatReq) (*chat.Chat, error) {
		tx, err := r.db.BeginTx(context.Background(), &sql.TxOptions{})
		if err != nil {
			log.Error(errors.WithStack(err))
			return nil, common.ErrInternal
		}

		id, err := r.createChat(tx, req.Name, req.OwnerID)
		if err != nil {
			if err := tx.Rollback(); err != nil {
				log.Error(errors.WithStack(err))
			}
			return nil, err
		}

		req.UsersIDs = append(req.UsersIDs, req.OwnerID)
		err = r.addChatUsers(tx, id, req.UsersIDs)
		if err != nil {
			if err := tx.Rollback(); err != nil {
				log.Error(errors.WithStack(err))
			}
			return nil, err
		}

		if err := tx.Commit(); err != nil {
			log.Error(errors.WithStack(err))
			return nil, common.ErrInternal
		}

		users := make([]*user.User, 0, len(req.UsersIDs))
		for _, id := range req.UsersIDs {
```

```
			users = append(users, user.New(id))
		}

		return chat.New(id, req.Name, user.New(req.OwnerID), users), nil
}

func (r *repo) createChat(tx *sql.Tx, name string, ownerID int) (int, error) {
		query := `INSERT INTO chats (name, owner) VALUES ($1, $2) RETURNING id`
		res := tx.QueryRow(query, name, ownerID)

		var id int
		if err := res.Scan(&id); err != nil {
			log.Error(errors.WithStack(err))
			return 0, common.ErrInternal
		}

		return id, nil
}

func (r *repo) addChatUsers(tx *sql.Tx, chatID int, usersIDs []int) error {
		var b bytes.Buffer
		b.WriteString(`INSERT INTO chats_users (chat_id, user_id) VALUES `)
		rawChatID := strconv.Itoa(chatID)
		for i, userID := range usersIDs {
			if i != 0 {
				b.WriteString(",")
			}
			b.WriteString("(")
			b.WriteString(rawChatID)
			b.WriteString(",")
			b.WriteString(strconv.Itoa(userID))
			b.WriteString(")")
		}

		query := b.String()
		_, err := tx.Exec(query)
		if err == sql.ErrTxDone {
			return common.ErrInternal
		}
		if err != nil {
			log.Error(errors.WithStack(err))
			return common.ErrInternal
		}

		return nil
}

func (r *repo) UpdateChat(c *chat.Chat) error {
		tx, err := r.db.Begin()
		if err != nil {
			log.Error(errors.WithStack(err))
```

```
                return common.ErrInternal
        }

        err = r.updateChat(tx, c.ID(), c.Name())
        if err != nil {
                if err := tx.Rollback(); err != nil {
                        log.Error(errors.WithStack(err))
                }
                return err
        }

        err = r.deleteChatUsers(tx, c.ID())
        if err != nil {
                if err := tx.Rollback(); err != nil {
                        log.Error(errors.WithStack(err))
                }
                return err
        }

        ids := slice.MustConvert(c.Users(), func(u *user.User) int {
                return u.ID()
        })
        ids = append(ids, c.Owner().ID())
        err = r.addChatUsers(tx, c.ID(), ids)
        if err != nil {
                if err := tx.Rollback(); err != nil {
                        log.Error(errors.WithStack(err))
                }
                return err
        }

        if err := tx.Commit(); err != nil {
                log.Error(errors.WithStack(err))
                return common.ErrInternal
        }

        return nil
}

func (r *repo) updateChat(tx *sql.Tx, chatID int, name string) error {
        query := `UPDATE chats SET name=$1 WHERE id=$2`

        _, err := tx.Exec(query, name, chatID)
        if err == sql.ErrTxDone {
                return common.ErrInternal
        }
        if err != nil {
                log.Error(errors.WithStack(err))
                return common.ErrInternal
        }
```

```
	return nil
}

func (r *repo) deleteChatUsers(tx *sql.Tx, chatID int) error {
	query := `DELETE FROM chats_users WHERE chat_id=$1`

	_, err := tx.Exec(query, chatID)
	if err == sql.ErrTxDone {
		return common.ErrInternal
	}
	if err != nil {
		log.Error(errors.WithStack(err))
		return common.ErrInternal
	}

	return nil
}

func (r *repo) ChatOwner(chatID int) (int, error) {
	query := `SELECT owner FROM chats WHERE id=$1`

	row := r.db.QueryRow(query, chatID)

	var ownerID int
	if err := row.Scan(&ownerID); err != nil {
		log.Error(errors.WithStack(err))
		return 0, common.ErrInternal
	}

	return ownerID, nil
}

func (r *repo) DeleteChat(id int) error {
	tx, err := r.db.Begin()
	if err != nil {
		log.Error(errors.WithStack(err))
		return common.ErrInternal
	}

	err = r.deleteChat(tx, id)
	if err != nil {
		if err := tx.Rollback(); err != nil {
			log.Error(errors.WithStack(err))
		}
		return err
	}

	err = r.deleteChatUsers(tx, id)
	if err != nil {
		if err := tx.Rollback(); err != nil {
			log.Error(errors.WithStack(err))
```

```
                }
                return err
        }

        if err := tx.Commit(); err != nil {
                log.Error(errors.WithStack(err))
                return common.ErrInternal
        }

        return nil
}

func (r *repo) deleteChat(tx *sql.Tx, id int) error {
        query := `DELETE FROM chats WHERE id=$1`

        _, err := tx.Exec(query, id)
        if err != nil {
                log.Error(errors.WithStack(err))
                return common.ErrInternal
        }

        return nil
}
```

```
package repository

type ChatModel struct {
        ID      int
        Name    string
        OwnerID  int
        UsersIDs []int
}

type CreateChatReq struct {
        Name    string
        OwnerID  int
        UsersIDs []int
}
```

```
package chat

import "social-media/internal/chats/domain/user"

type Chat struct {
        id   int
        name  string
        owner *user.User
        users []*user.User
}

func New(id int, name string, owner *user.User, users []*user.User) *Chat {
```

```
		return &Chat{
			id: id,
			name: name,
			owner: owner,
			users: users,
		}
}

func (c *Chat) ID() int {
		return c.id
}

func (c *Chat) Name() string {
		return c.name
}

func (c *Chat) Owner() *user.User {
		return c.owner
}

func (c *Chat) Users() []*user.User {
		return c.users
}
package user

type User struct{
		id int
		login string
		name string
		surname string
}

func New(id int) *User {
		return &User{
			id: id,
		}
}

func (u *User) ID() int {
		return u.id
}

func (u *User) Login() string{
		return u.login
}

func (u *User) Name() string {
		return u.name
}

func (u *User) Surname() string {
```

```
	return u.surname
}

func (u *User) AddInfo(login, name, surname string) {
	u.login = login
	u.name = name
	u.surname = surname
}

package service

import (
	"social-media/internal/common"
	"social-media/internal/common/app/log"
	"social-media/internal/common/clients"
	"social-media/internal/users/domain/user"
	"social-media/internal/users/repository"
)

type Service interface {
	CreateUser(string, string, string, string) (string, int, error)
	Login(string, string) (string, int, error)
	GetUser(string, int) (*user.User, error)
	UpdateUser(string, string, string, string, string) error
	GetUsersByInfo(string) ([]*user.User, error)
	FollowUser(string, int) error
	GetFollowedUsers(string) ([]*user.User, error)
	UsersInfo([]int)([]*user.User, error)
}

type userService struct {
	repo repository.Repository
	auth clients.AuthClient
}

func New(r repository.Repository, auth clients.AuthClient) Service {
	return &userService{
		repo: r,
		auth: auth,
	}
}

func (s *userService) CreateUser(login, name, surname, password string) (string, int, error) {
	var userID int
	var token string
	var finalErr error
	exists, err := s.repo.UserExists(login)
	if err != nil {
		finalErr = err
		return token, userID, finalErr
	}
```

```
		if exists {
			finalErr = common.ErrLoginExists
			return token, userID, finalErr
		}

		hashPsw, err := hashPassword(password)
		if err != nil {
			finalErr = err
			return token, userID, finalErr
		}

		userModel := repository.NewUserModel(login, name, surname, hashPsw)

		user, err := s.repo.CreateUser(userModel)
		if err != nil {
			finalErr = err
			return token, userID, finalErr
		}

		token, err = s.auth.GenerateToken(user.ID(), user.Login())
		if err != nil {
			finalErr = err
			return token, userID, finalErr
		}

		userID = user.ID()
		return token, userID, finalErr
}

func (s *userService) Login(login, password string) (string, int, error) {
		id, encodedPassw, err := s.repo.GetCredentials(login)
		if err != nil {
			return "", 0, err
		}

		if !checkPassword(password, encodedPassw) {
			return "", 0, common.ErrWrongCredentials
		}

		user := user.New(id, login)

		token, err := s.auth.GenerateToken(user.ID(), user.Login())
		if err != nil {
			return "", 0, err
		}
		return token, user.ID(), nil
}

func (s *userService) GetUser(token string, id int) (*user.User, error) {
		_, _, err := s.auth.ValidateToken(token)
		if err != nil {
```

```
                return nil, err
        }
        return s.repo.GetUser(id)
}

func (s *userService) UpdateUser(token, name, surname, bio, interests string) error {
        id, login, err := s.auth.ValidateToken(token)
        if err != nil {
                return err
        }
        user := user.New(id, login, user.WithName(name), user.WithSurname(surname),
                user.WithBio(bio), user.WithInterests(interests))

        err = s.repo.UpdateUser(user)
        if err != nil {
                log.Error(err)
                return common.ErrInternal
        }
        return nil
}

func (s *userService) GetUsersByInfo(info string) ([]*user.User, error) {
        return s.repo.GetUsersByInfo(info)
}

func (s *userService) FollowUser(token string, followedID int) error {
        id, _, err := s.auth.ValidateToken(token)
        if err != nil {
                return err
        }
        if id == followedID {
                return common.ErrInvalidData
        }

        exists, err := s.repo.SubscriptionExists(id, followedID)
        if err != nil {
                return err
        }
        if exists {
                return common.ErrSubscriptionExists
        }
        return s.repo.Subscribe(id, followedID)
}

func (s *userService) GetFollowedUsers(token string) ([]*user.User, error) {
        id, _, err := s.auth.ValidateToken(token)
        if err != nil {
                return nil, err
        }
        return s.repo.GetFollowedUsers(id)
}
```

```
func (s *userService) UsersInfo(ids []int)([]*user.User, error){
	return s.repo.UsersInfo(ids)
}

func (s *userService) UpdateReadMessages(token string, chatID int, lastReadMessageID string)
error {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return err
	}

	return s.repo.UpdateReadMessages(id, chatID, lastReadMessageID)
}

func (s *userService) UpdateReadPosts(token string, ownerID int, lastReadPostID string) error {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return err
	}

	return s.repo.UpdateReadMessages(id, ownerID, lastReadPostID)
}

func Remained(resources, deletedResources []string) (remainedResources []string, finalErr error) {
	resourcesSate := make(map[string]resourceLifecycle)

	for i := 0; i < len(resources); i++ {
		resourcesSate[resources[i]] = Created
	}

	for i := 0; i < len(deletedResources); i++ {
		deletedResource := deletedResources[i]
		_, ok := resourcesSate[deletedResource]
		if !ok {
			finalErr = common.ErrInvalidData
			return
		}
		resourcesSate[deletedResource] = Deleted
	}

	for resource, state := range resourcesSate {
		if state == Created {
			remainedResources = append(remainedResources, resource)
		}
	}
	return
}

package service
```

```
import (
	"social-media/internal/common"
	"social-media/internal/common/app/log"
	"social-media/internal/common/clients"
	"social-media/internal/common/files"
	"social-media/internal/posts/domain/message"
	"social-media/internal/posts/domain/post"
	"social-media/internal/posts/repository"

	"github.com/pkg/errors"
)

type Service interface {
	CreatePost(CreatePostReq) (*post.Post, error)
	GetPosts(string, int) ([]*post.Post, error)
	UpdatePost(UpdatePostReq) (*post.Post, error)
	DeletePost(string, string) error
	commentsService
	chatMessagesService
}

type postsService struct {
	repo  repository.Repository
	auth  clients.AuthClient
	users clients.UsersClient
}

func New(r repository.Repository, auth clients.AuthClient, users clients.UsersClient) Service {
	return &postsService{
		repo:  r,
		auth:  auth,
		users: users,
	}
}

func (s *postsService) CreatePost(req CreatePostReq) (*post.Post, error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		log.Error(errors.WithStack(err))
		return nil, common.ErrInvalidToken
	}

	imagesPaths, err := files.Process(req.Images)
	if err != nil {
		return nil, err
	}
	filesPaths, err := files.Process(req.Files)
	if err != nil {
		return nil, err
	}
```

```
		post, err := s.repo.CreatePost(repository.CreatePostReq{
			CreateMessageReq: repository.CreateMessageReq{
				UserID:      id,
				Text:        req.Text,
				FilesPaths:  filesPaths,
				ImagesPaths: imagesPaths,
			},
		})
		if err != nil {
			return nil, err
		}
		return post, nil
}

func (s *postsService) GetPosts(token string, userID int) ([]*post.Post, error) {
		_, _, err := s.auth.ValidateToken(token)
		if err != nil {
			return nil, err
		}

		return s.repo.UserPosts(userID)
}

func (s *postsService) UpdatePost(req UpdatePostReq) (*post.Post, error) {
		id, _, err := s.auth.ValidateToken(req.Token)
		if err != nil {
			return nil, err
		}

		p, err := s.repo.GetPost(req.ID)
		if err != nil {
			return nil, err
		}

		if p.UserID() != id {
			return nil, common.ErrForbidden
		}

		p.Update(req.Text, req.Images, req.Files, req.DeletedImages, req.DeletedFiles)

		if err := s.repo.UpdatePost(p); err != nil {
			return nil, err
		}
		return p, nil
}

func (s *postsService) DeletePost(token, id string) error {
		userID, _, err := s.auth.ValidateToken(token)
		if err != nil {
			return err
		}
```

```
	post, err := s.repo.GetPost(id)
	if err != nil {
		return err
	}

	if post.UserID() != userID {
		return common.ErrForbidden
	}

	return s.repo.DeletePost(id)
}

func addUsersFullNames[T message.MessageI](s *postsService, messages []T) error {
	if messages == nil {
		return nil
	}

	ids := make([]int, 0, len(messages))
	for _, comment := range messages {
		ids = append(ids, comment.UserID())
	}

	fullNames, err := s.users.UsersInfo(ids)
	if err != nil {
		return err
	}

	for i, fullName := range fullNames {
		messages[i].AddUserFullName(fullName.Name, fullName.Surname)
	}

	return nil
}

package service

import (
	"social-media/internal/common"
	"social-media/internal/common/files"
	"social-media/internal/posts/domain/comment"
	"social-media/internal/posts/repository"
)

type commentsService interface {
	PostComments(string, string) ([]*comment.Comment, error)
	CreatePostComment(CreatePostCommentReq) (*comment.Comment, error)
	DeletePostComment(string, string, string) error

	UpdateComment(UpdateCommentReq) (*comment.Comment, error)
```

```
	CommentComments(string, string) ([]*comment.Comment, error)
	CreateCommentComment(CreateCommentCommentReq) (*comment.Comment, error)
	DeleteCommentComment(string, string, string) error
}

func (s *postsService) PostComments(token, id string) ([]*comment.Comment, error) {
	_, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return nil, err
	}

	comments, err := s.repo.PostComments(id)
	if err != nil {
		return nil, err
	}

	addUsersFullNames(s, comments)

	return comments, nil
}

func (s *postsService) CreatePostComment(req CreatePostCommentReq) (*comment.Comment,
error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		return nil, err
	}

	imagesPaths, err := files.Process(req.Images)
	if err != nil {
		return nil, err
	}
	filesPaths, err := files.Process(req.Files)
	if err != nil {
		return nil, err
	}

	comment, err := s.repo.CreatePostComment(repository.CreatePostCommentReq{
		PostID: req.PostID,
		CreateMessageReq: repository.CreateMessageReq{
			UserID:      id,
			Text:        req.Text,
			ImagesPaths: imagesPaths,
			FilesPaths:  filesPaths,
		},
	})
	if err != nil {
		return nil, err
	}

	return comment, nil
```

```
}

func (s *postsService) DeletePostComment(token, parentID, commentID string) error {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return err
	}

	comment, err := s.repo.GetComment(commentID)
	if err != nil {
		return err
	}

	if comment.UserID() != id {
		return common.ErrForbidden
	}

	return s.repo.DeletePostComment(parentID, commentID)
}

func (s *postsService) UpdateComment(req UpdateCommentReq) (*comment.Comment, error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		return nil, err
	}

	comment, err := s.repo.GetComment(req.ID)
	if err != nil {
		return nil, err
	}

	if comment.UserID() != id {
		return nil, common.ErrForbidden
	}

	err = comment.Update(req.Text, req.Images, req.Files, req.DeletedImages,
req.DeletedFiles)
	if err != nil {
		return nil, err
	}

	if err := s.repo.UpdateComment(comment); err != nil {
		return nil, err
	}

	return comment, nil
}

func (s *postsService) CommentComments(token, commentID string) ([]*comment.Comment,
error) {
	_, _, err := s.auth.ValidateToken(token)
```

```
	if err != nil {
		return nil, err
	}
	comments, err := s.repo.CommentComments(commentID)
	if err != nil {
		return nil, err
	}

	addUsersFullNames(s, comments)

	return comments, nil
}

func (s *postsService) CreateCommentComment(req CreateCommentCommentReq)
(*comment.Comment, error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		return nil, err
	}

	imagesPaths, err := files.Process(req.Images)
	if err != nil {
		return nil, err
	}
	filesPaths, err := files.Process(req.Files)
	if err != nil {
		return nil, err
	}

	return s.repo.CreateCommentComment(repository.CreateCommentCommentReq{
		CommentID: req.ParentID,
		CreateMessageReq: repository.CreateMessageReq{
			UserID:      id,
			Text:        req.Text,
			ImagesPaths: imagesPaths,
			FilesPaths:  filesPaths,
		},
	})
}

func (s *postsService) DeleteCommentComment(token, parentID, commentID string) error {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return err
	}

	comment, err := s.repo.GetComment(commentID)
	if err != nil {
		return err
	}
```

```
	if comment.UserID() != id {
		return common.ErrForbidden
	}

	return s.repo.DeleteCommentComment(parentID, commentID)
}

package service

import (
	"social-media/internal/common"
	"social-media/internal/common/app/log"
	"social-media/internal/common/files"
	"social-media/internal/posts/domain/chatmessage"
	"social-media/internal/posts/repository"

	"github.com/pkg/errors"
)

type chatMessagesService interface {
	ChatMessages(string, int) ([]*chatmessage.ChatMessage, error)
	CreateChatMessage(CreateChatMessageReq) (*chatmessage.ChatMessage, error)
	UpdateChatMessage(UpdateChatMessageReq) (*chatmessage.ChatMessage, error)
	DeleteChatMessage(string, string) error
}

func (s *postsService) ChatMessages(token string, chatID int) ([]*chatmessage.ChatMessage,
error) {
	_, _, err := s.auth.ValidateToken(token)
	if err != nil {
		return nil, err
	}
	messages, err := s.repo.ChatMessages(chatID)
	if err != nil {
		return nil, err
	}

	addUsersFullNames(s, messages)
	return messages, nil
}

func (s *postsService) CreateChatMessage(req CreateChatMessageReq)
(*chatmessage.ChatMessage, error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		log.Error(errors.WithStack(err))
		return nil, common.ErrInvalidToken
	}

	images, err := files.Process(req.Images)
	if err != nil {
```

```
		return nil, err
	}
	files, err := files.Process(req.Files)
	if err != nil {
		return nil, err
	}

	m, err := s.repo.CreateChatMessage(repository.CreateChatMessageReq{
		CreateMessageReq: repository.CreateMessageReq{
			UserID:      id,
			Text:        req.Text,
			ImagesPaths: images,
			FilesPaths:  files,
		},
		ChatID: req.ChatID,
	})
	if err != nil {
		return nil, err
	}

	err = s.repo.UpdateReadMessages(m.UserID(), m.ChatID(), m.ID())
	if err != nil {
		return nil, err
	}

	return m, nil
}

func (s *postsService) UpdateChatMessage(req UpdateChatMessageReq)
(*chatmessage.ChatMessage, error) {
	id, _, err := s.auth.ValidateToken(req.Token)
	if err != nil {
		log.Error(errors.WithStack(err))
		return nil, common.ErrInvalidToken
	}

	message, err := s.repo.ChatMessage(req.ID)
	if err != nil {
		return nil, err
	}

	if message.UserID() != id {
		return nil, common.ErrForbidden
	}

	err = message.Update(req.Text, req.Images, req.Files, req.DeletedImages,
req.DeletedFiles)
	if err != nil {
		return nil, err
	}
```

```
	if err := s.repo.UpdateChatMessage(message); err != nil {
		return nil, err
	}

	return message, nil
}

func (s *postsService) DeleteChatMessage(token, messageID string) error {
	id, _, err := s.auth.ValidateToken(token)
	if err != nil {
		log.Error(errors.WithStack(err))
		return common.ErrInvalidToken
	}

	message, err := s.repo.ChatMessage(messageID)
	if err != nil {
		return err
	}

	if message.UserID() != id {
		return common.ErrForbidden
	}

	return s.repo.DeleteChatMessage(messageID)
}

package service

import (
	"social-media/internal/common"
	"social-media/internal/common/clients"
	"social-media/internal/common/slice"
	"social-media/internal/projects/domain/project"
	"social-media/internal/projects/repository"
)

type Service interface {
	CreateProject(string, string, string, string) (int, error)
	UpdateProject(string, int, string, string, string) error
	GetProjects(string) ([]*project.Project, error)
	DeleteProject(string, int) error
}

type service struct {
	auth clients.AuthClient
	users clients.UsersClient
	repo repository.Repository
}

func New(repo repository.Repository, auth clients.AuthClient, users clients.UsersClient) Service {
	return &service{
```

```
                repo: repo,
                users: users,
                auth: auth,
        }
}

func (s *service) CreateProject(token, name, description, stack string) (int, error) {
        userID, _, err := s.auth.ValidateToken(token)
        if err != nil {
                return 0, err
        }

        return s.repo.CreateProject(name, description, stack, userID)
}

func (s *service) UpdateProject(token string, projectID int, name, description, stack string) error {
        userID, _, err := s.auth.ValidateToken(token)
        if err != nil {
                return err
        }

        p, err := s.repo.GetProject(projectID)
        if err != nil {
                return err
        }

        if p.UserID() != userID {
                return common.ErrForbidden
        }

        return s.repo.UpdateProject(project.New(projectID, name, description, stack, userID))
}

func (s *service) GetProjects(token string) ([]*project.Project, error) {
        _, _, err := s.auth.ValidateToken(token)
        if err != nil {
                return nil, err
        }

        projects, err := s.repo.GetProjects()
        if err != nil {
                return nil, err
        }

        usersIDs := slice.MustConvert(projects, func(p *project.Project) int {
                return p.UserID()
        })
        fullNames, err := s.users.UsersInfo(usersIDs)
        if err != nil {
                return nil, err
        }
```

```
		for i, fullName := range fullNames {
			projects[i].SetUserLogin(fullName.Login)
		}

		return projects, nil
}

func (s *service) DeleteProject(token string, projectID int) error {
		userID, _, err := s.auth.ValidateToken(token)
		if err != nil {
			return err
		}

		p, err := s.repo.GetProject(projectID)
		if err != nil {
			return err
		}

		if p.UserID() != userID {
			return common.ErrForbidden
		}

		return s.repo.DeleteProject(projectID)
}
```

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { SharedService } from '../../services/shared.service';
import { WebsocketService } from '../../services/websocket.service';
import { CookieService } from 'ngx-cookie-service';
import { MatDialog } from '@angular/material/dialog';
import { ErrorComponent } from '../error/error.component';

export interface Response {
   token: string
		id: number
}

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent {
		hide = true;
		login: string = "";
		name: string = "";
		surname: string = "";
		password: string = "";
```

```
	confirmedPassword: string = "";

	constructor(
		private http: HttpClient,
		private router: Router,
		private sharedService: SharedService,
		private websocket: WebsocketService,
		private cookieService: CookieService,
		public dialog: MatDialog,
	){}

	onSubmit(){
		if (this.password != this.confirmedPassword){
			const dialogRef = this.dialog.open(ErrorComponent, {
				data: {
					code: "",
					message: "Login and Confirmed Login should be
equal",
					statusText: "Wrong Confirmed Login",
				},
				height: '400px',
				width: '800px',
			})
		}
		const postData = new FormData();
		const login = this.login;
		postData.append('login',  login);
		postData.append('name', this.name);
		postData.append('surname',  this.surname);
		postData.append('password',  this.password);

		this.http.post<Response>('https://localhost:8081/users', postData).subscribe({
			next: response => {
				console.log(response);
				this.saveData(response.token, login, response.id);

				this.websocket.start();
				this.sharedService.start();
				this.sharedService.bol = true;
				this.router.navigate(['/']);
			},
			error: err => {
				const dialogRef = this.dialog.open(ErrorComponent, {
					data: {
						code: err.status,
						message: err.error,
						statusText: err.statusText,
					},
					height: '400px',
					width: '800px',
				})
```

```
                        }
                });
                this.clearData();
        }

        saveData(token: string, login: string, id: number){
                const expiryDate = new Date();
                expiryDate.setHours(expiryDate.getHours() + 1);
                this.cookieService.set('token', token, expiryDate, '/', 'localhost', true, 'None');

                localStorage.setItem("login", login);
                localStorage.setItem("id", id.toString());
        }

        clearData(){
                this.login = "";
                this.name = "";
                this.surname = "";
                this.password = "";
                this.confirmedPassword = "";
        }
}
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА
РОЗРОБКА ОНЛАЙН ПЛАТФОРМИ ДЛЯ КОМУНІКАЦІЇ ПРОГРАМІСТІВ

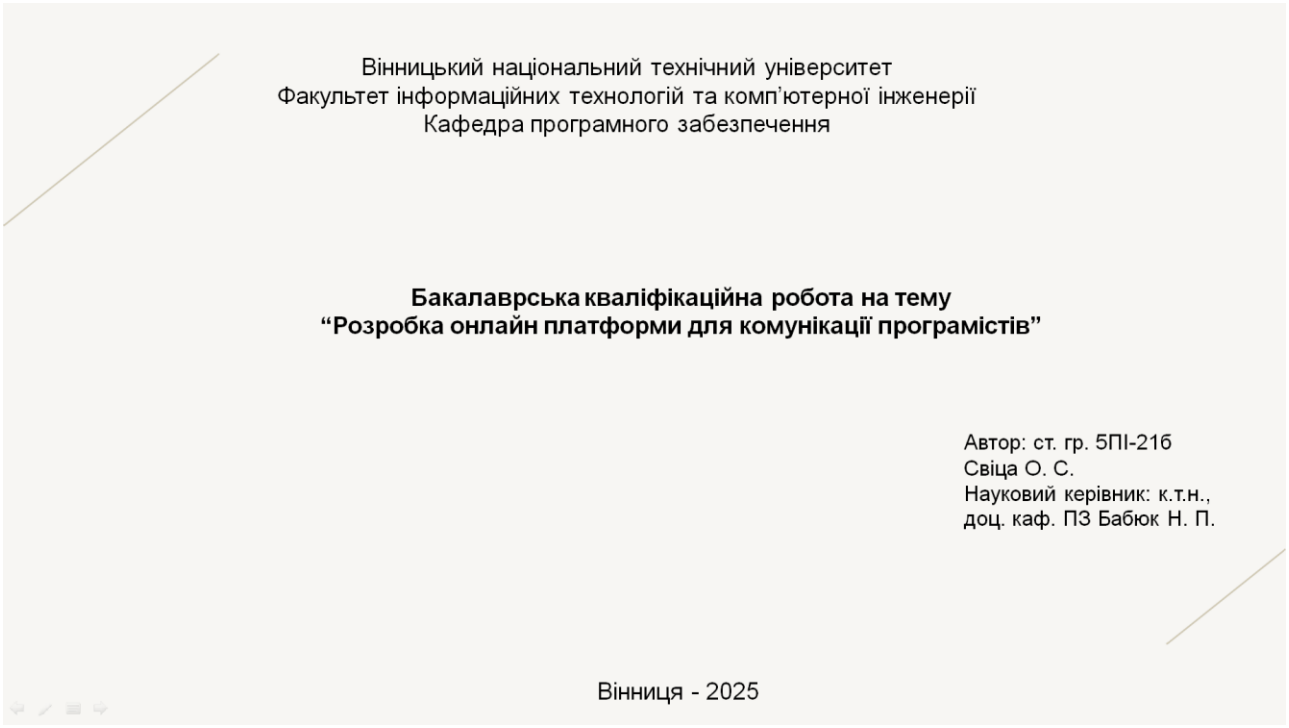

Рисунок Г.1 – Титульний аркуш

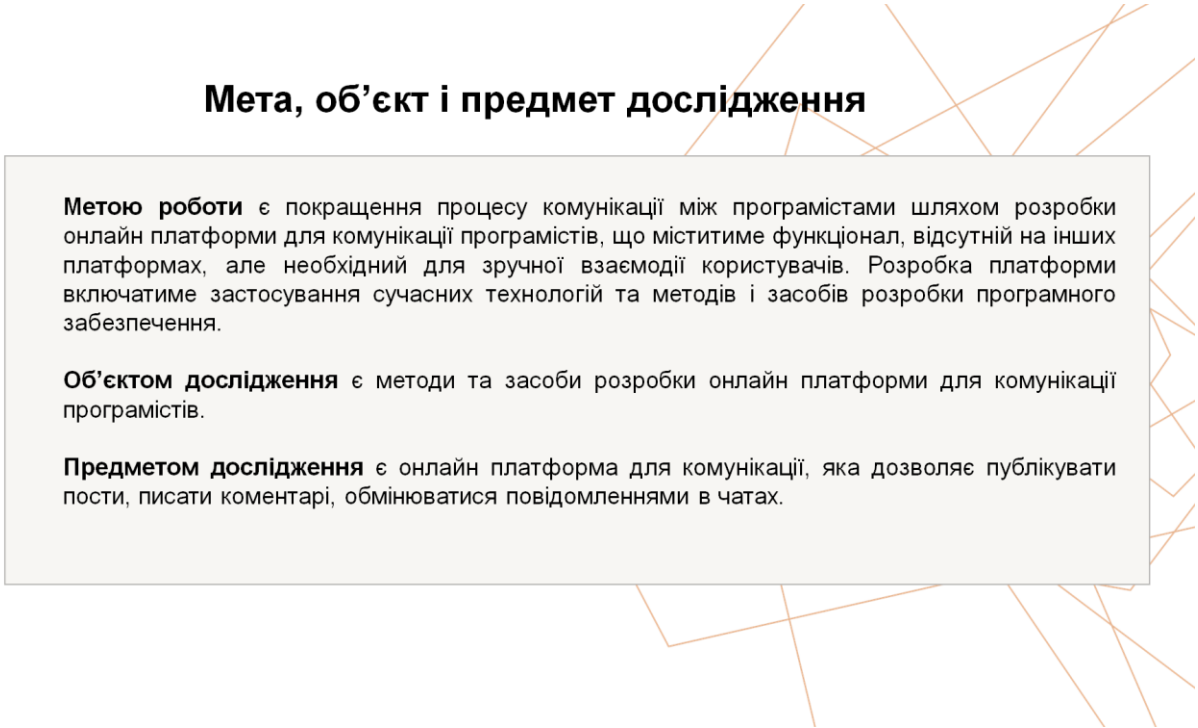

Рисунок Г.2 – Мета, об’єкт і предмет дослідження

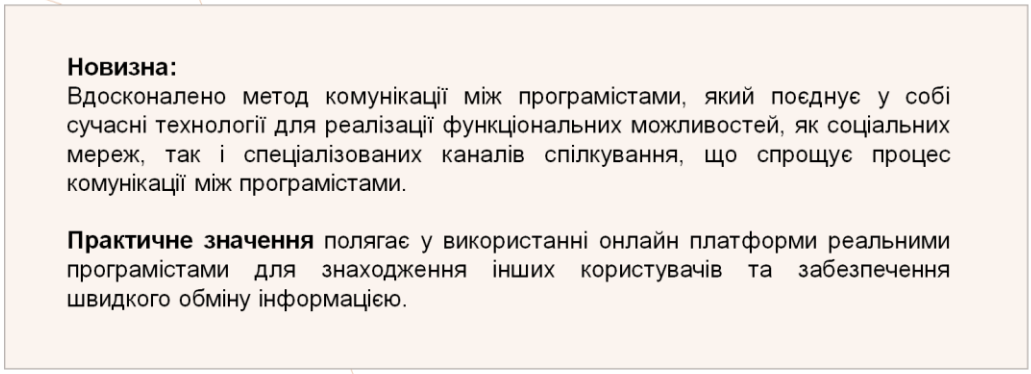

Рисунок Г.3 – Новизна і практичне значення роботи

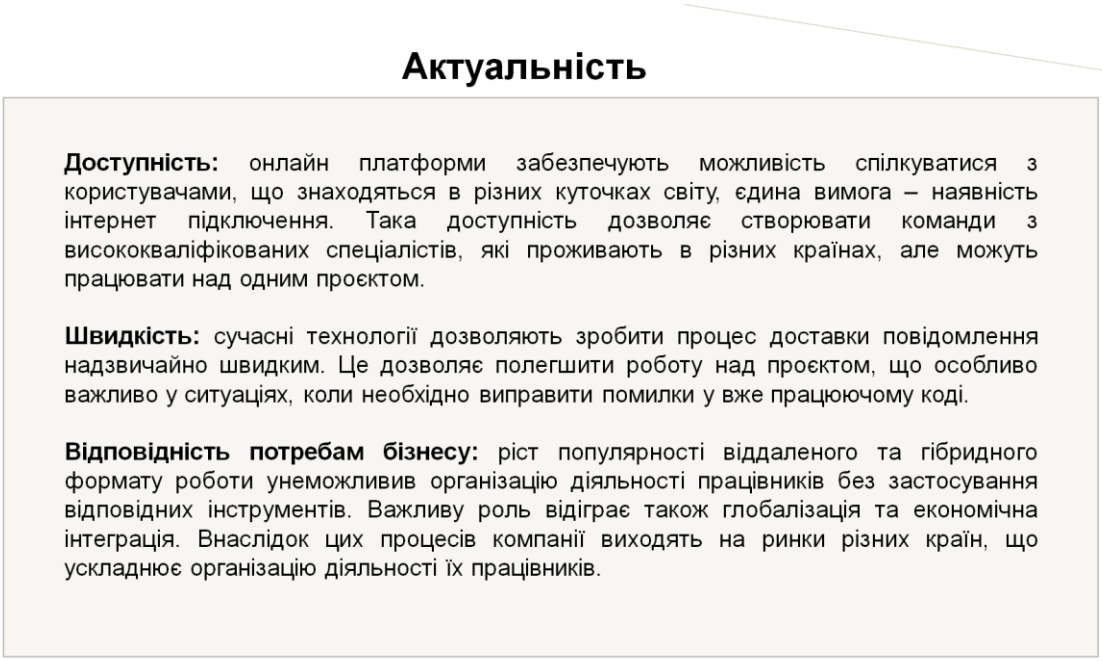

Рисунок Г.4 – Актуальність

Порівняльний аналіз аналогів

Характеристики	X	Facebook	StackOverflow	Платформа для комунікації
Публікація посту	+	+	+/-	+
Комунікація в чатах	+	+	+	+
Написання коментарів	+	+	+	+
Підсвітка коду	-	-	+	+
Оголошення про проєкти	-	-	-	+
Пошук користувачів	+	+	-	+
Загальна оцінка	70%	70%	60%	100%

Рисунок Г.5 – Порівняльний аналіз аналогів

Вибір мови програмування

Характеристики	Python	Java	Rust	C++	Golang
Швидкість	-	-	+	+	+
Конкурентне виконання коду	+/-	+	+	+	+
Простота	+	+/-	-	-	+
Статична типізація	-	+	+	+	+
Захищеність	-	+	+	-	+
Збірник сміття	+	+	-	-	+

Найменування операції	Golang	Python	Одиниці вимірювання
Бінарний пошук	20.8	2442.1	нс/оп
Сортування бульбашкою	0.09	6.7	с/оп
Читання з файлу	5305	58359	нс/оп
Опрацювання HTTP запиту	0.07	1.26	мс/запит

Рисунок Г.6 – Вибір мови програмування

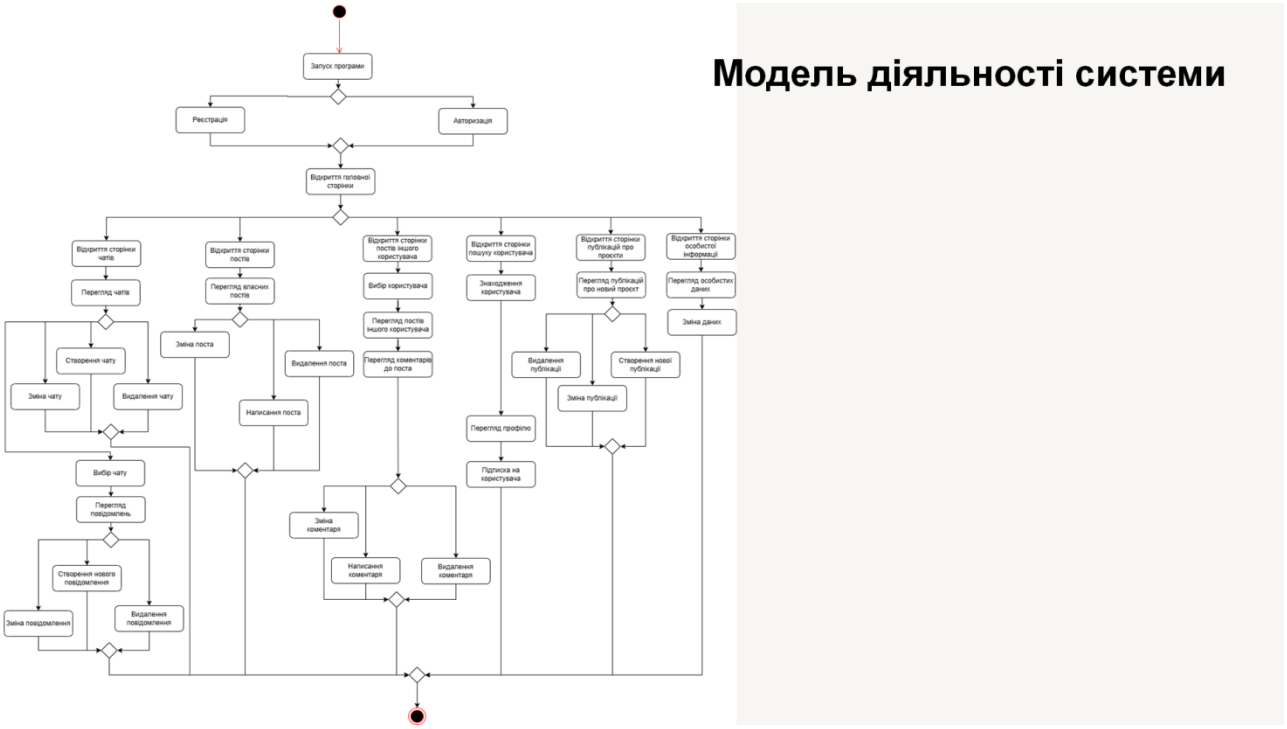

Рисунок Г.7 – Модель діяльності системи

Рисунок Г.8 – Алгоритм створення нового користувача

Рисунок Г.9 – алгоритм визначення видалених і доступних ресурсів

Рисунок Г.10 – Методи та засоби реалізації

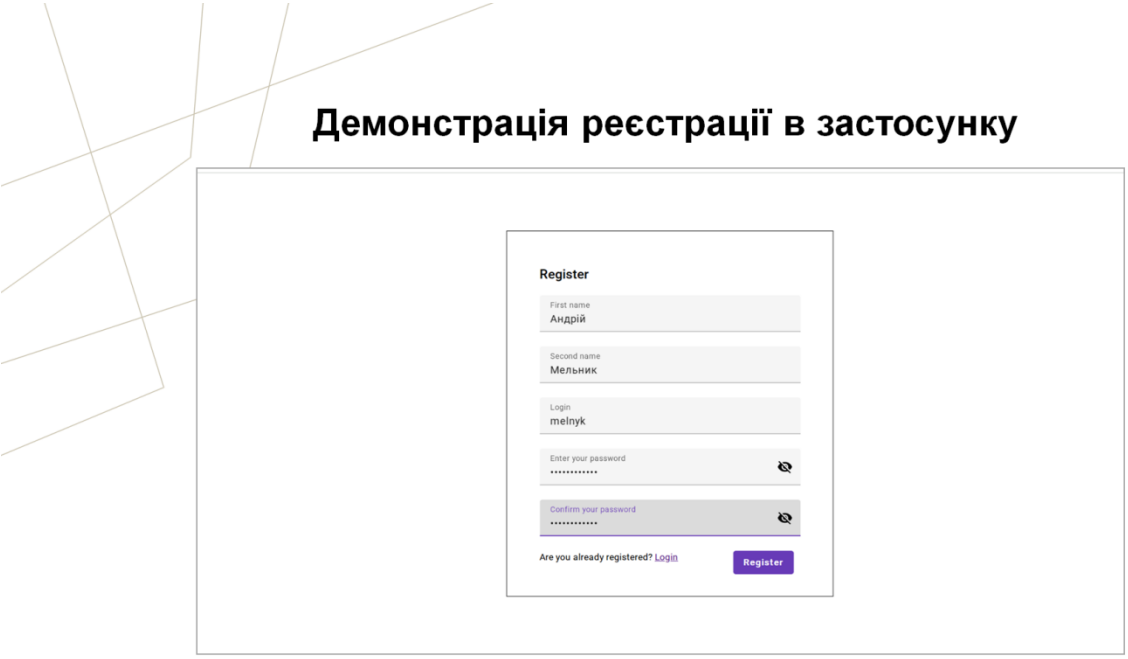

Рисунок Г.11 – Демонстрація реєстрації в застосунку

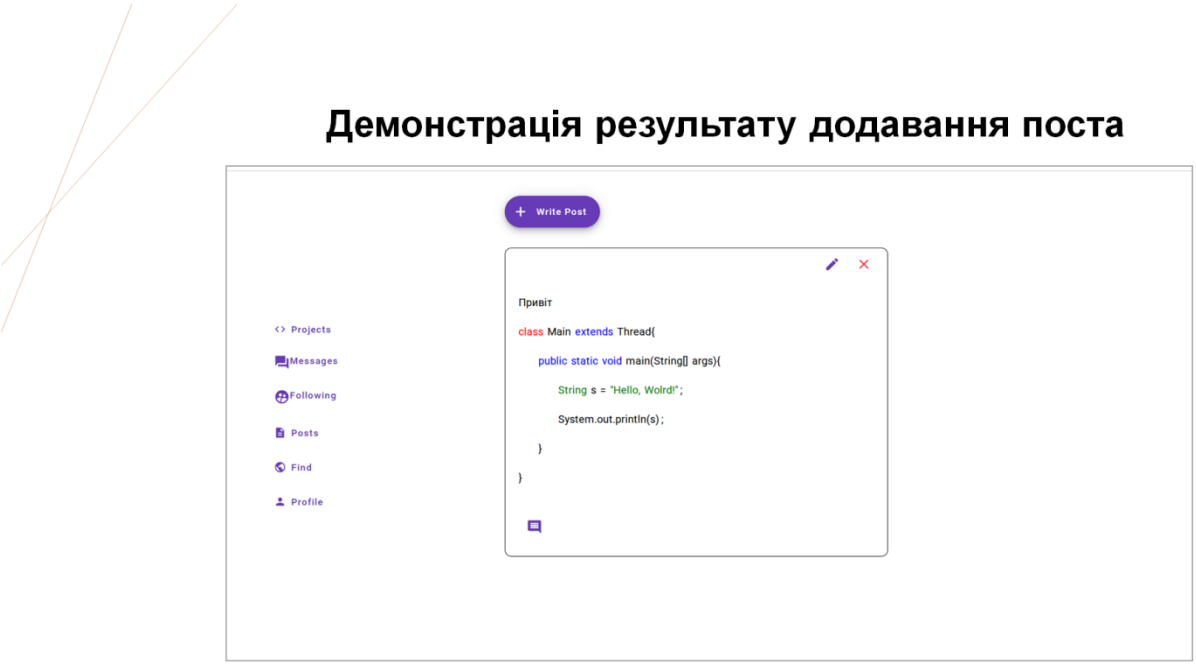

Рисунок Г.12 – Демонстрація результату додавання поста

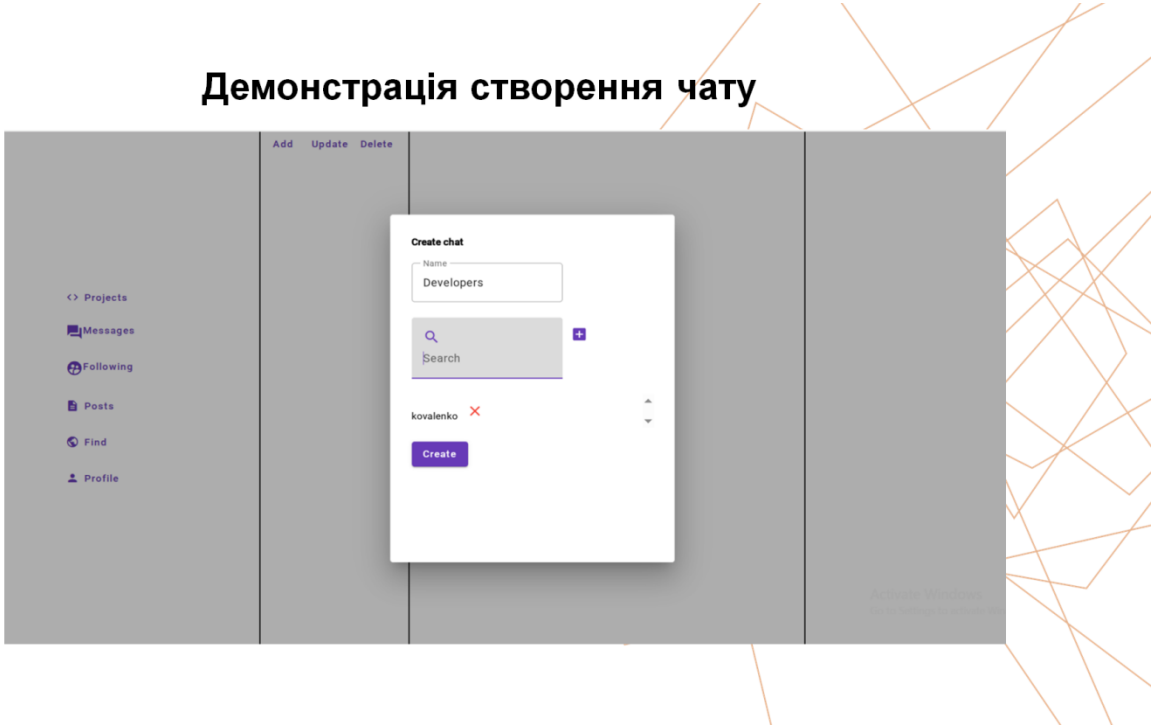

Рисунок Г.13 – Демонстрація створення чату

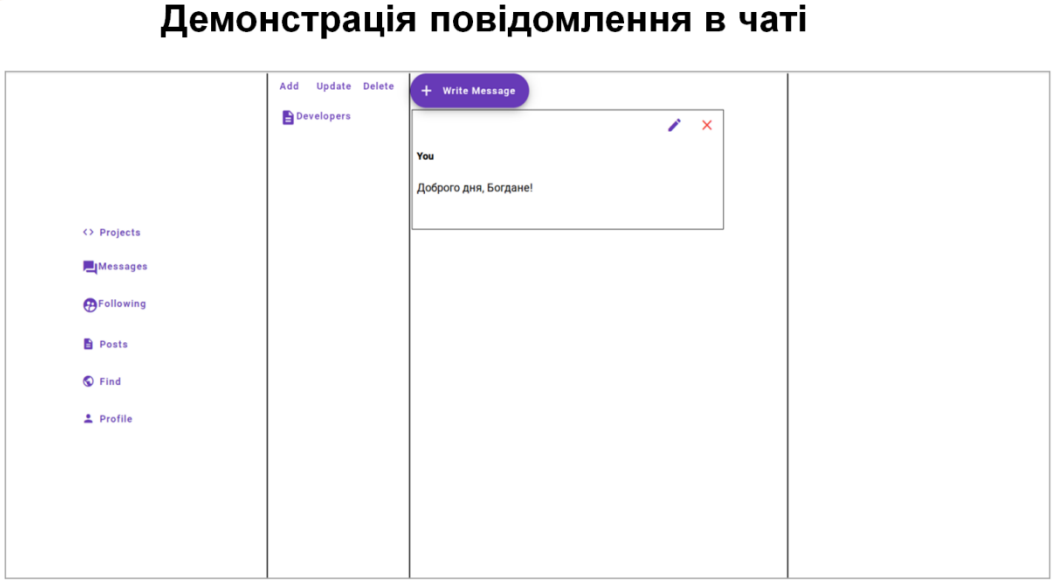

Рисунок Г.14 – Демонстрація повідомлення в чаті

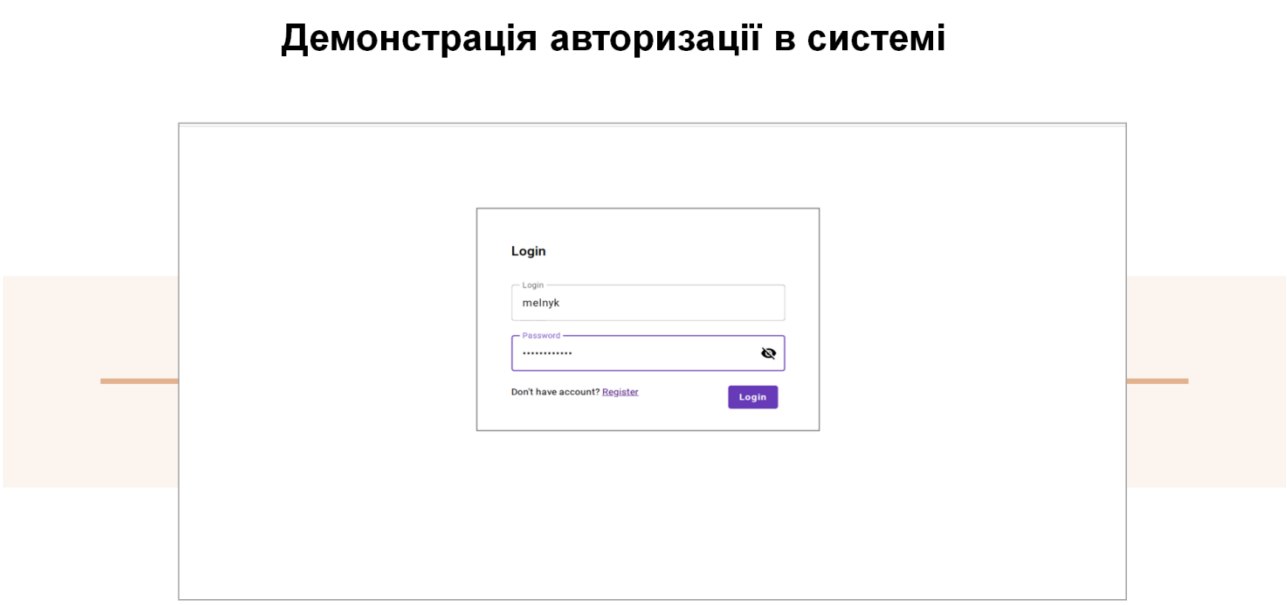

Рисунок Г.15 – Демонстрація авторизації в системі

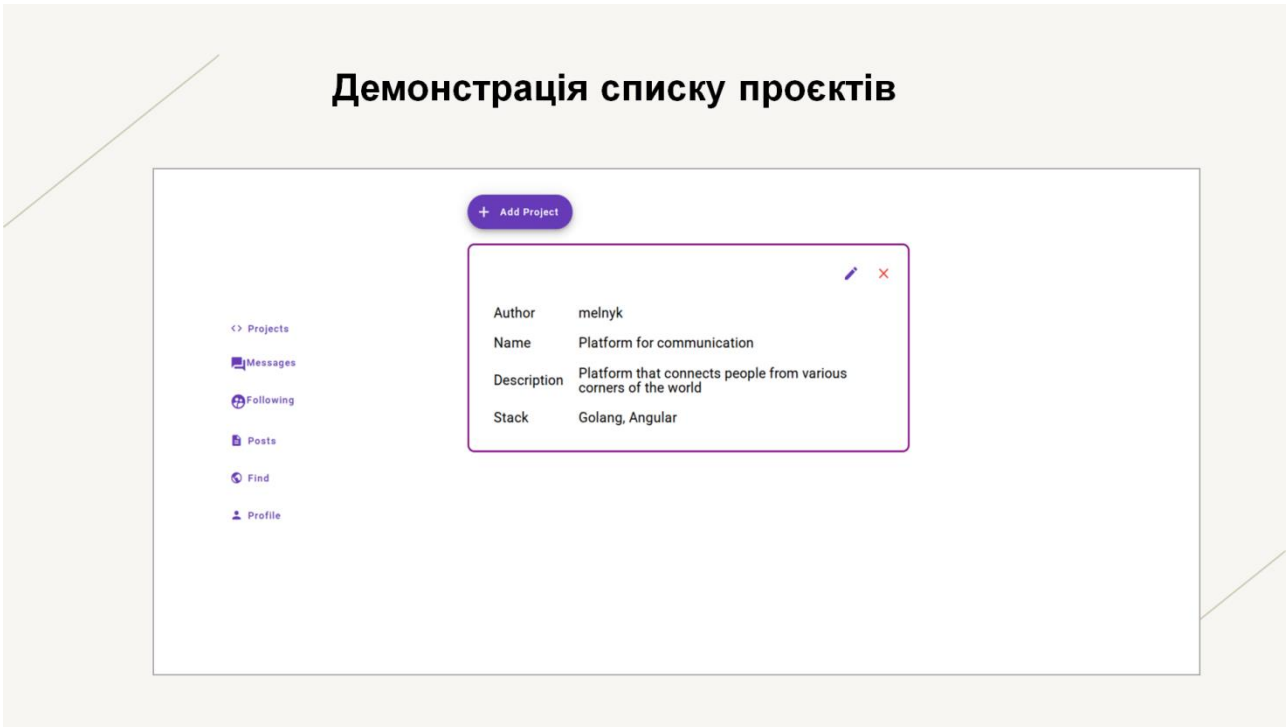

Рисунок Г.16 – Демонстрація списку проєктів

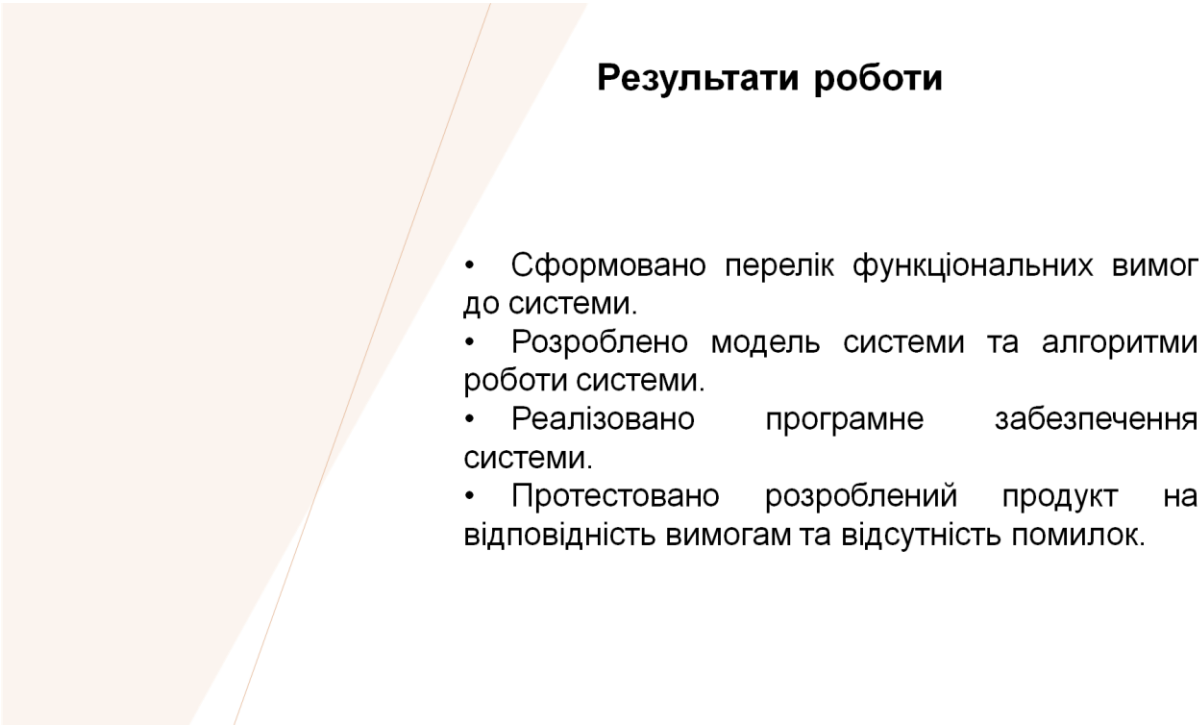

Рисунок Г.17 – Результати роботи

Рисунок Г.18 – Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

Рисунок Г.19 – Кінцевий слайд