ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра інформаційних технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Розробка веб-платформи для відеотрансляцій із підтримкою інтерактивного чату в реальному часі»

«Development of a web platform for video broadcasting with support for interactive real-time chat»

Виконав: здобувач денної форми навчання

спеціальності 122 Комп'ютерні науки .

(код, назва спеціальності)

Освітня програма Комп'ютерні науки .

(назва)

ШАЛАПСЬКИЙ Іван Андрійович

(прізвище, ім'я, по-батькові здобувача)

Керівник ФРАЗЕ-ФРАЗЕНКО О.О., к.т.н., доц. _____

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент ДОМАСКІН О.М. к.т.н., Начальник ЦІТ ОНЕУ

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій .
№ ___ від ___ _____ 2026 р.

Завідувачка кафедри

_____ КАЗАКОВА Надія

(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № 4
протокол № ___ від _____ 2026р.

Оцінка _____ / _____ / _____.

(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ КОПИЧЕНКО Іван

(підпис) (прізвище,ім'я)

Одеса 2026

АНОТАЦІЯ

Актуальність роботи зумовлена стрімким зростанням популярності сервісів живих відеотрансляцій (стрімінгу) та потребою у власних масштабованих рішеннях, які поєднують доставлення відео у реальному часі, інтерактивну взаємодію глядачів через чат і механізми монетизації контенту. Створення такої платформи вимагає сучасної серверної архітектури, здатної одночасно обслуговувати велику кількість підключень, гарантувати безпеку даних користувачів та забезпечувати мінімальні затримки доставлення повідомлень.

Мета роботи – спроєктувати та програмно реалізувати серверну частину веб-платформи, яка забезпечує реєстрацію й автентифікацію користувачів, проведення відеотрансляцій, обмін повідомленнями у чаті в реальному часі, систему підписок і монетизації, а також надсилання сповіщень.

Для досягнення мети в роботі вирішено такі завдання: проведено аналіз предметної області та аналогів; обґрунтовано вибір технологій і засобів розробки; визначено функціональні вимоги; виконано проєктування архітектури, бази даних і динаміки системи з використанням мови UML; здійснено програмну реалізацію серверної частини; описано тестування системи.

Практична цінність роботи полягає у створенні працездатної серверної частини веб-платформи відеотрансляцій, яку можна використовувати як основу для повноцінного стрімінгового сервісу та яка демонструє застосування сучасного стека технологій серверної розробки.

ABSTRACT

The relevance of the work is due to the rapid growth in the popularity of live video broadcasting (streaming) services and the need for their own scalable solutions that combine real-time video delivery, interactive viewer interaction via chat, and content monetization mechanisms. Creating such a platform requires a modern server architecture capable of simultaneously servicing a large number of connections, guaranteeing the security of user data, and ensuring minimal message delivery delays.

The purpose of the work is to design and programmatically implement the server part of a web platform that provides user registration and authentication, video broadcasting, real-time chat messaging, a subscription and monetization system, and sending notifications.

To achieve the goal, the following tasks were solved in the work: an analysis of the subject area and analogues was performed; the choice of technologies and development tools was justified; functional requirements were determined; the design of the architecture, database, and system dynamics using the UML language was performed; the software implementation of the server part has been carried out; the system testing is described.

The practical value of the work lies in the creation of a functional server part of a web video broadcasting platform, which can be used as the basis for a full-fledged streaming service and which demonstrates the use of a modern stack of server development technologies.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ,УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ 5

ВСТУП 6

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ 8

1.1 Дослідження предмету та особливостей розробки веб-платформи відеотрансляцій 8

1.2 Огляд та аналіз можливостей середовищ розробки 11

1.3 Огляд основних можливостей мов програмування 13

2 ОСНОВНІ МЕТОДИ ТА ОСОБЛИВОСТІ ТЕХНОЛОГІЙ РОЗРОБКИ 18

2.1 Огляд основних компонентів та життєвого циклу застосунку 18

2.2 Дослідження асинхронної моделі виконання та обробки запитів 20

2.3 Застосування механізму реального часу 23

2.4 Технологія потокового відео та механізми безпеки 25

3 ПРОЄКТУВАННЯ ВЕБ-ПЛАТФОРМИ 28

3.1 Вибір та обґрунтування архітектури системи 28

3.2 Проєктування функцій та взаємодії системи 31

3.3 Проєктування бази даних 39

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ 43

4.1 Розробка структури проєкту та підсистеми автентифікації 43

4.2 Розробка сервісів для роботи з даними та трансляціями 45

4.3 Реалізація чату реального часу та системи подій 47

4.4 Тестування програмного забезпечення 48

ВИСНОВКИ 50

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52

ПЕРЕЛІК СКОРОЧЕНЬ,УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – програмний інтерфейс застосунку;
CRUD (Create, Read, Update, Delete) – базові операції над даними;
DI (Dependency Injection) – впровадження залежностей;
DTO (Data Transfer Object) – об'єкт передавання даних;
GraphQL (Graph Query Language) – мова запитів до API та середовище виконання; HTTP / HTTPS – протокол передавання гіпертексту (захище-ний);
JSON (JavaScript Object Notation) – текстовий формат обміну даними;
ORM (Object-Relational Mapping) – об'єктно-реляційне відображення
RTMP (Real-Time Messaging Protocol) – протокол передавання медіапо-току;
SDK (Software Development Kit) – набір засобів розробки;
SFU (Selective Forwarding Unit) – сервер вибіркового пересилання медіапотоків;
SQL (Structured Query Language) – мова структурованих запитів;
TOTP (Time-based One-Time Password) – одноразовий пароль на основі часу;
UML (Unified Modeling Language) – уніфікована мова моделювання;
WebRTC (Web Real-Time Communication) – технологія зв'язку у реально-му часі;
WS (WebSocket) – протокол двостороннього зв'язку поверх TCP;
2FA (Two-Factor Authentication) – двофакторна автентифікація;
СУБД – система управління базами даних;

ВСТУП

Розробка програмного забезпечення для мережі Інтернет відіграє все більш важливу роль у сучасному інформаційному суспільстві. Сьогодні складно уявити сферу людської діяльності, яка б не була тією чи іншою мірою пов'язана з вебтехнологіями. Особливе місце серед вебзастосунків посідають платформи, що працюють з мультимедійним контентом у реальному часі, оскільки саме вони висувають найвищі вимоги до продук-тивності, масштабованості та надійності серверної частини.

За останні роки рівень попиту на сервіси живих відеотрансляцій (стрімінгу) постійно зростає. Це дозволяє зробити висновок про те, що розробка платформ для відеотрансляцій є актуальною та доцільною. Мільйони людей щодня переглядають прямі трансляції ігор, освітніх заходів, концертів та спілкування наживо, а кількість авторів контенту, які ведуть власні канали, невпинно збільшується. Поряд із цим зростає й значення інтерактивної складової – можливості глядачів спілкуватися між собою та з автором у режимі реального часу через текстовий чат.

Процес розробки серверної частини для подібних платформ вимагає враховувати низку специфічних обмежень та особливостей, пов'язаних із характером навантаження. Цільова архітектура має значний вплив на технології, які будуть використовуватися. Можна будувати монолітні рішення або застосовувати модульний підхід, можна обирати різні моделі організації взаємодії клієнта та сервера. Усі ці рішення безпосередньо впли-вають на якість та подальшу підтримуваність продукту.

Основним призначенням створення серверної частини вебплатформи для відеотрансляцій є забезпечення централізованого, безпечного та масштабованого керування всіма процесами платформи: від реєстрації користувачів до доставлення відеопотоку та повідомлень чату. Платформа створюється для широкого кола користувачів – як для авторів, які прагнуть вести власні трансляції, так і для глядачів, які бажають переглядати контент та

брати участь у житті спільноти. Окрім того, користувачі мають змогу фінансово підтримувати улюблених авторів через систему платних підписок, що є вбудованою функцією платформи.

Метою кваліфікаційної роботи є створення серверної частини веб-платформи, яка забезпечує проведення відеотрансляцій з підтримкою інтерактивного чату в реальному часі, що дозволяє користувачам отримувати швидкий доступ до контенту, спілкуватися між собою та підтримувати авторів, а авторам – вести трансляції та розвивати власні канали.

Для досягнення поставленої мети в роботі були вирішені наступні питання: проведено аналіз та дослідження технологій та засобів розробки серверних застосунків; сформульовано основні вимоги до інформаційного та програмного забезпечення; проведено проєктування структури системи, її функцій та бази даних; здійснено програмну реалізацію серверної частини веб-платформи з використанням сучасного стека технологій.

Об’єкт дослідження — процес організації та функціонування веб-сервісів для відеотрансляцій у реальному часі з можливістю взаємодії користувачів під час перегляду.

Предмет дослідження — методи, засоби та програмні компоненти розробки веб-платформи для відеотрансляцій із підтримкою інтерактивного чату в реальному часі, зокрема механізми передавання відеопотоку, обміну повідомленнями та забезпечення стабільної клієнт-серверної взаємодії.

Практична цінність кваліфікаційної роботи полягає в тому, що розроблена серверна частина забезпечує повний цикл функціонування стрімінгового сервісу та може бути використана як основа для створення повноцінної платформи відеотрансляцій усіма зацікавленими особами та організаціями, а також як навчальний приклад побудови серверних застосунків реального часу.

Робота містить 53 сторінки, 15 рисунків, 2 таблиці, 17 джерел посилань

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ

1.1 Дослідження предмету та особливостей розробки веб-платформи відеотрансляцій

В сучасному світі обсяг відеоконтенту, що споживається через мережу Інтернет, є домінувальним у загальній структурі інтернет-трафіку [1]. Серед усіх видів відеоконтенту окремо вирізняються живі відеотрансляції, оскільки вони передбачають передавання та споживання інформації одночасно з її створенням. Люди, які займаються створенням контенту, проведенням навчальних заходів, кіберспортом або просто спілкуванням з аудиторією, повинні мати зручний інструмент для організації мовлення, а їхні глядачі – зручний спосіб перегляду та взаємодії.

Відеотрансляція, або стрімінг, – це технологія безперервного передавання аудіо- та відеоданих від джерела до споживачів у режимі, наближеному до реального часу. На відміну від традиційного перегляду відео на запит, коли файл має бути повністю завантажений перед відтворенням, під час стрімінгу медіапотік розбивається на невеликі фрагменти, які послідовно передаються мережею та відтворюються по мірі надходження. Завдяки цьому глядач може почати перегляд майже миттєво та спостерігати за подіями, що відбуваються наживо, із затримкою лише в кілька секунд [11].

Інформація, що циркулює в межах платформи відеотрансляцій, є різнорідною за своєю природою. До неї належать персональні дані користувачів, дані про канали та трансляції, повідомлення чату, дані про підписки та фінансові операції. Кожен із цих видів інформації висуває власні вимоги до зберігання та обробки. Щоб бути корисною, інформація повинна бути актуальною, достовірною та доступною в потрібний момент часу. Особливо це стосується повідомлень чату та стану трансляцій, які мають оновлюватися в реальному часі.

Важливим елементом системи управління потоками інформації на платформі є технологічна інфраструктура для збору, обробки та розподілу даних. Особливе значення має взаємодія між підсистемою доставлення відео та підсистемою обміну текстовими повідомленнями, адже саме їх поєднання формує унікальний користувацький досвід інтерактивної трансляції. Система повинна враховувати як стратегічні інтереси (надійність та масштабованість), так і оперативно-тактичні потреби (мінімальна затримка доставлення повідомлень) [3].

Ключовою особливістю, що відрізняє платформу відеотрансляцій від звичайного відеохостингу, є інтерактивність. Глядач не лише переглядає відео, а й бере активну участь у трансляції: пише повідомлення у чат, реагує на події, підписується на канал та підтримує автора. Саме інтерактивний чат у реальному часі перетворює пасивний перегляд на активне спілкування та формує спільноту навколо каналу. Технічно це є найскладнішою з текстових підсистем, оскільки повідомлення мають доставлятися всім підключеним глядачам із мінімальною затримкою, а кількість одночасних учасників популярної трансляції може бути дуже великою (рис.1.1) [1].

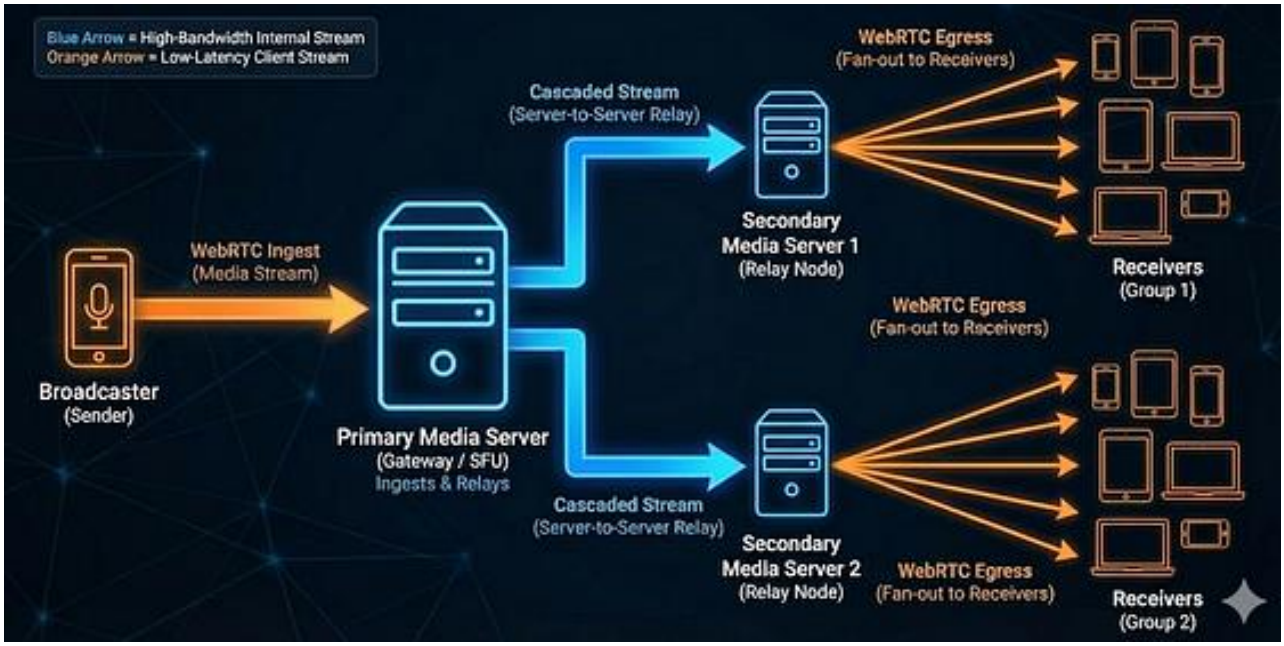


Рисунок 1.1 – Узагальнена схема передавання медіапотоку в системі відеотрансляції

Схема показує, що медіапотік надходить від автора трансляції до

серверної інфраструктури, після чого розподіляється між підключеними глядачами.

Таблиця 1.1 – Основні підсистеми веб-платформи відеотрансляцій

Підсистема	**Призначення**	**Основні функції**
Підсистема керування користувачами	Забезпечення роботи з обліковими записами користувачів	Реєстрація, автентифікація, авторизація, керування профілем користувача
Підсистема відеотрансляцій	Організація створення, запуску та завершення трансляцій	Створення трансляції, керування статусом трансляції, підключення глядачів
Підсистема інтерактивного чату	Забезпечення обміну повідомленнями між учасниками трансляції	Надсилання повідомлень, отримання повідомлень у реальному часі, зберігання історії чату
Підсистема зберігання даних	Централізоване збереження інформації про користувачів, трансляції та повідомлення	Робота з базою даних, збереження зв’язків між сутностями, підтримка цілісності даних
Підсистема безпеки	Захист доступу до функцій платформи та персональних даних	Перевірка прав доступу, захист облікових записів, контроль дій користувачів
Підсистема монетизації	Підтримка фінансової взаємодії між авторами контенту та глядачами	Підписки, платний доступ, обробка платежів
Підсистема сповіщень	Інформування користувачів про важливі події платформи	Повідомлення про початок трансляції, зміну статусу, події підписки

Як видно з таблиці 1.1, серверна частина платформи повинна забезпечувати не лише передавання відеопотоку, а й комплексне керування користувачами, даними, повідомленнями, доступом і платіжними операціями.

Проаналізувавши потреби користувачів та особливості предметної

області, можна зробити висновок, що сучасна платформа відеотрансляцій повинна поєднувати в собі кілька принципово різних підсистем: підсистему доставлення відео, підсистему обміну повідомленнями у реальному часі, підсистему зберігання даних, підсистему безпеки та автентифікації, а також підсистему монетизації. Основні переваги такого комплексного підходу перед окремими розрізненими рішеннями [2][3]:

– єдина точка керування всіма процесами платформи та узгодженість даних;

– можливість тісної інтеграції відео та чату для створення інтерактивного досвіду;

– централізоване забезпечення безпеки персональних даних та фінансових операцій;

– масштабованість та можливість обслуговування великої кількості одночасних користувачів;

– зручність подальшого розширення функціональності завдяки модульній архітектурі.

З недоліків можна відзначити лише підвищену складність розробки такої системи, що потребує застосування сучасних архітектурних підходів та спеціалізованих технологій. Саме тому ретельний вибір засобів розробки є критично важливим для успіху проєкту.

1.2 Огляд та аналіз можливостей середовищ розробки

Для розробки серверної частини веб-платформи необхідним є завдання вибору середовища розробки, що дозволить ефективно та зручно реалізувати необхідний функціонал системи. Сучасні інтегровані середовища розробки (IDE, Integrated Development Environment) надають програмісту широкий набір інструментів, що значно прискорюють процес написання, налагодження та супроводу коду.

Інтегроване середовище розробки – це комп'ютерна програма, що

допомагає програмістові розробляти нове програмне забезпечення чи модифікувати вже існуюче. Зазвичай IDE поєднує в собі текстовий редактор коду з підсвічуванням синтаксису та автодоповненням, компілятор або інтерпретатор, засоби налагодження (debugger), систему контролю версій та інструменти для тестування. Якісне середовище розробки суттєво підвищує продуктивність праці програміста та зменшує кількість помилок [2].

Першим було розглянуто середовище Visual Studio Code від компанії Microsoft. Це безкоштовний, кросплатформний та надзвичайно популярний редактор коду, який завдяки великій кількості розширень перетворюється на повноцінне середовище розробки. До переваг використання Visual Studio Code можливо віднести наступне:

– безкоштовність та відкритий код, доступність для всіх основних операційних систем;

– глибока інтеграція з мовою TypeScript, яка розроблена тією ж компанією;

– потужні засоби автодоповнення коду (IntelliSense) та навігації по проєкту;

– величезна екосистема розширень для будь-яких технологій та фреймворків;

– вбудована підтримка системи контролю версій Git та засобів налагодження;

– висока швидкодія та помірне споживання ресурсів[2].

Наступним було розглянуто середовище WebStorm від компанії JetBrains. Це потужне комерційне середовище розробки, спеціально орієнтоване на створення застосунків мовами JavaScript та TypeScript. Серед переваг використання WebStorm можливо визначити наступні:

– глибокий аналіз коду «з коробки» без потреби встановлювати додаткові розширення;

– потужні засоби рефакторингу, що дозволяють безпечно змінювати структуру коду;

– розвинені інструменти налагодження та інтеграція з тестовими фреймворками;

– зручна вбудована робота із системою контролю версій та базами даних.

Недоліками застосування WebStorm є те, що це платний продукт, який вимагає значно більше системних ресурсів порівняно з Visual Studio Code, а також може бути надмірним для невеликих проєктів. Проведений аналіз переваг та недоліків застосування середовищ розробки дозволяє здійснити вибір на користь застосування Visual Studio Code. Проаналізувавши особливості кожного із середовищ розробки, можна зробити висновок, що Visual Studio Code забезпечує оптимальне поєднання функціональності, продуктивності та зручності для розробки серверних застосунків мовою TypeScript, тому саме його обрано як основне середовище розробки.

1.3 Огляд основних можливостей мов програмування

Для розробки серверної частини веб-платформи, що здійснює проведення відеотрансляцій та обмін повідомленнями в реальному часі, необхідним є завдання вибору мови програмування, що дозволить реалізувати необхідний функціонал системи. Мова програмування значною мірою визначає не лише стиль написання коду, а й доступну екосистему бібліотек, продуктивність застосунку та зручність його супроводу [12].

Розглянутою була мова програмування JavaScript. JavaScript – це високорівнева мова програмування, яка спочатку створювалася для написання сценаріїв, що виконуються у вебпереглядачі. З появою платформи Node.js, побудованої на рушії V8, JavaScript отримав можливість виконуватися й на стороні сервера, що зробило його однією з найпоширеніших мов серверної розробки. До переваг використання мови можливо віднести наступне [3]:

– однакова мова на стороні клієнта та сервера, що спрощує розробку повноцінних застосунків;

– величезна екосистема пакетів npm

– найбільший у світі репозиторій програмних бібліотек;

– асинхронна неблокувальна модель виконання, ідеальна для мережевих застосунків;

– велика спільнота розробників та значна кількість документації й навчальних матеріалів.

Але мова програмування JavaScript має й наступні недоліки: динамічна система типів, яка, попри гнучкість, передбачає можливість появи помилок, що виявляються лише під час виконання програми; відсутність статичної перевірки типів ускладнює розробку та супровід великих проєктів зі складною структурою.

Наступною була розглянута мова програмування TypeScript. TypeScript – це сучасна типізована надбудова над мовою JavaScript, розроблена компанією Microsoft. Вона додає до JavaScript статичну систему типів, інтерфейси, перелічення та інші засоби, характерні для строго типізованих об'єктно-орієнтованих мов, водночас повністю зберігаючи сумісність із JavaScript. Перевагами використання такої мови є [13]:

– статична типізація, що дозволяє виявляти значну частину помилок ще на етапі компіляції;

– покращена читабельність коду та полегшене автодоповнення у середовищах розробки;

– зручність рефакторингу та супроводу великих проєктів зі складною доменною моделлю;

– підтримка сучасних можливостей мови та декораторів, що активно використовуються у фреймворку NestJS;

– повна сумісність із наявною екосистемою бібліотек JavaScript.

Проведений огляд основних можливостей мов програмування для здійснення розробки серверної частини веб-платформи, що забезпечить реалізацію всіх функціональних можливостей системи, дозволяє обрати як мову програмування TypeScript. Головними критеріями у виборі інструментів розробки стають надійність, зручність супроводу та продуктивність. Тому в

кваліфікаційній роботі будуть використовуватися мова програмування TypeScript та платформа Node.js, адже вони увібрали найкращі особливості та переваги сучасних інструментів для розробки серверного програмного забезпечення. [12]

В даному підрозділі було проаналізовано основні інструменти для розробки серверного програмного забезпечення. Також було доведено й обґрунтовано правильність вибору засобів для проєктування. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Visual Studio Code, а мовою програмування – TypeScript.

У ході кваліфікаційної роботи необхідно здійснити проєктування та розробити серверну частину веб-платформи, яка надає можливість проведення відеотрансляцій та обміну повідомленнями в інтерактивному чаті в реальному часі. Взаємодія з клієнтськими застосунками здійснюватиметься через єдиний програмний інтерфейс GraphQL, а робота з відеопотоком – через інтеграцію з медіасервером LiveKit [5][10].

Основним призначенням та метою розробки серверної частини веб-платформи для відеотрансляцій є забезпечення зручного та надійного програмного забезпечення для реалізації наступних функцій [1]:

– реєстрація, автентифікація та керування обліковими записами користувачів;
– проведення відеотрансляцій та керування їх станом;
– обмін повідомленнями в чаті в реальному часі;
– система підписок на канали та сповіщень;
– монетизація контенту через платні підписки.

У таблиці 1.2 подано узагальнені функціональні вимоги до серверної частини веб-платформи відеотрансляцій.

Таблиця 1.2 – Функціональні вимоги до серверної частини веб-платформи

Група вимог	**Функціональна вимога**	**Очікуваний результат**
Керування користувачами	Система повинна забезпечувати реєстрацію нових користувачів	Користувач може створити обліковий запис у системі
Керування користувачами	Система повинна підтримувати автентифікацію користувачів	Користувач отримує доступ до функцій відповідно до свого статусу
Керування користувачами	Система повинна забезпечувати редагування профілю	Користувач може змінювати власні персональні дані
Відеотрансляції	Система повинна дозволяти створення трансляції	Автор може створити нову трансляцію та підготувати її до запуску
Відеотрансляції	Система повинна підтримувати запуск і завершення трансляції	Автор може керувати життєвим циклом трансляції
Відеотрансляції	Система повинна забезпечувати підключення глядачів до трансляції	Глядач може переглядати доступну трансляцію
Інтерактивний чат	Система повинна забезпечувати надсилання повідомлень у чаті	Учасник трансляції може передати текстове повідомлення
Інтерактивний чат	Система повинна доставляти повідомлення іншим учасникам у реальному часі	Повідомлення відображається без ручного оновлення сторінки
Зберігання даних	Система повинна зберігати дані про користувачів, трансляції та повідомлення	Дані доступні для подальшого перегляду та обробки
Безпека	Система повинна обмежувати доступ до захищених операцій	Дії виконуються лише авторизованими користувачами
Монетизація	Система повинна підтримувати механізм підписок	Користувач може отримати доступ до платного функціоналу
Сповіщення	Система повинна надсилати користувачам повідомлення про важливі події	Користувач отримує інформацію про зміни, пов’язані з трансляціями або підписками

Наведені функціональні вимоги визначають мінімальний набір можливостей, необхідних для реалізації серверної частини платформи відеотрансляцій з інтерактивним чатом.

Для досягнення поставленої мети, в рамках виконання кваліфікаційної роботи необхідно вирішити наступні завдання:

– визначити основні функціональні можливості серверної частини платформи відеотрансляцій;

– здійснити дослідження та аналіз технологій та засобів розробки;

– визначити архітектурні рішення для проєктування системи;

– здійснити постановку завдання з зазначенням усіх функцій системи;

– здійснити проєктування системи із застосуванням мови UML;

– здійснити програмну реалізацію серверної частини;

– провести тестування розробленого програмного забезпечення.

На підставі проведеного дослідження предметної області визначено основні підсистеми, які повинні бути реалізовані у складі серверної частини платформи [2]. Основні підсистеми системи:

– підсистема автентифікації та облікових записів;

– підсистема профілю користувача;

– підсистема трансляцій та інтеграції з медіасервером [10];

– підсистема інтерактивного чату реального часу [9];

– підсистема підписок, сповіщень та монетизації.

Серверна частина веб-платформи для відеотрансляцій повинна забезпечувати оперативний доступ та достовірність інформації для всіх учасників інформаційного обміну в системі [14].

2 ОСНОВНІ МЕТОДИ ТА ОСОБЛИВОСТІ ТЕХНОЛОГІЙ РОЗРОБКИ

Розуміння того, як влаштовано і працює серверний застосунок, є дуже важливим для більш осмисленого проєктування та програмування. Сучасні серверні застосунки реального часу являють собою складну взаємодію між програмним кодом розробника та системними фреймворками й бібліотеками, які надають базову інфраструктуру. Системні фреймворки беруть на себе рутинні завдання – обробку мережевих з'єднань, маршрутизацію запитів, керування життєвим циклом об'єктів, – а розробник пише код, що налаштовує цю інфраструктуру під потреби конкретної предметної області. У цьому розділі послідовно розглянуто основні методи та особливості технологій, застосованих під час розроблення серверної частини веб-платформи.

2.1 Огляд основних компонентів та життєвого циклу серверного застосунку

В основі серверної частини платформи лежить фреймворк NestJS, який визначає чітку, добре структуровану архітектуру застосунку. Розуміння основних будівельних блоків та життєвого циклу застосунку є ключовим для подальшого проєктування системи [4].

Вхідною точкою кожного застосунку NestJS є функція начального завантаження (bootstrap), у якій створюється екземпляр застосунку на основі кореневого модуля. Під час запуску фреймворк аналізує дерево модулів, створює всі необхідні об'єкти-постачальники (providers), розв'язує залежності між ними та піднімає HTTP-сервер, готовий приймати запити. Розробнику не потрібно вручну керувати створенням більшості об'єктів – це бере на себе контейнер впровадження залежностей фреймворку [4].

Базовою одиницею структури застосунку NestJS є модуль (Module) – клас, позначений спеціальним декоратором, що групує пов'язані за призначенням компоненти та оголошує їхні залежності. Застосунок є деревом модулів із кореневим модулем, при цьому кожен модуль інкапсулює окрему функціональну підсистему: автентифікацію, трансляції, чат тощо. Така

організація забезпечує слабку зв'язаність підсистем та можливість їх незалежного розвитку.

Усередині модулів логіка розподілена між кількома типами компонентів відповідно до принципу розділення відповідальності:

– резолвери (Resolvers) - приймають запити, мутації та підписки GraphQL, перевіряють права доступу і делегують обробку сервісам; вони є аналогом контролерів у класичних REST-застосунках;

– сервіси (Services) - містять бізнес-логіку застосунку та є основними постачальниками, які впроваджуються в інші класи;

– об'єкти передавання даних (DTO) та вхідні типи - описують структуру вхідних даних і використовуються для їх перевірки (валідації);

– моделі (типи) - описують структуру даних, що повертаються клієнту через GraphQL [4][5].

Ключовим механізмом, на якому ґрунтується вся архітектура NestJS, є впровадження залежностей (Dependency Injection, DI). Замість того щоб класи самостійно створювали потрібні їм об'єкти, ці об'єкти створюються контейнером фреймворку та автоматично передаються через конструктор. Такий підхід зменшує зв'язаність коду, спрощує заміну реалізацій та значно полегшує модульне тестування, оскільки залежності легко підмінити імітаційними (mock) об'єктами (рис. 2.1).

Як видно з рисунка, запит від клієнта послідовно проходить через рівні системи. Спочатку шлюз GraphQL виконує синтаксичний розбір і перевірку запиту та застосовує охоронці (guards) і перехоплювачі (interceptors), що відповідають за автентифікацію, авторизацію та інші наскрізні завдання. Далі резолвер визначає, яку саме операцію потрібно виконати, та звертається до відповідного сервісу. Сервіс реалізує бізнес-логіку й, за потреби, звертається до рівня доступу до даних, який транслює виклики у запити до бази даних. Поперечні (інфраструктурні) сервіси – робота з Redis, медіасервером, платіжною системою тощо – підключаються до сервісного рівня через механізм впровадження залежностей. Така шарова організація забезпечує

чистоту коду, повторне використання компонентів та зручність супроводу системи [7][9].

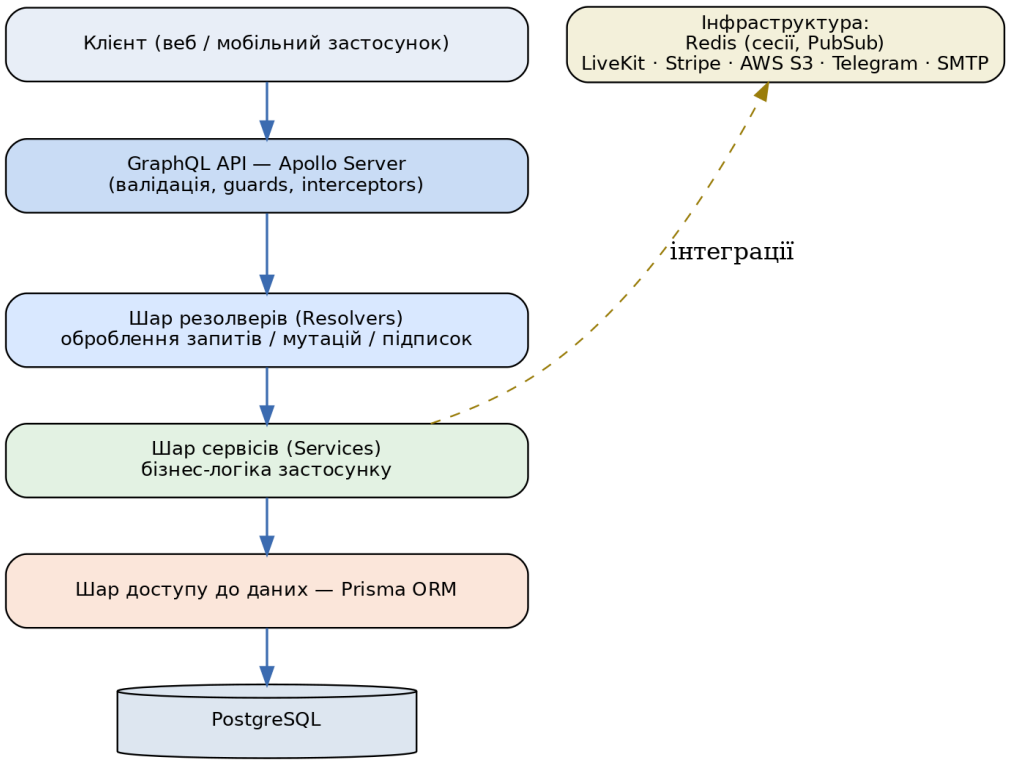


Рисунок 2.1 – Багаторівнева архітектура серверного застосунку на NestJS

2.2 Дослідження асинхронної моделі виконання та обробки запитів

Однією з найважливіших особливостей платформи Node.js, на якій ґрунтується серверна частина, є її модель виконання коду. На відміну від багатьох традиційних серверних платформ, що обробляють кожен запит в окремому потоці операційної системи, Node.js використовує однопотокову асинхронну неблокувальну модель на основі циклу подій (event loop) [12].

Сутність цієї моделі полягає в наступному. Основний код застосунку виконується в одному потоці, проте всі операції введення-виведення (звернення до бази даних, мережеві запити, читання файлів) виконуються асинхронно та не блокують цей потік. Коли застосунок ініціює таку операцію, він не очікує її завершення, а продовжує обробляти інші запити; щойно

операція завершиться, у чергу подій додається відповідна функція зворотного виклику, яку цикл подій виконає за першої нагоди.

Така поведінка гарантує, що тривала операція (наприклад, запит до бази даних) не зупиняє роботу всього сервера. Завдяки цьому один екземпляр застосунку Node.js здатний ефективно обслуговувати тисячі одночасних з'єднань за порівняно невеликого споживання ресурсів. Саме ця властивість робить Node.js особливо придатним для застосунків, орієнтованих на велику кількість одночасних мережевих з'єднань із порівняно невеликим обсягом обчислень на кожне з них, що точно відповідає профілю навантаження платформи відеотрансляцій з інтерактивним чатом [12].

Для зручної роботи з асинхронним кодом у сучасній мові TypeScript застосовується синтаксис асинхронних функцій (async/await), що дозволяє писати асинхронний код у послідовному, легко читабельному стилі, уникаючи так званого «пекла зворотних викликів» (callback hell). Розробник позначає функцію як асинхронну та використовує оператор очікування там, де потрібно дочекатися результату операції; при цьому потік виконання не блокується, а керування повертається до циклу подій [13].

Важливо враховувати, що оскільки основний код виконується в одному потоці, тривалі синхронні обчислення можуть заблокувати цикл подій та погіршити чутливість сервера. Тому ресурсомісткі операції (наприклад, оброблення зображень) виносяться у спеціалізовані бібліотеки, що виконують роботу поза основним потоком, а періодичні фонові завдання обробляються планувальником. Розуміння та правильне застосування асинхронної моделі є запорукою високої продуктивності серверної частини [12].

Для роботи зі збереженими даними в серверній частині платформи застосовано технологію об'єктно-реляційного відображення (ORM) на основі бібліотеки Prisma у поєднанні з реляційною СУБД PostgreSQL. Бібліотека Prisma призначена для застосунків, у яких модель даних є високо структурованою, та дозволяє працювати з базою даних не через прямі SQL-запити, а через типобезпечний програмний інтерфейс [7][8].

Замість визначення структур даних суто програмно, у Prisma використовується декларативний опис схеми у спеціальному файлі. У цьому файлі описуються моделі даних, їхні поля, типи та зв'язки між собою. Під час розробки на основі цього опису автоматично генерується клієнт Prisma – набір функцій для виконання операцій над даними, повністю узгоджений із типами мови TypeScript. Завдяки цьому помилки звернення до неіснуючих полів або невідповідності типів виявляються ще на етапі компіляції, а не під час виконання програми.

Prisma приховує більшість деталей роботи зі сховищем даних, дозволяючи розробнику зосередитися на тому, що дійсно робить застосунок унікальним. Керуючи моделлю даних застосунку, бібліотека істотно скорочує кількість коду, який необхідно написати. Фреймворк Prisma надає, зокрема, такі можливості [7]:

– типобезпечні операції створення, читання, оновлення та видалення (CRUD) над усіма моделями даних без написання SQL уручну;

– декларативний опис зв'язків між сутностями: один-до-одного, один-до-багатьох та багато-до-багатьох;

– автоматичне відображення імен полів програмної моделі у назви стовпців бази даних;

– механізм міграцій та синхронізації схеми бази даних із її описом;

– вбудований захист від SQL-ін'єкцій завдяки використанню параметризованих запитів;

– зручний візуальний інструмент для перегляду та редагування даних під час розробки.

Як сховище даних обрано реляційну СУБД PostgreSQL – надійну систему з відкритим кодом, що підтримує транзакції, зовнішні ключі та забезпечує цілісність даних. Поєднання PostgreSQL та Prisma дає змогу отримати водночас надійність перевіреної СУБД та зручність і безпеку сучасного типобезпечного інструменту доступу до даних. Сукупність моделей, описаних у схемі, формує доменну модель платформи: користувачі,

трансляції, повідомлення чату, категорії, підписки, сповіщення, плани спонсорства та транзакції [7][8]. Детальний опис цих сутностей та зв'язків між ними наведено у третьому розділі під час проєктування бази даних.

2.3 Застосування механізму реального часу

Найважливішою функціональною особливістю платформи є підтримка взаємодії в реальному часі – насамперед обміну повідомленнями в чаті під час трансляції. Для реалізації цього механізму застосовано сховище Redis у поєднанні з протоколом WebSocket та підписками GraphQL [9].

Redis – це високопродуктивне сховище структур даних, що зберігає інформацію в оперативній пам'яті, завдяки чому забезпечує надзвичайно низьку затримку доступу. У розробленій платформі Redis виконує дві важливі функції.

По-перше, Redis застосовується для зберігання сесій користувачів. Після успішного входу сервер створює сесію, ідентифікатор якої передається клієнту у вигляді захищеного cookie, а самі дані сесії зберігаються у Redis. Такий підхід, на відміну від збереження всього стану лише на боці клієнта, дозволяє серверу будь-якої миті примусово завершити сесію, зберігає контроль над активними входами та добре масштабується за наявності кількох екземплярів сервера.

По-друге, Redis є основою механізму публікації-підписки (publish/subscribe), що забезпечує роботу подій реального часу. Коли в системі відбувається подія – наприклад, надходить нове повідомлення чату, – відповідний сервіс публікує її у визначений канал Redis, а всі екземпляри застосунку, до яких підключені зацікавлені клієнти, отримують цю подію та розсилають її через активні з'єднання. Передавання повідомлень клієнтам здійснюється поверх протоколу WebSocket, що, на відміну від звичайного HTTP, забезпечує стійке двостороннє з'єднання між клієнтом і сервером. У термінах GraphQL цей механізм реалізується через підписки (Subscriptions).



Рисунок 2.2 – Механізм публікації-підписки (PubSub) для доставлення повідомлень чату

Застосування Redis як шини подій є ключовим для масштабованості платформи. Воно дозволяє розгортати кілька екземплярів серверного застосунку, при цьому повідомлення, опубліковане на одному екземплярі, коректно доставляється глядачам, підключеним до інших екземплярів. Без спільної шини подій горизонтальне масштабування чату реального часу було б неможливим, оскільки кожен екземпляр сервера «бачив» би лише власні підключення. Таким чином, поєднання Redis, WebSocket та підписок GraphQL забезпечує надійну та масштабовану доставку подій реального часу всім учасникам трансляції.

Взаємодія клієнтських застосунків із серверною частиною організована за допомогою технології GraphQL замість традиційного підходу REST. GraphQL – це мова запитів до API та середовище їх виконання, що було розроблено для подолання низки обмежень, властивих архітектурі REST [10].

Основна ідея GraphQL полягає в тому, що клієнт сам описує у запиті, які саме дані та зв'язані сутності йому потрібні, а сервер повертає рівно ці дані одним запитом до єдиної кінцевої точки. Це усуває дві поширені проблеми REST: надлишкове вибирання даних (over-fetching), коли сервер повертає більше інформації, ніж потрібно клієнту, та недостатнє вибирання (under-fetching), коли для отримання повної картини доводиться виконувати багато послідовних запитів до різних кінцевих точок. Для застосунку зі складною доменною моделлю, де сутності тісно пов'язані між собою, ці переваги є особливо цінними [10][11].

GraphQL визначає три типи операцій:

– запити (Query)

– для читання даних із сервера;

– мутації (Mutation)

– для створення, зміни або видалення даних;

– підписки (Subscription)

– для отримання потоку подій від сервера в реальному часі.

Саме вбудована в GraphQL підтримка підписок робить його природним вибором для платформи відеотрансляцій, оскільки дозволяє елегантно реалізувати доставлення повідомлень чату та сповіщень поверх єдиного, узгодженого інтерфейсу.

У роботі застосовано підхід «code-first», за якого схема GraphQL автоматично генерується із класів TypeScript, позначених відповідними декораторами. Це гарантує узгодженість між типами мови програмування та схемою API і виключає необхідність вручну підтримувати окремий опис схеми, що зменшує ймовірність помилок. Серверну реалізацію GraphQL побудовано на основі сервера Apollo, інтегрованого у фреймворк NestJS. Apollo Server бере на себе синтаксичний розбір запитів, їх перевірку щодо схеми, виконання резолверів та формування відповіді у форматі JSON, а також надає зручні засоби для розробки, тестування та документування API. Таким чином, поєднання GraphQL та сервера Apollo забезпечує гнучкий, продуктивний та зручний у супроводі інтерфейс взаємодії між клієнтом і сервером.

2.4 Технологія потокового відео та механізми безпеки

Найскладнішою у технічному плані частиною платформи є робота з відеопотоком. Для її реалізації застосовано медіасервер LiveKit з відкритим кодом, побудований на технології WebRTC. LiveKit бере на себе приймання вхідного відеопотоку від стрімера та його ретрансляцію глядачам, реалізуючи архітектуру SFU (Selective Forwarding Unit).

За архітектури SFU сервер приймає один медіапотік від мовника та вибірково пересилає його всім підключеним глядачам, не виконуючи

ресурсомісткого перекодування для кожного з них окремо. Це забезпечує низьку затримку та ефективне використання ресурсів сервера порівняно з альтернативними підходами, за яких сервер змішує або повністю перекодовує потоки (рисунок 2.3).

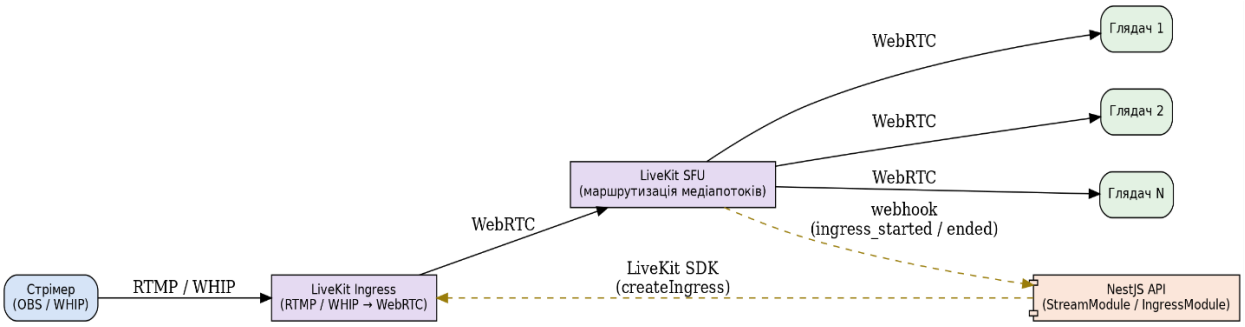


Рисунок 2.3 – Схема доставлення відеопотоку через медіасервер LiveKit

Для початку трансляції серверний застосунок за допомогою серверного SDK LiveKit створює так званий Ingress – точку приймання вхідного потоку. У відповідь LiveKit повертає адресу сервера та ключ трансляції, які стрімер вводить у програму мовлення, наприклад OBS Studio. Підтримуються протоколи RTMP – поширений стандарт для стрімінгових програм – та WHIP, сучасний протокол на основі WebRTC. Після початку надходження потоку LiveKit надсилає серверному застосунку службове повідомлення (webhook), на основі якого система позначає трансляцію як активну та сповіщає підписників. Використання готового медіасервера дозволяє зосередити зусилля на бізнес-логіці платформи, не реалізуючи самостійно складні низькорівневі механізми кодування та мережевого передавання медіа.

Окрім роботи з відео, повноцінна платформа взаємодіє з низкою зовнішніх сервісів. Платіжна система Stripe використовується для монетизації та бере на себе безпечне оброблення платіжних даних, створення продуктів і періодичних підписок. Месенджер Telegram застосовується як додатковий канал сповіщень та альтернативний спосіб входу. Хмарне об'єктне сховище застосовується для зберігання медіафайлів – аватарів користувачів та мініатюр

трансляцій, які попередньо обробляються (зміна розміру, оптимізація) перед збереженням. Транзакційна електронна пошта застосовується для надсилання листів підтвердження реєстрації та відновлення пароля.

Оскільки платформа працює з персональними даними та фінансовими операціями, питанням безпеки приділено особливу увагу. Реалізовано такі основні механізми захисту. Паролі користувачів ніколи не зберігаються у відкритому вигляді – перед збереженням пароль перетворюється на незворотний хеш за допомогою сучасного алгоритму Argon2, стійкого до атак з використанням спеціалізованого обладнання. Для додаткового захисту облікового запису реалізовано двофакторну автентифікацію на основі одноразових паролів TOTP (рисунок 2.4).

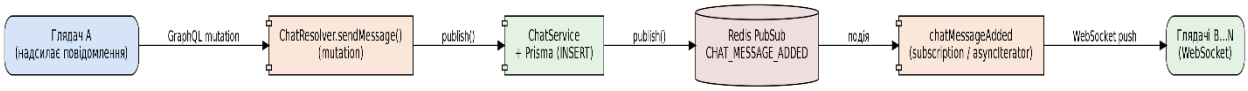


Рисунок 2.4 – Схема двофакторної автентифікації на основі TOTP

За цією схемою сервер генерує секретний ключ, формує на його основі QR-код, який користувач сканує застосунком-автентифікатором. Після цього під час входу, окрім пароля, потрібно вводити шестизначний код, що змінюється кожні 30 секунд. Підтвердження електронної пошти та відновлення пароля реалізовано через одноразові токени з обмеженим терміном дії. Стан автентифікації зберігається у серверних сесіях, а клієнту передається лише захищений ідентифікатор сесії, що дозволяє надійно завершувати сесії та контролювати активні входи.

3 ПРОЄКТУВАННЯ ВЕБ-ПЛАТФОРМИ

Проєктування є ключовим етапом розроблення програмного забезпечення, на якому формуються структура майбутньої системи, її поведінка та модель даних. Помилки, допущені на цьому етапі, обходяться значно дорожче за помилки, виявлені під час написання коду, тому проєктуванню приділено особливу увагу. У цьому розділі спроєктовано серверну частину платформи із застосуванням уніфікованої мови моделювання UML (Unified Modeling Language) – стандартної графічної нотації для візуалізації, специфікації та документування програмних систем. Мова UML дозволяє описати систему з різних точок зору за допомогою набору взаємодоповнювальних діаграм: одні з них відображають статичну структуру системи, інші – її динамічну поведінку. Послідовно розглянуто загальну архітектуру системи, модель функціональних вимог, динаміку взаємодії компонентів та модель бази даних.

3.1 Вибір та обґрунтування архітектури системи

Перш ніж переходити до детального проєктування окремих підсистем, необхідно обрати та обґрунтувати загальний архітектурний підхід до побудови системи. Від цього вибору залежать масштабованість, надійність та зручність подальшого супроводу платформи.

Сьогодні існує два основні підходи до побудови серверних застосунків: монолітна архітектура та мікросервісна архітектура. За монолітного підходу весь застосунок є єдиним розгортуваним модулем, тоді як за мікросервісного система розкладається на множину незалежних сервісів, що взаємодіють мережею. Кожен підхід має свої переваги та недоліки. Мікросервіси забезпечують гнучке незалежне масштабування окремих частин, проте суттєво ускладнюють розробку, розгортання та налагодження, потребують складної інфраструктури та вносять накладні витрати на мережеву взаємодію.

Для розроблюваної платформи обрано архітектуру модульного моноліту, побудованого на фреймворку NestJS. Цей підхід поєднує переваги обох рішень: з одного боку, застосунок залишається єдиним розгортуваним модулем, що спрощує розробку та експлуатацію; з іншого боку, завдяки чіткому поділу на модулі система зберігає внутрішню структурованість та слабку зв'язаність підсистем. За потреби в майбутньому окремі модулі можуть бути виділені в самостійні сервіси без переписування всієї системи. Такий вибір є оптимальним для проєкту відповідного масштабу та забезпечує добрий баланс між простотою та можливістю розвитку.

Систему розкладено на функціональні модулі, кожен з яких відповідає за окрему підсистему та взаємодіє з іншими через чітко визначені інтерфейси. Для відображення статичної структури системи та її зв'язків із зовнішніми сервісами використано діаграму компонентів UML (рисунок 3.1).

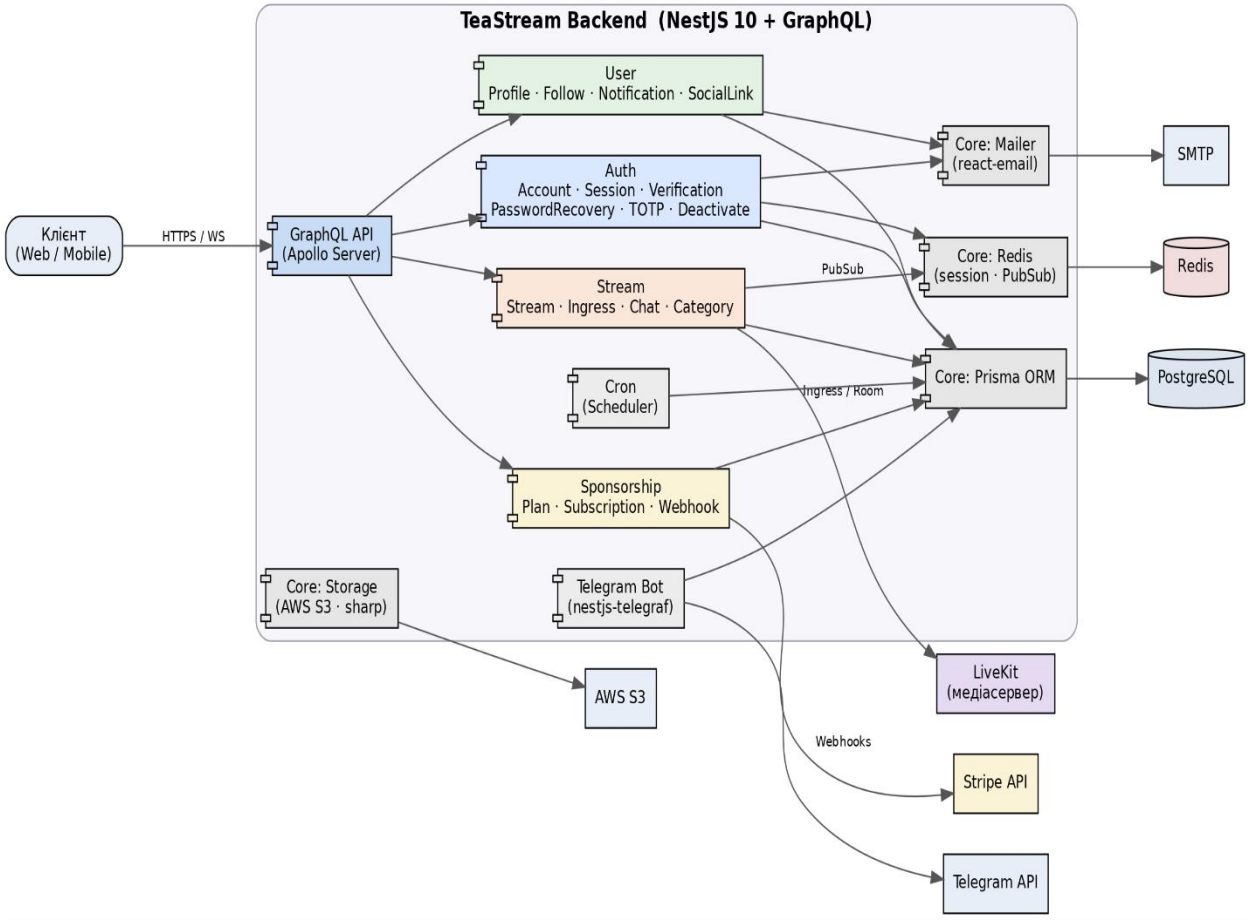


Рисунок 3.1 – Компонентна діаграма серверної частини платформи

У центрі діаграми розташовано серверний застосунок, побудований на NestJS та GraphQL. Усі запити клієнтів надходять через єдиний компонент GraphQL API (Apollo Server), який маршрутизує їх до відповідних функціональних модулів. На діаграмі виділено такі основні групи модулів:

Auth – підсистема автентифікації, що об'єднує модулі облікового запису, сесій, підтвердження пошти, відновлення пароля, двофакторної автентифікації та деактивації акаунта;

User – підсистема користувача, що містить модулі профілю, підписок, сповіщень та соціальних посилань;

Stream – підсистема трансляцій, до якої входять модулі трансляції, приймання потоку (Ingress), чату та категорій;

Sponsorship – підсистема монетизації: плани спонсорства, підписки та оброблення службових повідомлень платіжної системи;

Cron – підсистема планувальника фонових задач;

Telegram Bot – підсистема взаємодії з месенджером;

Core – інфраструктурні модулі: Prisma (доступ до даних), Redis (сесії та події), Mailer (пошта), Storage (зберігання медіа).

Праворуч на діаграмі показано зовнішні сервіси, з якими взаємодіє система: реляційна СУБД PostgreSQL, сховище Redis, медіасервер LiveKit, платіжна система Stripe, об'єктне сховище AWS S3, поштовий сервер SMTP та Telegram API. Стрілки відображають напрямки залежностей між компонентами. Така архітектура забезпечує слабку зв'язаність підсистем: зміни в одному модулі мінімально впливають на інші, а нові функції додаються підключенням нових модулів без переписування наявних.

Для опису фізичного розгортання системи та розподілу її компонентів по вузлах виконання використано діаграму розгортання (deployment diagram). Систему спроєктовано для роботи у контейнерному середовищі на основі Docker, що забезпечує однаковість оточення під час розробки та експлуатації, спрощує розгортання та масштабування (рисунок 3.2).

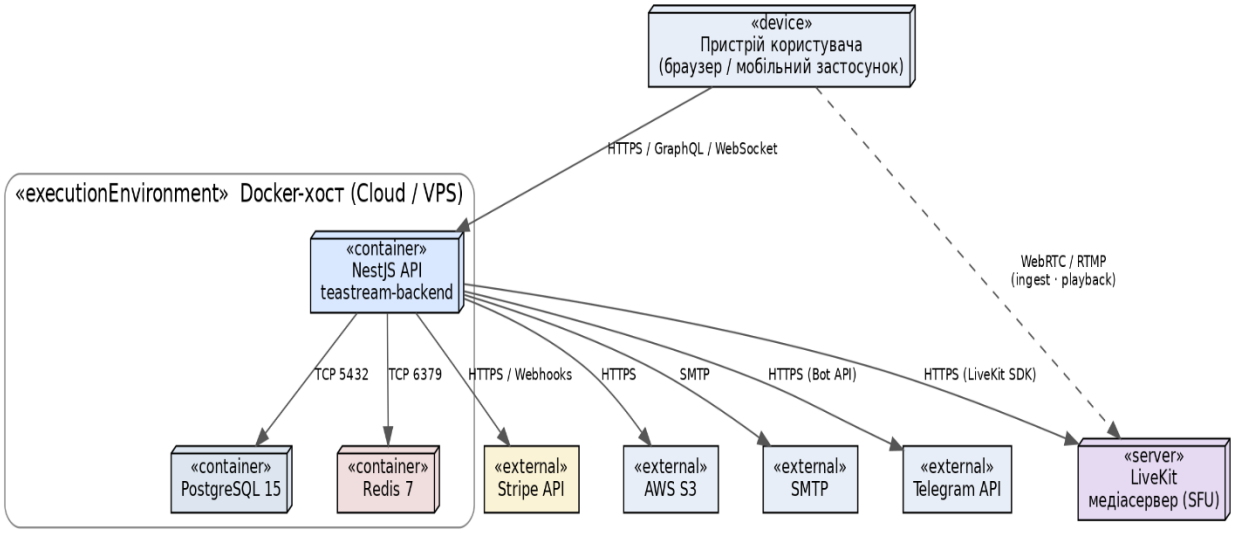


Рисунок 3.2 – Діаграма розгортання системи

На діаграмі розгортання відображено вузли системи та канали зв'язку між ними. Пристрій користувача (браузер або мобільний застосунок) взаємодіє із серверним застосунком за захищеним протоколом HTTPS, з використанням GraphQL та WebSocket для подій реального часу. Серверні компоненти – застосунок NestJS, СУБД PostgreSQL та сховище Redis – розгортаються як контейнери на хості Docker. Медіасервер LiveKit винесено в окремий вузол, оскільки робота з відео висуває специфічні вимоги до ресурсів; клієнт обмінюється з ним медіапотоком безпосередньо за протоколами WebRTC та RTMP, що зменшує навантаження на основний застосунок. Зовнішні сервіси (Stripe, AWS S3, SMTP, Telegram) взаємодіють із застосунком за протоколом HTTPS. Така схема розгортання забезпечує чіткий розподіл відповідальності між вузлами та можливість незалежного масштабування найбільш навантажених компонентів.

3.2 Проєктування функцій та взаємодії системи

Після визначення загальної архітектури необхідно формалізувати функціональні вимоги до системи та спроєктувати динаміку взаємодії її компонентів під час виконання основних сценаріїв роботи.

Для формалізації функціональних вимог із погляду користувачів побудовано діаграму прецедентів (use case diagram). Вона відображає акторів – зовнішніх стосовно системи учасників – та прецеденти (варіанти використання), тобто послідовності дій, які система виконує у відповідь на запити акторів (рисунок 3.3).

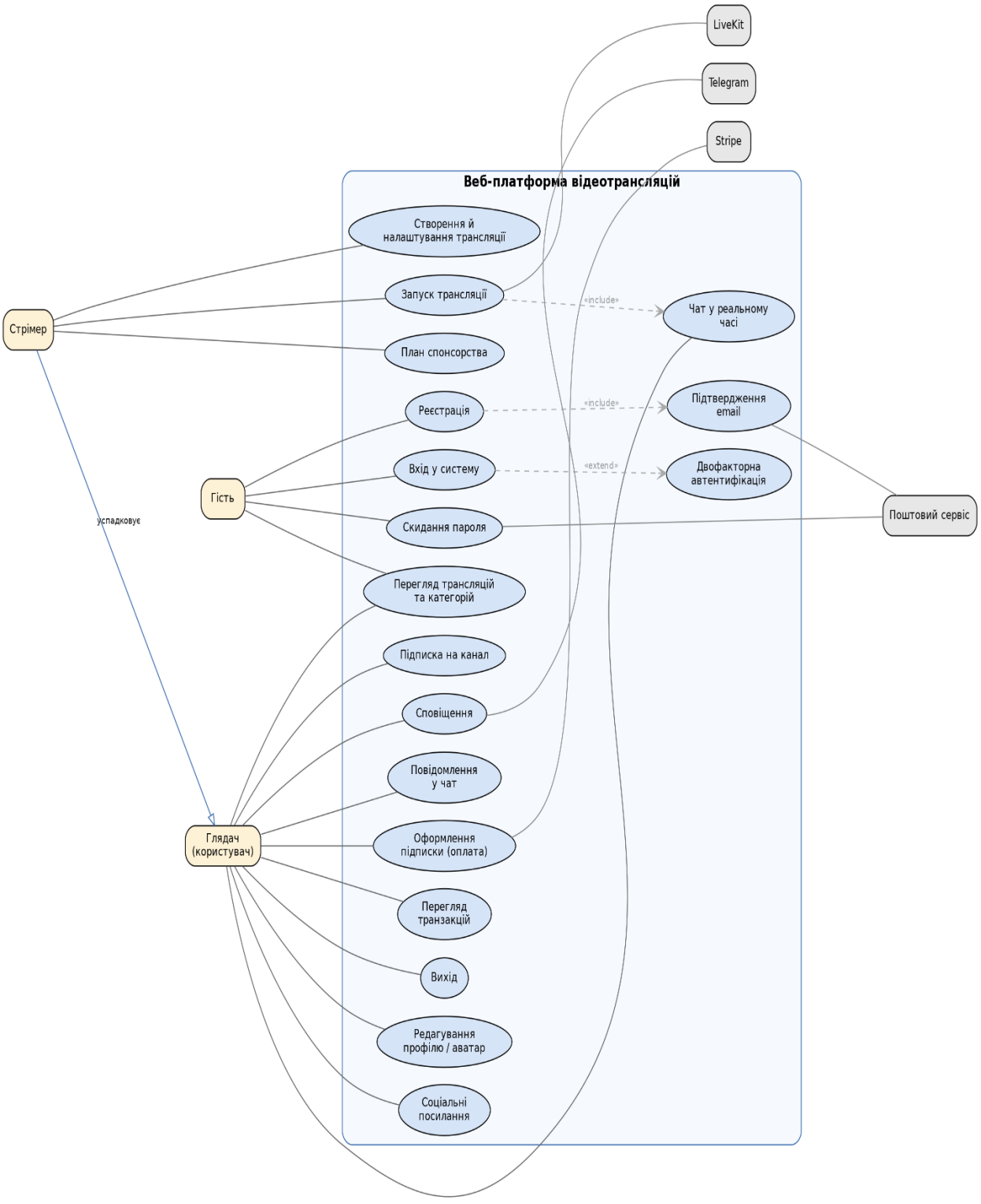


Рисунок 3.3 – Діаграма прецедентів (use case) веб-платформи

У системі визначено таких акторів-людей: **Гість** – неавтентифікований відвідувач; **Глядач** (зареєстрований користувач), який переглядає трансляції та бере участь у чаті; **Стрімер** – користувач, який, окрім можливостей глядача, проводить власні трансляції. Актор Стрімер успадковує всі прецеденти Глядача, що на діаграмі позначено відношенням узагальнення. Окрім людей, як актори виступають зовнішні системи: **Stripe** (оброблення платежів), **LiveKit** (приймання та ретрансляція відео), **Telegram** (доставлення сповіщень) і поштовий сервіс.

Гостю доступні прецеденти реєстрації, входу, відновлення пароля та перегляду каталогу трансляцій. Зареєстрований користувач отримує доступ до керування профілем, підписок на канали, участі у чаті, отримання сповіщень, оформлення платних підписок та перегляду транзакцій. Стрімеру додатково доступні створення й налаштування трансляцій, їх запуск та створення планів спонсорства. На діаграмі також відображено відношення між прецедентами. Зв'язок «include» означає обов'язкове включення одного прецеденту в інший: наприклад, прецедент «Реєстрація» включає «Підтвердження email». Зв'язок «extend» позначає необов'язкове розширення: прецедент «Двофакторна автентифікація» розширює «Вхід у систему» і застосовується лише за умови, що користувач увімкнув 2FA. Така модель прецедентів повністю охоплює сформульовані у першому розділі функціональні вимоги.

Для опису динамічної поведінки системи під час виконання ключових сценаріїв побудовано діаграми послідовності (sequence diagrams). Ці діаграми відображають обмін повідомленнями між об'єктами системи у часі та дозволяють детально специфікувати логіку обробки запитів. Розглянемо найважливіші сценарії.

Першим розглянуто сценарій реєстрації нового користувача з підтвердженням адреси електронної пошти (рисунок 3.4).

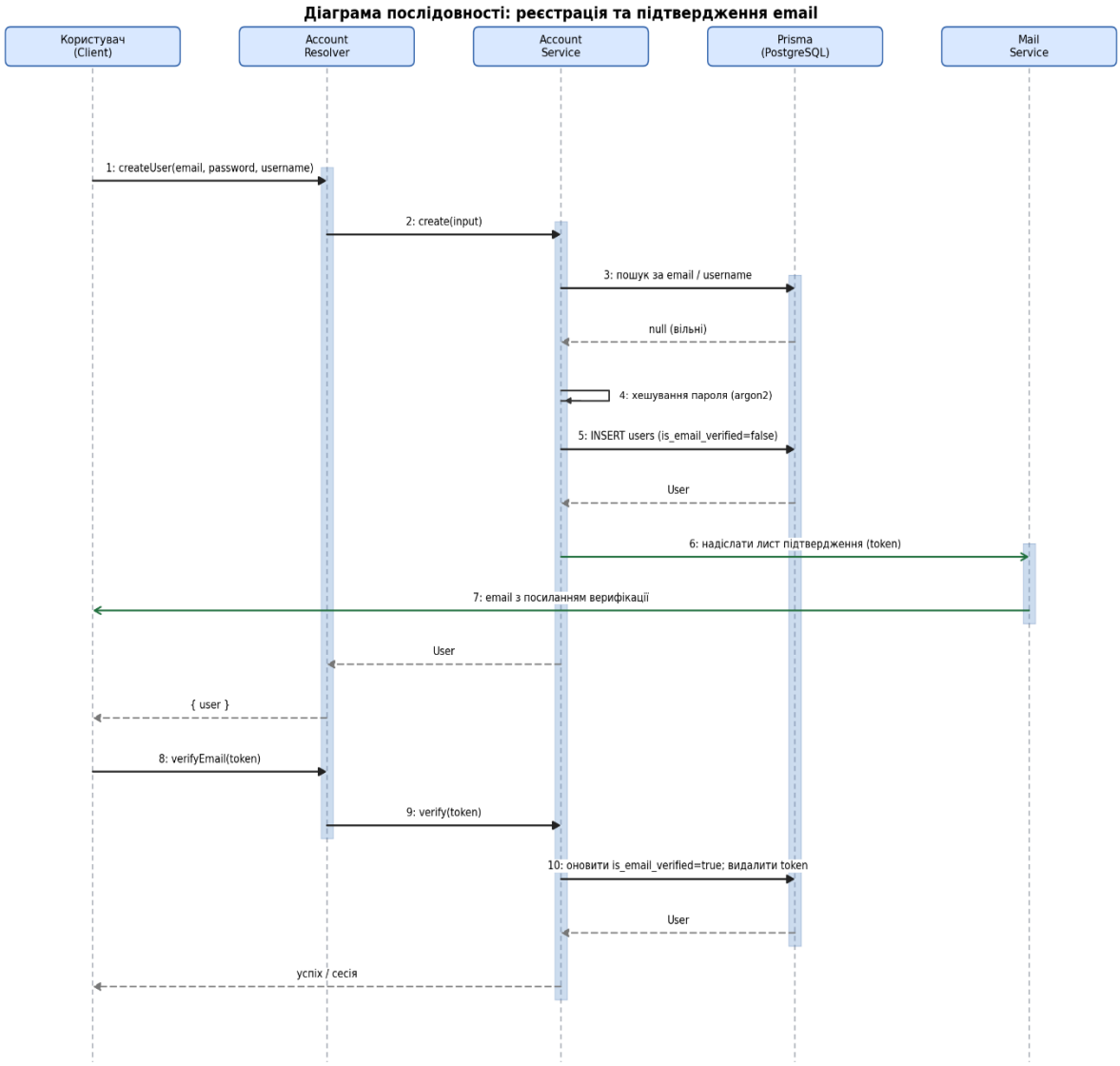


Рисунок 3.4 – Діаграма послідовності: реєстрація та підтвердження email

Як видно з діаграми, клієнт надсилає резолверу мутацію створення користувача, яка делегується сервісу облікових записів. Сервіс перевіряє унікальність адреси пошти та імені користувача, виконує хешування пароля за алгоритмом Argon2 та зберігає новий запис у базі даних із позначкою про непідтверджену пошту. Після цього сервіс надсилає на адресу користувача лист із посиланням для підтвердження. Коли користувач переходить за посиланням, виконується мутація верифікації: сервіс перевіряє чинність токена та позначає пошту як підтверджену. Така послідовність гарантує, що адреса дійсно належить користувачеві (рисунок 3.5).

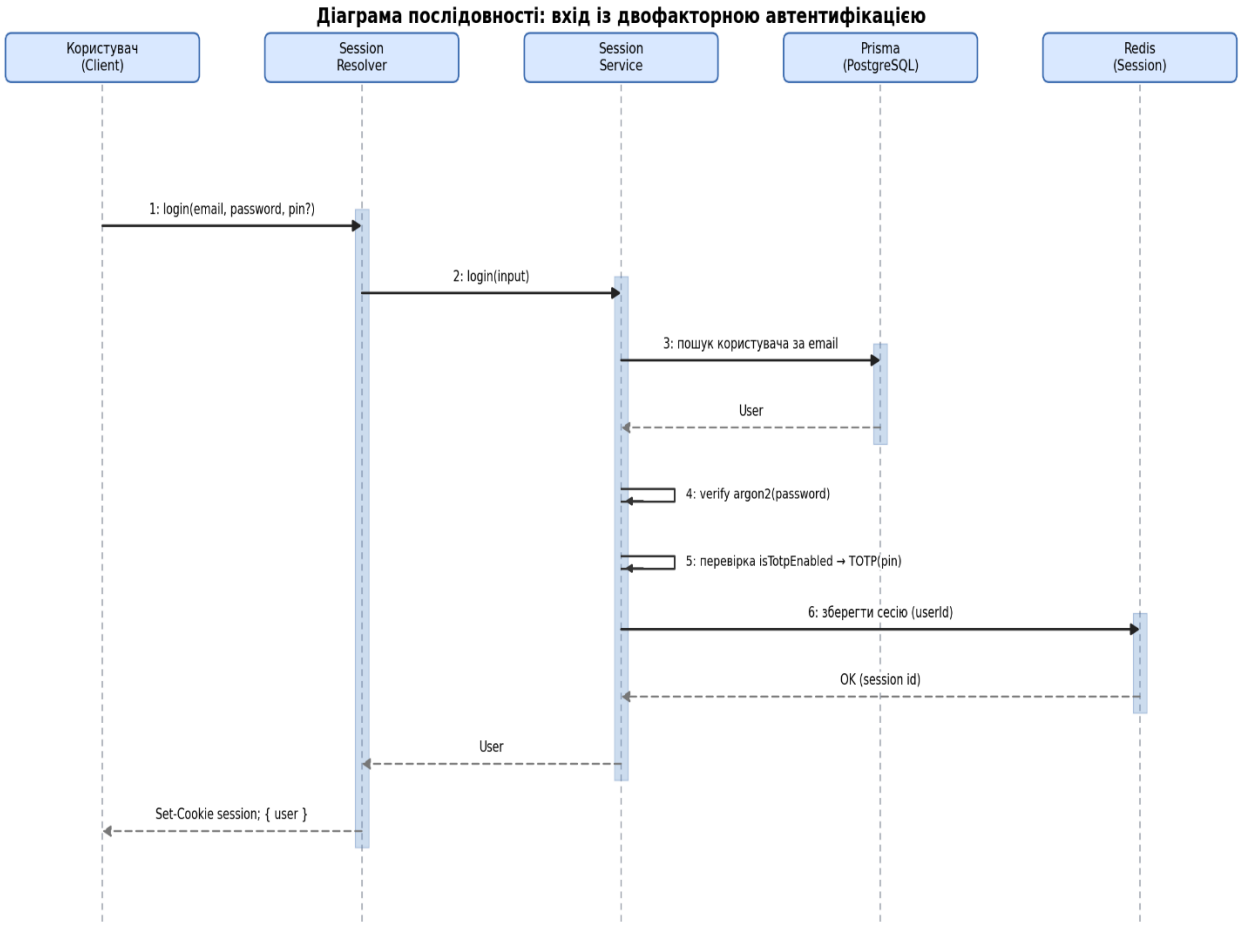


Рисунок 3.5 – Діаграма послідовності: вхід із двофакторною автентифікацією

Під час входу резолвер передає облікові дані сервісу сесій, який знаходить користувача за адресою пошти та перевіряє пароль шляхом порівняння його хешу зі збереженим значенням. Якщо для облікового запису увімкнено двофакторну автентифікацію, додатково перевіряється одноразовий код TOTP. Лише після успішного проходження всіх перевірок сервіс створює сесію, зберігає її в Redis та повертає клієнтові захищений cookie з ідентифікатором сесії. Такий багатоетапний процес забезпечує надійний захист облікових записів.

Третім, і найважливішим з погляду теми роботи, є сценарій обміну повідомленнями в чаті в реальному часі (рисунок 3.6).

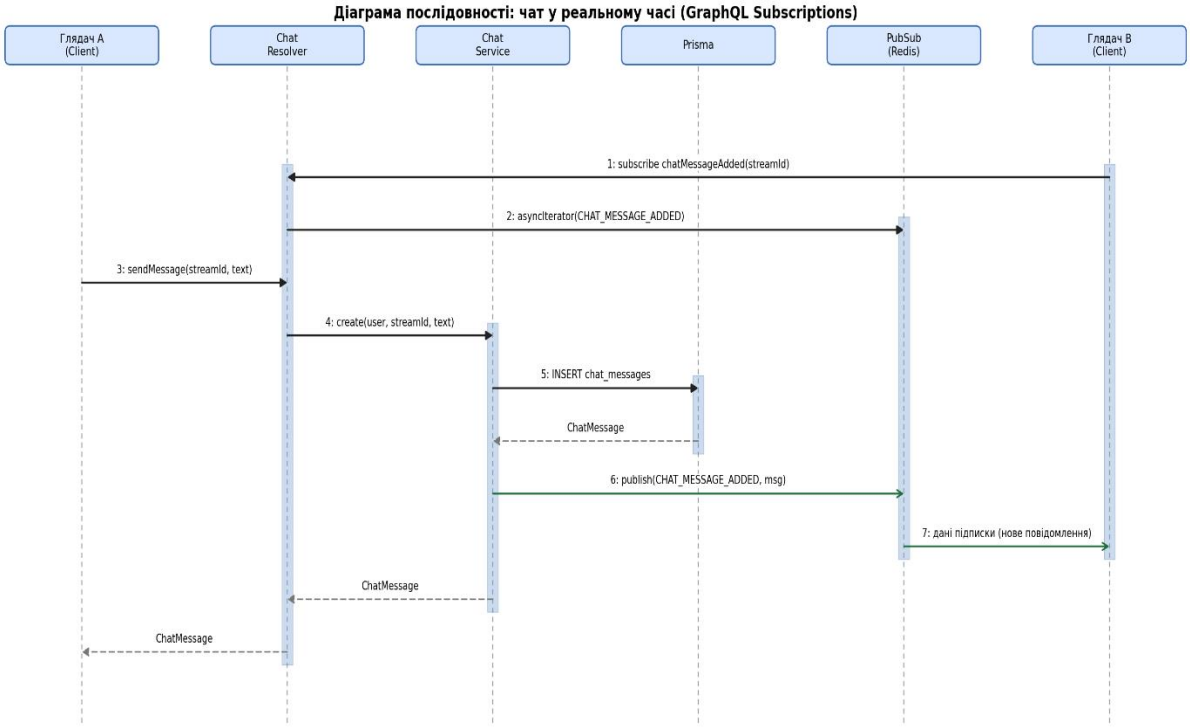


Рисунок 3.6 – Діаграма послідовності: чат у реальному часі (GraphQL Subscriptions)

Сценарій складається з двох частин. Спочатку глядач, який бажає отримувати нові повідомлення, оформлює підписку GraphQL на події конкретної трансляції; резолвер реєструє його як отримувача подій відповідного каналу. Коли інший глядач надсилає повідомлення, відповідна мутація зберігає його в базі даних та публікує подію в механізм публікації-підписки на основі Redis. Усі підписані клієнти миттєво отримують нове повідомлення через своє WebSocket-з'єднання. Саме ця схема забезпечує інтерактивність чату та мінімальну затримку доставлення повідомлень – ключову вимогу до платформи (рисунок 3.7).

Стрімер ініціює створення точки приймання потоку (Ingress); сервіс звертається до LiveKit через серверний SDK та отримує адресу сервера та ключ трансляції, які зберігає у записі трансляції й повертає стрімеру. Стрімер вводить ці дані у програму мовлення та починає передавати потік. Після початку надходження медіаданих LiveKit надсилає серверу службове

повідомлення, на основі якого трансляція позначається як активна, а підписники каналу отримують сповіщення про її початок.

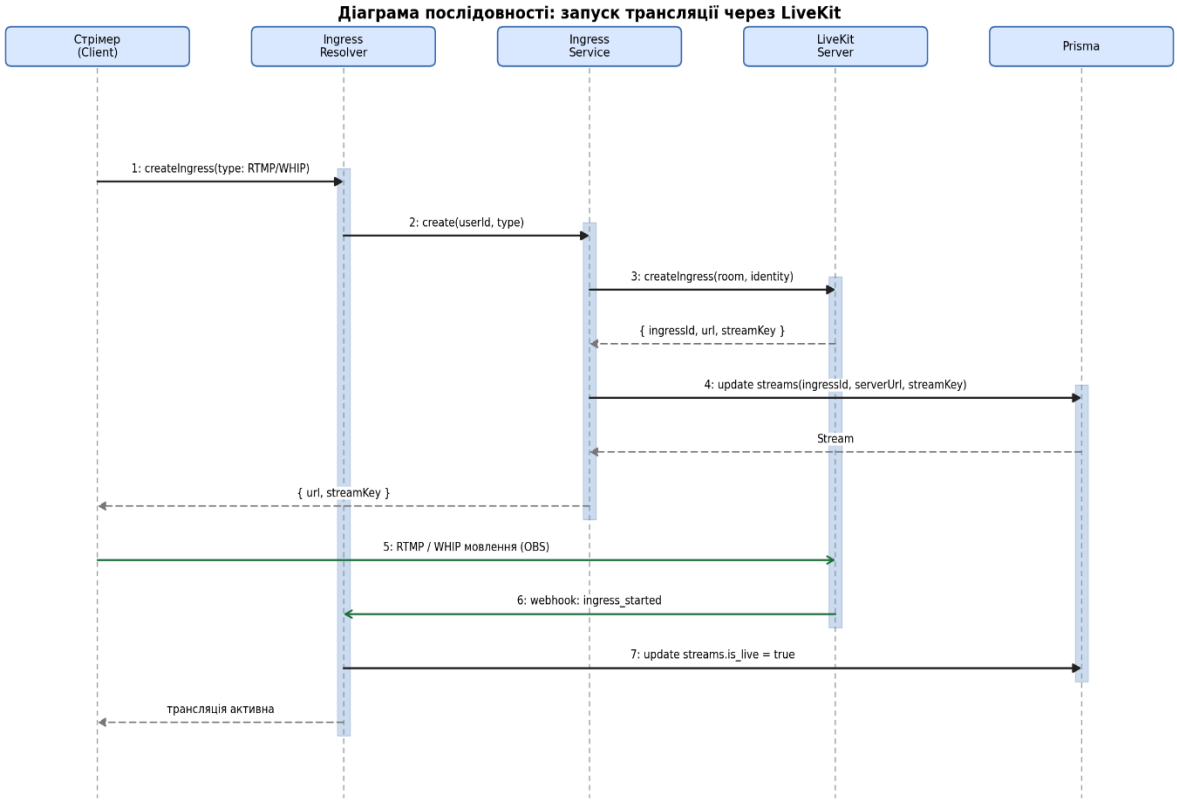


Рисунок 3.7 – Діаграма послідовності: запуск трансляції через LiveKit

Для узагальненого відображення процесу проведення трансляції як цілісного бізнес-процесу побудовано діаграму діяльності (activity diagram).На відміну від діаграм послідовності, що деталізують обмін повідомленнями між об'єктами, діаграма діяльності відображає потік керування та послідовність дій із розгалуженнями й циклами. (рисунок 3.8).

Діаграма діяльності описує повний життєвий цикл трансляції: стрімер створює трансляцію, налаштовує її параметри (категорію та режим чату), запитує точку приймання потоку та починає мовлення. Після перевірки активності потоку система позначає трансляцію як активну, розсилає сповіщення підписникам, після чого глядачі приєднуються до перегляду та беруть участь у чаті в реальному часі. Цикл обміну повідомленнями триває до завершення трансляції, після чого система знімає позначку активності та

звільняє ресурси медіасервера. Така модель наочно демонструє інтеграцію підсистем відео та чату в єдиний користувацький сценарій.



Рисунок 3.8 – Діаграма діяльності процесу проведення трансляції та чату

3.3 Проєктування бази даних

Завершальним етапом проєктування є розроблення моделі даних системи. Спочатку спроєктовано структуру класів предметної області, а потім – фізичну модель реляційної бази даних.

Для відображення статичної структури даних та зв'язків між сутностями побудовано діаграму класів (class diagram), наведену на рисунку 3.9. Діаграма класів є центральною структурною діаграмою UML і відображає класи системи, їхні атрибути, методи та відношення між собою.

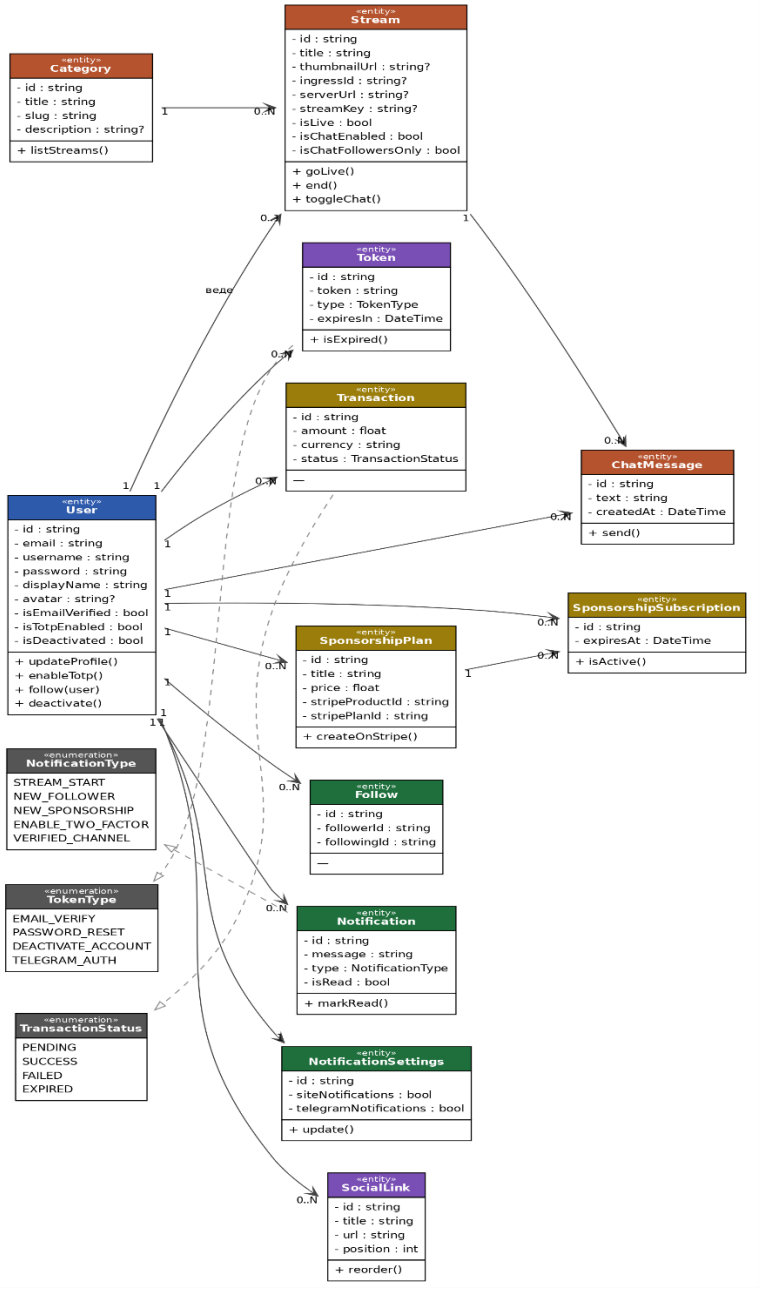


Рисунок 3.9 – Діаграма класів предметної області

На діаграмі відображено основні класи-сутності предметної області. Центральним є клас **User** (користувач), що містить дані облікового запису та пов'язаний з більшістю інших сутностей. Клас **Stream** (трансляція) описує сеанс мовлення з його параметрами та станом; клас **ChatMessage** (повідомлення чату) представляє окреме повідомлення; клас **Category** (категорія) слугує для каталогізації трансляцій. Класи **Follow** (підписка), **Notification** (сповіщення) та **NotificationSettings** (налаштування сповіщень) забезпечують соціальну складову платформи. Класи **SponsorshipPlan** (план спонсорства), **SponsorshipSubscription** (платна підписка) та **Transaction** (транзакція) реалізують монетизацію. Допоміжні класи **Token** (токен) та **SocialLink** (соціальне посилання) забезпечують верифікацію й оформлення профілю. Перелічення (enumeration) **TokenType**, **NotificationType** та **TransactionStatus** визначають допустимі значення відповідних типів. Між класами відображено асоціації з зазначенням кратності, що відповідає реальним зв'язкам предметної області.

На основі діаграми класів спроєктовано фізичну модель реляційної бази даних, відображену у вигляді діаграми «сутність-зв'язок» (ER-діаграми). Базу даних спроєктовано відповідно до вимог нормалізації, що усуває надлишковість даних та забезпечує їх цілісність (рисунок 3.10).

База даних складається з дванадцяти таблиць. Центральною є таблиця **users**, що зберігає дані облікових записів: адресу пошти, ім'я користувача, хеш пароля, відображуване ім'я, аватар, а також прапорці стану (підтвердження пошти, увімкнення двофакторної автентифікації, деактивація). З нею пов'язано більшість інших таблиць. Таблиця **streams** зберігає дані трансляцій, зокрема параметри інтеграції з LiveKit (ingress_id, server_url, stream_key) та налаштування режиму чату; вона пов'язана з користувачем відношенням «один-до-одного» та з категорією – «один-до-багатьох». Таблиця **chat_messages** зберігає повідомлення чату й пов'язана водночас із користувачем та трансляцією. Таблиця **categories** містить категорії трансляцій.

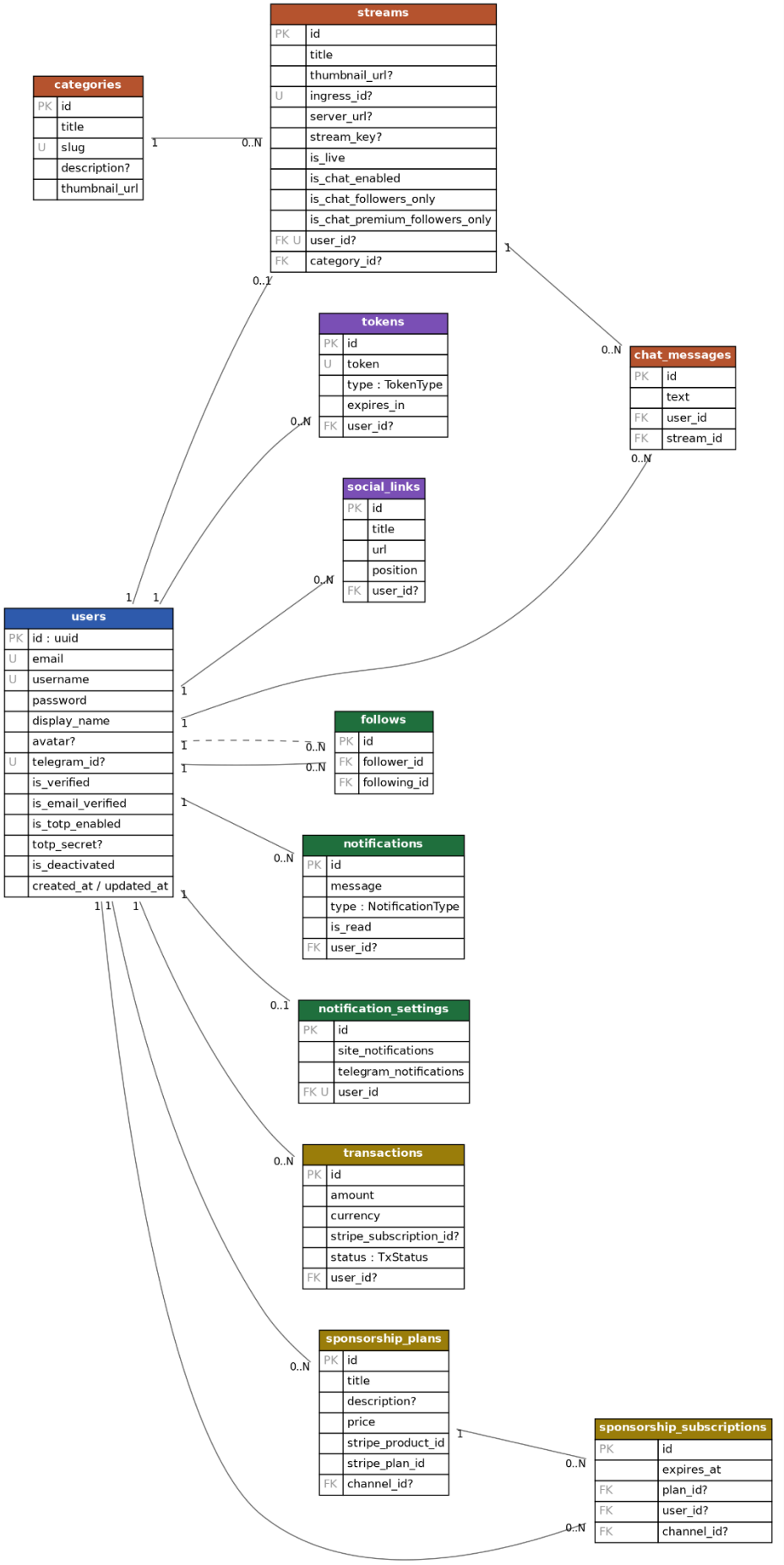


Рисунок 3.10 – Схема бази даних (ER-діаграма)

Соціальну складову реалізують таблиці **follows** (підписки між користувачами за схемою «багато-до-багатьох»), **notifications** та **notification_settings**. Підсистему безпеки обслуговує таблиця **tokens** (одноразові токени для підтвердження пошти, відновлення пароля та інших операцій), а профіль доповнює таблиця **social_links**. Монетизацію забезпечують таблиці **sponsorship_plans** (плани спонсорства, пов'язані з платіжними продуктами Stripe), **sponsorship_subscriptions** (активні платні підписки) та **transactions** (облік фінансових операцій). Зв'язки між таблицями реалізовано за допомогою зовнішніх ключів із зазначенням кратності, що гарантує цілісність даних на рівні СУБД.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

4.1 Розробка структури проєкту та підсистеми автентифікації

Структуру проєкту організовано відповідно до модульного підходу фреймворку NestJS. Увесь вихідний код розташовано в каталозі src, де він поділений на інфраструктурні модулі (core), функціональні модулі предметної області (modules) та спільні допоміжні засоби (shared). Опис схеми бази даних винесено в окремий каталог prisma.

```
src/
├── core/
│   ├── prisma/            # PrismaService, доступ до БД, seed-
скрипт
│   ├── redis/             # RedisService, провайдер PubSub
│   └── config/            # конфігурація застосунку
├── modules/
│   ├── auth/
│   │   ├── account/       # створення та керування акаунтом
│   │   ├── session/       # вхід/вихід, сесії
│   │   ├── verification/  # підтвердження email
│   │   ├── password-recovery/
│   │   ├── totp/          # двофакторна автентифікація
│   │   └── deactivate/
│   ├── stream/
│   │   ├── stream/        # трансляції
│   │   ├── ingress/       # інтеграція з LiveKit
│   │   ├── chat/          # чат реального часу
│   │   └── category/
│   ├── user/
│   │   ├── profile/  follow/  notification/
│   └── sponsorship/
│       ├── plan/  subscription/
├── shared/                # декоратори, guards, бібліотеки
(mail, storage)
├── app.module.ts          # кореневий модуль
└── main.ts                # точка входу (bootstrap)
prisma/
└── schema.prisma          # опис моделі даних
```

Лістинг 4.1 – Структура каталогів проєкту

Опис моделі даних виконано декларативно у файлі схеми Prisma. Кожна модель відповідає таблиці бази даних, а її поля – стовпцям(лістинг 4.2).

```
prisma
model User {
  id              String   @id @default(uuid())
  email           String   @unique
  password        String
  username        String   @unique
  displayName     String   @map("display_name")
  avatar          String?
  isEmailVerified Boolean  @default(false)
@map("is_email_verified")
  isTotpEnabled   Boolean  @default(false)
@map("is_totp_enabled")
  totpSecret      String?  @map("totp_secret")

  stream          Stream?
  chatMessages    ChatMessage[]
  followers       Follow[]  @relation(name: "followers")
  followings      Follow[]  @relation(name: "followings")

  createdAt       DateTime @default(now()) @map("created_at")
  updatedAt       DateTime @updatedAt @map("updated_at")

  @@map("users")
}
```

Лістинг 4.2 – Фрагмент опису моделі даних у схемі Prisma

Підсистему автентифікації реалізовано як набір модулів у складі групи auth. Розглянемо реалізацію створення облікового запису. Сервіс облікових записів перевіряє унікальність адреси пошти та імені користувача, хешує пароль за алгоритмом Argon2 і зберігає новий запис, після чого ініціює надсилання листа підтвердження (лістинг 4.3).

```
typescript
@Injectable()
export class AccountService {
  public constructor(
    private readonly prisma: PrismaService,
    private readonly verificationService: VerificationService,
  ) {}
public async create(input: CreateUserInput) {
```

```
    const { email, password, username } = input;
const isUsernameExists = await this.prisma.user.findUnique({
where: { username } });
    const isEmailExists = await this.prisma.user.findUnique({
where: { email } });
    const user = await this.prisma.user.create({
      data: {
        email,
        password: await hash(password), // хешування Argon2
        username,
        displayName: username,
      },
    });
await this.verificationService.sendVerificationToken(user);
    return user;
  }
}
```

Лістинг 4.3 – Створення облікового запису користувача

Вхід у систему реалізовано у модулі сесій. Сервіс знаходить користувача за адресою пошти, перевіряє пароль порівнянням його хешу зі збереженим значенням, за потреби перевіряє одноразовий код двофакторної автентифікації, після чого створює сесію та зберігає її в Redis. Двофакторну автентифікацію реалізовано у модулі totp із застосуванням стандарту TOTP: під час її увімкнення сервер генерує секретний ключ, на основі якого формується QR-код для застосунку-автентифікатора. Підтвердження пошти та відновлення пароля побудовано на одноразових токенах з обмеженим терміном дії, що зберігаються в таблиці токенів.

4.2 Розробка сервісів для роботи з даними та трансляціями

Доступ до бази даних в усіх модулях здійснюється через єдиний сервіс PrismaService, який інкапсулює згенерований клієнт Prisma та надається іншим класам через механізм впровадження залежностей. Такий підхід забезпечує єдину точку керування з'єднанням із базою даних та спрощує тестування.

Ключовою для платформи є реалізація трансляцій. Модуль трансляцій відповідає за створення та налаштування трансляцій, а модуль приймання потоку (ingress) – за інтеграцію з медіасервером LiveKit. Під час запуску трансляції сервіс звертається до LiveKit через серверний SDK, створює точку приймання потоку та зберігає отримані дані (адресу сервера та ключ трансляції) у записі трансляції (лістинг 4.4).

```
typescript
@Injectable()
export class IngressService {
  public constructor(
    private readonly prisma: PrismaService,
    private readonly livekit: LivekitService,
  ) {}
public async create(userId: string) {
    const ingress = await
this.livekit.ingressClient.createIngress(
      IngressInput.RTMP_INPUT,
      {
        name: userId,
        roomName: userId,
        participantIdentity: userId,
      },
    );

    await this.prisma.stream.update({
      where: { userId },
      data: {
        ingressId: ingress.ingressId,
        serverUrl: ingress.url,
        streamKey: ingress.streamKey,
      },
    });
return true;
  }
}
```

Лістинг 4.4 – Створення точки приймання потоку через LiveKit

Після того як стрімер починає мовлення, медіасервер LiveKit надсилає серверу службове повідомлення (webhook), яке обробляється окремим контролером. На його основі трансляція позначається як активна (поле isLive встановлюється у значення «істина»), а підписники каналу отримують

сповіщення про початок трансляції. Аналогічно реалізовано й інші сервіси предметної області – профілю користувача, підписок, категорій, сповіщень та монетизації, – кожен з яких інкапсулює відповідну бізнес-логіку та працює з базою даних через PrismaService.

4.3 Реалізація чату реального часу та системи подій

Найважливішою функціональною особливістю платформи є інтерактивний чат у реальному часі. Його реалізовано із застосуванням підписок GraphQL та механізму публікації-підписки на основі Redis. Резолвер чату містить мутацію надсилання повідомлення та підписку на нові повідомлення конкретної трансляції (лістинг 4.5).

```
@Resolver('Chat')
export class ChatResolver {
  public constructor(
    private readonly chatService: ChatService,
    @Inject('PUB_SUB') private readonly pubSub: PubSub,
  ) {}

  @Mutation(() => ChatMessageModel)
  public async sendMessage(
    @Args('streamId') streamId: string,
    @Args('text') text: string,
    @CurrentUser() user: User,
  ) {
    const message = await this.chatService.create(user.id,
streamId, text);
    this.pubSub.publish('CHAT_MESSAGE_ADDED', {
chatMessageAdded: message });
    return message;
  }
@Subscription(() => ChatMessageModel, {
    filter: (payload, variables) =>
      payload.chatMessageAdded.streamId === variables.streamId,
  })
  public chatMessageAdded(@Args('streamId') streamId: string) {
    return this.pubSub.asyncIterator('CHAT_MESSAGE_ADDED');
  }
}
```

Лістинг 4.5 – Резолвер чату з підпискою в реальному часі

Принцип роботи такий: коли користувач надсилає повідомлення, мутація sendMessage зберігає його в базі даних та публікує подію CHAT_MESSAGE_ADDED у механізм публікації-підписки. Усі клієнти, які оформили підписку chatMessageAdded на відповідну трансляцію, миттєво отримують нове повідомлення через своє WebSocket-з'єднання. Функція-фільтр гарантує, що кожен глядач отримує лише повідомлення тієї трансляції, яку він переглядає. Завдяки використанню Redis як шини подій ця схема коректно працює навіть за наявності кількох екземплярів сервера, що забезпечує масштабованість чату. Аналогічно за допомогою підписок реалізовано доставлення сповіщень користувачам у реальному часі.

4.4 Тестування програмного забезпечення

Для забезпечення якості та надійності розробленого програмного забезпечення проведено його тестування. Застосовано кілька видів тестування: модульне (unit), наскрізне (end-to-end) та ручне.

Модульне тестування спрямоване на перевірку коректності роботи окремих сервісів ізольовано від решти системи. Для його проведення використано фреймворк Jest. Завдяки механізму впровадження залежностей реальні залежності сервісів (зокрема доступ до бази даних) під час тестування замінюються імітаційними (mock) об'єктами, що дозволяє перевіряти бізнес-логіку незалежно від зовнішніх систем (лістинг 4.6).

```
typescript
describe('AccountService', () => {
let service: AccountService;
let prisma: { user: { findUnique: jest.Mock; create: jest.Mock }
};
beforeEach(() => {
prisma = { user: { findUnique: jest.fn(), create: jest.fn() } };
service = new AccountService(prisma as any, {
sendVerificationToken: jest.fn() } as any);
});
it('повертає помилку, якщо пошта вже існує', async () => {
prisma.user.findUnique
```

```
.mockResolvedValueOnce(null)            // username вільний
.mockResolvedValueOnce({ id: '1' });    // email зайнятий
await expect(
service.create({ email: 'a@a.com', password: 'pass', username:
'user' }),
).rejects.toThrow(ConflictException);
});
});
```

Лістинг 4.6 – Приклад модульного тесту

Наскрізне тестування перевіряє роботу системи в цілому – від надходження GraphQL-запиту до отримання відповіді – та виконується на тестовому екземплярі застосунку з тестовою базою даних. Ручне тестування проводилося за допомогою інтерактивного середовища Apollo Sandbox (GraphQL Playground), яке дозволяє формувати запити, мутації та підписки до сервера, переглядати схему API та перевіряти коректність відповідей. У ході тестування перевірено всі основні сценарії роботи системи: реєстрацію та підтвердження пошти, вхід із двофакторною автентифікацією, створення та запуск трансляції, обмін повідомленнями в чаті в реальному часі, оформлення підписок та надсилання сповіщень. Виявлені під час тестування недоліки було усунено.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи спроєктовано та програмно реалізовано серверну частину веб-платформи для відеотрансляцій із підтримкою інтерактивного чату в реальному часі. Поставлену мету досягнуто, а всі визначені на початку роботи завдання – виконано.

У першому розділі досліджено предметну область відеотрансляцій та інтерактивного чату, з'ясовано особливості розробки відповідного програмного забезпечення та проаналізовано платформи-аналоги. На основі аналізу сформульовано функціональні та нефункціональні вимоги до системи. Здійснено огляд та обґрунтований вибір засобів розробки: середовища Visual Studio Code та мови програмування TypeScript. Виконано постановку завдання та визначено перелік підсистем, які необхідно реалізувати.

У другому розділі детально розглянуто методи та технології, застосовані під час розроблення серверної частини. Проаналізовано модульну архітектуру та життєвий цикл застосунку на фреймворку NestJS, асинхронну неблокувальну модель виконання платформи Node.js, технологію управління моделлю даних на основі ORM Prisma та СУБД PostgreSQL, механізм реального часу на основі сховища Redis та підписок GraphQL, переваги інтерфейсу GraphQL і сервера Apollo, а також технологію потокового відео LiveKit та механізми безпеки. Це сформувало технологічну основу для проєктування системи.

У третьому розділі здійснено проєктування системи із застосуванням мови UML. Обґрунтовано вибір архітектури модульного моноліту та побудовано компонентну діаграму й діаграму розгортання. Функціональні вимоги формалізовано за допомогою діаграми прецедентів. Динаміку взаємодії компонентів описано чотирма діаграмами послідовності (реєстрація з підтвердженням пошти, вхід із двофакторною автентифікацією, обмін повідомленнями в чаті та запуск трансляції) і діаграмою діяльності. Спроєктовано модель даних у вигляді діаграми класів та ER-діаграми бази

даних із дванадцяти таблиць.

У четвертому розділі описано програмну реалізацію серверної частини. Наведено структуру проєкту та реалізацію основних підсистем: автентифікації з хешуванням паролів за алгоритмом Argon2 і двофакторною автентифікацією, роботи з даними та трансляціями з інтеграцією LiveKit, а також інтерактивного чату реального часу на основі підписок GraphQL і механізму публікації-підписки Redis. Описано процес тестування програмного забезпечення (модульне, наскрізне та ручне), що підтвердив працездатність системи.

Практична цінність роботи полягає в тому, що розроблена серверна частина забезпечує повний цикл функціонування стрімінгового сервісу та може бути використана як основа для створення повноцінної платформи відеотрансляцій або як навчальний приклад побудови серверних застосунків реального часу. Розроблену систему побудовано на сучасному стеку технологій, що забезпечує її масштабованість, безпеку та зручність подальшого розвитку.

Напрямами подальшого розвитку системи можуть бути: реалізація системи модерації чату та ролей користувачів; додавання запису трансляцій та їх перегляду у відкладеному режимі; впровадження рекомендаційної системи для каталогу трансляцій; розширення можливостей монетизації; а також підготовка системи до горизонтального масштабування шляхом виділення найбільш навантажених модулів в окремі сервіси.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цеслів О. В. Основи програмування та веб-дизайн : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2020. 150 с.
2. Гусєв В. В., Забарний О. В. Веб-програмування: HTML, CSS, JavaScript : навч. посіб. Харків : ХНУРЕ, 2022. 210 с.
3. Robbins J. N. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics. 5th ed. Sebastopol : O’Reilly Media, 2018. 808 p.
4. NestJS – A progressive Node.js framework : офіційна документація. URL: https://docs.nestjs.com (дата звернення: 20.05.2026).
5. GraphQL : a query language for your API : офіційний сайт. URL: https://graphql.org (дата звернення: 20.05.2026).
6. Apollo Server : документація. Apollo GraphQL. URL: https://www.apollographql.com/docs/apollo-server (дата звернення: 20.05.2026).
7. Prisma ORM : Next-generation Node.js and TypeScript ORM : доку-ментація. URL: https://www.prisma.io/docs (дата звернення: 20.05.2026).
8. PostgreSQL 15 Documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/15 (дата звернення: 18.05.2026).
9. Redis : The Real-time Data Platform : документація. URL: https://redis.io/docs (дата звернення: 18.05.2026).
10. LiveKit Documentation : open source WebRTC infrastructure. URL: https://docs.livekit.io (дата звернення: 18.05.2026).
11. WebRTC : Real-Time Communication for the Web : офіційний сайт. URL: https://webrtc.org (дата звернення: 18.05.2026).
12. Node.js Documentation. OpenJS Foundation. URL: https://nodejs.org/docs (дата звернення: 15.05.2026).
13. TypeScript Documentation. Microsoft. URL: https://www.typescriptlang.org/docs (дата звернення: 15.05.2026).

14. Stripe API Reference : документація. Stripe Inc. URL: https://stripe.com/docs/api (дата звернення: 14.05.2026).
15. Telegraf.js : Modern Telegram Bot Framework : документація. URL: https://telegraf.js.org (дата звернення: 14.05.2026).
16. Fielding R., Reschke J. Hypertext Transfer Protocol (HTTP/1.1) : RFC 7230–7235. Internet Engineering Task Force, 2014. URL: https://www.rfc-editor.org/rfc/rfc7230 (дата звернення: 10.05.2026).
17. Docker Documentation : офіційна документація. Docker Inc. URL: https://docs.docker.com (дата звернення: 14.05.2026).