Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Веб-платформа TOD для обміну інформацією за інтересами

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДПІПЗ.180117.01.11.ПЗ

Виконав студент IV курсу група ІПЗ-18-1 _____ Є. І. Слутяк
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ О. М. Яшина
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. пед. наук, доцент _____ Н. І. Праворська
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

3 червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
01 03 2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Слутяку Євгенію Івановичу

Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Веб-платформа TOD для обміну інформацією за інтересами

Керівник проєкту (роботи) Оксана Миколаївна Яшина, канд. техн. наук, доцент

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

Затверджена наказом ректора університету від 01.03.2022 р. № 18

2. Строк подання студентом проєкту (роботи) на кафедру 01.06.2022 р.

3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 17 шт)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Праворська Н. І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання «_01_»__березня_ 2022 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних тем ДП	01.12 – 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог розробки технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02.2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	Травень 2022 (згідно графіка)	
8 Перевірка ДП на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2022	
9 Підготовка до захисту ДП	з 01.06.2022	

Студент _____ Є. І. Слутяк
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О. М. Яковлєва
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: «Веб-платформа TOD для обміну інформацією за інтересами».

Автор проекту: Слутяк Євгеній Іванович

Керівник проекту: к. т. н., доцент Яшина Оксана Миколаївна.

Пояснювальна записка: 119 с., 10 рис., 17 табл., 3 дод., 46 джерел.

Графічна частина: 17 презентаційних слайдів.

ВЕБ-ПЛАТФОРМА, ФОРУМ, СОЦІАЛЬНА МЕРЕЖА, ВЕБ-САЙТ, ASP.NET CORE, C#, ENTITY FRAMEWORK CORE, THREE TIER ARCHITECTURE, MS SQL SERVER, DOCKER, HEROKU, JWT TOKEN, WEB-API, JAVASCRIPT, VUE.JS, REDIS CACHE.

Метою проекту є розробка веб-платформи для налаштування комунікації, а саме створення тем та проведення дискусій між користувачами мережі Інтернет.

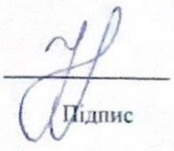
У даному дипломному проекті було проведено аналіз предметної області та найпопулярніших серед наявних веб-платформи, визначено конкретні вимоги до системи, розроблено загальну архітектуру програмного забезпечення, спроектовано структуру та усі зв'язки бази даних, спроектовано структуру програмного забезпечення. Визначено основні та похідні модулі, здійснено реалізацію системи згідно з раніше спроектованим проектом.

Для розробки веб-платформи було використано мову програмування C# та JavaScript, систему керування базами даних MS SQL Server та такі технології, як Entity Framework Core, Vue.js, ASP.NET Core.

У результаті роботи було реалізовано веб-платформу для обміну інтересами.

03.06.2022р.

Дата



Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ.180117.01.11.ПЗ	Пояснювальна записка	119		
2	А4		Завдання на дипломний	2		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	17		

Змн	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.180117.01.11.ВД		
Виконав		Слутяк Є. І.			Веб-платформа TOD для обміну інформацією за інтересами Відомість документів	Літ.	Арк. 4 / Акрушів 119
Керівник.		Яшина О. М.					
Н. Контр.		ПраворськаН І.				ХНУ, ІПЗ-18-1	
Затверд.		Бедратюк Л.П.					

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

DBS	–	Data Base Server
ООП	–	об’єктно-орієнтоване програмування
ПЗ	–	програмне забезпечення
БД	–	база даних
CSS	–	Cascading Style Sheets
HTTP	–	HyperText Transfer Protocol
ПЗ	–	програмне забезпечення
СУБД	–	Система управління базами даних
HTML	–	HyperText Markup Language
API	–	Application programming interface
UML	–	Unified Modeling Language
JS	–	JavaScript
JWT	–	Json Web Tokens
Json	–	JavaScript Object Notation
MVC	–	Model View Controller
MVVM	–	Model View ViewModel
TTA	–	Three Tier Architecture
SQL	–	Structured Query Language
DTO	–	Data Transfer Object

ВСТУП

Метою даного дипломного проекту є розробка веб-платформи для налаштування комунікації, а саме створення тем та проведення дискусій між користувачами мережі Інтернет.

Завданням даного дипломного проекту є дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація платформи та тестування готового продукту.

Кожен з користувачів мережі Інтернет стикався з дискусіями на форумах чи з коментарями під постами в соціальних мережах. Ці дискусії доводять важливу тезу про усе людство, що вони – комунікативні створіння. Тому їм надзвичайно важливо виражати свою думку, мати кумира, думка якого збігається з його думкою та слідкувати за висловленнями кожного з них. Комунікативним особам надзвичайно важливо дізнаватися та обговорювати думки інших людей з приводу будь-яких питань. Дискутант повинен бути розвинений різносторонньо, оскільки це показує оточуючим, що ця персона є освіченою, з нею є про що поговорити, і навіть якщо вона не є експертом у всьому, вона може підтримати розмову на будь-яку тему та пороздумувати над будь-якими питаннями.

Саме це і є причиною створення будь-яких комунікативних платформ: жага людини до спілкування. Існує програмне забезпечення, системи чи платформи, які розробляються та запускаються неприроднім шляхом, тобто для людини цей додаток не є необхідним, і якби дана компанія чи програміст не розробив цей продукт, користувачі навіть не помітило б цього. Натомість, якби Марк Цукерберг не розробив соціальну мережу Facebook [2], чи якби Стів Гаффман не створив Reddit [3], замість них це обов’язково зробив би хтось інший, оскільки ці платформи були створені природнім шляхом, оскільки людство потребує спілкування.

Всесвітня мережа Інтернет – один з найцінніших інструментів, винайдений людством за весь час свого існування. Цей інструмент дозволяє миттєво обмінюватися з іншим користувачем мережі на іншому кінці всесвіту будь-якою

інформацією – від звичайного тексту, до аудіо та відеоконтенту. Тому, маючи в руках такий потужний та важливий інструмент, розробники щодня розгортають нові платформи [4] та щодня покращують уже створені продукти у всіх можливих сферах, в тому числі і у сфері комунікації.

І оскільки людство потребує спілкування, створено чимало різноманітних додатків [5] та платформ [6] з метою максимально задовольнити потреби користувачів мережі Інтернет. Вони створюються, оскільки кожен розробник бачить проблеми інших продуктів та хоче розробити власний продукт, намагаючись уникнути усіх помилок, допущених його попередниками.

І серед безлічі створених платформ можна помітити особливість кожної з них: якісь платформи дуже влучно зайняли популярну нішу та завоювали користувачів саме завдяки своєрідній унікальності свого продукту, якісь платформи завоювали користувачів завдяки максимальній відкритості свого контенту та практично відсутності модерації, а якісь платформи навпаки завоювали свою популярність завдяки тому, що дають висловлюватися лише окремій групі людей, блокуючи висловлювання протилежної думки і таким чином створюючи максимально зручні умови для існування представників саме конкретної політичної, релігійної чи іншої спілки людей.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ТЕМОЮ ДП ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

За весь період свого існування людська раса відзначалася постійною жагою до вивчення чогось нового, до дослідження всього, що нас оточує, до спілкування і кооперації один з одним. Одна з рис, яка відрізняє людину від інших тварин – це максимально усвідомлене спілкування, при використанні якого можна досягати абсолютно будь-яких цілей: від вбивства велетенських мамонтів до організації польотів на Марс. Знаючи і розуміючи силу людської комунікації, необхідно зробити все для того, щоб налагодити її.

Вперше проект глобальної мережі інтернет описав американський науковець Джозеф Ліклайдер у 1960 р. [7]. У 1962 р. американський інженер Пол Берен створив концепцію Інтернету, яку описав у 12-томній роботі [8]. І вже 29 жовтня 1969 р. проект ARPANET спробував відправити перше повідомлення між двома американськими університетами: каліфорнійським та стенфордським. Завданням було відправити і отримати на іншому боці слово “LOGIN” [9]. Проте, перший експеримент видався невдалим – у стенфордському університеті отримали лише перші дві букви. З часом науковці просувалися вперед, з’являлися можливості відправити слово, фразу, текст, зображення, пісню, відео, а згодом і будь-який файл з будь-якою інформацією. Також з часом росла і швидкість передачі даних через мережу Інтернет: якщо у 1980-х роках максимальна швидкість передачі даних становила менше 1 кілобайта в секунду, то в 2010-х роках вона становила вже понад 100 мегабайт в секунду [10].

У XX ст. було запущено всесвітню мережу інтернет, завдяки якій люди отримали можливість обмінюватися даними незалежно від того, де знаходиться їх співрозмовник, якщо у нього є вихід до цієї самої мережі.

Однією з основних цілей мережі інтернет було саме спілкування між користувачами. Тому, з часом почали створюватися сайти, на яких люди могли

задавати запитання та інші люди давали на них відповідь, сайти де можна було обговорювати останні новини та теми. Такі сайти називалися форумами. Саме таку назву вони мали не просто так: в давньому Римі це слово означало велику площу в центрі міста, де збиралися люди для обговорення важливих тем, святкування великих днів та звершення суду над злочинцями. Час минув і тепер веб-сайти стали цією площею, а народом – інтернет-користувачі.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Веб-сайти для спілкування існують стільки ж, скільки й сама мережа Інтернет. Їх можна розділити на різні типи:

- форуми за конкретними інтересами;
- форуми для обговорення різних тем;
- соціальні мережі;
- сайти новин.

Форуми за конкретними інтересами відрізняються тим, що будь-який користувач цього форуму може розпочати обговорення теми, але за якимось конкретним напрямком, заздалегідь визначений тематикою цього сайту.

Наприклад, навчаючись на якісь із спеціальностей напрямку інформаційних технологій, за статистикою, студенти користуються такими веб-сайтами, як: Cyberforum, Stackoverflow, Habr. Саме ці форуми можна вважати найвідомішими серед програмістів країн СНД. Вони створені для того, щоб програмісти-початківці могли читати створені досвідченішими програмістами статті по різних мовах програмування, щоб ці ж початківці могли задавати запитання і отримувати на них відповіді, починаючи дискусію і обираючи з різних запропонованих варіантів. Також, відомими є форуми для обговорення конкретної гри. На таких форумах зазвичай гравці, які дізналися певні секрети або просто пройшли гру до кінця можуть ділитися всіма набутими знаннями та навичками з новачками цієї гри. Тому, можна зробити висновок, що цей тип сайтів об’єднує те, що більш

досвідчені в конкретній сфері користувачі діляться інформацією з менш досвідченими.

Форуми для обговорення різних тем відрізняються від попереднього типу форумів тим, що їх користувачі можуть почати обговорення будь-яких тем. Зазвичай, такі форуми відзначаються тим, що вони не ведуть серйозної модерації на своїх сайтах, таким чином користувачі можуть обговорювати навіть такі теми, які зазвичай видаляють і забороняють обговорювати. Найвідомішими серед таких сайтів є 4chan, 2ch, Pikabu, Reddit, Twitter. З часом, відкритість цих сайтів і досить слабка модерація нашкодили цим сайтам: наразі їх вміст – це переважно обговорення і поширення контенту для дорослих. Навіть всесвітньовідома соціальна мережа Twitter зараз має саме такий вміст. Це дуже легко відслідкувати, якщо переглянути список найпопулярніших наразі тегів у постах.

Тому, можна зробити висновок: якщо ви хочете створити сайт для обговорення різних тем з конкретними принципами і обмеженнями, потрібно встановити достатній рівень модерації, щоб фільтрувати контент, яким ділитимуться користувачі вашого сайту. Інакше користувачі, які бажають уникати специфічного контенту, будуть вимушені перестати користуватися форумом і дуже сильно звузиться коло обговорюваних тем та коло користувачів.

Соціальні мережі виділені окремим пунктом і взагалі присутні в цьому списку тому, що в соціальних мережах також відбуваються постійні обговорення, але виділені окремим пунктом тому, що в звичайних форумах не акцентується увага на профілях користувачів, на сторонньому контенті, такому як ігри, додатки та музика. І основна різниця полягає в тому, що соціальні мережі в першу чергу орієнтовані на спілкування в приватних повідомленнях і створених бесідах із заздалегідь визначеними користувачами мережі.

Найвідомішими представниками таких сайтів є Facebook, Instagram, Tumblr, LinkedIn. Також до цього списку можна включити мережі ВКонтакте та Одноклассники, які є одними з найпопулярніших в країнах СНД, але є невідомими в інших країнах. Причина, чому Twitter відноситься саме до форумів, а не до соціальних мереж, випливає із низького акценту на вище названих ознаках

соціальної мережі, а саме додатковому контенті (музика, ігри, додатки) та приватних повідомленнях.

Сайти новин не являються форумами чи соціальними мережами по їх звичайному визначенню, але вони також являються сайтами, де відбуваються обговорення різних тем, висвітлених у статтях та репортажах, написаних на цих сайтах, тому їх можна вважати окремим видом сайтів для комунікації. Зазвичай такі сайти є аналогами якихось інших засобів масової інформації, наприклад: телевізійна передача «Телевізійна служба новин» (ТСН) телеканалу 1+1 має свій аналог – веб-сайт компанії. Проте є й звичайні сайти, на яких поширюються новини. Одним з таких є сайт спортивних новин «СПОРТ.ЮА», на якому також проходять обговорення всіх подій, описаних у статтях.

Веб-сайт, який є темою цієї дипломної роботи, належить до другої групи форумів, а саме до форумів для обговорення різних тем. Тому, зважаючи на особливості цього типу сайтів, необхідно створити належну модерацію і знайти баланс в цензурі контенту.

1.3 Визначення вимог до програмного забезпечення та розробка ТЗ

Функціонал та всі вимоги майбутнього застосунку описано, користуючись діаграмою варіанті використання.

В таблиці 1.1 описано всіх акторів майбутньої діаграми (тобто всіх користувачів виконуваного продукту). В таблиці 1.2 описані безпосередньо самі варіанти використання (ВВ).

Таблиця 1.1 – Опис акторів

Актор	Опис
1	2
Неавторизований користувач	Має можливість переглядати усі створені на сайті теми, виконувати пошук по темах сайту за тегами, назвою, автором, а також реєструватися для розширення доступного функціоналу.

Кінець таблиці 1.1

1	2
Авторизований користувач	Має можливість переглядати усі створені на сайті теми, теми, додані ним до своїх улюблених, рекомендовані за допомогою тегів інтересів користувача теми, створювати власні теми, коментувати існуючі теми, реагувати («+» або «-») на існуючі теми та коментарі під ними, редагувати власний профіль та створені ним теми, коментувати теми, редагувати і видаляти написані цим користувачем коментарі, змінювати теги інтересів, переглядати створені ним теми, переглядати теми, в обговореннях яких він приймав участь, додавати теми до списку улюблених, видаляти і редагувати власні теми, а також виконувати пошук по темах сайту за тегами, автором та назвою, переглядати власний рейтинг на сайті. Для допущення користувача до цього функціоналу йому необхідно авторизуватися через сторінку логіну.
Адміністратор	Має доступ до перегляду всіх тем, а також до їх видалення, видалення коментарів та блокування окремих користувачів. Відповідно, має доступ до перегляду видалених тем, коментарів та заблокованих користувачів.

Таблиця 1.2 – Опис варіантів використання

Актор	Назва ВВ	Опис ВВ
1	2	3
Неавторизований користувач	Перегляд усіх тем	Користувач може переглядати усі теми, створені іншими користувачами на цьому сайті.
	Пошук тем	Користувач може виконувати пошук серед усіх тем сайту за такими фільтрами, як теги теми, автор теми та її назва.
	Реєстрація	Користувач має можливість зареєструватися і створити власний профіль на сайті для розширення доступного йому функціоналу, заповнивши спеціальну реєстраційну форму.
	Авторизація	Для доступу до усіх вище названих функцій, користувачу необхідно успішно пройти процедуру авторизації на сторінці логіну.
Авторизований користувач	Перегляд усіх тем	Користувач може переглядати усі теми, створені іншими користувачами на цьому сайті.
	Пошук тем	Користувач може виконувати пошук серед усіх тем сайту за такими фільтрами, як теги теми, автор теми та її назва, одразу додавати таку тему до улюблених, реагувати на неї.

Продовження таблиці 1.2

1	2	3
	Перегляд улюблених тем	Користувач може переглядати усі теми, які він раніше додавав до списку улюблених.
	Перегляд рекомендованих тем	Користувач може переглядати теми, які йому рекомендує система за допомогою тегів інтересів цього користувача.
	Перегляд обговорених тем	Користувач може переглядати усі теми, в обговореннях яких він приймав участь (коментував).
	Перегляд власних тем	Користувач може переглядати усі створені ним теми.
	Перегляд профілю	Користувач може переглядати власний профіль з усією наданою раніше особистою інформацію.
	Перегляд тегів інтересів	Користувач може переглядати усі внесені ним раніше теги, за допомогою яких система підбираю користувачу рекомендовані теми.
	Створення теми	Користувач має можливість створити власну тему: обрати їй назву, позначити необхідними тегами, та описати саму тему.
	Редагування теми	В будь-який момент користувач може редагувати назву, опис та теги власної теми.
	Видалення теми	За необхідності, користувач може видалити власну тему.
	Редагування тегів інтересів	Користувач може змінити теги, за допомогою яких система підбирає йому рекомендовані теми.
	Створення коментаря	Користувач може написати коментар під будь-якою темою.
	Редагування коментаря	Користувач може редагувати написаний ним коментар.
	Видалення коментаря	Користувач може видалити створений ним коментар.
	Реакція на тему	Користувач може позитивно або негативно відреагувати на будь-яку тему.
	Реакція на коментар	Користувач може позитивно або негативно відреагувати на будь-який коментар.
	Додати до улюблених	Користувач може додати будь-яку (окрім власної) тему до списку улюблених.
	Перегляд рейтингу	Користувач може переглядати власний рейтинг на сайті, обрахований системою за власними критеріями.
Адміністратор	Перегляд усіх тем	Користувач може переглядати усі теми, створені іншими користувачами на цьому сайті.
	Видалення коментаря	Користувач може видалити коментар будь-якого користувача, якщо він не відповідає правилам сайту.

Кінець таблиці 1.2

1	2	3
	Видалення теми	Користувач може видалити будь-яку тему будь-якого, якщо вона не відповідає правилам сайту.
	Блокування користувача	Користувач може заблокувати будь-якого користувача, якщо він вчиняв дії, що не відповідають правилам сайту визначену кількість разів.

Згідно з описаними вище варіантами використання, було побудовану діаграму варіантів використання, яку подано на рисунку 1.1

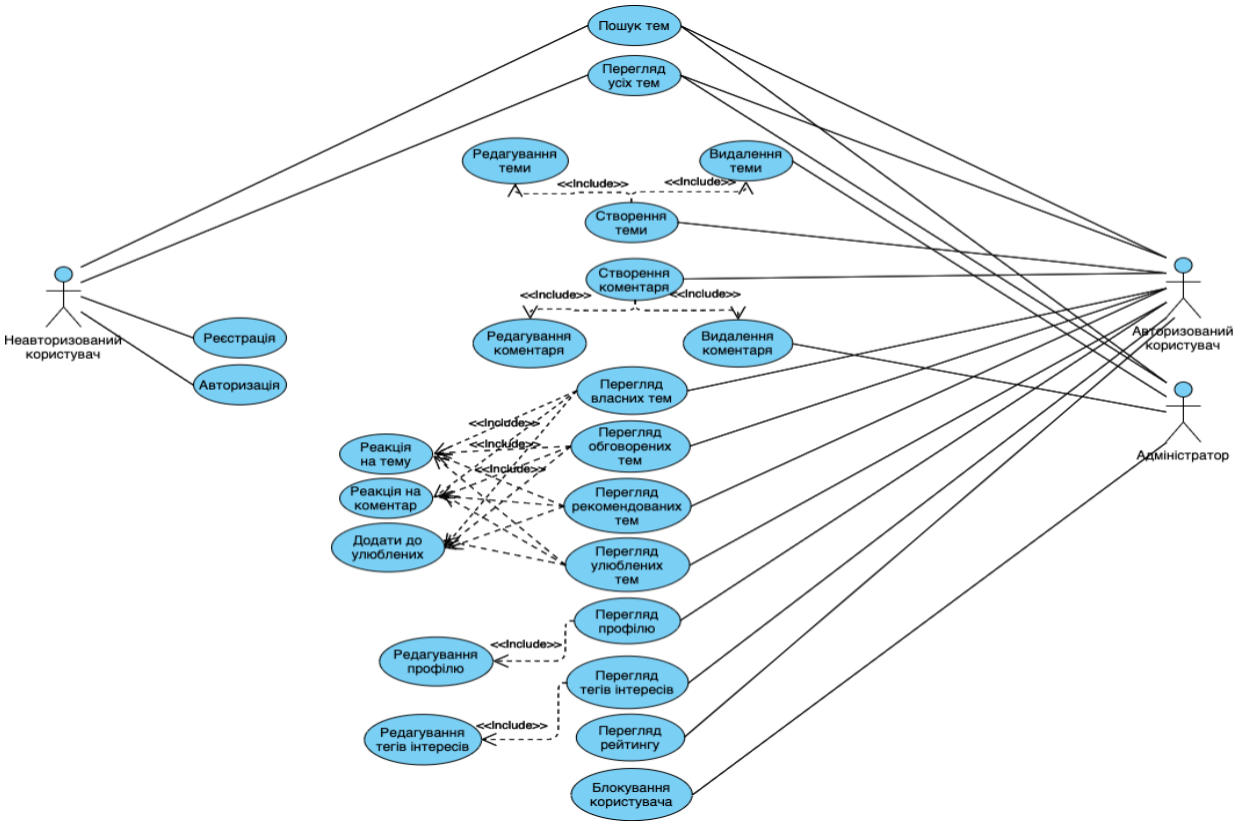


Рисунок 1.1 – Діаграма варіантів використання

Отже, в першому розділі було проаналізовано предметну область веб-платформи а також усі найпопулярніші з наявних аналогічних додатків, платформ та сервісів. При виконанні даного розділу було визначено увесь майбутній функціонал платформи та описано технічне завдання, розміщене у додатку А даного дипломного проекту.

2 ПРОЕКТУВАННЯ ВЕБ-ПЛАТФОРМИ

2.1 Архітектура та функціональна структура додатка

Під час проектування будь-якого додатку, розробник має безліч варіантів архітектури. І вибір правильної архітектури залежить від великої кількості факторів: призначення додатку, технологія або технології, які використовуються для його реалізації, потенційний розмір проекту та кількість потенційних варіантів розширення його функціоналу.

Темою даного дипломного проекту є веб-платформа. Як і будь-який інший вид програмного забезпечення, веб-платформа має свої поширені варіанти архітектури програмного забезпечення. Найпопулярнішими з них є MVC (TLA) [11], MVVM [12] та TTA [13]. Оскільки кожен з них має свої значні відмінності, необхідно детально розглянути їх окремо.

MVC (Model-View-Controller) – реалізація архітектурного підходу TLA-типу (Three Layer Architecture). Він передбачає собою логічне розділення програмного забезпечення на три основних шари, як це виходить з назви цього шаблону: View, Controller та Model.

Представлення (View) – шар шаблону проектування, який являє собою відображення користувацького інтерфейсу, тобто його лицьову частину. Якщо розглядати використання цього шаблону в рамках веб-платформи, зазвичай цей шар містить в собі HTML-розмітку та таблиці стилів (CSS, SCSS та інші).

Модель (Model) – шар шаблону проектування, який відповідає за представлення моделі даних, збереження, зміну і діставання цих даних. Тобто іншими словами, це шар який працює безпосередньо з базою даних.

Контролер (Controller) – шар шаблону проектування, який відповідає за всю бізнес-логіку програмного забезпечення, тобто це шар, який поєднує між собою дії користувачів (представлення) та дані, з якими вони можуть працювати (модель). Зазвичай в ньому знаходяться класи, які відповідають за отримання інформації від моделей, її обробки, передачу на представлення. Також, якщо передбачається такий функціонал, представлення може передавати деяку

інформацію контролеру, який її оброблятиме за необхідними алгоритмами та передаватиме в модель для подальшого зберігання цих даних.

Узагальнену схему поведінки даного шаблону можна спостерігати на рисунку 2.1.

Рисунок 2.1 – Схема шаблону проектування MVC

Оскільки при виконанні дипломного проекту одна з мов програмування, яку я використовуватиму є мова програмування C#, важливо підмітити, що ця мова має окремий набір технологій для можливості реалізації проектів з таким типом архітектури. Так, ASP.NET MVC та ASP.NET MVC Core є фреймворками, створеними для зручної роботи з такими проектами.

Хоч сам шаблон проектування виглядає абсолютно логічним і ідеально продуманим, з логічним розділенням основних обов'язків, він має деякі недоліки, які буде описано після аналізу і порівняння з шаблоном проектування TTA.

TTA (Three Tier Architecture) – шаблон проектування, який передбачає логічний і фізичний розподіл проекту на три рівні: рівень представлення (Presentation Tier), рівень логіки (Logic Tier) та рівень даних (Data Tier).

На перший погляд, судячи з назви шаблону та логіки його розподілу на окремі частини, можна зробити висновок, що TLA та TTA – це однакові шаблони проектування. Проте це не так [20].

Слово «layer» у назві TLA означає лише логічний розподіл цих «шарів». А це означає, що весь код проекту, всі його класи, інтерфейси та інші файли будуть, швидше за все, знаходитися в одному проекті.

Натомість, слово «tier» у назві TTA означає не лише логічне, але й фізичне розділення проекту на окремі «рівні», що в свою чергу означає відокремлення бізнес-логіки проекту від бази даних і представлення на стільки, що вони можуть

навіть знаходитися на різних серверах і взаємодіяти з великою кількістю інших систем.

Ця різниця є дуже значною, оскільки за допомогою такого розділення, яке пропонує ТТА, можна набагато легше реалізувати принципи SOLID, значно зменшити зв’язність не лише між цими рівнями, але й різними елементами цих рівнів.

Важливим є також те, що проектування за таким підходом полегшує подальшу підтримку та розширення проекту, оскільки кожна з частин усіх рівнів є максимально абстрагованою від інших і підміна однієї з них чи додавання нової ніяк не повпливає на працездатність системи.

Ще однією з переваг такої архітектури програмного забезпечення є значно простіше тестування як за допомогою unit-тестів, так і за допомогою інтеграційних тестів. В тому числі, такий підхід до розподілу проекту на частини дозволяє тестувати кожну з них незалежно від іншої, значно простіше підміняючи реалізації.

На рисунку 2.2 зображена загальна схема даного шаблону проектування.

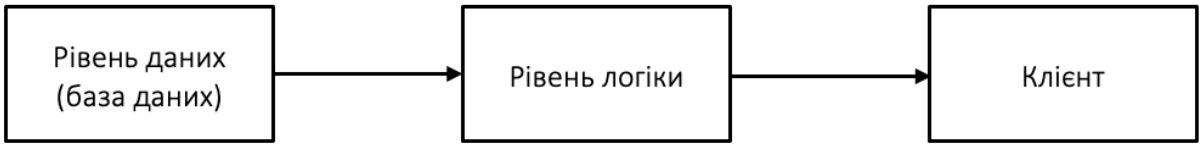


Рисунок 2.2 – Схема шаблону проектування ТТА

MVVM (Model-View-View Model) – шаблон проектування, який також передбачає логічне розділення проекту на три частини: модель (Model), представлення (View) та модель представлення (View Model).

Суть та призначення моделі і представлення не змінюються відносно описаних раніше шаблонів проектування, проте з’являється новий елемент – модель представлення.

Модель представлення (View Model) – це шар шаблону проектування, який дозволяє зв’язувати представлення не з інтерфейсом, а з певною моделлю, яка описуватиме її. Це необхідно для того, щоб звільнити контролер представлення

від коду, необхідного для взаємодії з моделлю, оскільки тепер саме модель представлення відповідає за взаємодію користувача з даними, а контролер містить в собі іншу бізнес-логіку.

Цей шаблон має значну перевагу в універсальності написаних моделей представлення, оскільки одна модель представлення може взаємодіяти з великою кількістю різноманітних представлень, дані якої може вмістити і описати. Таким чином, розробник звільняє себе від написання великої кількості однакового коду.

Ще однією з переваг цього шаблону є те, що він дозволяє зручно описувати та користуватися масивними контролерами з великою кількістю бізнес-логіки, оскільки все, що не стосується бізнес-логіки, винесене в модель представлення, і не займає зайвого місця та робить код легшим для читання та розуміння.

Також необхідно зауважити, що проекти на архітектурі MVVM є більш зручними для unit-тестування.

На рисунку 2.3 зображена загальна схема поведінки шаблону проектування MVVM.

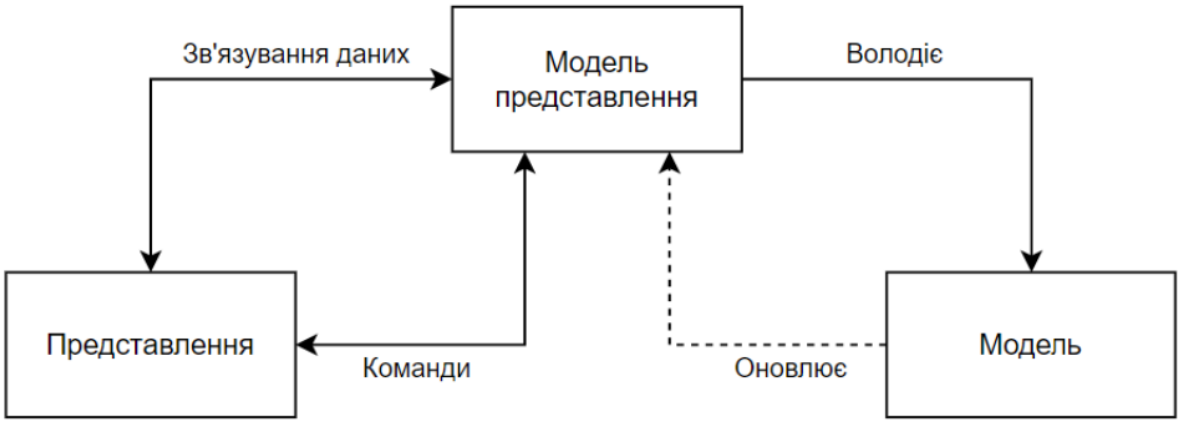


Рисунок 2.3 – Схема шаблону проектування MVVM

Отож, підсумовуючи перераховані варіанти шаблонів проектування, для розробки даного програмного забезпечення було обрано шаблон Three Tier Architecture. Вибрано саме цей шаблон завдяки зручності розподілу всіх його частин, легшому розширенню функціонала при такому проектуванні, та зручній подальшій довготривалій підтримці.

Також при виконанні даного дипломного проекту необхідно виконувати тестування, яке значно зручніше виконувати при використанні саме цього шаблону проектування.

2.2 Опис структури даних та моделі бази даних

База даних додатку зберігатиме в собі інформацію про користувачів, теми, коментарі, теги, реакції на теми та коментарі, скарги на користувачів, теми та коментарі, теги інтересів кожного користувача та улюблені теми кожного користувача.

Також, для підвищення ефективності усі зв’язки між таблицями буде винесено в окремі проміжні таблиці, такі як: скарги на коментарі, скарги на теми, скарги на користувачів, коментарі теми, теги теми, реакції на теми, реакції на коментарі, коментарі користувача та теми користувача.

Отже, база даних додатку матиме такі основні сутності:

- User (Зберігає дані про користувача);
- Topic (Зберігає інформацію про створену користувачем тему);
- Commentary (Зберігає інформацію про створений коментар);
- Reaction (Зберігає інформацію про реакцію, залишену користувачем на коментарі або темі);
- Tag (Зберігає інформацію про тег);
- Report (Зберігає інформацію про скаргу, подану на коментар, тему чи користувача);
- FavoriteTopic (Зберігає інформацію про користувача та його улюблені теми);
- UserTag (Зберігає інформацію про користувача та його теми інтересів).

Також, база даних міститиме наступні проміжні сутності:

- UserTopic (Зв’язує інформацію про користувача та його теми);

- UserCommentary (Зв’язує інформацію про користувача та його коментарі);
- UserReport (Зв’язує інформацію про скарги на користувача);
- UserTopicReaction (Зв’язує інформацію про реакції користувача на теми);
- UserCommentaryReaction (Зв’язує інформацію про реакції користувача на коментарі);
- TopicReport (Зв’язує інформацію про скарги на тему);
- TopicCommentary (Зв’язує інформацію про коментарі теми);
- TopicTag (Зв’язує інформацію про теги теми);
- CommentaryReport (Зв’язує інформацію про скарги на коментар).

Розглянемо кожну з цих сутностей більш детально.

Сутність User має наступні атрибути:

- Id (первинний ключ);
- Username (ім’я користувача, під яким його бачитимуть усі інші користувачі);
- Email (електронна пошта користувача);
- Role (роль користувача);
- PasswordHash (хеш пароля користувача);
- Rating (рейтинг користувача);
- PhotoUrl (посилання на місце збереження фотографії профілю користувача);
- Status (статус профіля користувача).

Сутність Topic містить такі атрибути:

- Id (первинний ключ);
- Title (заголовок теми);
- CreatedUtc (час створення теми);
- Status (статус теми).

Сутність Commentary містить наступні атрибути:

- Id (первинний ключ);

- Text (текст коментаря);
- CreatedUtc (час створення коментаря);
- Status (статус коментаря).

Сутність Report містить такі атрибути:

- Id (первинний ключ);
- Type (тип скарги);
- Description (опис скарги).

Сутність Reaction має такі атрибути:

- Id (первинний ключ);
- ReactionValue (значення реакції);
- CreatedUtc (час створення реакції).

Сутність Tag містить наступні атрибути:

- Id (первинний ключ);
- Text (текст тега);
- UsedCount (кількість разів, коли тег відмічали в темах);
- Status (статус тега).

Сутність FavoriteTopic має такі атрибути:

- UserId (первинний ключ користувача);
- TopicId (первинний ключ теми).

Сутність UserTag містить наступні атрибути:

- UserId (первинний ключ користувача);
- TagId (первинний ключ тега).

Сутність UserTopic містить наступні атрибути:

- UserId (первинний ключ користувача);
- TopicId (первинний ключ теми).

Сутність UserCommentary містить наступні атрибути:

- UserId (первинний ключ користувача);
- CommentaryId (первинний ключ коментаря).

Сутність UserReport містить наступні атрибути:

- UserId (первинний ключ користувача);

- ReportId (первинний ключ скарги).

Сутність UserTopicReaction містить наступні атрибути:

- UserId (первинний ключ користувача);
- TopicId (первинний ключ теми);
- ReactionId (первинний ключ скарги).

Сутність UserCommentaryReaction містить наступні атрибути:

- UserId (первинний ключ користувача);
- CommentaryId (первинний ключ коментаря);
- ReactionId (первинний ключ скарги).

Сутність TopicReport містить наступні атрибути:

- TopicId (первинний ключ теми);
- ReportId (первинний ключ скарги).

Сутність TopicCommentary містить наступні атрибути:

- TopicId (первинний ключ теми);
- CommentaryId (первинний ключ коментаря).

Сутність TopicTag містить наступні атрибути:

- TopicId (первинний ключ теми);
- TagId (первинний ключ тега).

Оскільки одна і та ж тема може бути в списку улюблених у багатьох користувачів, між таблицями користувачів і тем налаштовано зв’язок «багато-до-багатьох» за допомогою проміжної таблиці FavoriteTopics. Ідентична ситуація і з сутністю UserTag – один і той же тег може бути в списку інтересів одразу у багатьох користувачів. Саме тому необхідно налаштувати звя’зок «багато-до-багатьох» та створити для цих цілей проміжну сутність UserTag.

Наступні сутності бази даних є проміжними для забезпечення між відповідними сутностями зв’язків «один-до-багатьох»:

- UserTag (проміжна сутність між сутностями User та Tag);
- UserTopic (проміжна сутність між сутностями User та Topic);
- UserCommentary (проміжна сутність між сутностями User та Commentary);

- UserReport (проміжна сутність між сутностями User та Report);
- UserTopicReaction (проміжна сутність між сутностями User, Topic та Reaction);
- UserCommentaryReaction (проміжна сутність між сутностями User, Commentary та Reaction);
- TopicReport (проміжна сутність між сутностями Topic та Report);
- TopicCommentary (проміжна сутність між сутностями Topic та Commentary);
- TopicTag (проміжна сутність між сутностями Topic та Tag).

В результаті аналізу та проектування бази даних, було розроблено логічну схему бази даних, зображену на рисунку 2.4.

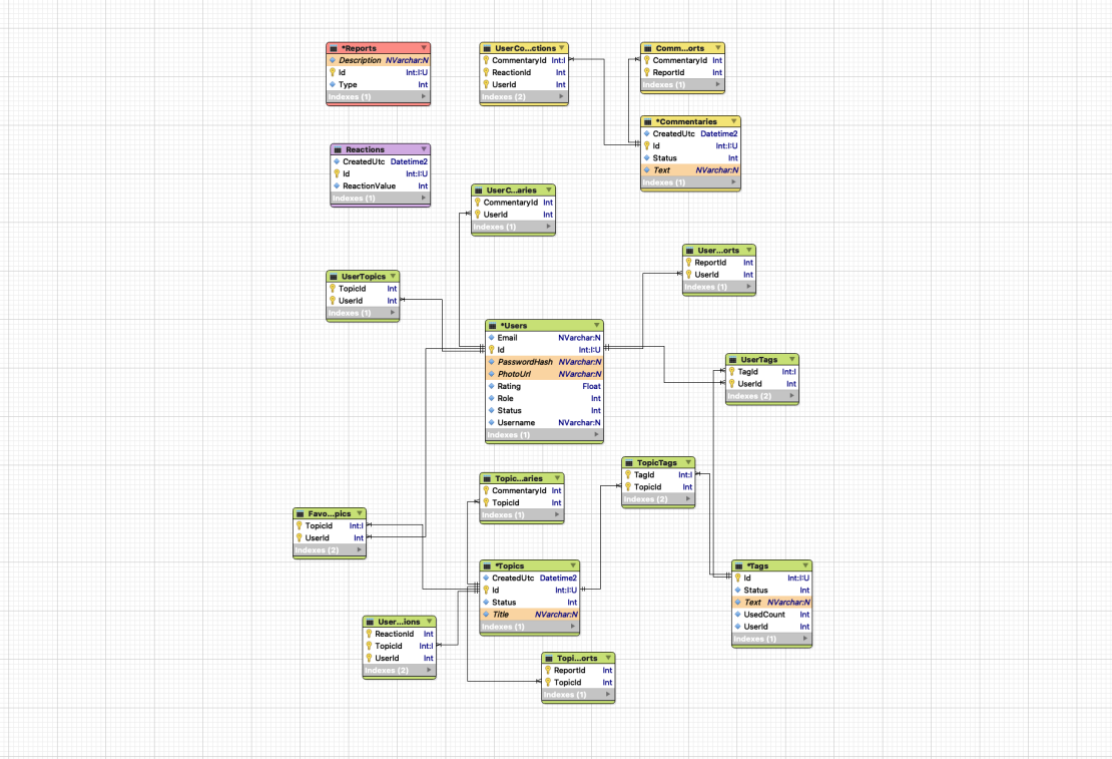


Рисунок 2.4 – Логічна схема бази даних

2.3 Проектування серверної частини веб-додатка

Після проведення попереднього аналізу функціональної структури додатка та проаналізувавши всі найкращі практики в проектуванні таких додатків [21], було визначено основний модуль, який в свою чергу розділяється на кілька під модулів. Таким чином, було виведено таку структуру модулів серверної частини додатка:

а) серверний рівень;

1) рівень доступу до даних;
2) рівень бізнес-логіки;
3) рівень мережевої взаємодії з рівнем клієнтського додатку;

Оскільки при створенні серверного рівня додатку використовується шаблон проектування TTA, кожен з під модулів реалізований через призму максимальної абстрактності, що є важливим аспектом для уможливлення подальшого розвитку платформи, розширення функціоналу та підміни реалізацій. Також, така абстрактність кожного модуля, в подальшій перспективі, робить легшою взаємодію великої кількості розробників, оскільки кожен з них використовуватиме інтерфейс взаємодії інших модулів не знаючи нічого про їхню реалізацію, і чи існує вона взагалі.

Таким чином, розглянемо детальніше кожен із трьох модулів серверної частини веб-додатка.

Рівень доступу до даних являється базовим для серверного модуля, оскільки даний додаток не може працювати без збереження даних в базу даних та її подальшої обробки. В першу чергу цей рівень представлений контекстом, за допомогою якого виконується безпосередня взаємодія з базою даних за допомогою різних технологій, які надають розробники платформи .NET Core, такими як Entity Framework Core [22], LINQ-запити [23] та інші. Також, цей рівень представлений великою кількістю моделей, які виступають описом усіх сутностей, необхідних для взаємодії бази даних з рівнем доступу до даних та в подальшому з рівнем бізнес-логіки. Рівень доступу до даних повинен містити у

собі інтерфейси взаємодії з кожною із представлених сутностей, а також імплементувати їх.

Наступним модулем є модуль бізнес-логіки. Він представлений сутностями, які описують та імплементують усю логіку для коректного збереження і зв'язування між собою усіх даних.

Також у даному модулі будуть описані сервіси для шифрування деяких чутливих даних, таких як паролі користувачів. Для шифрування буде використано алгоритм AES, описаний раніше інженерами корпорації Microsoft, які працювали над створенням платформи .NET, зокрема бібліотеки System.Security.Cryptography, яка містить у собі велику кількість описаних алгоритмів шифрування. Серед усіх представлених класів, для використання при створенні даного додатку було обрано клас Rfc2898DeriveBytes [24].

В додатках такого типу важливим елементом безпеки також є розпізнавання і надання доступу користувачам до певно інформації, та обмеження доступу до іншої інформації. Саме тому при розробці додатку буде використано технологію JWT (JSON Web Tokens) [25], яка дозволяє легко та точно розпізнавати користувачів, надавати їм доступ при коректному поводженні з системою та обмежувати доступ при некоректному поводженні з системою. Завдяки цій технології можна обмежувати час авторизації користувачів у випадку, якщо користувач перестає користуватися системою, та авторизуватися з різних пристроїв для більшої зручності.

Для реалізації авторизації за допомогою саме цієї технології необхідно також використовувати сервіси з кешування даних для підвищення ефективності пошуку, вставки та видалення JWT-токенів. Зазвичай ці сервіси мають функцію автоматичного видалення даних після проходження певної, визначеної користувачем, кількості часу, що значно полегшує роботу технології авторизації, оскільки якщо користувач і матиме певний токен, він існуватиме тільки певний період часу, і після проходження цього часу, токен перестане існувати і при спробі доступу до інформацію за допомогою цього токена, він не пройде валідацію і доступ не буде надано.

Також це певною мірою забезпечує користувачів від зловмисників, які також не матимуть доступу до інформації довше, ніж існуватиме цей токен. Проте, час існування токена може бути довгим, і зловмисники можуть мати доступ до особистої інформації користувача достатньо довго. Для вирішення цієї проблеми при проектуванні було вирішено зберігати з токеном також хеш пароля користувача. Таким чином, після зміни пароля користувачем, викрадений токен моментально перестане бути валідним доступ зловмисників до інформації буде закритий.

Усі необхідні класи і сервіси для коректної роботи цієї технології буде описано в модулі бізнес-логіки додатку.

2.4 Проектування клієнтської частини веб-додатка

Так само як і серверна частина, клієнтська частина складається з одного основного модуля і кількох похідних:

а) рівень клієнтського додатку;

1) рівень мережевої взаємодії із серверним рівнем;
2) рівень користувацького інтерфейсу.

При створенні клієнтського модуля, його не обов’язково розділяти на менші модулі ні фізично, ні логічно. Проте, це полегшує командну розробку додатку і дозволяє легше увійти в розробку і розширення додатку новими розробниками.

Рівень мережевої взаємодії з рівнем клієнтського додатку необхідний для створення з’єднання між серверною та клієнтською частиною додатку. Таким чином, сервер стає централізованою системою, яка самостійно керує всіма рухами даних, забезпечуючи доступ до цих даних, а клієнтом для користуванням цими даними може стати будь-який пристрій, який має браузер. Цей модуль містить в собі контролери, які забезпечують коректне отримання і обробку запитів, а також надсилання відповіді клієнту.

Окрім цього він використовує різне проміжне програмне забезпечення, яке дозволяє валідувати інформацію, яка приходить на сервер та одразу відкидати запити з некоректними даними, що значно підвищує ефективність і швидкість роботи сервера.

Рівень мережевої взаємодії клієнтського рівня із серверним рівнем є базовим модулем клієнтського рівня додатку. Він повинен описувати сервіси для коректного надсилання запитів та обробки отриманих відповідей, а також взаємодіяти з інформацією, отриманою від сервера або користувача. Важливими є сервіси для перевірки авторизації користувачів та проведення цієї авторизації, оскільки при неправильній роботі вони завдаватимуть значного удару по іміджу додатку та приноситимуть незручності користувачам, а також зможуть стати причиною витоку чутливих даних та надати доступ до даних користувача зловмисникам, які згодом зможуть використати цю інформацію проти користувача.

Рівень користувацького інтерфейсу є рівнем, який безпосередньо бачить кожен користувач додатку. В цьому модулі повинні знаходитися файли сторінок та компонентів сайту, таблиці стилів.

На усіх сторінках передбачається спільний компонент для зручної навігації по сайту.

Невід’ємними є також сторінки авторизації та реєстрації користувачів, які буде об’єднано для більшої зручності і зменшення кількості сторінок, що дозволить зробити систему більш компактною.

Оскільки кожен користувач при реєстрації вводить певну особисту інформацію, він повинен мати змогу згодом редагувати її, тому необхідним є також створення сторінки профілю користувача.

Метою даного додатку є створення середовища для дискусії, тому буде створено сторінки для групування тем за різними критеріями та сторінка для пошуку тем за автором, назвою та тегами. Також буде створено сторінку для написання нових тем.

2.5 Створення макета веб-додатка та дизайн

Для оформлення дизайну клієнтської частини додатка було вибрано чотири основних кольори: два відтінки сірого, які буде використано для створення фону безпосередньо сайту та тем, жовто-зелений колір, який буде використано як колір шрифту у деяких випадках та як основний колір деяких елементі користувацького інтерфейсу та білий колір, який буде використаний як основний колір тексту та, в деяких ситуаціях, фоновий колір деяких елементів користувацького інтерфейсу.

В першу чергу розробимо дизайн сторінки авторизації та реєстрації. Їх дизайн є надзвичайно важливим, оскільки зазвичай це перше, що бачитиме користувач при вході на сайт, і від його вражень, які базуватимуться в тому числі і на цих побачених сторінках, формуватиметься думка про додаток і доцільність його використання. У більшості сайтів функції авторизації та реєстрації розділяють на різні сторінки і все, що приносить це розділення, це лише додаткова сторінки, між якими треба перемикатися у разі зміни способу входу на сайт. Тому було обрано варіант об'єднання цих функцій на одній сторінці. В результаті аналізу та проектування, було визначено зовнішній вигляд сторінки входу на сайт, тобто сторінки авторизації та реєстрації, зображеної на рисунку 2.5.

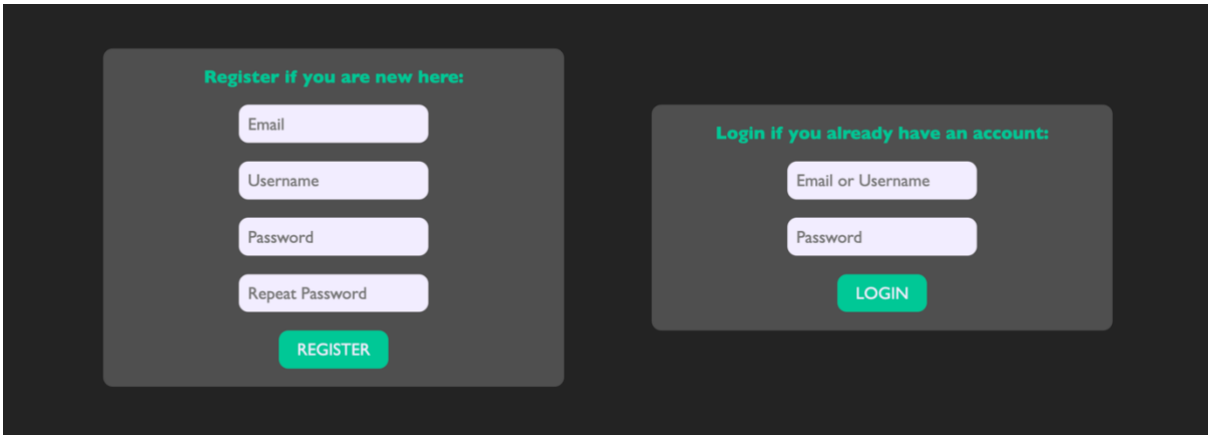


Рисунок 2.5 – Дизайн сторінки входу у систему сайту

Наступною сторінкою, яку бачитиме користувач після входу на сайт, буде сторінка з відображеними темами, попередньо відібраними за окремими групами,

наприклад улюблені, рекомендовані, обговорені теми. Єдиним важливим елементом на цій сторінці повинна бути тема. Тому дуже важливим є знайти баланс між тим, щоб додати туди максимальну кількість інформації і функціоналу та між простотою і невимушеністю. В результаті аналізу та проектування, було визначено зовнішній вигляд сторінки з переліком тем, зокрема дизайн самого елемента з інформацією про тему та деякими функціями для взаємодії з темою. Дизайн зображено на рисунку 2.6.

Рисунок 2.6 – Дизайн елемента з інформацією про тему

Важливим є дизайн місця, де проходитиме безпосереднє обговорення теми: повідомлення повинні легко розрізнятися, розділятися, на них повинно бути не занадто багато інформації та інше. Загалом є безліч додатків та сайтів, які реалізовували схожі техніки, тому при розробці дизайну буде проаналізовані усі найкращі практики та обрано найкращий варіант для максимальної зручності та привабливості даної сторінки і кожного її елемента окремо.

Отже, після проведення аналізу та проектування, було розроблено дизайн сторінки, на якій проходитиме безпосереднє обговорення теми. На сторінці необхідно зобразити плитку теми за таким самим шаблоном, за яким вона зображена на сторінці із переліком тем. Також необхідно вирівняти коментарі користувачів по різних краях, залежно від належності коментаря даному користувачу, чи будь-якому іншому. Коментарі поточного користувача необхідно виділити світло-зеленим кольором, а коментарі іншого користувача – темно-зеленим. У випадку, якщо користувач є неавторизованим, усі коментарі буде вирівняно по лівому краю, як коментарі інших користувачів та виділено темно-зеленим кольором. Результат зображено на рисунку 2.7.

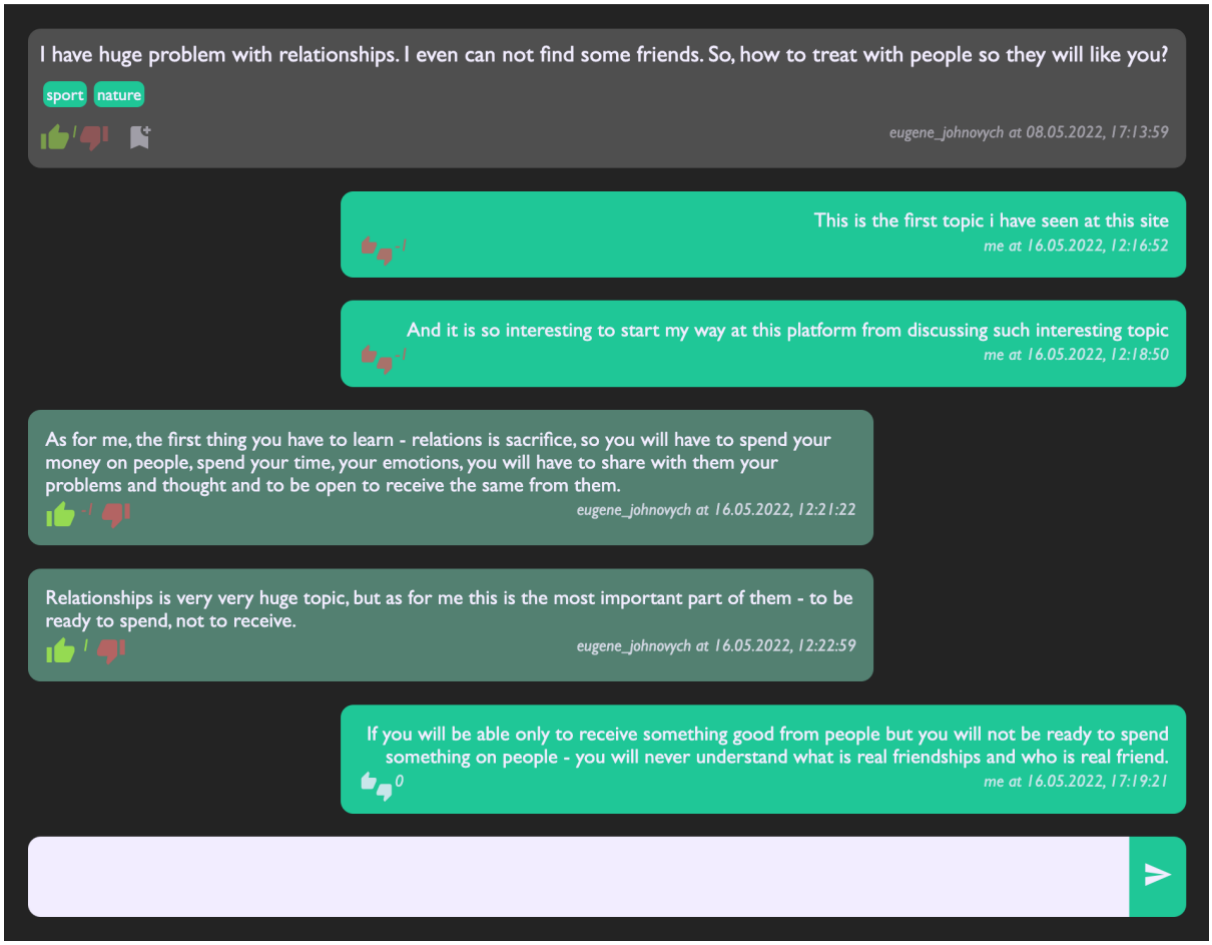


Рисунок 2.7 – Дизайн сторінки для обговорення теми

Наступною сторінку для проектування буде сторінка створення нової теми. Вона також не повинна містити жодної зайвої інформації – лише поля для введення даних та кнопка для підтвердження відправлення запиту. Отже, в результаті проведення аналізу та проектування було розроблено дизайн сторінки для створення нової теми. Результат зображено на рисунку 2.8.

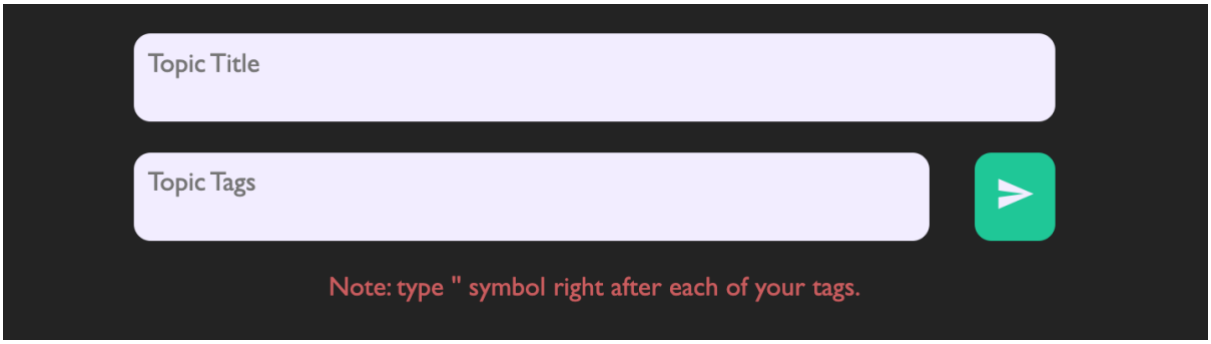


Рисунок 2.8 – Дизайн сторінки створення нової теми

2.6 Аналіз та вибір технологій і методів реалізації веб-додатка

Оскільки розробка даного додатка розділяється на два етапи: розробка серверної частини та розробка клієнтської частини, технології для них також треба використовувати різні.

Отже, для розробки серверної частини додатку в першу чергу використовуватиметься платформа .NET Core [26].

Оскільки серверна частина додатку буде написана на мові програмування C# за використання платформи .NET Core, досить зручною для використання буде саме база даних MS SQL Server, оскільки дана мова програмування та її платформа має деякі зручні інструменти для взаємодії.

Зокрема, для зручної роботи з базою даних використовуватиметься технологія Entity Framework Core та її похідні бібліотеки.

Для зручної побудови запитів в базу даних буде використовуватися технологія LINQ.

Для авторизації та автентифікації користувачів буде використано технологію JWT, яка дозволить легко захистити дані користувачів від втручання зловмисників.

Для швидкого збереження, отримання, редагування та видалення даних JWT буде використано технологію Redis. Цей сервіс має безкоштовний план з деякими обмеженнями, тому є зручним і вигідним для використання в умовах розробки.

Для шифрування та хешування чутливих даних користувачів буде використано алгоритм шифрування AES [27], один із способів якого описаний у класі Rfc2898DeriveBytes інженерами корпорації Microsoft, які і працювали над розробкою бібліотеки System.Security.Cryptography.

Для зручного надсилання тестових запитів на серверну частину буде використано технологію Swagger [28], оскільки її можна встановити для вашого проекту та налаштувати прямо в коді додатку після встановлення її бібліотек.

Для написання клієнтської частини додатка буде використана скриптова мова програмування JavaScript, а саме одна з її платформ – Vue.js [29]. Була обрана саме ця платформа у зв’язку з її легкістю у вивченні та простотою у порівнянні з іншими платформами JavaScript.

Також, при створення клієнтської частини додатка буде використано технологію Vue-x [30], яка дозволятиме зберігати та обробляти інформацію, необхідну для коректної роботи системи, прямо в браузері.

Для зручної маршрутизації, переадресації та переміщення між сторінками та адресами додатку буде використано технологію Vue-router [31].

Для швидкої та зручної валідації введених користувачем даних при реєстрації або авторизації, використовуватиметься технологія Vuelidate. Таким чином це значно підвищить ефективність роботи серверної частини додатку, оскільки деякі запити, які не потрібно відправляти на сервер для перевірки, буде перевірено на коректність ще на клієнтському рівні і відкинуто у разі помилок, про що користувач отримає сповіщення і вводитиме дані доки не введе їх коректно і тоді їх буде відправлено на обробку серверу. Це також значно знизить навантаження на сервер, що прискорить його роботу та зменшить трафік, що зробить утримування сервера в хост-компанії дешевшим і вигіднішим.

У результаті проведеного аналізу, було визначено основні модулі додатка та визначено їхнє призначення, сфера роботи та інше.

Також було спроектовано та визначено основні сторінки клієнтської частини додатка, визначено їх зовнішній вигляд та призначення.

Отже, у другому розділі було обрано майбутню структуру та архітектуру платформи, описано структуру моделей даних та структуру самої бази даних, обрано та спроектовано, згідно обраної архітектури, серверну та клієнтську частину платформи, спроектовано дизайн та розроблено макет клієнтської частини платформи. Також було проаналізовано та обрано усі технології, необхідні для реалізації платформи згідно усім раніше спроектованим вимогам.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка бази даних

Для розробки даного програмного забезпечення було обрано систему керування базами даних Microsoft SQL Server. Оскільки розробка проводилася на пристрої з операційною системою MacOS, розробляти додаток, який базується саме на цій системі керування базами даних без додаткового програмного забезпечення неможливо, оскільки ані MacOS не підтримує інструменти розробки цієї платформи, ані середовища розробки, такі як Visual Studio for Mac чи Visual Studio Code, не надають функціоналу чи додаткових інструментів для розробки з використанням таких технологій.

Для вирішення цієї проблеми було проведено аналіз наявного програмного забезпечення, яке надає можливість створити віртуальну машину для подальшого встановлення на неї Microsoft SQL Server та зручного зв'язку із сервером бази даних. В результаті проведеного аналізу було обрано декілька найвідоміших та найбільш поширених програмних застосунків, таких як Kubernetes [32], Docker [33] та VirtualBox [34].

Kubernetes – програмна система з відкритим вихідним кодом, створена для зручного автоматичного розгортання контейнеризованих додатків та їх подальшого масштабування і управління ними. Система була розроблена компанією Google у 2015 році, а вихідний код першої версії було опубліковано у 2014 році.

Сама система має досить складну архітектуру з великою кількістю компонентів, кожен з яких має окрему відповідальність і є невід'ємною частину усіх процесів. Найголовнішими серед них є вузол, модуль, том, контролер, оператори.

Вузол («node») – це окрема віртуальна або фізична машина, яка використовується для розгортання та запуску контейнерних додатків.

Модуль («pod») – базова одиниця, яка використовується при розгортанні та запуску контейнерного додатку всередині вузла. Використовується для

маршрутизації запитів та виділення кожному окремому додатку унікальної IP-адреси в межах даного кластера.

Том («volume») – загальний для усіх контейнерних додатків ресурс зберігання та користування даними в межах одного модуля.

Контролер («controller») – процес, який є загальним для усіх запущених вузлів та використовується для керуванням зміни стану різних компонентів системи. Усі запущені контролери зберігаються та виконуються в єдиному компоненті – Kubernetes Controller Manager.

Оператори («operators») - вид програмного забезпечення, який спеціалізується на додаванні до існуючих кластерів Kubernetes сервісів, таких як системи керування базами даних, системи моніторингу та системи кешування.

Сама система Kubernetes поділяється на два шари – шар керування кластерами та безпосередньо сам шар кластерів.

Підсистема керування кластерами складається з п’яти складових. Розглянемо їх детальніше.

Сервер API – невід’ємна складова системи, яка надає розробнику API у стилі REST. Використовується для зручного доступу до всіх функцій системи не лише зсередини, але й ззовні.

Менеджер контролерів (“controller manager”) – процес, який керує усіма наявними контролерами та взаємодіє з сервером API для виконання усіх команд.

Планувальник («scheduler») – надважливий елемент підсистеми керування, який вираховує, на якому вузлів повинен виконуватися конкретний модуль та займаєтсья розподілом навантаження так, щоб навантаження не перевищувало необхідних ресурсів.

Etcd – елемент підсистеми, який зберігає в собі конфігураційні дані всієї системи.

Kubectl – елемент підсистеми, який представляє інтерфейс командного рядка для зручного керування розробником усіх процесів системи.

На рисунку 3.1 зображено загальну схему архітектури системи Kubernetes.

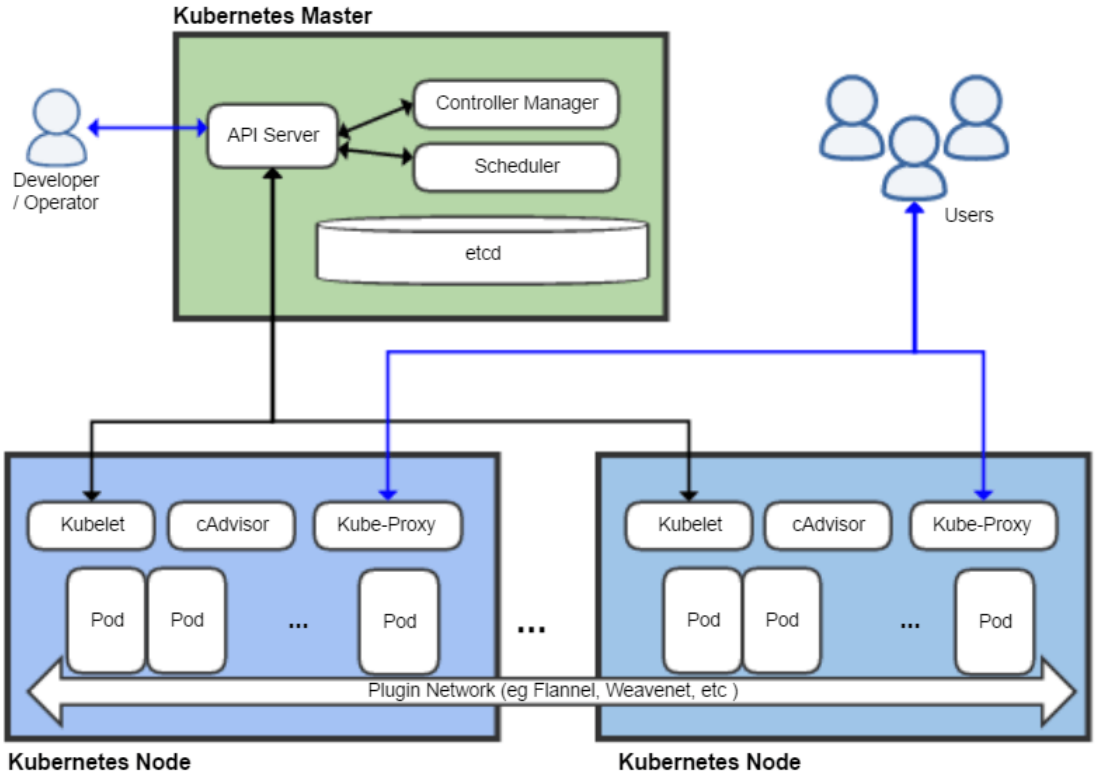


Рисунок 3.1 – Архітектура системи Kubernetes

VirtualBox – програмне забезпечення, створене для віртуалізації операційних систем на інших операційних системах. Таким чином, дана система дає можливість встановити та розгорнути на вашому пристрої будь яку іншу операційну систему та оперувати нею в будь-яких межах, які надає конкретна розгорнута операційна система.

Архітектура, функціонал та призначення даного програмного забезпечення значно відрізняється від системи Kubernetes, проте, все ще надає можливість для розгортання системи керування базою даних Microsoft SQL Server.

Docker – програмне забезпечення, що дозволяє віртуалізувати необхідні контейнерні додатки та ізолювати їх від основної операційної системи. Система Docker бере на себе всю роботу по розгортанню цих контейнерних додатків, їх обслуговуванню, маршрутизації даних та розподілу навантажень і ресурсів.

Docker має просту архітектуру, про структуру і елементи якої користувачу програмного забезпечення не обов’язково знати для зручного користування основними його функціями. Архітектура системи складається з трьох основних

елементів: клієнт, образ Docker, контейнер Docker. Розглянемо ці елементи детальніше.

Клієнт – прошарок архітектури Docker, за допомогою якого виконується керування усіма процесами системи. Клієнт включає в себе налаштування системи, ручне виділення пам’яті для системи, налаштування кожного контейнера за допомогою командного рядка, інструменти для завантаження, розгортання, запуску, зупинки та моніторингу контейнерних додатків у системі Docker.

Образ Docker («docker image») – елемент системи, який містить у собі операційну систему, розгорнутий у ньому додаток та всі необхідні додаткові сервіси.

Контейнер Docker – запущений образ Docker. Контейнери можна запускати, зупиняти, переносити та видаляти.

В результаті проведеного дослідження було обрано програмне забезпечення Docker як середовище для розгортання системи керування базами даних Microsoft SQL Server. Перевагою даної системи є її простота та досить вузька спеціалізація саме на необхідній в даному випадку сфері. Недоліком Kubernetes є його складність та велика кількість функціоналу, який не буде використано при розробці та розгортанні даного додатку, що ускладнить вивчення роботи програми та безпосередньо розгортання і запуск серверу бази даних. Недоліком VirtualBox є додаткове навантаження та додаткова робота по запуску операційної системи на наявній операційній системі. На відміну від Docker, VirtualBox буде більш помітним при роботі та споживатиме більше ресурсів. Також, при розгортанні необхідних сервісів на VirtualBox це розгортання, запуск та всі налаштування необхідно буде проводити вручну, натомість при використанні Docker відповідальність за всю цю роботу бере на себе образ і контейнер.

Після вибору середовища розгортання та запуску системи керування базами даних, необхідно обрати спосіб, яким створюватиметься та заповнюватиметься база даних.

Для взаємодії серверної частини додатку з базою даних було обрано технологію Entity Framework Core. Вона визначає та описує різні способи підключення бази даних, опису її сутностей та подальшої взаємодії з цими сутностями.

Інженерами, які працювали над розробкою Entity Framework Core, було введено такий підхід до роботи з базами даних, як “code-first” [35]. Такий підхід до проектування баз даних передбачає, що розробник повинен описати усі сутності бази даних як класи у мові програмування C#, після чого за допомогою інструментів, наданих розробниками технології, необхідно провести налаштування зв’язків між майбутніми сутностями та реєстрацію усіх майбутніх сутностей.

Таким чином, було проведено аналіз спроектованої бази даних та описано усі необхідні класи мови програмування C# для забезпечення зберігання відповідних даних та зв’язків між ними. Розглянемо усі класи детальніше.

Клас User описує сутність користувача. В ньому зберігатиметься уся основна інформація про користувача, така як унікальний ідентифікатор користувача у таблиці бази даних, ім’я, під яким користувача ідентифікуватимуть інші користувачі платформи, електронна пошта користувача, яка у подальшому може бути використана для відновлення паролю, отримання інформації від адміністрації сервісу, або для відновлення акаунту, роль користувача, яка необхідна для визначення рівня доступу до даних, хеш паролю користувача для автентифікації при кожній спробі авторизації, рейтинг користувача, який базується на реакціях на його пости, коментарі та додавання його постів в список улюблених, посилання на фото профілю користувача та статус акаунта користувача, який визначає, чи акаунт активний, заблокований або видалений. Клас User виглядає наступним чином:

```
public class User
{
      public int Id { get; set; }
      public string Username { get; set; }
      public string Email { get; set; }
      public Role Role { get; set; }
```

```
        public string PasswordHash { get; set; }
        public double Rating { get; set; }
        public string PhotoUrl { get; set; }
        public ContentStatus Status { get; set; }
}
```

Роль та статус активності акаунту користувача описано за допомогою перелічення – структури мови програмування C#. Перелічення Role виглядає наступним чином:

```
public enum Role
{
        User = 1,
        Moderator = 2,
        Administrator = 3
}
```

Перелічення ContentStatus виглядає наступним чином:

```
public enum ContentStatus
{
        Ok = 0,
        Banned = 1,
        DeletedByOwner = 2
}
```

Клас Topic описує сутність теми. Даний клас містить у собі унікальний ідентифікатор теми в таблиці бази даних, заголовок теми, час створення теми та статус активності теми, який так само як і у випадку з користувачем, може визначати що тема є заблокованою адміністрацією або видаленою самим користувачем. Клас Topic виглядає наступним чином:

```
public class Topic
{
        public int Id { get; set; }
        public string Title { get; set; }
        public DateTime CreatedUtc { get; set; }
        public ContentStatus Status { get; set; }
}
```

Клас Commentary описує сутність коментаря. Клас містить унікальний ідентифікатор коментаря в таблиці бази даних, текст коментаря, час створення та статус активності. Клас Commentary виглядає наступним чином:

```
public class Commentary
{
	public int Id { get; set; }
	public string Text { get; set; }
	public DateTime CreatedUtc { get; set; }
	public ContentStatus Status { get; set; }
}
```

Клас Reaction описує реакцію користувача на коментар або тему та містить у собі унікальний ідентифікатор реакції в таблиці бази даних, значення реакції та час реагування. Клас Reaction виглядає наступним чином:

```
public class Reaction
{
	public int Id { get; set; }
	public ReactionValue ReactionValue { get; set; }
	public DateTime CreatedUtc { get; set; }
}
```

Значення реакції описано переліченням ReactionValue та визначає позитивною чи негативною була реакція. Перелічення ReactionValue виглядає наступним чином:

```
public enum ReactionValue
{
	Negative = -1,
	Positive = 1
}
```

Клас Report описує сутність скарги користувача на іншого користувача, тему чи коментар. Клас містить у собі унікальний ідентифікатор скарги в таблиці бази даних, тип скарги та її додатковий опис від користувача. Клас виглядає наступним чином:

```
public class Report
{
	public int Id { get; set; }
	public ReportType Type { get; set; }
	public string Description { get; set; }
}
```

Тип скарги, описаний переліченням ReportType, може визначати, що на думку користувача контент був проявом чи містив насильство, був спамом,

представляв контент для дорослих або є проявом порушення правил спільноти іншого характеру. Перелічення виглядає наступним чином:

```
public enum ReportType
{
	Violence = 1,
	Spam = 2,
	NSFW = 3,
	Other = 4
}
```

Клас Tag описує сутність тегу теми, містить унікальний ідентифікатор тегу в таблиці бази даних, його текст, кількість разів, коли цей тег використовували в темах та статус активності тега. Клас Tag виглядає наступним чином:

```
public class Tag
{
	public int Id { get; set; }
	public string Text { get; set; }
	public int UsedCount { get; set; }
	public ContentStatus Status { get; set; }
}
```

Окрім перелічених та описаних класів, також є такі класи, як FavoriteTopic, UserReport, TopicTag, TopicReport, UserTopicReaction, CommentaryReport, UserCommentaryReaction, UserTopic, InterestTag, UserCommentary, TopicCommentary, необхідні для забезпечення усіх зв’язків, визначених на етапі проектування бази даних. Дані класи містять у собі унікальні ідентифікатори записів відповідних сутностей, які можна визначити з назви цих класів.

Наступним етапом створення бази даних підходом «code-first» є налаштування зв’язків між її сутностями. Налаштування зв’язків даної бази даних виглядає наступним чином:

```
builder.Entity<UserTopic>()
	.HasOne<User>()
	.WithMany()
	.HasForeignKey(t => t.UserId);
builder.Entity<UserTopic>()
	.HasOne<Topic>()
	.WithOne()
	.HasForeignKey<UserTopic>(t => t.TopicId);
builder.Entity<TopicCommentary>()
```

```
        .HasOne<Topic>()
        .WithMany()
        .HasForeignKey(c => c.TopicId);
builder.Entity<TopicCommentary>()
        .HasOne<Commentary>()
        .WithOne()
        .HasForeignKey<TopicCommentary>(c => c.CommentaryId);
builder.Entity<UserCommentary>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(c => c.UserId);
builder.Entity<UserCommentary>()
        .HasOne<Commentary>()
        .WithOne()
        .HasForeignKey<UserCommentary>(c => c.CommentaryId);
builder.Entity<TopicTag>()
        .HasOne<Topic>()
        .WithMany()
        .HasForeignKey(t => t.TopicId);
builder.Entity<TopicTag>()
        .HasOne<Tag>()
        .WithMany()
        .HasForeignKey(t => t.TagId);
builder.Entity<UserTopicReaction>()
        .HasOne<Topic>()
        .WithMany()
        .HasForeignKey(t => t.TopicId);
builder.Entity<UserTopicReaction>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(_ => _.UserId);
builder.Entity<UserTopicReaction>()
        .HasOne<Reaction>()
        .WithOne()
        .HasForeignKey<UserTopicReaction>(_ => _.ReactionId);
builder.Entity<UserCommentaryReaction>()
        .HasOne<Commentary>()
        .WithMany()
        .HasForeignKey(c => c.CommentaryId);
builder.Entity<UserCommentaryReaction>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(_ => _.UserId);
builder.Entity<UserCommentaryReaction>()
        .HasOne<Reaction>()
        .WithOne()
        .HasForeignKey<UserCommentaryReaction>(_ => _.ReactionId);
```

Після налаштування зв’язків між сутностями необхідно налаштувати первинні ключі цих сутностей. Налаштування ключів проводиться наступним чином:

```
builder.Entity<UserTopic>()
        .HasKey(ut => new { ut.UserId, ut.TopicId });
builder.Entity<UserCommentary>()
        .HasKey(uc => new { uc.UserId, uc.CommentaryId });
builder.Entity<TopicCommentary>()
        .HasKey(tc => new { tc.TopicId, tc.CommentaryId });
```

```
builder.Entity<UserTopicReaction>()
      .HasKey(tr => new { tr.UserId, tr.TopicId, tr.ReactionId });
builder.Entity<TopicReport>()
      .HasKey(tr => new { tr.TopicId, tr.ReportId });
builder.Entity<TopicTag>()
      .HasKey(tt => new { tt.TopicId, tt.TagId });
builder.Entity<UserCommentaryReaction>()
      .HasKey(cr => new { cr.UserId, cr.CommentaryId, cr.ReactionId });
builder.Entity<CommentaryReport>()
      .HasKey(cr => new { cr.CommentaryId, cr.ReportId });
builder.Entity<FavoriteTopic>()
      .HasKey(ft => new { ft.UserId, ft.TopicId });
builder.Entity<UserReport>()
      .HasKey(ur => new { ur.UserId, ur.ReportId });
builder.Entity<InterestTag>()
      .HasKey(ut => new { ut.UserId, ut.TagId });
```

Останнім етапом створення бази даних методом «code-first» є оголошення, або реєстрація, усіх сутностей, які складатимуть базу даних. Оголошення виглядає наступним чином:

```
public DbSet<User> Users { get; set; }
public DbSet<Topic> Topics { get; set; }
public DbSet<Commentary> Commentaries { get; set; }
public DbSet<Reaction> Reactions { get; set; }
public DbSet<Tag> Tags { get; set; }
public DbSet<Report> Reports { get; set; }
public DbSet<InterestTag> InterestTags { get; set; }
public DbSet<FavoriteTopic> FavoriteTopics { get; set; }
public DbSet<UserReport> UserReports { get; set; }
public DbSet<TopicTag> TopicTags { get; set; }
public DbSet<TopicReport> TopicReports { get; set; }
public DbSet<UserTopicReaction> UserTopicReactions { get; set; }
public DbSet<CommentaryReport> CommentaryReports { get; set; }
public DbSet<UserCommentaryReaction> UserCommentaryReactions { get; set; }
public DbSet<UserTopic> UserTopics { get; set; }
public DbSet<UserCommentary> UserCommentaries { get; set; }
public DbSet<TopicCommentary> TopicCommentaries { get; set; }
```

Після створення класів і налаштування всіх необхідних зв’язків та ключів, для коректної роботи з базою даних та зручного подальшого її оновлення, зручно використовувати технологію міграцій [36]. Завдяки міграціям, після будь-яких змін внесених в базу даних, її не потрібно створювати заново, а всі необхідні сутності та поля створяться в базі даних автоматично, без втрати будь-яких даних.

Таким чином, для створення міграції необхідно відкрити консоль чи термінал у директорії проекту та ввести команду за наступним шаблоном:

```
dotnet ef migrations add <Назва міграції> --startup-project <Назва проекту, з яким розпочинається виконання додатку> --project <Назва проекту, в якому знаходиться або знаходитиметься папка з міграціями>
```

Після створення міграції, необхідно підтвердити зміни та оновити базу даних, ввівши команду за наступним шаблоном:

```
dotnet ef database update --startup-project <Назва проекту, з яким розпочинається виконання додатку> --project <Назва проекту, в якому знаходиться папка з міграціями>
```

3.2 Розробка програмних модулів

На етапі проектування даний дипломний проект було розділено на дві частини: серверна частина та клієнтська частина. Кожна з цих частин поділяється на окремі модулі. Розглянемо детальніше кожен з них.

Однією зі складових серверної частини додатку є домен сервера, який займається зв’язуванням бази даних з всією іншою логікою додатку і складається з міграцій бази даних, моделей, які є класами та описують сутності бази даних, класом-контекстом, в якому оголошуються, або реєструються, сутності бази даних, налаштовуються зв’язки між цими сутностями та визначаються їх первинні і зовнішні ключі.

Ще одним важливим модулем домену додатку є класи та інтерфейси, за допомогою яких реалізується шаблон проектування репозиторій [37]. Цей модуль необхідний для визначення в ньому усіх необхідних методів та команд для доступу в базу даних: отримання даних, запис даних, оновлення даних та видалення. Такий підхід до реалізації зв’язування бази даних з іншими модулями додатку дозволяє виконати легку підміну реалізації усіх методів та команд доступу до бази даних у випадку якщо, наприклад, необхідно буде змінити використовувану базу даних. Також цей підхід дозволяє легко розширювати реалізацію усіх модулів та полегшує подальшу підтримку як бази даних, так додатку в цілому.

Наступною складовою серверної частини дипломного проекту є проект з всіма сервісами та бізнес-логікою. Він повинен містити класи, які описують всі можливі помилки, що можуть бути утворені чи отримані в результаті роботи сервера, класи які описують всі можливі запити, які можуть надходити до сервера від клієнта чи будь-яких мікро-сервісів та класи, які описують усі можливі відповіді, які повертатимуться від сервера до клієнта та до будь-яких інших мікро-сервісів. У цій частині серверу також буде реалізовано шаблон проектування репозиторій, який буде надавати змогу легко розширяти функціонал платформи, підміняти будь-яку бізнес-логіку будь-якого модуля додатку.

Саме у цьому проекті додатку було реалізовано всю логіку модулів реєстрації нових користувачів, авторизації існуючих користувачів, зміни даних акаунту користувачами, створення, редагування та отримання тем, створення, отримання та редагування коментарів, створення тегів, їх додавання до тем та до списку тегів інтересів кожного користувача, створення реакцій, модерація користувачів, тем та коментарів. Розглянемо кожен з цих модулів детальніше.

При реєстрації користувача ще на етапі проектування та вирішення концепції платформи, було вирішено що кожен користувач повинен мати унікальний нікнейм, унікальну електронну пошту. Саме тому, в модулі реєстрації було реалізовано необхідні методи для пошуку наявності в системі користувачів з таким нікнеймом або електронною поштою, введеною користувачем у формі реєстрації. У разі наявності у системі користувача з таким нікнеймом чи електронною поштою, з модуля повертається помилка з повідомленням про те, ще користувач з такими даними вже існує. Таким чином повністю виключається можливість, що у системі буде кілька користувачів з однаковими даними, які повинні бути унікальними.

Важливим елементом модуля реєстрації є збереження пароля користувача, для отримання можливості звіряти та таким чином автентифікувати користувача при його наступній спробі авторизації.

Оскільки зберігати та транспортувати на стільки важливу інформацію в її першопочатковому вигляді небезпечно, пароль необхідно шифрувати або

хешувати. Таким чином, було реалізовано модуль хешування паролю за допомогою класу Rfc2898, описаного інженерами Microsoft при розробці бібліотеки System.Security.Cryptography. В модулі визначено два метода: GetHash та VerifyPassword. Метод GetHash отримує на вхід один параметр і займається вирахуванням хешу з стрічки, яку вводить користувач в полі паролю форми реєстрації. Реалізація методу виглядає наступним чином:

```
public string GetHash(string password)
{
	var salt = new byte[SALT_SIZE];
	new RNGCryptoServiceProvider().GetBytes(salt);
	var pbkdf2 = new Rfc2898DeriveBytes(password, salt, ITERATIONS);
	var hash = pbkdf2.GetBytes(HASH_SIZE);
	var hashBytes = new byte[HASH_SIZE + SALT_SIZE];
	Array.Copy(salt, 0, hashBytes, 0, SALT_SIZE);
	Array.Copy(hash, 0, hashBytes, SALT_SIZE, HASH_SIZE);
	var passwordHash = Convert.ToBase64String(hashBytes);
	return passwordHash;
}
```

Метод VerifyPassword отримує на вхід дві стрічки – пароль, введений користувачем в полі паролю форми авторизації, та хеш паролю, отриманий після пошуку в базі даних системи користувача з таким нікнеймом або поштою, введеною користувачем в полі логіну на формі авторизації. Таким чином, метод вираховує хеш із введеного користувачем з форми авторизації та порівнює його з наявним у базі даних. Якщо паролі співпадають, метод повертає значення «true», в іншому випадку – значення «false». Реалізація методу виглядає наступним чином:

```
public bool VerifyPassword(string password, string passwordHash)
{
	var hashBytes = Convert.FromBase64String(passwordHash);
	var salt = new byte[SALT_SIZE];
	Array.Copy(hashBytes, 0, salt, 0, SALT_SIZE);
	var pbkdf2 = new Rfc2898DeriveBytes(password, salt, ITERATIONS);
	var hash = pbkdf2.GetBytes(HASH_SIZE);
	for (int i = 0; i < HASH_SIZE; i++)
	{
		if (hashBytes[i + SALT_SIZE] != hash[i])
		{
			return false;
		}
	}
	return true;
```

Ще одним важливим модулем системи авторизації у даному дипломному проекті є використання технології JWT (Json Web Token). Для її реалізації необхідно провести певні налаштування, описати конфігурацію використання технології та самі методи для створення необхідних токенів.

Для зручної конфігурації технології JWT було створено відповідний клас JWTTokenConfiguration, який описує необхідні поля конфігурації, а саме: Secret, Issuer, Audience, AccessTokenExpiresInMinutes та RefreshTokenExpiresInMinutes. Клас виглядає наступним чином:

```
public class JwtTokenConfiguration
{
      public string Secret { get; set; }
      public string Issuer { get; set; }
      public string Audience { get; set; }
      public int AccessTokenExpiresInMinutes { get; set; }
      public int RefreshTokenExpiresInMinutes { get; set; }
}
```

Для автоматичної конфігурації технології, ініціалізація та присвоєння всіх необхідних даних проводитиметься при кожному запуску сервера, значення властивостей зберігатимуться в змінних середовища машини, на якій розгортатиметься та запускатиметься сервер. Процес конфігураціє виглядає наступним чином:

```
public static IServiceCollection AddJwtTokenConfiguration(this IserviceCollection
services, out JwtTokenConfiguration jwtConfiguration)
{
      jwtConfiguration = new JwtTokenConfiguration()
      {
            Secret = Environment.GetEnvironmentVariable(«TOD_JWT_SECRET»),
            Issuer = Environment.GetEnvironmentVariable(«TOD_JWT_ISSUER»),
            Audience = Environment.GetEnvironmentVariable(«TOD_JWT_AUDIECE»),
      AccessTokenExpiresInMinutes =
int.TryParse(Environment.GetEnvironmentVariable(«TOD_JWT_AT_EXPIRATION»),
      out int accExpiration) ? accExpiration : 20,
      RefreshTokenExpiresInMinutes =
int.TryParse(Environment.GetEnvironmentVariable(«TOD_JWT_RT_EXPIRATION»),
      out int refExpiration) ? refExpiration : 30,
      };
      services.AddSingleton(jwtConfiguration);
      return services;
}
```

Для збереження JWT-токенів було використано сервіс RedisCache [38]. Для його використання необхідно реалізувати його підключення, налаштування, та методи для отримання та записування даних на сервер сервісу.

Для зручного налаштування сервісу було реалізовано клас RedisConfiguration з необхідними для налаштування властивостями. Клас виглядає наступним чином:

```
public class RedisConfiguration
{
	public string InstanceName { get; set; }
	public string Address { get; set; }
	public int Port { get; set; }
	public string Password { get; set; }
}
```

Для реалізації налаштування сервісу було реалізовано відповідний метод. Як і у випадку з налаштуванням технології JWT, для збереження значень буде використано змінні середовища машини, на якій розгортатиметься та запускатиметься сервер. Метод конфігурації виглядає наступним чином:

```
public static IserviceCollection AddRedisConfiguration(this IserviceCollection
services, out RedisConfiguration redisConfiguration)
{
	redisConfiguration = new RedisConfiguration
	{
		InstanceName = Environment.GetEnvironmentVariable(
			«RedisCacheName»),
		Address = Environment.GetEnvironmentVariable(
			«RedisCacheAddress»),
		Port = int.Parse(Environment.GetEnvironmentVariable(
			«RedisCachePort»)),
		Password = Environment.GetEnvironmentVariable(
			«RedisCachePassword»),
	};
	services.AddSingleton(redisConfiguration);
	return services;
}
```

Для роботи з сервісом Redis було створено клас RedisService та реалізовано в ньому основні методи – метод SaveAsync для збереження даних до сервісу, метод DeleteAsync для видалення даних з сервісу та метод GetRefreshTokenAsync для отримання даних з сервісу. Метод SaveAsync виглядає наступним чином:

```
public async Task SaveAsync(string accessToken, int passwordHashCode, string
refreshToken, TimeSpan deleteAfter, CancellationToken cancellationToken =
default)
{
	try
	{
		var key = this.GetKey(accessToken, passwordHashCode);
		var options = new DistributedCacheEntryOptions
		{
			AbsoluteExpirationRelativeToNow = deleteAfter
		};
		await this.distributedCache.SetStringAsync(
			key, refreshToken, options, cancellationToken);
	}
	catch
	{
		throw new RedisException();
	}
}
```

Реалізація методу DeleteAsync для видалення даних з сервісу виглядає наступним чином:

```
public async Task DeleteAsync(string accessToken, int passwordHashCode,
CancellationToken cancellationToken = default)
{
	try
	{
		var key = this.GetKey(accessToken, passwordHashCode);
		await this.distributedCache.RemoveAsync(key, cancellationToken);
	}
	catch
	{
		throw new RedisException();
	}
}
```

Реалізація методу GetRefreshTokenAsync для отримання даних з сервісу виглядає наступним чином:

```
public async Task<string> GetRefreshTokenAsync(string accessToken, int
passwordHashCode, CancellationToken cancellationToken = default)
{
	var key = this.GetKey(accessToken, passwordHashCode);

	try
	{
		var refreshToken = await this.distributedCache.GetStringAsync(
			key, cancellationToken);
		return refreshToken;
	}
	catch
	{
		throw new RedisException();
```

Наступним було реалізовано модуль тем, який займається створенням тем, їх редагуванням, отриманням з бази даних то поміченням їх як видалених.

Для реалізації модуля тем було створено клас TopicService, в якому було реалізовано методи GetTopicByIdAsync, GetTopicsAsync, GetFavoritesAsync, GetMyTopicsAsync, GetDiscussedTopicsAsync, GetRecommendedTopicsAsync, CreateAsync, UpdateAsync, AddToFavoritesAsync, SearchTopicsAsync, MarkTopicDeletedAsync. Розглянемо деякі з цих методів детальніше.

Метод CreateAsync виконує створення теми та її додавання в базу даних. При створенні теми необхідно перевірити список текстів тегів, які сервер отримує разом із запитом з клієнта. При перевірці тегів, проводиться пошук наявності таких тегів у базі даних. Якщо тег з таким текстом уже є у системі, в проміжній сутності TopicTags зберігається поле з інформацією про належність даного тега до цієї теми та збільшується кількість разів, коли цей тег зазначався в темах. У разі, якщо такого тега немає, він створюється та також запис з цим тегом додається в проміжну сутність TopicTags. Також, після перевірки списку тегів, створюється та додається до бази даних запис з темою і запис в проміжну сутність UserTopics. Реалізація методу CreateAsync виглядає наступним чином:

```
public async Task<CreateTopicResponse> CreateAsync(CreateTopicRequest request,
int userId)
{
      var user = await this.contentValidator.GetAndValidateUserAsync(userId);
      var topicSearchResult = await this.topicRepository.GetByTitleAsync(
            request.Title);
      if (topicSearchResult != null)
      {
            throw new TopicAlreadyExistsException();
      }
      var topic = new Topic
      {
            Title = request.Title,
            CreatedUtc = DateTime.UtcNow,
            Status = ContentStatus.Ok
      };
      topic = await this.topicRepository.CreateAsync(topic);
      await this.CreateUserTopicAsync(userId, topic.Id);
      var topicTags = await this.GetTopicTagsAsync(
            userId, topic.Id, request.Tags);
      return new CreateTopicResponse
      {
            Id = topic.Id,
            Title = topic.Title,
            Author = new UserDto(user),
```

```
            CreatedUtc = topic.CreatedUtc,
            Tags = topicTags
      };
}
```

Наступним модулем було описано модуль коментарів. В даному модулі було реалізовано методи, необхідні для створення, зміни, видалення та отримання коментарів. Для реалізації даного модуля було створено клас CommentaryService та імплементовано методи GetTopicCommentariesAsync, CreateCommentaryAsync, UpdateCommentary та MarkCommentaryDeleted. Також для зручності вибору обговорених тем, саме в модулі коментарів було реалізовано метод GetLatestDiscussedTopicsIds. Реалізація даного методу виглядає наступним чином:

```
public async Task<List<int>> GetLatestDiscussedTopicsIds(int userId, int count =
20)
{
      var userCommentariesIds = this.userCommentaryRepository.
            GetCommentariesIdsByUserId(userId);
      var topicsIds = new List<int>();
      while (topicsIds.Count != count)
      {
            var commentaryId = userCommentariesIds.FirstOrDefault();
            if (commentaryId == 0)
            {
                  return topicsIds;
            }
            var topicId = await this.topicCommentaryRepository.
                  GetTopicIdByCommentaryId(commentaryId);
            var topic = await this.topicRepository.GetAsync(topicId);
            if (topic.Status != ContentStatus.Ok)
            {
                  userCommentariesIds.RemoveAt(0);
                  continue;
            }
            if (topicsIds.Contains(topicId))
            {
                  userCommentariesIds.RemoveAt(0);
                  continue;
            }
            topicsIds.Add(topicId);
      }
      return topicsIds;
}
```

Наступним було реалізовано модуль реакцій. Для реалізації модуля реакцій було створено клас ReactionService. В даному класі було описано методи для отримання за бази даних реакцій на коментарі та теми, методи для створення

реакцій користувача на теми та коментарі та методи для отримання реакцій певного користувача на певний коментар або на певну тему. Таким чином, у класі було реалізовано методи GetReactionsByCommentaryIdAsync, GetReactionsByTopicIdAsync, ReactOnTopicAsync, ReactOnCommentaryAsync, GetUserTopicReactionById, GetUserCommentaryReactionById.

Розглянемо детальніше метод ReactOnTopicAsync. У даному методі відбувається перевірка на статус активності акаунта користувача, який бажає відреагувати на тему та перевірка статусу активності теми, на яку хоче відреагувати користувач. У випадку, якщо акаунт користувача або тема є неактивними, тобто такими які були помічені видаленими або заблоковані адміністрацією платформи, з модуля повернеться помилка з відповідним повідомленням. Також проводиться перевірка на те, чи не є дана тема контентом користувача, який хоче відреагувати на тему. У випадку, якщо тема належить даному користувачу, з модуля повернеться помилка з відповідним повідомленням. Було реалізовано перевірку на те, чи користувач раніше реагував на цю тему та чи реакція не мала таке ж значення, як і зараз. Цю перевірку необхідно провести, щоб користувач не міг нескінченну кількість разів негативно чи позитивно реагувати на тему. У випадку, якщо користувач раніше не реагував на тему або якщо значення реакції відрізняється від попереднього, рейтинг теми оновлюється. Реалізація методу виглядає наступним чином:

```
public async Task<bool> ReactOnTopicAsync(
      int topicId, int userId, ReactionValue value)
{
      var topic = await this.contentValidator.
            GetAndValidateTopicAsync(topicId);
      var user = await this.contentValidator.GetAndValidateUserAsync(userId);
      var authorId = await this.userTopicRepository.
            GetUserIdByTopicIdAsync(topicId);
      if (authorId == userId)
      {
            throw new ContentBelongsToYouException();
      }
      var reactionId = await this.userTopicReactionRepository.
            GetByUserIdAndTopicId(userId, topicId);
      if (reactionId == 0)
      {
            var newReaction = new Reaction
            {
                  ReactionValue = value,
```

```
                CreatedUtc = DateTime.UtcNow
            };
            newReaction = await this.reactionRepository.
                CreateAsync(newReaction);
            await this.CreateUserTopicReactionAsync(
                userId, topicId, newReaction.Id);
            await this.UpdateAuthorRating(
                topicId, authorId, (int)value, ContentType.Topic);
            return true;
        }
        var reaction = await this.reactionRepository.GetAsync(reactionId);
        if (reaction.ReactionValue == value)
        {
            throw new AlreadyReactedException(ContentType.Topic, value);
        }
        reaction.ReactionValue = value;
        await this.reactionRepository.UpdateAsync(reaction);
        await this.UpdateAuthorRating(
            topicId, authorId, (int)value * 2, ContentType.Topic);
        return true;
}
```

Далі було реалізовано модуль тегів. Для реалізації було створено клас TagService та реалізовано методи для отримання тегу за його унікальним ідентифікатором у таблиці бази даних, отримання за текстом тегу, метод для отримання списку тегів конкретної теми, а також методи для створення тегу та збільшення числа згадувань тегу в темах. Таким чином у даному модулі було реалізовано методи GetByIdAsync, GetByTitleAsync, CreateAsync, IncreaseUsedCountAsync, GetByTopicIdAsync.

Наступним було реалізовано модуль тегів інтересів користувача. Модуль необхідний для налаштування роботи з рекомендованими темами, тобто з темами, які містять такі ж теги, які знаходяться в списку тегів інтересів конкретного користувача. Для реалізації модуля було створено клас InterestTagService та описано в ньому методи GetUserInterestTagsAsync (для отримання тегів інтересів користувача) та метод UpdateUserInterestsTagsAsync (для оновлення тегів інтересів користувача). Давайте розглянемо детальніше метод UpdateUserInterestsTagsAsync:

```
public async Task<InterestTagsResponse> UpdateUserInterestTagsAsync(int userId,
List<string> tagsText)
{
      var user = await this.contentValidator.GetAndValidateUserAsync(userId);
      await this.interestTagRepository.DeleteUserInterestTagsAsync(userId);
      var tags = new List<Tag>();
```

```
        foreach (var tagText in tagsText)
        {
                var tag = await this.tagService.GetByTitleAsync(tagText);
                if (tag == null)
                {
                        tag = await this.tagService.CreateAsync(tagText, userId);
                }
                if (tag.Status == ContentStatus.Banned)
                {
                        continue;
                }
                tags.Add(tag);
                var interestTag = new InterestTag
                {
                        UserId = userId,
                        TagId = tag.Id
                };
                await this.interestTagRepository.CreateAsync(interestTag);
        }
        return new InterestTagsResponse
        {
                Tags = tags
        };
}
```

Останнім модулем, який було описано в шарі бізнес-логіки додатку був модуль модерації платформи. Для реалізації модуля було створено клас ModerationService, в якому було реалізовано методи для отримання скарг на користувачів, теми та коментарі та методи для блокування користувачів, тем та коментарів та методи для створення скарг на користувачів, теми та коментарі. Таким чином в класі було реалізовано методи GetUsersReportsAsync, GetTopicsReportsAsync, GetCommentariesReportsAsync, ReportOnUserAsync, ReportOnTopicAsync, ReportOnCommentaryAsync, BanUserAsync, BanTopicAsync, BanCommentaryAsync. Розглянемо детальніше метод BanUserAsync. Необхідною мірою при блокуванні користувача є перевірка, чи має даний користувач необхідний дозвіл на виконання такої діяльності, тобто чи має користувач роль модератора або адміністратора. Реалізація метода виглядає наступним чином:

```
public async Task BanUserAsync(int adminId, int userId)
{
        var admin = await this.contentValidator.
                GetAndValidateUserAsync(adminId);
        if (admin.Role != Role.Administrator || admin.Role != Role.Moderator)
        {
                throw new PermissionDeniedException();
        }
        var user = await this.userService.GetByIdAsync(userId);
        if (user == null)
```

```
        {
            throw new NotFoundException(ContentType.User);
        }
        user.Status = ContentStatus.Banned;
        await this.userService.UpdateAsync(user);
}
```

Останнім шаром серверної частини додатку є проект Web, який призначений для налаштування зв'язку між сервером системи та її веб-клієнтом. У даному шарі знаходяться контролери, за допомогою яких визначаються API шляхи для запитів на сервер та файли конфігурації проекту для його коректного розгортання та запуску. Таким чином у проекті визначено класи контролерів AccountController, який призначений для обробки усіх запитів, пов'язаних з акаунтом користувача, клас TopicController, який призначений для обробки усіх запитів, пов'язаних з темами, клас CommentaryController для обробки запитів, пов'язаних з коментарями, клас ReactionController для обробки запитів з реагування на теми і коментарі та клас ModerationController для обробки усіх запитів з модерації платформи адміністрацією.

Серед файлів конфігурації важливо відмітити клас Program.cs – саме у цьому класі, починаючи з версії 6 платформи .NET [39], відбувається конфігурація та налаштування сервера. У цьому файлі відбувається конфігурація JWT та Redis сервісів а також підключення технології Swagger.

При проектуванні клієнтської частини додатку її було розділено на дві частини – модуль взаємодії з серверною частиною додатку та модуль інтерфейсу користувача.

Першим було реалізовано модуль взаємодії клієнтської частини додатку із серверною частиною додатку. В даному модулі було описано файли з методами для зручного формування запитів та самі методи запитів, файли з методами для налаштування та коректної роботи маршрутизації по сайту, файли з методами для виконання авторизації та автентифікації користувача на сайті та файли з методами для збереження важливої інформації в стані додатку і в сховищі браузера. Розглянемо детальніше файл зі збереженням інформації у стан і сховище та метод для відправки запиту та обробки відповіді з сервера.

У файлі store.js описано методи для ініціалізації та оновлення даних у стані та сховищі. При ініціалізації сторінки необхідно перевіряти, чи є користувач авторизованим, оскільки від цього залежить зовнішній вигляд та наповнення сайту. Саме тому при ініціалізації сторінки відбувається спроба зчитування об'єкту користувача зі сховища, оскільки у сховищі об'єкти знаходяться до моменту, коли браузер буде закрито. Згодом ці дані отримуються та оновлюються на сторінках за допомогою техніки "computed properties" [40], наявної у фреймворку Vue.js. Файл store.js виглядає наступним чином:

```
import { createStore } from "vuex";
export default createStore({
  state: {
    authObject: localStorage.getItem("authObject") ?
JSON.parse(localStorage.getItem("authObject")) : null,
  },
  mutations: {
    authObject(state, newAuthObject) {
      localStorage.setItem("authObject", JSON.stringify(newAuthObject));
      state.authObject = newAuthObject;
    }
  },
  actions: {
    authObject: async function (state, newAuthObject) {
      state.commit("authObject", newAuthObject);
    }
  },
  getters: {
    authObject(state) {
      return state.authObject;
    }
  },
});
```

У методі fetchApi описані та перевіряються усі можливі сценарії отримання відповідей на запити до сервера. Таким чином, використовуючи технологію JWT, термін придатності токена користувача може закінчуватися і його необхідно оновити. Саме тому відбувається перевірка на те, який є статус код отриманої відповіді. Якщо статус код відповіді становить «401» [41], це означає що користувач або не авторизований, або його токен більше не дійсний. Тоді система намагається виконати запит на оновлення токена користувача і якщо відповідь успішна, система ще раз повторює той запит, який користувач намагався виконати до оновлення токена. Код метода виглядає наступним чином:

```
async function fetchApi(path, options) {
  let response = await fetch(path, options);
  if (response.status == 401) {
    try {
      var refreshResponse = await refreshToken();
      if (refreshResponse.ok) {
        const authObject = await refreshResponse.json();
        await authenticationService.updateAuthObject(authObject);
        options.headers[“Authorization”] = “Bearer “ + authObject.accessToken;
        return await fetchApi(path, options);
      }
    } catch(error) {
      authenticationService.logout();
      router.push(“/login”);
    }
    authenticationService.logout();
    router.push(“/login”);
  }
  if (response.ok) {
    try {
      return await response.json();
    } catch (error) {
      console.log(error);
    }
    return “OK”;
  }
  throw { status: response.status, statusText: response.statusText, text:
JSON.parse(await response.text()) }
}
```

У наступному модулі клієнтського додатку, модулі інтерфейсу користувача, знаходяться файли з кодом сторінок та компонентів сайту, файли зі стилями сторінок, компонентів та всіх інших елементів сайту а також методи для обробки інформації, яка отримується з інтерфейсу користувача за допомогою введення даних в форму чи натискань на інтерактивні елементи сторінки та методи для обробки інформації, яка надходить на сторінки від модуля взаємодії із серверною частиною додатку. Розглянемо детальніше код сторінки Login.vue:

```
<template>
  <div style=»display: flex; margin: 5% 10%;»>
    <register-form @onRegisterSubmit=”register”
:apiException=”registerException”></register-form>
    <login-form @onLoginSubmit=”login” :apiException=”loginException”></login-
form>
  </div>
</template>

<script>
import LoginForm from “../components/forms/LoginForm.vue”;
import RegisterForm from “../components/forms/RegisterForm.vue”;
import authenticationService from “../services/authenticationService”;

export default {
```

```
  components: {
    LoginForm,
    RegisterForm,
  },
  data() {
    return {
      registerException: “”,
      loginException: “”,
    }
  },
  methods: {
    register: async function (registerObject) {
      try {
        await authenticationService.register(registerObject);
      } catch (error) {
        this.registerException = error.text;
        return;
      }

      this.$router.push(“/feed”);
    },
    login: async function (loginObject) {
      try {
        await authenticationService.login(loginObject.login,
loginObject.password);
      } catch (error) {
        this.loginException = JSON.parse(error.text);
        return;
      }

      this.$router.push(“/feed”);
    }
  }
}
</script>
```

При спробі увійти в систему, користувач переходить на дану сторінку і йому надається вибір – зареєструватися, тобто створити новий акаунт, або авторизуватися, тобто увійти в уже існуючий акаунт. Саме тому в блоці HTML-розмітки сторінки знаходиться лише один блочний елемент, який містить у собі два дочірні елементи – register-form та login-form – компоненти сторінки, які для зручності описані у інших файлах. Розглянемо детальніше код компонента login-form:

```
<template>
  <div class=”auth-container”>
      <div>
        <h3 class=”auth-header”>Login if you already have an account:</h3>
        <div class=”auth-field” :class=”{ error: v$.login.$errors.length }”>
          <input v-model=”login” class=”auth-input” placeholder=”Email or
Username” />
          <div style=”color: red;” v-for=”error of v$.login.$errors”
:key=”error.$uid”>
```

```
            <label>{{ error.$message }}</label>
          </div>
        </div>
        <div class="auth-field" :class="{ error: v$.lPassword.$errors }">
          <input v-model="lPassword" class="auth-input" type="password"
placeholder="Password"/>
          <div style="color: red;" v-for="error of v$.lPassword.$errors"
:key="error.$uid">
            <label>{{ error.$message }}</label>
          </div>
        </div>
        <div class="auth-field">
          <button class="auth-button" @click="this.submitForm()">Login</button>
        </div>
        <div v-if="apiException" class="auth-field" style="color: red; opacity:
0.8;">
          {{apiException}}
        </div>
      </div>
    </div>
</template>
<script>
import useVuelidate from "@vuelidate/core";
import { required, helpers } from "@vuelidate/validators";
export default {
  name: "login-form",
  setup() { return { v$: useVuelidate() } },
  data() {
    return {
      login: "",
      lPassword: "",
    }
  },
  validations() {
    return {
      login: {
        required: helpers.withMessage("Login is required", required),
      },
      lPassword: {
        required: helpers.withMessage("Password is required", required),
      },
    }
  },
  methods: {
    validate: function () {
      this.v$.$touch();
      if (this.v$.$invalid) return false;
      return true;
    },
    submitForm: function () {
      if (!this.validate()) return;
      let loginObject = {
        login: this.login,
        password: this.lPassword,
      };
      this.$emit("onLoginSubmit", loginObject);
    }
  },
  props: {
    apiException: { Type: String, },
  },
}
</script>
```

У даному компоненті відбувається декілька основних процесів: відображення форми для введення даних авторизації, валідація цих даних, тобто перевірка введених даних на коректність та відправка цих даних на сторінку входу в систему, тобто на батьківський елемент. Також даний компонент виконує функцію відображення помилки, яка було отримана у відповідь від сервера на запит про авторизацію користувача за такими даними.

3.3 Керівництво користувача

Даний додаток передбачає чотири типи користувачів:

- неавторизований користувач – користувач, який користується деяким функціоналом системи, але не виконує вхід в систему;
- авторизований користувач – користувач, який виконує вхід в систему та має змогу користуватися усіма функціями платформи;
- модератор – авторизований користувач, який має доступ до усіх функцій звичайного авторизованого користувача, проте окрім цих функцій має можливість модерувати контент та переглядати скарги користувачів на інших користувачів, теми та коментарі;
- адміністратор – авторизований користувач, який також має доступ до усіх наявних функцій платформи, має доступ до перегляду скарг на користувачів, теми, коментарі, має можливість модерації контенту, а також доступ до надання авторизованим користувачам ролі модератора.

Розглянемо покроково користування платформою в кожному з випадків.

При вході на сайт неавторизований користувач бачить сторінку з останніми опублікованими темами, може переглянути рейтинг цих тем, авторів тем, час їх створення та безпосередньо заголовки тем. Також неавторизований користувач має доступ до сторінки пошуку тем за тегами, автором та заголовком. Натиснувши на заголовок теми користувача буде переадресовано на сторінку

теми, де неавторизований користувач зможе бачити користувачі інших авторизованих користувачів та рейтинги цих коментарів.

Для розширення кількості доступного функціоналу неавторизований користувач може провести вхід у систему за допомогою процедури реєстрації або авторизації. Для цього користувачу необхідно перейти на сторінку входу в систему та заповнити відповідну форму коректними даними: при реєстрації необхідно вказати унікальний нікнейм та електронну пошту, ввести бажаний пароль та повторити його для підтвердження його коректності. У випадку, якщо користувач з таким нікнеймом або електронною поштою вже існує, користувачу буде виведено відповідне повідомлення.

При проведенні процедури авторизації користувач необхідний мати раніше зареєстрований акаунт та пам’ятати мінімальні реєстраційні дані – пароль до акаунту та нікнейм або адресу електронної пошти. Ввівши коректні дані, користувача буде авторизовано та надано доступ до усього функціоналу відповідно до його ролі.

Після авторизації користувачу стануть доступними функції коментування тем, можливість створювати власні теми, реагувати на теми інших користувачів, коментарі інших користувачів та функція додавання тем до списку улюблених. Після додавання теми до списку улюблених, користувач може переглянути список улюблених тем на окремій вкладці сторінки переліку тем.

Після створення власної теми або власного коментаря користувач матиме можливість редагування створеного контенту, наприклад додати або вилучити певні теги з теми, змінити заголовок теми або змінити текст коментаря. У випадку, якщо користувачу буде необхідно видалити тему чи коментар, він зможе скористатися кнопкою видалення на формі з відповідним контентом. Переглянути власні теми можна у окремій вкладці на сторінці переліку тем.

Також у кожного авторизованого користувача є можливість переглядати та змінювати дані акаунту, в тому числі теги інтересів поточного користувача. Після додавання або зміни тегів інтересів користувач може переглядати теми,

рекомендовані йому системою на основі його інтересів у окремій вкладці на сторінці переліку тем.

Після коментування певної теми вона з’явиться у списку останніх обговорених тем. Останні 20 обговорених тем можна переглянути у окремій вкладці на сторінці переліку тем.

Кожен авторизований користувач має можливість відправити скаргу на певний контент платформи – на іншого користувача, на тему створену іншим користувачем або на коментар створений іншим користувачем. Для цього необхідно натиснути на кнопку скарги на формі відповідного контенту. Після натискання даної кнопки користувач побачить модальне вікно з формою, у якій необхідно обрати тип скарги з представлених та, за необхідності, додати якийсь додатковий опис до цієї скарги.

Якщо роль авторизованого користувача визначається як «Модератор», користувачу стануть доступними функції перегляду усіх скарг на користувачів, теми та коментарі та функція блокування користувача, теми чи коментаря.

Роль користувача також може визначатися як «Адміністратор». Зазвичай користувач отримує цю роль через ручну зміну значення поля «Роль» запису цього користувача в базі даних. Окрім усіх інших функцій, які включаються до функціоналу неавторизованих користувачів, авторизованих користувачів та модераторів, користувач з роллю «Адміністратор» має доступ до функції надання іншим авторизованим користувачам ролі «Модератор» та забирати такий привілей у користувачів з роллю «Модератор» по тим чи іншим причинам.

3.4 Технічні характеристики веб-додатка

Оскільки взаємодія між двома основними частинами додатку, серверною та клієнтською, відбувається за принципом взаємодії «клієнт-сервер», ці частини додатку є абсолютно незалежними одна від одної і можуть бути розміщені на різних машинах. Особливістю в технічних вимогах додатку є те, що сервер та

клієнт хоч і є централізованими системами, проте код клієнта виконуватиметься на пристроях користувачів.

Саме тому вимоги для коректного функціонування необхідно розділити на дві частини – вимоги до централізованої системи, на якій розгортатиметься додаток та вимоги до пристроїв, на яких додатково виконуватиметься клієнтська частина додатку. Розглянемо детальніше вимоги до централізованої системи.

В першу чергу хостинг-сервіс повинен мати швидкі та місткі накопичувачі даних для швидкої обробки даних з бази даних та у разі потреби машина повинна мати можливість розширюватися. Оскільки коректна робота сервера безпосередньо залежить від обміну даними з зовнішніми клієнтами, машина повинна бути забезпечена високошвидкісним інтернет-з’єднанням.

Таким чином, узагальнені мінімальні вимоги для коректного функціонування серверної частини додатку виглядають наступним чином:

- ОС Ubuntu, Debian, Fedora, Microsoft Windows Server;
- процесор Intel Xeon E5;
- 4 ГБ DDR4 ОЗП;
- 10 ГБ постійної пам’яті твердотільного накопичувача.

Для готовності до збільшення кількості користувачів, кількості запитів та кількості даних, які необхідно обробляти рекомендується використання машини з наступними вимогами:

- ОС Ubuntu, Debian, Fedora, Microsoft Windows Server;
- три процесора Intel Xeon E5;
- 16 ГБ DDR5 ОЗП;
- 50 ГБ постійної пам’яті твердотільного накопичувача.

Для коректного функціонування клієнтської частини додатку на пристроях користувачів необхідно щоб пристрій користувачів забезпечував можливість виходу користувача в мережу Інтернет. Також пристрій повинен мати встановлений будь-який браузер, який підтримує контент, написаний на мові програмування JavaScript [42], а також мати невелику кількість оперативної

пам’яті для швидкого проведення обрахунків клієнтської частини додатку, які відбуваються на пристрої користувача.

Таким чином, узагальнені мінімальні вимоги для коректного функціонування клієнтської частини додатку виглядають наступним чином:

- ОС Android 4.0, iOS, Windows Phone, Windows XP/7/8/10, MacOS, Linux, Ubuntu;
- будь-який двоядерний процесор
- 500 МБ ОЗП;
- 500 МБ постійної пам’яті.

Рекомендовані вимоги виглядають наступним чином:

- ОС Android 4.0, iOS, Windows Phone, Windows XP/7/8/10, MacOS, Linux, Ubuntu;
- будь-який чотириядерний процесор;
- 2 ГБ ОЗП;
- 1 ГБ постійної пам’яті.

3.5 Завантаження веб-додатка на хостинг

Завантаження, розгортання та запуск додатка буде відбуватися на платформі Heroku [43].

Heroku – це безкоштовний сервіс, на якому можна розгортати додатки, написані на таких мовах програмування, як JavaScript, Ruby, Java, PHP, Python, Go, Scala та Clojure. В списку мов програмування, додатки яких доступні для розгортання, немає мови програмування C#, проте сервіс надає можливість публікації на своїх серверах Docker-контейнерів. На етапі проектування додатку ця інформація мала певний вплив на вибір технологій. Таким чином, використання при розробці проекту програмного забезпечення Docker дозволить легше та швидше виконати завантаження, розгортання та запуск додатка на сервісі Heroku.

В першу чергу, для публікації свого додатку на сервісі Heroku необхідно зареєструвати акаунт користувача. Після реєстрації на сервісі, користувачу доступна функція створення Heroku-додатка. При створенні Heroku-додатка необхідно вказати його тип та вид програмного забезпечення, яке згодом розгортатиметься та запускатиметься у межах даного Heroku-додатка. Оскільки ми розгортатимемо саме Docker-контейнер, обираємо відповідний тип. Після створення Heroku-додатка також необхідно обрати метод публікації, серед яких сервіс надає такі варіанти, як публікація з використанням Heroku Git, публікація з використанням сервісу GitHub або з використанням Container Registry. Для завантаження, розгортання та запуску даного додатка на сервісі Heroku було обрано метод публікації Container Registry, оскільки цей метод публікації має додаткову оптимізацію та усі необхідні команди для публікації Docker-контейнерів.

Для роботи з методом публікації Container Registry необхідно завантажити Heroku CLI. Після завантаження програмного забезпечення Heroku CLI необхідно скомпілювати проект сервера у публікаційному варіанті. Для виконання компіляції у такому виді, необхідно відкрити термінал у кореневій папці проекту та ввести наступну команду:

```
dotnet publish -c Release
```

Також необхідно провести ідентичну процедуру для проекту клієнтського додатку. Для проведення збірки клієнтського додатку, написаного на мові програмування JavaScript при використанні фреймворку Vue.js, у варіанті публікації, необхідно відкрити термінал у кореневій папці проекту та ввести наступну команду:

```
npm run build
```

Після отримання необхідних файлів їх потрібно запакувати в архів з розширенням «tar.gz». Для цього необхідно відкрити термінал у кореневій папці проекту та ввести наступну команду:

```
tar -czvf tod.tar.gz -C dist .
```

Після того як ми отримаємо усі необхідні файли проекту, скомпільовані для публікації, нам необхідно створити Docker-файл та помістити усі ці файли в одну папку. Для публікації на сервіс Heroku необхідно провести авторизацію з терміналу на пристрої, з якого відбуватиметься публікація. Для авторизації необхідно відкрити термінал та ввести наступну команду:

```
heroku login
```

Після введення даної команди, буде автоматично відкрито браузер та сторінку для авторизації на сайті сервісу Heroku. Після проходження авторизації сторінку буде закрито та перенаправлено назад до вікна терміналу.

У разі успішної авторизації користувач має можливість перейти в папку зі всіма файлами, які необхідно розмістити на сервісі Heroku та ввести наступні команди:

```
heroku container:push web
heroku container:release web
```

Отже, в результаті виконання третього розділу було розроблено базу даних, також було реалізовано усі раніше спроектовані модулі та частини платформи як на серверній стороні, так і на клієнтській стороні платформи. Також було описано керівництво користувача з поясненням роботи усього функціоналу платформи через призму їх використання користувачем. В даному розділі було визначено усі необхідні технічні характеристики платформи та завантажено додаток на безкоштовній сервіс з веб-хостингу Heroku.

4 ТЕСТУВАННЯ ВЕБ-ДОДАТКА

4.1 Вибір та обґрунтування методів тестування веб-додатка

При розробці веб-додатків зазвичай використовується два види тестування: unit-тестування [44] та інтеграційні тести [45]. Розглянемо кожен з видів тестування детальніше.

Unit-тестування – вид тестування, який передбачає розділення коду та логіки додатка на менші частини, модулі та окреме тестування кожного з таких модулів для перевірки коректності обробки даних на кожному окремому етапі роботи системи.

Інтеграційне тестування – вид тестування, який передбачає об’єднання раніше протестованих модулів для перевірки коректності їх взаємодії.

Для проведення тестування даного дипломного проекту було обрано unit-тестування, оскільки воно дасть можливість максимально детально перевірити працездатність кожного елемента додатку та кожного його модуля.

4.2 Перевірка на помилки за допомогою unit-тестів

При unit-тестуванні серверної частини додатку було проведено детальне тестування кожного класу-сервісу, які відповідають за усю бізнес-логіку додатка та знаходяться у шарі бізнес-логіки серверної частини додатка.

Також було проведено тестування усіх наявних класів-контролерів, які відповідають за взаємодію серверної чистини додатку з клієнтським додатком та знаходиться у верхньому шарі серверної частини додатка.

Отже, розглянемо детальніше тестування кожного з представлених модулів.

Проведемо тестування класу UserService. Усі описані сценарії зображено у таблиці 4.1.

Таблиця 4.1 – Тестові сценарії класу UserService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання користувача по його унікальному ідентифікатору	Backend.Services	GetUserByValidId_ReturnsUser	Коректний унікальний ідентифікатора користувача	Екземпляр сутності користувача
Отримання null в результаті пошуку користувача по не коректному унікальному ідентифікатору користувача	Backend.Services	GetUserByInvalidId_ReturnsNull	Некоректний унікальний ідентифікатор користувача	Null-значення
Отримання користувача по його унікальному нікнейму	Backend.Services	GetUserByValidUsername_ReturnsUser	Коректний унікальний нікнейм користувача	Екземпляр сутності користувача
Отримання null в результаті пошуку користувача по некоректному унікальному нікнейму	Backend.Services	GetUserByInvalidUsername_ReturnsNull	Некоректний унікальний нікнейм користувача	Null-значення
Оновлення даних про користувача в базі даних	Backend.Services	UpdateUser_UpdatesUser	Екземпляр сутності користувача	Оновлені дані про користувача

Проведемо тестування класу JwtAccountService. Усі описані сценарії зображено у таблиці 4.2.

Таблиця 4.2 – Тестові сценарії класу JwtAccountService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання урізаного екземпляра користувача в результаті пошуку по коректній адресі електронної пошти	Backend.Services	GetUserDtoByValidEmail_ReturnsUserDto	Коректна електронна адреса користувача	Екземпляр сутності користувача
Отримання null-значення в результаті пошуку по некоректній адресі електронної пошти	Backend.Services	GetUserDtoByInvalidEmail_ReturnsNull	Некоректна електронна адреса користувача	Null-значення
Отримання урізаного екземпляра користувача в результаті пошуку за коректним нікнеймом	Backend.Services	GetUserDtoByValidUsername_ReturnsUserDto	Коректний нікнейм користувача	Екземпляр сутності користувача

Кінець таблиці 4.2

1	2	3	4	5
Отримання урізаного екземпляра користувача в результаті пошуку за коректним унікальним ідентифікатором користувача	Backend .Services	GetUserDtoBy ValidId _ReturnsUserDto	Коректний унікальний ідентифікатор користувача	Екземпляр сутності користувача
Отримання null-значення в результаті пошуку за некоректним унікальним ідентифікатором користувача	Backend .Services	GetUserDtoBy InvalidId _ReturnsNull	Некоректний унікальний ідентифікатор користувача	Null-значення
Успішна операція авторизації користувача при введенні коректних даних	Backend .Services	GetLoginResponse ByValidAuthData _ReturnsLogin Response	Коректні авторизаційні дані	Дані результату авторизації
Помилка операції авторизації при введенні некоректних даних	Backend .Services	GetLoginResponse ByInvalidAuthData _ReturnsNull	Некоректні авторизаційні дані	Null-значення
Успішна операція реєстрації користувача при введенні коректних даних	Backend .Services	GetRegisterResponse ByValidRegData _ReturnsRegister Response	Коректні реєстраційні дані	Дані результату реєстрації
Помилка операції реєстрації при введенні некоректних даних	Backend .Services	GetRegisterResponse ByInvalidRegData _ReturnsNull	Некоректні реєстраційні дані	Null-значення

Проведемо тестування класу JwtTokenService. Усі описані сценарії зображено у таблиці 4.3.

Таблиця 4.3 – Тестові сценарії класу JwtTokenService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання токена доступу	Backend .Services	GetAccessToken _ReturnsAccessToken	Екземпляр сутності користувача	Токен доступу
Отримання токена відновлення	Backend .Services	GetRefreshToken _ReturnsRefreshToken		Токен відновлення
Отримання значення true при валідації коректного токена доступу	Backend .Services	ValidTokenValidation _ReturnsTrue	Коректний токен доступу	Значення true
Отримання значення false при валідації некоректного токена	Backend .Services	InvalidTokenValidation _ReturnsTrue	Некоректний токен доступу	Значення false

Проведемо тестування класу Rfc2898PasswordHasher. Усі описані сценарії зображено у таблиці 4.4.

Таблиця 4.4 – Тестові сценарії класу Rfc2898PasswordHasher

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання хешу пароля	Backend .Services	GetPasswordHash _ReturnsPassword Hash	Пароль користувача	Хеш пароля користувача
Отримання значення true при валідації коректного пароля	Backend .Services	ValidateValidPassword _ReturnsTrue	Коректний пароль користувача	Значення true
Отримання значення false при валідації некоректного пароля	Backend .Services	ValidateInvalidPassword _ReturnsTrue	Некоректний пароль користувача	Значення false

Проведемо тестування класу RedisService. Усі описані сценарії зображено у таблиці 4.5.

Таблиця 4.5 – Тестові сценарії класу RedisService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання токена доступу в результаті пошуку за коректними даними	Backend .Services	GetValidAccessToken _ReturnsAccessToken	Коректний токен доступу	Токен доступу
Отримання помилки в результаті пошуку за некоректними даними	Backend .Services	GetInvalidAccessToken _ReturnsError	Некоректний токен доступу	Помилка
Збереження до бази сервісу токена доступу	Backend .Services	SaveAccessToken _SavesAccessToken	Токен доступу	Збережений токен доступу
Видалення токена доступу з бази сервісу при коректному токені доступу	Backend .Services	ValidAsseccToken Deleting_DeletesToken	Коректний токен доступу	Видалений токен доступу
Помилка при спробі видалення некоректного токена доступу з бази сервісу	Backend .Services	InvalidAccessToken Deleting_ReturnsError	Некоректний токен доступу	Помилка

Проведемо тестування класу ModerationService. Усі описані сценарії зображено у таблиці 4.6.

Таблиця 4.6 – Тестові сценарії класу ModerationService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання скарг на користувачів	Backend .Services	GetUsersReports _ReturnsUsersReports	Токен доступу	Список скарг на користувачів
Отримання скарг на теми	Backend .Services	GetTopicReports _ReturnsTopicsReports	Токен доступу	Список скарг на теми
Отримання скарг на коментарі	Backend .Services	GetCommentaries Reports_Returns CommentariesReports	Токен доступу	Список скарг на коментарі
Блокування користувача	Backend .Services	UserBanning _BansUser	Токен доступу та ідентифікатор користувача	Статус користувача змінився на «Заблокований»
Блокування теми	Backend .Services	TopicBanning _BansTopic	Токен доступу та ідентифікатор користувача	Статус теми змінився на «Заблокований»
Блокування коментаря	Backend .Services	CommentaryBanning _BansCommentary	Токен доступу та ідентифікатор користувача	Статус коментаря змінився на «Заблокований»
Створення скарги на користувача	Backend .Services	UserReportCreating _CreatesUserReport	Дані форми скарги	Створена скарга на користувача
Створення скарги на тему	Backend .Services	TopicReportCreating _CreatesTopicReport	Дані форми скарги	Створена скарга на тему
Створення скарги на коментар	Backend .Services	CommentaryReport Creating_Creates UserReport	Дані форми скарги	Створена скарга на коментар

Проведемо тестування класу TopicService. Усі описані сценарії зображено у таблиці 4.7.

Таблиця 4.7 – Тестові сценарії класу TopicService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання останніх тем	Backend .Services	GetLatestTopics _ReturnsLatestTopics		Список останніх тем
Отримання улюблених тем	Backend .Services	GetFavoriteTopics _ReturnsFavoriteTopics	Токен доступу	Список улюблених тем

Кінець таблиці 4.7

1	2	3	4	5
Отримання останніх обговорених тем	Backend .Services	GetDiscussedTopics _Returns DiscussedTopics	Токен доступу	Список останніх обговорених тем
Отримання власних тем	Backend .Services	GetMyTopics _ReturnsMyTopics	Токен доступу	Список власних тем
Створення теми	Backend .Services	TopicCreation _CreatesTopic	Токен доступу, заповнена форма створення теми	Створена тема
Пошук по темах	Backend .Services	TopicSearch _ReturnsTopics	Заповнена форма пошуку тем	Список тем, які відповідають критеріям пошукового запиту
Оновлення теми	Backend .Services	TopicUpdating _UpdatesTopic	Заповнена форма оновлення теми	Оновлені дані про тему
Видалення теми	Backend .Services	TopicMarkingDeleted _MarksTopicDeleted	Токен доступу	Статус теми змінено на «Видалена користувачем»

Проведемо тестування класу CommentaryService. Усі описані сценарії зображено у таблиці 4.8.

Таблиця 4.8 – Тестові сценарії класу CommentaryService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання коментарів теми	Backend .Services	GetTopicCommentaries _ReturnsTopic Commentaries		Список коментарів теми
Створення коментаря	Backend .Services	CommentaryCreation _CreatesCommentary	Токен доступу, заповнена форма створення коментаря	Створений коментар
Оновлення коментаря	Backend .Services	CommentaryUpdating _UpdatesCommentary	Токен доступу, заповнена форма оновлення коментаря	Оновлені дані коментаря

Кінець таблиці 4.8

Видалення коментаря	Backend .Services	CommentaryMarking Deleted_Marks CommentaryDeleted	Токен доступу	Статус коментаря змінено на «Видалений користувачем»
Отримання списку унікальних ідентифікаторів обговорених тем	Backend .Services	GetLatestDiscussed TopicsIds_Returns LatestDiscussed TopicsIds	Токен доступу	Список ідентифікаторів обговорених тем

Проведемо тестування класу ReactionService. Усі описані сценарії зображено у таблиці 4.9.

Таблиця 4.9 – Тестові сценарії класу ReactionService

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання реакцій на тему	Backend .Services	GetTopicReactions _ReturnsTopicReactions	Ідентифікатор теми	Список реакцій
Отримання реакцій на коментар	Backend .Services	GetCommentary Reaction_Returns CommentaryReactions	Ідентифікатор коментаря	Список реакцій
Отримання реакції користувача на коментар	Backend .Services	GetUserCommentary Reaction_ReturnsUser CommentaryReaction	Токен доступу, ідентифікатор теми	Реакція користувача на тему
Отримання реакції користувача на коментар	Backend .Services	GetUserTopic Reaction_ReturnsUser TopicReaction	Токен доступу, ідентифікатор коментаря	Реакція користувача на коментар
Реагування користувача на тему	Backend .Services	ReactingOnTopic _CreatesUser TopicReaction	Токен доступу, ідентифікатор теми	Створена реакція користувача на тему
Реагування користувача на коментар	Backend .Services	ReactingOnCommentary _CreatesUser CommentaryReaction	Токен доступу, ідентифікатор коментаря	Створена реакція користувача на коментар

Проведемо тестування класу AccountController. Усі описані сценарії зображено у таблиці 4.10.

Таблиця 4.10 – Тестові сценарії класу AccountController

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Запит на реєстрацію користувача	Backend.Web	RegisterRequest_RegistersUser	Заповнена форма реєстрації	Зареєстрований новий користувач
Запит на авторизацію користувача	Backend.Web	AuthorizationRequest_AuthorizesUser	Заповнена форма авторизації	Отримано дані авторизації
Запит на відновлення токену	Backend.Web	RefreshTokenRequest_RefreshesToken	Токен відновлення	Отримано новий токен
Запит на отримання інформації про даного користувача	Backend.Web	GetMe_Returns UserInformation	Токен доступу	Отримано дані про користувача
Запит на отримання список тегів інтересів	Backend.Web	GetUserInterests ReturnsUserInterests	Токен доступу	Список інтересів
Запит на оновлення списку тегів інтересів	Backend.Web	UpdateUserInterests_UpdatesUserInterests	Токен доступу, список тегів	Оновлений список інтересів

Проведемо тестування класу TopicController. Усі описані сценарії зображено у таблиці 4.11.

Таблиця 4.11 – Тестові сценарії класу TopicController

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Запит на отримання теми	Backend.Web			
Запит на отримання останніх тем	Backend.Web	GetLatestTopics_ReturnsLatestTopics		Список останніх тем
Запит на отримання улюблених тем	Backend.Web	GetFavoriteTopics_ReturnsFavoriteTopics	Токен доступу	Список улюблених тем
Запит на отримання рекомендованих тем	Backend.Web	GetRecommended Topics_Returns RecommendedTopics	Токен доступу	Список рекомендованих тем
Запит на отримання обговорених тем	Backend.Web	GetDiscussedTopics_Returns DiscussedTopics	Токен доступу	Список останніх обговорених тем
Запит на отримання власних тем	Backend.Web	GetMyTopics_ReturnsMyTopics	Токен доступу	Список власних тем

Кінець таблиці 4.11

1	2	3	4	5
Запит на пошук теми	Backend .Web	TopicSearch _ReturnsTopics	Заповнена форма пошуку тем	Список тем, які відповідають критеріям пошукового запиту
Запит на оновлення теми	Backend .Web	TopicUpdating _UpdatesTopic	Заповнена форма оновлення теми	Оновлені дані про тему
Запит на видалення теми	Backend .Web	TopicMarkingDeleted _MarksTopicDeleted	Токен доступу	Статус теми змінено на «Видалена користувачем»

Проведемо тестування класу CommentaryController. Усі описані сценарії зображено у таблиці 4.12.

Таблиця 4.12 – Тестові сценарії класу CommentaryController

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Запит на отримання коментарі теми	Backend .Web	GetTopicCommentaries _ReturnsTopic Commentaries		Список коментарів теми
Запит на створення коментаря	Backend .Web	CommentaryCreation _CreatesCommentary	Токен доступу, заповнена форма створення коментаря	Створений коментар
Запит на оновлення коментаря	Backend .Web	CommentaryUpdating _UpdatesCommentary	Токен доступу, заповнена форма оновлення коментаря	Оновлені дані коментаря
Запит на видалення коментаря	Backend .Web	CommentaryMarking Deleted_Marks CommentaryDeleted	Токен доступу	Статус коментаря змінено на «Видалений користувачем»

Проведемо тестування класу ReactionController. Усі описані сценарії зображено у таблиці 4.13.

Таблиця 4.13 – Тестові сценарії класу ReactionController

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Запит на отримання реакцій на тему	Backend.Web	GetTopicReactions_ReturnsTopicReactions	Ідентифікатор теми	Список реакцій
Запит на отримання реакцій на коментар	Backend.Web	GetCommentaryReaction_ReturnsCommentaryReactions	Ідентифікатор коментаря	Список реакцій
Запит на отримання реакції користувача на коментар	Backend.Web	GetUserCommentaryReaction_ReturnsUserCommentaryReaction	Токен доступу, ідентифікатор теми	Реакція користувача на тему
Запит на отримання реакції користувача на коментар	Backend.Web	GetUserTopicReaction_ReturnsUserTopicReaction	Токен доступу, ідентифікатор коментаря	Реакція користувача на коментар
Запит на реагування користувача на тему	Backend.Web	ReactingOnTopic_CreatesUserTopicReaction	Токен доступу, ідентифікатор теми	Створена реакція користувача на тему
Запит на реагування користувача на коментар	Backend.Web	ReactingOnCommentary_CreatesUserCommentaryReaction	Токен доступу, ідентифікатор коментаря	Створена реакція користувача на коментар

Проведемо тестування класу ModerationController. Усі описані сценарії зображено у таблиці 4.14.

Таблиця 4.14 – Тестові сценарії класу ModerationController

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Запит на отримання скарг на користувачів	Backend.Web	GetUsersReports_ReturnsUsersReports	Токен доступу	Список скарг на користувачів
Запит на отримання скарг на теми	Backend.Web	GetTopicReports_ReturnsTopicsReports	Токен доступу	Список скарг на теми
Запит на отримання скарг на коментарі	Backend.Web	GetCommentariesReports_ReturnsCommentariesReports	Токен доступу	Список скарг на коментарі
Запит на блокування користувача	Backend.Web	UserBanning_BansUser	Токен доступу та ідентифікатор користувача	Статус користувача змінився на «Заблокований»

Кінець таблиці 4.14

1	2	3	4	5
Запит на блокування коментаря	Backend.Web	CommentaryBanning_BansCommentary	Токен доступу та ідентифікатор користувача	Статус коментаря змінився на «Заблокований»
Запит на створення скарги на користувача	Backend.Web	UserReportCreating_CreatesUserReport	Дані форми скарги	Створена скарга на користувача
Запит на створення скарги на тему	Backend.Web	TopicReportCreating_CreatesTopicReport	Дані форми скарги	Створена скарга на тему
Запит на створення скарги на коментар	Backend.Web	CommentaryReportCreating_CreatesUserReport	Дані форми скарги	Створена скарга на коментар

4.3 Валідація та верифікація додатка

Верифікація додатку частково відбувалася за використання технології StyleCop [46], яка допомагає виявити елементарні невідповідності в логіці написаного коду, максимально оптимізувати його та привести до єдиного стандарту. Таким чином, код було позбавлено основних помилок, які завадять на початковому етапі коректно відпрацювати програмі та приведено до загального стандарту написання коду. Також верифікація проводилася за допомогою регулярного складання діаграм класів кожного модуля, що дозволяло постійно спостерігати за коректністю написаної архітектури та звіряти, чи відповідає архітектура кожного з модулів тій, яка була складена на етапі проектування додатку.

Валідація додатку відбувалася за допомогою unit-тестування та за допомогою звірення кінцевого результату написання коду з діаграмою варіантів використання, складеною на етапі проектування та звіренням з технічним завданням, описаним на етапі проектування. На етапі валідації додатку перевірялася коректність та відповідність функціоналу, заявленого на етапі проектування. Наявний функціонал проходив перевірку на безперебійність та безпомилковість виконання в будь-яких умовах та при будь-яких вхідних даних.

4.4 Аналіз результатів тестування додатка

Після проведення усіх написаних тестів, було проаналізовано результати кожного з проведених тестів. Результати проведення тестування додатка за допомогою unit-тестування наведено у таблиці 4.15.

Таблиця 4.15 – Результати проведеного unit-тестування додатку

Назва сценарію	Метод	Вхідні дані	Очікуваний результат	Результат
1	2	2	4	5
Отримання користувача по його унікальному ідентифікатору	GetUserByValidId _ReturnsUser	Коректний унікальний ідентифікатора користувача	Екземпляр сутності користувача	Правильно
Отримання користувача по його унікальному нікнейму	GetUserByValid Username _ReturnsUser	Коректний унікальний нікнейм користувача	Екземпляр сутності користувача	Правильно
Оновлення даних про користувача в базі даних	UpdateUser _UpdatesUser	Екземпляр сутності користувача	Оновлені дані про користувача	Правильно
Помилка операції реєстрації при введенні некоректних даних	GetRegisterResponse ByInvalidRegData _ReturnsNull	Некоректні реєстраційні дані	Null-значення	Правильно
Отримання токена доступу	GetAccessToken _ReturnsAccessToken	Екземпляр сутності користувача	Токен доступу	Правильно
Отримання токена відновлення	GetRefreshToken _ReturnsRefreshToken		Токен відновлення	Правильно
Отримання значення true при валідації коректного токена доступу	ValidTokenValidation _ReturnsTrue	Коректний токен доступу	Значення true	Правильно
Отримання токена доступу в результаті пошуку за коректними даними	GetValidAccessToken _ReturnsAccessToken	Коректний токен доступу	Токен доступу	Правильно

Продовження таблиці 4.15

1	2	2	4	5
Збереження до бази сервісу токена доступу	SaveAccessToken _SavesAccessToken	Токен доступу	Збережений токен доступу	Правильно
Отримання скарг на користувачів	GetUsersReports _ReturnsUsersReports	Токен доступу	Список скарг на користувачів	Правильно
Блокування теми	TopicBanning _BansTopic	Токен доступу та ідентифікатор користувача	Статус теми змінився на «Заблокований»	Правильно
Створення скарги на коментар	CommentaryReport Creating_Creates UserReport	Дані форми скарги	Створена скарга на коментар	Правильно
Отримання останніх тем	GetLatestTopics _ReturnsLatestTopics		Список останніх тем	Правильно
Отримання улюблених тем	GetFavoriteTopics _ReturnsFavoriteTopics	Токен доступу	Список улюблених тем	Правильно
Створення теми	TopicCreation _CreatesTopic	Токен доступу, заповнена форма створення теми	Створена тема	Правильно
Пошук по темах	TopicSearch _ReturnsTopics	Заповнена форма пошуку тем	Список тем, які відповідають критеріям пошукового запиту	Правильно
Оновлення коментаря	CommentaryUpdating _UpdatesCommentary	Токен доступу, заповнена форма оновлення коментаря	Оновлені дані коментаря	Правильно
Видалення коментаря	CommentaryMarking Deleted_Marks CommentaryDeleted	Токен доступу	Статус коментаря змінено на «Видалений користувачем»	Правильно
Отримання списку унікальних ідентифікаторів обговорених тем	GetLatestDiscussed TopicsIds_Returns LatestDiscussed TopicsIds	Токен доступу	Список ідентифікаторів обговорених тем	Правильно
Отримання реакцій на тему	GetTopicReactions _ReturnsTopicReactions	Ідентифікатор теми	Список реакцій	Правильно
Отримання реакцій на коментар	GetCommentary Reaction_Returns CommentaryReactions	Ідентифікатор коментаря	Список реакцій	Правильно
Запит на реєстрацію користувача	RegisterRequest _RegistersUser	Заповнена форма реєстрації	Зареєстрований новий користувач	Правильно

Продовження таблиці 4.15

1	2	2	4	5
Запит на відновлення токену	RefreshTokenRequest _RefreshesToken	Токен відновлення	Отримано новий токен	Правильно
Запит на отримання інформації про даного користувача	GetMe_Returns UserInformation	Токен доступу	Отримано дані про користувача	Правильно
Запит на отримання список тегів інтересів	GetUserInterests ReturnsUserInterests	Токен доступу	Список інтересів	Правильно
Запит на оновлення списку тегів інтересів	UpdateUserInterests _UpdatesUserInterests	Токен доступу, список тегів	Оновлений список інтересів	Правильно
Запит на отримання теми				Правильно
Запит на отримання обговорених тем	GetDiscussedTopics _Returns DiscussedTopics	Токен доступу	Список останніх обговорених тем	Правильно
Запит на отримання власних тем	GetMyTopics _ReturnsMyTopics	Токен доступу	Список власних тем	Правильно
Запит на створення теми	TopicCreation _CreatesTopic	Токен доступу, заповнена форма створення теми	Створена тема	Правильно
Запит на видалення теми	TopicMarkingDeleted _MarksTopicDeleted	Токен доступу	Статус теми змінено на «Видалена користувачем»	Правильно
Запит на отримання реакцій на тему	GetTopicReactions _ReturnsTopicReactions	Ідентифікатор теми	Список реакцій	Правильно
Запит на отримання реакцій на коментар	GetCommentary Reaction_Returns CommentaryReactions	Ідентифікатор коментаря	Список реакцій	Правильно
Запит на реагування користувача на коментар	ReactingOnCommentary _CreatesUser CommentaryReaction	Токен доступу, ідентифікатор коментаря	Створена реакція користувача на коментар	Правильно
Запит на отримання скарг на користувачів	GetUsersReports _ReturnsUsersReports	Токен доступу	Список скарг на користувачів	Правильно
Запит на отримання скарг на теми	GetTopicReports _ReturnsTopicsReports	Токен доступу	Список скарг на теми	Правильно

Кінець таблиці 4.15

1	2	2	4	5
Запит на створення скарги на користувача	UserReportCreating _CreatesUserReport	Дані форми скарги	Створена скарга на користувача	Правильно
Запит на створення скарги на тему	TopicReportCreating _CreatesTopicReport	Дані форми скарги	Створена скарга на тему	Правильно
Запит на створення скарги на коментар	CommentaryReport Creating_Creates UserReport	Дані форми скарги	Створена скарга на коментар	Правильно

Отже, під час проведення тестування дипломного проекту було проведено аналіз усіх важливих місць додатку, збої в яких можуть призвести до значного витоку даних або зміни існуючих даних та призвести до збоїв у роботі платформи.

Після проведення аналізу усіх вразливих місць додатку, було обрано метод тестування – unit-тестування.

Після вибору методу тестування було написано усі необхідні тести, проведено безпосереднє тестування. Після проведеного тестування було проаналізовано результати усіх проведених тестів та визначено, що усі тести було проведено успішно.

Отже, проаналізувавши написання, проведення та підбиття результатів тестування, можна зробити висновок, що платформа працює справно та за її функціоналом не спостерігається неочікуваної поведінки або неправильної обробки вхідних даних.

Таким чином в результаті виконання четвертого розділу було проаналізовано наявні методи тестування та обрано найоптимальніший з них у контексті функціоналу даної платформи. Після визначення методу тестування платформу було протестовано, проведено валідацію та верифікацію платформи. Результати проведеного тестування, валідації та верифікації показали, що платформа працює справно, аномалії у її роботі відсутні, усі модулі приймають та обробляють дані коректно.

ВИСНОВКИ

У результаті виконання даного дипломного проекту розроблено веб-платформу для налаштування комунікації, а саме створення тем та проведення дискусій між користувачами мережі Інтернет.

У першому розділі було досліджено предметну область веб-платформи, а також усі найпопулярніші з наявних аналогічних додатків, платформ та сервісів.

В другому розділі було спроектовано усі основні та допоміжні модулі, спроектовано базу даних, створено ескізи усіх майбутніх сторінок та компонентів клієнтского додатку. Під час проектування дипломного проекту було обрано усі технології, необхідні для максимально зручної та ефективної реаоізації уього раніше спроектованого функціоналу.

У третьому розділі було розроблено базу даних, також було реалізовано усі раніше спроектовані модулі та частини платформи як на серверній стороні, так і на клієнтській стороні платформи. Також було описано керівництво користувача з поясненням роботи усього функціоналу платформи через призму їх використання користувачем.

У четвертому розділі було проведено детальне тестування усіх модулів та підмодулів, що дозволило перевірити платформу на коректність роботи на кожному з наявних етапів роботи, а також перевірити коректність обробки вхідних даних як коректних, так і некоректних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Л. П. Бедратюк. Дипломний проект: методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун, О. М. Яшина. – Хмельницький: ХНУ, 2020. – 77 с.CSS [Електронний ресурс] / CSS – Режим доступу до ресурсу: https://devdocs.io/css/. (дата звернення – 27.01.2022).

2. WebsiteRating. URL: https://www.websiterating.com/research/facebook-statistics/ (дата звернення: 12.03.2022).

3. Web-Promo. URL: https://web-promo.ua/ua/blog/dinamika-rosta-auditorii-socialnyh-setej-cravnivaem-kvartalnye-otchety-datareportal-za-2020-i-2021-gody/ (дата звернення: 15.03.2022).

4. Marketing Media Review. URL: https://mmr.ua/show/kolichestvo-polzovatelej-soczialnyh-setej-v-mire-prevysilo-4-5-milliarda, 2018. – 310с (дата звернення: 14.03.2022).

5. BlazingByte. URL: https://blazingbyte.site (дата звернення: 18.03.2022).

6. RaccoonsGames. URL: https://www.raccoons.games (дата звернення: 18.03.2022).

7. J. C. R. Licklider. Man-Computer Symbiosis. – New York: Portfolio/Penguin, 1960. – 260с.

8. P. Baran. Coverage estimate of FM, TV and Power Facilities Useful in broadband distributed network. – The Rand Corporation: PrintTM, 1962. – 340с.

9. Engineering and Technology History Wiki. URL: https://ethw.org/Milestones:Inception_of_the_ARPANET,_1969 (дата звернення: 20.04.2022).

10. Токар ЮА. URL: https://tokar.ua/read/19365 (дата звернення: 23.04.2022).

11. Towards Data Science. URL: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1 (дата звернення: 26.04.2022).

12. GeeksforGeeks. URL: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/ (дата звернення: 26.04.2022).

13. IBM. URL: https://www.ibm.com/cloud/learn/three-tier-architecture (дата звернення: 26.04.2022).

14. Березенко В. В. PR як сфера наукового знання : монографія / за заг. наук. ред. В. М. Манакіна. Запоріжжя : ЗНУ, 2015. 362 с.

15. Бутко М. П., Неживенко А. П., Пепа Т. В. Економічна психологія : навч. посіб. / за ред. М. П. Бутко. Київ : ЦУЛ, 2016. 232 с.

16. Дахно І. І., Алієва-Барановська В.М. Право інтелектуальної власності : навч. посіб. / за ред. І. І. Дахна. Київ : ЦУЛ, 2015. 560 с.

17. Микитів Г. В., Кондратенко Ю. Позатекстові елементи як засіб формування медіакультури читачів науково-популярних журналів. Актуальні проблеми медіаосвіти в Україні та світі : зб. тез доп. міжнар. наук.-практ. конф., м. Запоріжжя, 3-4 берез. 2016 р. Запоріжжя, 2016. С. 50–53.

18. Коваль Л. Плюси і мінуси дистанційної роботи. *Урядовий кур'єр*. 2017. 1 листоп. (№ 205). С. 5.

19. Bletskan D. I., Glukhov K. E., Frolova V. V. Electronic structure of 2H-SnSe2: ab initio modeling and comparison with experiment. Semiconductor Physics Quantum Electronics & Optoelectronics. 2016. Vol. 19, No 1. P. 98–108.

20. Freefeast of Info Feeding Your Brains. URL: https://freefeast.info/tutorials-for-beginners/dotnet-tutorials/3-tier-vs-3-layer-architecture-difference-between-3-layer-and-3-tier-architecture/ (дата звернення: 01.05.2022).

21. Quora. URL: https://www.quora.com/What-is-server-design (дата звернення: 01.05.2022).

22. Microsoft Docs. URL: https://docs.microsoft.com/en-us/ef/core/ (дата звернення: 04.05.2022).

23. TutorialsTeacher. URL: https://www.tutorialsteacher.com/linq/what-is-linq (дата звернення: 04.05.2022).

24. Microsoft Docs. URL: https://docs.microsoft.com/en/dotnet /system.security.cryptography.rfc2898 (дата звернення: 04.05.2022).

25. JWT. URL: https://jwt.io/introduction (дата звернення: 04.05.2022).

26. C# Corner. URL: https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/ (дата звернення: 12.05.2022).

27. Lattice Semiconductor. URL: https://latticesemi.com/reference-designs/aesdecryption-documentation (дата звернення: 12.05.2022).

28. Swagger. URL: https://swagger.io/docs/specification/about/ (дата звернення: 12.05.2022).

29. Vue.js. URL: https://vuejs.org/guide/introduction.html (дата звернення: 12.05.2022).

30. Vuex Documentation. URL: https://vuex.vuejs.org (дата звернення: 12.05.2022).

31. Vue Router Documentation. URL: https://router.vuejs.org/introduction/ (дата звернення: 12.05.2022).

32. Red Hat. URL: https://www.redhat.com/en/topics/containers/what-is-kubernetes (дата звернення: 24.05.2022).

33. Opensource.com. URL: https://opensource.com/resources/what-docker (дата звернення: 24.05.2022).

34. Computer Hope. URL: https://www.computerhope.com/virtualbox.htm (дата звернення: 25.05.2022).

35. Entity Framework Tutorial. URL: https://www.entityframework-tutorial.net/what-is-code-first.aspx (дата звернення: 25.05.2022).

36. CloudBees. URL: https://www.cloudbees.com/blog/database-migration (дата звернення: 26.05.2022).

37. DevIQ. URL: https://deviq.com/design-patterns/repository-pattern (дата звернення: 27.05.2022).

38. Instaclustr. URL: https://www.instaclustr.com/blog/what-is-redis/ (дата звернення: 27.05.2022).

39. Microsoft Docs. URL: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6 (дата звернення: 27.05.2022).

40. Vue.js. URL: https://vuejs.org/guide/essentials/computed.html (дата звернення: 28.05.2022).

41. ListPrivilege.com. URL: https://leastprivilege.com/2014/10/02/401-vs-403/ (дата звернення: 29.05.2022).

42. Can I use. URL: https://caniuse.com/documents/mdn-javascript_builtins_function_apply (дата звернення: 30.05.2022).

43. Mentormate. URL: https://mentormate.com/blog/what-is-heroku-used-for-cloud-development/ (дата звернення: 31.05.2022).

44. Smartbear. URL: https://smartbear.com/learn/automated-testing/what-is-unit-testing/ (дата звернення: 31.05.2022).

45. TechTarget. URL: https://www.techtarget.com/definition/integration-testing/ (дата звернення: 01.06.2022).

46. JonDJones. URL: https://www.jondjones.com/frameworks/what-is-stylecop/ (дата звернення: 01.06.2022).

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки веб-платформи для обміну інформацією за інтересами «TOD.». Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: «Веб-платформа для обміну інформацією за інтересами «TOD.»».

2 Призначення розробки

2.1 Функціональне призначення

Веб-платформа для обміну інформацією за інтересами «TOD.» призначена для організації спілкування то інших видів комунікації між користувачами цієї платформи для обговорення цікавих цим користувачам тем.

2.2 Експлуатаційне призначення

Веб-платформа повинна експлуатуватися на будь-якому пристрої, який має доступ до мережі Інтернет та має встановлений веб-браузер.

Кінцевим користувачем веб-платформи може стати будь-яка особа.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Веб-платформа повинна забезпечувати можливість виконання таких функцій:

- можливість реєстрації;
- можливість авторизації;
- можливість перегляду усіх тем;
- можливість перегляду улюблених тем;
- можливість перегляду рекомендованих тем;
- можливість перегляду власних тем;
- можливість перегляду обговорених тем;
- можливість переглянути профіль;
- можливість редагувати власний профіль;

- можливість переглядати власний рейтинг;
- можливість переглядати власні теги інтересів;
- можливість редагувати власні теги інтересів;
- можливість залишати коментарі під темами;
- можливість редагувати власні коментарі;
- можливість видалення власного коментаря;
- можливість створення нової теми;
- можливість редагування власної теми;
- можливість видалення власної теми;
- можливість пошуку серед усіх тем сайту;
- можливість додавання тем до списку улюблених;
- можливість блокування користувача для адміністратора;
- можливість видалення будь-якого коментаря для адміністратора;
- можливість видалення будь-якої теми для адміністратора.

3.2 Вимоги до надійності

Додаток повинен забезпечувати наступні вимоги до надійності:

- усі деструктивні дії (видалення теми, коментаря) повинні отримувати додаткове підтвердження від користувача перед тим, як ці зміни будуть застосовані.

3.3 Умови експлуатації та вимоги до технічних засобів

Для успішного користування даною веб-платформою користувачу необхідно мати стабільне інтернет з’єднання, додаток веб-браузера, а також його пристрій повинен відповідати наступним мінімальним вимогам:

- будь-яка операційна система;
- будь-який двох ядерний процесор;
- 1 ГБ ОЗП;
- 200 МБ постійної вільної пам’яті;

3.4 Вимоги до інформаційної та програмної сумісності

При розробці веб-платформи буде використовуватися високорівнева об’єктно-орієнтована мова програмування C# і її фреймворк ASP.NET для написання серверної частини платформи. Також, для написання лицьової частини сайту буде використана

мова програмування JavaScript, зокрема один з його фреймворків – Vue.js. В якості СКБД для зберігання даних буде використаний MS SQL Server, оскільки обрана для написання серверної частини платформи мова програмування має зручний інструмент для роботи з цією СКБД, а саме – Entity Framework Core.

3.5 Спеціальні вимоги

Веб-платформа повинна мати зручний, зрозумілий та привабливий для користувача дизайн.

4 Вимоги до програмної документації

При здачі проекта замовнику, розробник зобов’язається надати наступні документи:

- текстовий вміст програми;
- опис розробленої програми;
- технічне завдання проекту;
- керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки веб-платформи для обміну інформацією за інтересами «TOD.» подані у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.2022 – 31.01.2022	Дослідження предметної області та постановка задачі. Технічне завдання на розробку ПЗ	Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання
Ескізний проект 01.02.22 – 28.02.22	Проектування ПЗ. Детальний проект ПЗ	Проектування програмного забезпечення

Кінець таблиці А.1

Програмна реалізація 29.02.22 – 19.03.22	Програмна реалізація. Розроблений програмний засіб	Програмна реалізація
Тестування ПЗ 01.03.22 – 10.04.22	Тестування програмного забезпечення. Тест-кейси і тестові сценарії	Тестування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення

6 Порядок контролю та приймання

Контроль над якістю програмного забезпечення здійснюватиметься користувачами системи, які будуть підключені до тестування платформи на етапі тестування ПЗ. Приймання відбувається після виставлення платформи на хостинг на одному з відповідних сервісів.

ДОДАТОК Б

(обов’язковий)

ПРОГРАМНИЙ КОД

Код класу TopicService

```
public class TopicService : ITopicService
      {
        private readonly ITopicRepository topicRepository;
        private readonly ITopicTagRepository topicTagRepository;
        private readonly IUserTopicRepository userTopicRepository;
        private readonly IFavoriteRepository favoriteRepository;
        private readonly IUserService userService;
        private readonly ICommentaryService commentaryService;
        private readonly ITagService tagService;
        private readonly IReactionService reactionService;
        private readonly IInterestTagService interestTagService;
        private readonly IContentValidator contentValidator;

        public TopicService(ITopicRepository topicRepository,
            ITopicTagRepository topicTagRepository,
            IUserTopicRepository userTopicRepository,
            IFavoriteRepository favoriteRepository,
            IUserService userService,
            ICommentaryService commentaryService,
            ITagService tagService,
            IReactionService reactionService,
            IInterestTagService interestTagService,
            IContentValidator contentValidator)
            {
            this.topicRepository = topicRepository;
            this.topicTagRepository = topicTagRepository;
            this.userTopicRepository = userTopicRepository;
            this.favoriteRepository = favoriteRepository;
            this.userService = userService;
            this.commentaryService = commentaryService;
            this.tagService = tagService;
            this.reactionService = reactionService;
            this.interestTagService = interestTagService;
            this.contentValidator = contentValidator;
            }

        public async Task<Topic> GetByIdAsync(int topicId)
        {
            return await this.topicRepository.GetAsync(topicId);
        }

        public async Task<TopicData> GetTopicByIdAsync(int userId, int id)
        {
            var topic = await this.contentValidator.GetAndValidateTopicAsync(id);

            var authorId = await
this.userTopicRepository.GetUserIdByTopicIdAsync(id);

            var user = await this.contentValidator.GetAndValidateUserAsync(authorId);

            var tags = (await this.tagService.GetByTopicIdAsync(id)).ToList();

            var reactions = await this.reactionService.GetByTopicIdAsync(id);

            var positive = reactions.Where(r => r.ReactionValue ==
ReactionValue.Positive).Count();
            var negative = reactions.Count - positive;

            var fromFavorite = (FavoriteTopic)null;
```

```
            if (userId != 0)
            {
                fromFavorite = await
this.favoriteRepository.GetByUserIdAndTopicId(userId, topic.Id);
            }

            return new TopicData
            {
                Id = topic.Id,
                Title = topic.Title,
                CreatedUtc = topic.CreatedUtc,
                Author = new UserDto(user),
                Tags = tags,
                Rating = positive - negative,
                IsInFavorite = fromFavorite != null
            };
        }

        public async Task<GetTopicsResponse> GetTopicsAsync(int userId, int skip = 0,
int offset = 20)
        {
            var topics = await this.topicRepository.GetRangeAsync(skip, offset);

            if (topics == null)
            {
                return new GetTopicsResponse
                {
                    Topics = new()
                };
            }

            var topicsData = new List<TopicData>();
            foreach (var topic in topics)
            {
                var authorId = await
this.userTopicRepository.GetUserIdByTopicIdAsync(topic.Id);

                var author = await this.userService.GetByIdAsync(authorId);

                var tags = (await
this.tagService.GetByTopicIdAsync(topic.Id)).ToList();

                var rating = await this.GetTopicRating(topic.Id);

                var fromFavorite = (FavoriteTopic)null;
                if (userId != 0)
                {
                    fromFavorite = await
this.favoriteRepository.GetByUserIdAndTopicId(userId, topic.Id);
                }

                topicsData.Add(new TopicData
                {
                    Id = topic.Id,
                    Title = topic.Title,
                    CreatedUtc = topic.CreatedUtc,
                    Author = new UserDto(author),
                    Tags = tags,
                    Rating = rating,
                    IsInFavorite = fromFavorite != null
                });
            }

            return new GetTopicsResponse
```

```
            {
                Topics = topicsData.Skip(skip).Take(offset).ToList()
            };
        }

        public async Task<GetTopicsResponse> GetFavoritesAsync(int userId, int skip =
0, int offset = 20)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);

            var userFavoritesIds = this.favoriteRepository.GetByUserId(userId);

            if (userFavoritesIds.Count == 0)
            {
                return new GetTopicsResponse
                {
                    Topics = new()
                };
            }

            var topicsData = await this.GetTopicsDataByTopicsIds(userFavoritesIds);

            return new GetTopicsResponse
            {
                Topics = topicsData.Skip(skip).Take(offset).ToList()
            };
        }

        public async Task<GetTopicsResponse> GetMyTopicsAsync(int userId, int skip =
0, int offset = 20)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);

            var userTopicsIds = this.userTopicRepository.GetTopicsIdByUserId(userId);

            if (userTopicsIds.Count == 0)
            {
                return new GetTopicsResponse
                {
                    Topics = new()
                };
            }

            var topics = await this.GetTopicsDataByTopicsIds(userTopicsIds);

            return new GetTopicsResponse
            {
                Topics = topics.Skip(skip).Take(offset).ToList()
            };
        }

        public async Task<GetTopicsResponse> GetDiscussedTopicsAsync(int userId, int
skip = 0, int offset = 20)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);

            var discussedTopicsIds = await
this.commentaryService.GetLatestDiscussedTopicsIds(userId);
            var discussedTopics = new List<TopicData>();
            foreach (var topicId in discussedTopicsIds)
            {
                var topicData = await this.GetTopicByIdAsync(userId, topicId);
                if (topicData == null)
                {
```

```
                    continue;
                }
                discussedTopics.Add(topicData);
            }

            return new GetTopicsResponse
            {
                Topics = discussedTopics.Skip(skip).Take(offset).ToList()
            };
        }

        public async Task<GetTopicsResponse> GetRecommendedTopicsAsync(int userId,
int skip = 0, int offset = 20)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);

            var interestTags = await
this.interestTagService.GetUserInterestTagsAsync(userId);
            var topicsIds = new List<int>();
            foreach (var tag in interestTags.Tags)
            {
                var topicsIdsByTag = await
this.topicTagRepository.GetByTagIdAsync(tag.Id);
                topicsIds = topicsIds.Union(topicsIdsByTag).ToList();
            }

            var topics = (await
this.GetTopicsDataByTopicsIds(topicsIds.OrderByDescending(t => t).ToList()))
                .Skip(skip).Take(offset).ToList();

            return new GetTopicsResponse
            {
                Topics = topics
            };
        }

        public async Task<CreateTopicResponse> CreateAsync(CreateTopicRequest
request, int userId)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);

            var topicSearchResult = await
this.topicRepository.GetByTitleAsync(request.Title);

            if (topicSearchResult != null)
            {
                throw new TopicAlreadyExistsException();
            }

            var topic = new Topic
            {
                Title = request.Title,
                CreatedUtc = DateTime.UtcNow,
                Status = ContentStatus.Ok
            };

            topic = await this.topicRepository.CreateAsync(topic);

            await this.CreateUserTopicAsync(userId, topic.Id);

            var topicTags = await this.GetTopicTagsAsync(userId, topic.Id,
request.Tags);

            return new CreateTopicResponse
```

```
            {
                Id = topic.Id,
                Title = topic.Title,
                Author = new UserDto(user),
                CreatedUtc = topic.CreatedUtc,
                Tags = topicTags
            };
        }

        public async Task<TopicData> UpdateAsync(int userId, int topicId,
CreateTopicRequest request)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);
            var authorId = await
this.userTopicRepository.GetUserIdByTopicIdAsync(topicId);
            if (userId != authorId)
            {
                throw new PermissionDeniedException();
            }

            var topic = await
this.contentValidator.GetAndValidateTopicAsync(topicId);

            var topicByTitle = await
this.topicRepository.GetByTitleAsync(request.Title);
            if (topicByTitle != null)
            {
                throw new TopicAlreadyExistsException();
            }

            topic.Title = request.Title;
            topic = await this.topicRepository.UpdateAsync(topic);
            await this.topicTagRepository.DeleteTopicTagsAsync(topicId);
            var topicTags = await this.GetTopicTagsAsync(userId, topicId,
request.Tags);

            var rating = await this.GetTopicRating(topicId);

            return new TopicData
            {
                Id = topic.Id,
                Title = topic.Title,
                CreatedUtc = topic.CreatedUtc,
                Author = new UserDto(user),
                Tags = topicTags,
                Rating = rating,
                IsInFavorite = false,
            };
        }

        public async Task UpdateAsync(Topic topic)
        {
            await this.topicRepository.UpdateAsync(topic);
        }

        public async Task<bool> AddToFavoritesAsync(int topicId, int userId)
        {
            var topic = await
this.contentValidator.GetAndValidateTopicAsync(topicId);

            var user = await this.contentValidator.GetAndValidateUserAsync(userId);

            var authorId = await
this.userTopicRepository.GetUserIdByTopicIdAsync(topicId);
```

```
            if (userId == authorId)
            {
                throw new ContentBelongsToYouException();
            }

            var fromFavorites = await
this.favoriteRepository.GetByUserIdAndTopicId(userId, topicId);

            if (fromFavorites != null)
            {
                throw new TopicAlreadyInFavoritesException();
            }

            var favorite = new FavoriteTopic
            {
                UserId = userId,
                TopicId = topicId
            };

            await this.favoriteRepository.CreateAsync(favorite);

            await this.UpdateAuthorRatingAsync(authorId, 1);

            return true;
        }

        public async Task<GetTopicsResponse> SearchTopicsAsync(TopicSearchRequest
request)
        {
            if (string.IsNullOrWhiteSpace(request.Author)
                && string.IsNullOrWhiteSpace(request.Title)
                && (request.Tags == null || request.Tags.Count == 0))
            {
                return null;
            }

            var topicsIds = (List<int>)null;

            if (!string.IsNullOrWhiteSpace(request.Author))
            {
                topicsIds = new();
                topicsIds.AddRange(await
this.SearchTopicByAuthorAsync(request.Author));
            }

            if (!string.IsNullOrWhiteSpace(request.Title))
            {
                topicsIds = await this.SearchTopicByTitleAsync(request.Title,
topicsIds);
            }

            if (request.Tags == null || request.Tags.Count != 0)
            {
                topicsIds = await this.SearchTopicByTagsAsync(request.Tags,
topicsIds);
            }

            if (topicsIds == null || topicsIds.Count == 0)
            {
                throw new NotFoundException(ContentType.Topic);
            }

            var topicsData = await this.GetTopicsDataByTopicsIds(topicsIds);
```

```
            return new GetTopicsResponse
            {
                Topics = topicsData
            };
        }

        public async Task MarkTopicDeletedAsync(int userId, int topicId)
        {
            var user = await this.contentValidator.GetAndValidateUserAsync(userId);
            var authorId = await
this.userTopicRepository.GetUserIdByTopicIdAsync(topicId);
            if (userId != authorId)
            {
                throw new PermissionDeniedException();
            }

            var topic = await
this.contentValidator.GetAndValidateTopicAsync(topicId);
            topic.Status = ContentStatus.DeletedByOwner;
            await this.topicRepository.UpdateAsync(topic);
        }

        private async Task<List<TopicData>> GetTopicsDataByTopicsIds(List<int>
topicsIds)
        {
            var topicsData = new List<TopicData>();

            foreach (var topicId in topicsIds)
            {
                var topic = await this.topicRepository.GetAsync(topicId);

                if (topic == null || topic.Status == ContentStatus.Banned ||
topic.Status == ContentStatus.DeletedByOwner)
                {
                    continue;
                }

                var authorId = await
this.userTopicRepository.GetUserIdByTopicIdAsync(topicId);

                var author = await this.userService.GetByIdAsync(authorId);

                if (author == null || author.Status == ContentStatus.Banned ||
author.Status == ContentStatus.DeletedByOwner)
                {
                    continue;
                }

                var tags = (await
this.tagService.GetByTopicIdAsync(topicId)).ToList();

                var reactions = await
this.reactionService.GetByTopicIdAsync(topicId);

                var positive = reactions.Where(r => r.ReactionValue ==
ReactionValue.Positive).Count();
                var negative = reactions.Count - positive;

                topicsData.Add(new TopicData
                {
                    Id = topicId,
                    Title = topic.Title,
                    CreatedUtc = topic.CreatedUtc,
```

```
                Author = new UserDto(author),
                Tags = tags,
                Rating = positive - negative
            });
        }

        return topicsData;
    }
```

Код класу JwtAccountService

```
public class JwtAccountService : IAccountService
      {
          private readonly IUserRepository userRepository;
        private readonly IPasswordHasher passwordHasher;
        private readonly ITokenService tokenService;
        private readonly IRedisService redisService;
        private readonly JwtTokenConfiguration jwtConfig;

          public JwtAccountService(
          IUserRepository userRepository,
          IPasswordHasher passwordHasher,
          ITokenService tokenService,
          IRedisService redisService,
          JwtTokenConfiguration jwtConfig)
          {
                this.userRepository = userRepository;
          this.passwordHasher = passwordHasher;
          this.tokenService = tokenService;
          this.redisService = redisService;
          this.jwtConfig = jwtConfig;
          }

        public async Task<UserDto> GetUserByEmailAsync(string email)
        {
            var user = await this.userRepository.GetUserByEmailAsync(email);

            if (user == null)
            {
                throw new NotFoundException(ContentType.User);
            }

            return new UserDto(user);
        }

        public async Task<UserDto> GetUserByUsernameAsync(string username)
        {
            var user = await this.userRepository.GetUserByUsernameAsync(username);

            if (user == null)
            {
                throw new NotFoundException(ContentType.User);
            }

            return new UserDto(user);
        }

        public async Task<UserDto> GetUserByIdAsync(int id)
        {
            var user = await this.userRepository.GetAsync(id);
```

```
            if (user == null)
            {
                throw new NotFoundException(ContentType.User);
            }

            return new UserDto(user);
        }

        public async Task<LoginResponse> LoginUserAsync(LoginRequest request)
        {
            var user = await this.userRepository.GetUserByEmailAsync(request.Login);

            if (user == null)
            {
                user = await
this.userRepository.GetUserByUsernameAsync(request.Login);

                if (user == null)
                {
                    throw new NotFoundException(ContentType.User);
                }
            }

            if (user.Status == ContentStatus.Banned)
            {
                throw new BannedContentException(ContentType.User);
            }

            if (user.Status == ContentStatus.DeletedByOwner)
            {
                throw new DeletedContentException(ContentType.User);
            }

            var passwordMatch = this.passwordHasher.VerifyPassword(request.Password,
user.PasswordHash);

            if (!passwordMatch)
            {
                throw new PasswordMismatchException();
            }

            return await LoginUserAsync(user);
        }

        public async Task<LoginResponse> LoginUserAsync(User user)
        {
            var accessToken = this.tokenService.GetAccessToken(user);
            var refreshToken = this.tokenService.GetRefreshToken();
            var deleteAfter =
TimeSpan.FromMinutes(this.jwtConfig.RefreshTokenExpiresInMinutes);

            await this.redisService.SaveAsync(accessToken,
user.PasswordHash.GetHashCode(), refreshToken, deleteAfter);

            return new LoginResponse
            {
                User = new UserDto(user),
                AccessToken = accessToken,
                RefreshToken = refreshToken,
                AccessTokenExpiresAt =
DateTime.UtcNow.AddMinutes(this.jwtConfig.AccessTokenExpiresInMinutes),
                RefreshTokenExpiresAt =
DateTime.UtcNow.AddMinutes(this.jwtConfig.RefreshTokenExpiresInMinutes)
```

```
            };
        }

        public async Task<User> RegisterUserAsync(RegisterRequest request)
        {
            var user = await this.userRepository.GetUserByEmailAsync(request.Email);

            if (user != null)
            {
                throw new EmailAlreadyTakenException();
            }

            user = await
this.userRepository.GetUserByUsernameAsync(request.Username);

            if (user != null)
            {
                throw new UsernameAlreadyTakenException();
            }

            user = new User
            {
                Username = request.Username,
                Email = request.Email,
                PasswordHash = this.passwordHasher.GetHash(request.Password),
                Rating = 0,
                Role = Role.User,
                Status = ContentStatus.Ok
            };

            await this.userRepository.CreateAsync(user);

            return user;
        }

        public async Task<LoginResponse> RefreshTokenAsync(RefreshTokenRequest
request)
        {
            var tokensOwner = await
this.tokenService.ValidateTokenRefreshing(request.AccessToken, request.RefreshToken);

            await this.redisService.DeleteAsync(request.AccessToken,
tokensOwner.PasswordHash.GetHashCode());

            var loginResponse = await this.LoginUserAsync(tokensOwner);

            return loginResponse;
        }
    }
```

Код класу AccountController

```
[Authorize]
      [ApiController]
      [Route("api/account")]
      public class AccountController : ControllerBase
      {
            private readonly IAccountService accountService;
            private readonly IInterestTagService interestTagService;
```

```
            public AccountController(IAccountService accountService,
                  IInterestTagService interestTagService)
            {
                  this.accountService = accountService;
                  this.interestTagService = interestTagService;
            }

            [AllowAnonymous]
            [HttpPost("register")]
            [ProducesResponseType(typeof(LoginResponse),
StatusCodes.Status201Created)]
            [ProducesResponseType(typeof(string), StatusCodes.Status400BadRequest)]
            public async Task<ActionResult<LoginResponse>> Register(RegisterRequest
request)
        {
                  User user;

            try
            {
                        user = await this.accountService.RegisterUserAsync(request);
            }
                  catch (EmailAlreadyTakenException ex)
            {
                        return BadRequest(ex.Message);
            }
                  catch (UsernameAlreadyTakenException ex)
            {
                        return BadRequest(ex.Message);
            }

                  var response = await this.accountService.LoginUserAsync(user);

                  return response;
        }

            [AllowAnonymous]
            [HttpPost("login")]
            [ProducesResponseType(typeof(LoginResponse), StatusCodes.Status200OK)]
            [ProducesResponseType(typeof(string), StatusCodes.Status401Unauthorized)]
            public async Task<ActionResult<LoginResponse>> Login(LoginRequest
request)
        {
                  try
            {
                        var response = await
this.accountService.LoginUserAsync(request);

                        return response;
            }
                  catch (NotFoundException ex)
            {
                        return Unauthorized(ex.Message);
            }
                  catch (PasswordMismatchException ex)
            {
                        return Unauthorized(ex.Message);
            }
        }

            [HttpPost("refresh-token")]
            [ProducesResponseType(typeof(LoginResponse), StatusCodes.Status200OK)]
            [ProducesResponseType(typeof(string), StatusCodes.Status401Unauthorized)]
            [ProducesResponseType(typeof(string), StatusCodes.Status403Forbidden)]
```

```
            public async Task<ActionResult<LoginResponse>>
RefreshToken(RefreshTokenRequest request)
        {
                try
            {
                        var response = await
this.accountService.RefreshTokenAsync(request);

                        return response;
            }
                catch (InvalidTokenException ex)
            {
                        return BadRequest(ex.Message);
            }
                catch (RedisException ex)
            {
                        return Forbid(ex.Message);
            }
        }

            [HttpGet("me")]
            [ProducesResponseType(typeof(UserDto), StatusCodes.Status200OK)]
            [ProducesResponseType(typeof(string), StatusCodes.Status401Unauthorized)]
            public async Task<ActionResult<UserDto>> GetMe()
        {
                try
            {
                        var currentUserId = base.HttpContext.GetCurrentUserId();

                        var user = await
this.accountService.GetUserByIdAsync(currentUserId);

                        return user;
            }
                catch (InvalidTokenException ex)
                {
                        return Unauthorized(ex.Message);
                }
                catch (NotFoundException ex)
            {
                        return Unauthorized( ex.Message);
            }
        }

            [HttpGet("interests")]
            [ProducesResponseType(typeof(InterestTagsResponse),
StatusCodes.Status200OK)]
            [ProducesResponseType(typeof(string), StatusCodes.Status401Unauthorized)]
            public async Task<ActionResult<InterestTagsResponse>>
GetUserInterestTags()
        {
                try
            {
                        var currentUserId = base.HttpContext.GetCurrentUserId();
                        var tags = await
this.interestTagService.GetUserInterestTagsAsync(currentUserId);

                        return tags;
                }
                catch (InvalidTokenException ex)
                {
                        return Unauthorized(ex.Message);
                }
                catch (NotFoundException ex)
```

```
                  {
                        return Unauthorized(ex.Message);
                  }
            }

            [HttpPut("interests")]
            [ProducesResponseType(typeof(InterestTagsResponse),
StatusCodes.Status200OK)]
            [ProducesResponseType(typeof(string), StatusCodes.Status401Unauthorized)]
            public async Task<ActionResult<InterestTagsResponse>>
UpdateUserTagInterests(UpdateInterestTagsRequest request)
            {
                  try
                  {
                        var currentUserId = base.HttpContext.GetCurrentUserId();
                        var tags = await
this.interestTagService.UpdateUserInterestTagsAsync(currentUserId, request.TagsText);

                        return tags;
                  }
                  catch (InvalidTokenException ex)
                  {
                        return Unauthorized(ex.Message);
                  }
                  catch (NotFoundException ex)
                  {
                        return Unauthorized(ex.Message);
                  }
            }
      }
```

Код компонента AuthorizationForm.vue

```
<template>
  <div style="display: flex; margin: 5% 10%;">
    <register-form @onRegisterSubmit="register"
:apiException="registerException"></register-form>

    <login-form @onLoginSubmit="login" :apiException="loginException"></login-form>
  </div>
</template>

<script>
import LoginForm from "../components/forms/LoginForm.vue";
import RegisterForm from "../components/forms/RegisterForm.vue";
import authenticationService from "../services/authenticationService";

export default {
  components: {
    LoginForm,
    RegisterForm,
  },
  data() {
    return {
      registerException: "",
      loginException: "",
    }
  },
  methods: {
    register: async function (registerObject) {
```

```
      try {
        await authenticationService.register(registerObject);
      } catch (error) {
        this.registerException = error.text;
        return;
      }

      this.$router.push("/feed");
    },
    login: async function (loginObject) {
      try {
        await authenticationService.login(loginObject.login, loginObject.password);
      } catch (error) {
        this.loginException = JSON.parse(error.text);
        return;
      }

      this.$router.push("/feed");
    }
  }
}
</script>

<style>
.auth-container {
  margin: auto 4%;
  width:50%;
  border-radius: 10px;
  background-color: #4F4F4F;
}
.auth-header {
  text-align: center;
  color: #00C896;
}
.auth-field {
  text-align: center;
  margin-bottom: 4%;
}
.auth-input {
  color: #00C896;
  background-color: #F2ECFF;
  border-radius: 10px;
  padding: 2% 2%;
  font-size: large;
  font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
  border: none;
  outline: none;
}
.auth-button {
  font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
  font-size: large;
  text-transform: uppercase;
  border-radius: 10px;
  background-color: #00C896;
  color: #F2ECFF;
  border: none;
  padding: 2% 4%;
}
.auth-button:hover {
  background-color: #518071;
}
</style>
```

Код сторінки Login.vue

```
<template>
  <div class="auth-container">
      <div>
        <h3 class="auth-header">Login if you already have an account:</h3>

        <div class="auth-field" :class="{ error: v$.login.$errors.length }">
          <input v-model="login" class="auth-input" placeholder="Email or Username"
/>
          <div style="color: red;" v-for="error of v$.login.$errors"
:key="error.$uid">
            <label>{{ error.$message }}</label>
          </div>
        </div>
        <div class="auth-field" :class="{ error: v$.lPassword.$errors.length }">
          <input v-model="lPassword" class="auth-input" type="password"
placeholder="Password"/>
          <div style="color: red;" v-for="error of v$.lPassword.$errors"
:key="error.$uid">
            <label>{{ error.$message }}</label>
          </div>
        </div>
        <div class="auth-field">
          <button class="auth-button" @click="this.submitForm()">Login</button>
        </div>
        <div v-if="apiException" class="auth-field" style="color: red; opacity:
0.8;">
          {{apiException}}
        </div>
      </div>
    </div>
</template>

<script>
import useVuelidate from "@vuelidate/core";
import { required, helpers } from "@vuelidate/validators";

export default {
  name: "login-form",
  setup() {
    return { v$: useVuelidate() }
  },
  data() {
    return {
      login: "",
      lPassword: "",
    }
  },
  validations() {
    return {
      login: {
        required: helpers.withMessage("Login is required", required),
      },
      lPassword: {
        required: helpers.withMessage("Password is required", required),
      },
    }
  },
  methods: {
    validate: function () {
```

```
      this.v$.$touch();

      if (this.v$.$invalid) {
        return false;
      }
      return true;
    },
    submitForm: function () {
      if (!this.validate())
      {
        return;
      }

      let loginObject = {
        login: this.login,
        password: this.lPassword,
      };

      this.$emit("onLoginSubmit", loginObject);
    }
  },
  props: {
    apiException: {
      Type: String,
    },
  },
}
</script>

<style>
.auth-container {
  margin: auto 4%;
  width:50%;
  border-radius: 10px;
  background-color: #4F4F4F;
}
.auth-header {
  text-align: center;
  color: #00C896;
}
.auth-field {
  text-align: center;
  margin-bottom: 4%;
}
.auth-input {
  color: #00C896;
  background-color: #F2ECFF;
  border-radius: 10px;
  padding: 2% 2%;
  font-size: large;
  font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
  border: none;
  outline: none;
}
</style>
```

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Веб-платформа TOD для обміну інформацією за інтересами

Виконав:
ст. гр. ІПЗ-18-1 Слутяк Євгеній
Керівник роботи:
к. т. н., доцент Яшина О. М.

Мета та завдання дослідження

- **Мета дослідження**: розробка веб-платформи для налаштування комунікації, а саме створення тем та проведення дискусій між користувачами мережі Інтернет .
- **Задачі:**
  - дослідити предметну область, провести постановку задачі;
  - спроектувати програмне забезпечення;
  - розробити програмну реалізацію;
  - провести тестування готового продукту.

Актуальність

- За статистикою, усі соціальні мережі та інші комунікаційні платформи нараховують більше 2 млрд унікальних користувачів. Ця цифра зростає кожного року в середньому на 200 мільйонів. Також за даними статистики комунікаційні платформи є такими додатками, якими люди користуються найчастіше: близько 70% користувачів платформ, які мають акаунти, користуються нею щодня. Не дивлячись на велику кількість наявних комунікаційних платформ, за статистикою щороку з'являється щонайменше одна платформа з кількістю користувачів більше 100 мільйонів.
- Тим не менше, за статистикою будь-яка компанія час від часу отримує відтік користувачів у зв'язку зі зміною політики чи підходу до роботи платформи. Кожна з платформ обов'язково має переважаючу більшість користувачів об'єднаних певними поглядами.

Предметна область

- Дана веб-платформа розробляється для налаштування комунікації між користувачами всесвітньої мережі Інтернет шляхом створення постів та їх коментування.
- Предметною областю даного дослідження є найпопулярніші із представлених комунікаційних платформ кількох типів: соціальних мереж, форумів та сайтів новин.
- В ході дослідження було визначено межі предметної області даного дипломного проекту.

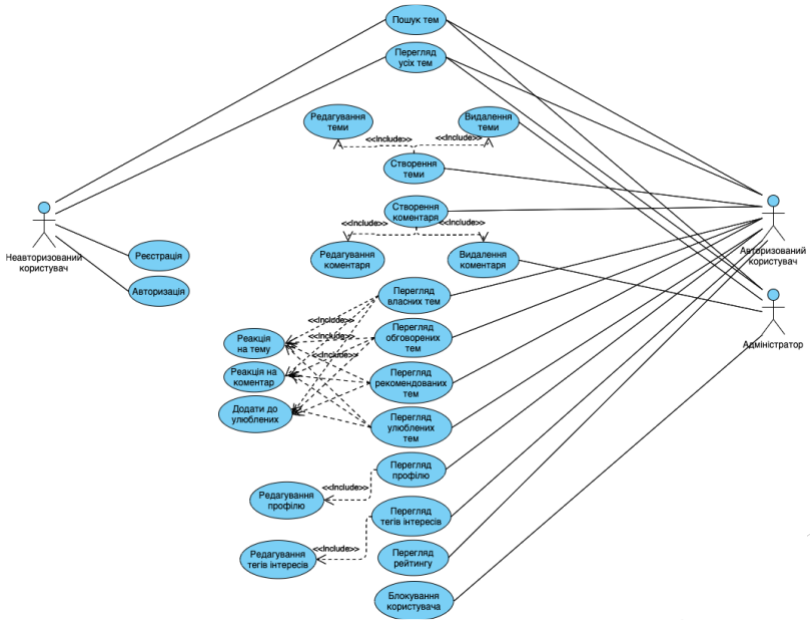
Аналіз стану проблеми та інших рішень

- **Переваги наявних рішень:**
  - велика кількість серверів;
  - імплементація найсучасніших технологій;
  - велика кількість додаткових сервісів;
- **Недоліки наявних рішень:**
  - заангажованість адміністрації;
  - великий вплив від рекламодавців;
  - великий вплив від інших компаній;
  - часта зміна політики.

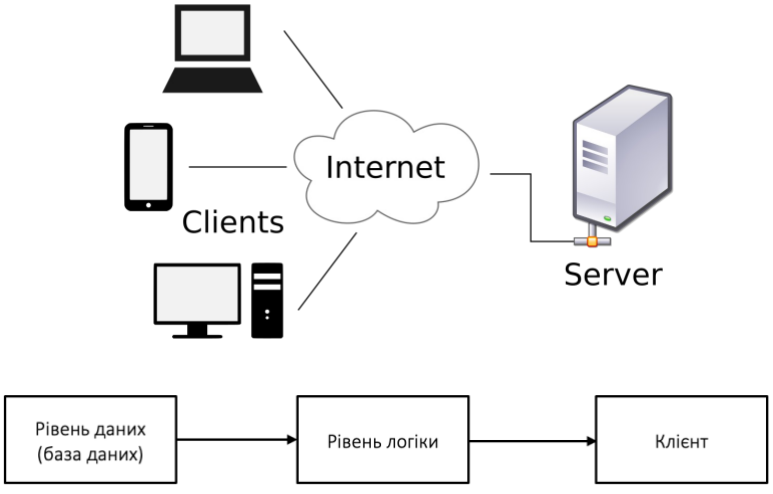
Вимоги до ресурсу

- Реєстрація/авторизація;
- перегляд тем;
- сортування списку тем;
- пошук тем;
- створення тем;
- редагування тем;
- видалення тем;
- коментування тем;
- редагування коментарів;
- видалення коментарів;
- відправка скарг на контент;
- блокування контенту.

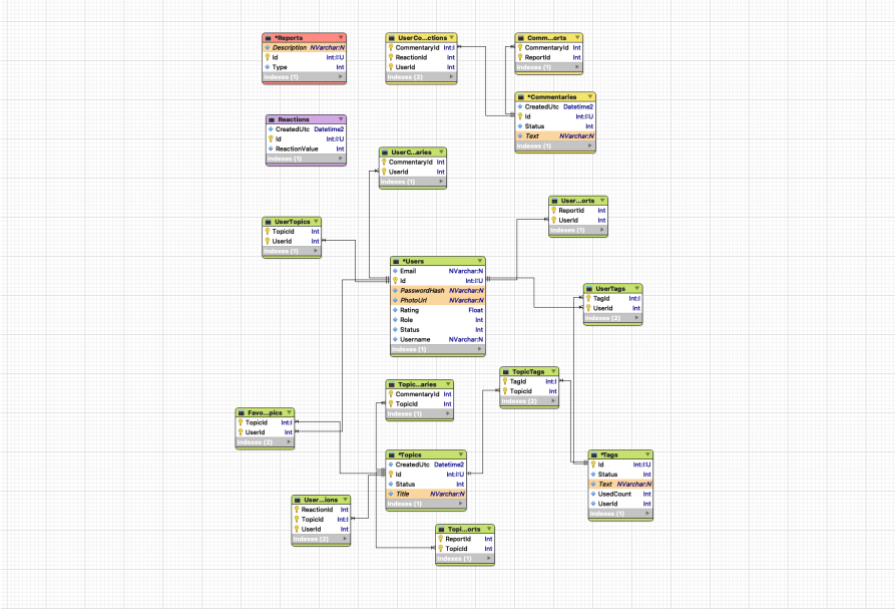
Діаграма варіантів використання



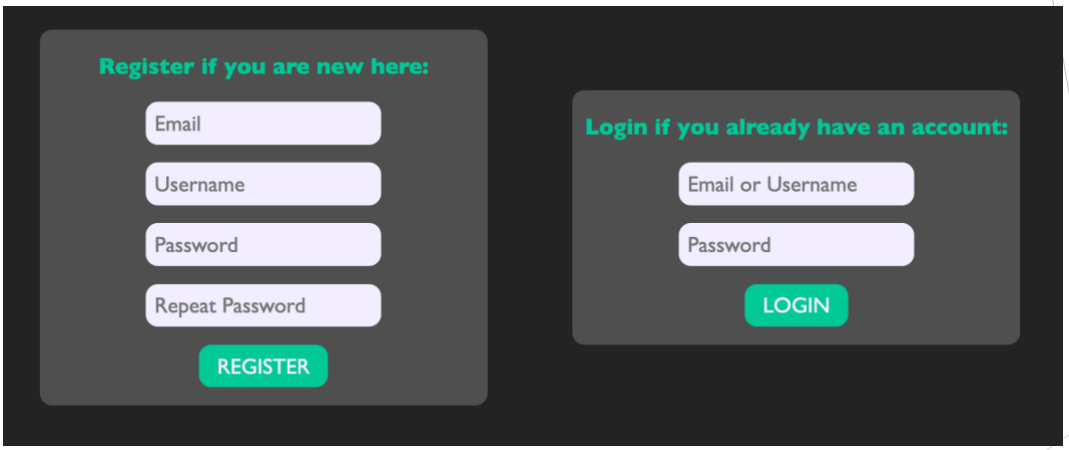
Архітектура



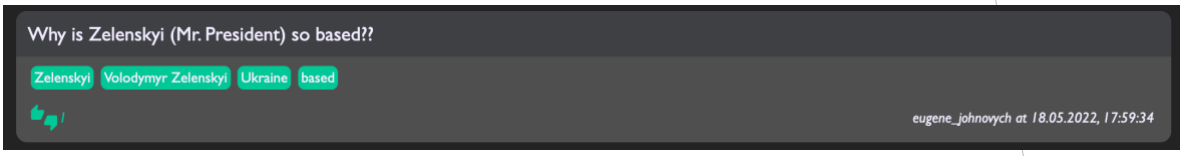
Модель бази даних



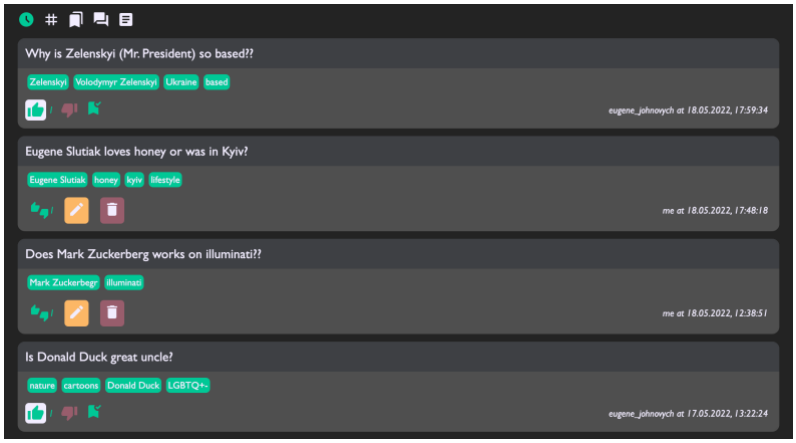
Сторінка входу в систему



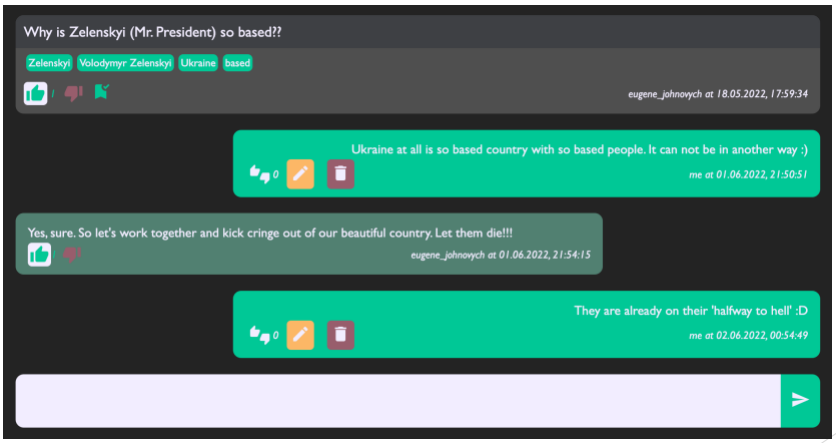
Варіації дизайну плитки теми



Сторінка відображення тем за фільтрами



Сторінка теми



Результати виконання дипломного проекту

- В результаті виконання дипломного проекту отримано веб-платформу TOD для обміну інформацією за інтересами шляхом публікації постів та вступання в дискусію шляхом коментування цих постів.

Перспективи розвитку продукту

- Дана веб-платформа спроектована таким чином, що може легко розширяти функціонал та при необхідному фінансуванні технічної та комерційної частини проекту може стати популярною комунікаційною платформою та охопити велику кількість користувачів.

Висновок

- проаналізовано предметну область;
- визначено переваги та недоліки існуючих рішень;
- розроблено технічне завдання проекту;
- спроектовано серверну та користувацьку частину платформи та розроблено її дизайн;
- розроблено програмну реалізацію веб-платформи;
- протестовано платформу на наявність помилок.

Дякую за увагу

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Слутяк Є. І.

Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-18-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений 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.

03.06.2022р.
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 15%**

	Документ		Сумарний збіг по Базі Даних	
ID: 104334 Назва: Веб-платформа TOD для обміну інформацією за інтересами Додано в БД: 2022-06-01 Автора: Є. І. Слутяк Керівники: О. М. Яшина Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	115500	963	6157 (5%)	78 (8%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1011420435

Дата перевірки:
01.06.2022 20:10:07 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2022 21:36:52 EEST

ID користувача:
100005589

Назва документа: **Слутяк Антиплагіат**

Кількість сторінок: **90** Кількість слів: **18184** Кількість символів: **146819** Розмір файлу: **1.31 MB** ID файлу: **1011301389**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.67%
Схожість

Найбільша схожість: **3.78%** з джерелом з Бібліотеки (ID файлу: **1008265198**)

3.04% Джерела з Інтернету	335	Сторінка 92
6.8% Джерела з Бібліотеки	123	Сторінка 94

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Підозріле форматування	28 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Слутяк Євгеній Іванович

Тема Веб-платформа TOD для обміну інформацією за інтересами

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень – ; кількість сторінок записки: 119

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті було досліджено і проаналізовано предметну область, усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення для обміну інформацією за інтересами . Було розглянуто інструменти для реалізації дипломного проекту, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконано відповідно до поставленого завдання та з дотриманням всіх вимог. Тема роботи в повній мірі обгрунтована й розкрита, проведено аналіз актуальності та відомих досліджень в межах обраної теми, поставлені завдання, які у роботі виконані, та розроблено необхідне програмне забезпечення.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі було доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено змістовний аналіз предметної області, її структурних та функціональних особливостей, проведено аналіз наявного програмно-технічного забезпечення даної предметної області та визначено функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення з обміну інформацією за інтересами. У другому розділі проведено аналіз наявних архітектур та функціональних структур додатків даного типу, розглянуто їх переваги і недоліки та визначено, що систему буде виконано за основою клієнт-серверної архітектури, зокрема серверна частина додатку буде виконана в тришаровій архітектурі. Також було описано структуру і усі моделі бази даних та безпосередньо спроектовано саму базу даних додатку. У третьому розділі виконано практичну розробку програмних модулів і описано їх особливості. В четвертому розділі було виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки на сьогодні є багато різний комунікативних та соціальних веб-платформ для обміну інформацією за інтересами, проте всі вони мають свої недоліки, які так чи інакше відкидають від них велику частину аудиторії. Також було застосовано сучасні

інформаційні технології для побудови програмного продукту та актуальні архітектурні рішення.

5. Негативні сторони проекту Суттєві недоліки відсутні. Деякі масштабні об'єкти пояснювальної записки, такі як Таблиця 1.2 і Таблиця 4.15, варто було розташувати у додатках. Також в пояснювальній записці не наведено докладний опис реалізованих функцій створеної інформаційної системи. Втім, це не знижує значущості одержаних результатів.

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

7. Відгук про дипломний проект в цілому Дипломний проект заслуговує позитивної оцінки, всі поставлені завдання виконані. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики дипломного проекту. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка дипломного проекту Дипломний проект виконаний у повному обсязі й відповідає поставленій задачі. Рекомендована оцінка «відмінно» (4,75=A=)
доцент к.т.н. кафедри КІІС
Яметчук Єлизавета Геннадіївна

РЕЦЕНЗЕНТ

"01" 06 2022р.

(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: «Веб-платформа TOD для обміну інформацією за інтересами»

Автор: Слутяк Євгеній Іванович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, 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.	
5	Інше:	

Підтвердження:

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

1) у тексті дипломної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, назвах розділів/підрозділів тощо та в назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 8,67% і адресується до 458 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь дипломної роботи.

Керівник _____ О. М. Яшина

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк