Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОДЕРАЦІЇ КОНТЕНТУ В СОЦІАЛЬНИХ МЕРЕЖАХ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-1
Мартинов О. С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Путятіна О. Є.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Мартинову Олексію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного забезпечення для модерації контенту в соціальних мережах

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 28 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, нормативні матеріали з модерації цифрового контенту.

4. Перелік питань, що потрібно опрацювати в роботі
1. Проаналізувати предметну область модерації фото- та відеоконтенту в соціальних мережах.
2. Спроєктувати архітектуру, модель даних, черги модерації, рольовий доступ і механізм журналювання рішень.
3. Реалізувати вебзастосунок, перевірити основні сценарії роботи модератора й адміністратора та оформити результати.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність модерації мультимедійного контенту, постановка задачі, архітектура системи, модель даних, інтерфейси модератора й адміністратора, результати тестування.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів модерації контенту	20.04.26-22.04.26	
5	Формування кроків вирішення проблеми	23.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	21.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Путятіна О. Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 89 с., 14 табл., 23 рис., 22 джерел.

ВЕБЗАСТОСУНОК, ІСТОРІЯ РІШЕНЬ, МОДЕРАЦІЯ КОНТЕНТУ, ПОЛІТИКА МОДЕРАЦІЇ, СОЦІАЛЬНІ МЕРЕЖІ, ЧЕРГА МОДЕРАЦІЇ, EXPRESS, JWT, POSTGRESQL, PRISMA, RBAC, REACT.

Об’єктом роботи є процес модерації контенту в соціальних мережах.

Предметом роботи є методи та програмні засоби організації черг модерації, рольового доступу, застосування політик, фіксації рішень, ведення історії модерації та адміністративного керування системою.

Метою роботи є підвищення ефективності та контрольованості ручної модерації мультимедійного контенту шляхом розробки вебзастосунку з чергами обробки, рольовим доступом та аудитом рішень.

У роботі проаналізовано предметну область, сформовано вимоги, обґрунтовано архітектуру, спроєктовано модель даних і реалізовано прототип VideoModerationTool. Результатом є вебзастосунок для перегляду фото і відео, застосування політик, повторного розгляду неоднозначних матеріалів, ведення історії рішень та адміністративного контролю.

CONTENT MODERATION, DECISION HISTORY, EXPRESS, JWT, MODERATION POLICY, MODERATION QUEUE, POSTGRESQL, PRISMA, RBAC, REACT, SOCIAL NETWORKS, WEB APPLICATION.

The object of the work is the process of content moderation in social networks.

The subject of the work is the methods and software tools for organizing moderation queues, role-based access, policy application, decision recording, moderation history and administrative control.

The purpose of the work is to improve the efficiency and controllability of manual multimedia content moderation by developing a web application with processing queues, role-based access and decision auditing.

The work analyzes the domain, defines requirements, justifies the architecture, designs the data model and implements the VideoModerationTool prototype. The result is a web application for reviewing photo and video content, applying moderation policies, handling ambiguous cases, maintaining decision history and providing administrative control.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Аудитоспроможність – властивість системи, за якої її дії, рішення та зміни стану придатні до подальшої перевірки та аналізу

БД – база даних

Політика модерації – правило системи, що задає наслідок для матеріалу, наприклад блокування, обмеження для неповнолітніх або переміщення на повторну перевірку

Черга модерації – прикладно організований набір елементів контенту з певним статусом і правилами видачі модератору для опрацювання

ШІ – штучний інтелект

API – Application Programming Interface (інтерфейс програмування застосунків)

Docker Compose – інструмент для локального запуску кількох взаємопов'язаних сервісів однією командою

Express – серверний фреймворк для Node.js, який використовується для створення API та організації серверної логіки

JWT – JSON Web Token (компактний формат токена для передавання тверджень про автентифікованого користувача)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

PostgreSQL – реляційна система керування базами даних, у якій зберігаються користувачі, автори контенту, елементи черг, політики та історія модерації

Prisma – ORM, що використовується в серверній частині для типізованого доступу до PostgreSQL

RBAC – Role-Based Access Control (модель керування доступом на основі ролей користувачів)

React – бібліотека для побудови клієнтського інтерфейсу на основі компонентів

REST – Representational State Transfer (стиль організації прикладного API через стандартизовані HTTP-запити)

ВСТУП

Соціальні мережі перетворилися на один із центральних каналів масової комунікації, обміну досвідом, самопрезентації, комерційної активності та розповсюдження новин. Разом із цим різко зросли обсяги користувацького контенту, який надходить до платформ у вигляді фото, відео, коротких описів, метаданих і повторних публікацій. За таких умов підтримання безпечного цифрового середовища вже не може зводитися лише до зберігання файлів або до реакції на окремі скарги. Платформа повинна мати впорядкований процес модерації, у межах якого кожен матеріал отримує визначений статус, а рішення щодо нього можна пояснити, перевірити та, за потреби, переглянути [1–5].

Актуальність теми додатково посилюється зміною регуляторного середовища. Сучасні підходи до врядування платформ дедалі сильніше наголошують не тільки на результаті модерації, а й на процедурній якості самого процесу: хто ухвалює рішення, за яким правилом, чи можна пов’язати дію з конкретною політикою, чи зберігається історія, чи можливий повторний розгляд неоднозначного випадку [2], [6–10]. Для цифрової платформи модерація є не допоміжною дрібницею, а частиною її інфраструктурної логіки, яка впливає на довіру користувачів, сталість правил участі та загальну безпечність інформаційного середовища [1–3].

Особливу складність становить модерація мультимедійного контенту. Якщо текстове повідомлення часто можна оцінити за мовними ознаками, то фото і відеоматеріали вимагають врахування сюжету, візуального контексту, тривалості, підпису, походження публікації, історії автора та можливих наслідків рішення [4–5, 11–16]. Через це суто автоматичний підхід не є достатнім для всіх випадків, а ручна перевірка потребує спеціально організованого програмного середовища.

Практичний напрям роботи полягає у створенні вебзастосунку, який підтримує роботу модератора з чергами фото і відеоматеріалів, застосування політик, фіксацію рішень та адміністративний контроль за станом системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОДЕРАЦІЇ КОНТЕНТУ В СОЦІАЛЬНИХ МЕРЕЖАХ

1.1 Модерація як функція платформного врядування і предмет окремого системного аналізу

Модерація контенту в соціальних мережах є не окремою дією з видалення небажаного матеріалу, а сталою функцією платформного врядування. Вона визначає межі допустимої участі користувачів, порядок реагування на порушення, видимість матеріалів і спосіб пояснення рішень. Якщо така функція організована неструктуровано, платформа стикається не лише з небажаними публікаціями, а й з проблемою нерівномірних, непояснених або суперечливих рішень.

У науковій літературі модерацію описують з різних позицій. T. Gillespie розглядає її як приховану інфраструктурну працю платформи, без якої відкритий користувацький простір швидко втрачає керованість [1]. R. Gorwa пов'язує модерацію з ширшим поняттям платформного врядування, де важливими є правила, відповідальність, інституційний контроль і регуляторне середовище [2]. R. Gorwa, R. Binns і C. Katzenbach показують, що автоматизація не усуває політичних та організаційних суперечностей модерації, а переносить їх у площину проєктування алгоритмів, процедур і контролю [3].

Спільним у цих підходах є визнання того, що модерація не може бути зведена до технічного фільтра. Вона має процедурний характер: матеріал надходить до певного середовища, отримує статус, переглядається відповідальною особою або автоматизованим засобом, після чого рішення повинно мати зрозумілий наслідок. Відмінність між підходами полягає в акцентах. Одні автори підкреслюють невидиму працю модераторів, інші аналізують платформну владу, треті досліджують межі автоматизації. Для кваліфікаційної роботи важливим є саме перетин цих позицій: програмне

забезпечення для модерації має підтримувати не лише перегляд матеріалу, а й контрольований порядок дій.

Соціальна платформа відрізняється від звичайного сховища файлів тим, що матеріали з'являються постійно і здебільшого без попередньої редакційної перевірки. Потік користувацьких фото, відео, підписів, коментарів і повторних публікацій створює ситуацію, у якій рішення щодо одного матеріалу може залежати від контексту автора, походження публікації, попередніх скарг або чутливості зображеного змісту. Тому предметна область модерації містить не тільки сам контент, а й правила його руху через процес перевірки.

За відсутності керованого маршруту обробки виникають типові організаційні ризики: дублювання перегляду одного матеріалу, втрата підстав рішення, нерівномірне навантаження на модераторів, недостатня прозорість для адміністратора і складність повторної перевірки спірного випадку. Такі ризики мають змішану природу. Вони пов'язані з технічною інфраструктурою, організацією праці, нормативними очікуваннями щодо прозорості та фактичною здатністю платформи пояснити свої рішення [2], [22].

Узагальнене місце модерації в роботі соціальної платформи показано на рисунку 1.1. Схема відображає не конкретну програмну реалізацію, а предметну логіку: надходження матеріалу, розподіл на перевірку, ухвалення рішення, зміну стану і подальший контроль.

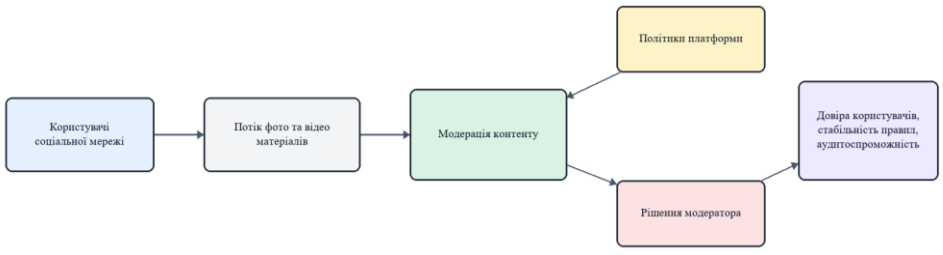

Рисунок 1.1 – Місце модерації контенту в загальній роботі соціальної платформи

1.2 Типи контенту і складність їх оцінювання

Складність модерації залежить від типу матеріалу. Текст, фото і відео створюють різні умови оцінювання, хоча в реальних соціальних мережах вони часто поєднуються в одній публікації. Текстовий фрагмент можна аналізувати за мовними ознаками, але підтекст, евфемізми, жаргон, цитування або іронія ускладнюють пряме тлумачення. Фото додає візуальний контекст: сцена, символіка, монтаж, наявність людей, вік учасників, видимі предмети і підпис можуть змінити оцінку одного й того самого зображення. Відео є ще складнішим через тривалість, зміну сцен, звук, накладений текст і фрагменти, які можуть містити порушення лише протягом кількох секунд.

Мультимедійний матеріал не варто розглядати як ізольований файл. Для модерації важливими є тип контенту, назва або підпис, джерело, автор, час появи, попередній статус, скарги, контекст поширення та можливість повторного перегляду. Якщо ці дані відокремлені одне від одного, модератор або автоматизований засіб отримує неповну картину. У задачах попереднього аналізу зображень важливими є статистичні характеристики та логічна обробка структурного опису, що може бути використано як допоміжний сигнал для майбутнього розвитку системи [23–24]. У результаті схожі матеріали можуть отримувати різні рішення, а спірні випадки складно пояснювати після завершення перевірки.

Фото здебільшого потребує уважного огляду одного або кількох кадрів. Основні труднощі виникають тоді, коли значення матеріалу залежить від дрібних деталей, символіки, культурного контексту або підпису. Відео потребує іншого темпу роботи: модератор має переглянути послідовність сцен, повернутися до фрагмента, зіставити візуальний ряд з описом або звуковою доріжкою. Через це відеоконтент створює вищий ризик пропуску прихованого порушення, ніж статичне зображення.

Окремою групою є матеріали, що потребують повторного розгляду. Такий стан не обов’язково пов’язаний з типом файла. Фото може бути спірним через

підпис або походження, відео – через короткий неоднозначний фрагмент, текст – через контекст цитування. Повторний розгляд потрібний тоді, коли первинне рішення не дає достатньої впевненості або коли матеріал повинен пройти додаткову перевірку через чутливість наслідків.

Дані, які впливають на оцінювання одиниці контенту, подано на рисунку 1.2. Вони показують, що модерація спирається на сукупність ознак, а не лише на вміст файла. Для автоматизованої попередньої оцінки зображень релевантними є підходи з модифікацією простору даних, нечіткою кластеризацією та непараметричним статистичним аналізом структурних компонентів [25–27].

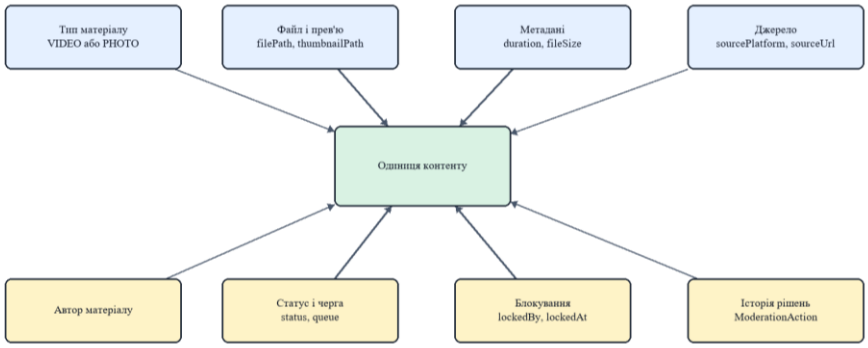

Рисунок 1.2 – Дані, які враховуються під час модерації одиниці контенту

Різниця між текстом, фото і відео має практичне значення для організації процесу. Потік відео потребує більше часу на один матеріал, потік фото потребує швидкого візуального огляду, а повторний розгляд має бути виділений як окремий процедурний стан. Для подальшого розвитку такого контуру можна враховувати методи пошуку об’єктів у відеопотоці та класифікації зображень за локальними дескрипторами, однак у межах цієї роботи вони залишаються допоміжними, а не основним механізмом ухвалення рішення [28–29]. Такий поділ не означає, що кожен тип контенту потребує ізольованої системи; він означає, що програмна модель повинна зберігати достатній контекст для різних сценаріїв перевірки.

У практичній системі ці ознаки не можна розглядати ізольовано від часу опрацювання. Текстовий матеріал часто дає змогу швидко встановити

порушення за ключовими словами або явними формулюваннями, однак фото і відео потребують іншого темпу перегляду. Модератор має зупинятися на деталях, перевіряти взаємозв'язок з підписом, за потреби повертатися до фрагмента відео і зіставляти зміст із правилами платформи. Якщо інтерфейс не підтримує таку роботу, помилки виникають не лише через людський чинник, а й через недосконалу організацію самого процесу.

Для фото і відео важливими є також метадані, які не завжди прямо належать до вмісту файла. Один і той самий кадр може мати різну оцінку залежно від автора, попереднього статусу, скарг, опису або черги, з якої матеріал потрапив до модератора. Тому програмна система має подавати медіафайл разом із супровідними даними, а не перетворювати модерацію на простий перегляд зображення чи відеоплеєра. У разі подальшого розширення корисними можуть бути методи структурного розпізнавання, ієрархічних ознак і перетворення описів зображень у класифікаційні ознаки [30–32]. Саме поєднання вмісту, контексту і процедурного стану дає змогу ухвалювати відтворювані рішення.

Окремої уваги потребує стан повторного розгляду. Він виникає тоді, коли первинна перевірка не дала достатньої визначеності або коли застосована політика потребує додаткового підтвердження. Для соціальної платформи це не виняткова ситуація, а нормальний елемент контрольованого процесу. Якщо повторний розгляд не відокремлено від первинної черги, матеріал може губитися серед нових надходжень або повторно видаватися без урахування попереднього рішення. Тому на рівні предметної області доцільно виділяти не тільки тип контенту, а й процесуальний статус матеріалу.

1.3 Ручна, автоматизована, гібридна, реактивна і проактивна модерація як прикладні моделі організації процесу

Моделі модерації можна поділити за двома ознаками. Перша ознака стосується суб'єкта оцінювання: рішення ухвалює людина, автоматизований

механізм або їх поєднання. Друга ознака стосується моменту втручання: матеріал перевіряється до поширення, після скарги або після внутрішнього сигналу. Такий поділ важливий, бо термін «модерація» часто охоплює різні за суттю процеси.

Ручна модерація означає, що змістовне рішення ухвалює людина. Її перевага полягає в здатності враховувати контекст, винятки, намір, культурні ознаки та неоднозначні ситуації [4], [5]. Водночас ручна робота повільніша, залежить від якості інструкцій і може давати нерівномірні результати між різними модераторами. Без формалізованого маршруту така модель швидко стикається з дублюванням рішень, чергами невизначеного стану і слабкою простежуваністю.

Автоматизована модерація спирається на алгоритмічне виявлення, фільтрацію, класифікацію або пріоритизацію матеріалів. Її сильна сторона – масштабованість, швидкість первинної обробки і здатність працювати з великими потоками даних [17], [19], [21]. Слабкі сторони пов’язані з пояснюваністю, помилковими спрацюваннями, зміщенням у навчальних даних і складністю оцінювання контексту [3], [18]. Автоматичне рішення може бути швидким, але не завжди достатньо обґрунтованим для спірних мультимедійних випадків.

Гібридна модерація поєднує машинний сигнал із людським рішенням. У такій моделі автоматизований засіб може попередньо визначати ризик, сортувати матеріали, виділяти підозрілі фрагменти або пропонувати варіанти категорій, але остаточне рішення лишається за людиною. Цей підхід зменшує навантаження на модераторів, однак не усуває потреби в прозорих правилах, журналі дій і можливості повторного розгляду [4], [11].

Реактивна модерація починається після сигналу: скарги, повідомлення, ескалації або внутрішнього маркера ризику [17]. Вона добре відповідає середовищам, де весь матеріал неможливо перевірити до публікації. Проактивна модерація працює до широкого поширення або до остаточного допуску матеріалу [19]. Її перевага полягає у зменшенні ризику появи порушення перед

великою аудиторією, але така модель потребує більше ресурсів і чіткішого попереднього відбору.

Порівняння моделей не дає абсолютної переваги одній з них. Ручна модерація краще працює з контекстом, але гірше масштабується. Автоматизована модерація швидка, проте потребує контролю помилок. Гібридна модель зменшує навантаження, але ускладнює відповідальність між алгоритмом і людиною. Реактивний підхід дешевший для потоку, але допускає появу проблемного матеріалу до втручання. Проактивний підхід краще запобігає поширенню порушень, однак може створювати затримки та вимагати значних ресурсів.

Участь людини в контурі модерації не суперечить автоматизації. Суперечність полягає в іншому: яку саме частину процесу потрібно автоматизувати. Можна автоматизувати змістовний вердикт, попереднє ранжування ризику, маршрутизацію матеріалів, фіксацію підстав рішення або статистичний контроль виконання. Для програмного забезпечення, орієнтованого на ручну модерацію, найбільш важливою стає автоматизація маршруту і контролю, а не заміна людського судження.

Окремої уваги потребує питання категорій порушень. У практиці соціальних платформ модераційні правила зазвичай охоплюють насильство, мову ворожнечі, сексуалізований контент, небезпечні дії, самопошкодження, шахрайство, спам, порушення авторських прав і матеріали, небезпечні для неповнолітніх [6–9]. Ці категорії не є рівнозначними за способом оцінювання. Частина з них краще піддається формальним ознакам, наприклад повторюваний спам або явно заборонені зображення. Інші потребують ширшого контексту: політична сатира, документальні кадри насильства, художнє зображення або цитування образливого висловлювання можуть бути помилково витлумачені без урахування ситуації.

Звідси випливає важлива методична вимога: правила модерації мають бути не лише переліком заборон, а й системою наслідків. Для одного матеріалу достатньо обмеження видимості, для іншого потрібне блокування, а для третього

– повторний розгляд. Бінарна модель «дозволити або видалити» спрощує роботу, але погано описує реальні випадки, де порушення залежить від віку аудиторії, контексту публікації, повторюваності дій автора або суспільної значущості матеріалу.

Операційна якість модерації також не зводиться до точності одного рішення. Важливими є швидкість отримання матеріалу на перевірку, час перебування в черзі, частка повторних переглядів, кількість рішень за категоріями, навантаження на виконавців і наявність спірних випадків. Такі показники не замінюють змістовної оцінки, але дозволяють побачити, чи не накопичуються матеріали в окремому потоці, чи не перевантажений один тип контенту, чи не виникає надмірна кількість повернень на повторний розгляд.

У літературі про автоматизовану модерацію часто наголошується на точності класифікації, однак для ручного або гібридного процесу не менш важливою є відтворюваність процедури [17–21]. Якщо два подібні матеріали проходять різні маршрути без пояснення, навіть правильний фінальний статус не створює довіри до процесу. Тому програмна підтримка модерації повинна зберігати не лише кінцеве рішення, а й послідовність дій, у межах якої це рішення було отримано.

Порівняння моделей модерації показує, що для розроблюваної системи ключовою є не заміна людини автоматичним класифікатором, а впорядкування ручного процесу. Автоматизовані засоби можуть бути корисними для попереднього сортування, оцінки ризику або виявлення очевидних повторів, проте остаточне рішення щодо спірного мультимедійного матеріалу часто потребує контекстного тлумачення. Через це система повинна підтримувати роботу модератора так, щоб технічний інтерфейс не приховував підстав рішення і не ускладнював подальшу перевірку.

Гібридна модель є найбільш придатною для поступового розвитку такого застосунку. На першому етапі вона може спиратися переважно на ручну модерацію, але вже містити формалізовані черги, політики, ролі та журнал дій. На наступних етапах до цього процесу можна додавати автоматичні індикатори

ризику, не змінюючи базову логіку відповідальності: система пропонує або структурує інформацію, а рішення фіксується як дія конкретного уповноваженого користувача. Такий підхід зменшує залежність від непрозорих автоматичних оцінок і залишає простір для нормоконтролю, аудиту та навчального аналізу.

Реактивна і проактивна модерація також не є взаємовиключними. Реактивний підхід потрібний для опрацювання матеріалів, на які вже надійшли скарги або які були позначені як проблемні. Проактивний підхід потрібний для первинного перегляду черг, де матеріал ще не спричинив конфлікту, але може порушувати правила. У межах навчального вебзастосунку ці підходи відображаються через розподіл контенту між чергами VIDEO, PHOTO і REVIEW, а також через можливість перевести спірний матеріал на повторну перевірку.

1.4 Прозорість, повторний розгляд і процедурна справедливість

Якість системи модерації не можна оцінювати лише за швидкістю обробки матеріалів. Якщо після рішення неможливо встановити, хто його ухвалив, на підставі якого правила, який стан був до перевірки і який став після неї, процес лишається слабко придатним для контролю. У літературі це пов’язують з вимогами прозорості, підзвітності, можливості перегляду та розподілу відповідальності [10], [12–14].

Прозорість у прикладному сенсі означає, що рішення пояснюється не лише вільним коментарем модератора, а й даними процесу. Для цього потрібні принаймні відомості про матеріал, особу або роль виконавця, час дії, попередній і новий стан, застосоване правило та текстове обґрунтування. Якщо хоча б один з цих елементів відсутній, подальший аналіз рішення стає неповним.

Повторний розгляд потрібний для випадків, у яких негайне схвалення або блокування є надто грубим рішенням. Це можуть бути неоднозначні

відеофрагменти, матеріали з чутливим контекстом, публікації з можливими ознаками сатири або ситуації, де треба порівняти матеріал з додатковими правилами. Повторний розгляд не є ознакою помилки первинної модерації; він є окремим процедурним механізмом для складних випадків.

Процедурна справедливість у модерації означає передбачуваність правил, можливість пояснення наслідку і наявність каналу для перегляду неоднозначного рішення. Для соціальних мереж це важливо через масштаб впливу модерації: рішення щодо одного матеріалу може зачіпати видимість автора, доступ аудиторії до інформації і довіру до самої платформи. Саме тому модераційна система повинна підтримувати не тільки дію, а й її доказовий слід.

Аудитоспроможність означає придатність процесу до подальшої перевірки. Якщо історії переходів немає, перевірка перетворюється на реконструкцію з пам'яті або з окремих зовнішніх повідомлень. Натомість журнал дій, зв'язок рішення з правилом, збереження попереднього та нового стану створюють основу для адміністративного контролю, статистичного аналізу і виявлення повторюваних проблем.

Звідси формуються процедурні вимоги до програмного забезпечення модерації. Воно має підтримувати контроль доступу, простежуваність рішень, відокремлення первинної перевірки від повторного розгляду, фіксацію підстав, керування правилами і статистичний контроль. Ці вимоги належать до предметної області загалом; конкретний спосіб їх реалізації залежить від архітектури системи і розглядається в наступних розділах.

Практична складність процедурної справедливості полягає в тому, що користувачі й адміністратори оцінюють процес з різних позицій. Для користувача важливо, щоб рішення не виглядало випадковим і щоб складний випадок міг бути переглянутий. Для адміністратора важливо бачити навантаження, причини типових рішень і повторювані проблеми в правилах. Для модератора важливо мати стабільний порядок роботи, достатній контекст і зрозумілий спосіб фіксації підстави. Якісна модераційна система повинна поєднувати ці очікування в одному процесі.

Прозорість не означає повного розкриття внутрішніх правил кожному користувачеві, адже це може створити можливості для обходу політик. Водночас внутрішня непрозорість також небезпечна: якщо організація не може відтворити власне рішення, вона втрачає здатність навчати модераторів, виправляти помилки й доводити сталість процедур. Тому достатній рівень прозорості полягає у збереженні структурованих даних про дію, правило, наслідок і час виконання.

Ще одним аспектом є межа між модерацією і цензуруванням. Модерація в межах соціальної платформи спрямована на підтримання правил середовища, захист користувачів і керованість потоків контенту. Проте без процедурного контролю вона може сприйматися як довільне вилучення матеріалів. Саме тому важливими стають повторний розгляд, історія дій, формалізовані підстави і розмежування ролей між виконавцем рішення та адміністративним рівнем контролю.

1.5 Узагальнення вимог предметної області до організації модерації

Аналіз предметної області показує, що система модерації повинна працювати не як набір розрізнених екранів або списків, а як керований процес. Першою вимогою є наявність зрозумілого маршруту матеріалу. Користувацький контент має переходити між станами за визначеними правилами: очікування перевірки, робота модератора, ухвалення рішення, повторний розгляд або завершення обробки.

Другою вимогою є контроль доступу. У процесі модерації різні учасники не повинні мати однакові повноваження. Модератор працює з матеріалом і фіксує рішення, тоді як адміністративний рівень відповідає за правила, користувачів, статистичний огляд і контроль якості процесу. Такий поділ зменшує ризик довільної зміни правил під час робочої перевірки.

Третьою вимогою є недопущення дублювання роботи. Якщо один матеріал одночасно потрапляє до кількох виконавців, виникає ризик суперечливих рішень, втрати часу і нечіткої історії обробки. Тому процес повинен містити механізм тимчасового закріплення матеріалу за виконавцем або інший спосіб уникнення паралельної роботи з одним і тим самим випадком.

Четверта вимога стосується зв'язку рішення з правилом. Модераційна дія не повинна бути лише кнопкою без пояснення. Вона має бути прив'язана до політики, категорії порушення або іншої формалізованої підстави. Це дає змогу відрізнити випадкове рішення від рішення, ухваленого за визначеним правилом.

П'ятою вимогою є збереження історії. Для адміністративного контролю важливо знати не тільки актуальний стан матеріалу, а й послідовність дій: хто виконав перевірку, коли вона відбулася, який був попередній стан, який наслідок отримано і яка підстава зазначена. Без такої історії неможливо якісно аналізувати помилки, повторні звернення або навантаження на процес.

Шостою вимогою є підтримка повторного розгляду. Частина мультимедійних матеріалів має спірний характер, тому система повинна дозволяти відокремити такі випадки від звичайного потоку. Повторний розгляд не замінює первинну модерацію, а доповнює її там, де рішення потребує більшої обережності.

Узагальнення цих вимог створює перехід від теоретичного аналізу до постановки задачі. Предметна область вимагає не автоматичного розпізнавання всіх порушень, а впорядкування ручної роботи з мультимедійним контентом, фіксації рішень, рольового розмежування і прозорого маршруту обробки.

Для подальшого проєктування важливо, що наведені вимоги мають різний рівень абстракції. Частина з них описує предметну логіку: контент має перебувати в певному стані, рішення повинно мати підставу, а повторний розгляд має бути виділений як окремий сценарій. Інша частина описує організацію доступу: модератор не повинен виконувати адміністративні операції, а адміністратор має бачити агреговану картину без втручання в кожну

окрему дію. Третя частина стосується якості даних: історія рішень має зберігати достатньо відомостей для аналізу, навчання і виправлення помилок.

Усі ці вимоги пов’язані між собою. Неможливо забезпечити прозору історію без формалізованої моделі даних; неможливо уникнути дублювання роботи без механізму закріплення матеріалу; неможливо пояснити наслідок рішення без політик модерації та визначених ефектів. Тому постановка задачі не зводиться до створення кількох сторінок інтерфейсу. Вона передбачає побудову цілісного маршруту обробки, у якому кожна дія користувача має перевірений доступ, зберігається в базі даних і змінює стан матеріалу за зрозумілими правилами.

З погляду предметної області найбільш ризиковими є місця переходу між станами. Саме в них можуть виникати суперечності: матеріал одночасно відкритий кількома модераторами, рішення не пов’язане з політикою, повторний розгляд не відображений у черзі або історія не містить достатньої інформації про попередній стан. Тому подальші розділи мають показати не тільки вибір технологій, а й те, як ці технології підтримують процесуальну цілісність модерації.

1.6 Постановка задачі

Актуальність задачі зумовлена тим, що ручна модерація мультимедійного контенту в соціальних мережах потребує не тільки перегляду окремих фото або відео, а й керованої процедури обробки. Для соціальних платформ характерний безперервний потік матеріалів з різними типами медіа, джерелами, підписами та рівнем контекстної неоднозначності. Без програмної підтримки така робота швидко перетворюється на неструктурований перегляд записів, де складно контролювати черговість, відповідального модератора, причину рішення і подальший стан матеріалу.

Якщо такий потік не має керованого маршруту, виникають повторні перевірки одного й того самого матеріалу, нерівномірне навантаження на модераторів, втрати даних про підстави рішення та складність подальшого адміністративного контролю.

У межах цієї роботи аналізується процес обробки фото і відеоматеріалів від моменту їх потрапляння до системи до ухвалення рішення модератором, фіксації результату та, за потреби, повторного розгляду. Оптимізації потребує не змістовне судження модератора як таке, а маршрут його роботи: розподіл матеріалів за чергами, недопущення одночасної обробки одного випадку кількома людьми, швидкий доступ до релевантних даних, контроль ролей і збереження історії рішень.

Об’єктом роботи є процес модерації контенту в соціальних мережах.

Метою роботи є підвищення ефективності та контрольованості ручної модерації мультимедійного контенту шляхом розробки вебзастосунку з чергами обробки, рольовим доступом та аудитом рішень.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати особливості модерації фото і відеоконтенту в соціальних мережах;

– розглянути ручну, автоматизовану, гібридну, реактивну та проактивну моделі модерації;

– визначити процедурні вимоги до системи модерації, пов’язані з контролем доступу, простежуваністю, повторним розглядом і журналом рішень;

– сформувати функціональні та нефункціональні вимоги до вебзастосунку VideoModerationTool;

– спроєктувати модель даних для користувачів, контенту, черг, політик модерації та історії рішень;

– реалізувати серверну частину, клієнтський інтерфейс і локальне середовище запуску системи;

– перевірити роботу основних сценаріїв модератора та адміністратора на демонстраційних даних.

2 ПРОЄКТУВАННЯ ТА ТЕХНОЛОГІЧНЕ ОБҐРУНТУВАННЯ СИСТЕМИ VIDEOMODERATIONTOOL

2.1 Архітектурна концепція системи

Проєктування VideoModerationTool спирається на поділ системи на чотири логічні рівні: користувацький інтерфейс, серверний прикладний рівень, рівень даних і файлове сховище медіаматеріалів. Такий поділ є доцільним для системи модерації, оскільки перегляд матеріалу, ухвалення рішення, збереження історії та адміністративний контроль мають різну природу і не повинні змішуватися в одному місці [17–18, 22].

Користувацький інтерфейс призначений для щоденної роботи модератора й адміністратора. Він показує доступні черги, відкриває матеріал для перегляду, надає форму прийняття рішення, відображає історію та статистичні дані. При цьому інтерфейс не повинен самостійно визначати остаточний статус контенту. Його роль полягає у зручному поданні даних і передаванні наміру користувача на серверний рівень.

Серверний прикладний рівень виконує операції, які впливають на цілісність модераційного процесу. До них належать перевірка доступу, видача наступного елемента з черги, тимчасове закріплення матеріалу за модератором, застосування політик, зміна статусу, створення запису історії та формування адміністративних агрегатів. Саме цей рівень повинен залишатися джерелом правил, тому що клієнтський застосунок може бути перезавантажений, змінений або виконаний у різних браузерах.

Рівень даних зберігає структуровану частину предметної області: користувачів системи, авторів матеріалів, одиниці контенту, статуси, черги, політики та журнал дій. Окреме файлове сховище використовується для самих фото і відеоматеріалів, тоді як база даних містить посилання на ці файли та їхній процесуальний стан. Такий підхід не перевантажує реляційну модель великими

двійковими об’єктами і водночас дозволяє пов’язати файл із рішеннями модераторів.

На верхньому рівні система складається з таких компонентів:

– клієнтського рівня, який відповідає за сторінки входу, черги, робоче місце модератора, історію та адміністративні екрани; – серверного API, який приймає запити, перевіряє права доступу і виконує предметну логіку; – реляційної бази даних, у якій формалізовано сутності, стани та зв’язки; – файлового сховища, у якому розміщуються демонстраційні медіаматеріали; – відтворюваного середовища запуску, яке дозволяє підняти всі складові в узгодженому стані.

Архітектура розмежовує візуальну взаємодію і зміну критичних станів. Інтерфейс ініціює дію та показує результат, але рішення про статус матеріалу, перевірка прав доступу і журналювання залишаються на сервері. Надмірна самостійність клієнта у зміні станів ускладнила б контроль цілісності процесу і збільшила ризик обхідних сценаріїв.

Модульність у цій системі має не лише технічний, а й предметний зміст. Окремо розглядаються автентифікація, черги, модераційні рішення, історія, політики, статистика, користувачі та контент. Завдяки цьому зміна логіки черг не вимагає одночасної зміни адміністративної статистики, а модифікація політик не повинна впливати на механізм входу.

На рисунку 2.1 зображено загальну архітектуру VideoModerationTool, яка відображає взаємодію клієнтської частини, серверного API, бази даних і файлового сховища медіаматеріалів.

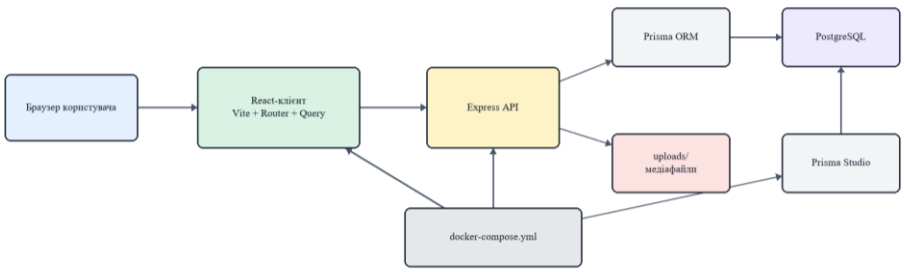

Рисунок 2.1 – Загальна архітектура VideoModerationTool

Архітектурна концепція системи пов’язана з вимогами, сформованими у попередньому розділі. Потреба в прозорості й аудитоспроможності зумовлює наявність окремої моделі журналу рішень. Потреба у поділі ролей пов’язана з моделлю користувачів і проміжною перевіркою авторизації. Потреба у повторній перевірці відображена у формалізованій черзі повторного розгляду. Потреба у відтворюваності забезпечується контейнеризованим підходом до локального запуску.

Такий поділ також полегшує подальше розширення системи. Якщо в майбутньому потрібно буде додати зовнішнє джерело контенту, це насамперед вплине на шар отримання та зберігання матеріалів. Якщо знадобиться допоміжна автоматична оцінка ризику, її можна додати як окремий сервісний компонент, не змінюючи базову логіку ручного рішення. Якщо потрібно буде розширити аналітику, вона може спиратися на вже наявний журнал дій.

Під час формування архітектури окремо враховано межу між прикладною логікою і презентаційним рівнем. Для системи модерації ця межа має принципове значення, тому що помилка в інтерфейсі не повинна створювати некоректний стан у базі даних. Наприклад, якщо модератор випадково повторно натиснув кнопку ухвалення рішення або оновив сторінку після отримання матеріалу, серверна частина має перевірити актуальний стан елемента, права користувача і наявність закріплення. Клієнт може допомогти уникнути випадкових дій, однак остаточна перевірка повинна залишатися на сервері.

Архітектурна концепція також враховує різну тривалість операцій. Перегляд фото може бути коротким, тоді як робота з відео потребує більше часу. Через це система не повинна розглядати відкриття матеріалу як миттєву дію без стану. Закріплення елемента за модератором дає змогу відобразити, що матеріал уже перебуває в роботі, а не просто існує в загальній черзі. Такий підхід поєднує інтерфейсну зручність із вимогою недопущення паралельного опрацювання одного й того самого матеріалу.

Важливою архітектурною вимогою є відтворюваність демонстраційного середовища. Для кваліфікаційної роботи система має бути придатною не лише

для опису, а й для перевірки основних сценаріїв на підготовлених даних. Тому архітектура передбачає узгоджену роботу клієнта, сервера, бази даних і файлового сховища. Якщо один із цих компонентів відсутній або має непогоджені дані, модераційний процес втрачає цілісність: у черзі можуть бути записи без файлів, політики без ефектів або історія без користувача, який виконав дію.

Окремо виділено адміністративний контур. Він не є простим розширенням робочого місця модератора, оскільки має іншу мету: не ухвалення рішення щодо одного матеріалу, а підтримання довідників, користувачів, політик і статистичного огляду. У проєктуванні це означає, що адміністративні сторінки повинні спиратися на ті самі дані, але надавати інший рівень узагальнення. Такий поділ зменшує ризик змішування оперативної модерації з налаштуванням правил системи.

2.2 Обґрунтування технологій клієнтської частини

Клієнтська частина проєкту має підтримувати не одну форму, а кілька взаємопов’язаних робочих сценаріїв: вхід користувача, вибір черги, перегляд матеріалу, застосування політики, перегляд історії та адміністративне керування. Через це доцільним є компонентний підхід, за якого інтерфейс складається з незалежних, але узгоджених частин. React відповідає цій вимозі, оскільки дозволяє будувати повторно використовувані елементи сторінок, таблиць, форм, статусних позначок і модальних вікон.

Вибір односторінкового вебзастосунку пояснюється характером роботи модератора. У процесі перевірки користувач не повинен переходити між повністю незалежними HTML-сторінками та втрачати стан інтерфейсу. Навігація між панеллю черг, робочим місцем модератора, історією та адміністративними екранами має сприйматися як єдиний робочий простір. Такий

підхід зменшує кількість зайвих переходів і підтримує серійне опрацювання матеріалів.

Для локального запуску клієнтської частини обрано інструмент швидкого складання. У контексті кваліфікаційної роботи це важливо не стільки через продуктивність самого інструмента, скільки через відтворюваність демонстрації. Застосунок повинен швидко запускатися під час перевірки, а процес складання не має вимагати складної ручної конфігурації.

Навігаційний рівень потрібний для розмежування доступних частин інтерфейсу. Після входу користувач переходить до робочого простору, але набір доступних сторінок залежить від ролі. Для модератора основними є черги, перегляд матеріалів та історія. Для адміністратора додаються статистика, користувачі, політики і керування контентом. Клієнтське приховування сторінок не замінює серверну авторизацію, але робить інтерфейс зрозумілішим і зменшує кількість помилкових дій.

Окрему роль відіграє керування серверним станом. У системі модерації багато даних змінюється після дій користувача: кількість доступних елементів у чергах, поточний матеріал, перелік активних політик, журнал рішень, адміністративні показники. Якщо кожний компонент самостійно керуватиме запитами й кешем, інтерфейс швидко стане суперечливим. Тому доцільним є використання окремого механізму, який кешує відповіді, оновлює їх після мутацій і дозволяє синхронізувати екран із сервером.

HTTP-клієнт у такій архітектурі повинен централізувати базову адресу API, передавання токена, обробку помилок і формування запитів. Це зменшує кількість випадкових рядків адрес у компонентах і дозволяє прив’язати клієнтський код до предметних операцій: отримання черг, отримання наступного матеріалу, застосування політики, перегляд історії, отримання статистики.

Бібліотека готових інтерфейсних компонентів використовується як засіб уніфікації форм, таблиць, карток, тегів, сповіщень і модальних вікон. Для системи модерації це практично важливо: користувач регулярно працює з табличними списками, формами рішень, фільтрами та статусами. Узгоджені

компоненти зменшують візуальну випадковість і залишають більше уваги на змісті модераційного процесу.

У таблиці 2.1 наведено основні бібліотеки клієнтської частини та пояснено їх роль у реалізації інтерфейсу системи.

Таблиця 2.1 – Бібліотеки клієнтської частини та їх роль

Бібліотека	**Роль у системі**	**Проєктне обґрунтування**
React	Компонентна побудова інтерфейсу	підтримка кількох пов’язаних робочих сценаріїв
Vite	Складання та локальний запуск	швидка перевірка і відтворюваність демонстрації
React Router	Навігація між сторінками	розмежування робочого й адміністративного контурів
TanStack Query	Керування серверним станом	оновлення черг, історії, політик і статистики після дій
Axios	HTTP-взаємодія з API	централізоване формування запитів і обробка токена
Ant Design	Базові UI-компоненти	форми, таблиці, теги, повідомлення та модальні вікна

Структура клієнтської частини проєктується як два робочі контури. Контур модератора починається з входу, переходить до панелі черг, відкриває робоче місце перегляду матеріалу і завершує дію записом в історії. Адміністративний

контур доповнює його статистикою, користувачами, політиками і загальним керуванням матеріалами. Обидва контури мають спільну стилістику і спільну модель автентифікації, але різні права доступу.

На рисунку 2.2 зображено структуру навігації клієнтського застосунку, яка показує основні переходи між сторінками модератора й адміністратора.

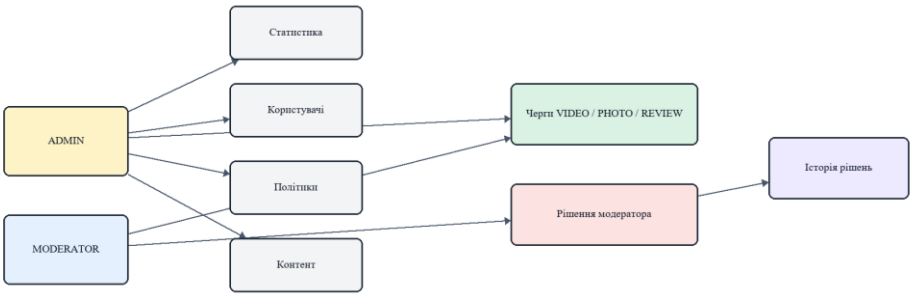

Рисунок 2.2 – Структура навігації клієнтського застосунку

Клієнтська частина має підтримувати не лише відображення даних, а й послідовність роботи користувача. Для модератора важливо швидко зрозуміти, у якій черзі він працює, який матеріал відкрито, які політики доступні і що станеться після вибору дії. Через це інтерфейс повинен бути побудований навколо повторюваних станів: завантаження списку черг, отримання наступного елемента, перегляд матеріалу, вибір рішення, підтвердження результату і перехід до наступної одиниці контенту.

Використання компонентного підходу дає змогу розділити ці стани на зрозумілі частини. Компонент черги відповідає за вибір робочого потоку, компонент перегляду матеріалу – за подання фото або відео, форма політик – за вибір підстав рішення, а сторінка історії – за контроль уже виконаних дій. Такий розподіл полегшує підтримку інтерфейсу, оскільки зміна відображення політики не повинна вимагати зміни всієї сторінки модерації.

Клієнтська частина також має враховувати помилки зв’язку і втрату актуальності стану. Якщо елемент черги вже заблоковано іншим користувачем або його статус змінився після відкриття сторінки, інтерфейс повинен отримати відповідну відповідь від сервера і не створювати враження, що дія виконана

успішно. Для цього обмін із API має бути побудований навколо чітких відповідей, а не лише навколо факту натискання кнопки. Повідомлення про помилки, порожню чергу або відсутність доступу є частиною модераційного процесу, а не другорядною деталлю інтерфейсу.

Для адміністратора клієнтська частина виконує іншу функцію. Вона має давати змогу переглядати користувачів, політики, статистику і контент у більш узагальненому вигляді. Адміністративний екран не повинен копіювати робоче місце модератора, оскільки його задача полягає в контролі системи, а не в послідовному перегляді кожного матеріалу. Саме тому в проєктуванні клієнтської частини враховано розмежування навігації, ролей і доступних дій.

2.3 Обґрунтування технологій серверної частини

Серверна частина системи має виконувати роль центру прийняття прикладних рішень. Вона приймає HTTP-запити від клієнта, перевіряє користувача, контролює роль, працює з базою даних, змінює статуси матеріалів і створює записи журналу. Для такої задачі обрано Express у поєднанні з TypeScript. Express дає просту модель маршрутів і проміжної обробки запитів, а TypeScript дозволяє формалізувати типи ролей, статусів, дій і структур відповідей.

Проміжна обробка запитів використовується як архітектурний механізм для повторюваних перевірок. Спочатку запит має пройти автентифікацію, тобто підтвердити особу користувача. Далі, для адміністративних операцій, виконується авторизація за роллю. Розділення цих перевірок важливе: користувач може бути коректно автентифікований, але не мати права змінювати політики або переглядати адміністративну статистику.

Для автентифікації застосовується токенний підхід. Після входу користувач отримує токен доступу, а приватні запити передають його серверу. Сервер відновлює з токена ідентичність користувача і застосовує подальші

правила доступу. Така схема добре підходить для вебзастосунку з окремими клієнтським і серверним рівнями, оскільки не вимагає зберігати стан сесії в пам’яті серверного процесу.

Реляційна база даних PostgreSQL обрана через потребу в чітких зв’язках між користувачами, авторами контенту, матеріалами, політиками і діями модераторів. Для модераційного процесу важлива не лише поточна картка матеріалу, а й історія зміни його стану. Реляційна модель природно підтримує такі зв’язки, а також дозволяє будувати фільтри й агреговану статистику.

Prisma ORM використовується як шар доступу до даних. ORM, або об’єктно-реляційне відображення, дає змогу працювати з таблицями через типізовані моделі прикладного коду. У цій роботі це зменшує ризик помилок у назвах полів, полегшує зміни схеми і робить зв’язок між моделлю даних та серверною логікою більш явним.

Логічна декомпозиція серверного рівня відповідає предметним підсистемам: автентифікація, черги, рішення модерації, історія, політики, статистика, користувачі та контент. Такий поділ не є просто організацією файлів. Він відображає різні групи відповідальності. Черги відповідають за маршрутизацію матеріалу, політики – за підстави рішення, історія – за простежуваність, статистика – за адміністративний огляд.

На рисунку 2.3 зображено модульну структуру серверної частини, що демонструє поділ API на логічні групи операцій.

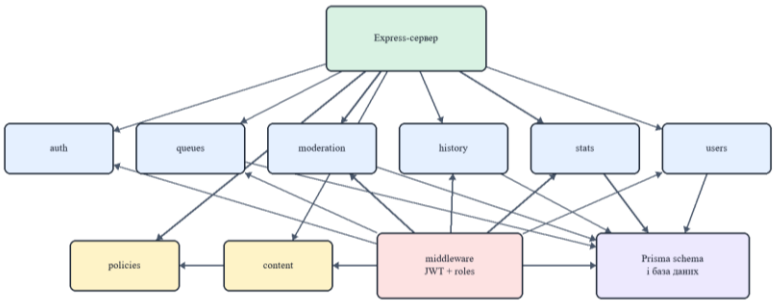

Рисунок 2.3 – Модульна структура серверної частини

У таблиці 2.2 наведено логічні групи серверних операцій VideoModerationTool та їх призначення в загальній роботі системи.

Таблиця 2.2 – Логічні групи серверних операцій VideoModerationTool

Група операцій	**Тип доступу**	**Що забезпечує**
Автентифікація	публічний або автентифікований	вхід, оновлення токена, отримання профілю
Черги	автентифікований	підрахунок, перегляд і видача матеріалів
Модераційні рішення	автентифікований	схвалення, застосування політик, пропуск
Історія	автентифікований	перегляд журналу рішень і фільтрація
Політики	автентифікований, частково адміністративний	перегляд і керування правилами
Статистика	адміністративний	агрегований огляд стану системи
Користувачі	адміністративний	керування обліковими записами та ролями
Контент	адміністративний або змішаний	підтримка матеріалів і повернення до черг

Використані серверні технології не розглядаються як самоціль. Їхня роль полягає в тому, щоб підтримати вимоги до керованості процесу: користувач повинен бути ідентифікований, дія повинна бути дозволена роллю, зміна статусу має виконуватися на сервері, а історія рішення має зберігатися разом із контекстом.

Серверна частина є місцем, де предметні правила перетворюються на перевірювані операції. Вона приймає запити від клієнта, але не повинна довіряти

їм без перевірки. Кожна дія модератора має проходити через авторизацію, перевірку ролі, перевірку стану матеріалу і визначення допустимого переходу. Така вимога випливає з самої природи модерації: користувацький інтерфейс може бути змінений або відкритий у кількох вкладках, але сервер має залишатися єдиним джерелом істини щодо стану процесу.

Для системи модерації важливим є розмежування маршрутів API за змістом операцій. Операції входу й оновлення токенів належать до автентифікації; отримання наступного елемента, пропуск і застосування політики – до робочого процесу модератора; керування політиками, користувачами і статистикою – до адміністративного контуру. Такий поділ полегшує перевірку прав доступу і робить структуру API ближчою до предметної області.

Сервер також відповідає за узгодження роботи з базою даних і файловим сховищем. Медіафайл зберігається як файл, однак його стан, тип, автор, черга, закріплення і рішення зберігаються в реляційній моделі. Тому серверна логіка має працювати з двома різними типами ресурсів: із файлами, які потрібно віддати для перегляду, і з транзакційними записами, які визначають процесуальний стан матеріалу. Помилка в одному з цих шарів не повинна непомітно руйнувати інший.

Під час проєктування серверної частини враховано, що частина операцій має бути атомарною з погляду предметного процесу. Наприклад, застосування політики не може складатися з незалежних кроків, де статус матеріалу змінено, а запис історії не створено. У такому разі система втратить пояснюваність рішення. Тому операції, які змінюють стан матеріалу, мають одночасно фіксувати відповідну дію в журналі і знімати або оновлювати робоче закріплення.

2.4 Проєктування моделі даних і журналювання рішень

Модель даних є основою системи модерації, оскільки саме вона відокремлює матеріал як об’єкт перевірки від рішення щодо нього, користувача, який це рішення ухвалив, і політик, на які це рішення спиралося. У проєктній моделі виділено внутрішнього користувача системи, автора контенту, одиницю контенту, політику модерації, дію модератора і зв’язок між дією та політикою.

Внутрішній користувач описує адміністратора або модератора. Для системи важливо зберігати не лише електронну адресу і роль, а й зв’язок цього користувача з виконаними діями. Без такого зв’язку історія рішень втратила б відповідального виконавця, а статистика за модераторами стала б неможливою.

Автор контенту представляє користувача умовної соціальної мережі, матеріали якого потрапляють до модерації. Він не є внутрішнім користувачем системи. Це розмежування потрібне для того, щоб не змішувати оператора модерації з джерелом контенту. Крім того, рішення щодо одного матеріалу можуть мати наслідки для ширшого профілю автора, наприклад у випадку повторюваних порушень.

Одиниця контенту є центральним об’єктом перевірки. Вона поєднує тип матеріалу, посилання на файл, автора, джерельні метадані, процесуальний статус, робочу чергу і ознаки тимчасового закріплення. Проєктна модель навмисно розділяє тип матеріалу і чергу. Тип відповідає природі файла, а черга описує робочий процес. Завдяки цьому відео або фото може бути не тільки на первинній перевірці, а й у повторному розгляді.

Політика модерації формалізує підставу рішення. Вона має назву, опис, ознаку активності та ефект. Наявність окремої сутності політики дозволяє не перетворювати правила на набір жорстко зашитих умов. Адміністратор може змінювати довідник політик, а історія рішень усе одно зберігає зв’язок із тими правилами, які було застосовано.

Дія модератора фіксує факт рішення. Для аудитоспроможності недостатньо зберігати лише поточний статус матеріалу. Потрібно знати

попередній статус, новий статус, тип дії, відповідального користувача, час і пояснювальну примітку. Якщо до рішення застосовано кілька політик, між дією і політиками використовується проміжний зв'язок. Це дозволяє зберегти не одну умовну причину, а повний набір правил, які стали підставою рішення.

На рисунку 2.4 зображено ER-діаграму бази даних VideoModerationTool, яка відображає основні сутності системи та зв'язки між ними.

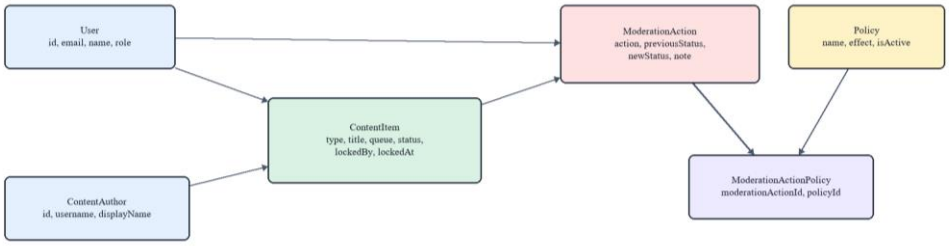

Рисунок 2.4 – ER-діаграма бази даних VideoModerationTool

У таблиці 2.3 наведено сутності моделі даних і пояснено їх призначення в процесі модерації контенту.

Таблиця 2.3 – Сутності моделі даних і їх призначення

Сутність	**Призначення**	**Ключові зв'язки**
1	**2**	**3**
Користувач системи	адміністратор або модератор	дії модерації, тимчасово закріплені матеріали
Автор контенту	джерело користувацьких матеріалів	одиниці контенту
Одиниця контенту	фото або відеоматеріал у процесі перевірки	автор, черга, статус, журнал дій
Політика	формальне правило модерації	дії, у межах яких її застосовано

Продовження таблиці 2.3

1	2	3
Дія модератора	запис про ухвалене рішення	матеріал, модератор, попередній і новий статус
Зв’язок дії і політики	фіксація застосованих правил	дія модератора, політика

Модель станів має обмежувати допустимі значення ролей, типів контенту, статусів, черг, ефектів політик і типів дій. Обмежений набір значень зменшує ризик випадкових текстових статусів і робить переходи перевірюваними. Для модераційної системи це особливо важливо, бо один помилковий статус може вивести матеріал із черги або зробити його невидимим для повторного розгляду.

Поля журналу рішень мають прикладне значення для аудиту. Ідентифікатор матеріалу показує, щодо якого контенту ухвалено рішення, ідентифікатор модератора визначає відповідального користувача, тип дії відокремлює схвалення, пропуск і застосування політики, попередній та новий статус показують перехід стану, примітка зберігає пояснення, а час дозволяє відновити хронологію. У такій моделі журнал не є допоміжним записом, а стає основою простежуваності всієї процедури.

Модель даних побудовано так, щоб вона відображала не лише об’єкти системи, а й зв’язки між ними. Користувач пов’язаний із діями, які він виконує; контент пов’язаний з автором, типом, статусом і шляхом до файла; політика задає допустимий наслідок; історія зберігає факт рішення. Саме зв’язки між цими сутностями роблять дані придатними для подальшого аналізу. Без них система мала б лише поточний стан матеріалу, але не могла б пояснити, як цей стан було отримано.

Для журналювання важливо зберігати не тільки фінальний результат, а й контекст переходу. Запис історії має показувати, яка дія виконана, ким вона виконана, до якого матеріалу належить, які політики застосовано і який новий

стан отримано. Це дає змогу аналізувати типові рішення модераторів, знаходити спірні випадки, оцінювати навантаження і перевіряти, чи відповідають дії встановленим правилам. Журнал у такій системі виконує не декоративну, а контрольну функцію.

Реляційна модель є доречною через потребу в цілісності зв'язків. Для модерації небажаною є ситуація, коли історія посилається на неіснуючий матеріал, політика видалена без урахування вже створених рішень або користувач має роль, не передбачену системою. Обмеження схеми, зв'язки між таблицями і типізований доступ через ORM допомагають зменшити кількість таких помилок. Водночас модель не повинна бути надмірно складною: для навчальної системи достатньо сутностей, які прямо підтримують черги, політики, ролі і журнал дій.

Окремим рішенням є збереження медіафайлів поза базою даних. Це зменшує навантаження на реляційне сховище і робить демонстраційний набір файлів простішим для перегляду й оновлення. У базі даних зберігається шлях до файла та службові відомості, а сам файл розміщується у файловому сховищі. Такий поділ відповідає характеру системи: база даних контролює процес модерації, а файловий рівень забезпечує доступ до мультимедійного вмісту.

2.5 Проєктування черг модерації, блокувань і повторної перевірки

Черги в системі проєктуються як прикладна модель маршрутизації матеріалів, а не як окремий зовнішній брокер повідомлень. Для навчального і демонстраційного проєкту це виправдано: модель лишається прозорою, не з'являється зайвий інфраструктурний компонент, але сценарії конкурентної роботи кількох модераторів усе одно можна перевірити.

Активний матеріал описується поєднанням робочої черги, процесуального статусу і ознак тимчасового закріплення. Черга визначає, у якому робочому потоці перебуває матеріал. Статус показує його поточний стан. Закріплення за

модератором потрібне для того, щоб один і той самий матеріал не був одночасно виданий двом користувачам.

Для первинної перевірки передбачено окремі черги фото і відеоматеріалів. Це відповідає різній природі об'єктів перегляду: фото оцінюється як статичний візуальний об'єкт, тоді як відео потребує перегляду тривалості, кадрів і можливого контексту. Черга повторної перевірки має інший зміст. Вона описує не тип файла, а процесуальний стан: матеріал уже був розглянутий, але потребує додаткової уваги.

Сервіс черг на рівні проєктної моделі виконує три групи задач. По-перше, він дає змогу підрахувати доступні та зайняті елементи в кожній черзі. По-друге, він повертає список матеріалів у межах обраної черги. По-третє, він реалізує вибір наступного елемента для конкретного модератора. Саме третя задача є найкритичнішою, оскільки від неї залежить відсутність конфліктів доступу.

Алгоритм отримання наступного елемента працює так:

Крок 1. Спочатку звільняються прострочені блокування, тобто матеріали, які були закріплені, але не завершені в межах допустимого часу.

Крок 2. Далі система перевіряє, чи є в поточній черзі елемент, уже закріплений за тим самим модератором.

Крок 3. Якщо такий елемент є, він повертається повторно, щоб не втрачався контекст роботи.

Крок 4. Якщо власного елемента немає, система шукає найстаріший доступний матеріал без активного закріплення.

Крок 5. Після знаходження такого матеріалу він закріплюється за модератором і повертається для перегляду.

Ця послідовність відрізняється від простого вибору першого доступного запису. Пріоритет власного закріпленого матеріалу потрібний для нормальної поведінки після оновлення сторінки або короткого переходу між розділами. Без такого правила модератор міг би втратити матеріал у роботі, хоча рішення щодо нього ще не ухвалено.

Для користувача внутрішні деталі блокувань не є видимою частиною роботи. Він очікує простішої поведінки: матеріал не зникає після оновлення сторінки, не відкривається одночасно в іншого модератора і не втрачається після короткого переходу між розділами. Саме цю поведінку забезпечує спроєктована логіка черг.

Окремої уваги потребує черга повторної перевірки. Її функція не зводиться до приховування контенту. Вона виокремлює матеріали, які потребують повторного або додаткового розгляду. Концептуально така черга відрізняється від первинних черг тим, що не описує тип файла. Вона описує стадію процесу.

На рисунку 2.5 зображено алгоритм отримання наступного елемента черги, який враховує активні блокування, повторне відкриття власного матеріалу та вибір доступного елемента.

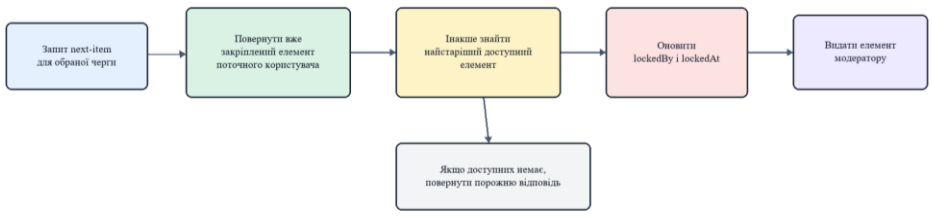

Рисунок 2.5 – Алгоритм отримання наступного елемента черги

У таблиці 2.4 наведено відповідність між статусами контенту та чергами, у яких матеріал може перебувати під час модерації.

Таблиця 2.4 – Відповідність між статусами контенту та чергами

Стан матеріалу	**Черга**	**Смисл**
1	**2**	**3**
очікує первинної перевірки, відео	черга відеоматеріалів	первинна перевірка відеоконтенту
очікує первинної перевірки, фото	черга фотоконтенту	первинна перевірка фотоконтенту

Продовження таблиці 2.4

1	2	3
потребує повторного розгляду	черга повторної перевірки	повторний або додатковий розгляд
схвалений	поза активними чергами	матеріал не потребує подальшої дії
заблокований	поза активними чергами	матеріал визнано неприйнятним
обмежений	поза активними чергами	матеріал отримав часткове обмеження

Сценарії блокування мають бути передбачуваними для користувача. Коли модератор уперше бере елемент, система закріплює його за цим користувачем і фіксує час початку роботи. Якщо сторінку оновлено, той самий користувач отримує вже закріплений матеріал повторно. Інший модератор у цей час має отримати інший доступний елемент. Прострочене блокування знімається автоматично, а після рішення або пропуску матеріал виходить із поточного закріплення.

Черга модерації у цій роботі розглядається як процесуальний механізм, а не як звичайний список записів. Її призначення полягає в тому, щоб видати модератору релевантний матеріал, не допустити одночасної роботи кількох користувачів з одним елементом і зберегти передбачуваний порядок обробки. Для цього кожен матеріал повинен мати стан, який дає змогу визначити, чи доступний він для видачі, чи вже перебуває в роботі, чи потребує повторного розгляду.

Механізм закріплення потрібний через реальний характер роботи з мультимедіа. Модератор може переглядати відео кілька хвилин, відволіктися, оновити сторінку або тимчасово втратити з’єднання. Якщо система не фіксує, що елемент уже видано конкретному користувачу, той самий матеріал може бути переданий іншому модератору. Це створює ризик подвійних рішень і

суперечливих записів історії. Закріплення розв'язує цю проблему, але воно має бути обмежене в часі, щоб матеріал не залишався заблокованим назавжди.

Термін дії блокування в 15 хвилин є компромісом між стабільністю роботи модератора і доступністю черги. З одного боку, цей час достатній для перегляду більшості демонстраційних фото і коротких відеоматеріалів. З іншого боку, якщо модератор припинив роботу або закрив сторінку, матеріал через певний час повертається до пулу доступних елементів. Такий підхід підтримує безперервність процесу і не потребує ручного адміністративного втручання для кожного завислого випадку.

Повторна перевірка виділена окремо, тому що вона має інший зміст, ніж первинна модерація. Матеріал у черзі REVIEW уже пройшов певний етап оцінювання або був визнаний спірним. Для такого матеріалу важливо бачити попередній контекст, оскільки повторне рішення не повинно прийматися як повністю новий випадок. Проєктування черг тому враховує не тільки тип файла, а й причину, з якої матеріал опинився у відповідному процесуальному стані.

У системі також потрібно визначити поведінку для порожніх черг. Порожня черга не є помилкою застосунку; це нормальний стан, коли доступних матеріалів певного типу немає або всі вони тимчасово закріплені за іншими користувачами. Інтерфейс і API мають відрізняти таку ситуацію від технічної помилки. Це важливо для коректної роботи модератора: він має розуміти, чи потрібно обрати іншу чергу, зачекати завершення блокування або повідомити адміністратора про відсутність тестових даних.

2.6 Проєктування політик модерації та результатів рішень

Політика в системі є формалізованим правилом, яке пов'язує рішення модератора з певною процедурною підставою. Спрощена логіка «схвалити / не схвалити» для такого процесу занадто бідна: частину матеріалів треба блокувати, частину – обмежувати для неповнолітніх, частину – повертати на повторний

розгляд. Тому політика повинна мати не тільки назву й опис, а й формальний наслідок.

Проєктна модель передбачає три базові типи наслідків політики. Повне блокування використовується для матеріалів, які не можуть залишатися доступними. Переведення на повторний розгляд застосовується тоді, коли рішення потребує додаткового контексту або перевірки іншим способом. Обмеження для неповнолітніх описує проміжний випадок, коли матеріал не вилучається повністю, але його доступність має бути звужена.

Особливість модераційного рішення полягає в тому, що до одного матеріалу може бути застосовано кілька політик. Наприклад, матеріал може одночасно містити ознаки вікового обмеження і підставу для повного блокування. У такій ситуації не можна покладатися на випадковий порядок вибору політик. Потрібне правило пріоритету, за яким підсумковий наслідок визначається найсуворішою політикою.

У спроєктованій моделі суворість наслідків упорядковується так: повне блокування має найвищий пріоритет, повторний розгляд є проміжним наслідком, обмеження для неповнолітніх має нижчий пріоритет. Це не означає, що нижчий пріоритет є менш важливим з погляду безпеки користувачів. Йдеться лише про правило вибору підсумкового стану, коли кілька наслідків конкурують між собою.

Сам процес прийняття рішення має три основні варіанти:

– схвалення матеріалу, після якого він виходить з активної черги; – застосування однієї або кількох політик, після чого визначається підсумковий статус; – пропуск матеріалу, коли статус не змінюється, але робоче закріплення знімається і факт взаємодії зберігається.

В усіх трьох випадках система не повинна обмежуватися зміною поточного стану матеріалу. Потрібно зберігати дію в журналі, інакше пізніше неможливо пояснити, чому матеріал має саме такий статус. При застосуванні політик журнал має зберігати також перелік правил, на які спиралося рішення. Так формується зв’язок між процедурною підставою і фактичним наслідком.

На рисунку 2.6 зображено процес застосування політики модерації, починаючи від вибору правила модератором і завершуючи зміною статусу матеріалу.

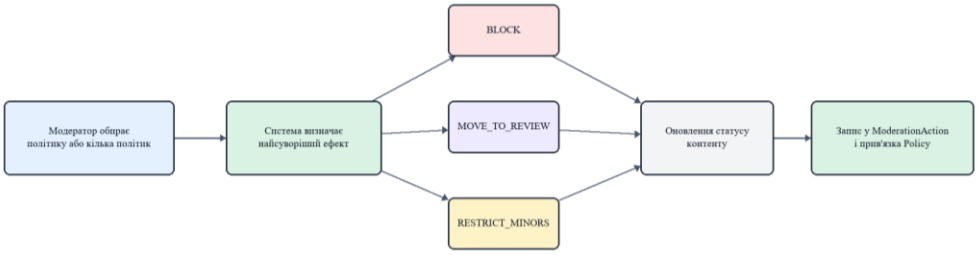

Рисунок 2.6 – Процес застосування політики модерації

У таблиці 2.5 наведено ефекти політик модерації та результат, який вони формують для одиниці контенту.

Таблиця 2.5 – Ефекти політик і результат для контенту

Ефект політики	**Підсумковий статус**	**Смисл для модераційного процесу**
блокування	заблокований	матеріал визнано неприйнятним
повторний розгляд	у повторній перевірці	потрібен додатковий розгляд
вікове обмеження	обмежений	матеріал не блокується повністю, але отримує обмеження

Тип дії модератора визначає, що саме змінюється в системі. Схвалення переводить матеріал до завершеного позитивного стану. Застосування політики визначає підсумковий статус за найсуворішим наслідком. Пропуск не змінює статус, але знімає робоче закріплення і залишає слід в історії. Така модель

відокремлює фактичний перегляд матеріалу від рішення, яке має процедурні наслідки.

Політика модерації формалізує підставу рішення. Без політик дія модератора була б лише ручною зміною статусу, яку складно порівнювати з іншими рішеннями. Політика дає змогу пов’язати рішення з конкретним правилом і наслідком: блокуванням, обмеженням для неповнолітніх або переміщенням на повторний розгляд. Це створює більш прозорий процес, у якому фінальний стан матеріалу має пояснення.

У проєктуванні політик важливо розмежувати назву, опис і ефект. Назва потрібна для швидкого вибору у формі, опис пояснює зміст правила, а ефект визначає, як система змінить статус матеріалу. Такий поділ дає змогу адміністратору уточнювати формулювання політик, не змінюючи програмну логіку наслідків. Водночас ефекти мають залишатися обмеженим набором, оскільки саме вони впливають на маршрут матеріалу.

Система повинна підтримувати застосування кількох політик одночасно, тому що один матеріал може порушувати кілька правил. Наприклад, відео може бути неприйнятним для всієї платформи і водночас містити ознаки, які за іншого контексту вимагали б лише вікового обмеження. Якщо такі політики обрано разом, підсумковий результат не повинен залежати від випадкового порядку їх вибору в інтерфейсі. Для цього визначено пріоритет суворості: блокування має перевагу над повторним розглядом, а повторний розгляд – над віковим обмеженням.

Таке правило суворості робить рішення передбачуваним. Модератор бачить набір застосованих політик, але система самостійно визначає найсуворіший наслідок і переводить матеріал у відповідний стан. Це зменшує кількість ручних помилок і забезпечує однакове трактування схожих випадків. Якщо в майбутньому набір політик розширюватиметься, доцільно зберегти цю логіку як окремий рівень визначення результату, а не розміщувати її лише в інтерфейсній формі.

Крім застосування політики, система має підтримувати схвалення і пропуск. Схвалення є позитивним завершенням перевірки, коли матеріал не потребує санкцій. Пропуск не повинен змінювати змістовний статус матеріалу, але має зняти робоче закріплення і зафіксувати факт дії. Це важливо, тому що пропуск також є частиною історії роботи модератора: він показує, що матеріал був виданий, але рішення щодо нього не було прийнято в межах цієї спроби.

2.7 Формалізація методу маршрутизації контенту та вибору результату модерації

Вибраний спосіб розв’язання задачі спирається на процедурну маршрутизацію одиниць мультимедійного контенту через черги, тимчасове закріплення, рольовий доступ, політики й журнал дій. Формально матеріал можна розглядати як об’єкт, що має тип, статус, чергу, автора, шлях до файла, додаткові метадані та ознаки тимчасового закріплення за модератором. Система змінює не сам медіафайл, а його процесуальний стан.

Формалізовано дві пов’язані задачі: вибір наступного матеріалу для модератора і визначення підсумкового статусу після рішення. Перша задача працює з множиною матеріалів, які мають активну чергу, статус і ознаки закріплення. Друга задача працює з множиною застосованих політик і впорядкованим набором їхніх наслідків. Такий поділ потрібний тому, що маршрутизація відповідає за організацію роботи, а вибір статусу – за наслідок рішення.

Маршрутизація елемента визначається парою «черга – статус». Для первинної перевірки фото та відео використовується стан очікування, а тип робочої черги відокремлює фотоматеріали від відеоматеріалів. Для повторного розгляду використовується окремий процесуальний стан. Такий підхід простіший за введення окремої таблиці завдань черги, але достатній для

навчальної системи, оскільки кожен матеріал має тільки один активний процесуальний стан.

Якщо в майбутньому система потребуватиме складної історії призначень або кількох паралельних команд, цю модель можна буде розширити окремою сутністю завдання. У межах цієї роботи таке ускладнення не потрібне, бо головною задачею є керований маршрут ручної модерації, а не повноцінна система розподілу навантаження між командами.

Алгоритм видачі наступного елемента спирається на пріоритет уже закріпленого матеріалу. Якщо модератор відкрив сторінку, отримав елемент і оновив браузер, система не повинна видавати йому інший матеріал, поки попередній залишається закріпленим. Тому перевірка власного закріпленого елемента виконується перед пошуком нового доступного матеріалу. Лише після цього застосовується вибір найстарішого незакріпленого запису.

Окремо формалізовано вибір результату при застосуванні кількох політик. Кожна політика має наслідок, а наслідки впорядковані за суворістю. Найсуворішим є блокування, далі йде повторний розгляд, найнижчий пріоритет має вікове обмеження. Якщо модератор вибрав кілька політик, підсумковий статус визначається не випадковим порядком їх передавання з інтерфейсу, а максимально суворим наслідком.

Метод організації процесу можна подати як послідовність станів. Спочатку матеріал перебуває у відповідній активній черзі. Потім серверний рівень закріплює його за модератором і фіксує час початку роботи. Після перегляду модератор виконує одну з дій: схвалення, застосування політики або пропуск. Система змінює статус, знімає закріплення, записує дію в журнал і, за потреби, пов’язує її із застосованими політиками.

У спрощеному вигляді метод маршрутизації можна описати так. Нехай C – множина матеріалів, q – обрана черга, u – поточний модератор, t – поточний час, а L – допустима тривалість закріплення. Спочатку для всіх елементів звільняються закріплення, для яких час початку роботи разом із допустимою тривалістю менший за поточний час. Далі виконується пошук елемента, де черга

дорівнює q, а відповідальним користувачем є u. Якщо такий елемент існує, він повертається без вибору нового матеріалу. Якщо власного елемента немає, система шукає найстаріший запис з чергою q, відповідним статусом і порожнім закріпленням, після чого призначає його користувачу u.

Вибір результату при застосуванні політик також можна подати як окрему функцію. Для кожного наслідку задається пріоритет: блокування має найбільшу суворість, повторний розгляд – проміжну, вікове обмеження – нижчу. Для множини обраних політик система знаходить максимальний за пріоритетом наслідок і перетворює його на статус матеріалу. Якщо політики не застосовуються, рішення може бути схваленням або пропуском, де перше змінює статус на схвалений, а друге залишає попередній статус без зміни.

Обраний метод оптимізує не автоматичне розпізнавання порушення, а керованість ручної модерації. Він зменшує дублювання роботи, робить маршрут обробки відтворюваним, забезпечує рольове розмежування і залишає дані для подальшого аналізу.

3 РЕАЛІЗАЦІЯ СИСТЕМИ МОДЕРАЦІЇ КОНТЕНТУ

3.1 Організація репозиторію та локального середовища запуску

Практична реалізація VideoModerationTool зосереджена в одному репозиторії, який містить код клієнта і сервера, схему БД, міграції, сценарії локального запуску та каталог медіафайлів. Усі суттєві артефакти системи знаходяться в одному контрольованому просторі, а запуск не вимагає ручного рознесення компонентів по різних середовищах.

Каталог client/ містить інтерфейс користувача, що побудований навколо сторінок входу, черг, модерації, історії та адміністративних екранів. Каталог server/ містить API, проміжну обробку запитів, бізнес-логіку та конфігурацію роботи з БД. У server/prisma/ зберігаються схема, міграції та сценарій наповнення демоданими. Каталог uploads/ містить тестові фото, відео та прев'ю, на які посилаються записи ContentItem. Така структура розмежовує код, дані, медіа і службові матеріали.

Локальне середовище запуску описано у docker-compose.yml. Воно складається з трьох сервісів: server, client і prisma-studio. Сервер публікується на порту 3001, клієнт – на 5173, Prisma Studio – на 5555. База PostgreSQL підключається через host.docker.internal, що в поточній конфігурації означає використання БД на хості розробника. Для демонстраційного та навчального сценарію такий підхід є прийнятним, оскільки спрощує середовище й дозволяє зосередитися на прикладній логіці.

Запуск командою docker compose up --build піднімає систему у відомій конфігурації: складається клієнт, запускається сервер, відкривається інтерфейс і лишається доступ до БД через Prisma Studio. Це важливо не через сам факт використання Docker, а через повторюваність демонстрації. Одна й та сама команда має приводити до одного й того самого робочого стану.

Під час практичної реалізації не можна було залишити клієнт, сервер і базу даних як три окремі демонстраційні частини. Docker Compose у цій роботі

використовується як спосіб зібрати їх в один робочий стан: сервер отримує змінні середовища, клієнт звертається до API, а Prisma Studio дає змогу перевірити, чи справді зміни статусів і журнал дій зберігаються в БД.

Працездатність у такому проєкті підтверджується не лише скріншотами інтерфейсу. Після модераційної дії має змінитися запис у даних.

Каталог uploads/ використовується як локальне сховище демонстраційних матеріалів. У реальній соціальній мережі файли могли б надходити із зовнішнього сховища або платформи, але в межах цієї роботи локальний каталог дає змогу стабільно відтворювати сценарії з фото, відео та прев'ю. Записи в таблиці ContentItem зберігають шляхи до цих файлів, а клієнт формує URL для відображення матеріалу в браузері. Така схема дозволяє перевірити основну логіку модерації без підключення до зовнішнього API, що зменшує залежність демонстрації від мережі або сторонніх сервісів.

На рисунку 3.1 зображено структуру репозиторію та середовища розгортання VideoModerationTool, що показує розподіл клієнтської, серверної частини й допоміжних файлів проєкту.

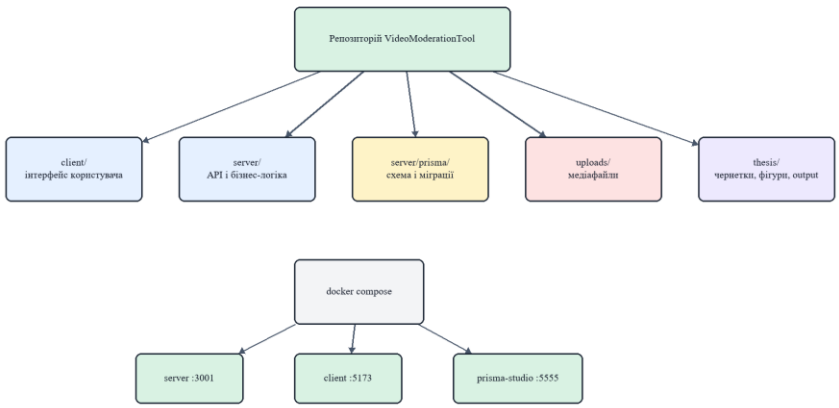

Рисунок 3.1 – Структура репозиторію та середовища розгортання VideoModerationTool

У таблиці 3.1 наведено сервіси Docker Compose та пояснено їх призначення під час локального запуску системи.

Таблиця 3.1 – Сервіси Docker Compose та їх призначення

Сервіс	**Порт**	**Призначення**	**Практична роль**
`server`	3001	Express API	автентифікація, черги, модерація, історія, політики, статистика
`client`	5173	React-клієнт	інтерфейс модератора та адміністратора
`prisma-studio`	5555	Prisma Studio	сервісний перегляд даних БД

3.2 Реалізація автентифікації, авторизації та ролей користувачів

Першою точкою взаємодії користувача із системою є сторінка входу LoginPage.tsx. Вона реалізує форму для введення електронної пошти та пароля, після чого надсилає запит на /api/auth/login. Серверний модуль auth перевіряє облікові дані, формує токен доступу і токен оновлення, а клієнт зберігає результат у власному контексті автентифікації. На практиці це можна простежити в client/src/hooks/useAuth.tsx: після входу токени записуються до localStorage, а при повторному відкритті сторінки виконується authApi.getMe(), щоб відновити дані поточного користувача. Без цього сценарію приватні маршрути черг, історії та адміністративних сторінок залишаються недоступними.

Автентифікація і авторизація в реалізації розведені окремо. Перша відповідає на питання, хто виконує запит. Друга – чи має цей користувач право на конкретну дію. На серверному боці проміжна функція authenticate перевіряє

токен, а requireRole(’ADMIN’) використовується для маршрутів статистики й адміністративних операцій.

Клієнт також приховує недоступні сторінки через захищені маршрути, але це не замінює серверну перевірку. Інтерфейс лише не показує зайвий шлях користувачу; остаточний контроль лишається в API.

Рольова модель системи навмисно проста і складається з двох ролей: MODERATOR і ADMIN. Це відповідає практичному поділу робочих функцій. Модератор повинен мати можливість відкривати панель черг, отримувати матеріал, переглядати його, застосовувати політики, схвалювати, пропускати та переглядати історію рішень. Адміністратор, крім цього, отримує доступ до панелі статистики, сторінок користувачів, політик та контенту. Такий набір ролей достатньо відбиває концепцію RBAC і водночас не ускладнює систему надмірною ієрархією.

Ролі відображаються не лише в БД. Після входу користувач потрапляє в єдиний каркас застосунку, але набір доступних елементів навігації та зміст сторінок залежать від повноважень. Модератор не бачить адміністративних інструментів, які не стосуються його щоденної роботи.

У клієнтській частині логіка входу не обмежується одноразовим запитом до сервера. Контекст автентифікації зберігає користувача, токен доступу та токен оновлення, тому після перезавантаження сторінки застосунок може відновити сесію без повторного введення пароля. Водночас це не скасовує перевірку на сервері: приватні маршрути все одно очікують коректний токен. Такий поділ дозволяє зберегти зручність роботи модератора і не переносити критичні рішення безпеки в браузер.

Адміністративна роль має окрему практичну вагу. Саме адміністратор керує користувачами, політиками, статистикою і загальним переліком контенту. Якщо такі функції були б доступні модератору, робочий процес став би менш контрольованим: користувач, який ухвалює рішення щодо матеріалу, міг би одночасно змінювати правила, за якими він працює. Тому в реалізації

адміністративні маршрути перевіряються окремо, а на клієнті вони винесені в окремі сторінки навігації.

На рисунку 3.2 зображено сторінку входу до системи модерації контенту, через яку користувач проходить автентифікацію.

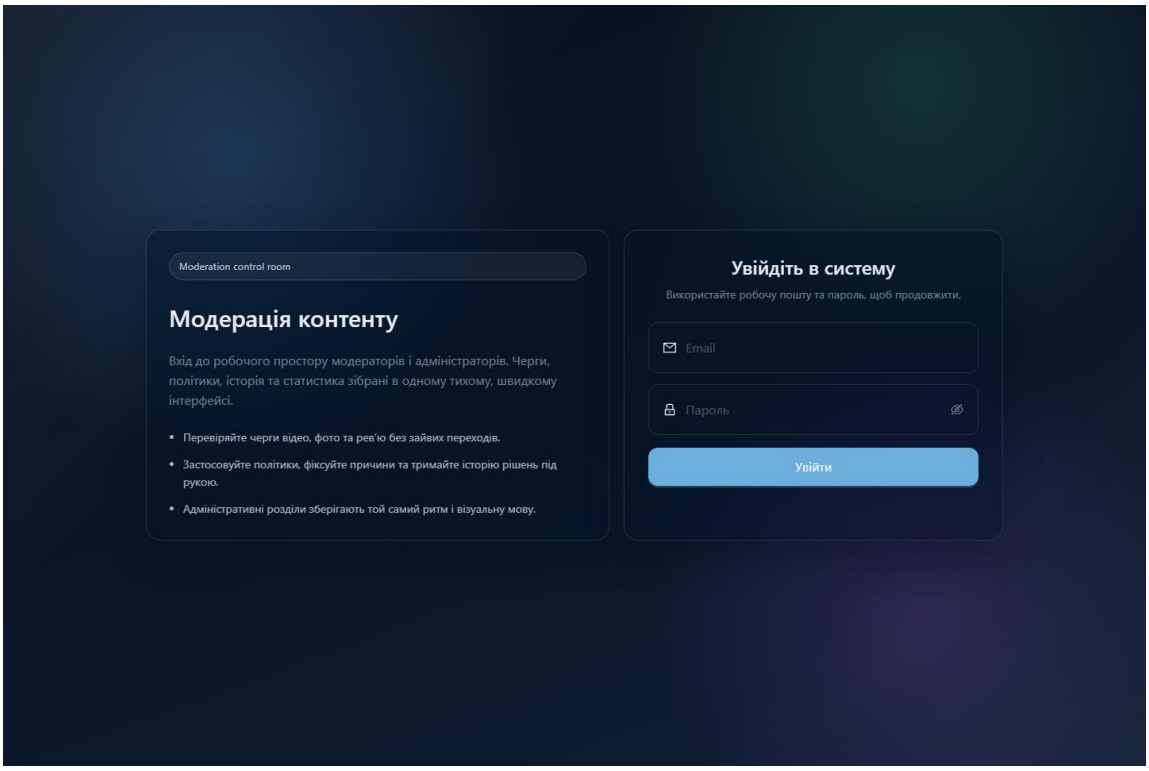

Рисунок 3.2 – Сторінка входу до системи модерації контенту

На рисунку 3.3 зображено головний інтерфейс після входу користувача, який надає доступ до основних робочих розділів системи.

У таблиці 3.2 наведено ролі користувачів і доступні для них дії в системі VideoModerationTool.

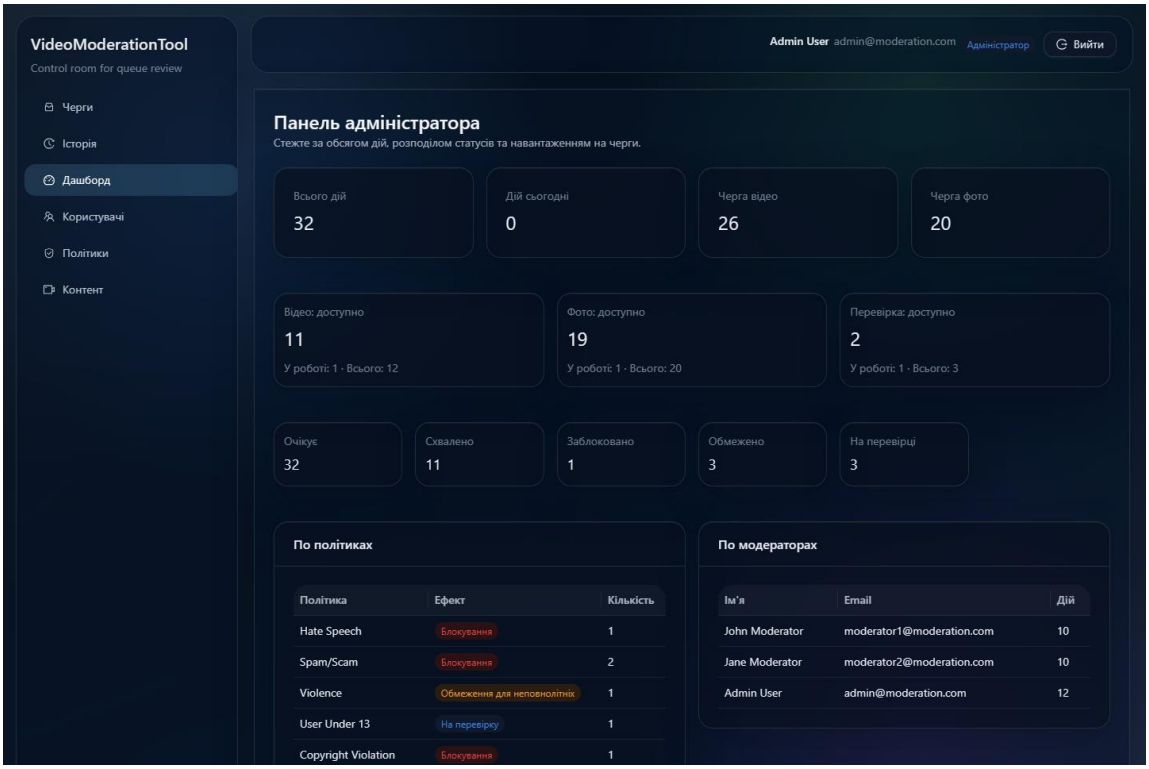

Рисунок 3.3 – Головний інтерфейс після входу користувача

Таблиця 3.2 – Ролі користувачів і доступні дії в системі

Роль	**Дозволені дії**	**Обмеження**
`MODERATOR`	робота з чергами, перегляд матеріалів, схвалення, застосування політик, пропуск, історія	відсутній доступ до статистики, користувачів, повного керування політиками
`ADMIN`	усі дії модератора, статистика, користувачі, політики, контент	адміністративні функції вимагають окремої перевірки ролі на сервері

3.3 Реалізація панелі черг модерації та робочого місця модератора

Сторінка QueuesPage.tsx виконує роль стартового робочого екрана модератора. Вона відображає три типи черг: VIDEO, PHOTO, REVIEW. Дані надходять із маршруту /api/queues, який повертає для кожної черги кількість доступних елементів, елементів у роботі та загальну кількість. Модератор бачить навантаження і вибирає напрям обробки без послідовного відкриття кожної черги.

Робоче місце модератора реалізоване у ModerationPage.tsx. Цей компонент не просто показує матеріал, а організовує повний цикл взаємодії з ним. Після входу на сторінку викликається queuesApi.getNext(queue), який звертається до /api/queues/:type/next. Якщо система вже закріпила елемент за поточним користувачем, він повертається повторно. Якщо такого елемента немає, сервер підбирає найстаріший доступний матеріал без блокування. Після цього UI показує медіа, метадані автора, джерела, статусу і стану блокування. На одному екрані поєднано програвач або перегляд зображення, блок Descriptions із полями sourcePlatform, sourceUrl, sourceExternalId, sourceCaption, поле для примітки та таблицю поточної черги.

Важливо, що сторінка модерації поєднує дві суттєві частини: поточний вибраний елемент і таблицю поточної черги. Така компоновка зменшує когнітивні витрати користувача: модератор бачить не лише сам матеріал, а й загальний список черги, де можна зрозуміти, які елементи ще очікують рішення, які вже знаходяться в роботі і який матеріал закріплений саме за ним. У практичному сенсі це знижує ризик «втрати» контенту в REVIEW або помилкового враження, що черга очистилася.

Для відео та фото використовуються різні засоби відображення. Якщо currentQueueItem.type === ’VIDEO’, рендериться відеоплеєр із джерелом, сформованим через resolveMediaUrl(currentQueueItem.filePath). Для фото використовується <img> із тими самими принципами формування URL. Це дає змогу не жорстко прив’язувати інтерфейс до localhost:3001, а зберігати коректну

роботу при зміні API origin, що вже є важливим кроком до гнучкішого розгортання. Додатково інтерфейс показує, чи елемент уже «забрано мною» або «у роботі в іншого модератора», що робить блокування видимим для користувача, а не прихованим технічним механізмом.

Окремо варто підкреслити логіку звільнення елемента. Під час виходу зі сторінки або явної дії release клієнт викликає /api/queues/:type/release, а сервер звільняє елемент лише якщо він справді був закріплений за поточним користувачем. Такий захист від випадкового втручання іншого користувача є важливим практичним наслідком коректно реалізованого контролю доступу.

Проблема черги повторної перевірки була однією з найбільш показових під час доведення системи до працездатного стану. Якщо матеріал повертається до REVIEW, користувач очікує, що після входу до цієї черги він побачить відповідний елемент або зрозуміле повідомлення про реальний стан черги. Через це логіка getNextItem() була організована так, щоб спочатку шукати власний закріплений елемент, а вже потім брати новий доступний запис. Така поведінка усуває враження, що черга «очищається» при вході на сторінку, і робить повторний розгляд передбачуваним.

На рівні інтерфейсу робоче місце модератора повинно підтримувати швидке серійне опрацювання матеріалів. Після виконання рішення сторінка інвалідує кеш запитів, оновлює лічильники черг і отримує наступний елемент. Це важливо для реального процесу, де модератор не повинен після кожного рішення вручну повертатися на головну панель або оновлювати браузер. У такій логіці інтерфейс працює як операційний інструмент, а не як набір окремих статичних сторінок.

Окрему увагу приділено відображенню стану блокування. Якщо елемент закріплений за поточним користувачем, це подається як власний робочий матеріал. Якщо він закріплений за іншим модератором, система не повинна дозволяти паралельне рішення щодо нього. Таке відображення робить серверний механізм блокування зрозумілим на рівні користувацького сценарію.

На рисунку 3.4 зображено панель із чергами модерації, яка дає змогу модератору перейти до перевірки відео, фото або матеріалів повторного розгляду.

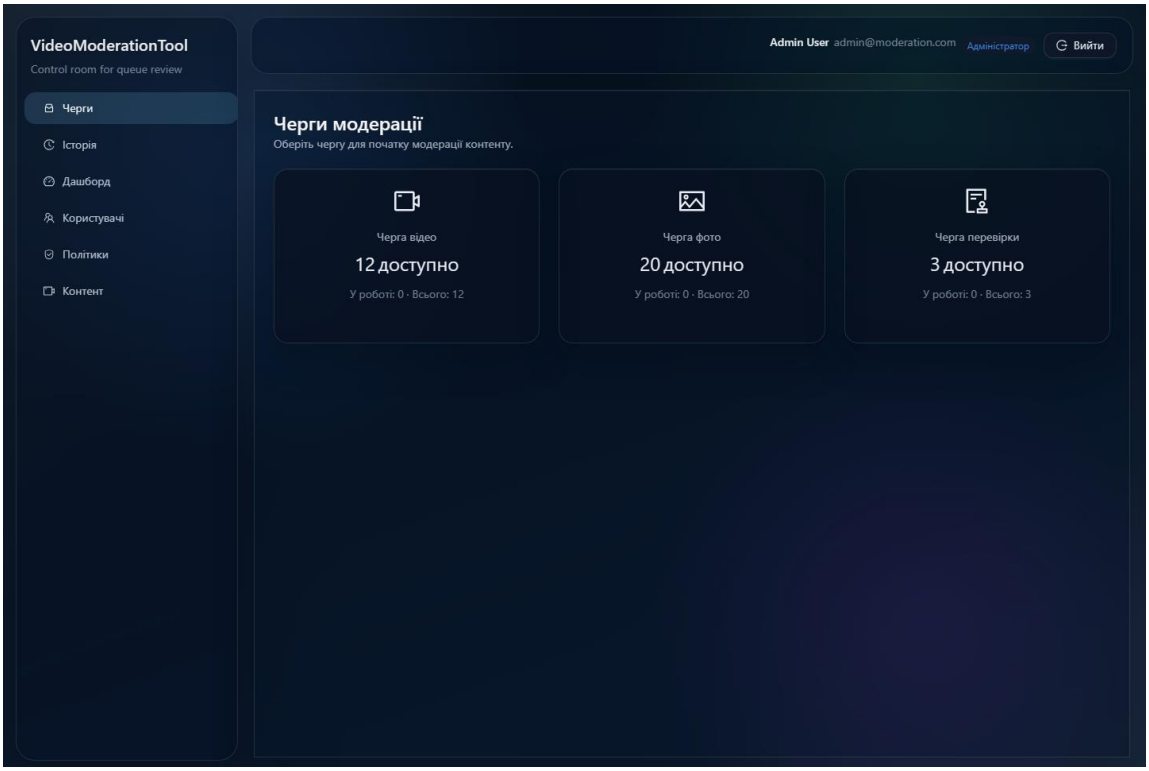

Рисунок 3.4 – Панель із чергами модерації

На рисунку 3.5 зображено перегляд елемента у черзі VIDEO, де модератор може оцінити відеоматеріал і прийняти рішення щодо нього.

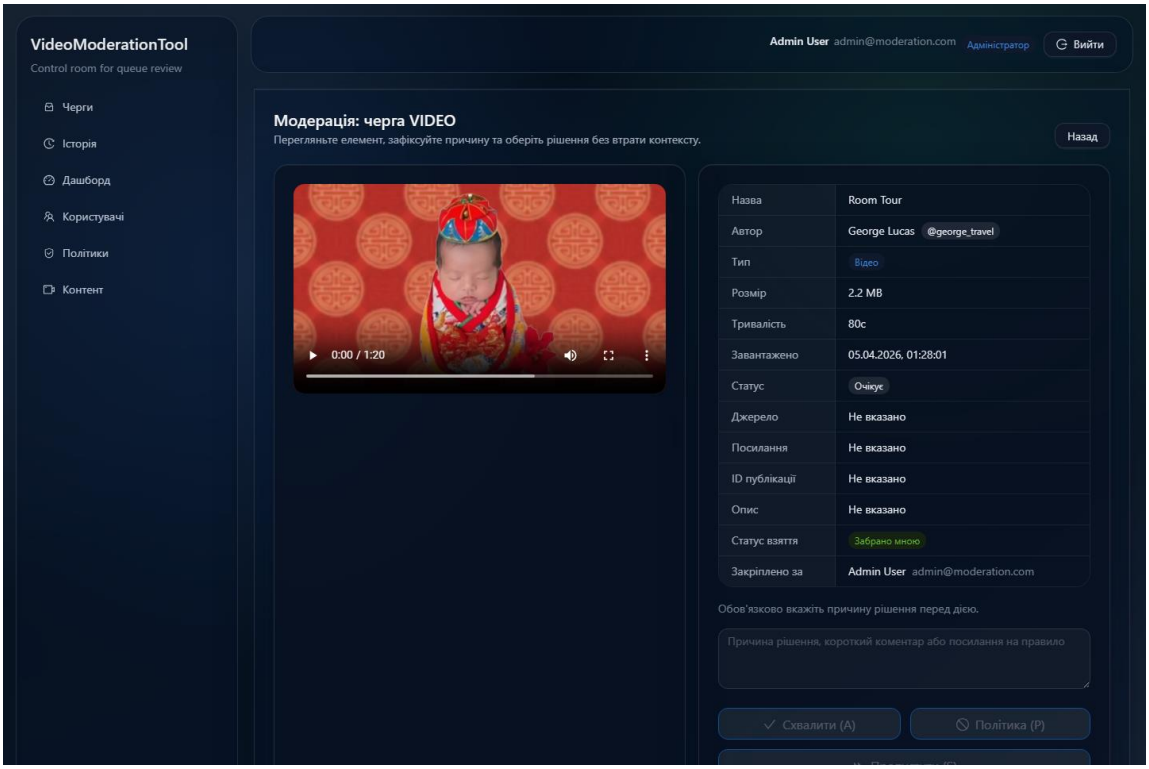

Рисунок 3.5 – Перегляд елемента у черзі VIDEO

На рисунку 3.6 зображено перегляд елемента у черзі PHOTO, що демонструє сценарій роботи зі статичним зображенням.

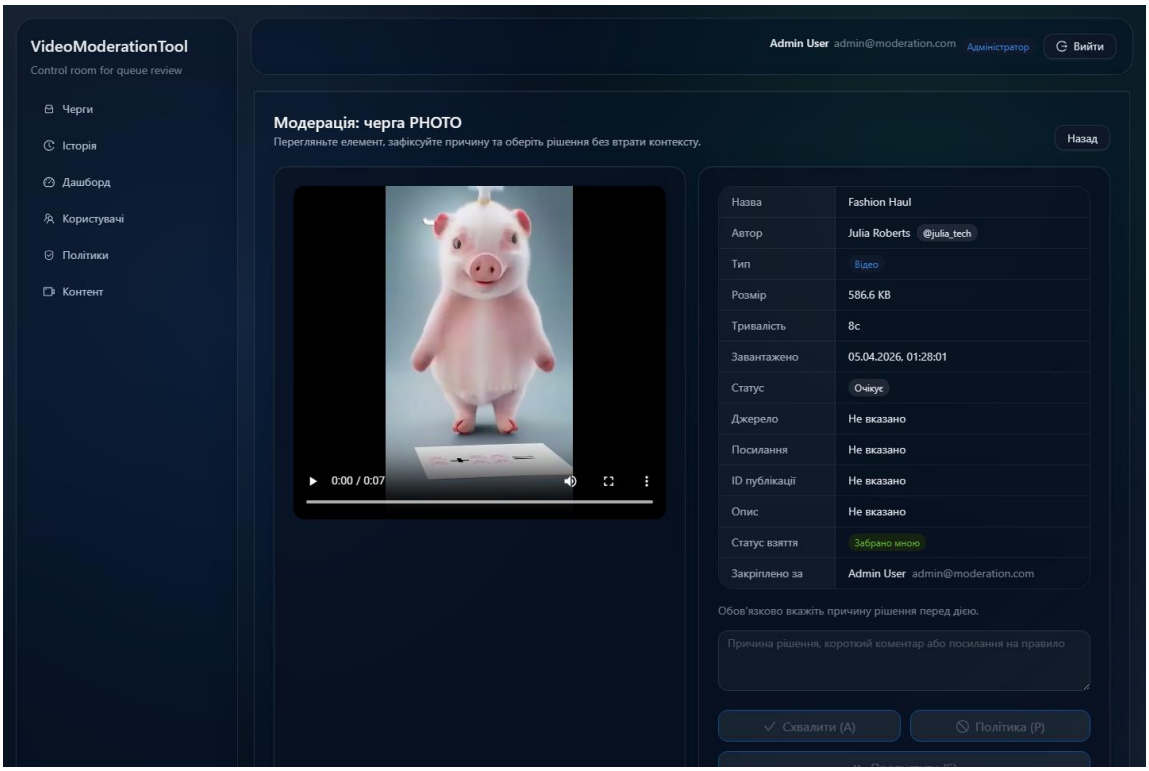

Рисунок 3.6 – Перегляд елемента у черзі PHOTO

На рисунку 3.7 зображено інтерфейс роботи з чергою REVIEW, призначеною для повторного розгляду неоднозначних матеріалів.

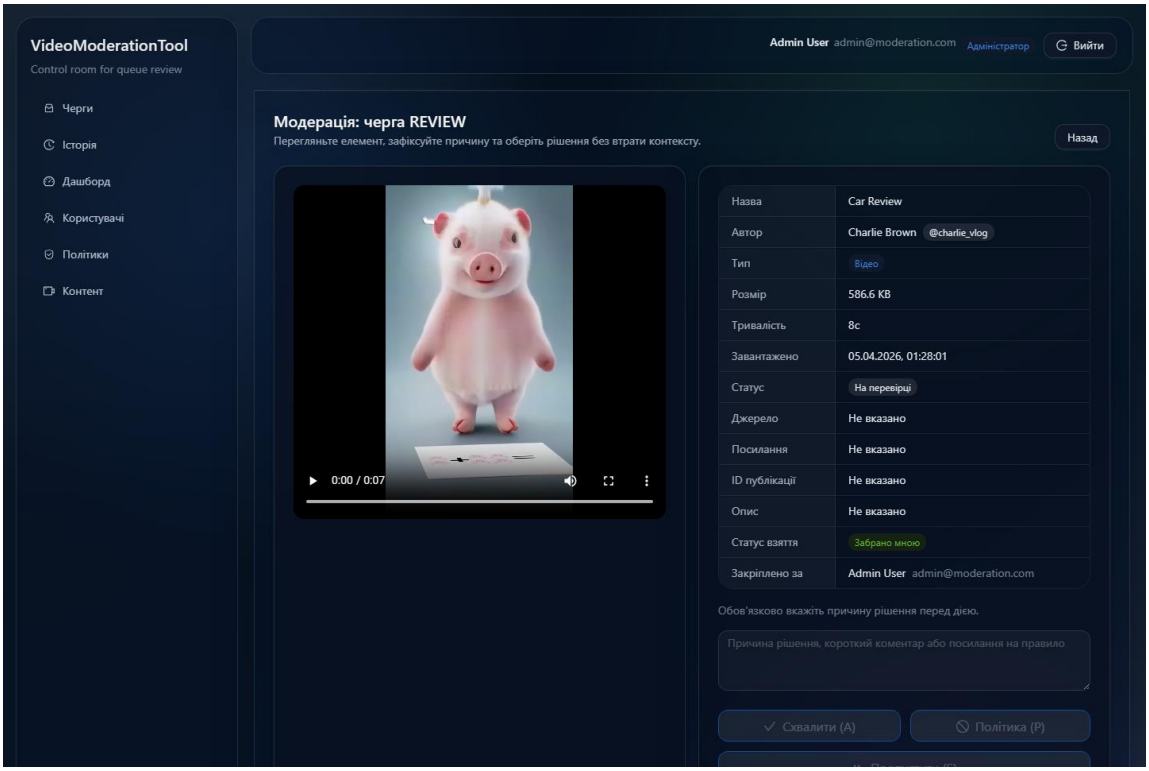

Рисунок 3.7 – Інтерфейс роботи з чергою REVIEW

У таблиці 3.3 наведено API-маршрути, які забезпечують роботу з чергами модерації та отримання матеріалів для перевірки.

Таблиця 3.3 – Маршрути API для роботи з чергами модерації

Маршрут	**Метод**	**Практична функція**
1	**2**	**3**
`/api/queues`	`GET`	підрахунок стану черг для панелі
`/api/queues/:type/items`	`GET`	список елементів конкретної черги

Продовження таблиці 3.3

1	2	3
`/api/queues/:type/next`	`GET`	отримання наступного елемента для модератора
`/api/queues/:type/release`	`POST`	звільнення елемента, закріпленого за користувачем

3.4 Реалізація застосування політик і фіксації рішень

Ключова цінність системи проявляється не в самому перегляді контенту, а в тому, як саме фіксується рішення щодо нього. У ModerationPage.tsx для цього використано три дії: схвалення, відкриття модального вікна політик та пропуск. Перед виконанням будь-якої з дій інтерфейс очікує введення змістової примітки. Така вимога не є випадковою: вона підтримує процедурну логіку системи і підвищує якість подальшого аналізу історії.

Схвалення викликає moderationApi.approve, що відповідає серверному методу approveContent. На сервері перевіряється існування елемента, фіксується попередній статус і в рамках транзакції одночасно оновлюється запис ContentItem та створюється ModerationAction. Застосування транзакції тут є важливим технічним рішенням, оскільки воно не дозволяє опинитися в ситуації, коли статус уже змінено, але дія не потрапила в історію, або навпаки.

Механізм застосування політик складніший. Клієнт показує модальне вікно зі списком активних політик, а сервер приймає масив policyIds. Далі сервіс applyPolicies знаходить активні політики, визначає найсуворіший ефект через EFFECT_PRIORITY, обчислює новий статус, оновлює елемент контенту й створює дію модератора разом із записами у ModerationActionPolicy. Саме ця

багатокрокова логіка є одним із головних доказів функціональної цілісності системи.

Правило суворості винесене на серверний бік, а не залишене в інтерфейсі. У moderation.service.ts ефекти політик зіставляються з числовим пріоритетом: BLOCK має найвищу суворість, MOVE_TO_REVIEW займає проміжне місце, а RESTRICT_MINORS використовується як м'якший наслідок. Після вибору кількох політик сервер перебирає їхні ефекти, знаходить найсуворіший і тільки після цього перетворює його на новий статус контенту. Через це порядок політик у запиті не впливає на результат.

Перетворення наслідку політики на статус виконується окремою логікою. BLOCK переводить матеріал у BLOCKED, MOVE_TO_REVIEW – в IN_REVIEW, а RESTRICT_MINORS – у RESTRICTED. Такий поділ важливий для супроводу: якщо надалі буде додано новий тип наслідку, його потрібно буде явно пов'язати з новим або наявним статусом, а не розкидати умовні перевірки по різних частинах програми.

Зв'язок дії з політиками зберігається через ModerationActionPolicy. Це означає, що після застосування кількох правил історія містить не лише підсумковий статус, а й перелік політик, які стали підставою рішення. Для сторінки історії та адміністративного аналізу це суттєво: можна відновити не тільки факт блокування або обмеження, а й конкретну процедурну причину.

Окремо в реалізації враховано прикладну логіку для політики User Under 13. Якщо така політика застосовується і для автора накопичується значна частка проблемних матеріалів, система може змінити стан пов'язаного контенту автора та позначити профіль як заблокований. Цей сценарій показує, що рішення модератора в програмі може впливати не лише на один елемент, а й на ширший контекст автора. У другому розділі така поведінка описується як проєктна можливість наслідку політики, а в цьому розділі вона фіксується як конкретна серверна реалізація.

Пропуск також не є «порожньою» операцією. Хоча він не змінює статус матеріалу, система створює запис у ModerationAction, де newStatus дорівнює

previousStatus. Так зберігається хронологія взаємодії з елементом і з'являється матеріал для подальшого аналізу випадків, які модератори часто пропускають.

Сторінка політик PoliciesPage.tsx є адміністративним місцем керування правилами. Через неї адміністратор створює, редагує або деактивує політики, які потім з'являються в робочому вікні модератора. У клієнтській реалізації для цього використано форму з полями назви, опису та ефекту, а також окремий Switch для ввімкнення і вимкнення правила без його видалення.

Модератор у цій схемі застосовує політику, але не змінює її формальне визначення.

Політики в системі працюють як довідник правил, а не як жорстко зашитий набір умов у коді клієнта. Це означає, що модератор бачить тільки активні політики, а адміністратор може змінювати їхній склад без зміни інтерфейсної логіки робочого місця. Для навчальної системи це особливо корисно, тому що дозволяє демонструвати різні наслідки модерації на однаковій архітектурі.

Серверна частина не довіряє клієнту у виборі підсумкового статусу. Клієнт передає ідентифікатори політик і примітку, але саме сервер отримує політики з БД, перевіряє їхню активність, визначає найсуворіший ефект і створює записи журналу. Це захищає систему від ситуації, коли інтерфейс випадково або навмисно передасть статус, що не відповідає реальній політиці.

Вимога примітки до рішення також має практичний зміст. Без пояснення в полі note історія дій перетворилася б на технічний список зміни статусів. Наявність текстового обґрунтування дозволяє адміністратору або іншому модератору зрозуміти причину рішення, особливо у випадках повторного розгляду. У цій роботі примітка використовується як мінімальний механізм пояснюваності ручної модерації.

На рисунку 3.8 зображено форму застосування політики до матеріалу, у якій модератор обирає відповідне правило та фіксує рішення.

На рисунку 3.9 зображено результат фіксації рішення модератора після застосування дії до матеріалу.

На рисунку 3.10 зображено сторінку політик модерації, через яку адміністратор може переглядати наявні правила системи.

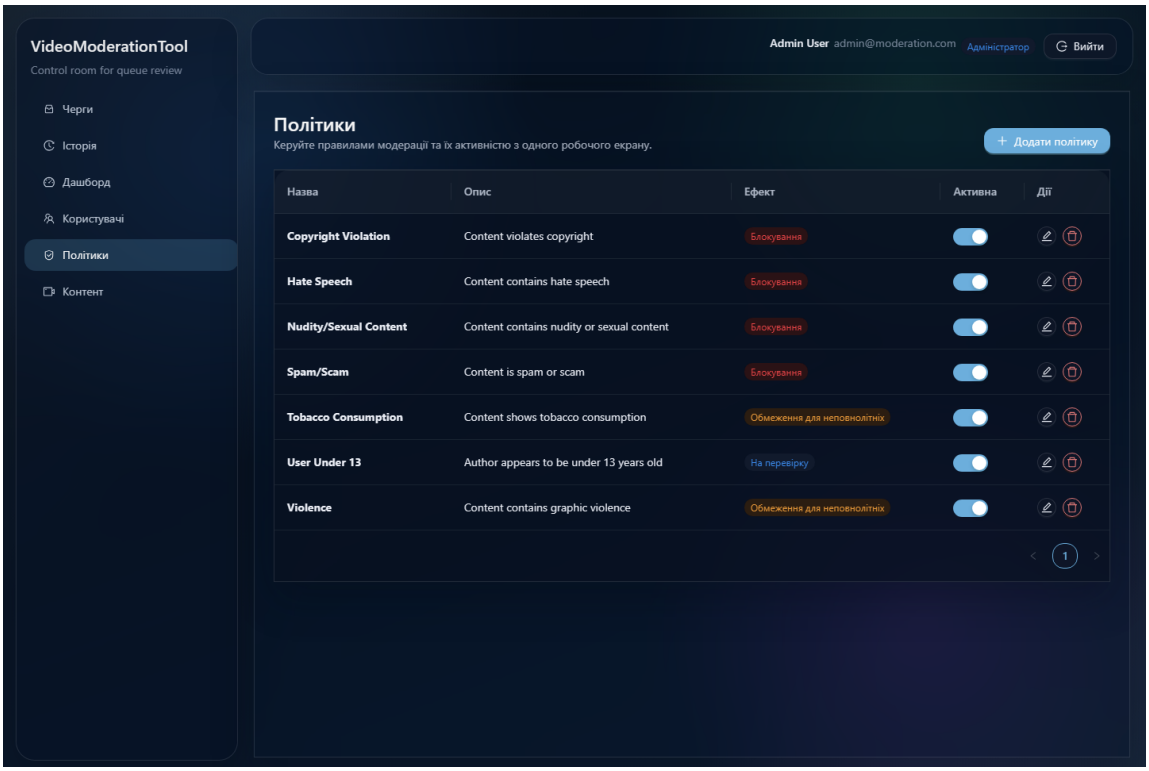

Рисунок 3.8 – Форма застосування політики до матеріалу

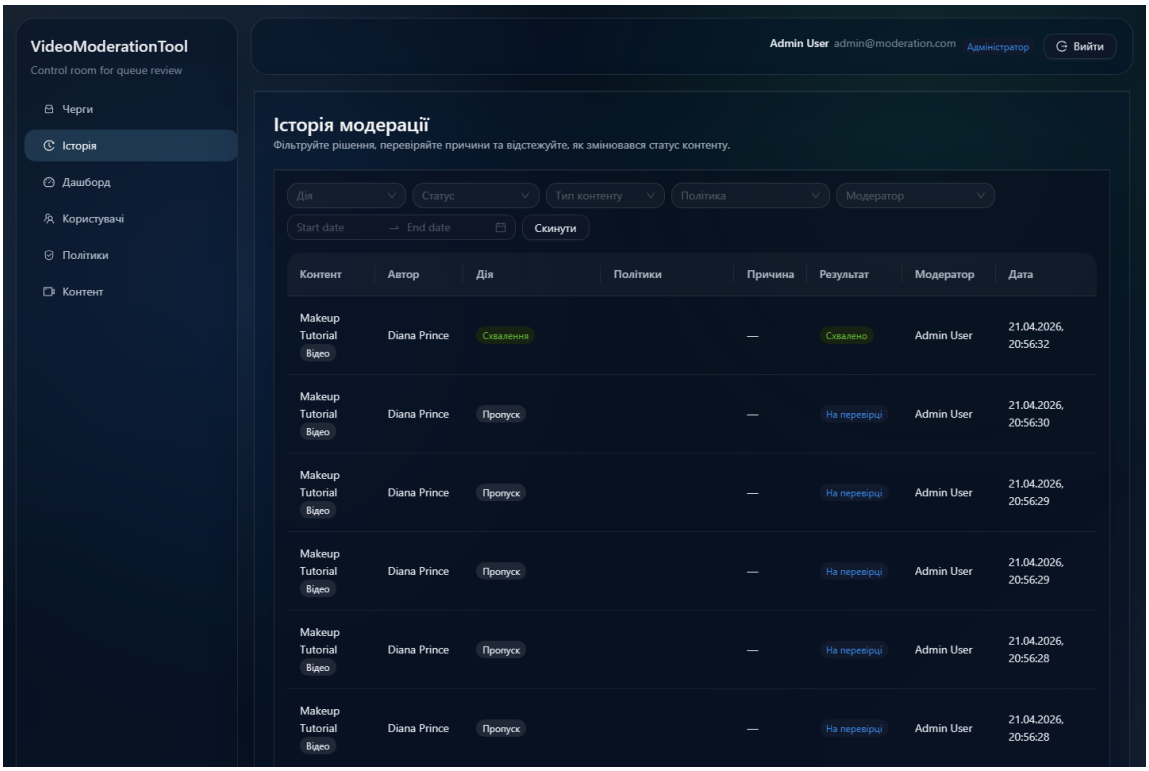

Рисунок 3.9 – Результат фіксації рішення модератора

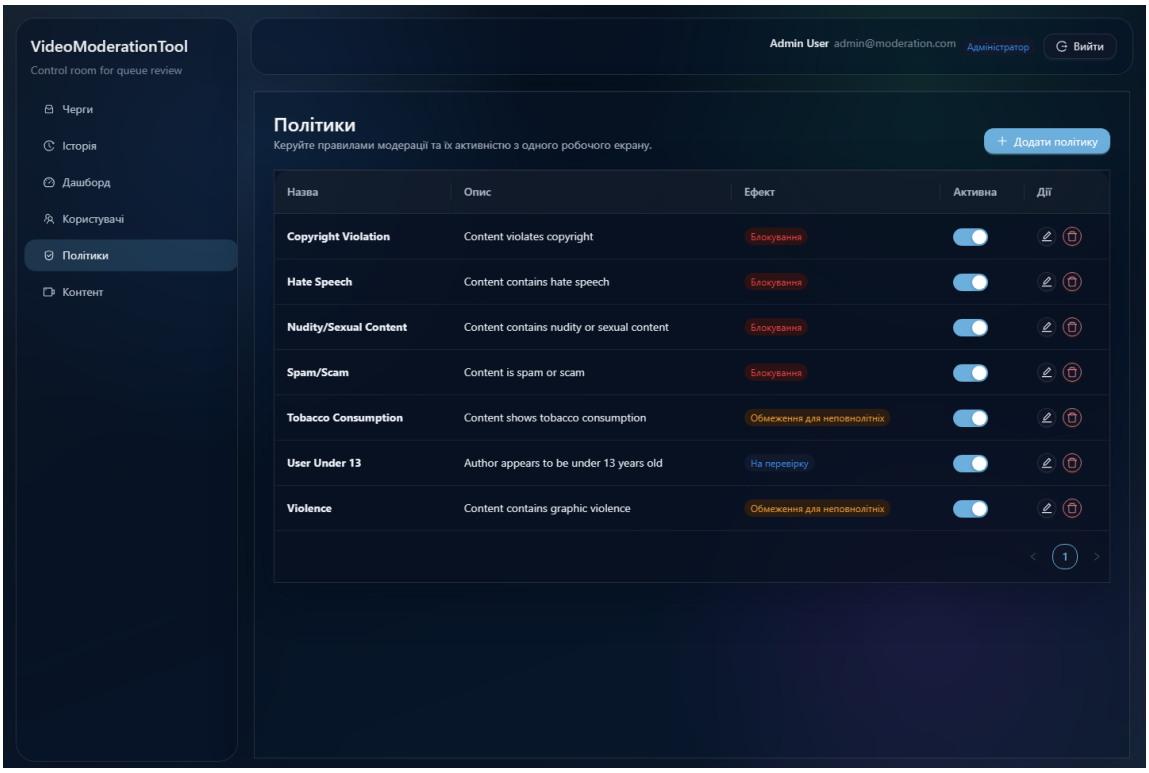

Рисунок 3.10 – Сторінка політик модерації

У таблиці 3.4 наведено API-запити, які використовуються для прийняття рішень модератором і фіксації результатів перевірки.

Таблиця 3.4 – API-запити для прийняття рішень модератора

Маршрут	**Метод**	**Призначення**
`/api/moderation/:contentId/approve`	`POST`	схвалення матеріалу
`/api/moderation/:contentId/apply-policy`	`POST`	застосування однієї або кількох політик
`/api/moderation/:contentId/skip`	`POST`	пропуск матеріалу без зміни статусу

У таблиці 3.5 наведено відповідність між обраною дією модератора і новим статусом контенту в системі.

Таблиця 3.5 – Відповідність між обраною дією і новим статусом контенту

Дія	**Можливий новий статус**	**Коментар**
`APPROVE`	`APPROVED`	матеріал схвалено
`APPLY_POLICY` з `BLOCK`	`BLOCKED`	найсуворіше обмеження
`APPLY_POLICY` з `MOVE_TO_REVIEW`	`IN_REVIEW`	повторний розгляд
`APPLY_POLICY` з `RESTRICT_MINORS`	`RESTRICTED`	часткове обмеження
`SKIP`	статус не змінюється	фіксується як окрема дія в історії

3.5 Реалізація історії, статистики та адміністративних сторінок

Історія модерації в VideoModerationTool не є додатковим журналом «для довідки». Саме через неї можна простежити, хто працював із матеріалом, яка дія була виконана і як змінився статус. Модуль history.routes.ts реалізує маршрут /api/history, який підтримує пагінацію, фільтрацію за модератором, типом дії, новим статусом, типом контенту, політикою та діапазоном дат.

Клієнтська HistoryPage.tsx виводить ці дані у структурованому вигляді, дозволяючи користувачу простежити, хто саме ухвалив рішення, до якого контенту воно застосовувалося і яка політика при цьому була використана. На практиці це робить систему не лише робочою, а й придатною до внутрішнього аудиту та аналізу помилок.

Статистична панель DashboardPage.tsx орієнтована на адміністратора. Вона звертається до маршрутів /api/stats/overview, /api/stats/by-policy і /api/stats/by-moderator. З отриманих даних формуються показники загальної кількості дій, кількості дій за день, розподілу контенту за статусами, обсягу

матеріалу в кожній черзі, статистики за політиками та активності модераторів. На рівні сервера ці агрегати будуються через count і groupBy, а на рівні клієнта відображаються у вигляді окремих карток, таблиці політик та таблиці модераторів. Ця функціональність демонструє, що система може працювати не лише як інструмент індивідуального прийняття рішень, а і як засіб управлінського огляду.

Сторінка користувачів UsersPage.tsx підтримує сценарії адміністративного керування внутрішніми користувачами системи, зокрема створення облікового запису, зміну ролі та редагування пароля. Сторінка політик PoliciesPage.tsx підтримує довідник правил модерації. Сторінка контенту ContentPage.tsx дає змогу переглядати одиниці контенту, які присутні в системі, а також повертати елемент назад у VIDEO, PHOTO або REVIEW через окреме модальне вікно. Ці сторінки формують адміністративний контур поряд із робочим екраном модератора.

Реалізація складається з двох узгоджених контурів роботи: оперативного модераторського та адміністративного. Подальша перевірка має охоплювати обидва контури, а не лише сторінку перегляду окремого матеріалу.

Історія рішень виконує роль зв'язку між цими контурами. Для модератора вона показує, що дія не втрачена після переходу до наступного матеріалу. Для адміністратора вона є джерелом перевірки якості процесу: можна побачити, які політики застосовуються частіше, які матеріали частіше потрапляють у повторний розгляд і які модератори виконували певні дії. Через це історія має підтримувати фільтри, а не бути тільки хронологічним списком.

Статистичний модуль не замінює журнал рішень, а агрегує його для адміністративного огляду. Наприклад, кількість дій за день показує поточне навантаження, розподіл за політиками допомагає побачити домінантні типи порушень, а статистика за модераторами дає уявлення про активність користувачів системи. У навчальному проєкті ці показники побудовані на демонстраційних даних, але сама структура підходить для подальшого розширення.

Сторінка контенту має окреме значення для повторної перевірки. Адміністратор може переглядати матеріали та повертати їх у відповідну чергу, якщо необхідно повторно перевірити рішення або змоделювати інший сценарій. Це доповнює робочий процес модератора і дозволяє демонструвати життєвий цикл матеріалу без ручного втручання в базу даних.

На рисунку 3.11 зображено сторінку історії рішень, яка дає змогу переглядати виконані дії модераторів і зміни стану матеріалів.

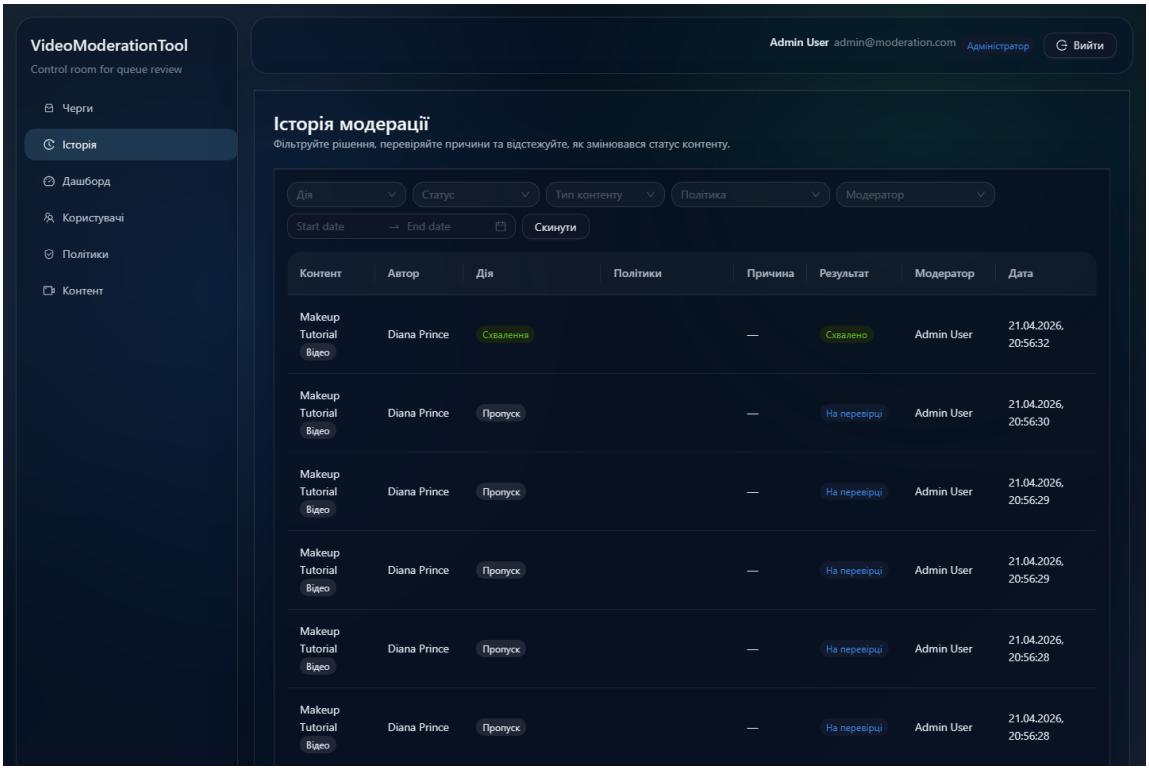

Рисунок 3.11 – Сторінка історії рішень

На рисунку 3.12 зображено статистичну панель адміністратора, що відображає узагальнені показники роботи системи модерації.

На рисунку 3.13 зображено сторінку керування користувачами, яка використовується для адміністративного контролю облікових записів.

На рисунку 3.14 зображено сторінку керування контентом, де адміністратор може переглядати матеріали та їх поточний стан.

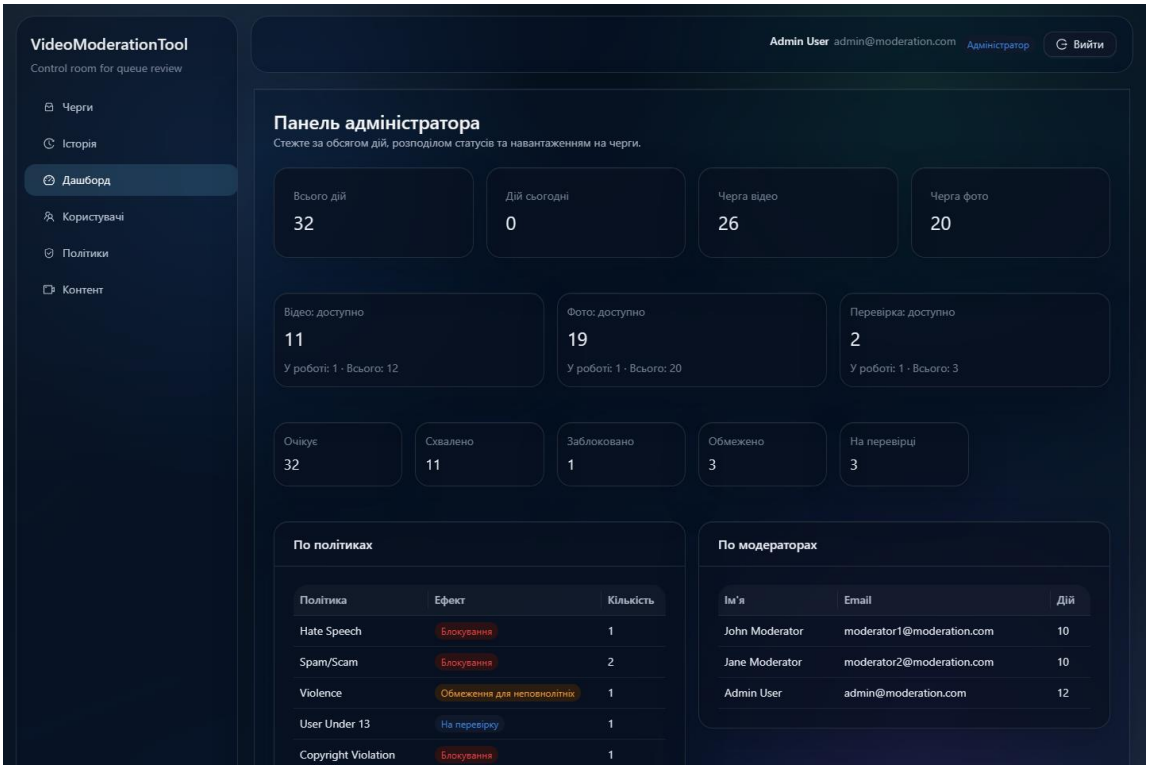

Рисунок 3.12 – Статистична панель адміністратора

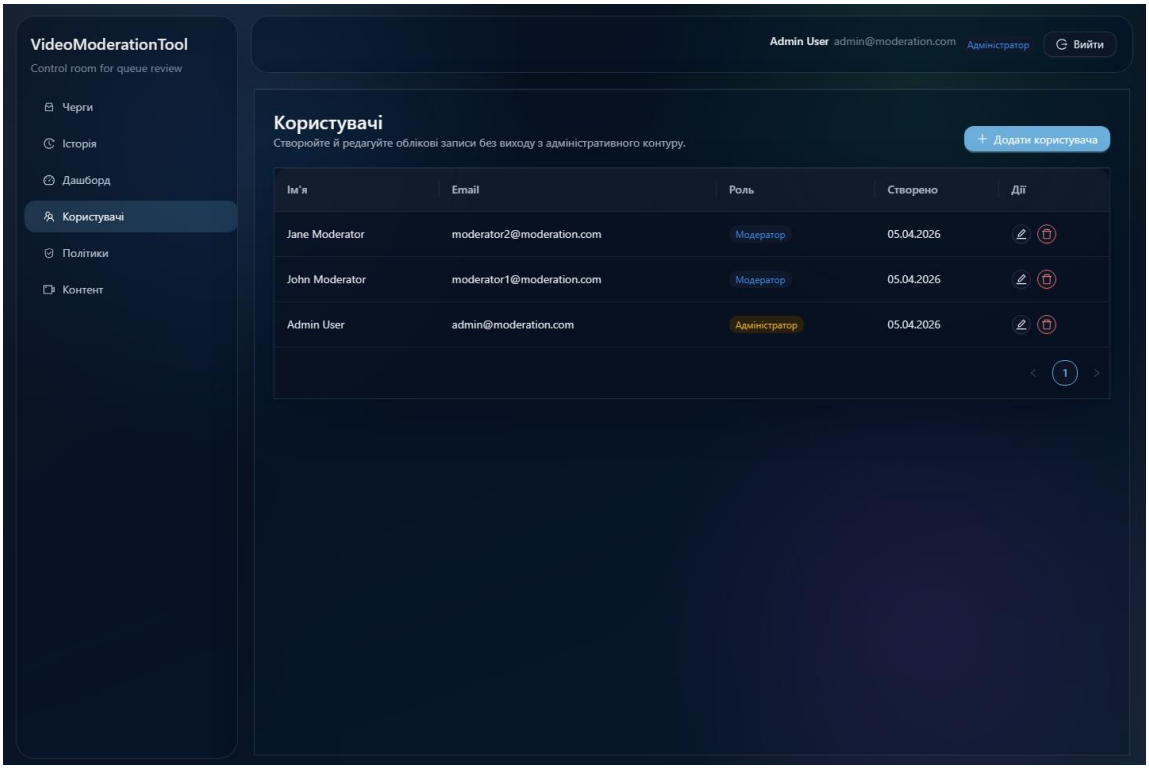

Рисунок 3.13 – Сторінка керування користувачами

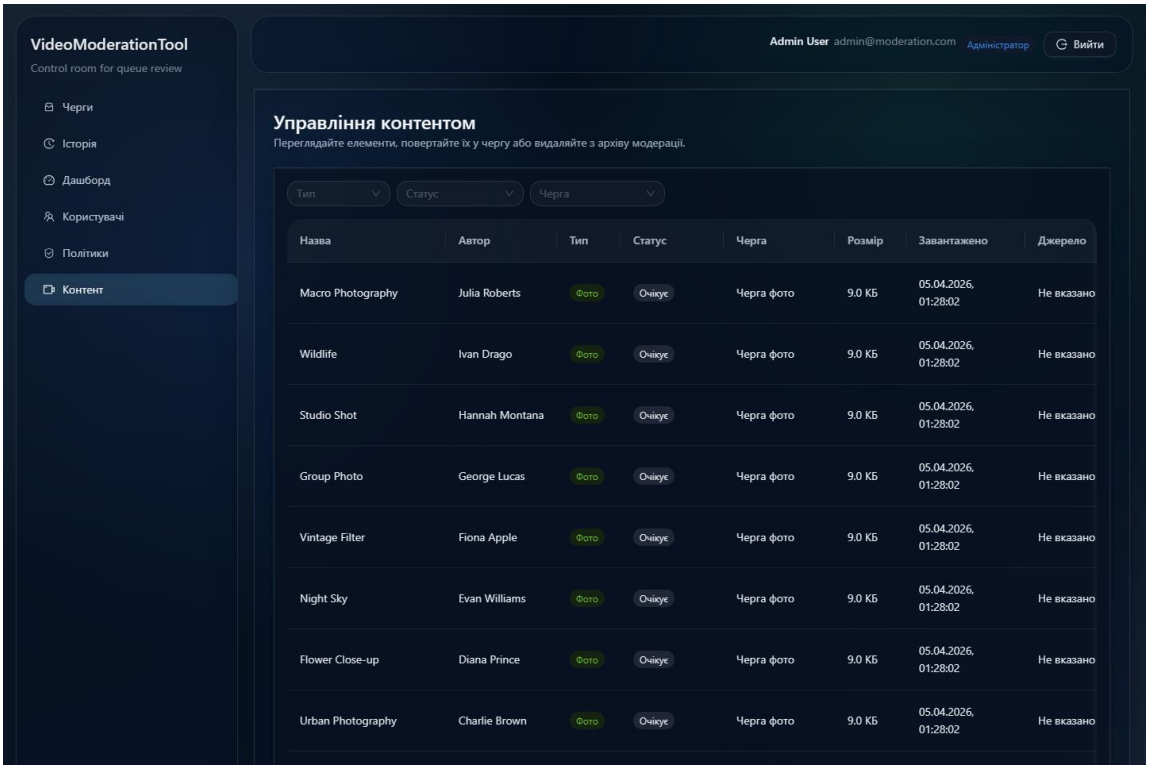

Рисунок 3.14 – Сторінка керування контентом

У таблиці 3.6 наведено адміністративні сторінки системи та пояснено їх призначення.

Таблиця 3.6 – Адміністративні сторінки та їх призначення

Сторінка	**Головна функція**	**Практична користь**
`DashboardPage`	агрегований огляд системи	оцінка навантаження і результативності модерації
`UsersPage`	робота з користувачами	підтримка рольової моделі доступу
`PoliciesPage`	робота з політиками	централізоване керування правилами
`ContentPage`	перегляд і підтримка матеріалів	контроль стану контенту в системі

3.6 Тестування і перевірка працездатності системи

Перевірка працездатності VideoModerationTool має сценарний характер. У цій роботі немає моделі машинного навчання, для якої потрібно рахувати точність класифікації або повноту виявлення порушень. Перевіряється інше: чи збирається система, чи запускаються сервіси, чи відповідають приватні API, чи проходить матеріал через чергу, рішення і журнал.

Технічна частина перевірки починається зі збірки й локального запуску. Для серверної частини контролюється складання TypeScript-коду, доступність Prisma-схеми, застосування міграцій та стабільна робота контейнера server. Для клієнтської частини перевіряється складання і відображення сторінок входу, черг, модерації, історії та адміністративних екранів. Окремим технічним критерієм є коректна робота Docker Compose-конфігурації, де сервіс client має бачити server, а сервер – доступ до PostgreSQL і каталогу uploads/. Під час фактичної перевірки використовувалися команди docker compose up --build, docker compose exec server npx prisma migrate deploy, npm run build для сервера і складання клієнтської частини з локально доступним середовищем Node.js.

Окрема група перевірок стосується прикладної логіки. Відкрита сторінка ще не означає, що модераційний процес працює. Необхідно довести, що:

– користувач може пройти автентифікацію; – приватні маршрути не відкриваються без токена; – панель черг відображає реальні дані; – модератор отримує наступний елемент із відповідної черги; – блокування елемента запобігає конфліктам; – рішення щодо контенту змінює статус і записується в історію; – REVIEW працює як окремий повторний сценарій, а не як втрата контенту; – адміністративні сторінки повертають і відображають узгоджені дані.

Для практичного програмного рішення найбільш релевантними є ручні функціональні сценарії, доповнені перевіркою HTTP-відповідей ключових API. Замість загальної формули «система працює» фіксуються конкретні дії: увійти, відкрити чергу, отримати матеріал, застосувати рішення, побачити запис в історії.

Під час перевірки використовувалася демонстраційна база даних із наперед підготовленими користувачами, політиками, авторами та матеріалами. Це дає однакову стартову точку перед кожним запуском.

Перевірявся не окремий екран, а повний маршрут: вхід, вибір черги, перегляд контенту, застосування рішення, оновлення статусу, запис у журналі та відображення зміни в адміністративній статистиці. У системі модерації помилка часто виникає саме на межі між клієнтом, сервером і БД, тому ізольована перевірка сторінки не дає достатньої впевненості.

Окремо перевірялися негативні сценарії. Якщо запит до приватного маршруту виконується без токена, система не повинна повертати робочі дані. Якщо користувач без адміністративної ролі звертається до адміністративного маршруту, сервер має відмовити в доступі. Якщо черга порожня, клієнт повинен показати зрозумілий стан, а не аварійну помилку. Якщо матеріал уже закріплено за поточним модератором, повторне відкриття сторінки повинно повертати саме цей матеріал. Такі перевірки пов'язані з реальними ризиками системи, а не з формальним натисканням кнопок.

У таблиці 3.7 наведено сценарії функціонального тестування VideoModerationTool, які перевіряють основні дії користувачів у системі.

Таблиця 3.7 – Сценарії функціонального тестування VideoModerationTool

№	**Сценарій**	**Кроки**	**Очікуваний результат**
1	**2**	**3**	**4**
1	Вхід адміністратора	Відкрити сторінку логіну, ввести коректні облікові дані адміністратора, підтвердити вхід	Користувач переходить до робочого інтерфейсу, токен автентифікації збережено

Продовження таблиці 3.7

1	2	3	4
2	Вхід модератора	Використати облікові дані модератора	Доступно робоче місце модератора без адміністративних привілеїв
3	Перегляд панелі черг	Відкрити `QueuesPage` після входу	Відображаються `VIDEO`, `PHOTO`, `REVIEW` з кількістю доступних і зайнятих елементів
4	Отримання елемента VIDEO	Перейти у чергу VIDEO	Система повертає доступний відеоматеріал або повідомлення про порожню чергу
5	Отримання елемента PHOTO	Перейти у чергу PHOTO	Система повертає доступний фотоматеріал або повідомлення про порожню чергу
6	Перевірка повторного відкриття власного елемента	Оновити сторінку модерації або повернутися до неї	Система повертає вже закріплений за користувачем елемент

Продовження таблиці 3.7

1	2	3	4
7	Застосування політики	Ввести примітку, вибрати політику, підтвердити дію	Статус матеріалу змінюється відповідно до ефекту політики, запис потрапляє в історію
8	Схвалення матеріалу	Ввести примітку, натиснути кнопку схвалення	Статус переходить у `APPROVED`, блокування знімається, дія зафіксована
9	Пропуск елемента	Ввести примітку, натиснути кнопку пропуску	Блокування знімається, статус не змінюється, дія зберігається
10	Робота з REVIEW	Перевести матеріал до `IN_REVIEW`, відкрити REVIEW	Матеріал доступний у REVIEW-черзі для повторного розгляду
11	Перегляд історії	Відкрити сторінку історії після виконання кількох дій	Видно дії, модераторів, статуси, політики і часові мітки

Продовження таблиці 3.7

1	2	3	4
12	Перегляд статистики	Увійти як адміністратор і відкрити `DashboardPage`	Показані агреговані показники за діями, чергами, політиками та модераторами
13	Перегляд користувачів	Відкрити `UsersPage` як адміністратор	Сторінка повертає список користувачів і не доступна звичайному модератору
14	Перегляд політик	Відкрити `PoliciesPage`	Активні та неактивні політики відображаються коректно
15	Перегляд контенту	Відкрити `ContentPage`	Адміністратор бачить матеріали й їхні стани

Черга REVIEW перевірялася окремо. Саме в цьому сценарії користувач найлегше сприймає затримку або порожній список як «очищення» черги, хоча матеріал міг бути закріплений або перебувати в іншому стані. Після входу на сторінку система має або показати вже закріплений елемент, або коректно повідомити про реальний стан черги. Технічно ця поведінка пов’язана з логікою getNextItem() і пошуком ownedItem у queues.service.ts.

Перевірка черг окремо показала, що матеріали не зникають із процесу через сам факт відкриття сторінки повторної перевірки. Сервер повертає власний закріплений елемент, а інтерфейс додатково показує поточний стан черги. У цій частині дублювання і втрата контексту обмежуються маршрутизацією на сервері, а не перекладаються на уважність користувача.

Сценарій APPLY_POLICY перевірявся через зв'язок між політикою і підсумковим статусом. Політика з ефектом BLOCK переводить матеріал у BLOCKED; MOVE_TO_REVIEW – у IN_REVIEW і REVIEW-чергу; RESTRICT_MINORS – у RESTRICTED. У кожному випадку дія зберігається в журналі, а при застосуванні політик створюється зв'язок через ModerationActionPolicy.

Тут важливий не візуальний ефект натискання кнопки, а сталий запис у БД.

Перевірка сценарію SKIP потрібна для підтвердження того, що пропуск не руйнує маршрут обробки. У цьому випадку статус матеріалу не змінюється, але блокування знімається і створюється окремий запис дії. Така поведінка важлива для реальної роботи: модератор може пропустити матеріал через нестачу контексту або технічну проблему, але сама взаємодія з матеріалом усе одно має бути видимою в історії.

Не менш важливим є тестування приватних API. Для системи модерації критично, щоб маршрути черг, історії, статистики, політик і користувачів не були доступні без токена. Тому окремим класом перевірок є звернення до цих маршрутів без авторизації та перевірка того, що сервер повертає помилку доступу, а не порожню відповідь або аварійне завершення процесу. У фактичній перевірці після стабілізації сервера маршрут /api/health повертав 200 OK, а /api/queues без токена – коректну помилку авторизації, що підтвердило відокремлення загальнодоступного маршруту перевірки доступності сервера від приватних робочих API.

У таблиці 3.8 наведено перевірки API та очікувані відповіді серверної частини для ключових маршрутів системи.

Таблиця 3.8 – Перевірка API та очікувані відповіді

Маршрут	**Умова перевірки**	**Очікуваний результат**
`/api/health`	без токена	`200 OK`, ознака доступності сервера
`/api/auth/login`	коректні дані	успішна відповідь із токенами й даними користувача
`/api/queues`	без токена	помилка авторизації
`/api/queues`	з токеном	JSON із кількістю елементів у чергах
`/api/queues/video/items`	з токеном	список елементів VIDEO
`/api/queues/video/next`	з токеном	об’єкт контенту або повідомлення про порожню чергу
`/api/history`	з токеном	пагінований список дій модерації
`/api/stats/overview`	з токеном адміністратора	агреговані показники системи
`/api/policies?active=true`	з токеном	список активних політик

У процесі фактичної роботи над системою було виявлено технічний ризик: при незастосованій міграції Prisma сервер починав падати на запитах до історії через відсутність колонки sourcePlatform у таблиці content_items. Технічна перевірка для такого вебзастосунку повинна включати не лише сценарії інтерфейсу, а й контроль стану міграцій. Після виконання npx prisma migrate deploy у контейнері сервера та його перезапуску запити до історії, черг, статистики й політик почали повертати коректні відповіді. Перевірка працездатності системи охоплює повний ланцюг «схема БД -> сервер -> API -> клієнт».

Цей випадок має практичне значення для аналізу результатів. Помилка ERR_EMPTY_RESPONSE на клієнті не завжди означає помилку інтерфейсу. У конкретній ситуації причина була на серверному боці: API не міг коректно виконати запит через невідповідність схеми БД очікуваній Prisma-моделі. Після застосування міграцій проблема зникла, що підтвердило необхідність перевіряти систему як повний стек, а не окремо клієнтську частину. Для модераційної системи це особливо важливо, бо користувач бачить лише повідомлення про помилку, але реальна причина може бути в базі даних або серверній логіці.

Додатково перевірялася узгодженість фронтенду і бекенду після зміни поведінки черг. Клієнтські запити до /api/queues, /api/queues/:type/items і /api/queues/:type/next повинні повертати дані в очікуваній формі, а сторінки не повинні показувати порожній стан, якщо сервер повернув закріплений матеріал. Така перевірка підтвердила, що виправлення логіки черг має видимий результат у користувацькому інтерфейсі.

У таблиці 3.9 наведено технічні перевірки збірки й запуску, які підтверджують працездатність основних компонентів проєкту.

Таблиця 3.9 – Технічні перевірки збірки й запуску

Перевірка	**Що саме контролюється**	**Очікуваний результат**
1	**2**	**3**
Збірка серверної частини	компіляція TypeScript-коду сервера	успішне завершення без помилок типізації
Збірка клієнтської частини	збірка React/Vite	успішне формування production bundle
Застосування міграцій	актуальність схеми БД	усі потрібні зміни присутні в PostgreSQL
Docker Compose	спільний запуск сервера, клієнта, Prisma Studio	сервіси піднімаються і доступні за очікуваними портами

Продовження таблиці 3.9

1	2	3
Перевірка демоданих	наявність початкових даних для сценаріїв	доступні користувачі, політики, фото, відео, історія

На рисунку 3.15 зображено діаграму розподілу тестових матеріалів за чергами, що демонструє наповнення VIDEO, PHOTO і REVIEW у демонстраційному наборі даних.

Рисунок 3.15 – Діаграма розподілу тестових матеріалів за чергами

Тестування не подається як «вичерпне» у промисловому значенні. У системі не реалізовано окремий набір автоматизованих інтеграційних тестів або наскрізних тестів, тому перевірка зосереджена на основних технічних і функціональних сценаріях: вхід у систему, отримання елемента, прийняття рішення, поява запису в історії, перегляд статистики та службових сторінок. Досвід із незастосованою міграцією також показав потребу в окремому контрольному списку старту середовища: перевірка міграцій, демоданих,

маршруту перевірки доступності сервера і основних приватних маршрутів після запуску контейнерів.

3.7 Аналіз отриманих результатів

Після реалізації VideoModerationTool демонстраційні фото й відео перестають бути просто файлами в каталозі uploads/. Кожен матеріал має запис у БД, активну або завершену чергу, статус і історію дій.

У робочому сценарії це виглядає доволі просто: модератор не шукає файл вручну, а отримує наступний елемент через API, бачить метадані, залишає примітку і переводить матеріал у стан, який потім видно в історії та адміністративних таблицях.

Маршрут обробки став явним. У системі є різниця між первинною перевіркою відео, первинною перевіркою фото і повторним розглядом. Для користувача це виглядає як три черги, але на рівні даних за цим стоять QueueType, ContentStatus, lockedBy і lockedAt.

Модель не оцінює зміст автоматично. Вона прибирає хаотичність ручної перевірки, де незрозуміло, хто взяв матеріал, чи був він уже переглянутий і чому потрапив у повторний розгляд.

Поля lockedBy і lockedAt у поєднанні з логікою видачі наступного елемента розв’язують практичну проблему паралельної роботи з одним матеріалом. Якщо два модератори одночасно відкрили б одне відео і ухвалили різні рішення, статус контенту та журнал дій стали б суперечливими. У реалізованій схемі матеріал тимчасово закріплюється за конкретним користувачем, а повторне відкриття сторінки повертає йому той самий елемент. Це не декоративна деталь інтерфейсу, а частина цілісності процесу.

Застосування політик також винесене з рівня окремої кнопки в окрему модель даних. Модератор може вибрати одну або кілька політик, а сервер визначає підсумковий статус за найсуворішим ефектом. Через це в системі

зберігається відмінність між блокуванням, обмеженням для неповнолітніх і переведенням у REVIEW. Для модераційного процесу така відмінність важлива, бо не кожен випадок потребує однакового наслідку. Логіка вибору наслідку може розглядатися як окремий елемент підтримки рішень, для якого релевантними є підходи нечіткої логіки та налаштування функцій належності [33–36].

Журнал ModerationAction робить рішення придатним до подальшої перевірки. У записі зберігаються матеріал, модератор, тип дії, попередній статус, новий статус, примітка і час. Адміністративний перегляд у такій системі спирається не лише на поточний статус контенту, а й на шлях до цього статусу. Саме цей шлях потрібний, коли необхідно з'ясувати, чому матеріал опинився в повторній перевірці або яка політика стала підставою для обмеження.

Розмежування робочого і адміністративного контурів не зводиться до різних пунктів меню. Модератор працює з чергами та рішеннями, тоді як адміністратор бачить користувачів, політики, контент і статистику. У такому поділі менше випадкових дій: користувач із роллю модератора не отримує зайвих адміністративних інструментів, а адміністратор може контролювати не один матеріал, а стан процесу загалом.

Статистичний контур у роботі має демонстраційний характер. На основі журналу дій уже можна будувати показники за політиками, модераторами, статусами й чергами. Для повнішої системи цей шар можна було б розширити до аналізу навантаження, частоти повторного розгляду і типових причин блокування. Такий напрям узгоджується з підходами до технологій прийняття рішень в інформаційних системах та аналізу багатовимірних даних [37–38].

У результату є помітне обмеження: система не класифікує зміст фото або відео автоматично і не виконує комп'ютерне розпізнавання зображень. Це не суперечить постановці задачі, бо метою було підвищення ефективності та контрольованості ручної модерації.

Автоматичний сигнал можна додати пізніше як допоміжний шар: у метадані матеріалу, пріоритет черги або попереднє позначення ризику. Для

зображень таким шаром може бути кластерне подання візуальних ознак, яке не ухвалює рішення замість модератора, але допомагає попередньо впорядкувати матеріали [39].

Ще одним обмеженням є локальний характер демонстраційних даних. Матеріали, користувачі й політики створені для перевірки сценаріїв, а не отримані з реальної соціальної мережі. Водночас локальна демонстрація дозволяє стабільно повторювати перевірку і не залежати від зовнішнього API. Для кваліфікаційної роботи це прийнятний компроміс, оскільки головним результатом є програмна реалізація маршруту модерації.

У реалізованому прототипі перевірено ті елементи, які безпосередньо пов’язані з початковою постановкою задачі: черги для відео, фото і повторної перевірки, рольовий доступ, застосування політик, журналювання рішень, адміністративна статистика і локальне середовище запуску. Разом вони формують не автоматичний класифікатор, а кероване робоче середовище для ручної модерації мультимедійного контенту.

3.8 Обмеження перевірки та подальше розширення тестування

Поточна перевірка охоплює основні сценарії роботи системи: рольовий доступ, панель черг, екран перегляду матеріалу, механізм фіксації рішень, повторну перевірку, історію дій і адміністративний контроль. Кожен із цих елементів підтримується спільною моделлю даних і набором узгоджених API-маршрутів.

Практичний результат особливо добре проявляється в тому, що система підтримує повний шлях одиниці контенту:

– матеріал знаходиться у відповідній черзі; – модератор отримує його в роботу; – система уникає одночасного конфліктного доступу; – щодо матеріалу виконується одне з допустимих рішень; – наслідок цього рішення зберігається у

статусі та в журналі; – адміністративний контур отримує можливість переглядати історію й агреговану статистику.

Подальше розширення перевірки може відбуватися за такими напрямами:

– інтеграція з реальними зовнішніми джерелами контенту або платформеними API; – додавання автоматизованих сигналів або попередньої класифікації як допоміжного, а не остаточного механізму; – розширення механізму апеляцій і повторного розгляду за рахунок окремих ролей або додаткових статусів; – впровадження детальнішого аудиту адміністративних дій; – розробка набору автоматизованих інтеграційних та наскрізних тестів; – розширення статистичних модулів для аналізу якості модерації, навантаження на команди та ефективності окремих політик.

Наведені обмеження визначають обсяг наступних перевірок і не дублюють підсумкові висновки роботи.

Перспективний напрям розширення системи пов’язаний із додаванням допоміжних сигналів комп’ютерного зору для попередньої оцінки фото та відеоматеріалів. У межах цієї роботи ці підходи не замінюють ручну модерацію, але можуть бути використані як джерело ризикових ознак, пріоритетів черги або підказок для модератора.

Другий напрям пов’язаний із підтримкою прийняття рішень у складних інформаційних системах. Дослідження нечіткої логіки, інтелектуальних засобів ухвалення рішень, аналізу багатовимірних даних і формування ознакового простору можуть бути використані для подальшого розвитку адміністративної аналітики та пояснюваного вибору політик модерації. Такий розвиток має залишатися допоміжним: остаточне рішення щодо спірного мультимедійного контенту повинно бути пов’язане з дією відповідального користувача й зафіксоване в журналі.

Отже, виконана перевірка підтверджує працездатність основних сценаріїв, а визначені обмеження показують напрям подальшого розвитку тестування. Для кваліфікаційної роботи цього достатньо, оскільки реалізовано й перевірено

повний маршрут від входу користувача до прийняття рішення, зміни стану матеріалу, фіксації історії та адміністративного перегляду результатів.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмне забезпечення для модерації контенту в соціальних мережах. Основна увага приділена не автоматичному розпізнаванню порушень, а організації ручної перевірки фото і відеоматеріалів: видачі матеріалу модератору, запобіганню паралельній обробці, застосуванню політик, фіксації рішень та адміністративному контролю.

Модерація в такій постановці є частиною інфраструктури платформи, а не допоміжним переглядом файлів. Фото і відео перевіряються по-різному: для зображення часто достатньо одного візуального контексту, тоді як відео потребує перегляду в часі та врахування фрагментів, підписів і джерела. Окрема черга REVIEW залишена для випадків, де первинного рішення недостатньо.

Архітектура побудована навколо поділу відповідальності. React відповідає за робочі сценарії модератора й адміністратора. Express і TypeScript обробляють авторизацію, маршрути черг, застосування політик і запис історії. PostgreSQL та Prisma фіксують сутності процесу: користувачів, авторів контенту, матеріали, політики та дії модерації.

Реалізований прототип підтримує робочий маршрут матеріалу: автентифікація користувача, доступ відповідно до ролі, вибір черги, відкриття матеріалу, рішення модератора і запис у журналі дій. Адміністративна частина охоплює користувачів, політики, контент, історію та статистику.

REVIEW-черга має окремий зміст, бо відображає не тип файла, а стан повторної перевірки.

Працездатність перевірялася через локальний запуск Docker Compose, застосування Prisma-міграцій, ключові API та сценарії інтерфейсу. Під час перевірки проявилася реальна технічна залежність: незастосована міграція Prisma спричиняла помилки серверних маршрутів і на клієнті виглядала як порожні відповіді. Після застосування міграцій маршрути історії, черг, статистики й політик почали працювати стабільно.

Цей випадок показує, що для такої системи недостатньо перевірити тільки сторінки клієнта. Клієнт, сервер і база даних мають розглядатися як один робочий стек.

Практична цінність роботи полягає у відтворюваній процедурі ручної модерації. Матеріал має статус, чергу, історію рішень і зв'язок із політиками; модератор працює з конкретним елементом, а адміністратор бачить стан процесу через службові сторінки й статистику. Це вже більше, ніж екран перегляду медіафайлів.

Подальший розвиток може охоплювати інтеграцію зовнішніх джерел контенту, механізм апеляцій, допоміжні автоматизовані сигнали попередньої оцінки, глибший статистичний аналіз і набір інтеграційних тестів. Усі ці напрями мають зберігати базову ідею роботи: автоматизація підтримує ручну модерацію, але не підміняє відповідальне рішення там, де потрібна оцінка контексту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gillespie, T. (2018). Custodians of the internet: Platforms, content moderation, and the hidden decisions that shape social media. Yale University Press.

2. Gorwa, R. (2024). The politics of platform regulation: How governments shape online content moderation. Oxford University Press.

3. Gorwa, R., Binns, R., & Katzenbach, C. (2020). Algorithmic content moderation: Technical and political challenges in the automation of platform governance. Big Data & Society, 7(1).

4. Wang, S., & Kim, K. J. (2023). Content moderation on social media: Does it matter who and why moderates hate speech? Cyberpsychology, Behavior, and Social Networking, 26(7), 527-534.

5. Ruckenstein, M., & Turunen, L. L. M. (2020). Re-humanizing the platform: Content moderators and the logic of care. New Media & Society, 22(6), 1026-1042.

6. He, Q., Hong, Y., & Raghu, T. S. (2025). Platform governance with algorithm-based content moderation: An empirical study on Reddit. Information Systems Research, 36(2), 1078-1095.

7. Cobbe, J. (2021). Algorithmic censorship by social platforms: Power and resistance. Philosophy & Technology, 34, 739-766.

8. Gomes, A. B., & Sultan, A. (2024). Problematizing content moderation by social media platforms and its impact on digital harm reduction. Harm Reduction Journal, 21, Article 194.

9. Li, L., & Zhou, K. (2025). When content moderation is not about content: How Chinese social media platforms moderate content and why it matters. New Media & Society, 27(11).

10. Kira, B. (2025). Regulatory intermediaries in content moderation. Internet Policy Review, 14(1), 1-26.

11. Lazaros, K.-P., Vrahatis, A. G., & Kotsiantis, S. (2026). Human-in-the-loop artificial intelligence: A systematic review of concepts, methods, and applications. Entropy, 28(4), Article 377.

12. Regulation (EU) 2022/2065 of the European Parliament and of the Council of 19 October 2022 on a single market for digital services and amending Directive 2000/31/EC (Digital Services Act). (2022). Official Journal of the European Union, L 277, 1-102.

13. Ortolani, P. (2023). The Digital Services Act, content moderation and dispute resolution. SSRN Electronic Journal.

14. Husovec, M. (2024). Principles of the Digital Services Act. Oxford University Press.

15. Husovec, M. (2024). Content moderation: Outline. In Principles of the Digital Services Act (pp. 185-201). Oxford University Press.

16. Husovec, M. (2024). Fair moderation process. In Principles of the Digital Services Act (pp. 202-248). Oxford University Press.

17. Llanso, E. J. (2020). No amount of «AI» in content moderation will solve filtering's prior-restraint problem. Big Data & Society, 7(1).

18. Arora, A., Nakov, P., Hardalov, M., Sarwar, S. M., Nayak, V., Dinkov, Y., Zlatkova, D., Dent, K., Bhatawdekar, A., Bouchard, G., & Augenstein, I. (2023). Detecting harmful content on online platforms: What platforms need vs. where research efforts go. ACM Computing Surveys, 56(3), Article 72, 1-17.

19. Horta Ribeiro, M., Cheng, J., & West, R. (2022). Post approvals in online communities. Proceedings of the International AAAI Conference on Web and Social Media, 16, 335-346.

20. Horta Ribeiro, M., Cheng, J., & West, R. (2023). Automated content moderation increases adherence to community guidelines. In Proceedings of the ACM Web Conference 2023 (pp. 2666-2676). Association for Computing Machinery.

21. Karabulut, D., Ozcinar, C., & Anbarjafari, G. (2023). Automatic content moderation on social media. Multimedia Tools and Applications, 82, 4439-4463.

22. Douek, E. (2022). Content moderation as systems thinking. Harvard Law Review, 136(2), 526-607. https://harvardlawreview.org/print/vol-136/content-moderation-as-systems-thinking/.

23. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al-Dhaifallah, M. (2021). Methods of classification of images on the basis of the values of statistical distributions for the composition of structural description components. IEEE Access, 9, 92964-92973.

24. Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). Image classification method modification based on model of logic processing of bit description weights vector. Telecommunications and Radio Engineering, 79(1), 59-69.

25. Gorokhovatskyi, V., & Tvoroshenko, I. (2020). Image classification based on the Kohonen network and the data space modification. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020) (Vol. 2608, pp. 1013-1026).

26. Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Vlasenko, N. V. (2020). Using fuzzy clustering in structural methods of image classification. Telecommunications and Radio Engineering, 79(9), 781-791.

27. Kobylin, O., Gorokhovatskyi, V., Tvoroshenko, I., & Peredrii, O. (2020). The application of non-parametric statistics methods in image classifiers based on structural description components. Telecommunications and Radio Engineering, 79(10), 855-863.

28. Tvoroshenko, I., & Zarivchatskyi, R. (2020). Analysis of existing methods for searching object in the video stream. In Abstracts of VI International Scientific and Practical Conference «About the problems of science and practice, tasks and ways to solve them» (pp. 500-505).

29. Tvoroshenko, I., & Tkachenko, D. (2020). Mechanisms of image classification based on descriptors of local features. In Abstracts of IV International Scientific and Practical Conference «Integration of scientific bases into practice» (pp. 443-448).

30. Daradkeh, Y. I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L. A., & Ahmad, N. (2021). Development of effective methods for structural image recognition using the principles of data granulation and apparatus of fuzzy logic. IEEE Access, 9, 13417-13428.

31. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Al-Dhaifallah, M. (2022). Classification of images based on a system of hierarchical features. Computers, Materials & Continua, 72(1), 1785-1797.

32. Gorokhovatskyi, V., Tvoroshenko, I., & Yakovleva, O. (2024). Transforming image descriptions as a set of descriptors to construct classification features. Indonesian Journal of Electrical Engineering and Computer Science, 33(1), 113-125.

33. Tvoroshenko, I. S., & Gorokhovatsky, V. O. (2019). Intelligent classification of biophysical system states using fuzzy interval logic. Telecommunications and Radio Engineering, 78(14), 1303-1315.

34. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for making reliable decisions on a variety of effective factors using fuzzy logic. International Journal of Advanced Computer Science and Applications, 11(5), 43-50.

35. Tvoroshenko, I. S., & Gorokhovatsky, V. O. (2020). Effective tuning of membership function parameters in fuzzy systems based on multi-valued interval logic. Telecommunications and Radio Engineering, 79(2), 149-163.

36. Tvoroshenko, I. S., & Gorokhovatsky, V. O. (2019). Modification of the branch and bound method to determine the extremes of membership functions in fuzzy intelligent systems. Telecommunications and Radio Engineering, 78(20), 1857-1868.

37. Творошенко, І. С. (2021). Технології прийняття рішень в інформаційних системах: Навчальний посібник. ХНУРЕ.

38. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент: Монографія. ХНУРЕ.

39. Гороховатський, В. О., Творошенко, І. С., & Сидоренко, Д. В. (2021). Класифікація зображень із використанням кластерного подання. In Інтелектуальні рішення-С. Обчислювальний інтелект (результати, проблеми, перспективи) (pp. 44-45).