МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

_____

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система управління користувацьким контентом на веб-форумі»

здобувача групи ІН-93 Кисленка Ярослава Володимировича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ярослав КИСЛЕНКО

(підпис)

Керівник
старший викладач,
кандидат фізико-математичних наук Оксана ШОВКОПЛЯС _____

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-93 Кисленка Ярослава Володимировича

1. Тема роботи: «Інформаційна система управління користувацьким контентом на веб-форумі» затверджую наказом по СумДУ від *«09» червня 2023 р. № 0646-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 20 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Огляд веб-форумів й користувацького контенту, визначення стратегій ефективного управління користувацьким контентом. 2) Огляд та вибір технологій, що будуть використовуватись для розроблюваної інформаційної системи. 3) Розроблення інформаційної системи управління користувацьким контентом на веб-форумі. 4) Аналіз отриманих результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Огляд веб-форумів й користувацького контенту, визначення стратегій ефективного управління користувацьким контентом*		
2	*Огляд та вибір технологій, що будуть використовуватись для розроблюваної інформаційної системи*		
3	*Розроблення інформаційної системи управління користувацьким контентом на веб-форумі*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 65 стр., 33 рис., 3 табл., 1 додаток, 20 використаних джерел.

Обґрунтування актуальності теми роботи – зростаюча кількість користувацького контенту на веб-форумах зумовила нагальну потребу в ефективних системах управління контентом.

Об'єкт дослідження – процес управління користувацьким контентом.

Предмет дослідження – методи та стратегії ефективного управління користувацьким контентом.

Мета роботи – розроблення інформаційної системи, яка ефективно управляє користувацьким контентом на веб-форумі, використовуючи сучасний стек технологій MERN та існуючі бібліотеки.

Результати – розроблено інформаційну систему, яка дозволяє користувачам брати активну участь на форумі, ставлячи запитання, надаючи відповіді, коментарі та присвоюючи відповідні теги. Система також включає систему голосування, репутації, політику контенту, ролі користувачів, панель модераторів, заходи фільтрації NSFW контенту, використання reCAPTCHA для запобігання спамерським атакам.

ІНФОРМАЦІЙНА СИСТЕМА, КОРИСТУВАЦЬКИЙ КОНТЕНТ, REACTJS, EXPRESSJS, NODEJS, MONGODB

ЗМІСТ

ВСТУП 5
1 ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ 6
1.1 Огляд веб-форумів і користувацького контенту 6
1.2 Основні стратегії управління користувацьким контентом 8
1.2.1 Політика контенту 9
1.2.2 Модерація 11
1.2.3 Система репутації користувачів 12
1.2.4 Сторонні системи управління користувацьким контентом 14
1.3 Аналіз аналогічних проєктів 15
1.4 Постановка задачі 18
2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ 20
2.1 Реалізація стека MERN 20
2.1.1 React 21
2.1.2 NodeJS 23
2.1.3 ExpressJS 24
2.1.4 MongoDB 25
2.2 Бібліотека NSFWJS 27
2.3 ReCAPTCHA 28
2.4 Проектування бази даних 30
3 ПРОГРАМНА РЕАЛІЗАЦІЯ 33
3.1 Розробка серверної частини 33
3.1.1 Ініціалізація 33
3.1.2 Установлення необхідних бібліотек 34
3.1.3 Запуск конфігу 35
3.1.4 Визначення архітектури 36
3.2 Структура клієнтської частини 37
3.3 Результат виконання роботи 38
ВИСНОВКИ 45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ 48

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки зростаюча кількість користувацького контенту на веб-форумах зумовила нагальну потребу в ефективних системах управління контентом.

Об’єкт дослідження. Процес управління користувацьким контентом.

Предмет дослідження. Методи та стратегії ефективного управління користувацьким контентом.

Гіпотеза. Впровадження комплексних стратегій управління користувацьким контентом і застосування ефективних методів модерації значно підвищить якість, релевантність і залученість користувацького контенту на веб-форумах, що призведе до підвищення рівня задоволеності користувачів і процвітання онлайн-спільноти.

Новизна. Описане в даній роботі програмне рішення дозволить досягати більшої ефективності в побудові інформаційної системи для приватних осіб або підприємств за рахунок реалізації розширених можливостей і функцій.

Апробація матеріалів роботи. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції студентів та молодих вчених «Інформатика, математика, автоматика» (ІМА – 2023).

Структура. Дана робота складається зі вступу, літературного огляду, методики вирішення поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

Зв'язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп’ютерних наук та пов’язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

1 ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Огляд веб-форумів і користувацького контенту

Веб-форуми, також відомі як дошки оголошень, – це онлайн-платформи, які сприяють спілкуванню та обміну інформацією між користувачами зі спільними інтересами. Вони служать віртуальним спільнотам, де люди мають можливість брати участь в дискусіях, шукати поради, відповіді на питання, ділитися власними знаннями, виражати думки тощо.

Існує велика кількість веб-форумів, що різняться між собою за своєю спрямованістю. Деякі з них можуть бути присвячені певним темам або нішам, таким як технології, ігри, кіно, здоров'я, музика і т.д. У загальному можна виділити такі основі типи форумів:

- Дискусійний: флагман форумів, що дозволяє створювати нові теми та вести поточні дискусії. Ініціювати розмову з відповідним контентом в ньому можна за допомогою редакційного календаря.
- Фідбек форум: надає клієнтам платформу для висловлення своїх думок, що гарно підходить для компаній, що хочуть скористатися цим фактом. Він є чудовим інструментом для досліджень і розробок, а також для інформування спільноти про власні ідеї чи оновлення програм.
- Форум статей: забезпечує централізоване розташування інформації, пропонує такі ресурси, як практичні статті, відео, оголошення, офіційні документи. Надійні просунуті функції пошуку дозволяють легко знаходити потрібну інформацію.
- Q&A (питання та відповіді): спеціалізується на обміні знаннями та вирішенні конкретних проблем. Користувачі можуть задавати питання на різні теми, а інші учасники відповідають на них [1].

Незважаючи на те, що соціальні мережі та месенджери стали більш популярними для спілкування, веб-форуми залишаються невід'ємною частиною

для онлайн-спільнот. Вони можуть бути потужним інструментом для бізнесу та організацій. Компанії можуть використовувати форуми для збору відгуків клієнтів, взаємодії аудиторії та формуванню лояльності до бренду. Окрім цього, онлайн-форуми можуть допомогти сформувати ключову частину комунікаційного плану працівників, оскільки електронна пошта знижує продуктивність [2].

Однією з ключових переваг веб-форумів є їхня здатність створювати відчуття спільноти та зв’язку між користувачами. Учасники можуть взаємодіяти з однодумцями, налагоджувати стосунки. Веб-форуми часто сприяють створенню сприятливого середовища для нетворкінгу та професійного розвитку. Користувачі можуть спілкуватися з іншими людьми у своїй галузі, встановлювати зв’язки з експертами, а також шукати поради чи наставництва. Ці зв’язки можуть призвести до кар’єрних можливостей та співпраці. Так, за результатами опитування, 75% роботодавців оцінюють командну роботу як “дуже важливу” [3].

Інтернет-форуми значною мірою покладаються на користувацький контент, тобто будь-який контент, створений і доданий учасниками форуму. Це можуть бути текстові повідомлення, аудіо, картинки, мультимедіа, приклади коду та багато іншого. Ця різноманітність робить форуми максимально привабливими для різних категорій користувачів, оскільки кожен може знайти різну форму виразу [4].

Активний користувацький контент не тільки робить форуми різноманітними, але також надає їм актуальності. Динамічний характер такого контенту стимулює зацікавленість користувачів, надає зростаючу цінність для своєї спільноти. На рисунку 1.1 яскраво зображено активну діяльність користувачів на одному із популярних форумів.

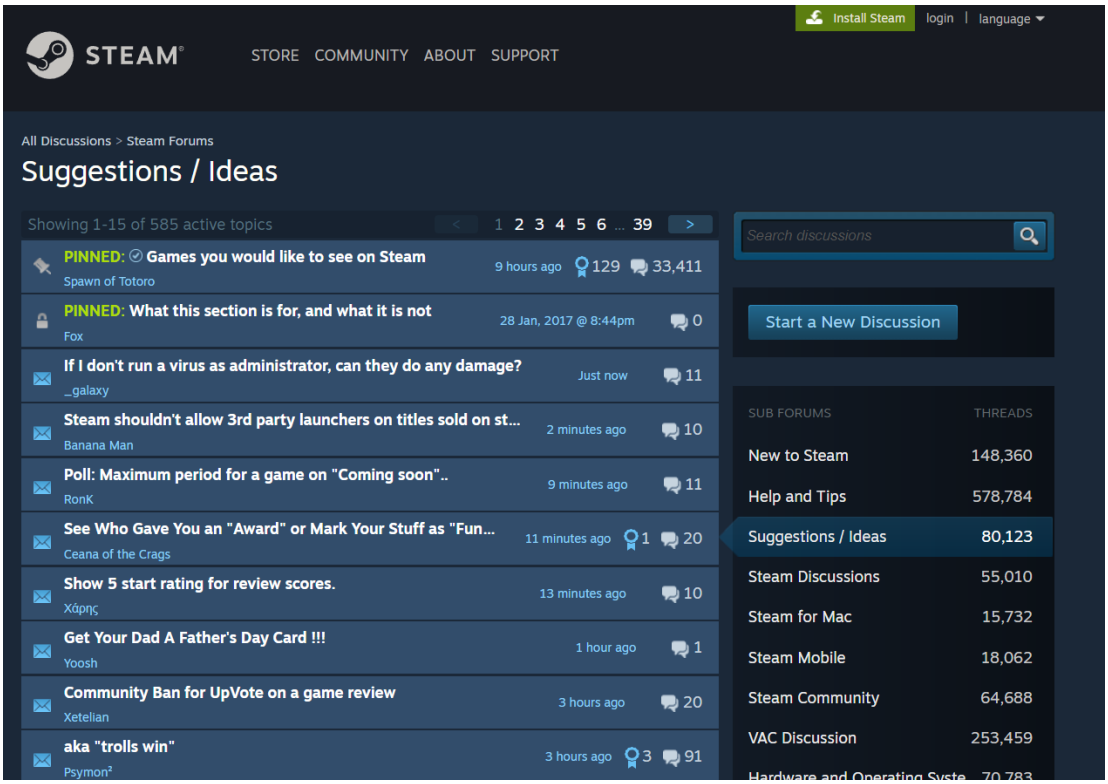


Рисунок 1.1 – Приклад активного форуму Steam Community

У загальному контексті онлайн-комунікацій та цифрової взаємодії користувацький контент посідає важливе місце. Незалежно від того, чи це дискусії на веб-форумах, дописи в соціальних мережах, статті в блогах, користувацький контент має значний вплив на те, як ми сприймаємо, взаємодіємо та приймаємо рішення. Більше того, понад 25% результатів пошуку для 20 найбільших брендів пов’язані з контентом, створений користувачами [4]. Все це підводить до висновку, що створення можливостей для користувацького контенту й уміння ним управляти, є критичними аспектами для побудови успішної онлайн-платформи.

1.2 Основні стратегії управління користувацьким контентом

В умовах експоненціального зростання користувацького контенту на різних платформах, підприємства або фізичні особи часто стискаються з проблемою управління такої інформації. Без ефективних стратегій управління, власниками онлайн-середовищ може бути складно просіювати величезну

кількість контенту, щоб знайти цінну інформацію, що не порушує встановлених правил. Компанія Google, яка є одним із головних лідерів у сфері інтернет-технологій, рекомендує дійсно непогані рішення щодо контролю якості контенту, створеного користувачами [5], а саме:

- Публікація політики щодо контенту
- Можливість повідомляти про порушення (реактивна модерація)
- Використання капчі при надісланні контенту для захисту від спам-атак
- Створення різних рівнів довірених користувачів
- Найм модераторів
- Створення автоматизованої системи фільтрації контенту
- Використання сторонніх інструментів для управління

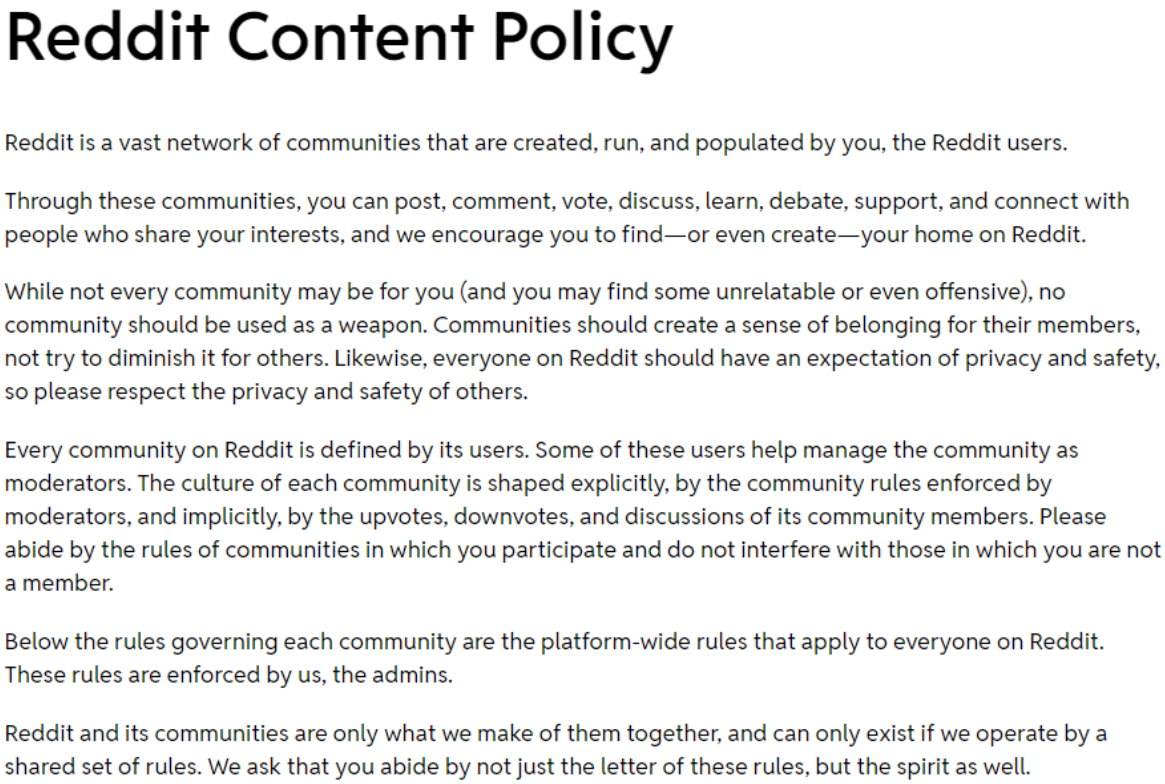
Нижче буде розглянуто детально ці стратегії для успішного керування контентом, створеним користувачем.

1.2.1 Політика контенту

Політика контенту відіграє незамінну роль у керуванні користувацьким контентом, вона є фундаментом платформи. Вона встановлює набір керівних принципів і правил, які визначають, що вважається прийнятним або неприйнятним. Чітко сформульована політика контенту встановлює очікування щодо поведінки користувачів, забезпечує виконання правових вимог і сприяє створенню безпечного середовища для всіх учасників платформи.

Для досягнення максимальної ефективності політики контенту, її рекомендується розміщувати на видимому місці на сайті, забезпечуючи легку доступність для користувачів. Помітні сповіщення або об'яви можуть служити засобом періодичного нагадування користувачам про правила.

У наведеному прикладі наочно представлено частину політики контенту з відомого форуму Reddit (рис. 1.2).



Reddit Content Policy

Reddit is a vast network of communities that are created, run, and populated by you, the Reddit users.

Through these communities, you can post, comment, vote, discuss, learn, debate, support, and connect with people who share your interests, and we encourage you to find—or even create—your home on Reddit.

While not every community may be for you (and you may find some unrelatable or even offensive), no community should be used as a weapon. Communities should create a sense of belonging for their members, not try to diminish it for others. Likewise, everyone on Reddit should have an expectation of privacy and safety, so please respect the privacy and safety of others.

Every community on Reddit is defined by its users. Some of these users help manage the community as moderators. The culture of each community is shaped explicitly, by the community rules enforced by moderators, and implicitly, by the upvotes, downvotes, and discussions of its community members. Please abide by the rules of communities in which you participate and do not interfere with those in which you are not a member.

Below the rules governing each community are the platform-wide rules that apply to everyone on Reddit. These rules are enforced by us, the admins.

Reddit and its communities are only what we make of them together, and can only exist if we operate by a shared set of rules. We ask that you abide by not just the letter of these rules, but the spirit as well.

Рисунок 1.2 – Приклад політики контенту на Reddit

Перейшовши за посиланням, можна більш детально познайомитись з документом. Зокрема він включає наступні положення про:

- Якість і релевантність: Reddit наголошує на важливості створення цінної інформації, яка є корисною для інших
- Заборонений контент: Reddit суворо забороняє пропаганду насильства, мову ворожнечі, переслідування та будь-який контент, що порушує чиїсь права
- Авторське право: Reddit очікує, що користувачі поважатимуть закони про авторське право і надсилатимуть автентичний контент
- Конфіденційність і захист даних: Reddit вимагає поважати приватність інших і утримуватися від поширення особистої інформації без згоди

Переглянутий приклад контент-політики надає основні принципи, які можна використовувати при створенні власної комплексної політики контенту.

1.2.2 Модерація

Модерація – це процес перегляду та моніторингу створеного користувачем контенту для забезпечення його відповідності принципам і політикам спільноти. Модератори виступають в ролі агентів платформи, які перевіряють і, при необхідності, видаляють контент, що порушує правила. Кожна платформа має свої власні підходи до модерації, які враховують її бренд, репутацію і цілі щодо привабливості нових користувачів [6]. Необхідно розуміти різницю між різними типами модерації для визначення слабких і сильних сторон кожної із них. На рисунку 1.3 показані найпоширеніші типи модерації створеного користувачами вмісту.

Рисунок 1.3 – Основні типи модерації контенту

Одним із найпоширеніших є пост-модерація, коли створений контент користувачем публікуються одразу, а потім переглядається модераторами. На противагу цьому, пре-модерація передбачає перевірку і схвалення перед публікацією.

Інший підхід – реактивна модерація, коли модератори реагують на скарги користувачів на конкретний контент. Це дозволяє швидко реагувати на порушення правил, але для їх виявлення необхідні відгуки користувачів.

Досить потужним інструментом являється автоматизована модерація, яка використовує технології штучного інтелекту (ШІ) та машинного навчання для аналізу та фільтрації користувацького контенту. Використовуючи алгоритми та розпізнавання образів, системи автоматизованої модерації можуть швидко обробляти великі обсяги контенту та виявляти потенційні порушення. Хоча штучний інтелект та машинне навчання може здатися ідеальним рішенням для масштабу і складності модерації контенту, існують етичні міркування, які слід взяти до уваги перед автоматизацією цього процесу. Укорінення упереджень за допомогою ШІ може призвести до несправедливих блокувань користувачів [7].

Альтернативним методом є розподілена модерація, коли завдання модерації покладаються на спільноту довірених користувачів, які дотримуються заздалегідь визначених правил. Такий підхід розподіляє навантаження серед спільноти, але вимагає ефективного управління спільнотою та чітких інструкцій.

1.2.3 Система репутації користувачів

Система репутації користувачів є популярним механізмом, що використовується онлайн-платформами для оцінки та аналізу надійності та поведінки спільноти. Деякі поширені способи використання цих систем можна знайти на веб-сайтах, таких як Stack Exchange, Yahoo! Answers, Quora [8].

Шляхом надання користувачам балів на основі їх дій і внесків, система надає об'єктивну оцінку надійності та взаємодії. Позитивна поведінка винагороджується підвищенням репутації, тоді як неправомірна дія призводить до її зниження. Накопичуючи бали репутації, користувачі можуть розблокувати певні привілеї або отримати доступ до додаткових функцій на платформі. Наприклад, користувачі з високою репутацією можуть отримати доступ до ексклюзивних форумів, брати участь у розширених дискусіях або отримати привілеї модератора, щоб допомогти забезпечити дотримання правил спільноти. З іншого боку, користувачі з низькою репутацією можуть зіткнутися з

обмеженнями, такими як обмежені можливості публікації, повільніший час схвалення дописів або неможливість брати участь у певних заходах.

Однією з ключових переваг системи репутації користувачів є її здатність стимулювати бажану поведінку. Люди завжди прагнуть отримати високу репутацію, що сприяє підвищенню соціального престижу. Це в свою чергу підвищує якість більшості публікацій, а всі користувачі стають більш мотивованими ділитися своїм досвідом.

Система репутації може мати систему почесних значків, на зразок: «Новачок», «Дослідник», «Ветеран» (рис. 1.4)

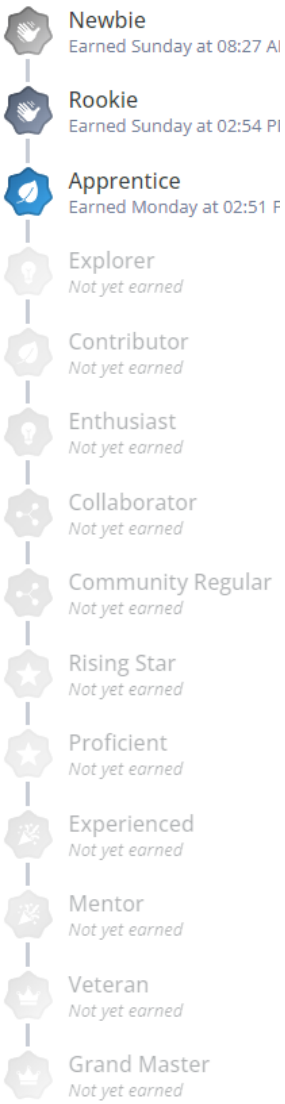


Рисунок 1.4 – Система почесних значків на одному із форумів

Ці почесні значки не лише відображають репутацію користувача, але й надають відчуття досягнення та прогресу, що може стимулювати їхню активність та залученість.

1.2.4 Сторонні системи управління користувацьким контентом

Сторонні системи управління користувацьким котентом – зовнішні плагіни або сервіси, які використовуються онлайн-платформами для забезпечення керування вмістом. Ці системи пропонують додаткові функції та можливості, що виходять за рамки нативних інструментів платформи, дозволяючи зробити процеси управління контентом більш ефективним та впорядкованим.

Однією з ключових переваг використання сторонніх систем управління контентом є можливість використовувати спеціалізований досвід і технології. Ці системи часто розробляються спеціалізованими компаніями або постачальниками послуг, які спеціалізуються на модерації та управлінні контентом. Як наслідок, вони пропонують розширені функції, такі як автоматична фільтрація контенту, аналіз настроїв і розпізнавання зображень, щоб ефективніше виявляти та позначати неприйнятний контент або такий, що порушує політику.

Крім того, сторонні системи управління контентом часто постачаються з потужними інструментами звітності та аналітики. Ці інструменти надають цінну інформацію про поведінку користувачів, тенденції контенту та моделі залучення, що дозволяє платформам приймати рішення щодо модерації контенту та управління спільнотами на основі даних. Вони дозволяють виявляти нові тенденції, моделі зловживань або сфери, де можуть знадобитися додаткові зусилля з модерації.

Однією із відомих систем управління контентом, створеним користувачами є WordPress (рис. 1.5).

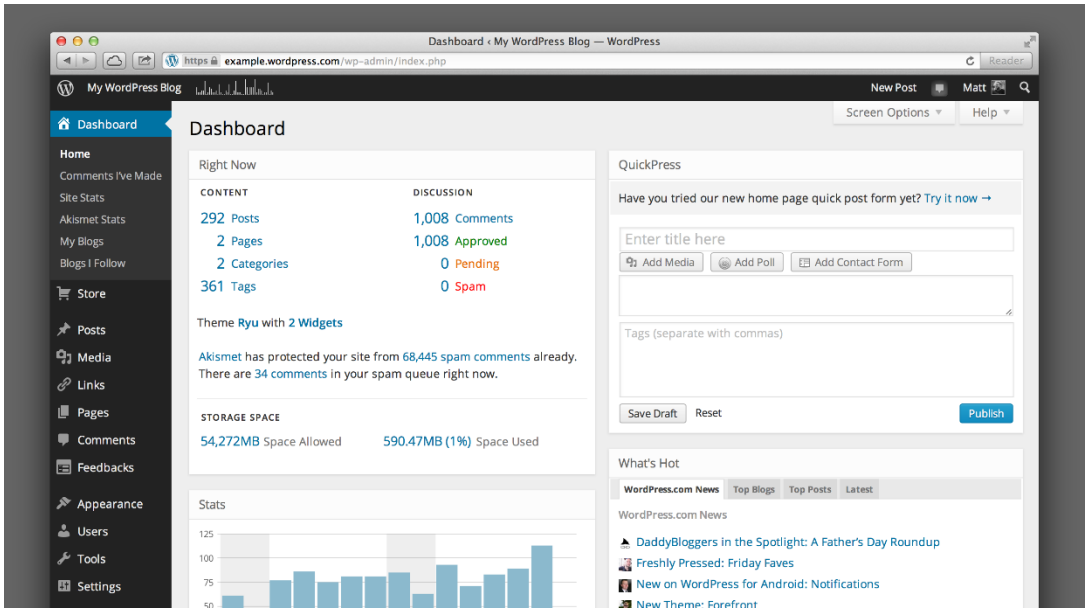


Рисунок 1.5 – Інтерфейс системи управління контентом WordPress

За допомогою WordPress власники сайтів можуть налаштовувати різні ролі та дозволи користувачів, що дозволяє їм контролювати рівень доступу та можливості для різних користувачів. Окрім того, WordPress пропонує безліч плагінів і розширень, спеціально розроблених для модерації контенту.

1.3 Аналіз аналогічних проєктів

Щоб отримати уявлення про управління користувацьким контентом, важливо проаналізувати існуючі платформи, які продемонстрували успіх у цій сфері. Два яскравих приклади - Stack Overflow та Reddit. Кожна платформа має свої унікальні особливості та підходи до управління користувацьким контентом.

Stack Overflow - це спеціалізована Q&A платформа, яка фокусується саме на темах, пов'язаних з програмуванням і розробкою програмного забезпечення. Основною метою Stack Overflow є створення професійного середовища для програмістів для обміну знаннями. Платформа підтримує високі стандарти якості та релевантності, суворо дотримуючись своєї політики щодо контенту. Модерація на Stack Overflow здійснюється за допомогою комбінації методів реактивної та пост-модерації.

Користувачі можуть задавати питання і публікувати відповіді, які потім переглядаються спільнотою. Інші користувачі можуть голосувати за питання і відповіді, оцінюючи їх якість і корисність (рис. 1.6).

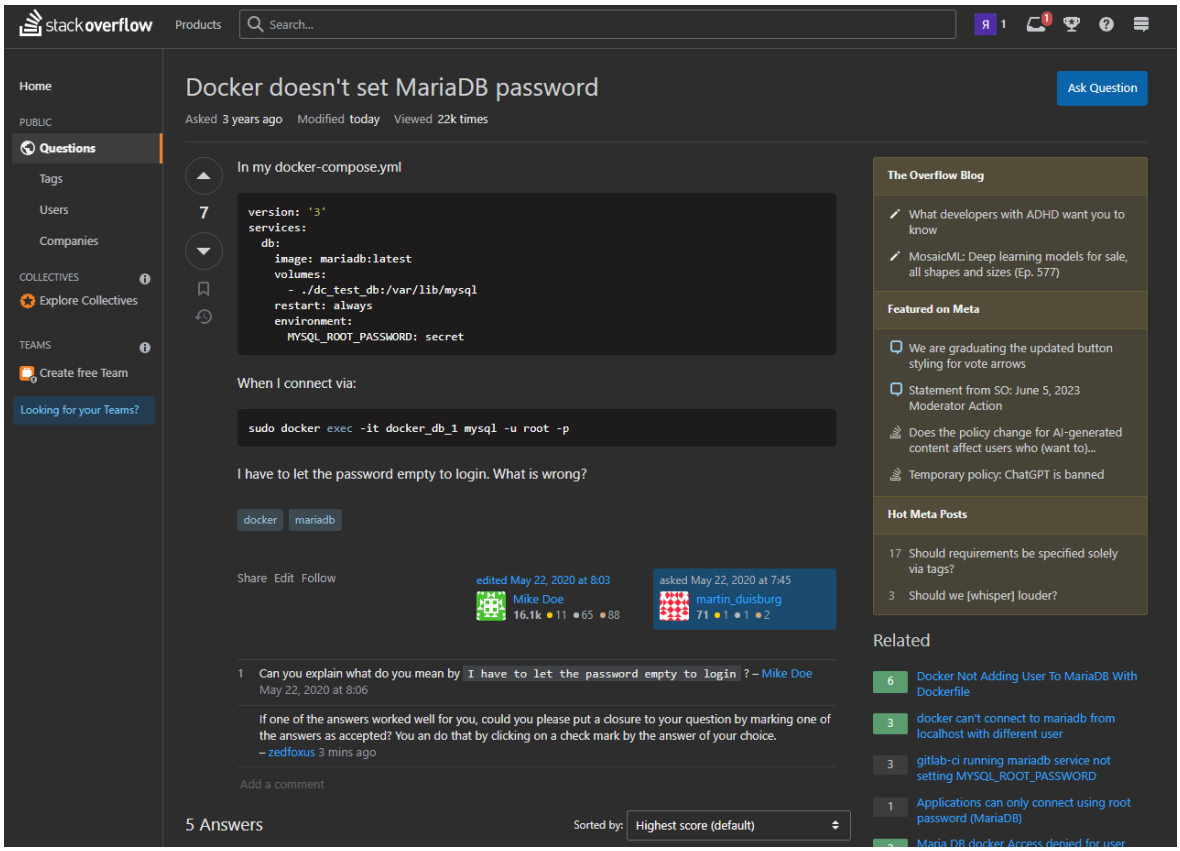


Рисунок 1.6 – Сторінка перегляду питання на Q&A форумі Stack Overflow

Крім того, на Stack Overflow реалізована система репутації. Користувачі заробляють бали репутації на основі своєї активності, наприклад, отримуючи схвальні відгуки на свої відповіді або ставлячи запитання, які добре сприймаються. Система репутації, реалізована Stack Overflow, відіграє важливу роль у запобіганні спаму та підтримці цілісності платформи. Оскільки користувачі накопичують бали репутації завдяки своїй активній та змістовній участі, спамерам стає складніше використовувати систему. Так, наприклад, для додавання коментаря необхідно досягти 50 очок репутації (рис. 1.7)

Рисунок 1.7 – Обмеження функціональності користувача з низькою репутацією

Reddit – це широка дискусійна онлайн-платформа, яка охоплює широкий спектр тем і спільнот, що відрізняє її від більш вузькоспеціалізованого Stack Overflow. На відміну від Stack Overflow, який зосереджується на темах, пов'язаних з програмуванням, сфера діяльності Reddit поширюється на різноманітні інтереси та дискусії (рис. 1.8).

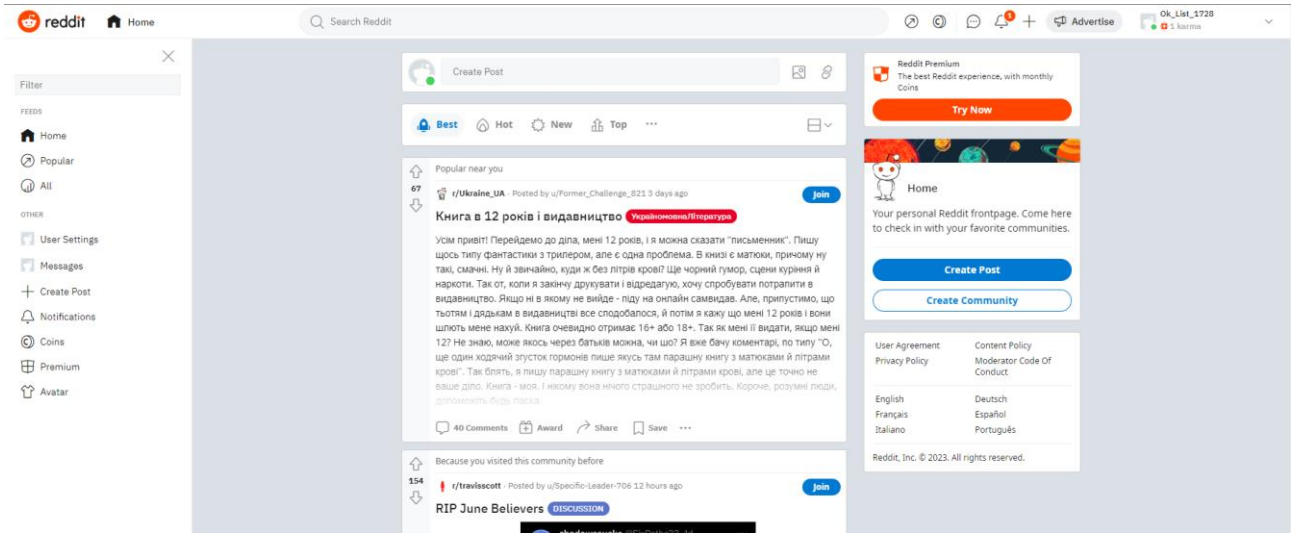


Рисунок 1.8 – Дискусійний форум Reddit

Важливим аспектом функціональності Reddit є система голосування, яка слугує основним механізмом впливу на видимість і рейтинг дописів та коментарів. За допомогою голосування користувачів спільнота колективно визначає популярність і релевантність контенту.

З точки зору модерації контенту, Reddit використовує переважно децентралізований та гібридний підхід. Кожен субреддіт, який представляє спільноту, присвячену певній темі, може мати власну команду модераторів, відповідальних за нагляд та управління контентом у відповідних спільнотах.

Однією з помітних відмінностей між Reddit та Stack Overflow є ширше прийняття різноманітності контенту на Reddit. На відміну від вужчого фокусу Stack Overflow, Reddit дозволяє розміщувати ширший спектр контенту, включаючи можливість розміщувати NSFW (небезпечний для роботи) контент у спеціальних субреддітах.

Важливо зазначити, що хоча і Stack Overflow, і Reddit мають механізми для управління користувацьким контентом, вони орієнтовані на різну аудиторію і слугують різним цілям. Stack Overflow робить акцент на професійному обміні знаннями в певній галузі, тоді як Reddit пропонує більш різноманітну і відкриту платформу для дискусій на різні теми та інтереси.

Ще одна платформа, про яку варто згадати, – **4chan**, яка використовує суттєво інший підхід до управління користувацьким контентом. 4chan працює з високим рівнем анонімності і практично немає модерації. Як наслідок, дотримання правил є непослідовним, що призводить до труднощів у підтримці стабільного рівня якості контенту. 4chan здобув погану славу через розміщення суперечливого й образливого контенту через свою нерегульованість.

Враховуючи специфіку поставленого завдання, можна зробити висновок, що платформа Stack Overflow є ідеальним прикладом для реалізації ефективного управління користувацьким контентом. Сфокусований підхід до тем, пов'язаних з програмуванням, система репутації та ретельний процес модерації забезпечують створення якісного та релевантного контенту. Використовуючи найкращі практики та стратегії Stack Overflow, подібні платформи можуть створити середовище, яке заохочує змістовний внесок користувачів, сприяє залученню спільноти та підтримує високі стандарти якості контенту.

1.4 Постановка задачі

Метою роботи є розробка інформаційної системи, яка ефективно управляє користувацьким контентом на веб-форумі, використовуючи сучасний стек технологій MERN (MongoDB, Express.js, React.js, Node.js) та існуючі бібліотеки.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Архітектура системи: розробка масштабованої та надійної системної архітектури, що включає стек технологій MERN. Це передбачає створення бази даних MongoDB для зберігання даних користувачів і контенту форуму, реалізацію внутрішнього сервера Express.js для обробки запитів API і створення фронтенду React.js для взаємодії з користувачем;
2) Управління користувацьким контентом: створення функціоналу для ефективного управління користувацьким контентом, що включатиме систему репутації та голосування, фільтрацію NSFW зображень та панель модератора;
3) Тестування та забезпечення якості: проведення тестування для виявлення будь-яких помилок або проблем у системі;

Проведення цих завдань дозволить розробити інформаційну систему, яка забезпечує ефективне управління користувацьким контентом на Q&A форумі.

2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Реалізація стека MERN

За останні роки ландшафт розробки веб-додатків значно змінився, представивши розробникам безліч варіантів вибору. Серед великої кількості наявних технологічних стеків та допоміжних інструментів, MERN стек вирізняється своєю ефективністю та гнучкістю у створенні повноцінних веб-систем.

MERN стек – це популярний інструмент, який спрощує процес створення універсальних додатків, використовуючи можливості Mongo, Express, React та NodeJS. Він значно скорочує час, необхідний для початкового налаштування, і дозволяє розробникам пришвидшити роботу за допомогою добре зарекомендованих технологій [9].

Значною перевагою MERN є те, що він базується на JavaScript, що дозволяє працювати лише з одною мовою програмування, спрощуючи процес розробки. Ілюстрацію архітектури MERN наведено на рисунку 2.1.

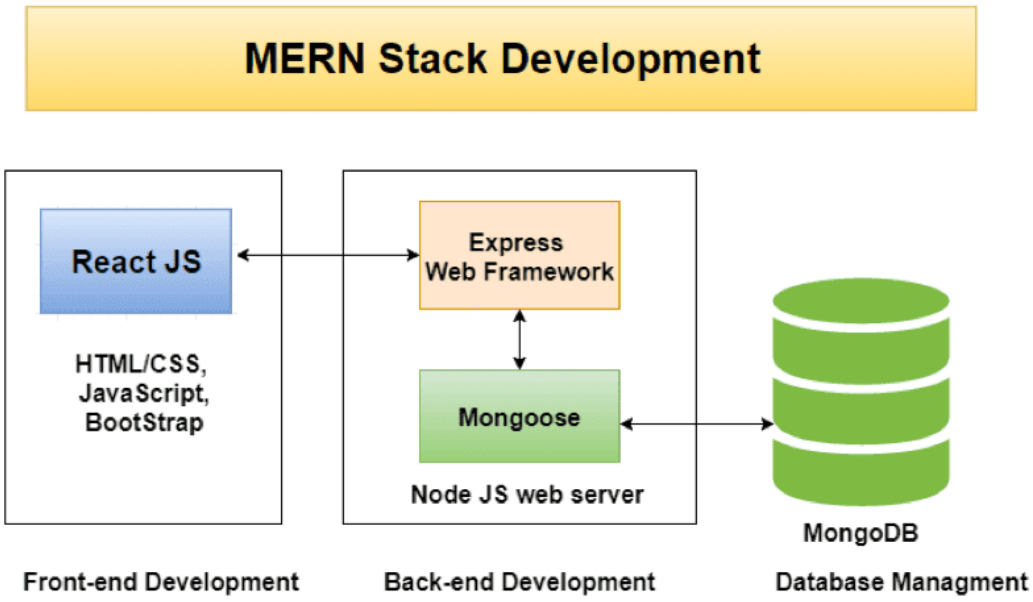


Рисунок 2.1 – Архітектура стека MERN [10]

Нижче будуть розглянуті технології, що входять до цього стеку і які будуть використані в проєкті.

2.1.1 React

Клієнтська частина проєкту буде розроблена з використанням React, популярної бібліотеки JavaScript для побудови користувацьких інтерфейсів. Підтримувана такими відомими платформами, як Facebook та Instagram [11], React пропонує архітектуру на основі компонентів, яка покращує зручність обслуговування коду та сприяє модульній розробці. Використовуючи React, можна створювати надійні додатки, здатні завантажувати нові дані без необхідності оновлювати всю сторінку. Такий підхід значно покращує швидкість роботи додатку та чуйність, забезпечуючи користувачам безперебійну роботу з ним. React слугує компонентом “View” в архітектурі Model-View-Controller (MVC) [12].

У React деревоподібна структура компонентів є фундаментальною концепцією, яка допомагає організувати та керувати користувацьким інтерфейсом додатку. Деревоподібна структура – це ієрархічне розташування компонентів, де кожен компонент може мати дочірні компоненти, вкладені в нього (рис. 2.2).

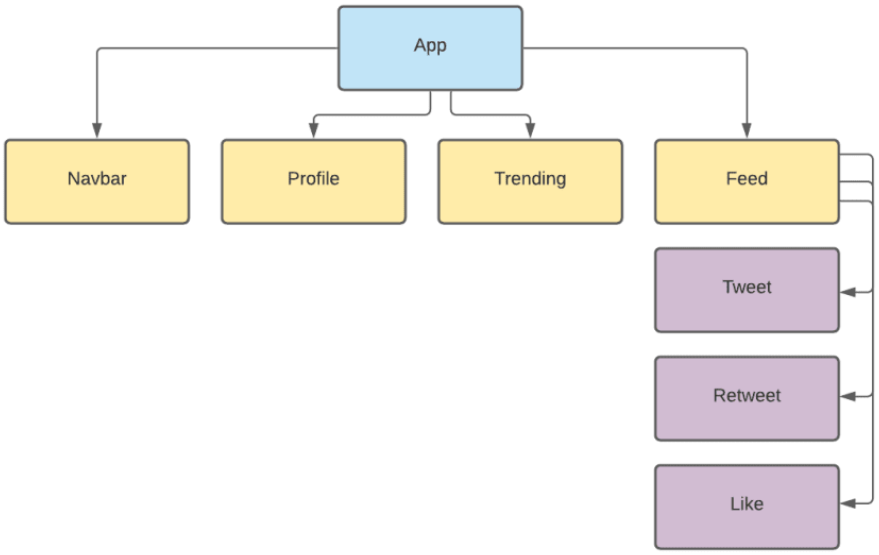


Рисунок 2.2 – Деревоподібна діаграма React-компонентів [13]

На вершині дерева знаходиться кореневий компонент, який слугує точкою входу для всього додатку. Від кореневого компонента дерево розгалужується по мірі додавання дочірніх компонентів. Кожен компонент представляє певну частину користувацького інтерфейсу і може складатися з інших менших компонентів.

Варто зазначити, що React використовує концепцію Virtual DOM (або Virtual Document Object Model) для оптимізації оновлення та відтворення компонентів у веб-додатках. Віртуальний DOM служить легким представленням фактичного DOM, що дозволяє React ідентифікувати та застосовувати лише необхідні зміни. Зводячи до мінімуму непотрібні оновлення фактичного DOM, React значно покращує продуктивність і ефективність візуалізації компонентів.

Крім того, React представляє JSX, синтаксичне розширення, яке дозволяє писати HTML-подібний код безпосередньо в JavaScript.

Разом з React буде використано Redux – менеджер станів. Redux – це бібліотека управління станами. Вона надає передбачуваний контейнер станів, який допомагає керувати даними додатку та забезпечує ефективну комунікацію між компонентами. З Redux стан додатку зберігається в єдиному незмінному об'єкті, який називається "state" (сховище). Він відрізняється тим, що використовує єдине сховище замість декількох сховищ, які часто зустрічаються в інших бібліотеках, натхненні Flux. Це сховище часто називають “єдиним джерелом істини”, оскільки воно слугує єдиною точкою доступу до стану програми [14].

Redux використовує редуктори, дії та підсилювачі (reducers, actions, middleware) як важливі елементи для управління станом. Редуктори – чисті функції, відповідальні за визначення того, як стан програми має оновлюватися у відповідь на надіслані дії. Дії – це корисні дані, які надсилаються для ініціювання змін стану в сховищі Redux. Підсилювачі діють як міст між диспетчеризацією дій і досягненням редукторів, надаючи додаткові функції, такі як обробка

асинхронних дій або перехоплення та модифікація дій до того, як вони досягнуть редукторів.

На рисунку 2.3 можна побачити архітектуру технології Redux.

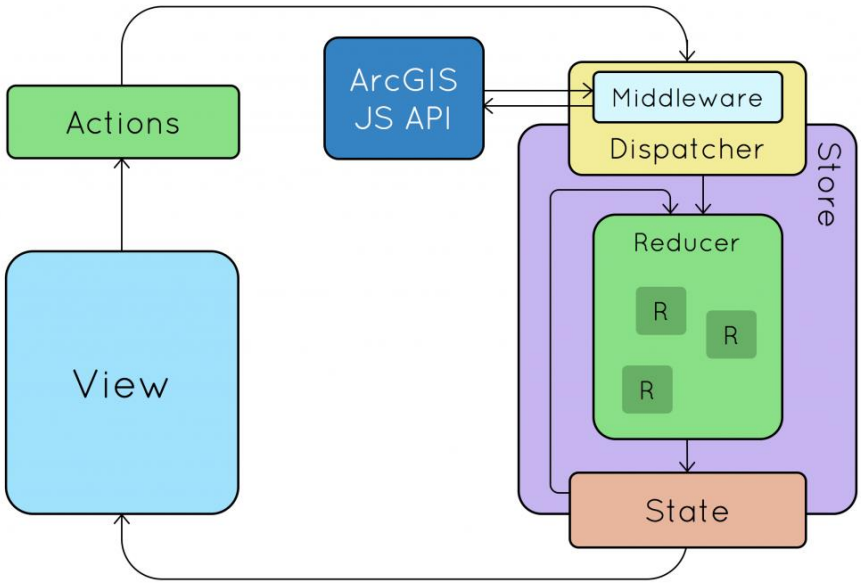


Рисунок 2.3 – Архітектура технології Redux [15]

На рисунку також можна побачити диспатчер, що є частиною сховища, який отримує дії та надсилає їх редукторам для оновлення стану. Він служить центральним центром для координації потоку дій і забезпечення їх належної обробки.

Таким чином, React і Redux дадуть змогу створити надійний та інтерактивний користувацький інтерфейс, забезпечуючи безперебійну роботу користувача. Використовуючи ці технології, буде розроблено динамічний, масштабований та зручний у використанні веб-додаток.

2.1.2 NodeJS

Для реалізації серверної частини буде використано NodeJS, середовище виконання, яке дозволяє створювати серверні додатки за допомогою JavaScript. Воно базується на движку V8 від Google Chrome [16].

Унікальною особливістю NodeJS є можливість використовувати одну і ту ж мову як і для фронтенд-, так і для бекенд-розробки. Саме тому, в рамках даної роботи немає необхідності переключатись на іншу мову програмування, оскільки JavaScript є єдиною мовою програмування, що використовується для розробки.

Ще однією особливістю NodeJS є ефективне використання пам’яті в порівнянні з іншими фреймворками [14]. Окрім цього, він добре сумісний с JSON, широко використовуваним форматом серіалізації для обміну даними. Ця вбудована підтримка JSON є дуже корисною при взаємодії з NoSQL базой даних MongoDB.

NodeJS використовує свою асинхронну природу для оптимізації продуктивності. На відміну від традиційних серверних моделей, які покладаються на кілька потоків, NodeJS працює в одному потоці. Такий підхід гарантує, що майже кожна операція в Node.js є асинхронною і використовує функції вищого порядку як слухачі подій [14]. Це забезпечує одночасну обробку декількох подій і покращує загальну масштабованість.

Ще NodeJS має потужний менеджер пакетів npm (Node Package Manager), який надає доступ до величезного сховища пакетів та бібліотек з відкритим кодом, що дозволить легко інтегрувати існуючі рішення для розроблюваного додатку.

2.1.3 ExpressJS

ExpressJS - це широко використовуваний фреймворк в NodeJS, який значно спрощує розробку RESTful API та веб-додатків. Він надає надійний набір можливостей і функцій, які спрощують виконання типових завдань, пов'язаних зі створенням серверних додатків. Express набув значної популярності в галузі і був прийнятий такими відомими компаніями, як Accenture, IBM та Uber. Таке широке використання свідчить про його надійність та ефективність у реальних виробничих умовах [17].

Коли сервер обробляє вхідний запит, він повинен направити цей запит до потрібної частини програми. Саме тут Express демонструє себе як ефективний фреймворк. Він надає можливості маршрутизації, які дозволяють визначити та керувати маршрутами. Можна легко створювати маршрути для обробки конкретних HTTP-запитів, таких як GET, POST, PUT і DELETE, і зіставляти їх з відповідними функціями контролера.

Крім того, ExpressJS пропонує вбудовану підтримку для роботи зі статичними файлами, такими як HTML, CSS та клієнтський JavaScript. Це дозволяє обслуговувати статичні ресурси безпосередньо з сервера, усуваючи потребу в окремих файлових серверах.

Вбудована підтримка проміжного програмного забезпечення ExpressJS є ще однією важливою особливістю. Функції проміжного програмного забезпечення можна використовувати для проміжної обробки запитів, таких як аутентифікація, перевірка даних, ведення журналів тощо. Вони можуть бути вибірково застосовані до певних маршрутів або глобально до всіх маршрутів, забезпечуючи гнучкість і контроль над процесом обробки запитів.

Використовуючи можливості ExpressJS, буде прискорено розробку бекенда, забезпечуючи ефективне опрацювання запитів, гнучку маршрутизацію і безшовну інтеграцію з іншими компонентами застосунку.

2.1.4 MongoDB

Для внутрішньої частини проекту буде використано MongoDB, дуже популярну базу даних NoSQL, щоб ефективно зберігати та керувати даними. MongoDB відноситься до категорії документних баз даних, де дані зберігаються в гнучких документах з використанням формату BSON, що розшифровується як двійковий JSON (рис. 2.4).

Рисунок 2.4 – Документи колекції в базі даних Mongo

На відміну від традиційних реляційних баз даних, які зберігають дані в структурованих рядках і стовпцях, MongoDB зберігає дані у вигляді пар ключ-значення в документах. Такий документно-орієнтований підхід забезпечує більшу гнучкість і дозволяє зберігати дані різної структури в одній колекції.

Формат BSON в MongoDB, двійкове представлення JSON, забезпечує ефективне зберігання та пошук даних. BSON підтримує різні типи даних і дозволяє вкладати документи і масиви, полегшуючи зберігання складних і динамічних структур даних.

Використовуючи модель бази даних документів MongoDB, буде отримано перевагу безсхемного дизайну, що означає, що можна адаптувати та змінювати структуру документів за потребою, без необхідності в суворих наперед визначених схемах. Така гнучкість особливо корисна при роботі з мінливими вимогами до даних і дозволяє безперешкодно інтегруватися з бекендом додатку.

Використання MongoDB в якості рішення для баз даних дозволить ефективно обробляти різноманітні та неструктуровані дані, забезпечуючи масштабованість та продуктивність для внутрішніх операцій додатку.

2.2 Бібліотека NSFWJS

Одним із важливих аспектів розробки веб-додатків є забезпечення того, щоб вміст, яким ділиться користувач і який відображається, був доречним і відповідав настановам спільноти. Для вирішення цієї проблеми використовується бібліотека NSFWJS. NSFWJS – це зручна бібліотека, яка дозволяє розробникам виявляти та фільтрувати потенційно відвертий або небезпечний вміст у зображеннях, не вимагаючи попередніх знань про машинне навчання.

NSFWJS використовує алгоритми машинного навчання для аналізу та класифікації зображень на основі їхнього вмісту. Вона може ідентифікувати різні типи небезпечного контенту, включно з оголеною натурою, контентом для дорослих, насильством та іншими відвертими матеріалами. Використовуючи можливості моделей глибокого навчання, NSFWJS забезпечує ефективний засіб фільтрації контенту.

Коли ми говоримо про модель машинного навчання, по суті, мається на увазі файл, який пройшов тривале навчання, щоб ідентифікувати та класифікувати різні типи контенту. У нашому випадку моделі були спеціально навчені класифікувати зображення за п'ятьма різними категоріями. Коли ми пропускаємо зображення через модель, вона надає нам JSON-файл, що містить передбачення, які вказують на ймовірність належності зображення до кожної з цих категорій.

П'ять класів, які визначають модель NSFW, є наступними:

- *Drawing* – картини мистецтва
- *Hentai* – порнографічне мистецтво, непридатне для більшості неробочих середовищ
- *Neutral* – нейтральна категорія включає загальний і необразливий вміст
- *Porn* – порнографічний контент
- *Sexy* – непристойний провокаційний контент

Кожному класу в моделі NSFW присвоюється значення від 0 до 1. У контексті розробки Q&A форуму для програмування, значення `hentai > 0.3%` та `porn > 0.3%` вважаються абсолютно неприйнятними, тому планується автоматично відхиляти такі зображення (рис. 2.5).

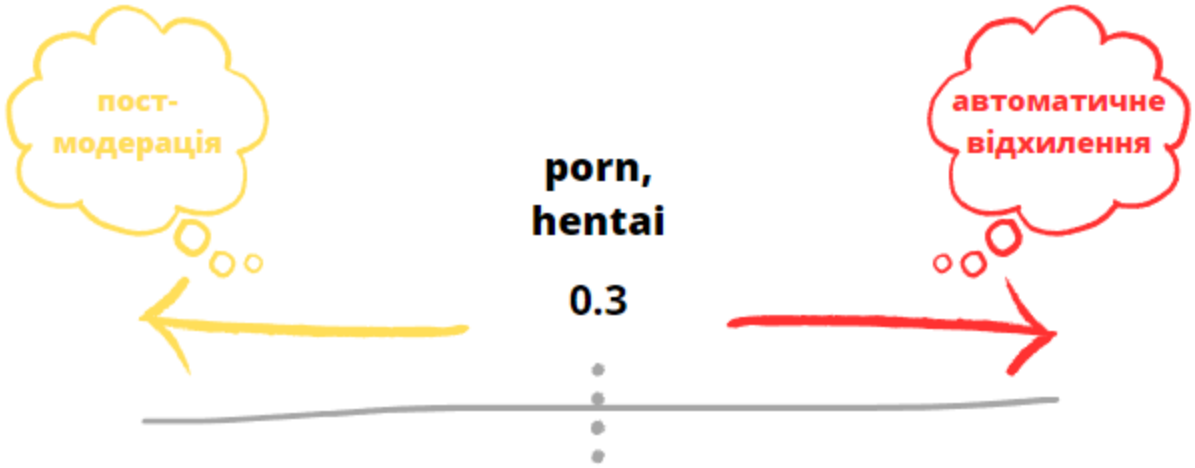


Рисунок 2.5 – Схема автоматичного визначення неприйнятного вмісту

Важливо зазначити, що хоча інтеграція NSFWJS забезпечує цінний рівень захисту від неприйнятного контенту, система не є досконалою і може мати певні обмеження. Моделі машинного навчання покладаються на шаблони та навчальні дані, і в процесі класифікації завжди існує ймовірність хибнопозитивних або хибнонегативних результатів.

2.3 ReCAPTCHA

З поширенням автоматизованих ботів і зловмисних дій в інтернеті потреба в надійних методах перевірки людської присутності стала першочерговою. Щоб підвищити безпеку та запобігти спаму на платформі, вирішено інтегрувати технологію ReCAPTCHA в систему. ReCAPTCHA розшифровується як “повністю автоматизований публічний тест Тьюринга для розрізнення комп’ютерів і людей”, використовується в різних програмах для розрізнення введених даних людиною та комп’ютером [18].

ReCAPTCHA є продуктом компанії Google, широко використовується такими компаніями, як Facebook, Twitter, Steam [19]. На сьогоднішній день випущено три версії reCAPTCHA. Підтримка першої закінчилась ще в 2018 році.

Друга версія reCAPTCHA, відома як reCAPTCHA v2, пропонує користувачам тести на розпізнавання зображень. Користувачі повинні ідентифікувати конкретні об'єкти на зображеннях, навіть якщо вони спотворені або представлені в різних контекстах. Залучаючи мільйони користувачів до цього процесу, reCAPTCHA збирає цінні дані, які сприяють навчанню моделей машинного навчання для анотування зображень та інших програм штучного інтелекту.

На відміну від попередньої версії, reCAPTCHA v3 використовує вдосконалений алгоритм аналізу ризиків для оцінки поведінки користувачів на веб-сайтах. Цей алгоритм враховує час, витрачений на заповнення форми, що дозволяє комплексно оцінити активність людини та бота.

Для відчутного доказу людської взаємодії на проєкті вирішено впровадити другу версію. Щоб інтегрувати reCAPTCHA v2 у веб-форум, потрібні ключі API для встановлення з’єднання з проектом reCAPTCHA. Отримання цих ключів є простим процесом, що включає наступні кроки:

4) Відвідати сайт reCAPTCHA (https://www.google.com/recaptcha) і перейти до розділу “Консоль адміністратора”
5) Увійти в обліковий запис Google або створити новий
6) Зареєструвати веб-форум, надавши інформацію про доменне ім’я та мітку ідентифікації проєкту
7) Обрати версію reCAPTCHA v2 та відправити форму

На рисунку 2.6 можна побачити приклад заповненої форми.

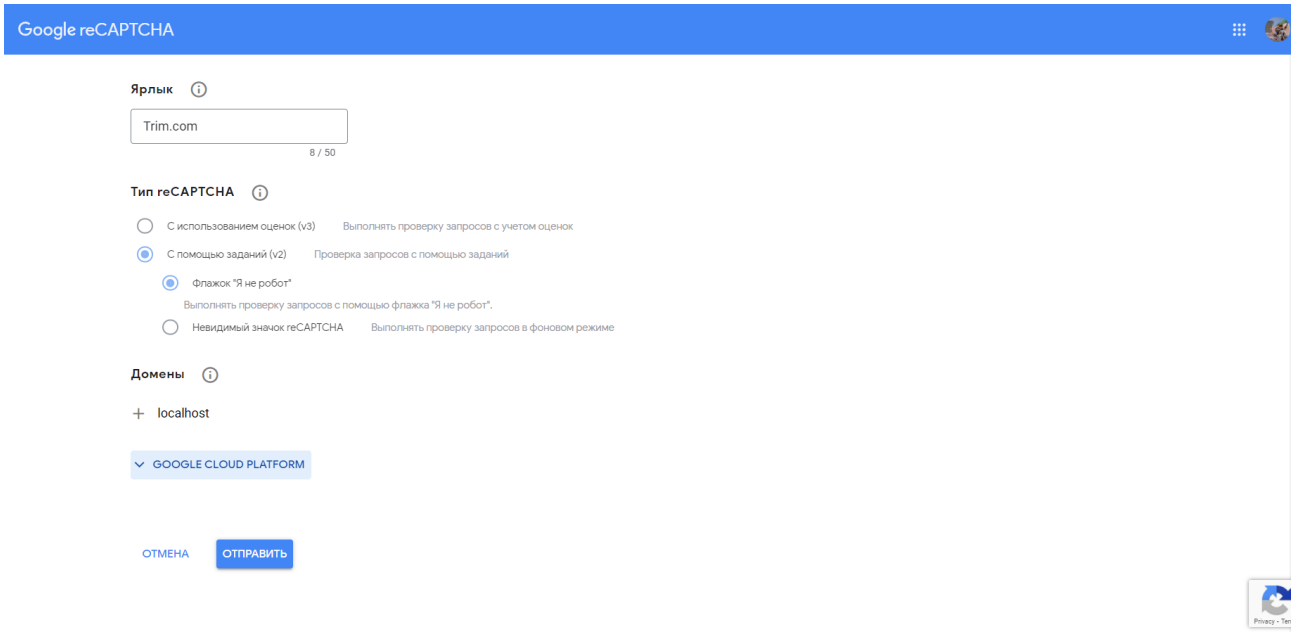


Рисунок 2.6 – Приклад заповненої форми на отримання API ключів для капчі

Після успішної реєстрації проєкту, сайт надасть два ключі: “Site key” та “Secret key”.

"Site key" - це публічний ключ, який необхідно інтегрувати в сторінки веб-форуму.

"Secret key" - це приватний ключ, який слід зберігати в захищеному місці на сервері веб-форуму. Він використовується для перевірки відповідей reCAPTCHA на серверній стороні та захисту від потенційних зловмисних дій.

Інтеграція reCAPTCHA в веб-форум дійсно забезпечить надійний захист від автоматизованих спам-ботів та зловживань. Це дозволить ефективно фільтрувати небажану активність та забезпечити безпеку інтеракції з користувачами. Однак, варто враховувати, що деякі користувачі можуть відчувати деякий дискомфорт або не зрозуміти процес проходження тестів reCAPTCHA. Це може відбутися через неясні інструкції, вимогу вибрати відповіді з неякісних або нечітких зображень.

2.4 Проєктування бази даних

Для полегшення процесу проектування бази даних використаємо діаграму зв’язків-сутностей (ERD). ERD – це графічне представлення, яке візуально

зображує сутності, зв’язки та атрибути в системі [20], надаючи чіткий і організований огляд структури бази даних.

Сутності – це об’єкт в БД, що представляють інтерес в системі, яка моделюється. Прикладом сутностей можуть бути клієнти, продукти, працівники.

Зв’язки описують асоціацію між сутностями. Зв'язки можуть бути один-до-одного, один-до-багатьох або багато-до-багатьох. Наприклад, клієнт може мати кілька замовлень, встановлюючи зв'язок один-до-багатьох між сутністю клієнта та сутністю замовлення.

Атрибути – це характеристики або властивості сутності. Вони надають додаткову інформацію про сутності. Атрибути можуть описувати різні аспекти, такі як ім'я, адреса, дата народження або кількість товару. Вони допомагають надати детальну інформацію про об'єкти і дозволяють зберігати і знаходити конкретні дані в базі даних.

Зобразимо відносини і їх зв’язки на ER-діаграмі (рис. 2.7).

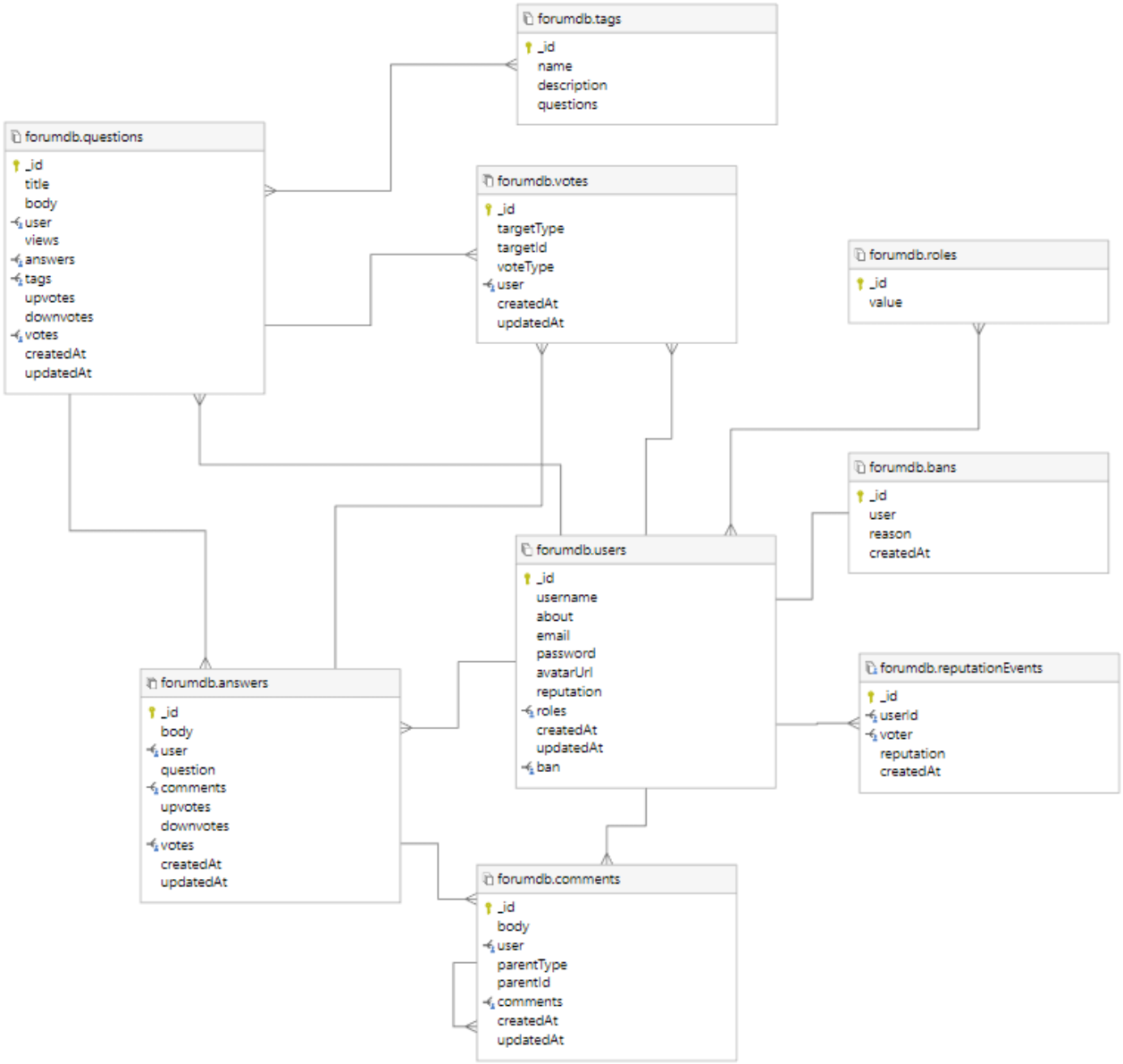


Рисунок 2.7– ER-діаграма веб-додатку

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка серверної частини

Розробка серверної частини веб-форуму включала низку кроків для забезпечення надійного та функціонального бекенду. У цьому розділі будуть розглянуті різні аспекти реалізації серверної частини.

3.1.1 Ініціалізація

Для створення додатку використовувався програмний засіб Visual Studio Code (рис. 3.1).

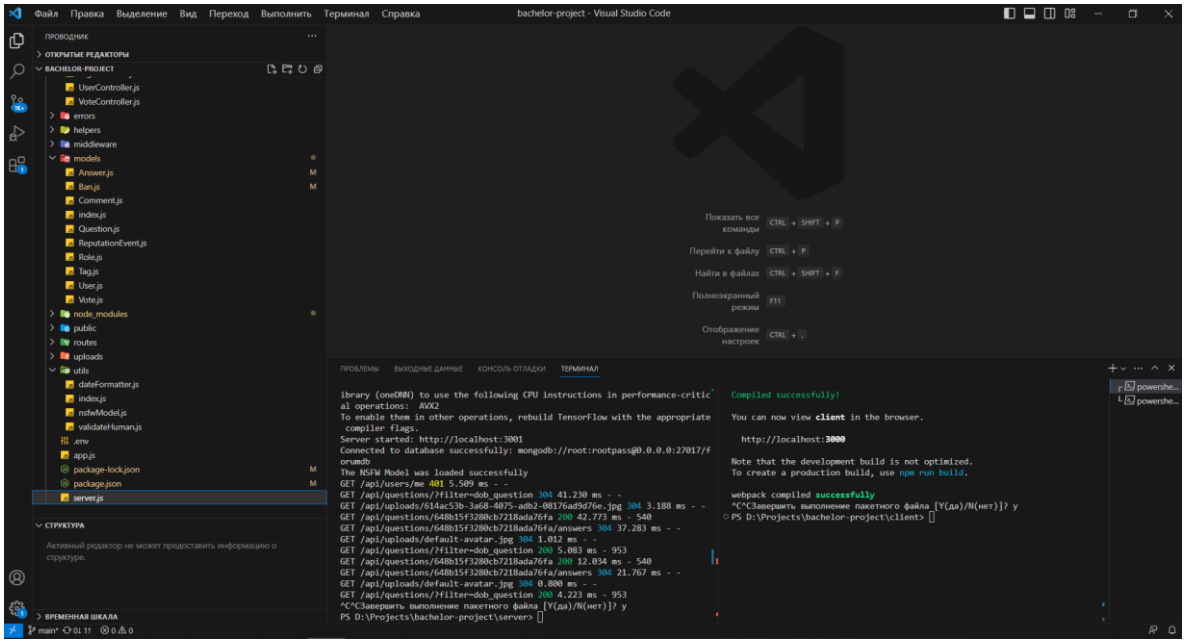


Рисунок 3.1– Вікно програми Visual Studio

Щоб ініціалізувати серверну частину була виконана команда `npm init`. Ця команда згенерувала файл package.json, що являє собою файл-маніфест, який містить метадані та список залежностей, необхідних для функціонування (рис. 3.2)

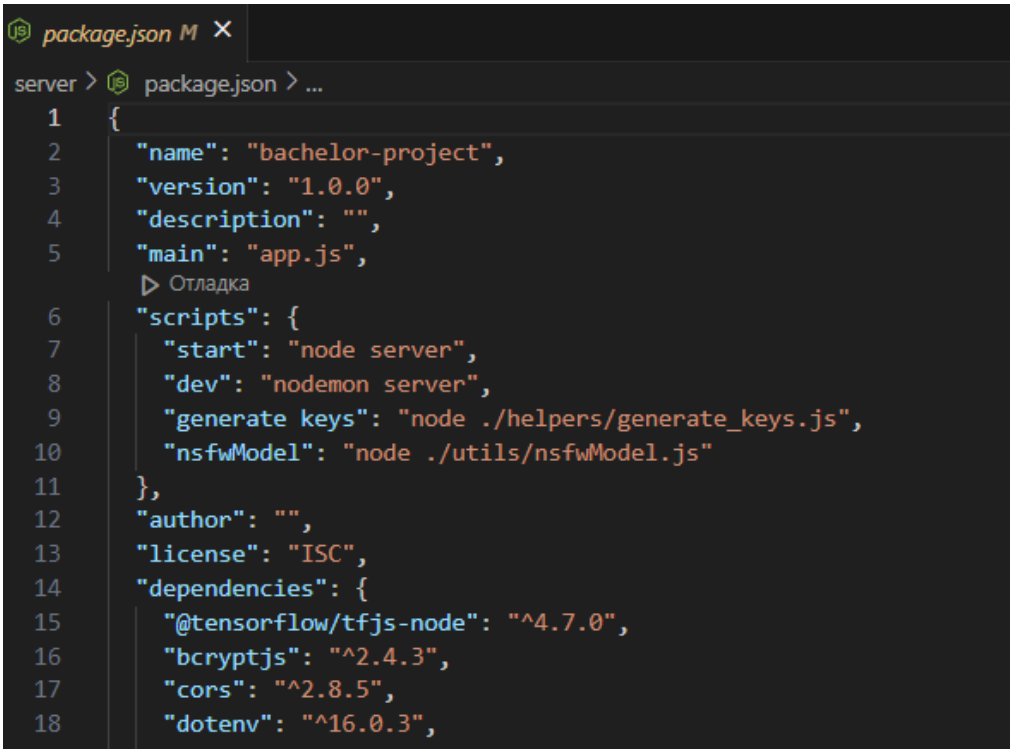


Рисунок 3.2– Сніпет маніфест-файлу package.json

3.1.2 Установлення необхідних бібліотек

Після завершення ініціалізації NodeJS, наступним кроком було встановлення необхідних бібліотек та модулів. Для їх підключення було використано команду `npm install <назва бібіліотеки>`. У таблиці 3.1 наведено основні бібліотеки, які були використані для реалізації проєкту.

Таблиця 3.1 – Використані бібліотеки і короткий опис

Назва бібліотеки	**Короткий опис**
tfjs-node	Бібліотека для запуску моделей TensorFlow в Node.js
bcryptjs	Бібліотека для безпечного хешування та перевірки паролів
cors	Проміжне програмне забезпечення для сервера, що дозволило йому відповідати на запити з різних джерел
dotenv	Модуль для завантаження змінних оточення з файлу
express	Фреймворк Node.js для спрощення розробки на стороні сервера
jsonwebtoken	Бібліотека для генерації та перевірки JSON-веб-токенів
mongoose	Інструмент для підключення до MongoDB

multer	Проміжне ПЗ для завантаження файлів
nsfwjs	Бібліотека для фільтрації NSFW-контенту
passport	Проміжний модуль для автентифікації

3.1.3 Запуск конфігу

Перш ніж приступити до розробки нашого серверного додатку, необхідно було створити надійну та гнучку систему конфігурації. Це дозволило б легко адаптувати додаток до різних середовищ і налаштовувати його поведінку відповідно до конкретних вимог. Для цього ми використали конфігураційний файл, який містив різні налаштування та параметри.

Щоб полегшити управління конфіденційними даними, такими як ключі API, облікові дані бази даних та інші параметри конфігурації, була використана бібліотека dotenv. Цей інструмент дозволив завантажувати змінні оточення з файлу .env у середовище виконання програми. Відокремивши конфіденційну інформацію від кодової бази, була підвищена безпека додатку. У файлі config.js було визначено набори властивостей, зокрема налаштування хосту та порту сервера, рядок підключення до БД та ін. (рис. 3.3).

index.js

server > config > index.js > <unknown> > "mongodb"

```
require('dotenv').config()

module.exports = {
    log_level: process.env.LOG_LEVEL || 'dev',
    "nodejs": {
        host: process.env.APP_HOST || 'localhost',
        port: process.env.APP_PORT || 3001,
    },
    "mongodb": {
        host: process.env.DB_HOST || 'localhost',
        port: process.env.DB_PORT || 27017,
        dbName: process.env.DB_NAME || 'forumdb',
        username: process.env.DB_USERNAME || 'root',
        password: process.env.DB_PASSWORD || 'rootpass'
    },
    "jwt": {
        secret: process.env.JWT_SECRET || 'd5493ebc04db681fe34c67a9af018a8d08ef53775490f8a7f72bcb1687bd867f',
        expiresIn: process.env.JWT_EXPIRES_IN || '1h'
    }
}
```

Рисунок 3.3– Конфігураційний файл

3.1.4 Визначення архітектури

Для того, щоб забезпечити добре структурований і підтримуваний серверний додаток, важливо визначити чітку архітектуру. Архітектура надає високорівневий огляд того, як різні компоненти та модулі взаємодіють один з одним, сприяючи розділенню проблем та масштабованості (таблиця 3.2, рис. 3.4).

Таблиця 3.2 – Структура серверної частини

Назва папки	**Призначення**
config	Конфігураційні файли, які містять налаштування додатку та зовнішних сервісів
controllers	Контролери, що містять логіку обробки запитів та взаємодії з моделями
errors	Файли для обробки та кастомізації помилок
helpers	Допоміжні функції для використання в додатку, включає підключення до БД
middleware	Middleware-функції для обробки запитів перед передачею до контролерів
models	Моделі, що відображають структуру та логіку бази даних
routes	Роути, які визначають маршрутизацію запитів та виклик контролерів
uploads	Завантажені файли, які можуть бути доступні з сервера
utils	Утиліти, які використовуються в додатку

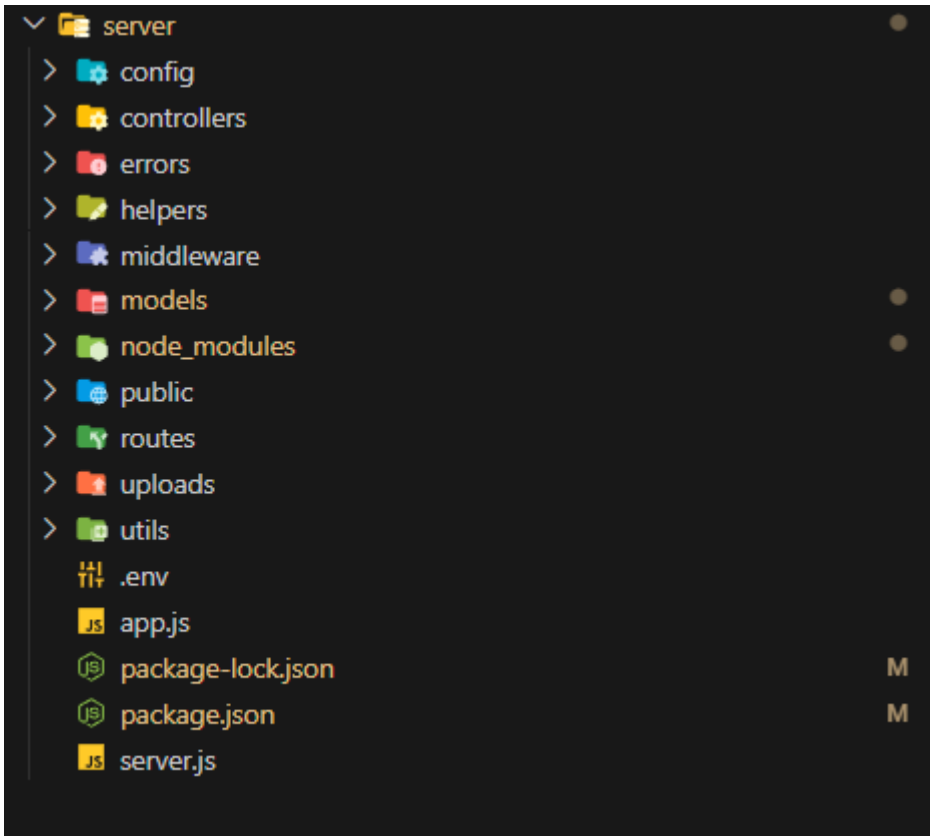


Рисунок 3.4 – Архітектура серверної частини

3.2 Структура клієнтської частини

Структура клієнтської частини включає набір папок та файлів, які організовані для ефективної розробки та управління фронтенд-аспектами проекту. В таблиці 3.3 і на рисунку 3.5 наведена файлова структура клієнтської частини.

Таблиця 3.3 – Структура клієнтської частини

Назва папки	Призначення
src	Основна папка, в якій розміщуються всі файли із вихідним кодом клієнтської частини
components	Компоненти React, які використовуються у проекті. Кожен компонент має свою окрему папку з файлами розширення jsx та scss
pages	Окремі сторінки застосунку, які включають компоненти та маршрутизацію
redux	Управлінням станом додатку за допомогою бібліотеки Redux

utils	Утиліти, які використовуються в клієнтській частині проєкт
public	Статичні файли, які доступні для клієнта без обробки сервером

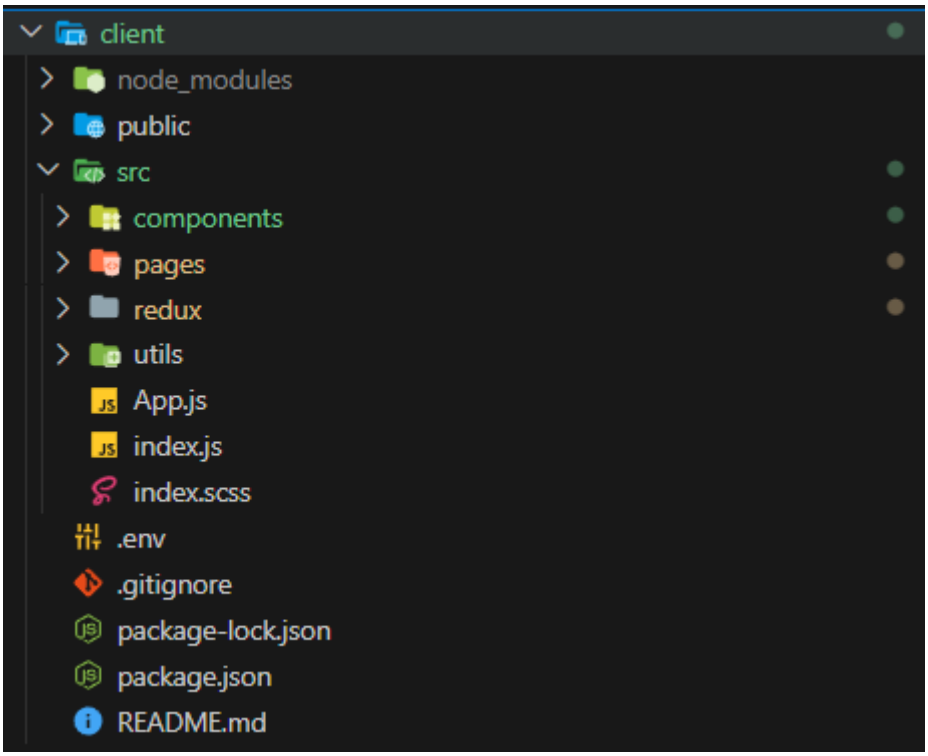


Рисунок 3.5 – Структура клієнтської частини додатку

3.3 Результат виконання роботи

У результаті виконання кваліфікаційної роботи бакалавра були досягнуті такі результати:

- Реалізовано механізм авторизації та реєстрації користувачів
- Додавання, редагування, видалення питань, відповідей, коментарів та їх перегляд
- Присвоєння відповідних тегів питанням
- Можливість голосувати за контент
- Система репутації користувачів
- Система ролей
- Перегляд профілю

– Імплементація reCAPTCHA для боротьби зі спамом
– Фільтрація небезпечних зображень
– Опублікована політика контенту
– Панель модератора

Головна сторінка сайту зображена на рисунку 3.6. При натисканні на заголовок питання потрапляє на сторінку перегляду питання (рис. 3.7).

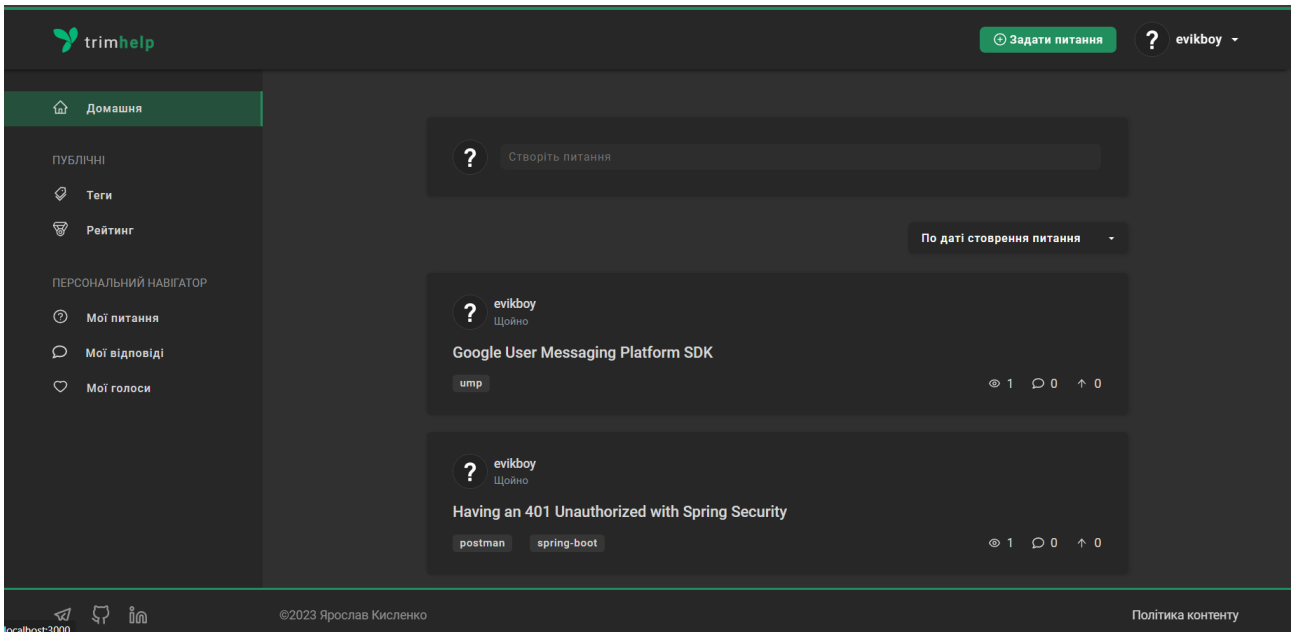


Рисунок 3.6 – Головна сторінка сайту

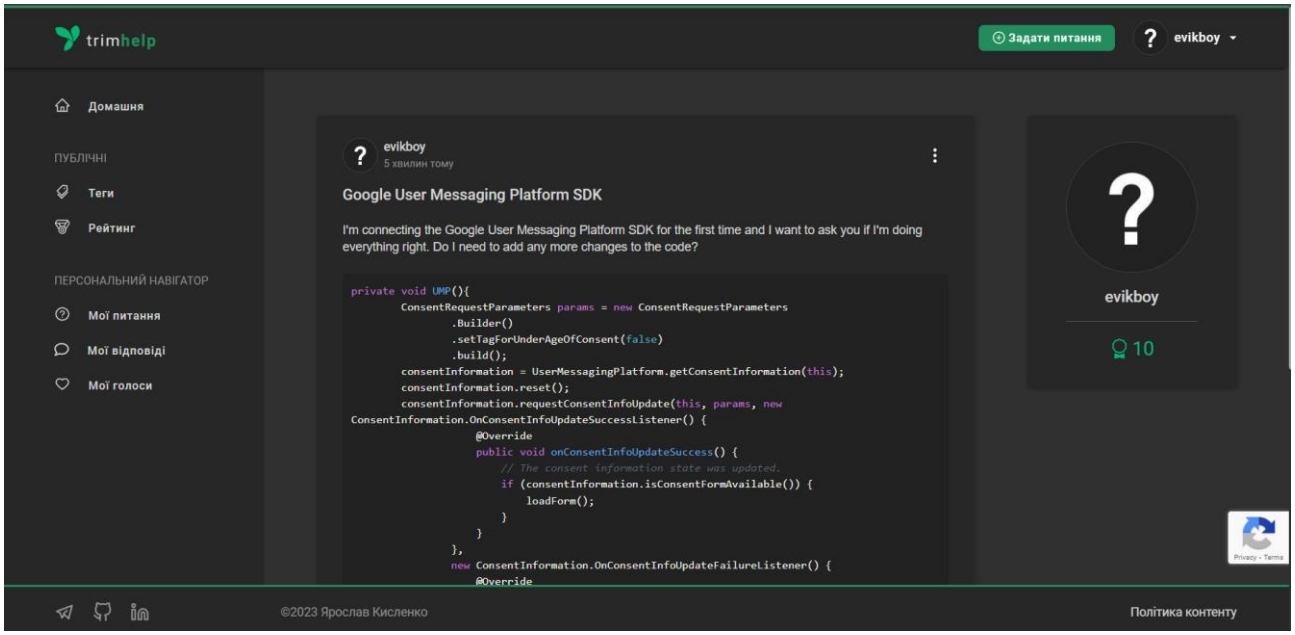


Рисунок 3.7– Сторінка перегляду питання

Авторизований користувач може опублікувати свою відповідь під питанням (рис. 3.8).

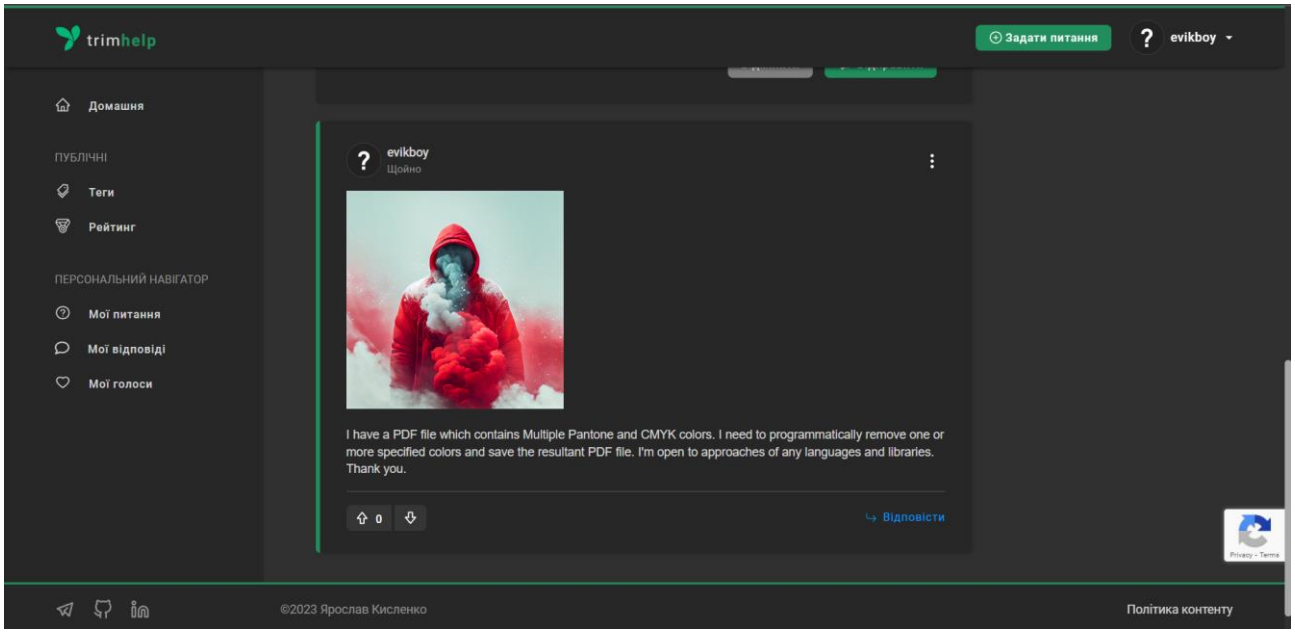


Рисунок 3.8 – Публікація відповіді авторизованим користувачем

Авторизований користувач може відредагувати або видалити лише свій контент. При успішній операції, на екрані в лівому куті з’явиться повідомлення про успіх (рис. 3.9, рис. 3.10).

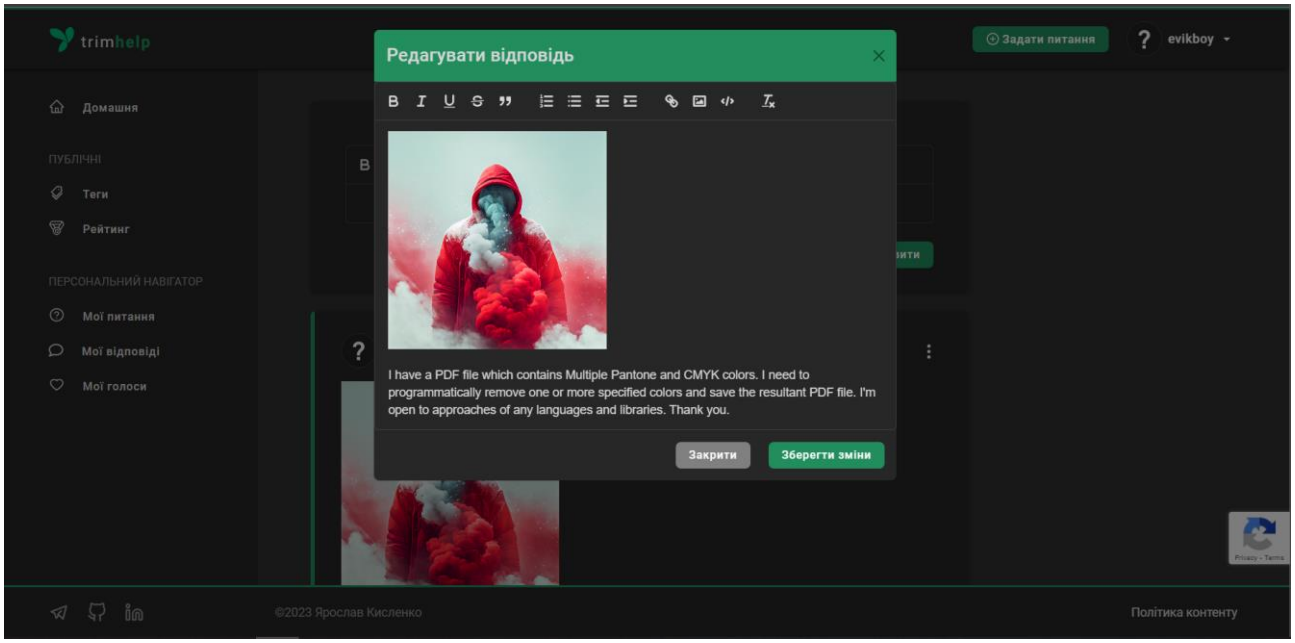


Рисунок 3.9– Вікно редагування відповіді

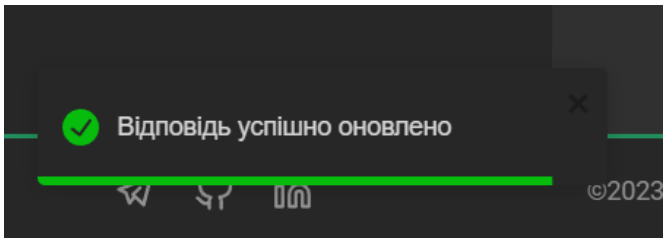


Рисунок 3.10 – Інформування користувача про успішне радагування

Користувач не має права голосувати за свої повідомлення, щоб не зловживати цим в цілях підняття репутації (рис. 3.11).

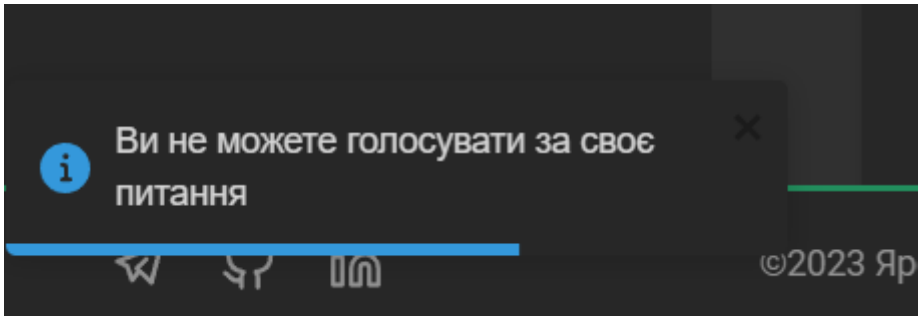


Рисунок 3.11– Спроба проголосувати за власний контент

Для відправки коментарів, користувач має заробити більше 50 репутації (рис. 3.12).

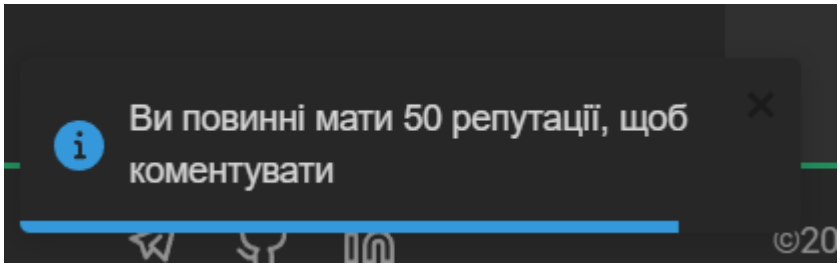


Рисунок 3.12 – Обмеження для неактивних користувачів

При неодноразовій відправці відповідей на питання, користувач має роз’язати капчу (рис. 3.13).

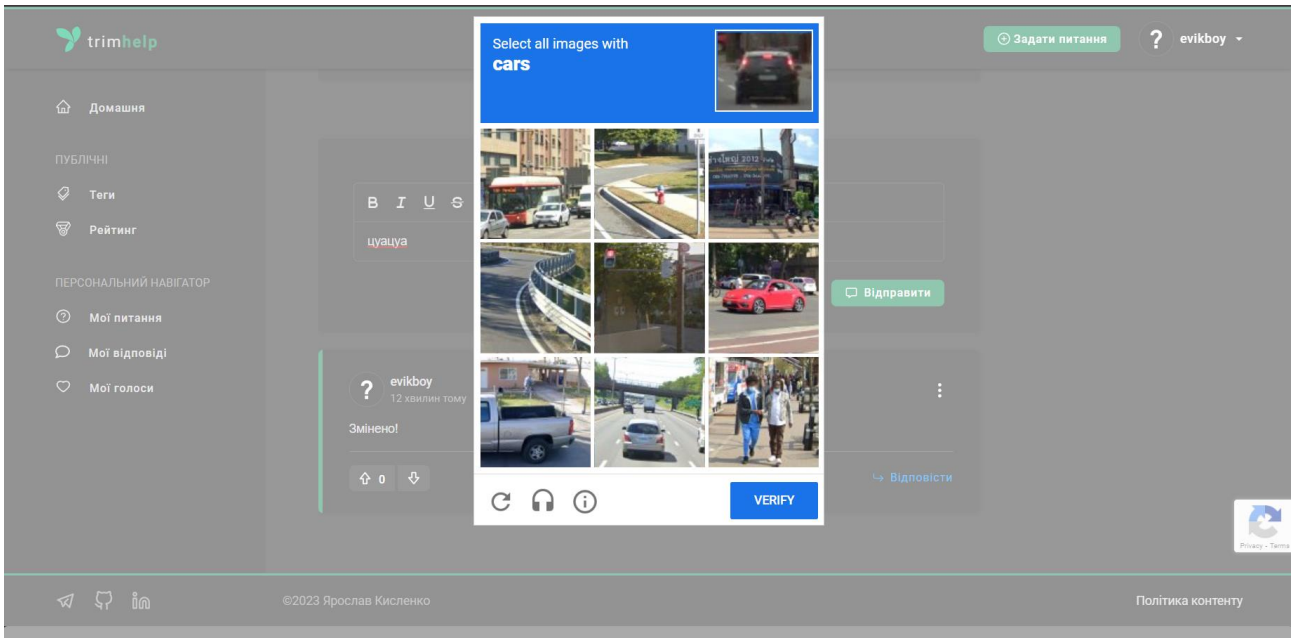


Рисунок 3.13 – Вікно reCAPTCHA

Якщо користувач завантажить в редактор тексту небезпечне зображення, то виникне помилка (рис. 3.14).

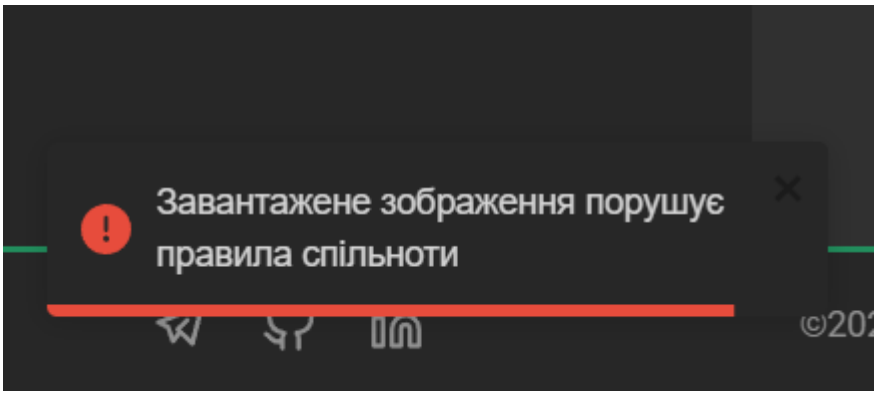


Рисунок 3.14– спроба користувача завантажити NSFW контент

Сторінка профілю користувача зі зібраною його статистикою показана на рисунку 3.15. Сторінки модератора – на рисунках 3.16 – 3.18.

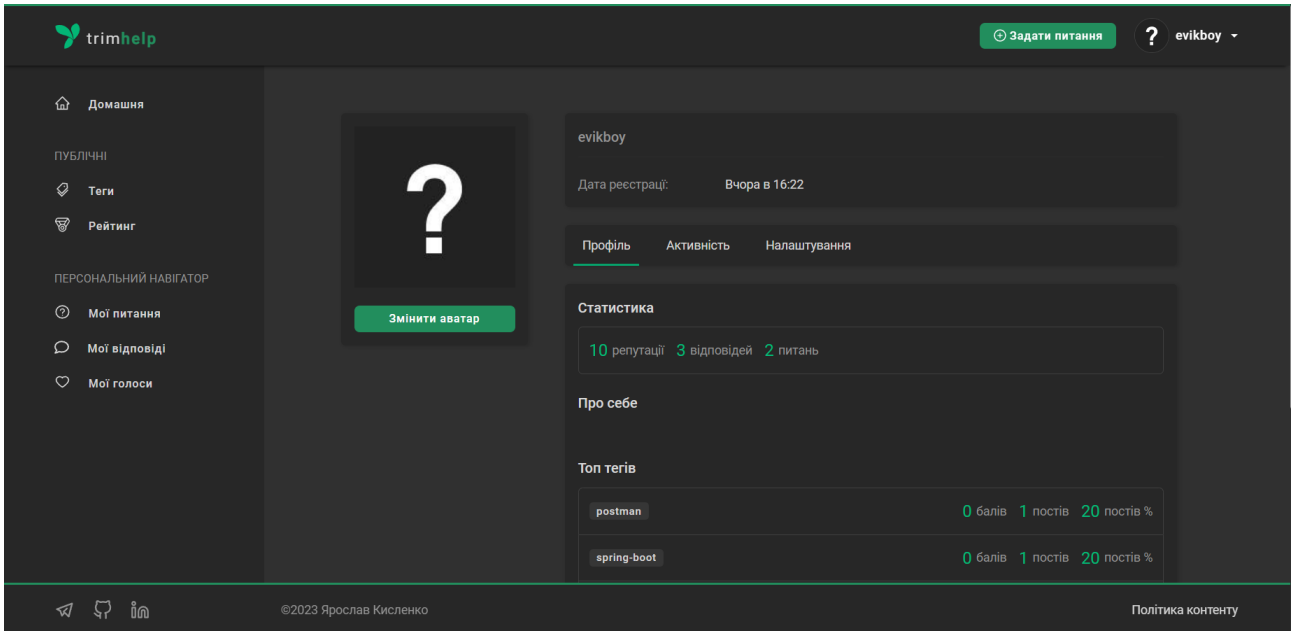


Рисунок 3.15 – Сторінка профілю користувача

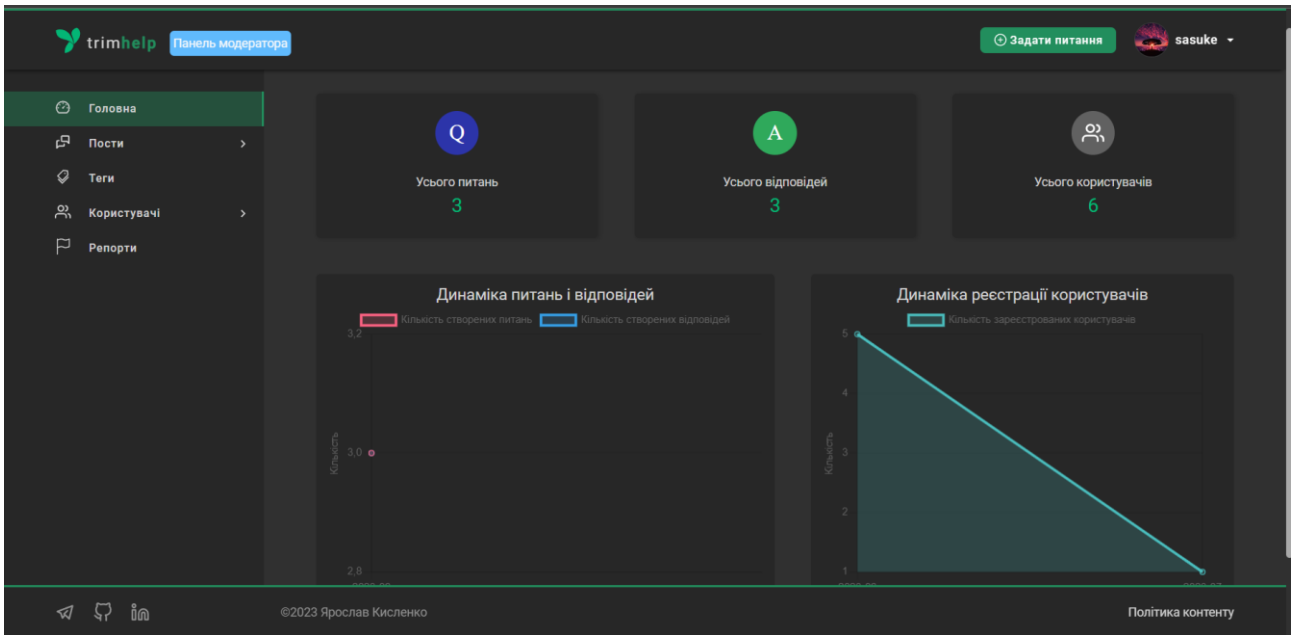


Рисунок 3.16 – Панель модератора

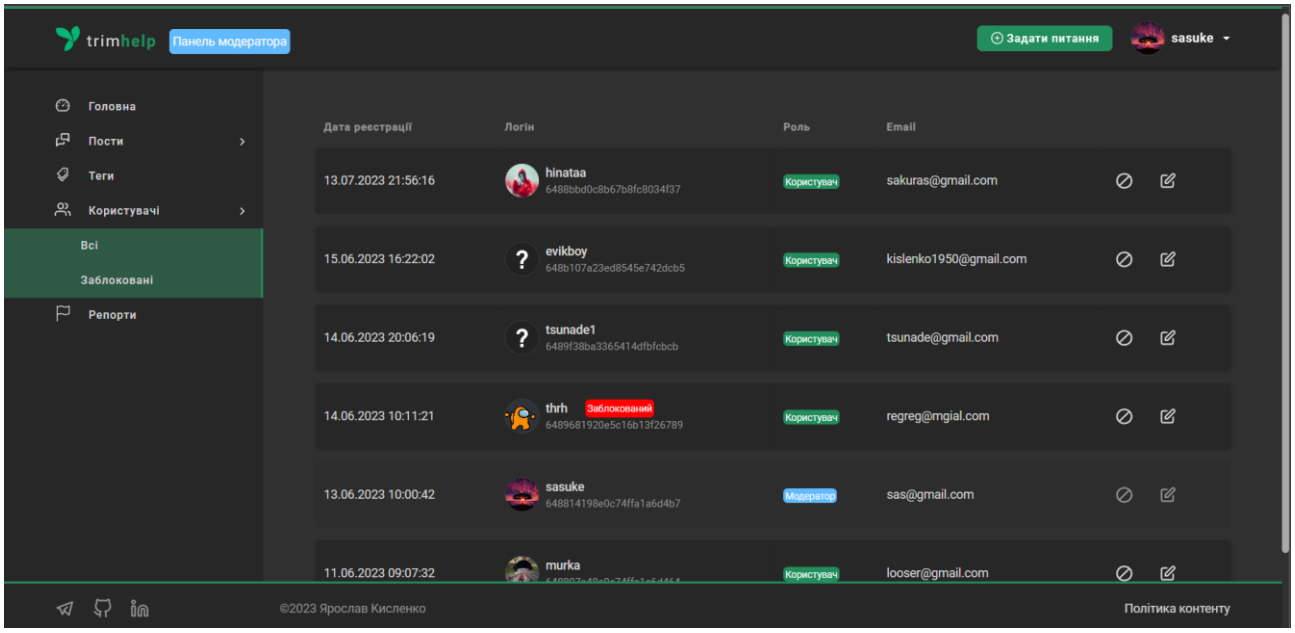


Рисунок 3.17– Панель управління користувачами

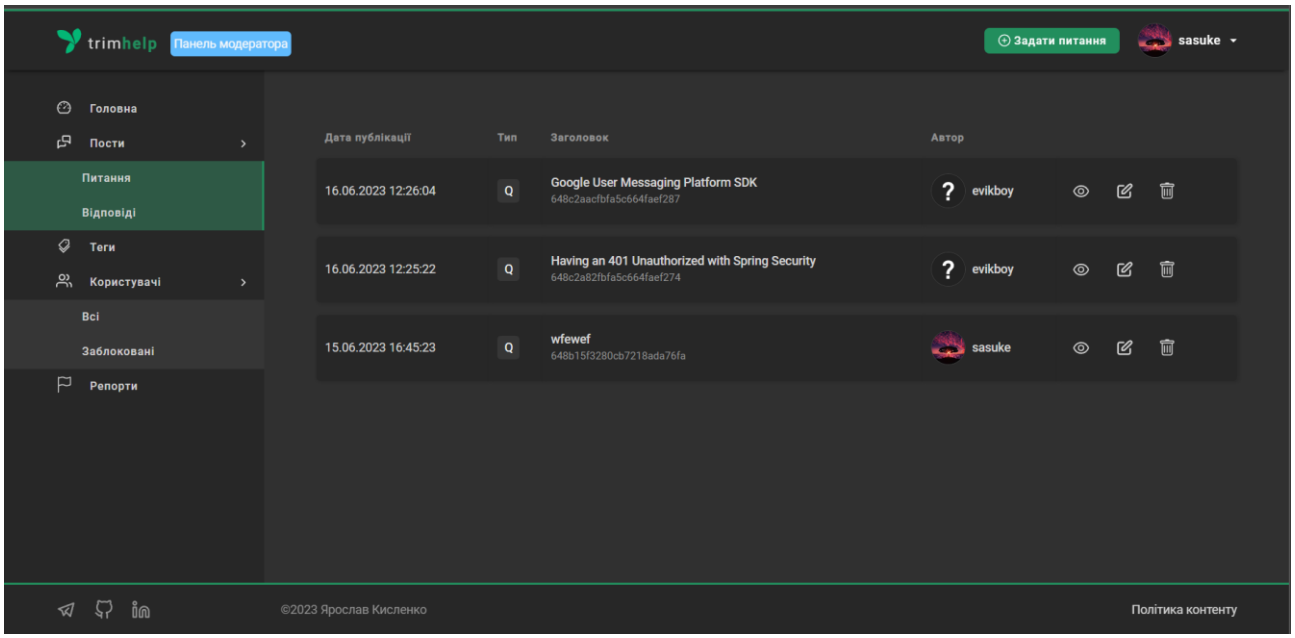


Рисунок 3.18– Панель управління питаннями та відповідями

ВИСНОВКИ

У результаті виконання кваліфікаційної бакалаврської роботи розроблено інформаційну систему управління користувацьким контентом на веб-форумі та виконано такі завдання.

1) Проведено комплексний огляд літератури, в якому вивчалися існуючі дослідження та знання про веб-форуми та користувацький контент. Основна увага була зосереджена на визначенні та аналізі ефективних стратегій управління та роботи з користувацьким контентом.
2) Здійснено аналіз аналогічних систем з метою виявлення ключових факторів.
3) Розглянуто та обрано сучасні інструменти розробки, такі як MERN стек технологій, щоб забезпечити ефективну реалізацію системи управління контентом.
4) Розроблено ER-діаграму бази даних, що відображає зв’язки між різними сутностями системи та структуру зберігання даних.
5) Спроєктовано структуру серверної та клієнтської частин. Для кожної частини були визначені основні компоненти та модулі, що забезпечують їх функціонування.
6) Реалізовано серверну сторону, включаючи розробку та інтеграцію необхідних компонентів і модулів.
7) Реалізовано клієнтську сторону системи, забезпечивши її функціонування та взаємодію з сервером.
8) Проведено тестування системи з метою виявлення будь-яких помилок, які могли б вплинути на її функціональність та надійність.

Варто відзначити, що система не є ідеальною і ще існують певні аспекти, які можна покращити. Одним з таких аспектів є необхідність розширення можливостей модерації контенту, включаючи інструменти для швидкого виявлення небажаного вмісту, додавання панелі адміністратора для управління ролями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Type of Online Forum is Right for My Community? | Forumbee Blog [Electronic resource]. URL: https://forumbee.com/article/y7h1xv/what-type-of-online-forum-is-right-for-my-community- (accessed: 04.06.2023).
2. Top 5 Benefits Of Online Forums You Can’t Ignore in 2023 [Electronic resource]. URL: https://www.kahootz.com/benefits-of-online-forums/ (accessed: 04.06.2023).
3. 7 Workplace Collaboration Statistics to Drive Teamwork | ClearCompany [Electronic resource]. URL: https://blog.clearcompany.com/7-workplace-collaboration-statistics-that-will-have-you-knocking-down-cubicles (accessed: 04.06.2023).
4. User-Generated Content (UGC): Pros and Cons & SEO Benefit [Electronic resource]. URL: https://www.link-assistant.com/news/user-generated-content.html (accessed: 05.06.2023).
5. Good strategies for managing user-generated content - Google AdSense Help [Electronic resource]. URL: https://support.google.com/adsense/answer/3011869?sjid=3557706521043916 39-EU (accessed: 21.05.2023).
6. Roberts S.T. UCLA UCLA Previously Published Works Title Content moderation Publication Date Title Content Moderation Synonyms Content screening; community management; community moderation Author and Affiliation.
7. Gillespie T. Content moderation, AI, and the question of scale // Big Data Soc. SAGE Publications Ltd, 2020. Vol. 7, № 2.
8. Reputation system - Wikipedia [Electronic resource]. URL: https://en.wikipedia.org/wiki/Reputation_system (accessed: 05.06.2023).
9. Aggarwal S., Verma J. Comparative analysis of MEAN stack and MERN stack // International Journal of Recent Research Aspects. 2018. Vol. 5. P. 127–132.

10. How do MERN stack technologies work together? [Electronic resource]. URL: https://www.bocasay.com/how-does-the-mern-stack-work/ (accessed: 07.06.2023).
11. React JS How Your Business Can Benefit From This Technology [Electronic resource]. URL: https://triare.net/insights/why-use-react-js-examples-of-facebook-netflix-and-others/ (accessed: 07.06.2023).
12. Aggarwal S. Modern Web-Development using ReactJS // International Journal of Recent Research Aspects. 2018. Vol. 5. P. 133–137.
13. React Components In Theory - DEV Community [Electronic resource]. URL: https://dev.to/aditi05/react-components-in-theory-3n9f (accessed: 07.06.2023).
14. Internet Applications R. Ausarbeitungen zum Seminar. 2016.
15. React Redux: Building Modern Web Apps with the ArcGIS JS API [Electronic resource]. URL: https://www.esri.com/arcgis-blog/products/3d-gis/3d-gis/react-redux-building-modern-web-apps-with-the-arcgis-js-api/ (accessed: 07.06.2023).
16. Why develop with Node.js? [Electronic resource]. URL: https://www.bocasay.com/project-develop-nodejs/ (accessed: 07.06.2023).
17. Organizing your Express.js project structure for better productivity - LogRocket Blog [Electronic resource]. URL: https://blog.logrocket.com/organizing-express-js-project-structure-better-productivity/ (accessed: 07.06.2023).
18. Analysis of reCAPTCHA effectiveness. 2010.
19. reCAPTCHA - Wikipedia [Electronic resource]. URL: https://en.wikipedia.org/wiki/ReCAPTCHA (accessed: 16.06.2023).
20. Entity Relationship Diagram Definition , Explanation & Examples | Secoda [Electronic resource]. URL: https://www.secoda.co/glossary/entity-relationship-diagram (accessed: 08.06.2023).

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

A1 UserController.js

```
const conf = require('../config')
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')
const nswf = require('../utils/nsfwModel')
const fs = require('fs')
const path = require('path')
const { v4: uuidv4 } = require('uuid')
const { secret, expiresIn } = conf.jwt
const { User, Question, Answer, Vote, Tag, Role } =
require('../models')
const { errorWrapper, errorGenerator } = require('../errors')

const login = errorWrapper(async (req, res) => {
    const { email, password } = req.body
    const existingUser = await User.findOne({ email })

    if (!existingUser) errorGenerator({ statusCode: 401, message:
'Невірна адреса пошти або пароль'})

    const passwordMatch = await bcrypt.compare(password,
existingUser.password)

    if (!passwordMatch) errorGenerator({ statusCode: 401, message:
'Невірна адреса пошти або пароль'})

    const token = jwt.sign(
        { userId: existingUser._id, username:
existingUser.username, email: existingUser.email },
        secret,
        { expiresIn: expiresIn }
    )

    return res.status(200).json({
        user: existingUser,
        token: `Bearer ${token}`
    })
})

const register = errorWrapper(async (req, res) => {
    const { email, username, password } = req.body
    const existingEmail = await User.findOne({ email })
    const existingUsername = await User.findOne({ username })

    if (existingEmail) errorGenerator({ statusCode: 409, message:
'Пошта все зайнята. Спробуйте іншу'})
```

```
    if (existingUsername) errorGenerator({ statusCode: 409,
message: 'Логін вже зайнятий. Спробуйте інший'})

    const salt = await bcrypt.genSalt(10)
    const hashedPassword = await bcrypt.hash(password, salt)

    const userRole = await Role.findOne({value: 'BASIC_USER'})

    const newUser = new User({
        email,
        username,
        password: hashedPassword,
        roles: [userRole.value]
    })

    const savedUser = await newUser.save()

    const token = jwt.sign(
        { userId: newUser._id, username: newUser.username, email:
newUser.email },
        secret,
        { expiresIn: expiresIn }
    )

    return res.status(201).json({
        user: savedUser,
        token: `Bearer ${token}`
    })
})

const getMe = errorWrapper(async (req, res) => {
    const user = await User.findById(req.user.id)

    if (!user) errorGenerator({ statusCode: 404, message:
"Користувач не знайдений"})

    const token = jwt.sign(
        { userId: user._id, username: user.username, email:
user.email },
        secret,
        { expiresIn: expiresIn }
    )

    return res.status(200).json({
        user: user,
        token: token
    })
})

const getUserQuestions = errorWrapper(async (req, res) => {
    const { userId } = req.params
```

```
    const questions = await Question.find({ user: userId })
        .populate('user', 'username avatarUrl')
        .populate('tags')
    .sort({ createdAt: -1 })

    if (!questions) errorGenerator({ statusCode: 404, message:
'Питання не знайдені'})

    res.status(200).json({ questions })
})

const getUserAnswers = errorWrapper(async (req, res) => {
    const { userId } = req.params

    const answers = await Answer.find({ user: userId })
        .populate('question', 'title tags')
        .populate({
            path: 'question',
            select: 'title',
            populate: {
                path: 'tags',
                select: 'name',
            }
        })
    .sort({ createdAt: -1 })

    if (!answers) errorGenerator({ statusCode: 404, message:
'Відповіді не знайдені'})

    res.status(200).json({ answers })
})

const getUserVotes = errorWrapper(async (req, res) => {
    const userId = req.user.id

    const votes = await Vote.find({ user: userId, voteType: { $ne:
'Novote' } })
    .sort({ updatedAt: -1 })

    if (!votes) errorGenerator({ statusCode: 404, message: 'Голоси
не знайдені'})

    const formattedVotes = await Promise.all(
        votes.map(async (vote) => {
            const formattedVote = {
                targetType: vote.targetType,
                voteType: vote.voteType,
                user: vote.user,
                createdAt: vote.updatedAt,
            }
```

```
            if (vote.targetType === 'Question') {
                const question = await
Question.findById(vote.targetId).populate('tags', 'name')
                if (question) {
                    formattedVote.target = {
                        _id: question._id,
                        questionTitle: question.title,
                        questionId: question._id,
                        tags: question.tags,
                        user: question.user,
                    }
                }
            } else if (vote.targetType === 'Answer') {
                const answer = await
Answer.findById(vote.targetId)
                    .populate({
                        path: 'question',
                        select: 'title',
                        populate: {
                            path: 'tags',
                            select: 'name',
                    },
                })
                if (answer) {
                    formattedVote.target = {
                        _id: answer._id,
                        user: answer.user,
                        questionTitle: answer.question.title,
                        questionId: answer.question._id,
                        tags: answer.question.tags
                    }
                }
            }

            return formattedVote
        })
    )

    res.status(200).json({ votes: formattedVotes })
})

const getById = errorWrapper(async (req, res) => {
    const { userId } = req.params

    const user = await User.findById(userId)

    if (!user) errorGenerator({ statusCode: 404, message:
"Користувач не знайдений"})

    const questionCount = await Question.countDocuments({ user:
```

```
user._id })
    const answerCount = await Answer.countDocuments({ user:
user._id })

    const userQuestions = await Question.find({ user: user._id })
    const userAnswers = await Answer.find({ user: user._id })

    const allTags = userQuestions.flatMap(question =>
question.tags)

    const tagCounts = allTags.reduce((counts, tag) => {
        counts[tag] = (counts[tag] || 0) + 1
        return counts
    }, {})

    const tagIds = Object.keys(tagCounts)
    const tags = await Tag.find({ _id: { $in: tagIds } })

    const updatedTopTags = await Promise.all(
        tags.map(async (tag) => {
            const tagId = tag._id.toString()

            const questionIdsWithTag =
userQuestions.filter(question =>
question.tags.includes(tagId)).map(question => question._id)

            let voteCount = 0

            for (const questionId of questionIdsWithTag) {
                const question = await
Question.findById(questionId)
                if (question) {
                    voteCount += question.upvotes -
question.downvotes
                }
            }

            const answerIdsWithTag = await Answer.find( {
question: { $in: questionIdsWithTag }, user: userId
}).distinct('_id')
            for (const answerId of answerIdsWithTag) {
                const answer = await Answer.findById(answerId)
                if (answer) {
                    voteCount += answer.upvotes - answer.downvotes
                }
            }

            return {
                tag: tag.name,
                questionCount: questionIdsWithTag.length,
                answerCount: answerIdsWithTag.length,
```

```
                voteCount
            }
        })
    )
    updatedTopTags.sort((a, b) => b.voteCount - a.voteCount)
    const updatedTopTagsLimited = updatedTopTags.slice(0, 5)

    const topPosts = []

    for (const question of userQuestions) {
        const voteCount = question.upvotes - question.downvotes
        topPosts.push({
            postId: question._id,
            title: question.title,
            createdAt: question.createdAt,
            type: 'Question',
            voteCount
        })
    }

    for (const answer of userAnswers) {
        const question = await Question.findById(answer.question)
        if (question) {
            const voteCount = answer.upvotes - answer.downvotes
            topPosts.push({
                postId: answer._id,
                title: question.title,
                createdAt: answer.createdAt,
                type: 'Answer',
                voteCount
            })
        }
    }

    topPosts.sort((a, b) => {
        if (a.voteCount === b.voteCount) {
            return new Date(b.createdAt) - new Date(a.createdAt)
        } else {
            return b.voteCount - a.voteCount
        }
    })
    const topPostsLimited = topPosts.slice(0, 5)

    return res.status(200).json({ user, questionCount,
answerCount, topTags: updatedTopTagsLimited, topPosts:
topPostsLimited })
})

const getActivity = errorWrapper(async (req, res) => {
    const { userId } = req.params
```

```
    const userQuestions = await Question.find({ user: userId
}).sort({ createdAt: -1 })
    const userAnswers = await Answer.find({ user: userId }).sort({
createdAt: -1 }).populate('question', 'title')

    return res.status(200).json({ userQuestions, userAnswers })
})

const uploadAvatar = errorWrapper(async (req, res) => {
    const userId = req.user.id

    if (!req.file) {
        return res.status(400).json({ error: 'Файл аватара не
надано' })
    }

    const fileExtension = req.file.originalname.split('.').pop()
    const fileName = `${uuidv4()}.${fileExtension}`
    const newFilePath = `uploads/${fileName}`

    fs.renameSync(req.file.path, newFilePath)

    const isSafe = await nswf.isSafe(newFilePath)

    if (!isSafe) {
        fs.unlinkSync(newFilePath)
        errorGenerator({ statusCode: 400, message: 'Завантажене
зображення порушує правила спільноти'})
    }

    const user = await User.findByIdAndUpdate(
        userId,
        { avatarUrl: fileName },
        { new: true }
    )

    if (!user) errorGenerator({ statusCode: 404, message:
"Користувач не знайдений"})

    res.status(200).json({ avatarUrl: fileName })
})

const updateUserAbout = errorWrapper(async (req, res) => {
    const userId = req.user.id
    const { about } = req.body

    const user = await User.findByIdAndUpdate(
        userId,
        { about },
        { new: true }
    )
```

```
    if (!user) errorGenerator({ statusCode: 404, message:
"Користувач не знайдений" })

    res.status(200).json({ about })
})

const getStats = errorWrapper(async (req, res) => {
    const questionStats = await Question.aggregate([
        {
            $group: {
                _id: { $dateToString: { format: '%Y-%m', date:
'$createdAt' } },
                questionCount: { $sum: 1 },
            },
        },
        { $sort: { _id: 1 } }
    ])

    const answerStats = await Answer.aggregate([
        {
            $group: {
                _id: { $dateToString: { format: '%Y-%m', date:
'$createdAt' } },
                answerCount: { $sum: 1 },
            },
        },
        { $sort: { _id: 1 } }
    ])

    const userStats = await User.aggregate([
        {
            $group: {
                _id: { $dateToString: { format: '%Y-%m', date:
'$createdAt' } },
                userCount: { $sum: 1 },
            },
        },
        { $sort: { _id: 1 } },
    ])

    const questionTotalCount = questionStats.reduce((acc, stat) =>
acc + stat.questionCount, 0)
    const answerTotalCount = answerStats.reduce((acc, stat) => acc
+ stat.answerCount, 0)
    const userTotalCount = await User.countDocuments()

    const stats = {
        questionStats,
        answerStats,
        userStats,
```

```
        questionTotalCount,
        answerTotalCount,
        userTotalCount
    }

    res.json(stats)
})

const getAll = errorWrapper(async (req, res) => {
    const users = await User.find().sort({ createdAt: -1 })

    res.status(200).json(users)
})

const update = errorWrapper(async (req, res) => {
    const { userId } = req.params
    const { username, email, about } = req.body

    const user = await User.findById(userId)

    if (!user) errorGenerator({ statusCode: 404, message:
'Користувач не знайдений' })

    const existingEmailUser = await User.findOne({ email })
    const existingLoginUser = await User.findOne({ username })

    if (existingEmailUser && existingEmailUser._id.toString() !==
userId) errorGenerator({ statusCode: 409, message: 'Пошта вже
зайнята. Спробуйте іншу' })

    if (existingLoginUser && existingLoginUser._id.toString() !==
userId) errorGenerator({ statusCode: 409, message: 'Логін вже
зайнятий. Спробуйте інший' })

    user.username = username
    user.email = email
    user.about = about

    if (req.file) {
        const fileExtension =
req.file.originalname.split('.').pop()
        const fileName = `${uuidv4()}.${fileExtension}`
        const newFilePath = `uploads/${fileName}`

        fs.renameSync(req.file.path, newFilePath)

        if (user.avatarUrl && user.avatarUrl !== 'default-
avatar.jpg') {
            const previousAvatarPath = path.join(__dirname,
`../uploads/${user.avatarUrl}`)
```

```
            fs.unlink(previousAvatarPath, (err) => {
                if (err) {
                    console.error(err)
                }
            })
        }

        user.avatarUrl = fileName
    }

    const updatedUser = await user.save()

    res.status(200).json(updatedUser)
})

const getBannedUsers = errorWrapper(async (req, res) => {
    const bannedUsers = await User.find({ ban: { $ne: null }
}).sort({ createdAt: -1 })
    res.status(200).json(bannedUsers)
})

module.exports = {
    login,
    register,
    getMe,
    getUserQuestions,
    getUserAnswers,
    getUserVotes,
    getById,
    getActivity,
    uploadAvatar,
    updateUserAbout,
    getStats,
    getAll,
    update,
    getBannedUsers
}
```

A2 QuestionController.js

```
const { Question, Answer, Comment, Tag } = require('../models')
const { errorWrapper, errorGenerator } = require('../errors')

const getAll = errorWrapper(async (req, res) => {
    const { filter } = req.query

    let sortField
    if (filter === 'dob_question') {
        sortField = 'createdAt'
    } else if (filter === 'byLast_answer') {
        sortField = 'lastAnswerAt'
    } else if (filter === 'byNumber_views') {
```

```
    sortField = 'views'
} else if (filter === 'byNumber_answers') {
    sortField = 'answersCount'
} else if (filter === 'byNumber_votes') {
    sortField = { $subtract: ['$upvotes', '$downvotes'] }
} else {
    sortField = 'createdAt'
}

const questions = await Question.aggregate([
    {
        $addFields: {
            answersCount: { $size: '$answers' },
        },
    },
    {
        $lookup: {
            from: 'users',
            localField: 'user',
            foreignField: '_id',
            as: 'user',
        },
    },
    {
        $unwind: '$user',
    },
    {
        $lookup: {
            from: 'tags',
            localField: 'tags',
            foreignField: '_id',
            as: 'tags',
        },
    },
    {
        $lookup: {
            from: 'answers',
            localField: 'answers',
            foreignField: '_id',
            as: 'answers',
        },
    },
    {
        $addFields: {
            lastAnswerAt: { $max: '$answers.createdAt' },
        },
    },
    {
        $sort: {
            [sortField]: -1,
        },
```

```
        },
        {
            $project: {
                'user.password': 0,
                'user.email': 0,
                'user.createdAt': 0,
                'user.updatedAt': 0,
                'user.reputation': 0,
                'user.reputationEvents': 0,
                'votes': 0,
            }
        }
    ])

    if (!questions) errorGenerator(404, 'Питання не знайдені')

    res.status(200).json({ questions })
})

const getById = errorWrapper(async (req, res) => {
    const _id = req.params.questionId
    const userId = req.user

    let query = Question.findByIdAndUpdate(_id, { $inc: { views: 1
} })
    .populate('user', 'username avatarUrl reputation')
    .populate('tags')

    if (userId) {
        query = query.populate({
            path: 'votes',
            match: { user: userId }
        })
    }

    const question = await query

    if (!question) errorGenerator({ statusCode: 404, message:
'Питання не знайдено'})

    res.status(200).json(question)
})

const create = errorWrapper(async (req, res) => {
    const { title, body } = req.body
    const userId = req.user.id

    const question = new Question({
        title, body,
        user: userId
    })
```

```
    await question.save()

    res.status(201).json(question)
})

const removeCommentsRecursively = async commentId => {
    const comment = await Comment.findOneAndDelete({ _id:
commentId })
    console.log(comment)

    if (!comment) {
        return
    }

    const childCommentIds = comment.comments

    for (const childCommentId of childCommentIds) {
        await removeCommentsRecursively(childCommentId)
    }
}

const remove = errorWrapper(async (req, res) => {
    const { questionId } = req.params
    console.log(questionId)

    const deletedQuestion = await Question.findOneAndDelete({ _id:
questionId })

    if (!deletedQuestion) errorGenerator({ statusCode: 404,
message: 'Питання не знайдено'})

    const deletedAnswers = await Answer.find({ question:
questionId })
    const answerIds = deletedAnswers.map(answer => answer._id)

    await Answer.deleteMany({ question: questionId })

    const rootComments = await Comment.find({ parentId: { $in:
answerIds }, parentType: 'answers' })

    for (const rootComment of rootComments) {
        await removeCommentsRecursively(rootComment._id)
    }

    res.status(200).json(deletedQuestion)
})

const update = errorWrapper(async (req, res) => {
    const _id = req.params.questionId
    const { title, body, tags } = req.body
    console.log(tags)
```

```
    const tagArray = tags.split(',').map((tag) => tag.trim())
    const questionTags = []

    for (const tagName of tagArray) {
        let tag = await Tag.findOne({ name: tagName })

        if (!tag) {
            tag = new Tag({ name: tagName })
            await tag.save()
        }
        questionTags.push(tag._id)
    }

    const updatedQuestion = await Question.findOneAndUpdate({ _id
},
        { $set: { title, body, tags: questionTags } },
        { new: true }
    )

    if (!updatedQuestion) errorGenerator({ statusCode: 404,
message: 'Питання не знайдено' })

    await Tag.updateMany(
        { questions: _id },
        { $pull: { questions: _id } }
    )

    for (const tagId of questionTags) {
        await Tag.findByIdAndUpdate(tagId, { $addToSet: {
questions: { $each: [_id] } } })
    }

    res.status(200).json(updatedQuestion)
})

module.exports = {
    getAll,
    getById,
    create,
    remove,
    update,
}
```

A3 AnswerController.js

```
const { Answer, Question, Comment, Vote } = require('../models')
const { errorWrapper, errorGenerator } = require('../errors')
const { validateHuman } = require('../utils/validateHuman')

const getByQuestionId = errorWrapper(async (req, res) => {
    const { questionId } = req.params
```

```
    const userId = req.user

    let query = Answer.find({ question: questionId
}).populate('user comments', 'username avatarUrl')

    if (userId) {
        query = query.populate({
            path: 'votes',
            match: { user: userId }
        })
    }

    const answers = await query

    if (!answers) errorGenerator({ statusCode: 404, message:
'Відповіді не знайдено'})

    res.status(200).json(answers)
})

const create = errorWrapper(async (req, res) => {
    const { questionId } = req.params
    const { body, token } = req.body

    const human = await validateHuman(token)
    console.log(human)

    const userId = req.user.id

    const answer = new Answer({
        body,
        question: questionId,
        user: userId
    })
    await answer.save()
    await answer.populate('user', 'username avatarUrl')

    const question = await Question.findById(questionId)

    if (!question) errorGenerator({ statusCode: 404, message:
'Питання не знайдено'})

    question.answers.push(answer._id)

    await question.save()

    res.status(201).json(answer)
})

const update = errorWrapper(async (req, res) => {
    const { answerId } = req.params
```

```
    const { body } = req.body

    const updatedAnswer = await Answer.findByIdAndUpdate(
        answerId,
        { body },
        { new: true }
    ).populate('user', 'username avatarUrl')

    if (!updatedAnswer) {
        errorGenerator({ statusCode: 404, message: 'Відповідь не
знайдено' })
    }

    res.status(200).json(updatedAnswer)
})

const remove = errorWrapper(async (req, res) => {
    const { answerId } = req.params

    const deletedAnswer = await Answer.findByIdAndDelete(answerId)

    if (!deletedAnswer) errorGenerator({ statusCode: 404, message:
'Відповідь не знайдено' })

    const question = await
Question.findById(deletedAnswer.question)

    if (!question) errorGenerator({ statusCode: 404, message:
'Питання не знайдено' })

    deletedAnswer.comments.forEach(async (commentId) => {
        await Comment.findByIdAndDelete(commentId)
    })

    const index = question.answers.indexOf(answerId)
    if (index !== -1) {
        question.answers.splice(index, 1)
    }

    await question.save()

    res.status(200).json(deletedAnswer)
})

const getAll = errorWrapper(async (req, res) => {
    const answers = await Answer.find()
        .populate('question', 'title')
        .populate('user', 'username avatarUrl')
        .sort({ createdAt: -1 })

    if (!answers) errorGenerator({ statusCode: 404, message:
```

```
'Відповіді не знайдено'})

    res.status(200).json(answers)
})

module.exports = {
    getByQuestionId,
    create,
    update,
    remove,
    getAll
}
```

A4 CommentController.js

```
const { Comment, Answer, User } = require('../models')
const { errorWrapper, errorGenerator } = require('../errors')

const getByAnswerId = errorWrapper(async (req, res) => {
    const { answerId } = req.params

    const retrieveAllComments = async (parentId) => {
        const comments = await Comment.find({ parentId
}).populate('user', 'username avatarUrl').sort({ createdAt: 1 })

        const childComments = await Promise.all(
            comments.map(comment =>
retrieveAllComments(comment._id))
        )

        const flattenedChildComments = childComments.flat()

        return [...comments, ...flattenedChildComments].sort((a,
b) => a.createdAt - b.createdAt)
    }
    const allComments = await retrieveAllComments(answerId)

    res.status(200).json(allComments)
})

const create = errorWrapper(async (req, res) => {
    const { parentId, parentType } = req.params
    const { body, answerId } = req.body
    const userId = req.user.id

    const user = await User.findById(userId)

    if (user.reputation < 50) errorGenerator({ statusCode: 403,
message: 'У вас недостатньо репутації для відправки коментарів' })

    let parentEntity
    let answerEntity
```

```
    if (parentType === 'answers') {
        parentEntity = await Answer.findById(parentId)
    } else if (parentType === 'comments') {
        parentEntity = await Comment.findById(parentId)
    } else {
        errorGenerator({ statusCode: 400, message: 'Недійсний
parentType' })
    }

    answerEntity = await Answer.findById(answerId)

    if (!answerEntity) {
        errorGenerator({ statusCode: 404, message: 'Answer entity
не знайдено' })
    }

    if (!parentEntity) {
        errorGenerator({ statusCode: 404, message: 'Parent entity
не знайдено' })
    }

    const comment = new Comment({
        body,
        parentType,
        parentId,
        user: userId
    })

    await comment.save()
    await comment.populate('user', 'username avatarUrl')

    parentEntity.comments.push(comment._id)
    await parentEntity.save()

    // answerEntity.commentsCount += 1
    if (parentType === 'comments')
answerEntity.comments.push(comment._id)

    await answerEntity.save()

    res.status(201).json(comment)
})

module.exports = {
    getByAnswerId,
    create
}
```