Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Інформаційна система обміну повідомленнями та взаємодії користувачів
Назва теми

КвРІСТ 200196.20.01.08 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *126 «Інформаційні системи та технології»*
Шифр, назва

Освітня програма *«Інформаційні системи та технології»*
Назва

Виконав: студент III курсу, група ІСТс-20-1 _____ А.М. Луценко
Підпис Ініціали, прізвище

Керівник _____ К.Ю. Бобровнікова
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«26» червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 126 Інформаційні системи та технології

Освітня програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"11" 01 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Луценку Антону Михайловичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Інформаційна система обміну повідомленнями та взаємодії користувачів

Керівник проекту (роботи) Бобровнікова К.Ю., к.т.н. доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. №5

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження інформаційної системи обміну повідомленнями та взаємодії користувачів та постановка задачі

Проектування інформаційної системи обміну повідомленнями та взаємодії користувачів

Програмна реалізація інформаційної системи обміну повідомленнями та взаємодії користувачів

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Планування проектних робіт

Проектування архітектури інформаційної системи

Функціонування інформаційної системи

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «10» січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2023	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2023	виконано
3	Робота над розділом 1 – дослідження інформаційної системи обміну повідомленнями та взаємодії користувачів та постановка задачі	01.03.2023	виконано
4	Робота над розділом 2 – проектування інформаційної системи обміну повідомленнями та взаємодії користувачів	01.04.2023	виконано
5	Робота над розділом 3 – програмна реалізація інформаційної системи обміну повідомленнями та взаємодії користувачів	29.04.2023	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2023	виконано
7	Попередній захист ВКР	26.05.2023	виконано
8	Захист ВКР на засіданні ЕК	Червень 2023 року	

Студент _____ А.М. Луценко
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ К.Ю. Бобровнікова
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРІСТ 200196.20.01.08 ПЗ	Пояснювальна записка	56		
			Графічні матеріали			
2		КвРІСТ 200196.20.01.08	Планування проектних робіт	1		
3		КвРІСТ 200196.20.01.08	Проектування архітектури інформаційної системи	1		
4		КвРІСТ 200196.20.01.08	Функціонування інформаційної системи	1		

Зм	Арк	№ докум	Підпис	Дата	КвРІСТ 200196.20.01.08 ВП		
Розробив		Луценко			Відомість проекту	Літера	Аркуш / Аркушів
Перевір.		Бобровнікова				Н	1 / 1
Н. контр.		Лисенко				ХНУ, ІСТс-20-1	
Затв.		Говорущенко		26.06			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Інформаційна система обміну повідомленнями та взаємодії користувачів».

Автор роботи: Луценко Антон Михайлович.

Керівник роботи: Бобровнікова Кіра Юліївна.

Пояснювальна записка: 56 с., 39 рис., 4 табл., 3 дод., 40 джерел.

Графічна частина: 3 графічних додатки.

ІНФОРМАЦІЙНА СИСТЕМА, АРХІТЕКТУРА ІНФОРМАЦІЙНИХ СИСТЕМ, ФУНКЦІОНУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ, ОБМІН ПОВІДОМЛЕННЯМИ, БАЗА ДАНИХ, АУТЕНТИФІКАЦІЯ

Метою кваліфікаційної роботи є підвищення зручності обміну повідомленнями користувачів шляхом створення та реалізації інформаційної системи обміну повідомленнями та взаємодії користувачів.

Об'єктом дослідження є процес обміну повідомленнями та взаємодії користувачів.

Предметом дослідження є інформаційна система обміну повідомленнями та взаємодії користувачів.

Практичне значення має спроектована та реалізована інформаційна система обміну повідомленнями та взаємодії користувачів, яка забезпечує можливість створення блогу та обміну повідомленнями між користувачами.

Підпис студента

08.06.2023

Дата

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application programming interface

HTML – HyperText Markup Language

JWT - JSON Web Token

БД – База даних

UML - Unified Modeling Language

CSS - Cascading Style Sheets

ВСТУП

У сучасному цифровому світі, де спілкування та обмін інформацією стали необхідною складовою нашого повсякденного життя, інформаційні системи обміну повідомленнями та взаємодії користувачів займають важливе місце. Ці системи дозволяють користувачам створювати свої блоги, спілкуватись з іншими користувачами та обмінюватись ідеями та інформацією в онлайн-середовищі.

Актуальність теми полягає в потребі в розробці ефективної та зручної інформаційної системи обміну повідомленнями та взаємодії користувачів, яка задовольняє сучасні вимоги спілкування та обміну інформацією.

Метою кваліфікаційної роботи є підвищення зручності обміну повідомленнями користувачів шляхом створення та реалізації інформаційної системи обміну повідомленнями та взаємодії користувачів.

Предметом дослідження є інформаційна система обміну повідомленнями та взаємодії користувачів.

Об'єктом дослідження є процес обміну повідомленнями та взаємодії користувачів.

Практичне значення має спроектована та реалізована інформаційна система обміну повідомленнями та взаємодії користувачів, яка забезпечує можливість створення блогу та обміну повідомленнями між користувачами.

Для досягнення мети будуть проведені наступні кроки: аналіз сучасних тенденцій у сфері комунікації та обміну повідомленнями, визначення функціональних та нефункціональних вимог до системи, проектування архітектури системи, розробка необхідних модулів та компонентів, вибір технологій для реалізації проекту на C# .NET Core API, оцінка ефективності системи.

Результати цієї кваліфікаційної роботи не лише надають практичне значення для користувачів, але й сприятимуть розширенню знань у галузі розробки інформаційних систем обміну повідомленнями та взаємодії

користувачів. Вони можуть бути використані для подальшого вдосконалення та розширення подібних систем у майбутньому.

1 ДОСЛІДЖЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА ВЗАЄМОДІЇ КОРИСТУВАЧІВ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Інформаційні технології стали неодмінною частиною сучасного світу і займають важливе місце у бізнесі, громадському житті та особистому використанні. Одним з найбільш важливих засобів комунікації є обмін повідомленнями, який дозволяє людям швидко та ефективно обмінюватися інформацією [1].

Проте, існуючі інформаційні системи обміну повідомленнями не завжди відповідають вимогам користувачів та мають свої проблеми. У зв'язку з цим, ця тема є актуальною для подальшого розвитку інформаційних технологій та поліпшення якості обміну повідомленнями.

Інформаційна система обміну повідомленнями та взаємодії користувачів - це програмне забезпечення, яке дозволяє користувачам обмінюватися повідомленнями в електронному форматі. Такі системи використовуються в різних галузях, включаючи бізнес, освіту, зв'язок.

Однією з основних проблем існуючих інформаційних систем обміну повідомленнями є їх обмежена функціональність та складність використання. Більшість систем не забезпечують додаткові можливості, такі як відправка файлів великого розміру або можливість відеозв'язку. Крім того, користувачам часто доводиться виконувати багато дій, щоб відправити або отримати повідомлення.

Іншою проблемою є недостатня безпека та конфіденційність інформації, що обмінюється. Багато існуючих систем не мають належних заходів захисту інформації, що може призвести до втрати даних або їх незаконного доступу.

Також, проблемою є низька ефективність використання існуючих систем. Деякі системи можуть бути повільними та нестабільними, що призводить до затримок в обміні повідомленнями та загального зниження продуктивності користувачів.

Отже, проблеми можуть бути такі [2-3]:

1) безпека: одна з основних проблем при розробці інформаційної системи обміну повідомленнями та взаємодії користувачів - це безпека. Інформаційна система повинна забезпечити конфіденційність, цілісність та доступність інформації, що передається між користувачами. Інформаційна система повинна бути захищена від хакерських атак та інших видів кіберзлочинності;

2) масштабованість: інформаційна система повинна бути масштабована, тобто вона повинна здатися працювати з великою кількістю користувачів і обробляти великий обсяг даних;

3) користувацький інтерфейс: інформаційна система повинна мати зручний та інтуїтивно зрозумілий користувацький інтерфейс. Користувачі повинні бути здатні швидко та легко знаходити необхідні функції та операції;

4) сумісність: інформаційна система повинна бути сумісна з іншими програмними продуктами та пристроями, що використовуються користувачами;

5) підтримка: інформаційна система повинна мати ефективну підтримку, що забезпечує користувачам допомогу при виникненні проблем.

А завдання мають такий вигляд [4-5]:

1) розробка захисту від кіберзлочинності: інформаційна система повинна мати захист від хакерських атак та інших видів кіберзлочинності;

2) розробка масштабованості: інформаційна система повинна бути розроблена з урахуванням масштабованості та здатності обробляти великий обсяг даних;

3) розробка зручного користувацького інтерфейсу: користувацький інтерфейс повинен бути зручним та інтуїтивно зрозумілим для користувачів. Це означає, що система повинна мати легкість використання та зрозумілий дизайн, щоб користувачі могли легко знаходити необхідні функції та операції;

4) розробка сумісності: інформаційна система повинна бути розроблена з урахуванням сумісності з іншими програмними продуктами та пристроями, що використовуються користувачами. Це допоможе уникнути проблем зі збереженням та передачею даних;

5) розробка ефективної підтримки: інформаційна система повинна мати ефективну підтримку, що забезпечує користувачам допомогу при виникненні проблем. Це означає, що система повинна мати можливість звернення до служби підтримки та швидкий відгук на запити користувачів;

6) розробка функціональності: інформаційна система повинна мати ряд функціональності, які допоможуть користувачам взаємодіяти між собою та обмінюватися повідомленнями. Наприклад, система повинна мати можливість відправляти повідомлення, зберігати контакти, створювати групи користувачів, надавати можливість змінювати настройки приватності тощо;

7) розробка моніторингу та аналітики: інформаційна система повинна мати можливість моніторити та аналізувати взаємодію користувачів, щоб забезпечити їм кращу взаємодію та більш ефективний обмін повідомленнями. Наприклад, система повинна мати можливість аналізувати частоту взаємодію користувачів, досліджувати часові параметри відправки та отримання повідомлень, аналізувати популярність функцій системи;

8) забезпечення безпеки та конфіденційності: інформаційна система повинна бути розроблена з урахуванням безпеки та конфіденційності даних, які обмінюються користувачами. Це означає, що система повинна мати вбудовану систему захисту від хакерських атак, шифрування повідомлень та забезпечення доступу до них лише авторизованим користувачам;

9) розробка мобільності: інформаційна система повинна мати можливість працювати на різних мобільних пристроях, таких як смартфони та планшети. Це допоможе користувачам отримувати та відправляти повідомлення з будь-якого місця та в будь-який час;

10) розробка масштабованості: інформаційна система повинна бути розроблена з урахуванням можливості масштабування. Це означає, що система повинна мати можливість збільшення обсягу даних та користувачів без втрати продуктивності та ефективності роботи.

Окрім цього, можуть виникати й інші проблеми та завдання, такі як недостатня швидкість роботи системи, проблеми зі сумісністю між різними

платформами та пристроями, недостатня стійкість до перебоїв у мережі та технічних проблем, проблеми з безпекою даних та інші. Для розв'язання цих проблем необхідно ретельно аналізувати потреби користувачів та розробляти відповідні функції та можливості системи.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Порівняльний аналіз існуючих рішень в галузі інформаційної системи обміну повідомленнями та взаємодії користувачів допомагає визначити переваги та недоліки кожного рішення.

Інформаційні системи обміну повідомленнями та взаємодії користувачів є необхідними для забезпечення комунікації між співробітниками в організації. Розглянемо переваги та недоліки декількох існуючих рішень.

Gmail - це безкоштовна електронна пошта, надана компанією Google (рис. 1.1).

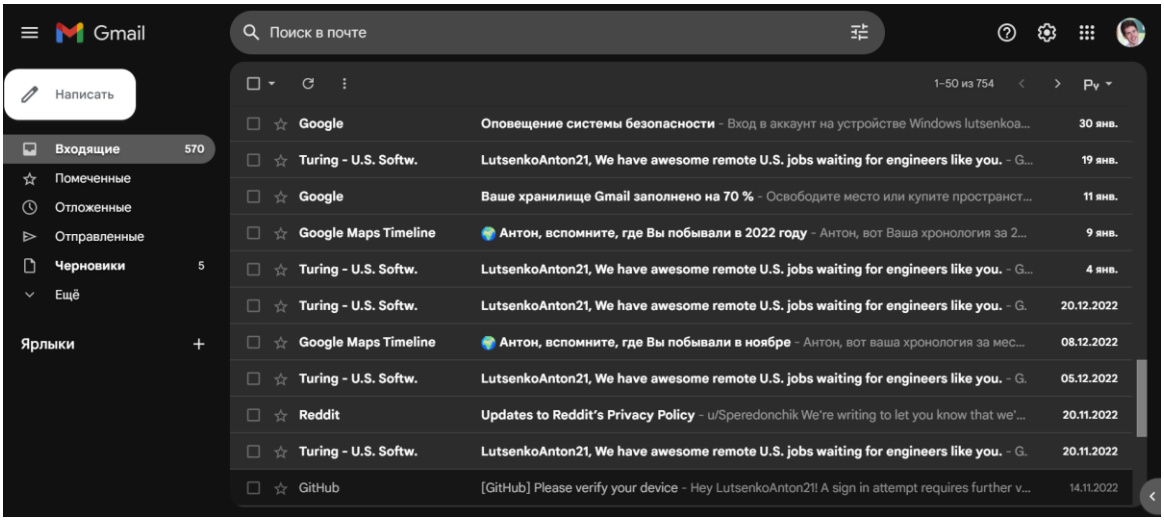


Рисунок 1.1 – Gmail.

Переваги використання Gmail:

1) широко використовується та доступна більшості користувачів;

2) дозволяє відправляти повідомлення з будь-якого пристрою, який має доступ до Інтернету;

3) можна відправляти повідомлення з великими вкладеннями;

4) можна відправляти повідомлення багатьом одночасно;

5) великий обсяг сховища: Gmail надає користувачам великий обсяг сховища для зберігання електронної пошти та вкладень. Завдяки цьому, ви можете зберігати велику кількість повідомлень і файли без необхідності постійно видаляти стару інформацію;

6) швидкий та ефективний пошук: Gmail має потужну систему пошуку, яка дозволяє швидко знаходити необхідні повідомлення за ключовими словами, відправниками або іншими параметрами. Це спрощує організацію та пошук пошти;

7) інтеграція з іншими сервісами Google: Gmail легко інтегрується з іншими сервісами Google, такими як Google Календар, Google Диск та Google Keep. Це дозволяє зручно обмінюватись даними та спільно працювати з колегами та партнерами;

8) безпека та захист даних: Gmail має високий рівень безпеки, включаючи захист від спаму, відсіювання шкідливих вкладень та застосування шифрування для захисту конфіденційної інформації.

Недоліки використання Gmail:

1) може бути переповнена непотрібними повідомленнями та спамом;

2) може бути неефективною для комунікації в режимі реального часу, так як не завжди користувачі можуть бути онлайн для отримання повідомлень;

3) недостатньо зручно для спільної роботи над документами та проектами;

4) залежність від інтернету: для використання Gmail потрібне постійне підключення до Інтернету. Це може створювати проблеми, якщо ви перебуваєте у зоні з обмеженим або відсутнім доступом до інтернету;

5) обмежена можливість редагування вкладень: Gmail має обмеження на редагування файлів без необхідності завантажувати їх на Google Диск. Це може

бути незручним, якщо вам потрібно вносити зміни в документи безпосередньо з поштової скриньки;

6) відображення реклами: Gmail відображає контекстну рекламу залежно від вмісту повідомлень. Для деяких користувачів це може бути небажаним, оскільки реклама може вважатися нав'язливою або порушувати конфіденційність;

7) політика приватності: як і більшість онлайн-сервісів, Gmail збирає та аналізує деяку інформацію про користувачів з метою персоналізації реклами та поліпшення послуг. Деяким користувачам це може бути проблематично з точки зору приватності.

Slack - це комунікаційний і колабораційний інструмент, призначений для командної роботи та спілкування у реальному часі (рис. 1.2).

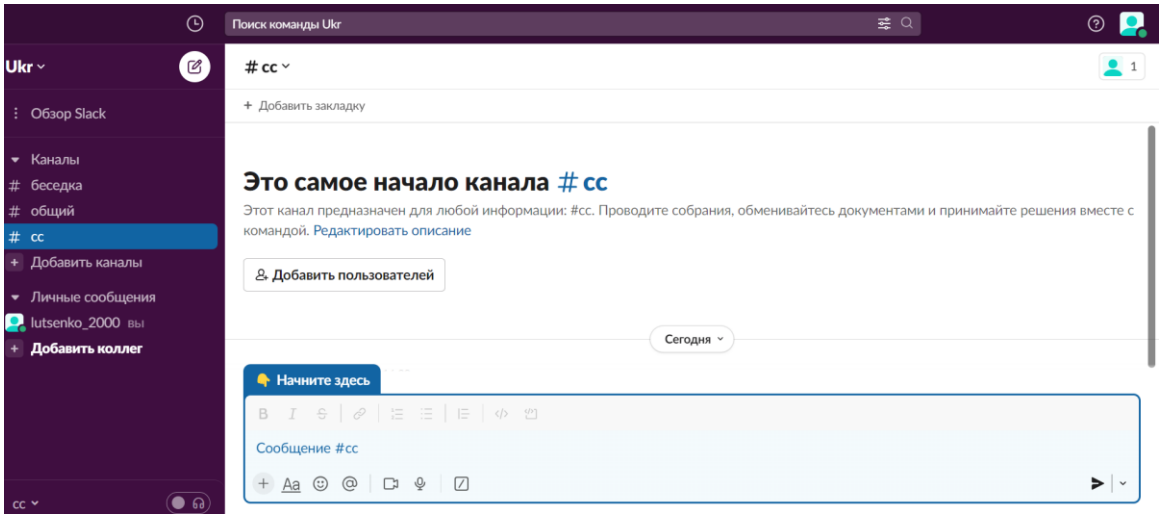


Рисунок 1.2 – Slack.

Переваги використання Slack:

1) можна створювати канали для комунікації з окремими групами співробітників та проектів;

2) дозволяє обмінюватися файлами та документами;

3) комунікація в реальному часі: Slack надає можливість спілкуватися з колегами в реальному часі через основні чати, приватні повідомлення та групові обговорення. Це сприяє швидкій і ефективній комунікації в команді та допомагає вирішувати проблеми миттєво;

4) організація комунікації за темами: Slack дозволяє створювати канали для різних проектів, команд або тем, що дозволяє зберігати комунікацію організованою та легко доступною для всіх учасників. Це полегшує знаходження необхідної інформації та уникнення замішання повідомлень;

5) інтеграція з іншими інструментами: Slack забезпечує можливість інтегрувати різні інструменти, такі як Google Drive, Trello, GitHub та інші. Це дозволяє зручно спільно працювати над проектами, обмінюватися файлами та отримувати сповіщення з різних сервісів у зручному чат-інтерфейсі;

6) архівування та пошук повідомлень: Slack зберігає історію повідомлень, що дозволяє зручно переглядати старі дискусії та повертатися до раніше обговорених питань. Крім того, вбудована система пошуку дозволяє швидко знаходити потрібну інформацію.

Недоліки використання Slack:

1) великий обсяг повідомлень: Завдяки безлічі повідомлень, які зберігаються в Slack, може виникати проблема з переглядом та пошуком потрібної інформації. Це може збільшувати час на пошук та ускладнювати організацію дискусій;

2) залежність від інтернету: для користування Slack необхідний постійний доступ до Інтернету. Це може бути недоліком у випадку відсутності з'єднання або обмеженого доступу до інтернету;

3) витрата часу на відповіді: постійний потік повідомлень може вимагати великої уваги та часу на відповіді. Це може впливати на продуктивність, особливо якщо користувачі постійно переривають свою роботу для відповіді на повідомлення;

4) вартість: Хоча Slack пропонує безкоштовну версію, але для отримання розширених функцій та можливостей, таких як збереження повідомлень на необмежений термін або розширена інтеграція, може бути необхідно підписати платний план;

5) може відволікати співробітників від робочих завдань.

Microsoft Teams - це колабораційна платформа, розроблена компанією Microsoft, яка призначена для комунікації, співпраці та організації робочих процесів у командах і організаціях (рис. 1.3).

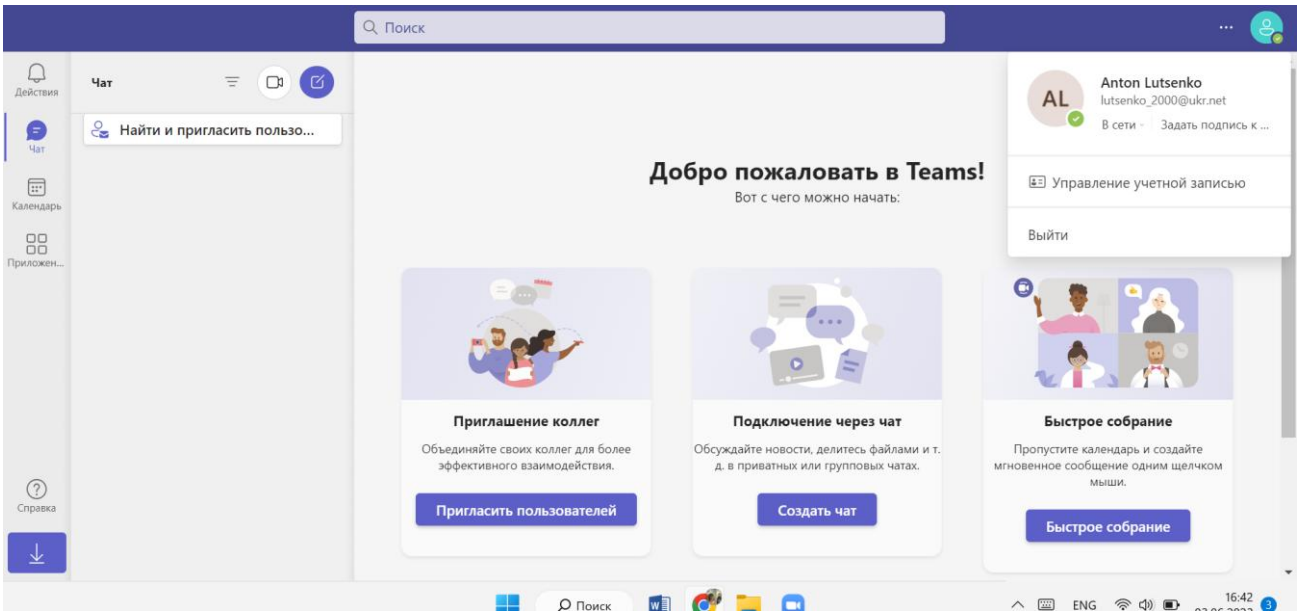


Рисунок 1.3 – Microsoft Teams.

Переваги використання Microsoft Teams:

1) інтеграція зі всесвітньо відомими продуктами Microsoft: Microsoft Teams має глибоку інтеграцію з іншими продуктами Microsoft, такими як Office 365, SharePoint, OneDrive та інші. Це дозволяє зручно спільно працювати над документами, обмінюватися файлами та використовувати різноманітні інструменти для спільного проектування;

2) відеозв'язок та аудіоконференції: Microsoft Teams надає можливість проводити відеозв'язок та аудіоконференції з учасниками з різних місць. Це дозволяє зберігати комунікацію більш особистою та ефективною, зокрема при спілкуванні з віддаленими членами команди або партнерами;

3) широкі можливості комунікації: Microsoft Teams пропонує різні канали комунікації, включаючи чати, групові обговорення та приватні повідомлення. Це дозволяє ефективно організовувати комунікацію в команді, спілкуватися один на один та обговорювати питання відповідно до потреб користувачів;

4) багатофункціональність: Microsoft Teams має широкий набір функціональностей, таких як спільне редагування документів в реальному часі, планування зустрічей, завдань та відстеження прогресу проектів. Це дозволяє зручно керувати проектами та виконувати різноманітні завдання безпосередньо в середовищі Teams.

Недоліки використання Microsoft Teams:

1) складність інтерфейсу: у Microsoft Teams є досить велика кількість функцій та можливостей, що може зробити інтерфейс складним для користувачів, які вперше з ним зустрічаються. Потрібно витратити певний час на навчання та ознайомлення з усіма можливостями платформи;

2) залежність від інтернету: для користування Microsoft Teams потрібне постійне підключення до Інтернету. Це може створювати проблеми у випадку обмеженого або відсутнього зв'язку, що може обмежити доступність платформи в деяких ситуаціях;

3) обмеження безкоштовної версії: хоча Microsoft Teams пропонує безкоштовну версію, але вона має обмеження в розмірі файлів, кількості учасників та функцій. Для отримання повного функціоналу та розширених можливостей, таких як збереження повідомлень на необмежений термін або інтеграція з іншими сервісами, може бути необхідно придбати платний план.

Zoom - це платформа для відеоконференцій та віртуальних зустрічей, яка надає можливість людям спілкуватись та працювати разом, незалежно від їх місцезнаходження (рис. 1.4).

Переваги використання Zoom:

1) простота використання: ZOOM має інтуїтивно зрозумілий і простий інтерфейс, що дозволяє швидко налаштувати та приєднатися до відеоконференції. Навіть незнайомі користувачі з легкістю можуть використовувати цю платформу без значних труднощів;

2) висока якість аудіо та відео: ZOOM надає високу якість передачі аудіо та відео під час відеоконференцій. Завдяки цьому, учасники можуть

насолоджуватися чітким звуком і чітким відеозображенням, що сприяє якісній комунікації;

3) розширені функціональні можливості: ZOOM пропонує багато корисних функцій, таких як можливість демонстрації екрану, запису відеоконференцій, використання віртуальних фонів та інші. Це дозволяє розширити можливості використання платформи і покращити спільну роботу та комунікацію;

4) масштабованість: ZOOM добре підходить для проведення відеоконференцій з великою кількістю учасників. Вона може підтримувати сесії з десятками, сотнями або навіть тисячами учасників одночасно.

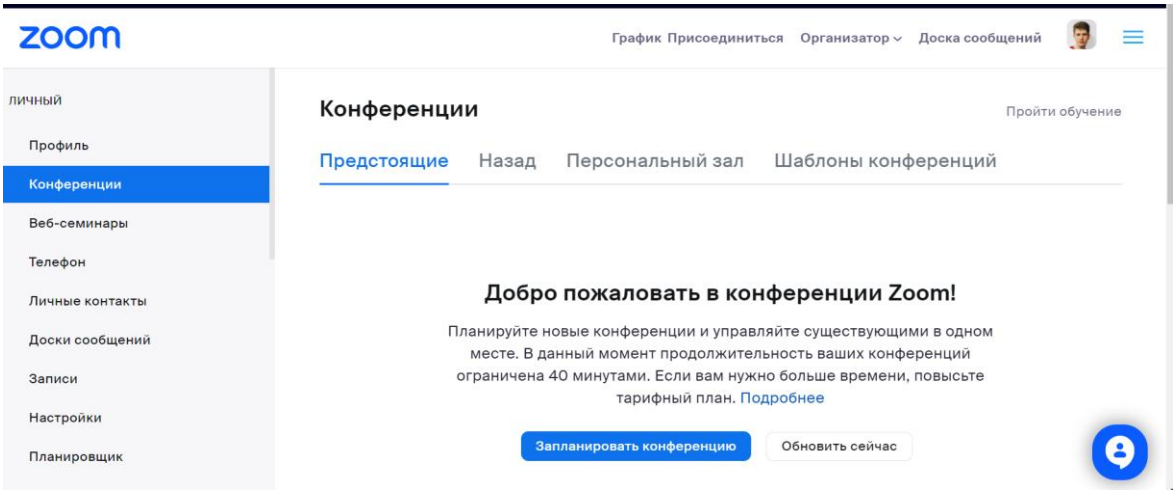


Рисунок 1.4 – Zoom.

Недоліки використання Zoom:

1) не дозволяє комунікувати в режимі реального часу, крім відеоконференцій;

2) обмеження безкоштовного плану: безкоштовна версія ZOOM має певні обмеження, такі як обмежений час тривалості відеоконференцій та обмежена кількість учасників. Для отримання повного функціоналу та розширених можливостей може знадобитися придбання платного плану;

3) проблеми з конфіденційністю даних: у минулому ZOOM мав певні проблеми з конфіденційністю даних, що створило обурення серед користувачів.

Хоча компанія внесла поліпшення у цьому напрямку, важливо бути обережним і приймати належні заходи безпеки під час використання платформи;

4) залежність від інтернету: для користування ZOOM потрібне стабільне підключення до Інтернету. Погане з'єднання може спричинити проблеми з якістю звуку та відео, а також викликати перебої в комунікації;

5) платформозалежність: завантажений на пристрої користувача, ZOOM має платформозалежність, що означає, що його можна використовувати тільки на сумісних пристроях та операційних системах.

Reddit є однією з найпопулярніших соціальних мереж та платформ обміну повідомленнями в Інтернеті. Він створений як форум з можливістю голосування, де користувачі можуть ділитись контентом, створювати пости, коментувати та голосувати за них. Reddit привертає мільйони користувачів з усього світу, які об'єднуються навколо різноманітних тем та спільнот (рис. 1.5).

Переваги використання Reddit:

1) розділення на тематичні підрозділи: Reddit має широкий вибір тематичних розділів, від політики та технологій до спорту та розваг. Це дозволяє користувачам знайти точну спільноту, яка відповідає їхнім інтересам і зацікавленості;

2) зміст згідно голосування: Reddit використовує систему голосування, де користувачі можуть голосувати "за" або "проти" певного вмісту. Це дозволяє кращому контенту отримувати більше визнання та потрапляти на вершину розділу;

3) коментарі та обговорення: Reddit надає можливість користувачам коментувати пости та взаємодіяти між собою. Це створює динамічні обговорення, де користувачі можуть додавати свої думки, задавати питання та отримувати відповіді;

4) редагування та форматування тексту: Reddit надає можливість форматування тексту у постах та коментарях, використовуючи спеціальні теги. Це дозволяє користувачам створювати зрозумілий та організований вміст.

Недоліки використання Reddit:

1) анонімність: Reddit дозволяє користувачам залишати коментарі та взаємодіяти без обов'язкової ідентифікації. Це може призводити до недостатньої відповідальності та сприяти поширенню негативного вмісту, тролінгу та кібербулінгу;

2) якість вмісту: через велику кількість користувачів та вмісту, який розміщується на Reddit, якість вмісту може значно варіюватись. Іноді можуть з'являтись малоякісні або неперевірені повідомлення та пости;

3) модерація: Reddit має систему модерації, але вона залежить від кожного підрозділу. Іноді підрозділи можуть бути недостатньо модерованими, що може призводити до поширення небажаного або образливого вмісту;

4) складність орієнтації: через велику кількість розділів та вмісту на Reddit, новим користувачам може бути складно орієнтуватись та знайти потрібну інформацію.

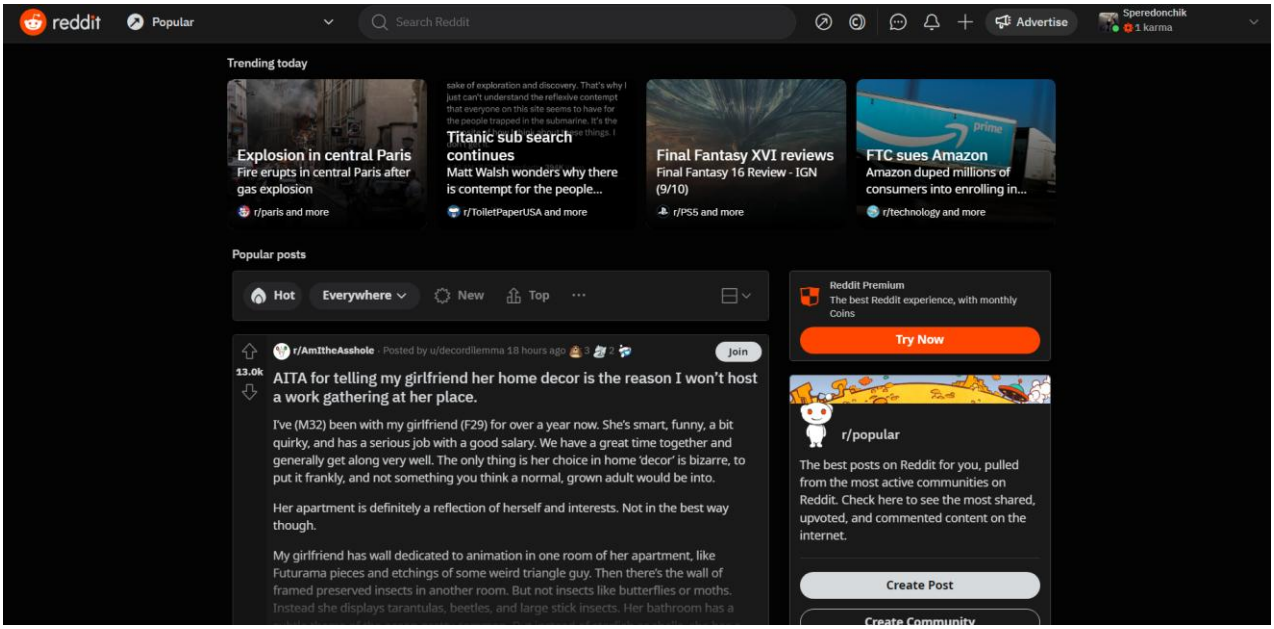


Рисунок 1.5 – Reddit.

Twitter є однією з найпопулярніших мікроблогінгових платформ в світі, яка дозволяє користувачам ділитись короткими повідомленнями, відомими як "твіти" (рис. 1.6).

Він створений як швидкий та простий спосіб обмінюватись інформацією, думками та новинами у режимі реального часу. Користувачі Twitter можуть створювати свій профіль, підписуватись на інших користувачів та читати їх твіти, а також коментувати та ретвітити їх.

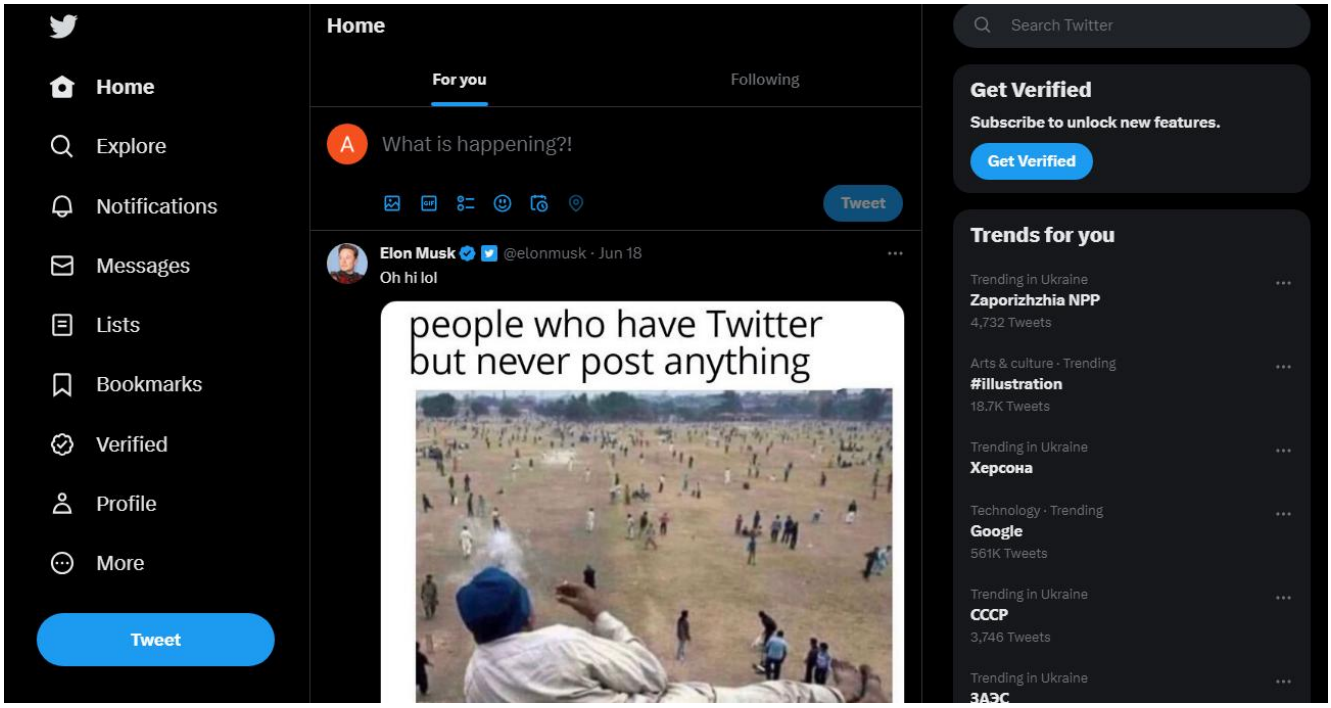


Рисунок 1.6 – Twitter.

Переваги використання Twitter:

1) швидкість та актуальність інформації: Twitter є відмінною платформою для отримання свіжих новин та оновлень у реальному часі. Користувачі можуть швидко дізнаватись про події, тренди та глобальні розмови, що відбуваються в різних сферах життя;

2) широкий охоплення та зв'язок з аудиторією: Twitter має велику глобальну аудиторію, що надає можливість користувачам зв'язатись з різними людьми, включаючи селебритів, впливових осіб, журналістів та експертів у різних галузях;

3) платформа для вираження думок та взаємодії: Twitter дозволяє користувачам висловлювати свої думки, ідеї та погляди у коротких твітах. Він

сприяє активній взаємодії з іншими користувачами шляхом коментування, лайків та ретвітів, що сприяє формуванню обговорень та спільнот.

Недоліки використання Reddit:

1) обмежена кількість символів: одним з недоліків Twitter є обмежена кількість символів у твітах, що може обмежувати глибину висловлення та унеможливлювати детальне пояснення думок;

2) зловживання та негативний контент: оскільки Twitter є відкритою платформою, він також може стикатись з проблемами зловживання, ненависті, булінгу та поширення неправдивої інформації;

3) швидкість інформаційного потоку: у зв'язку з великою кількістю твітів, які генеруються на платформі, інформаційний потік на Twitter може бути дуже швидким та важким для перегляду та сприйняття.

Загалом, кожна інформаційна система має свої переваги та недоліки, тому вибір залежить від потреб організації та її співробітників. Якщо комунікація в режимі реального часу є важливою, Slack або Microsoft Teams можуть бути кращими виборами, якщо ж потрібно багато обмінюватися документами та інтегрувати з іншими інструментами, то Microsoft Teams чи Slack можуть бути кращими варіантами. Для великих відеоконференцій Zoom може бути більш вигідним вибором, а електронна пошта може бути корисною для відправлення великих файлів та повідомлень без обмежень щодо розміру. Важливо також враховувати фінансові можливості та доступність сервісів для співробітників організації.

Зазначені інформаційні системи не є єдиними можливими варіантами, існують інші альтернативи, такі як Discord, Skype, Google Meet та інші, які також мають свої переваги та недоліки. При виборі системи важливо враховувати конкретні потреби та характеристики організації та її співробітників. Також слід пам'ятати, що ефективна комунікація включає не лише використання інформаційних систем, але й правильне використання комунікаційних каналів та культуру спілкування.

1.3 Вибір моделі життєвого циклу, апаратних та програмних ресурсів для реалізації поставленої задачі

Каскадна модель життєвого циклу є однією з найпоширеніших моделей при розробці інформаційних систем. Вона передбачає послідовне виконання фаз, що дозволяє керувати процесом розробки та забезпечити високу якість результуючої системи [5-7].

При розробці інформаційної системи обміну повідомленнями та взаємодії користувачів необхідно вибрати підходящу модель життєвого циклу, яка дозволить ефективно керувати процесом розробки та забезпечити якість результуючої системи. Один з варіантів моделі життєвого циклу, який може бути використаний це каскадна модель (рис. 1.7).

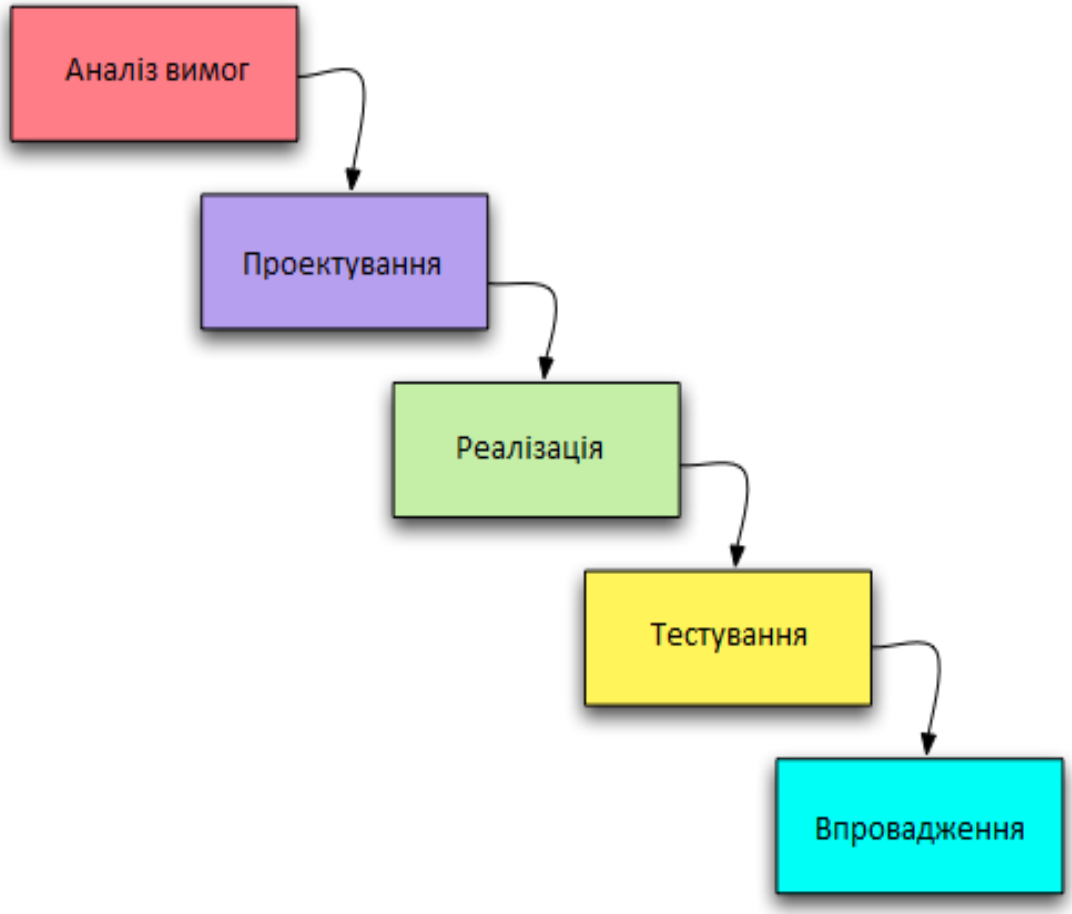


Рисунок 1.7 – Структура каскадної моделі.

Розглянемо кожну фазу каскадної моделі життєвого циклу [8-10]:

1) аналіз вимог: у цій фазі проводиться детальний аналіз вимог до системи. Визначаються функціональні та нефункціональні вимоги, проводиться вивчення потреб користувачів та встановлюються основні цілі та обмеження проекту;

2) проектування: на основі аналізу вимог розробляється архітектура системи та детальний план проекту. В цій фазі визначаються структура бази даних, вибір технологій та інструментів розробки;

3) реалізація: здійснюється реалізація системи на основі розробленого проекту. Код програми пишеться та тестується, база даних створюється та наповнюється даними. В результаті цієї фази отримується працездатна система, яка відповідає вимогам;

4) тестування: проводяться різноманітні тести для перевірки працездатності, відповідності вимогам та виявлення помилок. Тестування включає модульне тестування окремих компонентів, інтеграційне тестування системи в цілому, а також приймальне тестування залежно від вимог замовника;

5) впровадження: після успішного тестування система готова до впровадження. Вона розгортається на продуктивному середовищі та надається користувачам для використання. Здійснюється підготовка користувачів, навчання та підтримка системи.

Каскадна модель життєвого циклу підійде для даної дипломної роботи з причини наступних факторів:

1) послідовність: каскадна модель дозволяє проводити фази розробки послідовно, що полегшує керування проектом та забезпечує чіткість процесу;

2) документація: кожна фаза каскадної моделі вимагає детальної документації, що дозволяє зберігати інформацію та зробити проект зрозумілим для всіх учасників;

3) ризики: каскадна модель дозволяє виявляти ризики та помилки на ранніх етапах розробки, що дозволяє знизити витрати та запобігти подальшим проблемам;

4) якість: послідовне виконання фаз дозволяє забезпечити високу якість системи шляхом аналізу вимог, ретельного проектування та тестування;

5) прозорість: каскадна модель дозволяє зрозуміло відстежувати прогрес проекту, оцінювати витрати та встановлювати реалістичні терміни виконання.

Тому, використання каскадної моделі життєвого циклу є відповідним підходом для розробки інформаційної системи обміну повідомленнями та взаємодії користувачів.

Для запуску готового додатку, необхідно мати ПК з наступними характеристиками:

1) процесор: рекомендовано мати процесор з архітектурою x86 або x64 з тактовою частотою не менше 2.0 ГГц. Це дозволить додатку працювати швидко та ефективно обробляти запити користувачів;

2) оперативна пам'ять: мінімально рекомендовано мати 4 ГБ оперативної пам'яті;

3) жорсткий диск: для установки та роботи додатку потрібно мати достатньо місця на жорсткому диску. Рекомендовано мати щонайменше 20 ГБ вільного простору для установки програмного забезпечення та збереження даних;

4) операційна система: Windows, Linux та macOS.

Атрибути системи:

1) мова реалізації - HTML/CSS, C#;
2) база даних - Entity Framework;
3) групи користувачів - користувач;
4) стиль інтерфейсу - графічний;
5) технологія програмування – об’єктно орієнтоване програмування;
6) захист - надійний.

1.4 Висновки. Постановка задачі

У даному розділі було досліджено інформаційну систему обміну

повідомленнями та взаємодії користувачів. Метою дослідження було визначення основних вимог та потреб користувачів.

Для досягнення цієї мети потрібно буде виконати такі задачі:

1) розробити архітектуру системи;
2) розробити базу даних для зберігання та передачі даних;
3) вивчити підходи до вирішення задачі створення інформаційної системи;
4) розробити інтрерфейс користувача;
5) провести тестування розробленої інформаційної системи.

2 ПРОЕКТУВАННЯ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА ВЗАЄМОДІЇ КОРИСТУВАЧІВ

2.1 Планування виконання проектних робіт

Планування є важливим етапом перед початком проекту розробки інформаційної системи обміну повідомленнями та взаємодії користувачів. Його метою є створення структурованого плану дій, який дозволяє ефективно організувати роботу, заздалегідь визначити послідовність завдань, ресурси та терміни їх виконання.

Перед початком проектування системи обміну повідомленнями та взаємодії користувачів необхідно мати чіткий планувальний процес, який включає наступні кроки:

1) визначення завдань проекту: розкрити постановку задачі та цілі вашого проекту. Визначити обсяг робіт, результати, які потрібно досягти, та вимоги до системи;

2) аналіз вимог: ретельно проаналізувати та документувати вимоги до системи, включаючи функціональні, нефункціональні, технічні та дизайнерські вимоги. Уточнити, як система повинна взаємодіяти з користувачами та іншими системами;

3) розподіл завдань та ресурсів: визначити ролі та відповідальності учасників команди проекту. призначити відповідальних за кожне завдання та розподіліть ресурси, такі як робочий час, фінансові засоби та обладнання;

4) розробка графіка робіт: створити детальний графік робіт, в якому вказані терміни та послідовність виконання кожного завдання. Врахувати залежності між завданнями та можливі ризики;

5) управління ризиками: визначити потенційні ризики, які можуть виникнути під час проекту. Розробити стратегії мінімізації та управління ризиками. Створити план дій для кожного ідентифікованого ризику;

6) визначення бюджету: оцінити фінансові витрати на проект, включаючи ресурси, обладнання, програмне забезпечення та інші витрати. Визначити допустимий бюджет та розподіліть кошти між різними етапами проекту;

7) комунікація та звітність: встановити систему комунікації в команді проекту, включаючи зустрічі, звіти та засоби спілкування. Забезпечити своєчасний обмін інформацією та звітами про прогрес проекту;

8) етап контролю та оцінки: визначити механізми контролю та оцінки прогресу проекту. Періодично перевіряти виконання завдань, виконувати контрольні точки та аналізувати прогрес проекту;

9) управління змінами: розробити процес управління змінами, включаючи оцінку впливу змін на проект та процедури затвердження змін. Забезпечити систематичне оновлення та документацію змін проекту;

10) завершення проекту: провести фінальну оцінку проекту та підготувати звіт про проект. Визначити, які кроки необхідні для впровадження системи в експлуатацію та планування подальших кроків.

Ризик є невід'ємною частиною будь-якого проекту чи діяльності. Він вказує на можливість небажаних подій, які можуть вплинути на досягнення поставлених цілей та результативність проекту. Розуміння ризиків та їх належне управління є ключовими аспектами успішного виконання будь-якого проекту.

Ризик може виникнути з різних джерел, включаючи технічні проблеми, людські помилки, непередбачувані зміни у вимогах або зовнішні впливи. Він може мати негативний вплив на графік виконання проекту, бюджет, якість робіт та задоволення замовника.

Ефективне управління ризиками включає в себе ідентифікацію потенційних ризиків, оцінку їх імовірності виникнення та впливу, розробку стратегій усунення або зменшення ризиків, а також моніторинг та контроль над ними протягом усього проекту.

Успішне впровадження системи залежить від управління ризиками. Тому в рамках дослідження потрібно аналізувати та керувати ризиками, які можуть виникнути під час розробки та експлуатації цієї інформаційної системи.

Розуміння ризиків та вжиття відповідних заходів для їх управління є важливою складовою успіху проекту. Продовжуючи далі, розглянемо конкретні ризики та запропонуємо стратегії їх усунення, зосереджуючись на розробці та експлуатації інформаційної системи обміну повідомленнями та взаємодії користувачів (табл. 2.1).

Таблиця 2.1 – Ризики проекту та стратегії їх усунення

№	Ризик	Імовірність виникнення	Стратегія усунення	Імовірність після стратегії
1	Втрата даних	Висока	Регулярне резервування даних	Середня
2	Технічний збій	Середня	Регулярне обслуговування обладнання	Низька
3	Відсутність інтернет-з'єднання	Висока	Резервування доступних інтернет-каналів	Середня
4	Несанкціонований доступ до системи	Низька	Впровадження міцних механізмів аутентифікації та авторизації	Дуже низька
5	Втрата електропостачання	Середня	Установка додаткових джерел живлення, наприклад, резервних батарей	Низька

Ця таблиця містить перелік ризиків, їх імовірність виникнення, запропоновану стратегію усунення та імовірність виникнення після застосування

стратегії. Кожен ризик має свій унікальний номер для зручності ідентифікації та відстеження.

Ця таблиця допоможе своєчасно виявляти потенційні ризики, розробляти плани їх усунення та контролювати їх вплив на проект. Інформація про імовірність виникнення ризиків дозволяє приділити більше уваги найбільш критичним аспектам і забезпечити відповідні заходи для зменшення ризиків та забезпечення успішного завершення проекту.

2.2 Формування вимог до інформаційної системи

Формування вимог до інформаційної системи є важливим етапом проектування та розробки. Правильно сформульовані вимоги гарантують, що система буде відповідати потребам та очікуванням користувачів. Нижче наведено деякі можливі формування вимог до інформаційної системи [11-15]:

1) функціональні вимоги: це вимоги, що визначають, які функції повинна виконувати система. Наприклад, створення облікових записів користувачів, можливість створення та редагування повідомлень, взаємодія між користувачами через систему обміну повідомленнями;

2) нефункціональні вимоги: це вимоги, які не визначають конкретні функції системи, але стосуються її характеристик і якості. Наприклад, швидкість та продуктивність системи, безпека та конфіденційність інформації, зручність інтерфейсу користувача та інші аспекти, які впливають на задоволення потреб користувачів.

Функціональні вимоги до даного проекту:

1) реєстрація:

a) створити форму з п’ятьма полями для введення користувачем імені, фамілії, пошти, пароля та повторного пароля для ідентифікації у системі;

b) після здійснення запиту дані відправляються на сервер, де виконується перевірка коректності даних;

c) далі відбувається аналіз повідомлення із серверу – якщо користувача зареєстровано, відбувається подальша робота, якщо помилка – відбувається її аналіз;

d) якщо у БД не знайдено пошти – система виводить повідомлення, що не існує користувача з такою поштою.

2) авторизація:

a) створити форму з двома полями для введення користувачем пошти та пароля для ідентифікації у системі;

b) після запиту для входу в систему відбувається перевірка коректності даних;

c) після відправлення даних на сервер формується jwt токен – ідентифікатор окремого користувача для доступу до його даних;

d) якщо операція пройшла успішно – завантажити дані із віддаленої бази даних та надати права на користування сервісом.

3) додавання постів:

a) додати елемент інтерфейсу для додавання поста;

b) при додаванні надати можливість вказати ім’я поста, опис та картинок;

c) після додавання поста, сам пост повинен з’явитись на сторінці користувача з можливістю проведення операцій, редагування та видалення.

4) редагування постів:

a) створити форму для редагування поста;

b) створити кнопку для виклику форми;

c) після введення даних відправляється запит на сервер та повертається повідомлення із результатами виконання.

5) видалення постів:

a) додати кнопку для видалення постів;

b) після підтвердження відправляється запит на сервер та повертається повідомлення із результатами виконання.

6) створення повідомлень:

a) створити форму для створення повідомлень; вона повинна містити поле для введення логіна користувача, кому буде відправлене повідомлення та поле для самого тексту повідомлення;

b) створити кнопку для виклику форми;

c) після введення даних відправляється запит на сервер та повертається повідомлення із результатами виконання.

7) синхронізація даних із сервером:

a) дані для користувача повинні завантажуватись з сервера;

b) зміни повинні синхронізувати із сервером;

c) повідомлення про статус виконання операції на сервері повинне бути опрацьованим.

8) робота програмного забезпечення у форматі веб-сторінки:

a) за допомогою використання веб-сторінки у браузері повинна бути можливість для використання сервісу;

b) дані повинні бути синхронізовані із віддаленим сервером;

c) інтерфейс повинен бути зрозумілим та лаконічним.

Нефункціональні вимоги:

1) продуктивність: система повинна забезпечувати швидку та ефективну обробку повідомлень та діалогів між користувачами. Застосування оптимізаційних стратегій, кешування даних та оптимального використання ресурсів можуть сприяти досягненню високої продуктивності системи;

2) надійність: система повинна бути стійкою та надійною. Це означає, що вона має працювати без збоїв або відмов, витримувати навантаження та забезпечувати безперебійний доступ до повідомлень та функцій системи. Застосування механізмів резервування, реплікації даних та контролю доступу можуть сприяти надійності системи;

3) безпека: забезпечення безпеки інформації є важливим аспектом проекту. Система повинна мати механізми аутентифікації, авторизації та шифрування, щоб забезпечити конфіденційність та цілісність даних. Також варто враховувати заходи захисту від зламу, вторгнень та інших загроз безпеці;

4) зручність використання: система повинна мати зручний та інтуїтивно зрозумілий інтерфейс користувача, що сприяє легкому навчанню та швидкому освоєнню функціоналу. Налаштування персональних налаштувань, підказки та документація можуть покращити зручність використання системи;

5) сумісність: система повинна бути сумісною з різними платформами та пристроями. Вона повинна працювати на різних операційних системах (наприклад, Windows, macOS, Linux) та підтримувати різні браузери. Застосування стандартів розробки та використання відкритих протоколів можуть сприяти досягненню сумісності;

6) масштабованість: система повинна бути гнучкою та масштабованою, щоб відповідати зростаючим потребам користувачів. Вона повинна забезпечувати можливість додавання нових функцій, розширення обсягу даних та забезпечення високої доступності, навіть при збільшенні навантаження;

Верифікація вимог - це процес перевірки та підтвердження, що вимоги до інформаційної системи обміну повідомленнями та взаємодії користувачів виконуються правильно і повністю. Цей процес гарантує, що система відповідає потребам та очікуванням користувачів, а також відповідає вимогам безпеки, надійності та продуктивності [16-17].

Для проведення верифікації вимог можуть застосовуватися різні методи та техніки. Основні кроки верифікації включають наступне:

1) аналіз вимог: вимоги до системи повинні бути ретельно проаналізовані та зрозумілі для всіх зацікавлених сторін. Важливо переконатися, що вимоги коректні, повні та узгоджені;

2) перевірка вимог: перевірка вимог полягає у визначенні того, чи відповідає система вимогам, які були визначені на етапі аналізу. Це може включати перевірку коректності синтаксису, зрозумілості та консистентності вимог;

3) валідація вимог: валідація вимог - це процес підтвердження, що система відповідає реальним потребам та очікуванням користувачів. Це може включати

залучення користувачів до тестування та оцінки системи, проведення презентацій та демонстрацій функціоналу;

4) тестування: тестування допомагає перевірити, чи відповідає система вимогам, чи працює вона правильно та ефективно. Це може включати функціональне тестування, навантажувальне тестування, тестування безпеки та інші види тестування, щоб переконатися, що система працює належним чином у різних умовах;

5) перевірка відповідності: після завершення верифікації вимог проводиться перевірка відповідності - переконання, що система відповідає всім вимогам, які були визначені на етапі аналізу. Це може включати порівняння реальної функціональності та характеристик системи з вимогами документації.

Верифікація вимог - важлива складова частина процесу розробки інформаційної системи. Вона допомагає забезпечити якість та правильність реалізації системи, зменшує ризики помилок та невідповідності, а також сприяє задоволенню потреб та очікувань користувачів.

2.3 Проектування моделі інформаційної системи

Потрібно розробити модель інформаційної системи [18-21].

Модель - це математична модель, яка буде будуватись з точки зору клієнтського додатку (рис. 2.1).

Коли користувач заходить в додаток, йому потрібно пройти авторизацію. Якщо він не зареєстрований, тоді він вибирає переходить на сторінку реєстрації.

Як тільки він увійде на сайт, в нього буде можливість створити пост або написати повідомлення користувачу за його поштою. Він також може редагувати пости та видаляти їх.

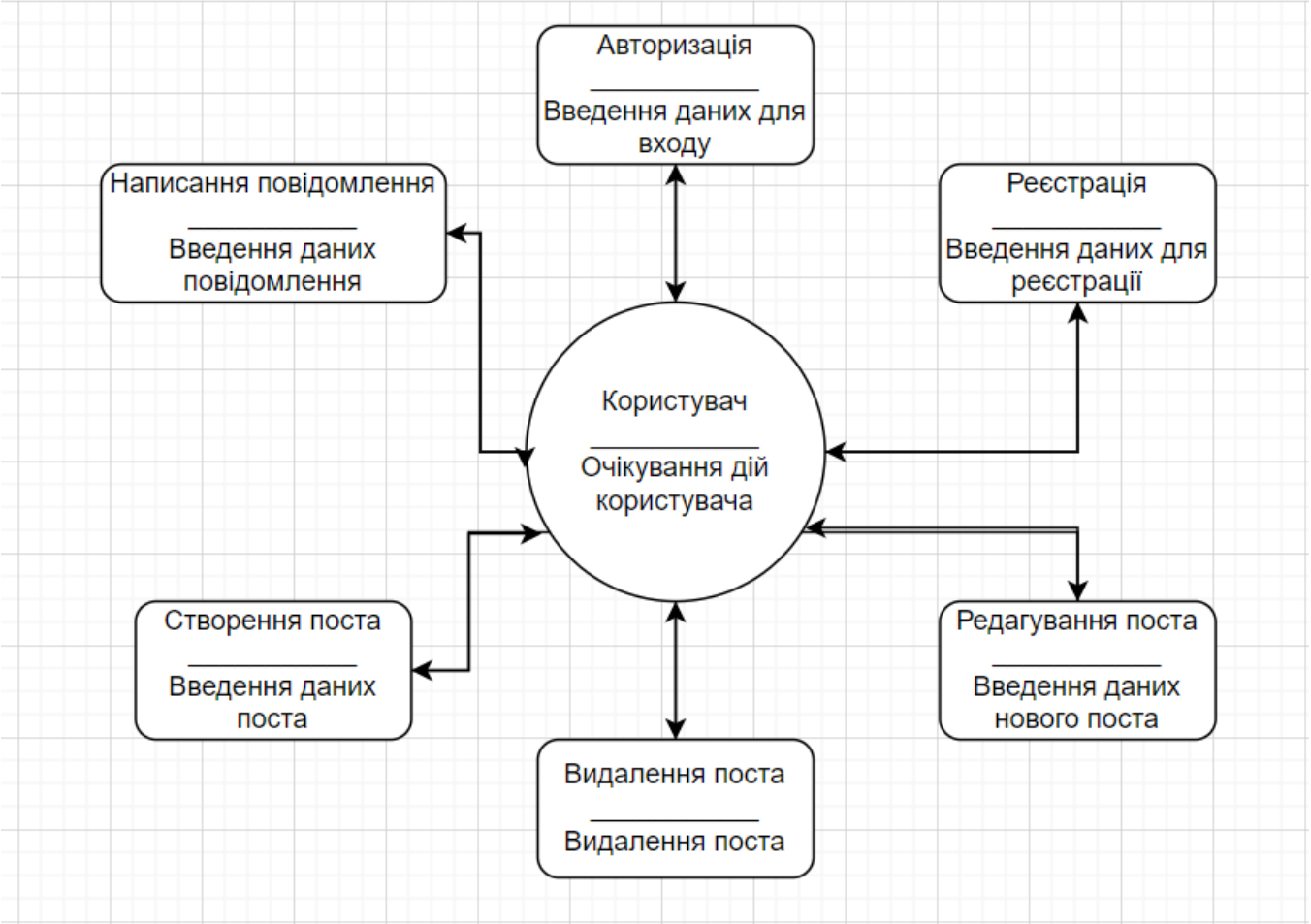


Рисунок 2.1 –Модель клієнтського додатку інформаційної системи.

Розробим декілька моделей за допомогою UML. Мова UML (Unified Modeling Language) - це стандартизована мова для візуалізації, специфікації, конструювання та документування програмних систем. Вона використовується для моделювання структури, поведінки, взаємодії та архітектури системи, незалежно від її масштабу та складності [18].

UML дозволяє створювати графічні моделі, які ілюструють різні аспекти системи, їх зв'язки та взаємодію. Мова UML є універсальною, тому її можна використовувати для будь-яких типів проектів, включаючи розробку програмного забезпечення, інформаційні системи, веб-додатки та інші [19-23].

Всього є 8 UML-діаграм:

1) діаграма варіантів використання (use case diagram);
2) діаграма класів (class diagram);
3) діаграма станів (statechart diagram);
4) діаграма діяльності (activity diagram);

5) діаграма послідовності (sequence diagram);
6) діаграма кооперації (collaboration diagram);
7) діаграму компонентів (component diagram);
8) діаграму розгортання (deployment diagram).

Діаграма варіантів використання необхідна, щоб продемонструвати основні можливості і сценарії використання інформаційної системи користувачем (рис. 2.2).

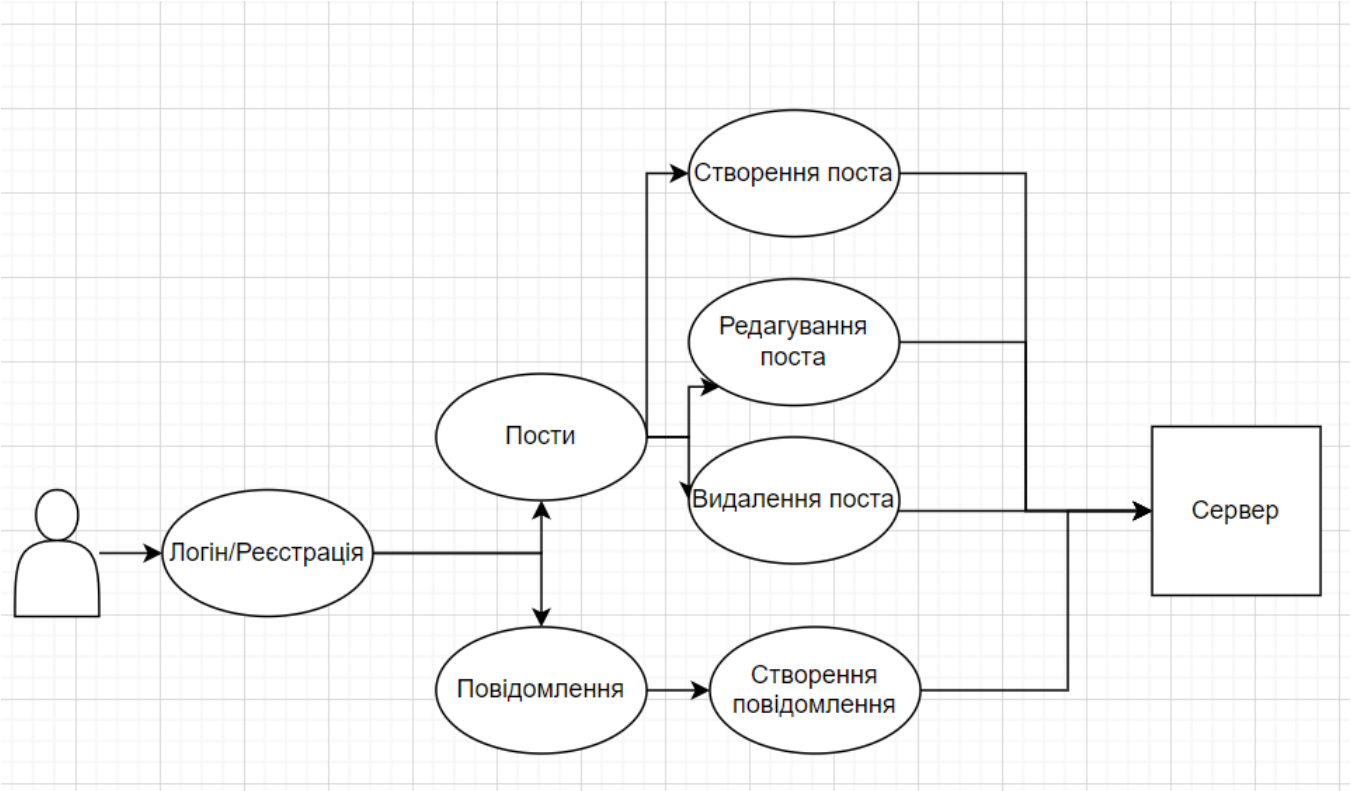


Рисунок 2.2 – Діаграма варіантів використання.

Діаграма класів (Class Diagram) - це структурна діаграма, що відображає основні класи, їх атрибути та взаємозв'язки між класами в системі. Вона дозволяє моделювати структуру системи, ідентифікувати класи, їх властивості та залежності, а також визначати спадкування та асоціації між класами [24-27].

Охарактеризуєм основні елементи бізнес-логіки та організації даних додатку:

1) користувач (User);

2) пост (Post);

3) повідомлення (Message).

Опишемо класи (рис. 2.3-2.5).

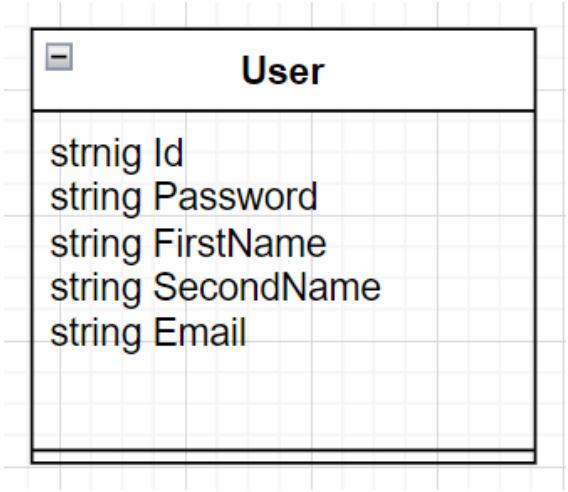


Рисунок 2.3 – Клас User.

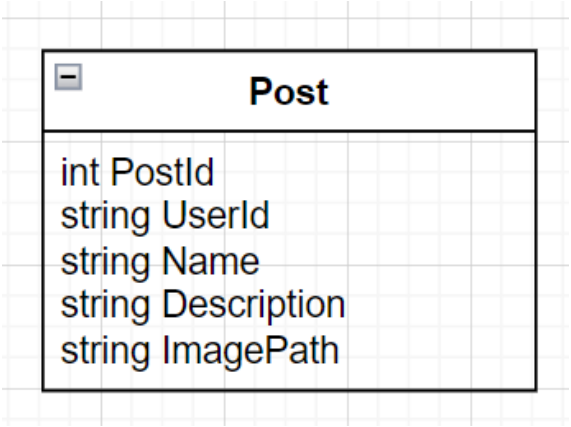


Рисунок 2.4 – Клас Post.

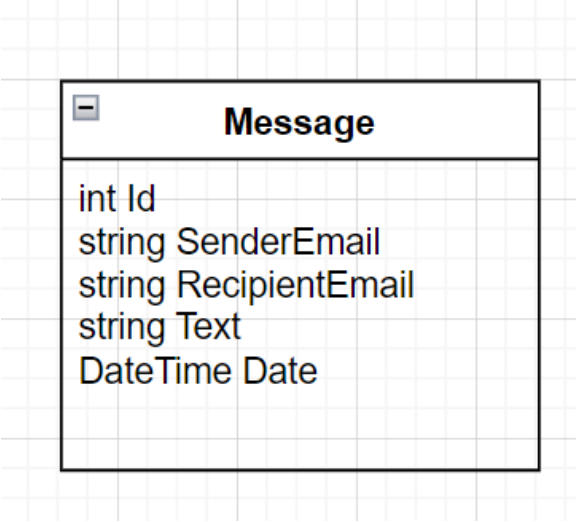


Рисунок 2.5 – Клас Message.

Тепер потрібно зв’язати ці класи (рис 2.6).

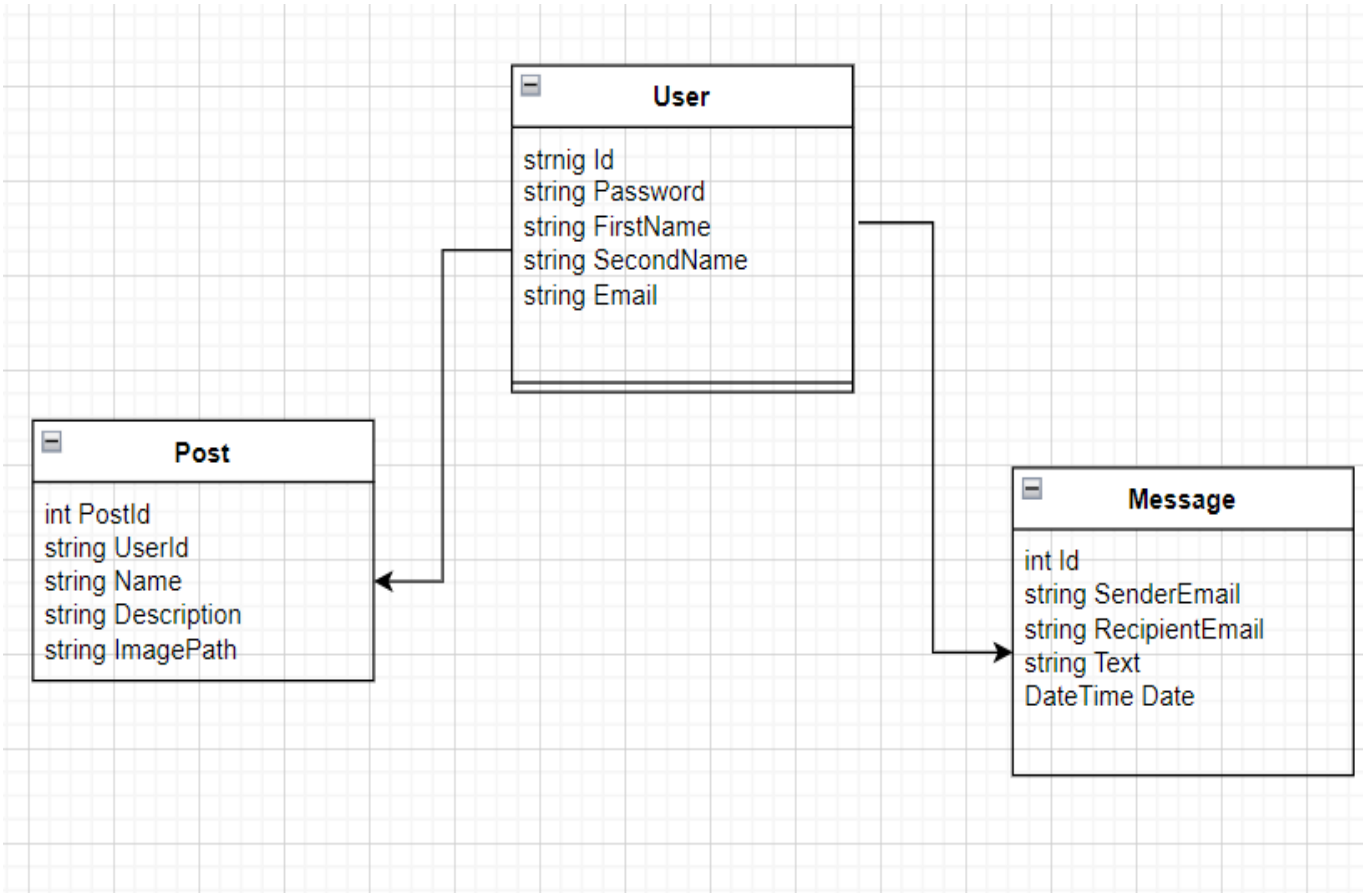


Рисунок 2.6 – Діаграма класів.

Основним класом системи є клас User, який містить дані для входу користувача в систему, а також дані, які користувач вніс в систему.

Розпишемо діаграму станів.

Діаграма станів матиме такі елементи:

1) стан "Незареєстрований користувач": цей стан відображає початковий стан користувача, який ще не зареєстрований в системі. У цьому стані користувач може бачити сторінку логіна або авторизації;

2) стан "Зареєстрований користувач": цей стан відображає стан користувача, який успішно зареєстрований в системі. В цьому стані користувач може створювати свої повідомлення, редагувати свої пости та спілкуватись з іншими користувачами через обмін повідомленнями;

3) стан "Створення повідомлення": цей стан відображає стан користувача, коли він створює нове повідомлення. У цьому стані користувач може вказати отримувачів та текст повідомлення;

4) стан "Перегляд повідомлень": цей стан відображає стан користувача, коли він переглядає список отриманих повідомлень. У цьому стані користувач може відкривати конкретні повідомлення для перегляду деталей;

5) стан "Створення поста": цей стан відображає стан користувача, коли він створює пост. У цьому стані користувач може вказати заголовок, текст та картинку;

6) стан "Редагування поста": цей стан відображає стан користувача, коли він редагує певний пост. У цьому стані користувач може змінити заголовок, текст та картинку;

7) стан "Видалення поста": цей стан відображає стан користувача, коли він видаляє певний пост. У цьому стані користувач може видалити певний пост, який він забажає.

Діаграма діяльності (Activity Diagram) - це тип діаграми, який моделює послідовність дій або активностей, які відбуваються в системі або процесі. Вона демонструє потік управління в системі, розгортання активностей та залежності між ними.

Діаграма діяльності матиме такі пункти:

1) початок: діаграма починається з активності, що позначає початок процесу, наприклад, "Вхід в систему" або "Створення нового повідомлення";

2) активності: кожна активність представляє дію або послідовність дій, які відбуваються в системі. Активності можуть мати залежності між собою, що вказують на порядок виконання;

3) рішення: у діаграмі можуть бути введені рішення або умови, які впливають на подальші активності. Наприклад, "Якщо користувач вибирає редагування повідомлення, перейти до активності 'Редагування повідомлення'";

4) розгалуження: розгалуження показує альтернативні шляхи або варіанти виконання. Наприклад, "Якщо користувач обирає публікацію повідомлення, перейти до активності 'Публікація повідомлення'";

5) завершення: діаграма закінчується активністю, що позначає кінець процесу, наприклад, "Вихід з системи" або "Надіслання повідомлення".

Діаграма кооперації (Collaboration Diagram) - це тип діаграми, який відображає взаємодію між об'єктами в рамках певної функціональної сценарію або процесу. Вона дозволяє візуалізувати, як об'єкти спілкуються один з одним, обмінюючи повідомлення та передаючи контроль [28-29].

Діаграма кооперації матиме такі пункти:

1) об'єкти:

a) користувачі: представлені окремими об'єктами, що взаємодіють з системою;

b) пости: об'єкт, що представляє окремий пост у системі;

c) повідомлення: об'єкт, який представляє окреме повідомлення в системі;

d) система: об'єкт, що представляє інформаційну систему обміну повідомленнями.

2) взаємодії:

a) користувачі можуть створювати нові повідомлення і надсилати їх до інших користувачів або публікувати пости;

a) користувачі можуть переглядати повідомлення;

b) блог постів може містити багато постів, які можуть бути відсортовані за датою або іншими параметрами;

c) система відповідає за обробку та передачу повідомлень між користувачами.

3) повідомлення:

a) повідомлення можуть містити текстовий контент, зображення або інші мультимедійні елементи;

b) користувачі можуть отримувати сповіщення про нові повідомлення.

Діаграма компонентів – це діаграма, який демонструє взаємодію між компонентами системи, їх структуру та залежності. Компоненти представляють собою виконувані частини програмного забезпечення або фізичні елементи, які взаємодіють між собою для виконання певних функцій [30].

Створимо Діаграму компонентів (рис 2.7).

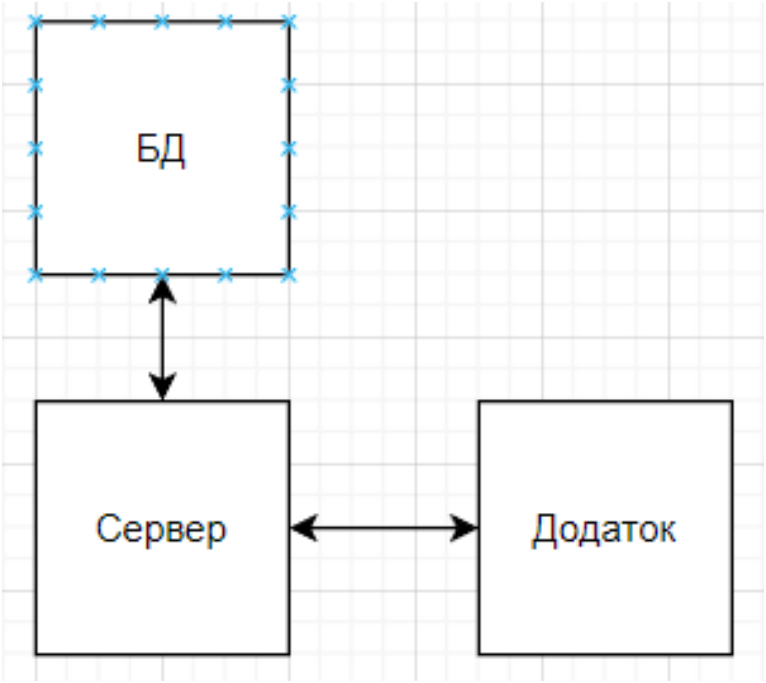


Рисунок 2.7 – Діаграма компонентів.

Тепер створимо діаграму послідовності. Вона дозволяє візуалізувати послідовність взаємодій між об'єктами в системі, показуючи порядок виконання повідомлень та взаємодію між ними.

Ця діаграма допомагає краще зрозуміти, як об'єкти взаємодіють між собою та які дії вони виконують у певному контексті (рис. 2.8).

Діаграма послідовності допомагає уточнити взаємодії та повідомлення між об'єктами системи, виявити можливі проблеми або недоліки в архітектурі системи та допомагає краще зрозуміти, як система повинна функціонувати та як її реалізувати [31].

У результатах аналізу та використання діаграми послідовності, можна зробити висновки про послідовність дій, залежності між об'єктами та можливі шляхи оптимізації процесів взаємодії.

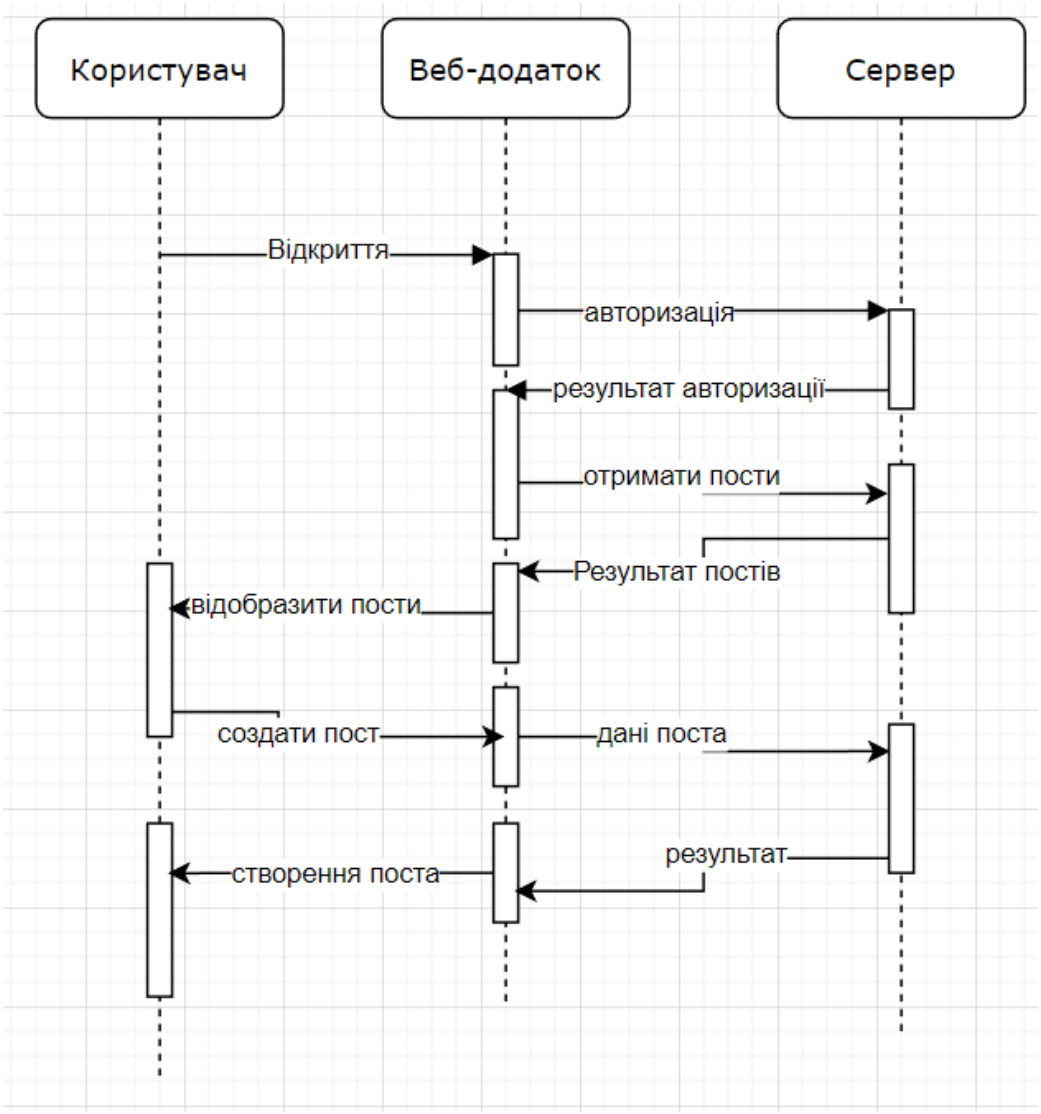


Рисунок 2.8 – Діаграма послідовності

Діаграма розгортання системи є графічним зображенням, яке демонструє фізичну архітектуру системи, розташування її компонентів та зв'язки між ними. Вона використовується для візуалізації розташування апаратних та програмних компонентів системи на фізичних серверах, мережевих вузлах або хмарних сервісах [32-33].

Діаграма розгортання системи матиме таке представлення:

1) користувач взаємодіє із системою використовуючи користувацький інтрефейс;

2) клієнт відправляє запити та отримує відповідь від web-сервера;

3) сервер надає користувацький інтерфейс, а також використовує інтерфейс бази даних для доступу до даних, їх зміни, видалення, редагування.

Дані користувача додатку будуть зберігатись в реляційній базі даних, можна на основі вже описаної діаграми класів визначити сутності, які будуть зберігатись в цій базі даних.

Тепер побудуємо інфологічну модель бази даних у вигляді ER-діаграми(рис. 2.9).

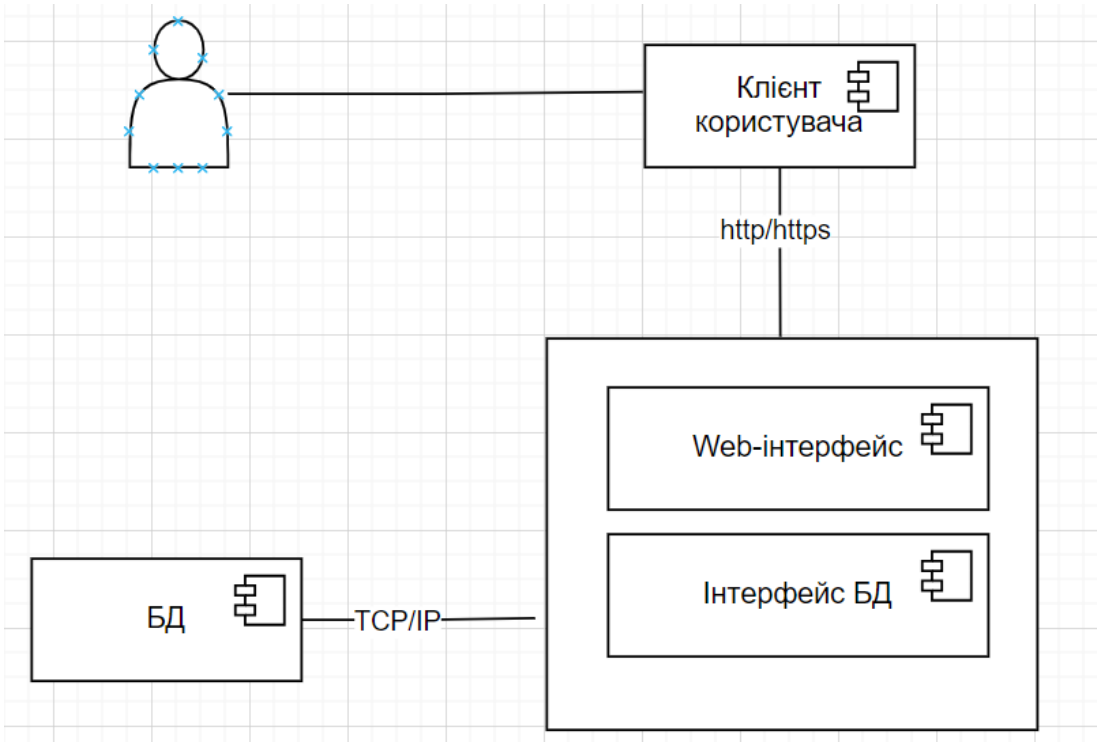


Рисунок 2.9 – UML-діаграма розгортання.

2.4 Висновки

В другому розділі кваліфікаційної роботи спроектовано архітектуру інформаційної системи обміну повідомленнями та взаємодії користувачів, а саме:

1) розроблено планування виконання проектних робіт;

2) проаналізовано ризики проекту;
3) сформульовано вимоги до інформаційної системи;
4) розроблено модель інформаційної системи;
5) розроблено UML-діаграми інформаційної системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА ВЗАЄМОДІЇ КОРИСТУВАЧІВ

3.1 Реалізація інформаційної системи обміну повідомленнями та взаємодії користувачів

Спочатку потрпібно створити проект в Visual Studio. Для реалізації інформаційної системи підйде .NET Core API (рис 3.1).

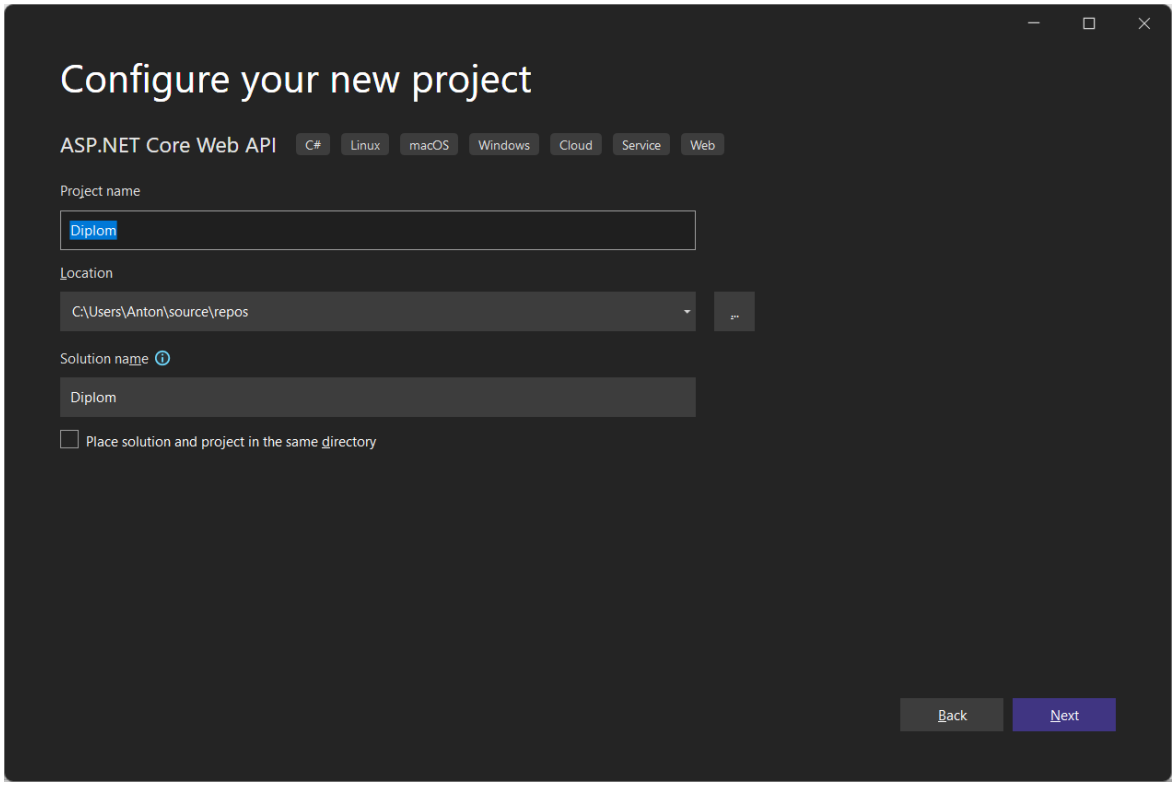


Рисунок 3.1 – Створення проекту.

Після створення проекту потрібно налаштувати базу даних. Для реалізації БД використаєм Entity Framework.

Для реалізації .NET Core API з використанням Entity Framework Core потрібно виконати кілька кроків:

1) встановити необхідні пакети;

2) створити клас моделі: Описати моделі даних, які будуть відображати таблиці в базі даних. Маємо такі класи User, Post та Message, що відповідають таблицям users, posts та messages відповідно (рис. 3.2-3.4);

```
Ссылок: 25
public class User : IdentityUser
{
    Ссылок: 0
    public string Password { get; set; }
    ссылка: 1
    public string FirstName { get; set; }
    ссылка: 1
    public string SecondName { get; set; }
```

Рисунок 3.2 – Клас User.

```
public class Post
{
    Ссылок: 3
    public int PostId { get; set; }
    Ссылок: 4
    public string UserId { get; set; }
    Ссылок: 2
    public string Name { get; set; }
    Ссылок: 2
    public string Description { get; set; }
    Ссылок: 2
    public string ImagePath { get; set; }
```

Рисунок 3.3 – Клас Post.

```
public class Message
{
    Ссылок: 0
    public int Id { get; set; }
    Ссылок: 4
    public string SenderEmail { get; set; }
    Ссылок: 6
    public string RecipientEmail { get; set; }
    Ссылок: 4
    public string Text { get; set; }
    Ссылок: 2
    public DateTime Date { get; set; }
}
```

Рисунок 3.4 – Клас Message.

3) створити контекст бази даних: Створити похідний клас від DbContext, який буде представляти контекст бази даних та містити набори даних (DbSet) для ваших моделей. Клас ApplicationContext, який містить властивості DbSet<User>, DbSet<Post> та DbSet<Message> (рис. 3.5);

```
public class ApplicationContext : DbContext
{
    Ссылок: 3
    public DbSet<User> Users { get; set; }
    Ссылок: 6
    public DbSet<Post> Posts { get; set; }
    Ссылок: 2
    public DbSet<Message> Messages { get; set; }

    Ссылок: 0
    public ApplicationContext(DbContextOptions<ApplicationContext> options)
        : base(options)
    {
        Database.EnsureCreated();
    }
```

Рисунок 3.5 – Клас ApplicationContext.

4) налаштувати підключення до бази даних: В файлі appsettings.json вказати рядок підключення до БД, включаючи інформацію про сервер, ім'я бази даних, облікові дані (рис. 3.6).

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=Diplom8;Trusted_Connection=True;"
},
```

Рисунок 3.6 – Рядок підключення до БД.

Після створення БД можна створювати нових користувачів, постів, повідомлень. Для цього в .NET CORE є контролери (рис. 3.7).

Для кожної моделі даних був створений свій контролер. Контролер відповідає за бізнес-логіку та реалізовує запити до сервера, які робляться клієнтами. Кожен запит являє собою метод контролера з визначеним типом HTTP- запиту, до якого є свій URL-шлях, вхідні дані, та повернення певного

результату (рис. 3.8).

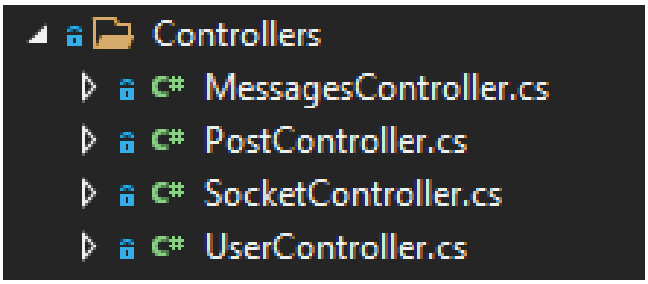


Рисунок 3.7 – Область контролерів.

```
[Route("api/[controller]")]
[ApiController]
ссылка: 1
public class UserController : ControllerBase
{
    private readonly UserService _userService;
    private readonly ApplicationContext _dbContext;
    private readonly UserManager<User> _userManager;
    private readonly ITokenService _tokenService;

    Ссылок: 0
    public UserController(UserService userService, ApplicationContext dbContext, UserManager<User> userManager, ITokenService tokenService)
    {
        _userService = userService;
        _dbContext = dbContext;
        _tokenService = tokenService;
        _userManager = userManager;
    }

    [HttpPost("Register")]
    Ссылок: 0
    public async Task<IActionResult> RegisterUser(RegisterUserModel model)
    {
        User user = new User
        {
            Email = model.Email,
            UserName = model.Email,
            FirstName = model.FirstName,
            SecondName = model.SecondName
        };

        if (model.Password != model.RepeatPassword)
            return BadRequest("Error");

        var result = await _userManager.CreateAsync(user, model.Password);
        if (!result.Succeeded)
        {
            return BadRequest(result.Errors);
```

Рисунок 3.8 – Приклад контролера UserController.

Кожен контролер має свої функції. В таблиці 3.1 відображено функції UserController.

Таблиця 3.1 - Опис функцій UserController

№ п/п	Назва функції	Опис функції
1	RegisterUser	Зареєструвати користувача
2	GetAllUsers	Отримати всіх користувачів
3	Login	Увійти в систему

Також розглянемо інші контролери, а саме PostController (табл. 3.2) та Socket Controller (табл. 3.3).

Таблиця 3.2 - Опис функцій PostController

№ п/п	Назва функції	Опис функції
1	GetAllPosts	Отримати всі пости
2	UpdatePost	Відреагувати пост
3	DeletePost	Видалити пост
4	GetPostById	Отримати пост за його ідентифікатором
5	CreatePost	Створити пост

Таблиця 3.3 - Опис функцій SocketController

№ п/п	Назва функції	Опис функції
1	GetAllMessages	Отримати всі повідомлення
2	SendMessage	Вілправити повідомлення
3	StartWebSocket	Піжключитись до веб сокету

Для забезпечення безпеки та автентифікації в системі обміну повідомленнями та взаємодії користувачів реалізуємо JWT token(рис. 3.9).

```
public string BuildToken(User user)
{
    Claim[] claims = new[] {
        new Claim(ClaimTypes.Name, user.UserName),
        new Claim(ClaimTypes.NameIdentifier,
        Guid.NewGuid().ToString())
    };

    string key = _config["Jwt:Key"].ToString();
    string issuer = _config["Jwt:Issuer"].ToString();

    TimeSpan expiryDuration = TimeSpan.FromHours(1);

    SymmetricSecurityKey securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
    SigningCredentials credentials = new SigningCredentials(securityKey, SecurityAlgorithms.H
    //JwtSecurityToken tokenDescriptor = new JwtSecurityToken(issuer, issuer, claims,
    //    expires: DateTime.Now.AddMinutes(EXPIRY_DURATION_MINUTES), signingCredentials: cred
    JwtSecurityToken tokenDescriptor = new JwtSecurityToken(issuer, issuer, claims,
        expires: DateTime.Now.Add(expiryDuration), signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(tokenDescriptor);
}
```

Рисунок 3.9 – Генерація токена.

Основні елементи коду для створення JWT токену:

1) визначаються клейми (claims) для токену. В даному випадку, використовуються клейми ClaimTypes.Name для імені користувача та ClaimTypes.NameIdentifier для унікального ідентифікатора;

2) отримуються значення ключа (key) та випускача (issuer) з конфігураційного файлу;

3) визначається тривалість дії токену (expiryDuration), у даному випадку 1 година;

4) створюється симетричний ключ безпеки (securityKey) на основі ключа, перетвореного в байтовий масив;

5) визначаються облікові дані для підпису токену (credentials) з використанням симетричного ключа;

6) створюється об'єкт JwtSecurityToken, який представляє токен, з вказанням випускача, клеймів, тривалості дії та облікових даних підпису;

7) застосовується метод WriteToken об'єкта JwtSecurityTokenHandler, щоб отримати рядкове представлення токену;

8) метод повертає рядкове представлення створеного JWT-токену;

9) отриманий JWT-токен може бути використаний для аутентифікації та авторизації користувача на сервері.

Після реалізації бекенду, настав час перейти до фронтенду. Фронтенд відповідає за взаємодію з користувачем, створення інтерфейсу, відображення даних та забезпечення зручної користувацької взаємодії.

Для входу в систему було створено сторінку логіну для авторизації SignIn (рис 3.10).

Компонент SignIn виконує наступні завдання:

1) відображає форму входу на сторінку;

2) обробляє введені дані користувача (електронна пошта і пароль);

3) виконує HTTP-запит до сервера для авторизації користувача за допомогою axios;

4) зберігає токен доступу та електронну пошту користувача в локальному сховищі (localStorage);

5) перенаправляє користувача на сторінку "/posts" після успішного входу;

6) виводить повідомлення про помилку, якщо виникає проблема з авторизацією.

```
export const SignIn = () => {
    const navigate = useNavigate()

    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');

    async function fetchSignIn(){
      try{
        const response = await axios({
            method: 'post',
            url:`http://localhost:43231/api/User/Login`,
            headers: {
                Authorization: `Bearer ${localStorage.getItem('access_token')}`
            },
            data: {
                email: email,
                password: password
            }
        });
```

Рисунок 3.10 – приклад компонента SignIn.

Для створення повідомлень було створено компонент Messages (рис 3.11), який відповідає за відображення сторінки повідомлень і взаємодію з сервером для отримання списку користувачів та обміну повідомленнями. Він має стани для збереження списку користувачів, списку повідомлень та значення введеного тексту. Компонент також має функцію для відправки нового повідомлення та функцію для отримання списку користувачів з сервера. Компонент рендерить форму для введення повідомлення, список повідомлень та список користувачів, які можна вибрати.

```
export const Messages = () => {
    const [users, setUsers] = useState([])
    const [messages, setMessages] = useState([])
    const [inputValue, setInputValue] = useState('')

    const handleSubmit = (e) => {
        e.preventDefault()

        if (inputValue) {
            setMessages(prevState => {
                const newState = [...prevState]
                newState.push(inputValue)
                return newState
            })
            setInputValue('')
        }
    }

    const getUsers = async () => {
        try{
            const response = await axios({
                method: 'get',
                url:`http://localhost:43231/api/User/GetAllUsers`,
                headers: {
                    Authorization: `Bearer ${localStorage.getItem('access_token')}`
                },
            });

            setUsers(response.data)
        } catch (e) {
            console.log(e)
        }
    }
```

Рисунок 3.11 – приклад компонента Messages.

Був створений компонент для постів Posts (рис 3.12) який відображає список постів. Він виконує HTTP-запит до сервера, отримує список постів та відображає їх на сторінці.

Основні елементи коду для компонета Posts:

1) імпортуються необхідні модулі та компоненти React;

2) використовується хук useState для збереження стану компонента. В даному випадку, стан posts використовується для збереження списку постів;

3) функція getPosts виконує HTTP-запит до сервера для отримання списку постів. Вона використовує бібліотеку axios для виконання GET-запиту до URL /api/Post/GetAllPosts на локальному сервері (http://localhost:43231). Запит включає заголовки з токеном доступу (access_token) з localStorage. Отримані дані про пости зберігаються в стані posts;

4) використовується useEffect для виклику функції getPosts один раз під час завантаження компонента;

5) компонент повертає JSX-розмітку, яка відображає список постів. Кожен пост рендериться як окремий блок з назвою, зображенням, описом та розділювальною лінією. Дані про пости беруться зі стану posts.

```
return (
    <div className={styles.posts}>
        <div className='container'>
            {
                posts.map(post => (
                    <div className={styles.post} key={post.postId}>
                        <div className={styles.title}>
                            {post.name}
                        </div>
                        <div className={styles.image}>
                            <img src={` ${post.imagePath}`} alt="model" />
                        </div>
                        <div className={styles.description}>
                            {post.description}
                        </div>
                        <span className={styles.postLine}></span>
                    </div>
                ))
            }
        </div>
    </div>
);
```

Рисунок 3.12 – приклад компонента Posts.

3.2 Приклад роботи інформаційної системи

Спочатку перевіряється функціональне тестування API додатка. Для цього використається Swagger (рис 3.13).

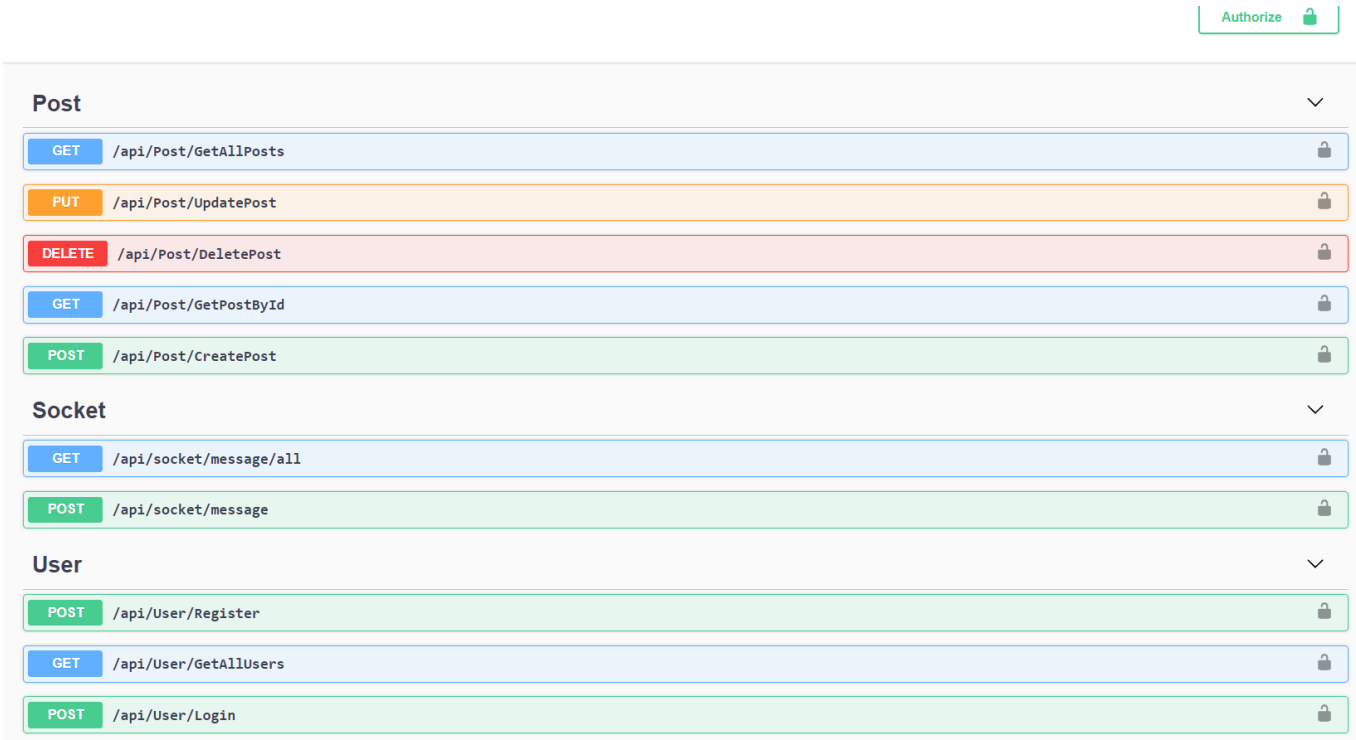


Рисунок 3.13 – Огляд Swagger.

Спочатку реєструється новий користувач (рис 3.14), для цього потрібно ввести такі поля:

1) firstName
2) secondName"
3) email
4) passwor
5) repeatPassword

Після успішної реєстрації або логіна, користувач отримує свій унікальний jwt token, який використовується для подальшої автентифікації при зверненні до ресурсів проекту.

Після реєстрації, у користувача з'являється можливість створювати пости (рис. 3.15). Після створення поста, він добавляється в БД.

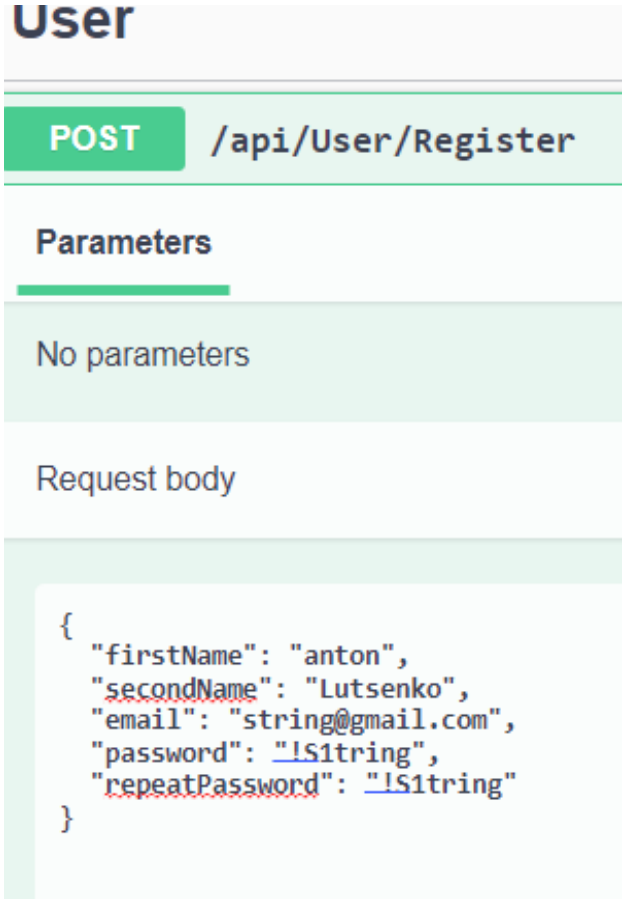


Рисунок 3.14 – Реєстрація юзера в Swagger.

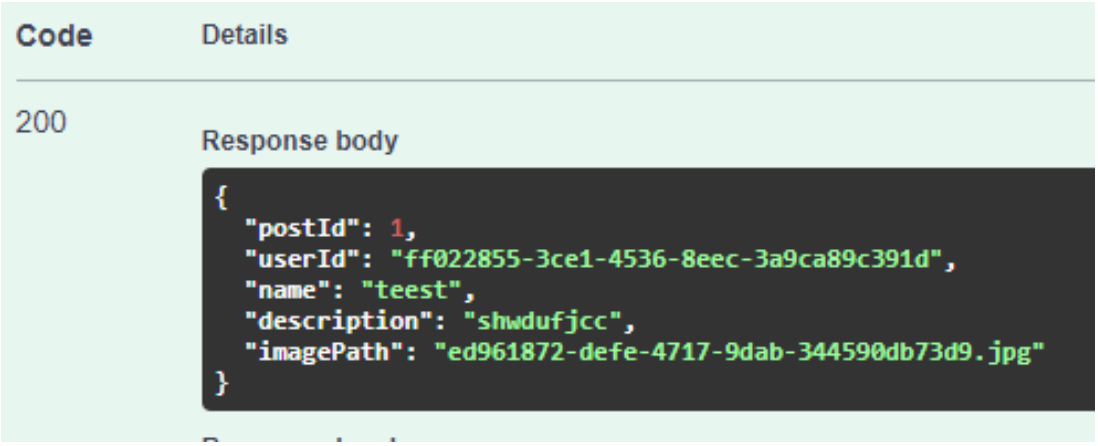


Рисунок 3.15 – Успішне створення поста.

Перевіряється запрос Message чи створюється повідомлення для користувача (рис. 3.16).

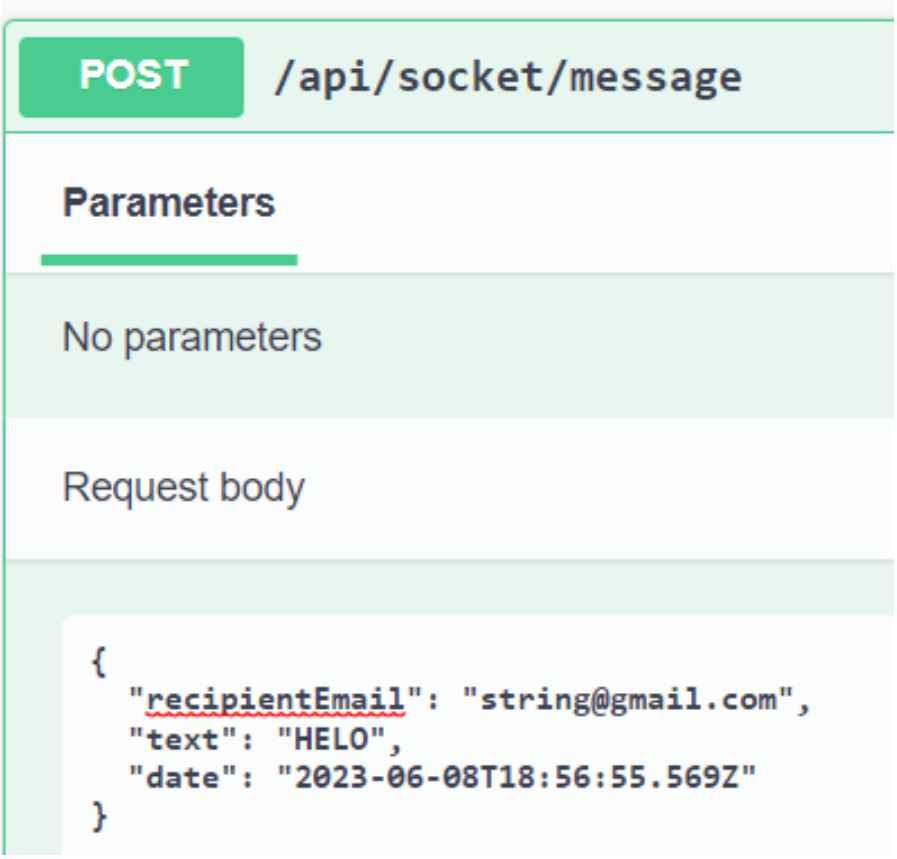


Рисунок 3.16 – Метод Message.

Після запиту показується повідомлення код стану 201, що означає що повідомлення створилось і створився новий ресурс в БД (рис. 3.17).

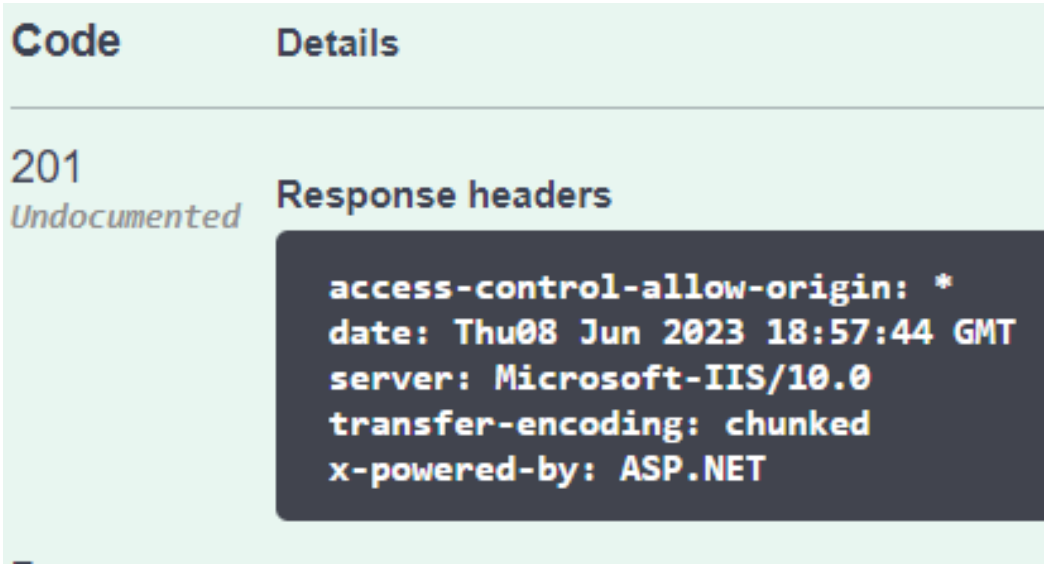


Рисунок 3.17 – Створення повідомлення.

Тепер в акаунті користувача, якому відправлялось повідомлення йде перевірка чи воно прийшло користувачу. Для цього потрібно залогінитись (рис.3.18) і завдяки токену перевірити нові повідомлення (рис. 3.19).

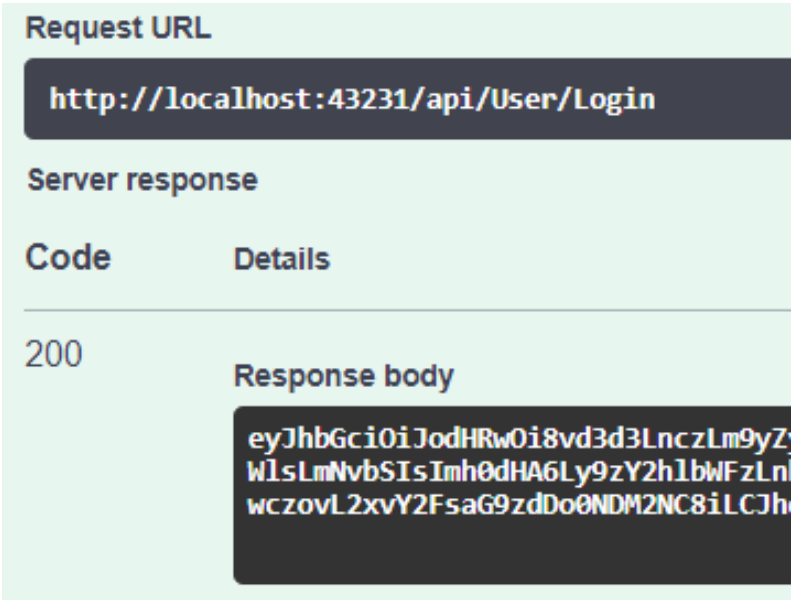


Рисунок 3.18 – Успішний вхід в систему.

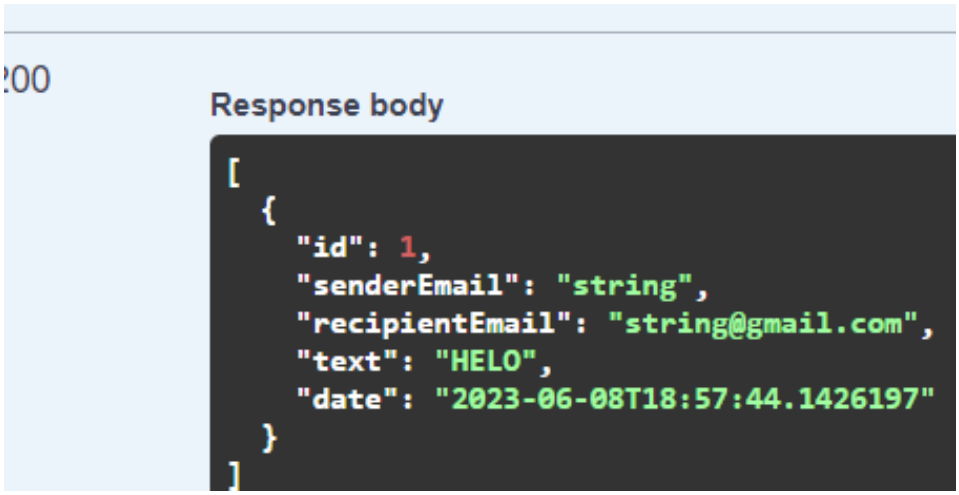


Рисунок 3.19 – Нові повідомлення.

Розглянемо, як виглядає інтерфейс користувача.

На початковій сторінці створено фому реєстрації (рис 3.20). Оскільки реєстрація була здійснена через Swagger, то використовується пошта та пароль існуючого користувача (рис.3.21).

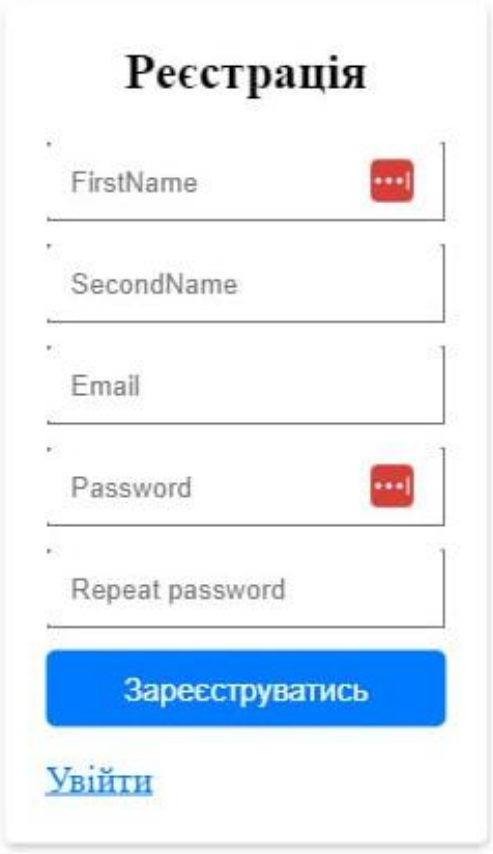


Рисунок 3.20 – Форма реєстрації.

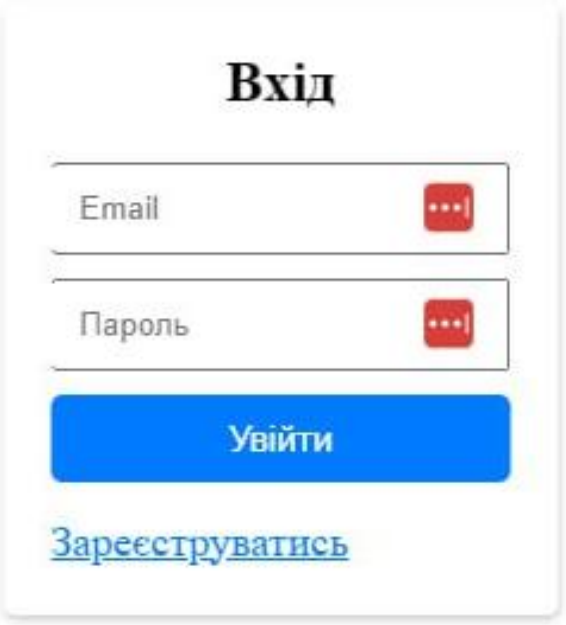


Рисунок 3.21 – Форма логіна.

Після входження на сторінку користувачу висвітлюється сторінка з його постами (рис. 3.22).

В користувача є можливість перейти на сторінку повідомлень, де він вибирає зареєстрованого користувача і може почати з ним переписку (рис. 3.23).

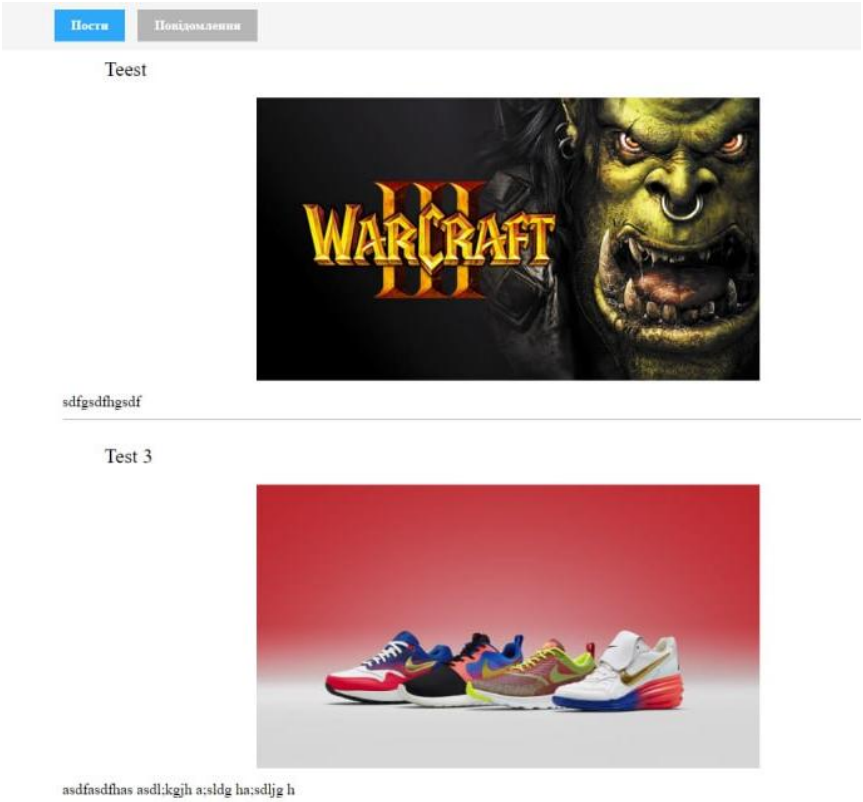


Рисунок 3.22 – Сторінка постів.

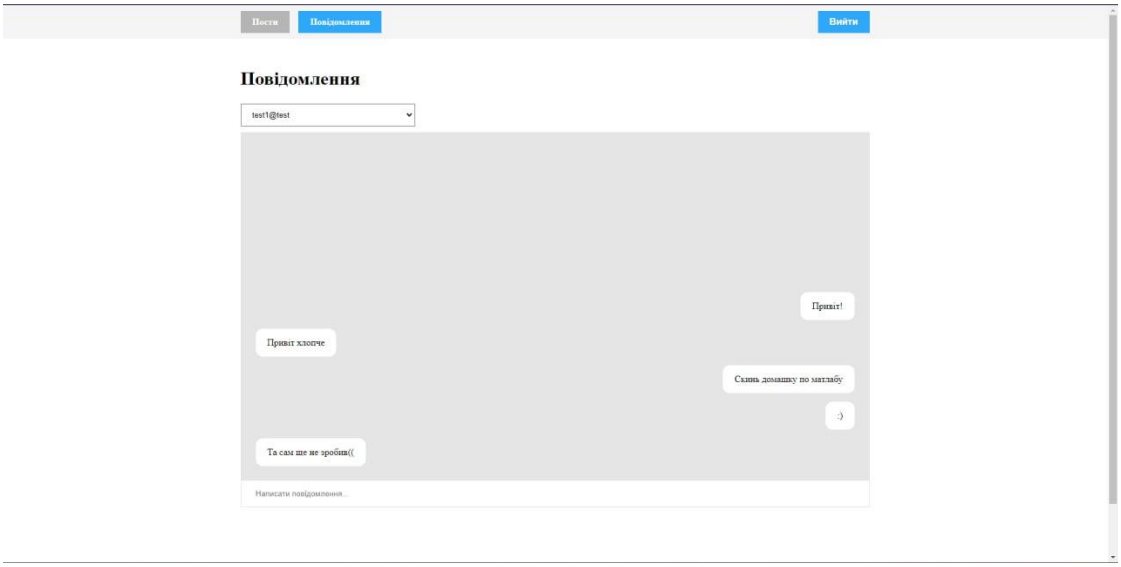


Рисунок 3.24 – Сторінка повідомлень.

3.3 Висновки

В третьому розділі кваліфікаційної роботи виконано реалізацію та тестування роботи інформаційної системи обміну повідомленнями та взаємодії користувачів, а саме: описано реалізацію інформаційної системи обміну повідомленнями та взаємодії користувачів; представлено приклади роботи веб додатку інформаційної системи обміну повідомленнями та взаємодії користувачів.

ВИСНОВКИ

У цій дипломній роботі було розглянуто інформаційну систему обміну повідомленнями та взаємодії користувачів. Дослідження та проектування цієї системи були виконані з метою розробки функціональності, яка дозволяє користувачам створювати свої блоги та взаємодіяти з іншими користувачами.

У процесі дослідження було ретельно проаналізовано існуючі інформаційні системи обміну повідомленнями та взаємодії користувачів, вивчено їх переваги та недоліки. На основі отриманих знань були сформульовані вимоги до системи, включаючи функціональні та нефункціональні вимоги.

Під час проектування системи було використано каскадну модель життєвого циклу, що дозволило послідовно розробляти і тестувати різні компоненти системи. Вибір апаратних та програмних ресурсів був обґрунтований з урахуванням вимог до продуктивності та масштабованості системи.

В першому розділі дипломної роботи було здійснено дослідження предметної області. Відбулось ознайомлення з інформаційною системою обміну повідомленнями та взаємодії користувачів.

В другому розділі дипломної роботи було спроектовано архітектуру інформаційної системи обміну повідомленнями та взаємодії користувачів; проаналізовано ризики проекту; розроблено модель інформаційної системи; розроблено UML-діаграми інформаційної системи.

В третьому розділі дипломної роботи було виконано реалізацію та тестування інформаційної системи обміну повідомленнями та взаємодії користувачів; представлені приклади веб-додатка та сервера інформаційної системи обміну повідомленнями та взаємодії користувачів.

Практична цінність цієї інформаційної системи полягає в тому, що вона надає зручні та ефективні засоби обміну повідомленнями та взаємодії користувачів. Користувачі мають можливість створювати свої блоги, обмінюватись повідомленнями з іншими користувачами та спілкуватись в режимі реального часу. Крім того, система забезпечує зручну синхронізацію даних та

надає користувачам зручний інтерфейс.

Мета роботи була досягнута шляхом розробки та реалізації інформаційної системи обміну повідомленнями та взаємодії користувачів, яка відповідає поставленим вимогам та враховує потреби користувачів. Результати цієї роботи можуть бути використані для подальшого вдосконалення системи та розширення її функціональності.

В цілому, розробка інформаційної системи обміну повідомленнями та взаємодії користувачів має великий потенціал для поліпшення комунікації та спілкування між користувачами. Подальші дослідження можуть спрямовуватись на розширення функціональності системи та покращення її продуктивності з метою забезпечення ще більш задоволенних користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інформаційні технології в сучасному світі. Електронний ресурс. URL: http://sophus.at.ua/publ/2013_12_19_20_kampodilsk/sekcija_7_2013_12_19_20/informacijni_tekhnologiji_v_suchasnomu_sviti/49-1-0-863 (дата звернення: 15.05.2023).
2. Мережні протоколи та технології. Електронний ресурс. URL: http://cs2310.ru/lectures/cs2310_network_protocols.pdf.
3. Технології безпеки Інтернету. Електронний ресурс. URL: https://stepmnvk.net.ua/bezpeka-v-%D1%96nternet%D1%96.html.
4. Кіберзлочинність у всіх її проявах: види, наслідки та способи боротьби. Електронний ресурс. URL: https://www.gurt.org.ua/articles/34602/ (дата звернення: 15.05.2023).
5. Каскадна модель (waterfall model). Електронний ресурс. URL: https://qalight.ua/baza-znaniy/kaskadna-model-waterfall-model/.
6. Моделі життєвого циклу ПЗ. Електронний ресурс. URL: https://pidru4niki.com/1701120547727/informatika/modeli_zhittyevogo_tsiklu.
7. Популярні життєві цикли розробки пз. Електронний ресурс. URL: https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/.
8. Моделі життєвого циклу. Електронний ресурс. URL: https://studfile.net/preview/7689499/page:4/.
9. Організація робіт зі створення інформаційних систем обліку. Стадії та етапи робіт зі створення та впровадження ICO. Електронний ресурс. URL: https://buklib.net/books/24427/.
10. Моделі і методи проектування інформаційних систем. Електронний ресурс. URL: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd8ce4d8e5e/20160217112601/165292/index.html.
11. Керування вимогами як об’єкт автоматизації. Електронний ресурс. URL: https://posibniki.com.ua/post-avtomatizovane-keruvannya-vimogami.

12. Керування ризиками. Функціональні й нефункціональні вимоги. Електронний ресурс. URL: https://ua.kursoviks.com.ua/metodychni_vkazivky/article_post/2271-laboratorna-robota-no2-na-temu-keruvannya-rizikami-funktsionalni-y-nefunktsionalni-vimogi.

13. Особливості функціональних вимог та не функціональних вимог. Електронний ресурс. URL: https://uk.myservername.com/features-functional-requirements.

14. Нефункціональні вимоги до системи: поняття та приклади. Електронний ресурс. URL: https://hi-news.pp.ua/kompyuteri/13402-nefunkconaln-vimogi-do-sistemi-ponyattya-ta-prikladi.html.

15. Опис функціональних вимог. Електронний ресурс. URL: https://tqm.com.ua/ua/kompleksna-avtomatyzatsiia-biznesu/funkcionalni-vimogi.

16. Верифікація та валідація програм. Електронний ресурс. URL: https://stud.com.ua/102371/informatika/verifikatsiya_validatsiya_program.

17. Верифікація та валідація. Електронний ресурс. URL: https://qalight.ua/baza-znaniy/verifikatsiya-ta-validatsiya/.

18. Моделі інформаційних систем. Електронний ресурс. URL: https://moyaosvita.com.ua/informatuka/modeli-informacijnix-sistem/.

19. Моделі і методи проектування інформаційних систем як навчальна дисципліна. Електронний ресурс. URL: https://ua.kursoviks.com.ua/kompyuterni/modeli-i-metodi-proektuvannya-informacijnih-sistem.

20. Моделі і методи проектування інформаційних систем. Електронний ресурс. URL: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd8ce4d8e5e/20160217112601/183252/index.html.

21. Системні особливості моделей інформаційних систем та систем прийняття рішень. Електронний ресурс. URL: https://studfile.net/preview/7768027/page:5/.

22. Основи UML. Електронний ресурс. URL: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html.

23. Уніфікована мова моделювання uml. Електронний ресурс. URL: https://stud.com.ua/138739/ekonomika/unifikovana_mova_modelyuvannya.

24. Як будувати UML-діаграми. Електронний ресурс. URL: https://dou.ua/forums/topic/40575/.

25. Використання уніфікованої мови моделювання. Електронний ресурс. URL: http://iwanoff.inf.ua/oop_kn/LabTraining05.html.

26. Застосування uml в дипломних роботах. Електронний ресурс. URL: https://dut.edu.ua/ua/news-1-626-7758-zastosuvannya-uml-v-diplomnih-robotah_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy.

27. Мова UML. Діаграма використання. Електронний ресурс. URL: http://p4ilka.blogspot.com/2018/12/uml.html.

28. Створення діаграми кооперації. Електронний ресурс. URL: https://vuzlit.com/957907/stvorennya_diagrami_kooperatsiyi.

29. Діаграма кооперації та правила її побудови. Активні і пасивні об'єкти. Зв'язки в кооперації і їх позначення. Електронний ресурс. URL: http://um.co.ua/9/9-2/9-29951.html.

30. Повне розуміння діаграми компонентів UML за допомогою легкого методу. Електронний ресурс. URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/.

31. Діаграма послідовності (Sequence Diagrams). Електронний ресурс. URL: https://www.maxzosim.com/sequence-diagrams/.

32. Діаграма розгортання. Електронний ресурс. URL: https://studfile.net/preview/5010027/page:6/.

33. Діаграма розгортання і правила її побудови. Електронний ресурс. URL: http://um.co.ua/9/9-2/9-29953.html.

34. Офіційна документація EF. Електронний ресурс. URL: https://docs.microsoft.com/ef/ (дата звернення: 15.05.2023).

35. Офіційний репозиторій EF на GitHub. Електронний ресурс. URL: https://github.com/dotnet/efcore (дата звернення: 15.05.2023).

36. EF Core в документації .NET. Електронний ресурс. URL: https://docs.microsoft.com/dotnet/core (дата звернення: 15.05.2023).

37. Entity Framework Tutorial (TutorialsTeacher). Електронний ресурс. URL: https://www.tutorialsteacher.com/efcore (дата звернення: 15.05.2023).

38. Офіційний веб-сайт JWT. Електронний ресурс. URL: https://jwt.io/ (дата звернення: 17.05.2023).

39. Офіційна специфікація JWT. Електронний ресурс. URL: https://tools.ietf.org/html/rfc7519 (дата звернення: 17.05.2023).

40. JWT Authentication в документації ASP.NET Core. Електронний ресурс. URL: https://docs.microsoft.com/aspnet/core/ (дата звернення: 17.05.2023).

Додаток А

(обов’язковий)

Копія креслення «Планування проектних робіт»

ПЛАНУВАННЯ ПРОЕКТНИХ РОБІТ

№	Ризик	Ймовірність виникнення	Стратегія усунення	Ймовірність після стратегії
1	Втрата даних	Висока	Регулярне резервування даних	Середня
2	Технічний збій	Середня	Регулярне обслуговування обладнання	Низька
3	Відсутність інтернет-з'єднання	Висока	Резервування доступних інтернет-каналів	Середня
4	Несанкціонований доступ до системи	Низька	Впровадження мінімальних механізмів аутентифікації та авторизації	Дуже низька
5	Втрата електропостачання	Середня	Установка довгкових джерел живлення, акумулят. резервних батарей	Низька

Аналіз вимог

Проектування

Реалізація

Тестування

Впровадження

КвРІСТ 200196.20.01.08

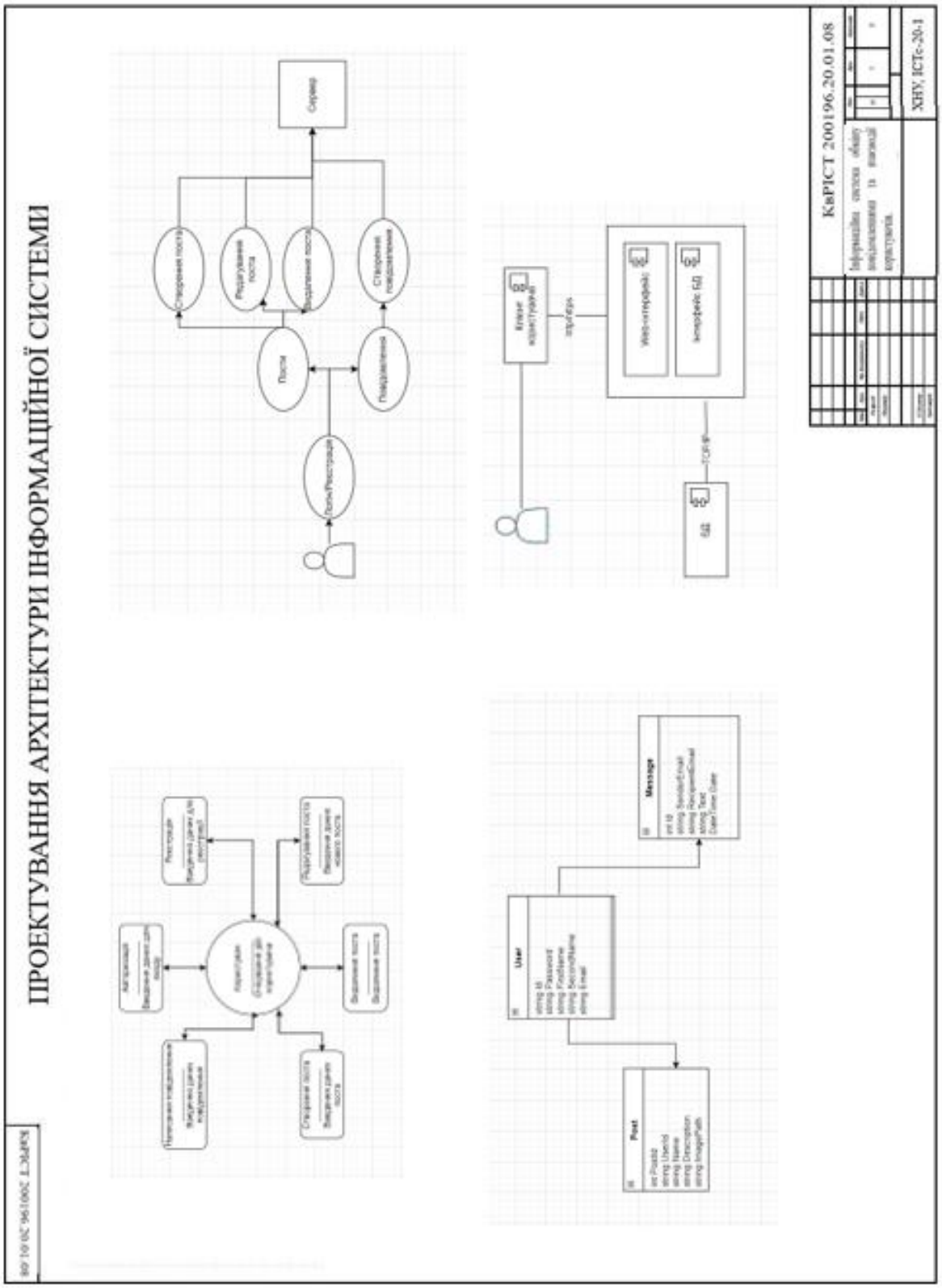
Інформаційна система обліку замовлення та виконання курсового проекту. Планування проектних робіт

ХНУ, ІСТс-20-1

Додаток Б

(обов’язковий)

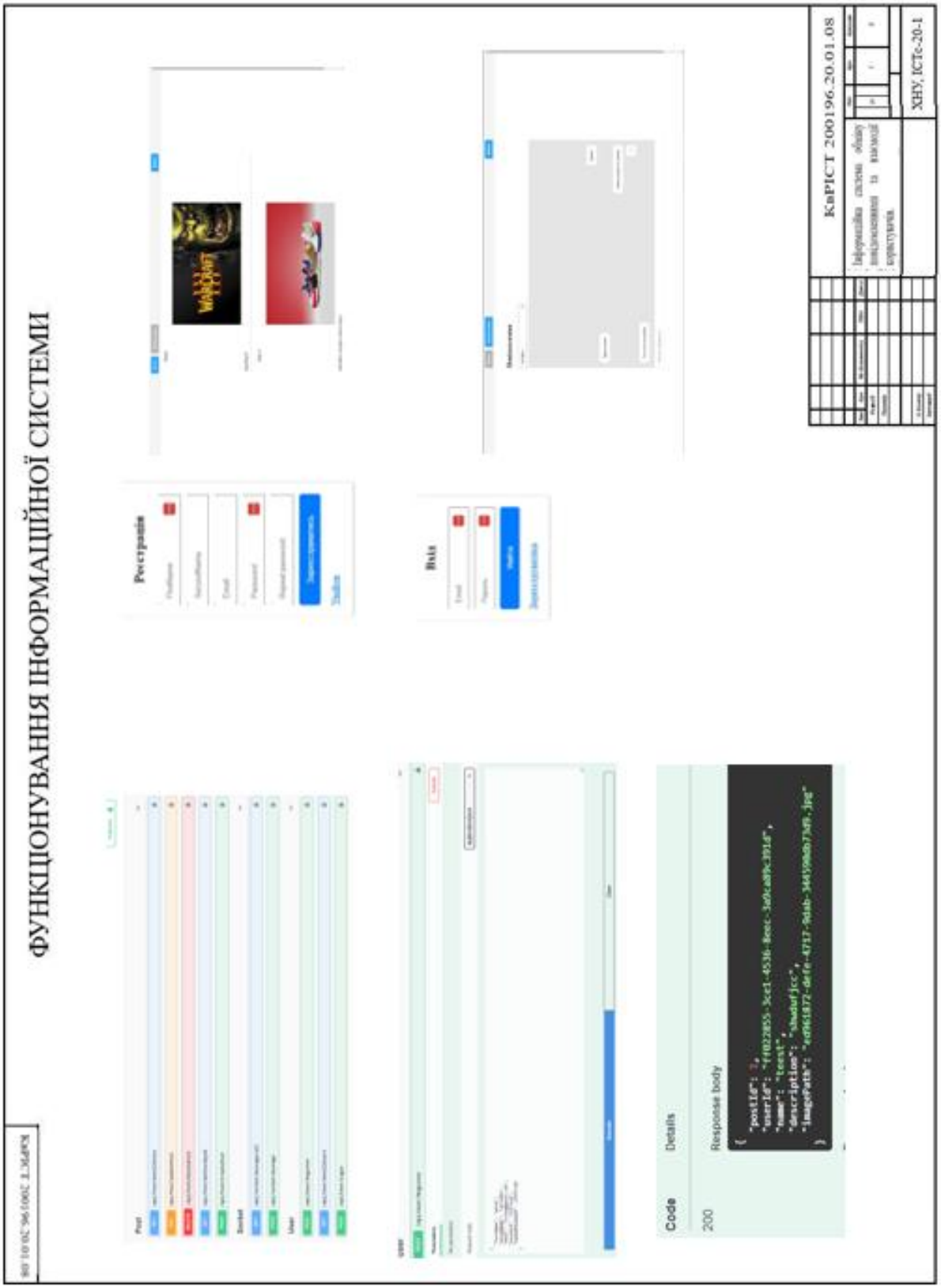
Копія креслення «Проектування архітектури інформаційної системи»



Додаток В

(обов’язковий)

Копія креслення «Функціонування інформаційної системи»



Додаток Г

Лістинг коду

UserController.cs

```
[Route("api/[controller]")]
    [ApiController]
    public class UserController : ControllerBase
    {
        private readonly UserService _userService;
        private readonly ApplicationContext _dbContext;
        private readonly UserManager<User> _userManager;
        private readonly ITokenService _tokenService;

        public UserController(UserService userService, ApplicationContext dbContext,
UserManager<User> userManager, ITokenService tokenService)
        {
            _userService = userService;
            _dbContext = dbContext;
            _tokenService = tokenService;
            _userManager = userManager;
        }

        [HttpPost("Register")]
        public async Task<IActionResult> RegisterUser(RegisterUserModel model)
        {
            User user = new User
            {
                Email = model.Email,
                UserName = model.Email,
                FirstName = model.FirstName,
                SecondName = model.SecondName
            };

            if (model.Password != model.RepeatPassword)
                return BadRequest("Error");

            var result = await _userManager.CreateAsync(user, model.Password);
            if (!result.Succeeded)
            {
                return BadRequest(result.Errors);
            }

            var validUser = _userService.GetUserByEmail(model.Email);
            string generatedToken = _tokenService.BuildToken(validUser);
            if (generatedToken != null)
            {
                return Ok(generatedToken);
            }

            return BadRequest("Error");
        }


        [HttpGet("GetAllUsers")]
        public IActionResult GetAllUsers()
        {
            var result = _userService.GetAllUsers();
            return Ok(result);
        }

        [HttpPost("Login")]
```

```
        public async Task<IActionResult> Login([FromBody] LoginViewModel model)
        {
            if (string.IsNullOrEmpty(model.Email) ||
string.IsNullOrEmpty(model.Password))
            {
                return BadRequest("Error");
            }

            var validUser = _userService.GetUserByEmail(model.Email);
            if (validUser == null)
                return BadRequest("Error");

            string generatedToken = _tokenService.BuildToken(validUser);
            if (generatedToken != null)
            {
                return Ok(generatedToken);
            }

            return BadRequest();
        }
    }
```

PostController.cs

```
[Route("api/[controller]")]
    [Authorize]
    [ApiController]
    public class PostController : ControllerBase
    {
        private readonly PostService _postService;
        private readonly ApplicationContext _dbContext;
        public PostController(PostService post, ApplicationContext dbContext)
        {
            _postService = post;
            _dbContext = dbContext;
        }


        [HttpGet("GetAllPosts")]
        public IActionResult GetAllPosts()
        {
            var user = (User)HttpContext.Items["User"];
            var result = _postService.GetAllPosts(user.Id);
            return Ok(result);
        }


        [HttpPut("UpdatePost")]
        public async Task<IActionResult> UpdatePost([FromForm] UpdatePostModel model)
        {
            var user = (User)HttpContext.Items["User"];
            var result = _postService.UpdatePost(model, user.Id);
            return Ok(result);
        }

        [HttpDelete("DeletePost")]
        public IActionResult DeletePost(int postId)
        {
            _postService.DeletePost(postId);
            return Ok();
        }

        [HttpGet("GetPostById")]
        public IActionResult GetPostById(int postId)
```

```
        {
            var result = _postService.GetPostById(postId);
            //return Ok(result);
            if (result == null)
            {
                return NotFound();
            }

            var imagePath = Path.Combine("Images", result.ImagePath);

            if (!System.IO.File.Exists(imagePath))
            {
                return NotFound();
            }

            var imageContent = System.IO.File.ReadAllBytes(imagePath);
            return new FileContentResult(imageContent, "image/png");
        }


        [HttpPost("CreatePost")]
        public async Task<IActionResult> CreatePost([FromForm] CreatePostModel model)
        {

            var uploadsFolder = Path.Combine(Directory.GetCurrentDirectory(),
"Images");
            if (!Directory.Exists(uploadsFolder))
            {
                Directory.CreateDirectory(uploadsFolder);
            }

            var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(model.Image.FileName);
            var filePath = Path.Combine(uploadsFolder, fileName);

            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                await model.Image.CopyToAsync(stream);// копіювання зображення в файл
            }
            var user = (User)HttpContext.Items["User"];
            Post post = new Post {
                Name = model.Name,
                Description = model.Description,
                ImagePath = fileName,
                UserId = user.Id
            };

            await _postService.CreatePost(post);

            return Ok(post);
        }
    }
```

SocketController.cs

```
[Authorize]
    [ApiController]
    [Route("api/socket")]
    [Produces("application/json")]
    [Consumes("application/json")]
    public class SocketController : ControllerBase
    {
        const int MaxMessageSymbolsCount = 1024;
        const int StringTypeBytesCount = 4;
```

```
        private readonly int MessageBufferSize = MaxMessageSymbolsCount *
StringTypeBytesCount;

        private readonly ILogger<SocketController> _logger;
        private readonly ApplicationContext _context;
        private readonly WebSocketConnectionManager _connectionManager;
        private readonly IMessageService _messageService;

        WebSocketManager ContextWebSockets => HttpContext.WebSockets;
        bool IsNotWebSocketRequest => !ContextWebSockets.IsWebSocketRequest;
        User ConnectedUser => (User)HttpContext.Items["User"];

        public SocketController(
            ILogger<SocketController> logger,
            ApplicationContext context,
            WebSocketConnectionManager connectionManager,
            IMessageService messageService
        )
        {
            _logger = logger;
            _context = context;
            _connectionManager = connectionManager;
            _messageService = messageService;
        }

        [HttpGet("message/all")]
        public IActionResult GetAllMessages()
        {
            _logger.Log(
                LogLevel.Information,
                $"Getting messages messages for {ConnectedUser.Email}..."
            );

            var messages = _context.Messages
                .Where(m => m.RecipientEmail == ConnectedUser.Email)
                .ToList();

            return Ok(messages);
        }

        [HttpPost("message")]
        public async Task<IActionResult> SendMessage(CreateMessageModel model)
        {
            try
            {
                var messageModel = new Message
                {
                    SenderEmail = ConnectedUser.Email,
                    RecipientEmail = model.RecipientEmail,
                    Text = model.Text,
                    Date = model.Date
                };

                _logger.Log(
                    LogLevel.Information,
                    $"Sending message to {messageModel.RecipientEmail}..."
                );

                await SendMessage(
                    messageModel.SenderEmail,
                    messageModel.RecipientEmail,
                    messageModel.Text
                );

                await _messageService.SaveMessage(
                    messageModel.SenderEmail,
                    messageModel.RecipientEmail,
```

```
                    messageModel.Text
                );

                return new StatusCodeResult((int)HttpStatusCode.Created);
            }
            catch (Exception ex)
            {
                var errorMessage = $"Error sending message to {model.RecipientEmail}
from {ConnectedUser.Email}";
                _logger.LogError(ex, errorMessage);
                return StatusCode(500, errorMessage);
            }
        }

        [Route("")]
        public async Task<IActionResult> StartWebSocket()
        {
            if (IsNotWebSocketRequest)
            {
                return BadRequest();
            }
            _logger.Log(
                LogLevel.Information,
                $"Receiving messages for {ConnectedUser.Email}..."
            );

            var webSocket = await CreateConnection(ConnectedUser.Email);
            var messageReceiveResult = await ReceiveMessages(ConnectedUser.Email,
webSocket);
            await CloseConnection(ConnectedUser.Email, webSocket,
messageReceiveResult);

            // TODO: Check why connection is closed

            return new StatusCodeResult((int)HttpStatusCode.Created);
        }

        private async Task SendMessage(string senderEmail, string recipientEmail,
string text)
        {
            var webSocket = _connectionManager.GetConnection(recipientEmail);

            if (webSocket?.State == WebSockets.WebSocketState.Open)
            {
                await SendMessage(senderEmail, text, webSocket);
            }
        }

        private async Task SendMessage(
            string senderEmail,
            string text,
            WebSockets.WebSocket webSocket
        )
        {
            var content = new IncommingMessageModel
            {
                SenderEmail = senderEmail,
                Message = text
            };
            var contentString = JsonConvert.SerializeObject(content);

            await webSocket.SendAsync(
                new ArraySegment<byte>(Encoding.UTF8.GetBytes(contentString)),
                WebSockets.WebSocketMessageType.Text,
                true,
                CancellationToken.None
            );
```

```
        }

        private async Task<WebSockets.WebSocket> CreateConnection(string userEmail)
        {
            var webSocket = await ContextWebSockets.AcceptWebSocketAsync();
            _connectionManager.AddConnection(userEmail, webSocket);

            _logger.Log(
                LogLevel.Information,
                $"Connection for user {userEmail} is established"
            );

            return webSocket;
        }

        private async Task<WebSockets.WebSocketReceiveResult> ReceiveMessages(
            string receiverEmail,
            WebSockets.WebSocket webSocket
        )
        {
            _logger.Log(
                LogLevel.Information,
                $"Receiving and saving messages for {receiverEmail}..."
            );

            var messageBuffer = new byte[MessageBufferSize];
            var messageReceivedResult = await ReceiveMessage(messageBuffer, webSocket);

            while (!messageReceivedResult.CloseStatus.HasValue)
            {
                var messageContent = GetMessageContent(messageBuffer,
messageReceivedResult);
                await _messageService.SaveMessage(
                    messageContent.SenderEmail,
                    receiverEmail,
                    messageContent.Message
                );

                messageReceivedResult = await ReceiveMessage(messageBuffer, webSocket);
            }

            return messageReceivedResult;
        }

        private async Task<WebSockets.WebSocketReceiveResult> ReceiveMessage(
            byte[] messageBuffer,
            WebSockets.WebSocket webSocket
        )
        {
            return await webSocket.ReceiveAsync(
                new ArraySegment<byte>(messageBuffer),
                CancellationToken.None
            );
        }

        private IncommingMessageModel GetMessageContent(
            byte[] buffer,
            WebSockets.WebSocketReceiveResult result
        )
        {
            var contentString = Encoding.UTF8.GetString(buffer, 0, result.Count);
            return JsonConvert.DeserializeObject<IncommingMessageModel>(contentString);
        }

        private async Task CloseConnection(
            string userEmail,
            WebSockets.WebSocket webSocket,
```

```
            WebSockets.WebSocketReceiveResult result
        )
        {
            _connectionManager.RemoveConnection(userEmail);
            await webSocket.CloseAsync(
                result.CloseStatus.Value,
                result.CloseStatusDescription,
                CancellationToken.None
            );

            _logger.Log(
                LogLevel.Information,
                $"Connection for {userEmail} is closed"
            );
        }

    }
```

Startup.cs

```
public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to
the container.
        public void ConfigureServices(IServiceCollection services)
        {
            //services.AddWebSockets();

            services.AddSingleton<WebSocketConnectionManager>();
            services.AddScoped<IMessageService, MessageService>();

            services.AddTransient<UserService>();
            services.AddTransient<UserRepository>();
            services.AddTransient<IUserRepository, UserRepository>();
            services.AddTransient<PostService>();
            services.AddTransient<PostRepository>();
            services.AddTransient<IPostRepository, PostRepository>();

            string connection = Configuration.GetConnectionString("DefaultConnection");

            services.AddDbContext<ApplicationContext>(options =>
                options.UseSqlServer(connection));
            services.AddControllersWithViews();
            services.AddControllers();
            services.AddScoped<ITokenService, TokenService>();
            services.AddIdentity<User, IdentityRole>()
                .AddEntityFrameworkStores<ApplicationContext>();
            services.AddScoped<UserManager<User>>();

            services.AddSwaggerGen((option) =>
            {
                option.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme()
                {
                    Name = "Authorization",
                    Type = SecuritySchemeType.ApiKey,
                    Scheme = "Bearer",
```

```
                    BearerFormat = "JWT",
                    In = ParameterLocation.Header,
                    Description = "JWT Authorization header using the Bearer scheme.
\r\n\r\n Enter 'Bearer' [space] and then your token in the text input
below.\r\n\r\nExample: \"Bearer 1safsfsdfdfd\"",
                });
                option.AddSecurityRequirement(new OpenApiSecurityRequirement
                {
                    {
                          new OpenApiSecurityScheme
                            {
                                Reference = new OpenApiReference
                                {
                                    Type = ReferenceType.SecurityScheme,
                                    Id = "Bearer"
                                }
                            },
                            new string[] {}
                    }
                });
            });

        }

.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
WebSocketConnectionManager connectionManager)
        {
            app.UseAuthentication();

            app.UseCors(builder => builder
            .AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader());

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"Diplom v1"));
            }

            app.Use(async (context, next) =>
            {
                if (context.WebSockets.IsWebSocketRequest)
                {
                    WebSockets.WebSocket webSocket = await
context.WebSockets.AcceptWebSocketAsync();
                    string userId = context.Request.Query["userId"];

                    connectionManager.AddConnection(userId, webSocket);
                    connectionManager.RemoveConnection(userId);
                }
                else
                {
                    await next();
                }
            });

            app.UseWebSockets();

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseMiddleware<JwtMiddleware>();
```

```
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
```

PostRepository.cs

```
public class PostRepository : IPostRepository
    {
        private readonly ApplicationContext _dbContext;

        public PostRepository(ApplicationContext databaseOptions)
        {
            _dbContext = databaseOptions;
        }

        public async Task CreatePost(Post post)
        {
           await _dbContext.Posts.AddAsync(post);
           await _dbContext.SaveChangesAsync();
        }

        public List<Post> GetAllPosts(string userId)
        {
            return _dbContext.Posts.Where(w => w.UserId == userId || w.UserId ==
null).ToList();
        }

        public Post UpdatePost(UpdatePostModel model, string userId)
        {
            var oldModel = _dbContext.Posts.FirstOrDefault(x => x.PostId ==
model.PostId && x.UserId == userId);
            if (oldModel != null)
            {
                oldModel.Name = model.Name;
                oldModel.Description = model.Description;
               // oldModel.ImagePath = model.Image;
                _dbContext.SaveChanges();
            }
            return oldModel;
        }

        public void DeletePost(int postId)
        {
            var post = _dbContext.Posts.FirstOrDefault(x => x.PostId == postId);
            if (post != null)
            {
                _dbContext.Posts.Remove(post);
                _dbContext.SaveChanges();
            }
        }

        public Post GetPostById(int postId)
        {
            return _dbContext.Posts.FirstOrDefault(x => x.PostId == postId) ;
        }
    }
```

TokenService.cs

```
public interface ITokenService
    {
        string BuildToken(User user);
        void AttachUserToContext(HttpContext context, string token);
    }
    public class TokenService : ITokenService
    {
        private const double EXPIRY_DURATION_MINUTES = 30;
        private readonly IConfiguration _config;
        private readonly UserService _userService;

        public TokenService(IConfiguration config, UserService userService)
        {
            _config = config;
            _userService = userService;
        }

        public void AttachUserToContext(HttpContext context, string token)
        {
            try
            {
                var tokenHandler = new JwtSecurityTokenHandler();
                var key = Encoding.ASCII.GetBytes(_config["Jwt:Key"].ToString());
                tokenHandler.ValidateToken(token, new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(key),
                    ValidateIssuer = false,
                    ValidateAudience = false,
                    // set clockskew to zero so tokens expire exactly at token
expiration time (instead of 5 minutes later)
                    ClockSkew = TimeSpan.Zero
                }, out SecurityToken validatedToken);

                var jwtToken = (JwtSecurityToken)validatedToken;
                string userName = jwtToken.Claims.First(x => x.Type ==
ClaimTypes.Name).Value;

                // attach user to context on successful jwt validation
                context.Items["User"] = _userService.GetUserByEmail(userName);
            }
            catch
            {
                // do nothing if jwt validation fails
                // user is not attached to context so request won't have access to
secure routes
            }
        }

        public string BuildToken(User user)
        {
            Claim[] claims = new[] {
                new Claim(ClaimTypes.Name, user.UserName),
                new Claim(ClaimTypes.NameIdentifier,
                Guid.NewGuid().ToString())
            };

            string key = _config["Jwt:Key"].ToString();
            string issuer = _config["Jwt:Issuer"].ToString();

            TimeSpan expiryDuration = TimeSpan.FromHours(1);
```

```
            SymmetricSecurityKey securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
            SigningCredentials credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256Signature);
            //JwtSecurityToken tokenDescriptor = new JwtSecurityToken(issuer, issuer,
claims,
            //    expires: DateTime.Now.AddMinutes(EXPIRY_DURATION_MINUTES),
signingCredentials: credentials);
            JwtSecurityToken tokenDescriptor = new JwtSecurityToken(issuer, issuer,
claims,
                expires: DateTime.Now.Add(expiryDuration), signingCredentials:
credentials);

            return new JwtSecurityTokenHandler().WriteToken(tokenDescriptor);
        }
    }
```

UNICHECK

User name:
Кафедра КІ

Check ID:
1015526024

Check date:
09.06.2023 12:01:17 EEST

Check type:
Doc vs Internet + Library

Report date:
09.06.2023 12:02:21 EEST

User ID:
100005591

File name: **Луценка_Інформаційна система обміну повідомленнями та взаємодії користувачів**

Page count: **55** Word count: **6627** Character count: **55307** File size: **2.25 MB** File ID: **1015180059**

Text modifications detected (similarity score might be affected)

9.97% Matches

Highest match: **5.16%** with Library source (File ID: **1014915262**)

8.18% Internet sources | 237 ..... Page 57

8% Library sources | 143 ..... Page 58

0.06% Quotes

Quotes | 1 ..... Page 59

No references found

0% Exclusions

No exclusions

Modifind

Text modifications detected. Find more details in the online report.

Suspicious formatting | 13 Pages

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

ID: 115417 Назва: БКР Інформаційна система обміну повідомленнями та взаємодії користувачів Додано в БД: 2023-06-09 Автора: А.М. Луценко Керівники: К.Ю. Бобровнікова Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	46572	444	2633 (6%)	34 (8%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Луценко Антон Михайлович

Тема: Інформаційна система обміну повідомленнями та взаємодії користувачів

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 56

1. Короткий зміст роботи та прийнятих рішень: У кваліфікаційній роботі було досліджено і проаналізовано предметну область. Був проведений аналіз відомих комерційних рішень, розглянуто їх переваги та недоліки і доведено актуальність розроблення нової інформаційної системи. Розглянуто інструменти для реалізації спроєктованих рішень, визначено ризики проекту, спроектовано та реалізовано інформаційну систему.

2. Висновок про відповідність роботи дипломному завданню: Кваліфікаційна робота відповідає поставленому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі дипломної роботи було здійснено дослідження предметної області та обґрунтовано модель життєвого циклу інформаційної системи. В другому розділі дипломної роботи було спроектовано архітектуру інформаційної системи обміну повідомленнями та взаємодії користувачів; проаналізовано ризики проекту; розроблено модель інформаційної системи; розроблено UML-діаграми інформаційної системи. В третьому розділі дипломної роботи

представлено реалізацію інформаційної системи обміну повідомленнями та взаємодії користувачів; представлені приклади інтерфейсів веб-додатка та сервера інформаційної системи обміну повідомленнями та взаємодії користувачів.

4. Позитивні сторони роботи: До позитивних сторін роботи можна віднести актуальність її тематики.

5. Негативні сторони роботи: Недостатня увага захисту повідомлень в розробленій інформаційній системі

6. Оцінка графічного оформлення та пояснювальної записки роботи: Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про роботу в цілому: Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє зрозуміти викладений матеріал в межах тематики проєктування. Графічний матеріал надає можливість наочно побачити деталі проектування інформаційної системи.

8. Інші зауваження:

9. Оцінка дипломної роботи: задовільно (3/E)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Клюх Ю.П. зав. кафедри кібербезпеки, к.т.н., доцент

"26" 06 2023 р. (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Луценка Антона Михайловича
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи ІСТс-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

22 квітня 2023 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інформаційна система обміну повідомленнями та взаємодії користувачів

Автор: Луценко Антон Михайлович

Спеціальність: 126 – Інформаційні системи та технології

Освітня програма: освітньо-професійна

Науковий керівник: Бобровнікова Кіра Юліївна, к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення, які мають місце в розділах аналізу існуючих аналогів та прототипів, не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення є фрагментарними, або мають належним чином оформленні посилання;
3) в якості запозичень в окремих місцях системою зафіксовано зарезервовані ключові слова мови програмування, які використовуються для розв'язку великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 9.97% і адресується до 380 першоджерел, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ К. Ю. Бобровнікова

Гарант ОП _____ С. М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко