Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система відео-стримінгу

(тема)

Виконав:
здобувач 4 року навчання, групи ПЗПІ-21-6

Вовк Д.А.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Ворочек О.Г.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Вовк Дмитро Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система відео-стримінгу
Затверджена наказом по університету від 19.05.2025 р № 397 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 10.06.2025
3. Вихідні дані до роботи Розробити веб сервіс для трансляції відео з прямим ефіром, що включає мікро сервісну архітектуру, клієнтську частину на React, серверну логіку на NodeJS та окремий відео сервер для обробки потоків. Забезпечити підтримку авторизації, реєстрації, 2FA, чату з модерацією, пошуку, вибору тегів трансляцій та перегляд у якості від 144p до максимально можливої.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	22.05.2025	*виконано*
3	Проектування ПЗ	24.05.2025	*виконано*
4	Розробка ПЗ	28.05.2025	*виконано*
5	Тестування ПЗ	30.05.2025	*виконано*
6	Оформлення пояснювальної записки	05.06.2025	*виконано*
7	Підготовка презентації та доповіді	06.06.2025	*виконано*
8	Попередній захист	09.06.2025	*виконано*
9	Нормоконтроль, рецензування	06.06.2025	*виконано*
10	Здача роботи у електронний архів	09.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	10.06.2025	*виконано*

Дата видачі завдання 20 лютого 2025р.

Здобувач (ка)_____ Дмитро ВОВК
(підпис)

Керівник роботи _____ доц. кафедри ПІ Ольга ВОРОЧЕК
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 59 стор., 6 рис., 17 джерел.

АВТОРИЗАЦІЯ, ТОКЕН, КОНТЕЙНЕР, СЕРВІС, СТРІМ, ПРЯМА ТРАНСЛЯЦІЯ, ЧАТ, NodeJS, TS, JWT

Об’єктом розробки є програмна платформа для прямих відео трансляцій.

Метою роботи є створення системи, яка надає користувачам можливість проводити та переглядати прямі трансляції з налаштуванням якості та інтерактивними функціями.

Метод розв’язання – мікро сервісна архітектура з використанням NodeJS для обробки відео та веб інтерфейсом на основі React.

У результаті розробки створено функціональну платформу прямих трансляцій з підтримкою перегляду відео, чату з мацерацією, реєстрації та авторизації користувачів (включно з 2FA), пошуку та вибору міток трансляції.

ABSTRACT

Explanatory note to the bachelor's work, 59 pages, 6 figures, 17 sources.

AUTHENTICATION, TOKEN, CONTAINER, SERVICE, STREAM, LIVE TRANSMISSION, CHAT, NodeJS, TS, JWT

The object of development is a software platform for live video broadcasts.

The purpose of the work is to create a system that provides users with the ability to conduct and view live broadcasts with quality settings and interactive features.

The solution method is a microservice architecture using NodeJS for video processing and a React-based web interface.

As a result of the development, a functional live streaming platform has been created that supports video viewing, chat with maceration, user registration and authorization (including 2FA), search and selection of broadcast labels.

ЗМІСТ

ВСТУП

Сфера онлайн-відео трансляцій стрімко розвивається та посідає важливе місце в сучасній цифровій інфраструктурі. Зростаюча популярність стримінгових платформ зумовлена потребою в реальному часі передавати контент великій кількості глядачів із можливістю інтерактивної взаємодії. Такий формат активно використовується в розважальній, освітній, ігровій та інформаційній сферах.

У світовій практиці спостерігається тенденція переходу до мікро сервісної архітектури при створенні високонавантажених систем, що забезпечує масштабованість, гнучкість і зручність в обслуговуванні. Також важливими аспектами залишаються якість відео, можливість спілкування глядачів під час трансляції, захист даних користувачів та персоналізація контенту.

Актуальність даної роботи полягає в необхідності створення сучасної платформи для прямих трансляцій, яка б задовольняла вимоги як з технічної, так і з функціональної точок зору. Особливої уваги потребують реалізація підтримки різних рівнів якості відео, інтеграція чату з модерацією, багаторівнева авторизація користувачів та зручні засоби навігації й пошуку трансляцій.

Мета роботи — розробка платформи для прямих відео трансляцій із сучасною архітектурою, що включає можливість запуску та перегляду трансляцій, спілкування в чаті, захищену реєстрацію та авторизацію користувачів, а також функції пошуку та тегування контенту.

Результати розробки можуть бути використані для створення незалежних стримінгових сервісів, навчальних чи дослідницьких проєктів, а також як частина більших інформаційно-розважальних платформ.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сфера онлайн-відеотрансляцій стала однією з ключових у сучасному цифровому середовищі. Завдяки розвитку високошвидкісного інтернету та популяризації інтерактивного контенту, попит на платформи, що забезпечують можливість стримінгу в реальному часі, значно зріс. Такі сервіси, як YouTube Live, Twitch, Facebook Live та інші, щодня обслуговують мільйони користувачів, надаючи інструменти для трансляції, спілкування з аудиторією та монетизації контенту.

Основними очікуваннями користувачів до таких платформ є стабільна якість відео, доступність трансляцій на різних пристроях, швидке масштабування під навантаження, а також функціональні можливості взаємодії – чат, реакції, модерація тощо. У контексті технічної реалізації, усе більше уваги приділяється використанню мікросервісної архітектури, що дозволяє незалежно масштабувати окремі компоненти системи, швидко впроваджувати нові функції та підтримувати високу доступність.

Типовий стримінговий сервіс складається з таких ключових елементів:

а) серверна частина, що забезпечує авторизацію, маршрутизацію запитів, обробку повідомлень у чаті, пошук та управління контентом;

б) сервер відео, який приймає потоки від стрімерів, трансформує їх у кілька варіантів якості (транскодування) та передає глядачам з урахуванням доступного трафіку й роздільної здатності пристрою;

в) вебінтерфейс, через який користувачі взаємодіють із платформою – переглядають трансляції, ведуть чат, шукають контент, реєструються тощо.

Особливе значення має підтримка декількох рівнів якості відео (від 144p до Full HD або джерельної), що забезпечується за допомогою адаптивного

потокового мовлення (наприклад, HLS або DASH)[1]. Це дозволяє забезпечити доступність контенту для користувачів із різною швидкістю з’єднання.

Ще одним важливим аспектом є система чату. Вона повинна бути інтерактивною, з мінімальними затримками, підтримувати модерацію, а також забезпечувати зручність комунікації. У багатьох сервісах модератори призначаються стрімером.

Функції авторизації й реєстрації мають включати двофакторну автентифікацію (2FA[10]), що підвищує безпеку облікових записів. Також важливим є можливість прив’язки тематики трансляції до тегів, що покращує навігацію та персоналізацію контенту для користувачів.

У межах сучасного програмного забезпечення особливу увагу приділяють масштабованості, стійкості до навантажень та безпеці особистих даних. Тому платформи розробляються із використанням технологій, здатних обробляти тисячі одночасних підключень, захищати інформацію, а також легко інтегрувати нові модулі без повного перезапуску системи.

У межах даної кваліфікаційної роботи реалізується платформа, що об'єднує всі ключові елементи сучасного стримінгового сервісу: прямі трансляції з вибором якості, чат з модерацією, реєстрацію й авторизацію з 2FA[10], систему тегів і пошуку. Це дозволяє продемонструвати практичну реалізацію повноцінного функціонального середовища для відеотрансляцій.

1.2 Виявлення та вирішення проблем

У процесі аналізу предметної області виявлено низку проблем, які стоять перед розробниками сучасних платформ для прямих відеотрансляцій.

Однією з ключових є складність забезпечення стабільного стримінгу при великій кількості глядачів та різноманітності їхніх пристроїв і швидкостей інтернет-з'єднання. Необхідно не лише транслювати відео в режимі реального

часу, а й забезпечити можливість перемикання якості без зупинки відтворення. Це потребує впровадження адаптивного потокового мовлення, що вимагає оптимізації транскодування та зберігання декількох варіантів потоку.

Іншою поширеною проблемою є організація ефективного чату під час трансляцій. Чат повинен:

а) мати низьку затримку;

б) підтримувати велику кількість повідомлень за короткий проміжок часу;

в) забезпечувати можливості модерації привілейованими користувачами.

Значну увагу необхідно приділяти системі автентифікації. Звичайна авторизація за логіном і паролем вже не забезпечує належного рівня безпеки. Тому сучасна платформа повинна:

а) підтримувати двофакторну автентифікацію (2FA[10]);

б) захищати персональні дані;

в) забезпечувати надійність входу до системи.

Проблемою також є персоналізація та доступність контенту. Щоб користувач швидко знаходив релевантні трансляції, необхідно реалізувати:

а) систему пошуку за ключовими словами;

б) підтримку тегів для категоризації трансляцій;

в) фільтрацію результатів за тематиками або іншими критеріями.

Ще одним викликом є масштабованість платформи. У зв’язку з цим потрібно:

а) розділити систему на незалежні компоненти;

б) впровадити мікросервісну архітектуру;

в) забезпечити можливість незалежного масштабування окремих частин.

У межах даного проєкту реалізовано:

а) адаптивну трансляцію відео різної якості (від 144p до максимальної);

б) інтерактивний чат із можливістю модерації;

в) реєстрацію, авторизацію та 2FA[10];

г) систему тегів і пошук;

д) масштабовану мікросервісну архітектуру.

1.3 Постановка задачі

Метою даної кваліфікаційної роботи є створення програмного забезпечення для платформи прямих відео трансляцій, що забезпечує якісний стримінг, взаємодію користувачів у реальному часі та масштабовану серверну інфраструктуру.

Для досягнення цієї мети необхідно розв'язати такі задачі:

а) розробити архітектуру системи на основі мікросервісного підходу, що забезпечить масштабованість, надійність та гнучкість у розгортанні окремих компонентів;

б) реалізувати модуль прямого мовлення, що дозволить:

1) транслювати відео у реальному часі;
2) переглядати трансляції з можливістю вибору якості (від 144p до максимальної доступної);
3) обробляти відео окремим сервером для зменшення навантаження на основну логіку системи;

в) забезпечити чат під час трансляцій, який підтримує:

1) обмін повідомленнями з мінімальною затримкою;
2) модерацію повідомлень привілейованими користувачами;

г) реалізувати систему користувачів, що включає:

1) реєстрацію та авторизацію;
2) двофакторну автентифікацію (2FA[10]);

д) створити функціонал для пошуку трансляцій, що дозволяє:

1) здійснювати пошук за ключовими словами;
2) фільтрувати трансляції за тегами (мітками);

е) розробити клієнтську частину (веб-інтерфейс) з використанням сучасного фреймворку (React), що забезпечує зручність, швидкодію.

Таким чином, реалізація вищевказаних задач дозволить створити функціональну, безпечну та масштабовану платформу для відеострімінгу з інтерактивною взаємодією користувачів.

1.4 Цільова аудиторія

Цільова аудиторія платформи для прямих відео трансляцій охоплює широке коло користувачів, зацікавлених у створенні та перегляді контенту в режимі реального часу. Її можна умовно поділити на кілька основних категорій:

а) стрімери (контент-мейкерки) — користувачі, які ведуть трансляції, створюють інформаційний, розважальний, навчальний або ігровий контент. Для них важливі:
   1) стабільність трансляції;
   2) якість відео;
   3) взаємодія з аудиторією через чат;
   4) можливість модерації;

б) глядачі — особи, що переглядають трансляції та взаємодіють з автором або іншими глядачами в чаті. Для них критичними є:
   1) швидкий доступ до трансляцій;
   2) вибір якості відео залежно від швидкості інтернету;
   3) зручний інтерфейс пошуку за інтересами;

в) модератори — користувачі, які мають привілеї щодо контролю чату. Їм необхідні:
   1) інструменти для фільтрації повідомлень;
   2) можливість блокування порушників;
   3) швидке реагування на спам та некоректну поведінку;

Таким чином, розробка системи орієнтована на задоволення потреб як контент-мейкерків, так і глядачів, водночас забезпечуючи зручні інструменти керування для модераторів.

1.5 Виявлення проблем

Після аналізу основних конкурентів у сфері стримінгових платформ (Twitch, YouTube Live, Facebook Live) можна визначити їхні сильні та слабкі сторони:

а) переваги платформи Twitch:

1) висока стабільність трансляцій;
2) широка спільнота глядачів і стрімерів;
3) зручні інструменти монетизації (пожертви, підписки, реклама);
4) потужна система рекомендацій і категоризації контенту;

б) недоліки Twitch:

1) висока конкуренція серед нових авторів;
2) слабкі вбудовані інструменти модерації без сторонніх сервісів;
3) вимоги до високошвидкісного інтернету як для стрімерів, так і для глядачів;
4) обмежена персоналізація інтерфейсу;

в) переваги YouTube Live:

1) інтеграція з YouTube-екосистемою;
2) можливість зберігати трансляції як відео;
3) підтримка високої якості відео;
4) висока доступність для широкої аудиторії;

г) недоліки YouTube Live:

1) обмеженість функцій взаємодії у чаті;
2) повільна модерація коментарів;
3) відсутність чіткої категоризації трансляцій за інтересами;

4) складний інтерфейс для початківців.

Усі ці недоліки вказують на потребу створення стримінгової платформи, орієнтованої на простоту використання, ефективну модерацію, адаптивну якість трансляцій та рівні можливості для нових авторів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Підсистема стримінгової платформи відповідає за організацію трансляцій, перегляд відео, взаємодію користувачів у чаті, а також керування правами доступу та авторизацією. Система має бути масштабованою, безпечною та зручною для користувача.

Функціональні вимоги:

а) платформа повинна підтримувати мікросервісну архітектуру;

б) можливість запуску прямих трансляцій з боку авторизованих користувачів;

в) перегляд трансляцій у якості від 144p до максимально доступної (залежно від каналу та пристрою);

г) реалізація чату до трансляцій з підтримкою:

1) текстових повідомлень;

2) модерації з боку привілейованих користувачів;

д) пошук трансляцій за тегами;

е) авторизація, реєстрація, двофакторна автентифікація (2FA[10]);

ж) можливість вибору міток (тегів) під час створення трансляції.

Нефункціональні вимоги:

а) висока доступність (підтримка великої кількості одночасних з'єднань);

б) мінімальна затримка відео потоку (stream latency);

в) підтримка масштабованості (горизонтальне розгортання мікро сервісів);

г) сумісність з браузерами Chrome, Firefox, Edge (останні 3 версії);

д) безпека — всі з’єднання мають бути захищені TLS;

е) відео потік має бути зашифрований, а токени доступу — з обмеженим терміном дії.

Технічні вимоги:

Мінімальні для клієнта (глядача):

Процесор: Intel Core i3;

ОЗП: 4 ГБ;

Відеокарта з підтримкою відео апаратного декодування H.264;

Браузер із підтримкою HTML5;

Інтернет-з’єднання: 5 Мбіт/с.

Рекомендовані:

Процесор: Intel Core i5 або AMD Ryzen 5;

ОЗП: 8 ГБ і більше;

GPU з підтримкою декодування H.265;

Широкосмуговий інтернет 25 Мбіт/с і більше.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування функціонального інтерфейсу користувача

Під час розробки інтерфейсу користувача було створено UML-діаграми варіантів використання для основних компонентів системи: головної сторінки та сторінки прямої трансляції. Це дозволяє систематизувати основні сценарії взаємодії користувача з програмним продуктом.

На головній сторінці користувач має змогу:

а) здійснити авторизацію або реєстрацію (із підтримкою 2FA[10]);

б) шукати контент за ключовими словами;

в) переглядати перелік підписок;

г) налаштовувати профіль, включаючи:

1) зміну пароля;
2) налаштування 2FA[10];
3) налаштування електронної пошти;
4) генерацію токена трансляції;
5) налаштування сесій;

д) налаштування трансляції:

1) Отримати/змінити токен трансляції;
2) Зміна тегів трансляції;
3) Зміна опису трансляції;
4) 4Зміна заголовка трансляції.

Відповідна UML-діаграма наведена нижче (рис. 3.1).

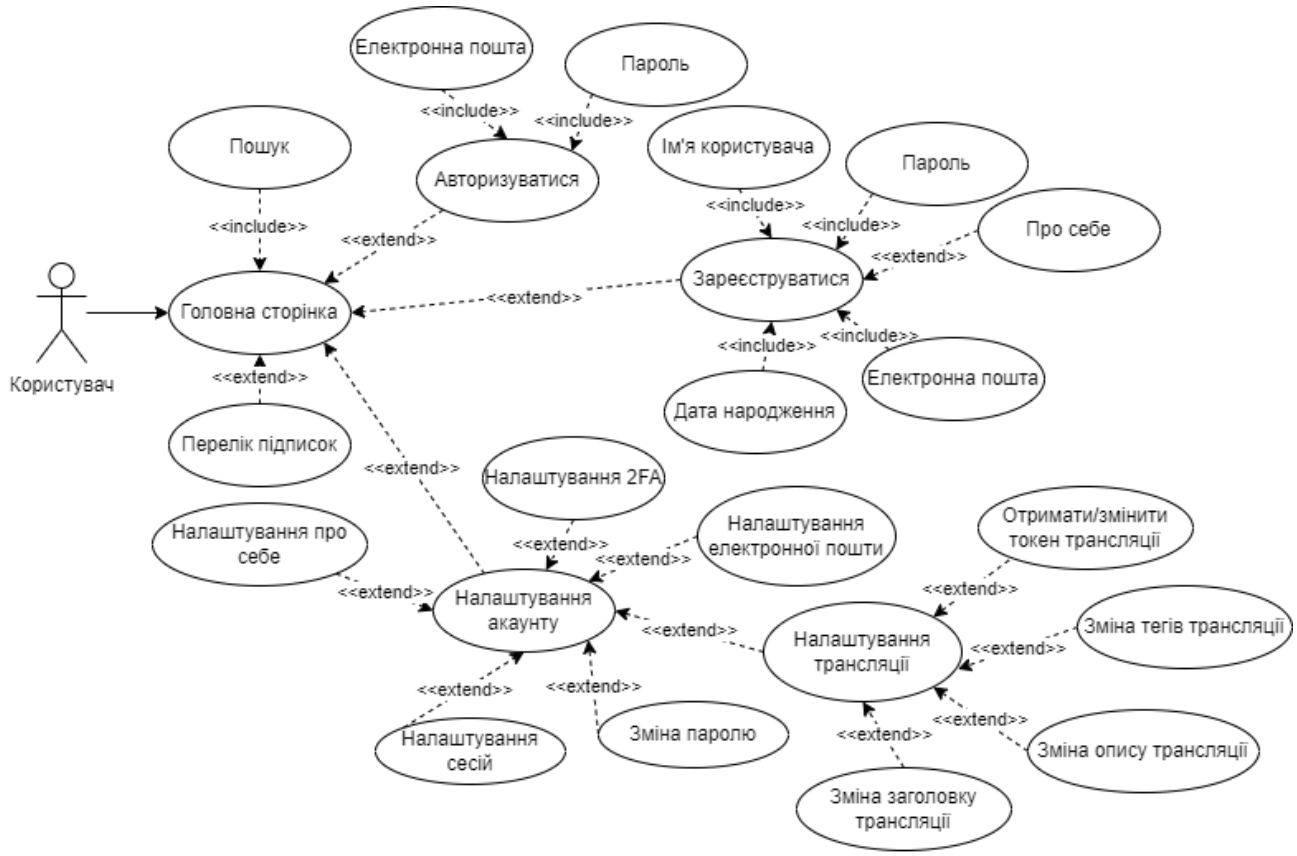


Рисунок 3.1 – UML-діаграма варіантів використання головної сторінки (рисунок виконаний самостійно)

Сторінка трансляції є центральним елементом взаємодії користувача із платформою. Вона надає всі необхідні інструменти для перегляду й участі у прямих ефірах. На сторінці трансляції користувач отримує доступ до:

а) плеєра трансляції з можливістю вибору якості;

б) чату трансляції;

в) перегляду назви, опису та тегів трансляції.

Це дозволяє забезпечити зручність перегляду та взаємодії з контентом у реальному часі. Відповідна діаграма подана нижче (рис. 3.2).

Рисунок 3.2 – UML-діаграма варіантів використання сторінки трансляції (рисунок виконаний самостійно)

Для зручної побудови запиту на сервер, де потрібна авторизація, було розроблено функцію fetchWithAuth, яка автоматично додає заголовок з JWT[3] токеном у запит.

```
const fetchWithAuth = useCallback(
  async (input: RequestInfo, init?: RequestInit): Promise<Response>
=> {
    const accessToken = localStorage.getItem('accessToken');
    if (!accessToken) {
      logout();
      throw new Error('User is not authenticated');
    }

    const headers = {
      ...init?.headers,
      Authorization: `Bearer ${accessToken}`,
    };

    let response = await fetch(input, { ...init, headers });

    if (response.status === 401) {
      const refreshed = await refreshToken();
      if (refreshed) {
        const newAccessToken = localStorage.getItem('accessToken');
        const newHeaders = {
          ...init?.headers,
          Authorization: `Bearer ${newAccessToken}`,
        };
```

```
            response = await fetch(input, { ...init, headers: newHeaders
});
          } else {
            throw new Error('User is not authenticated');
          }
        }

        return response;
      },
      [logout, refreshToken]
    );
```

3.2 Безпека

Система реалізує багаторівневу модель безпеки для захисту даних користувачів та забезпечення безпечної взаємодії між мікро сервісами. Комунікація як між сервісами, так і з клієнтами, захищена через використання JWT-токенів[3], мережевої ізоляції та централізованого керування доступом. Перелік сервісів продемонстровано на рисунку 3.3.

Безпека користувачів:

а) аутентифікація здійснюється через auth-service, який зберігає хеш пароля та email кожного користувача;

б) після входу, auth-service видає JWT-токен[3], підписаний приватним ключем. Токен має обмежений термін дії;

в) перевірка токенів здійснюється в main-api, де також реалізовано логіку: 2FA аутентифікації (через код)[10] та підтвердження входу через email;

г) для забезпечення цілісності ідентифікації, ID користувача є однаковим у всіх сервісах та формується/використовується відповідно до логіки main-api;

д) публічний ключ доступний для перевірки токенів будь-яким компонентом системи або клієнтом.

Безпека міжсервісної взаємодії:

а) сервіси отримують внутрішні JWT-токени[3] з service-manager-backend. Термін їх дії — близько 5 хвилин;

б) при первинному підключенні сервіс надсилає своє ім’я та секрет, після чого отримує токен;

в) автоматичне оновлення здійснюється через токен поновлення;

г) усі між сервісні запити супроводжуються перевіркою токена на боці отримувача.

Доступ і ізоляція

а) усі сервіси ізольовані всередині Docker-мережі[7] your-stream-net. Зовнішній доступ дозволено лише до Nginx, HLS і RTMP[14];

б) додавання нових сервісів можливе лише через service-manager-frontend, який доступний на окремому порту.

3.3 Проєктування Core бібліотеки

YourStream/Core — це централізована спільна бібліотека для всієї мікро сервісної екосистеми YourStream. Її мета — уникнення дублювання коду, забезпечення стандартизації спільних процесів (авторизація, комунікація, події, типи даних) та спрощення розробки нових сервісів через повторне використання перевірених компонентів.

Мікро сервісна архітектура передбачає наявність великої кількості ізольованих сервісів, кожен з яких може мати власну логіку, але часто виконує спільні завдання: автентифікація, перевірка токенів, взаємодія з AuthService, підписка на події, публікація в Redis.

Відсутність єдиної базової бібліотеки призвела б до:

а) дублювання однакового коду авторизації та комунікації в кожному сервісі;

б) потенційних розбіжностей у перевірці токенів чи структурі відповідей;

в) труднощів з оновленням логіки на всіх сервісах одночасно;

г) зниження підтримуваної та розширюваності екосистеми.

Core-бібліотека розв'язує ці проблеми, інкапсулюючи загальні механізми, які потребують лише одного джерела істини, спрощуючи масштабування та розвиток системи.

Одними з ключових компонентів бібліотеки є:

а) ServiceAuthVerifier — призначений для автентифікації сервісів між собою. Реалізує:
   1) підключення до AuthService за допомогою name та secret;
   2) отримання та оновлення JWT[3] токена;
   3) завантаження публічного ключа;
   4) middleware serviceAuthGuard для захисту внутрішніх ендпоінтів;
   5) утиліту buildServiceRequest для авторизованої між сервісної комунікації.

б) UserAuthVerifier — відповідає за перевірку користувацьких токенів:
   1) завантажує публічний ключ з AuthService;
   2) реалізує singleton-модель;
   3) автоматично оновлює ключ з певним інтервалом;
   4) експортує middleware guard для авторизації користувачів у будь-якому мікро сервісі.

Ці модулі дозволяють сервісам YourStream безпечно й стандартизовано взаємодіяти одне з одним, а також з користувачами, без потреби дублювати логіку в кожному з них.

3.4 Проєктування сервісів

Архітектура системи реалізована на мікро сервісному підході з використанням технологій контейнеризації (Docker[7]). Основні компоненти

взаємодіють через REST API, WebSocket[5] API та RTMP-протокол. Внутрішня взаємодія між сервісами відбувається за допомогою HTTP-запитів[2] у межах спільної Docker-мережі[7] your-stream-net. На діаграмі нижче (рис. 3.3), представленні всі сервіси та їх взаємодія.

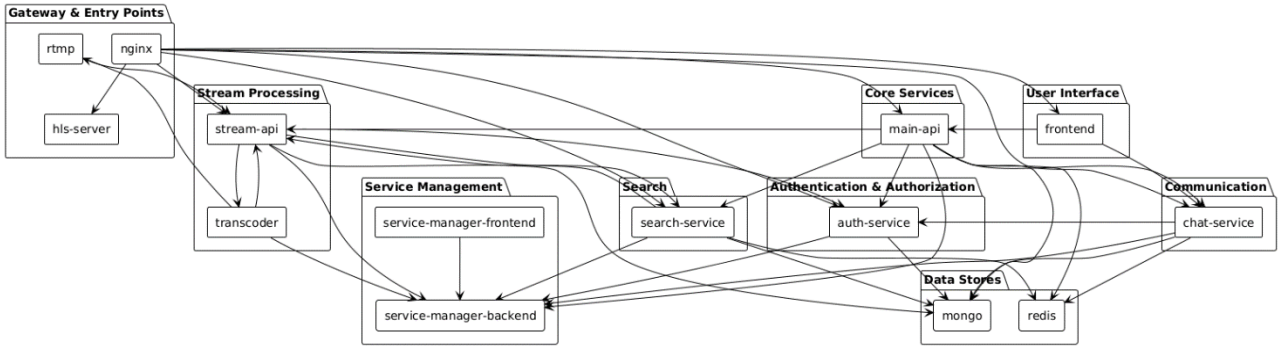


Рисунок 3.3 – Діаграма взаємодії між сервісами (рисунок виконаний самостійно)

Пакети системи:

а) Gateway & Entry Points – вхідні точки, реалізована через Nginx[15]. Він охоплює сервіси:
   1) rtmp - Сервіс, що реалізує прийняття прямих трансляції за протоколом RTMP. Реалізований з використанням Nginx[14][15];
   2) nginx - Східна точна для HTTP[2] комунікації. Proxy сервер, реалізований на базі Nginx[15];
   3) hls - Сервіс що надає доступ до HLS[1] фрагментів;

б) Stream Processing - обробка прямих трансляції. Містить:
   1) stream-api - Сервіс що обробляє запити початок та завершення від rtmp. Перевіряє доступ до rtmp, взаємодії з transcoder для керування транскодуванням;
   2) transcoder - Сервіс транскодування трансляції з rtmp у перелік потоків різної якості, від 144p до якості джерела, або максимально доступної, що задана у конфігураціях сервісу. Перелік якостей, у яких ведеться транскодування, відправляється stream-api;

в) Service Management - сервіс, що реалізує безпечну комунікацію між сервісами системи та сервіс з веб інтерфейсом, для адміністрування переліку сервісів;

г) Search - сервіс, що реалізує пошук трансляцій за тегами;

д) Authentication & Authorization - Сервіс, що реалізує авторизацію та аутентифікацію користувачів. Використовується різними сервісами для керування доступом;

е) Core Services - Містить дані користувачів, опрацювання 2FA[10], систему сесій, систему підтверджень;

ж) User Interface - React додаток, що реалізує користувацький інтерфейс;

з) Data Stores - Для зберігання даних було обрано MongoDB[12] та Redis для кешування.

Таким чином, взаємодія між сервісами відбувається логічно ізольовано, з чітко визначеними зонами відповідальності. Це забезпечує масштабованість, надійність та спрощує тестування, оновлення та розгортання окремих компонентів системи.

Детальне проєктування ключових сервісів подається у підпунктах нижче.

3.3.1 Проєктування реєстрації та авторизації користувачів

У рамках розробки мікросервісної архітектури було реалізовано окремий сервіс, відповідальний за обробку реєстрації та авторизації користувачів. Сервіс побудовано з використанням платформи Node.js[6] та фреймворку Express[8], із застосуванням бібліотеки Mongoose[13] для взаємодії з базою даних MongoDB[12] та JWT[3] для аутентифікації.

Сервіс реалізовано за модульним підходом, що передбачає розділення на такі основні компоненти (рис. 3.4):

а) маршрутизатори: auth, user, key — обробляють запити клієнтів;

б) сервіси: db.ts, jwt.ts — забезпечують з'єднання з базою даних і генерацію/перевірку токенів;

в) модель користувача: AuthUser — визначає структуру об'єкта користувача та методи роботи з ним.

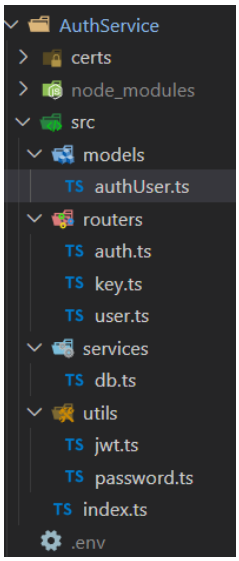


Рисунок 3.4 – Структура макросервісу авторизації та аутентифікації (рисунок виконаний самостійно)

Всі запити, окрім запиту до key можуть робити тільки інші сервіси. Розглянемо реалізовані маршрути та їх функціонал.

Маршрут POST /api/user реалізує процес створення нового користувача. Перед збереженням у базу перевіряється унікальність email та id. Пароль хешується з використанням утиліти passwordUtils.hash()[4], що підвищує безпеку збереження облікових даних. Після успішного створення, користувачу присвоюється дата створення у форматі UTC.

Маршрут POST /api/auth/signin відповідає за автентифікацію. З використанням middleware serviceAuthGuard запит перевіряється на валідність, після чого виконується пошук користувача за email. У разі успішної перевірки пароля створюється JWT-токен[3] з допомогою методу generateToken(), реалізованого безпосередньо в моделі користувача.

Реалізовано маршрут POST /api/auth/validate-password, який дозволяє перевірити правильність пароля без видачі токена. Це корисно для реалізації функціоналу повторної перевірки облікових даних у чутливих операціях.

Модель AuthUser побудовано за допомогою Mongoose, із полями:

а) _id (ObjectId),

б) email (унікальний, обов’язковий),

в) passwordHash,

г) createdAt, updatedAt.

До моделі додано методи:

а) comparePassword() — перевірка пароля,

б) generateToken() — створення JWT[3],

в) toResponse() — серіалізація без чутливих даних.

Аутентифікація реалізована з використанням відкритого/приватного RSA ключа. Токени підписуються з параметрами, визначеними в конфігурації (JWT_ALGORITHM, JWT_EXPIRATION, jwtid, kid). Публічний ключ доступний за маршрутом GET /api/key для перевірки токенів іншими сервісами.

Перед запуском сервіс перевіряє наявність обов’язкових змінних середовища: SERVICE_NAME, SERVICE_SECRET, AUTH_SERVICE_ADDRESS. За відсутності хоча б однієї — процес завершується аварійно.

Таким чином, реалізований сервіс забезпечує повноцінну реєстрацію та авторизацію користувачів із дотриманням вимог безпеки, масштабованості та інтеграції у мікросервісну архітектуру.

3.3.2 Архітектура для обробки прямих трансляцій

Система обробки прямих трансляцій побудована з використанням модуля RTMP сервера (Nginx)[14], сервісу API для керування потоками (stream-api), транскодера для обробки відео (transcoder) та системи зберігання HLS-потоків[1]

(рис. 3.5). Архітектура забезпечує безпечне публікування стрімів, обробку подій початку та завершення трансляцій, адаптивне перекодування потоку та надання інформації про стріми користувачам.

Використовується Nginx[15] з двома окремими конфігураційними файлами:

а) rtmp.conf — обробка RTMP-з’єднань[14], зокрема прослуховування порту 1935 і обробка подій on_publish та on_publish_done, які викликають відповідні HTTP-запити до[2] API;

б) hls.conf — обробка запитів до HLS-потоків[1], доступ до яких надається через location /hls/. Потоки зберігаються в /tmp/hls/, доступ відкритий через Access-Control-Allow-Origin: *.

Сервіс stream-api реалізований за допомогою Node.js[6] (Express[8]). Основні компоненти:

а) обробка подій RTMP:

1) POST /api/stream/on_publish: перевірка ключа стріму, перевірка користувача, встановлення прапора isLive, виклик транскодера;
2) POST /api/stream/on_publish_done: скидання параметрів стріму після завершення, зупинка транскодера;

б) оновлення інформації про стрім:

1) PUT /api/external: редагування назви, опису, тегів стріму. Теги передаються до search-service;

в) інформаційні запити:

1) Отримання повної або скороченої інформації про стрім (GET /api/external/:userId, :userId/short);

г) керування користувацькими ключами:

1) генерація нового ключа трансляції;
2) створення нового стріму;
3) отримання якості трансляції.

Модель StreamModel зберігає інформацію про стрім:

а) title, description, tags — метаінформація;

б) streamKey, isLive — автентифікація та статус;

в) source — технічні параметри потоку: роздільна здатність, пропорції, список якостей.

Сервіс transcoder відповідає за прийом подій запуску/завершення трансляції (POST /api/transcoder/start, stop) та запуск процесу FFMPEG.

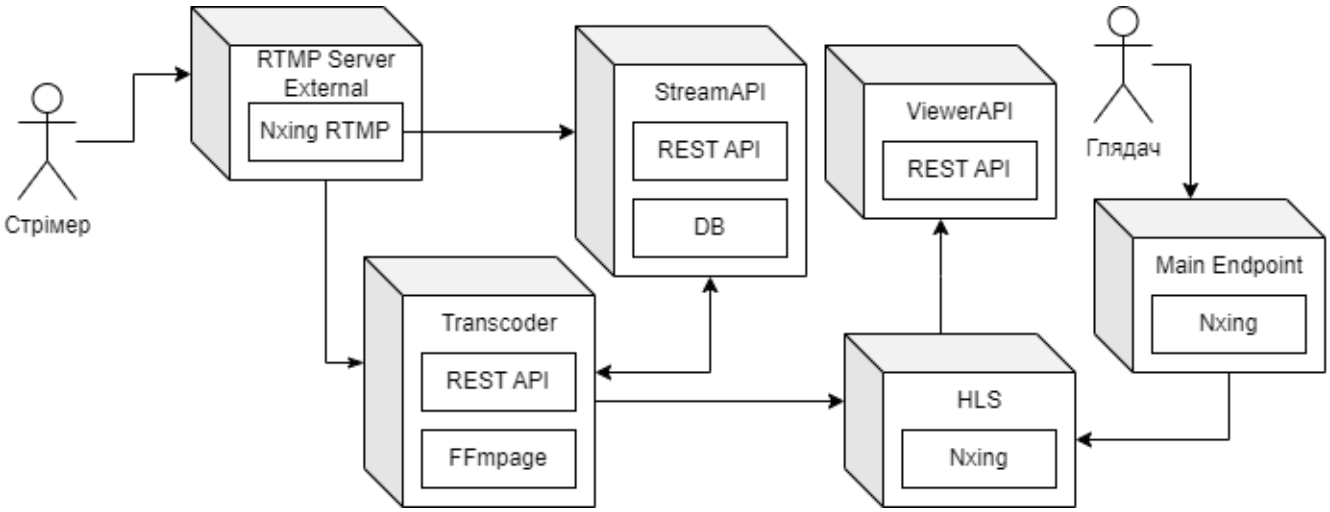


Рисунок 3.5 – UML-Обробки трансляції (рисунок виконаний самостійно)

Сервіс transcoder використовує команду FFmpeg[9] для обробки відеопотоку з RTMP-серверу[14] та створення HLS-потоків[1] різної якості. Команда будується динамічно залежно від роздільної здатності вхідного потоку. Нижче наведено типовий приклад команди:

```
ffmpeg -i rtmp://{ORIGINAL_RTMP_SERVER}/live/{streamKey}\
-c:a aac -ar 44100 -b:a 128k -ac 1 \
-c:v libx264 -preset veryfast -g 50 -sc_threshold 0 \
-map 0:v:0 -map 0:a:0 -s:v:0 1920x1080 -b:v:0 4500k \
-map 0:v:0 -map 0:a:0 -s:v:1 1280x720 -b:v:1 2500k \
-map 0:v:0 -map 0:a:0 -s:v:2 854x480 -b:v:2 1000k \
-f hls -hls_time 4 -hls_playlist_type event \
-hls_segment_filename "/tmp/hls/{userId}/{quality}/%03d.ts" \
"/tmp/hls/{userId}/{quality}/index.m3u8"
```

Пояснення параметрів:

- -i — вхідний RTMP-потік;
- -c:a aac — аудіокодек AAC, рекомендований для HLS[1];

- -ar 44100, -b:a 128k, -ac 1 — параметри аудіо (частота, бітрейт, кількість каналів);
- -c:v libx264 — відеокодек H.264;
- -preset veryfast — швидкий режим кодування;
- -g 50 — розмір групи кадрів (GOP), важливо для HLS[1];
- -sc_threshold 0 — відключення автоматичного розбиття сегментів при зміні сцени;
- -map — вибір відео та аудіо доріжок;
- -s:v — масштабування відео під відповідну якість (1080p, 720p, 480p);
- -b:v — відео бітрейт;
- -f hls — формат вихідного потоку — HLS[1];
- -hls_time 4 — довжина одного сегмента (4 секунди);
- -hls_playlist_type event — тип плейліста, використовується для подій;
- -hls_segment_filename — шлях до сегментів .ts[1];
- останній аргумент — шлях до головного HLS-файлу .m3u8[1].

Перед побудовою команди відбувається аналіз роздільної здатності вхідного потоку. Якщо, наприклад, джерело має 1280x720, то якості 1080p не буде створено. Це запобігає перевищенню вихідної якості при масштабуванні.

3.3.3 Проєктування сервісу чату трансляції

Чат трансляції реалізовано як окремий мікросервіс, що забезпечує збереження історії повідомлень, роботу WebSocket-з’єднання[5], модерацію користувачів та управління правами доступу. Архітектура сервісу побудована на основі REST API та WebSocket[5].

Сервіс запускається у файлі index.ts. У цьому модулі ініціалізується середовище, логування, підключення до інших сервісів (аутентифікація користувачів та сервісів), реєстрація маршрутів HTTP[2] та WebSocket[5].

```
app.use("/api/chat", chat.chatRouter);
app.use("/api/chat/settings", chatSettings);
wsInstance.app.ws("/api/chat/stream", chat.chatStreamRouter);
```

Основні компоненти чату:

- чат повідомлення (ChatMessage) — модель, яка описує структуру повідомлень, що зберігаються в базі даних;
- чат (Chat) — модель, що зберігає інформацію про власника чату, модераторів і заблокованих користувачів;
- сервіс ChatService — singleton, який відповідає за доставку повідомлень, виклики обробників та збереження історії.

Кожне повідомлення відправляється користувачем через HTTP POST-запит[2]:

```
POST /api/chat/message
{
  "chatId": "...",
  "content": "..."
}
```

Після надсилання повідомлення викликається метод sendMessage, який зберігає повідомлення в базу даних та розсилає його усім відкритим WebSocket-з’єднанням[5]:

```
await chatMessage.save();
this.notifyHandlers(message);
```

Для стрімінгу повідомлень реалізовано WebSocket-ендпоінт[5] /api/chat/stream, до якого клієнт підключається з параметром chatId. Після встановлення з’єднання сервер слухає події та передає їх клієнту. Встановлено підтримку ping/pong для підтримки з’єднання.

Також реалізовано ендпоінти для:

- отримання історії чату (/api/chat/history);
- створення чату (/api/chat/create);
- додавання/видалення модераторів (/api/chat/settings/moderator);

- блокування/розблокування користувачів (/api/chat/settings/block).

Доступ до частини ендпоінтів обмежений через middleware guard (для користувачів) та serviceAuthGuard (для сервісів).

Таким чином, чат трансляції реалізовано як ізольований компонент системи з підтримкою стрімінгу, модерації та збереження повідомлень.

3.3.4 Проєктування пошуку

Сервіс пошуку реалізує функціонал пошуку стрімів за тегами, а також оновлення тегів для конкретного стріму. Робота сервісу базується на REST API з доступом до бази даних MongoDB[12].

Сервіс ініціалізується у файлі index.ts, де налаштовуються змінні середовища, логування, підключення до сервісу аутентифікації та маршрутизація:

```
app.use("/api/search", search);
```

Основна логіка пошуку реалізована у маршрутизаторі search.ts. Пошук стрімів виконується через GET-запит:

```
GET /api/search?q=#tag1 #tag2&page=1&limit=10
```

- теги витягуються з параметра q (ті, що починаються з #);
- усі знайдені теги шукаються в базі Tag;
- формується множина унікальних streamId, які відповідають знайденим тегам;
- дані стрімів витягуються з окремого сервісу stream-api через /api/stream/batch.

Пошуковий запит повертає результат у форматі:

```
{
  "total": 3,
  "page": 1,
  "limit": 10,
```

```
  "results": [ /* масив стрімів */ ]
}
```

Для оновлення тегів стріму використовується POST-запит:

```
POST /api/search
{
  "streamId": "stream123",
  "tags": ["tag1", "tag2"]
}
```

Маршрут захищено middleware serviceAuthGuard. У запиті:

- додаються нові теги до стріму (створюються за потреби);
- видаляються старі теги, яких немає у новому списку.

Модель тегу описано у файлі tag.ts. Основні поля:

- name — унікальна назва тегу;
- streamIds — список ID стрімів, що мають цей тег;
- createdAt — дата створення тегу.

Таким чином, сервіс пошуку реалізує базову функціональність фільтрації стрімів за тегами з підтримкою пагінації та оновленням тегів.

3.3.5 Архітектура взаємодії між сервісами

У рамках переходу до мікросервісної архітектури важливо забезпечити захищену, контрольовану та масштабовану взаємодію між окремими сервісами системи. Основним способом комунікації між сервісами обрано HTTP-запити[2], які відбуваються виключно через внутрішню мережу, із чіткою перевіркою автентичності кожного запиту. Наступна функція формує запит до обраного сервісу:

```
async function buildServiceRequest(
serviceHost: string,
path: string,
```

```
    options: {
        method: "GET" | "POST" | "PUT" | "DELETE",
        headers?: Record<string, string>
        body?: object
    } = { method: "GET" },
    ): Promise<globalThis.Response> {
    const url = `${serviceHost}${path}`;
    const headers: Record<string, string> = {
            "Content-Type": "application/json",
         Authorization: `Bearer ${serviceAuthVerifier.token}`,
            ...options.headers,
    };

        const requestOptions: RequestInit = {
            method: options.method,
            headers,
        };

        if (options.body) {
         requestOptions.body = JSON.stringify(options.body);
        }

        try {
         const response = await fetch(url, requestOptions);
            return response;
        } catch (error) {
         logger.error(`[buildServiceRequest] Error while making request
to ${url}:`, error);
         throw new Error("Failed to make service request");
        }
    }
```

Для автентифікації запитів між сервісами реалізовано центр авторизації сервісів — окремий сервіс, відповідальний за видачу JWT токенів[3]. Кожен мікро сервіс має власну пару ключів (приватний та публічний), а також реєстраційні дані: унікальну назву та секретний пароль.

Реєстрація нового сервісу виконується вручну адміністратором через окрему адміністративну панель. Ця панель:

- доступна тільки локально, тобто не доступна ззовні чи через загальний інтерфейс API;
- працює окремо від основного endpoint системи, із власною ізольованою логікою доступу;
- дозволяє додавати нові сервіси, генерувати для них ключі та задавати політики доступу.

Процес авторизації сервісів:

а) сервіс звертається до менеджера сервісів, передаючи свою назву та пароль;

б) у разі успішної перевірки, менеджер видає:

1) JWT токен[3], підписаний приватним ключем менеджера;

2) Refresh токен, термін дії якого на 5 хвилин довший, ніж у JWT[3].

Процес перевірки:

а) сервіс-отримувач отримує токен із заголовку запиту;

б) перевіряє підпис за допомогою публічного ключа відповідного сервісу;

в) якщо токен корекнтий — запит обробляється. У протилежному випадку — відхиляється з відповідним повідомленням про помилку автентифікації.

Особливості:

- кожен сервіс зобов’язаний періодично оновлювати токен;
- публічні ключі для перевірки можуть зберігатися локально або бути доступні через окремий discovery endpoint;
- централізована видача токенів поєднується з децентралізованою перевіркою, що дозволяє масштабувати систему без втрати безпеки.

Цей підхід забезпечує:

- надійний контроль над комунікацією між сервісами;
- захищений обмін даними у внутрішній інфраструктурі;
- гнучке масштабування і просте адміністрування.

3.3.5 Архітектура веб застосунку

Для реалізації веб застосунку було обрано React і його фреймворк Next.js. Проєкт є структурованим та розділеним на компоненти (рис. 3.6).

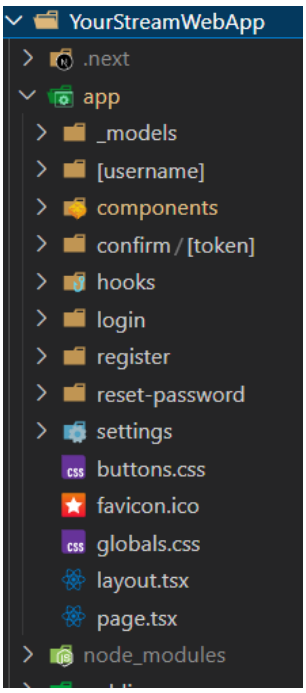


Рисунок 3.6 – Структура веб застосунку (рисунок виконаний самостійно)

У проєкті було реалізовано наступні шляхи:

- /[username] – сторінка трансляції користувача із іменем username;
- /login – сторінка авторизації;
- /register – сторінка реєстрації;
- /reset-password – сторінка скидання пароля;
- /reset-password/[token] – сторінка вказання нового пароля;
- /settings/general – налаштування прямої трансляції;
- /settings/security – налаштування безпеки акаунту;
- /confirm/[token] – сторінка підтвердження дії.

Також було розроблено наступні хуки:

- useAuth – для перевірки авторизації, запиту на оновлення токена, запиту на вихід користувача та функції що створює запит то вхідних точок, що потребують авторизації:
-

```
const fetchWithAuth = useCallback(
        async (input: RequestInfo, init?: RequestInit):
Promise<Response> => {
            const accessToken = localStorage.getItem('accessToken');
            if (!accessToken) {
                logout();
```

```
                throw new Error('User is not authenticated');
            }

            const headers = {
                ...init?.headers,
                Authorization: `Bearer ${accessToken}`,
            };

            let response = await fetch(input, { ...init, headers });

            if (response.status === 401) {
                const refreshed = await refreshToken();
                if (refreshed) {
                    const                newAccessToken                =
localStorage.getItem('accessToken');
                    const newHeaders = {
                        ...init?.headers,
                        Authorization: `Bearer ${newAccessToken}`,
                    };
                    response = await fetch(input, { ...init, headers:
newHeaders });
                } else {
                    throw new Error('User is not authenticated');
                }
            }

            return response;
        },
        [logout, refreshToken]
    );
```

- useChat – робота з чатом: отримання та відправка повідомлень;
- useSubscriptions – створено для роботи із переліком підписок користувача;
- useUser – хук для отримання даних про користувача за ідентифікатором або іменем користувача, також для отримання поточного користувача.

3.4 Опис та архітектура мікросервісу MainAPI

Мікросервіс MainAPI є центральною ланкою екосистеми YourStream. Він відповідає за роботу з користувачами (реєстрація, аутентифікація, сесії, підтвердження дій, двофакторна автентифікація, email-розсилки та ін.),

взаємодіючи з іншими сервісами через HTTP-запити[2] та обробку подій. Сервіс реалізовано з використанням Node.js[6] (TypeScript/JavaScript)[11], Express.js[8] та MongoDB[12] (через Mongoose).

Код організовано за класичним модульним підходом:

- Контролери для кожної доменної області (auth, 2fa, session, user, confirm) винесені в окремі файли.
- Сервіси для email, кешування, підтвердження дій абстраговані через інтерфейси (IEmailService, IConfirmService, CacheService) і реєструються як singleton-и через serviceManager. Це дозволяє легко змінювати реалізацію (наприклад, switch з RedisCacheService на MemoryCacheService) без зміни основного коду.
- Моделі користувача, сесій, підписок, підтверджень винесені у відповідні файли та використовуються через Mongoose.

Для інжекції залежностей використовується сервіс-локатор serviceManager, що дозволяє централізовано змінювати реалізації сервісів (наприклад, для тестування чи різних оточень).

MainAPI не зберігає паролі користувачів, а делегує всю автентифікацію зовнішньому AuthService через REST-запити (див. використання buildServiceRequest). Аналогічно, створення чатів та стрімів делегується ChatService та StreamService.

Сервіс має ключові функціональні блоки, такі як:

а) підтвердження — ключова особливість сервісу, реалізована за допомогою шаблону "Event-Handler":

1) схема Confirm: містить унікальний токен, тип події (email, password, 2fa), контекст, дату створення та час життя (expiration);
2) обробники підтверджень: для кожного типу (наприклад, EmailConfirmationRegisterEvent, ChangePasswordConfirmationEvent, TwoFactorAuthSetupConfirmationEvent) є окремий клас-обробник з методом handle();
3) використання: Для реєстрації, зміни email, відновлення паролю, входу з

2FA[10] тощо створюється токен, який потім має бути підтверджений користувачем через email або код;

4) цікава деталь: файли обробників подій підтримують розширення, а логіка підтвердження відділена від основних контролерів, що спрощує масштабування;

б) email-сервіс:

1) реалізовано через nodemailer з підтримкою HTML-шаблонів для різних типів повідомлень;
2) всі шаблони email динамічно підставляють параметри через метадані;
3) відправка email проводиться асинхронно із обробкою помилок;

в) двофакторна автентифікація (2FA) [10]:

1) для 2FA[10] використовується бібліотека speakeasy, яка генерує та перевіряє одноразові коди;
2) 2FA[10] можна увімкнути або вимкнути через окремі ендпоінти, а статус перевіряється окремим запитом;
3) секрет для 2FA[10] зберігається у полі twoFactorAuthSecret користувача;

г) робота з сесіями:

1) кожна сесія має refreshToken, час створення, lastUsedAt, т.д;
2) додавано захист: тільки поточна сесія, яка існує > N секунд, може керувати іншими сесіями (метод canManage);
3) для оновлення токену користується окремий endpoint, який генерує новий refreshToken та JWT[3];

д) користувачі, підписки, стрім-токени:

1) реалізована CRUD логіка для користувачів, підтримка підписок, генерація та оновлення стрім-токенів;
2) операції з підписками перевіряють наявність каналу та користувача, уникається підписка на самого себе;

е) інтеграція з іншими мікросервісами:

1) для створення користувача, чату та стріму викликаються HTTP-

запити[2] до інших сервісів (AuthService, ChatService, StreamService). При неуспішній відповіді — скасовується створення користувача;

2) для підтвердження подій (restore-password, change-email тощо) використовуються email-розсилки з унікальними токенами;
3) деякі сервіси (наприклад, кешування) реалізовані через Redis, що дозволяє масштабувати інфраструктуру;

ж) цікаві технічні деталі:

1) використання DI (Dependency Injection) через serviceManager — це дозволяє легко підміняти реалізації сервісів (наприклад, для тестування чи різних середовищ);
2) шаблони email: шаблони підвантажуються з файлів, всі параметри підставляються динамічно, тема листа береться із тегу <title>;
3) динамічне підключення роутерів через файлову систему — мінімізація хардкодингу маршрутів;
4) валідація полів на всіх рівнях (ручна + Mongoose Schema[13]) — захист від ін’єкцій та некоректних даних;
5) часові обмеження для керування сесіями (canManage) — додатковий рівень безпеки;
6) підтвердження дій винесено у окрему підсистему, що спрощує додавання нових типів підтверджень;
7) всі відповіді структуровані через BaseResponse — єдиний формат для результату, помилки або додаткових даних.

Мікросервіс MainAPI спроектовано з урахуванням сучасних підходів до побудови мікросервісної архітектури: ізольованість доменів, інтеграція через HTTP API[2], використання DI, централізований логінг, fault-tolerance, розширюваність за рахунок патернів Event-Handler та Service Locator. Особливу увагу приділено безпеці (валидація, 2FA[10], сесії), масштабованості (Redis, email через SMTP), та зручності підтримки/розширення (динамічне підключення роутерів, шаблони email, розширювані підтвердження дій).

4 АРХІТЕКТУРА РОЗГОРТАННЯ ТА МАРШРУТИЗАЦІЇ СЕРВІСІВ

4.1 Огляд розгортання

Інфраструктура платформи YourStream базується на мікросервісній архітектурі, де кожен мікросервіс розгортається у власному ізольованому контейнері Docker[7]. Всі сервіси взаємодіють у спільній мережі Docker[7] (your-stream-net), що спрощує адресацію та підвищує безпеку міжкомпонентної взаємодії.

Контейнери оркеструються через docker-compose[7], що дозволяє легко керувати масштабуванням, перезапуском та залежностями між сервісами.

Основні сервіси:

- main-api — монолітний API Gateway для взаємодії з користувачами.
- auth-service — окремий сервіс для аутентифікації та управління сесіями.
- chat-service — мікросервіс для обміну повідомленнями.
- stream-api, transcoder, rtmp, hls-server — комплекс для відеострімінгу (прийом, обробка, трансляція HLS/RTMP[1][14]).
- search-service — пошуковий сервіс (наприклад, для пошуку трансляцій чи користувачів).
- service-manager-backend, service-manager-frontend — адміністративний інтерфейс та бекенд для керування сервісами.
- Frontend — інтерфейс користувача (SPA).
- mongo, redis — сервіси зберігання даних та кешування.

4.2 Організація мережі та взаємодія

Всі сервіси підключені до єдиної Docker-мережі[7] your-stream-net з типом bridge. Це дозволяє контейнерам звертатись один до одного по імені сервісу, що виключає необхідність використання зовнішніх адрес та підвищує безпеку.

Для маршрутизації зовнішніх HTTP(S)-запитів[2] використовується фронтовий nginx[15], який розгортається в окремому контейнері та слухає стандартні порти 80/443. Nginx[15] виступає єдиною точкою входу до системи, маршрутизуючи запити до конкретних мікросервісів згідно з URL-префіксами.

Ключові правила nginx[15]:

- /api/— всі основні REST-запити до API (main-api).
- /api/chat/— проксування запитів до chat-service.
- /api/stream/— запити до stream-api (відеострімінг).
- /api/search/— пошукові запити.
- /auth-key/— запити до сервісу авторизації (auth-service).
- /hls/— статичний контент HLS [1](відео).
- /— всі інші запити (статичний frontend, SPA).

Приклад конфігурації nginx[15] (фрагмент):

```
location /api/ {
    proxy_pass http://main-api/api/;
}

location /api/chat/ {
    proxy_pass http://chat-service/api/chat/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}

location /api/stream/ {
    proxy_pass http://stream-api/api/external/;
}

location / {
    proxy_pass http://frontend/;
}
```

Для підтримки відеострімінгу застосовується декілька рівнів обробки:

- rtmp (на базі nginx-rtmp[14]): приймає вхідні RTMP-потоки, прокидає події on_publish/on_publish_done до stream-api для авторизації та реєстрації стріму.

- transcoder: отримує потік, виконує перекодування у HLS[1], розміщує плейлисти та сегменти у /tmp/hls.
- hls-server: окремий nginx[15], який віддає HLS-файли[1] кінцевим користувачам через /hls/.
- frontend: відтворює HLS-потік[1] через відеоплеєр.

У системі реалізовані додаткові маршрути, що не є публічними:

- Service Manager: окремий nginx[15] для адмінки, який проксує /api/ до backend-адмінки, а всі інші запити віддає як SPA.
- MongoDB[12], Redis: не публікуються зовні, доступні лише у внутрішній мережі для backend-сервісів.

Усі сервіси мають явно описані залежності (depends_on). Наприклад, main-api стартує лише після запуску основних бекенд-сервісів та сховищ, що гарантує коректне підняття всієї систем.

Зовнішні порти відкриті лише для nginx[15] (80/443), frontend (3000 для розробки), service-manager-frontend (8019). Всі внутрішні сервіси взаємодіють по приватній мережі. База даних та кеш не доступні ззовні.

Така архітектура дозволяє легко масштабувати сервіси, швидко впроваджувати нові компоненти, оновлювати окремі частини системи без впливу на інші. Розподіленість та чітка маршрутизація забезпечують високу доступність та fault-tolerance, а єдина точка входу (nginx[15]) — спрощує управління безпекою та SSL-термінацією.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для тестування системи YourStream використовувалося мануальне тестування. Основними підвидами такого тестування були:

– функціональне тестування — перевірка коректності виконання бізнес-логіки (наприклад, створення та запуск трансляції, взаємодія з чатом, авторизація користувачів);

– нефункціональне тестування — перевірка швидкодії, стабільності та зручності інтерфейсу (включаючи адаптивність вебзастосунку до різних екранів та коректність роботи відеоплеєра);

– інтеграційне тестування — перевірка взаємодії між мікросервісами, таких як AuthService, StreamAPI, ChatService, зокрема перевірка цілісності токенів та передачі даних через HTTP/WebSocket;

– перевірка на відмовостійкість — симуляція збоїв у відеопотоці, недоступності окремих сервісів та спостереження за стабільністю платформи;

– тестування граничних випадків — перевірка поведінки системи при неправильному введенні даних (некоректні email, порожні поля, неіснуючі URL), перевірка обробки виключень.

Автоматизовані тести не були реалізовані у межах кваліфікаційної роботи через обмежений термін розробки, а також високий пріоритет функціонального прототипування. Водночас, архітектура системи дозволяє легко інтегрувати засоби автоматизованого тестування (наприклад, за допомогою Jest для Unit/Integration тестів на рівні API та Playwright або Cypress для UI-тестів).

ВИСНОВКИ

Архітектура платформи YourStream є яскравим прикладом сучасної мікросервісної системи, побудованої із застосуванням технологій контейнеризації (Docker[7]) та ізоляції компонентів у межах спільної внутрішньої мережі. Всі основні елементи — від API Gateway до стрімінгових, чат- та пошукових сервісів — взаємодіють через стандартизовані протоколи REST API, WebSocket[5] або RTMP[14], а маршрутизація зовнішніх HTTP/HTTPS-запитів[2] централізовано здійснюється проксі-сервером nginx.

Взаємодія між сервісами реалізована через внутрішню мережу your-stream-net, що дозволяє виключити зовнішнє адресування та забезпечити мережеву ізоляцію. Це підвищує безпеку та спрощує адміністрування, оскільки зовнішній доступ дозволений лише до строго визначених точок входу (nginx, rtmp, hls).

Логічна структура системи сегментована на спеціалізовані пакети:

- Gateway & Entry Points (nginx, rtmp, hls-server) — єдині вхідні точки для HTTP[2] та відеопотоків;
- Stream Processing (stream-api, transcoder) — обробка трансляцій та керування якістю потоків;
- Service Management — централізоване управління сервісами та видання токенів;
- Authentication & Authorization — аутентифікація користувачів і сервісів через JWT[3];
- Core Services — робота з користувачами, сесіями, підтвердженнями, 2FA[10];
- Search, Chat, Frontend — окремі сервіси для пошуку, спілкування і користувацького інтерфейсу;
- Data Stores (MongoDB[12], Redis) — збереження та кешування даних.

Безпека системи забезпечується на декількох рівнях:

- Усі сервіси ізольовані у внутрішній мережі;
- Міжсервісна автентифікація реалізована через централізовану видачу JWT-токенів[3], перевірку підпису і обмеження часу дії;

- Кожен мікросервіс має свій унікальний ключ та ідентифікатор, а всі критичні запити захищені middleware;
- Для перевірки користувачів використовуються окремі токени, а логіка 2FA[10] і підтвердження дій винесена на рівень Core Services;
- Додавання нових сервісів чи розширення функціоналу можливе лише через ізольований адміністративний інтерфейс.

Core-бібліотека забезпечує стандартизацію спільних механізмів: автентифікацію сервісів, валідацію токенів, централізований логінг, єдину структуру відповідей, що суттєво спрощує розробку, спільне оновлення та тестування мікросервісної екосистеми.

Клієнтський інтерфейс реалізований на базі Next.js та React[16] із чітко структурованою маршрутизацією та зручними сценаріями взаємодії для користувача. Для основних сторінок і сценаріїв розроблено UML-діаграми, що систематизують вимоги до функціоналу.

Загалом, така архітектура забезпечує:

- гнучке масштабування кожного компонента;
- високу надійність та fault-tolerance;
- простоту розгортання, оновлення і супроводу;
- чіткі зони відповідальності та контроль доступу;
- швидке впровадження нових функцій із мінімальними ризиками для стабільності всієї системи.

Побудова YourStream на основі мікросервісів, ізоляції, централізованої безпеки та повторного використання Core-компонентів створює надійну основу для розвитку, модернізації та розширення функціоналу у відповідь на потреби користувачів і бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Contributors to Wikimedia projects. HTTP live streaming - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/HTTP_Live_Streaming (дата звернення: 13.05.2025).
2. Contributors to Wikimedia projects. HTTP - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/HTTP (дата звернення: 13.05.2025).
3. Contributors to Wikimedia projects. JSON web token - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 13.05.2025).
4. Contributors to Wikimedia projects. SHA-2 - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/SHA-2 (дата звернення: 13.05.2025).
5. Contributors to Wikimedia projects. WebSocket - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/WebSocket (дата звернення: 13.05.2025).
6. Crypto | node.js v24.0.2 documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/ (дата звернення: 13.05.2025).
7. Docker: accelerated container application development. *Docker*. URL: https://www.docker.com/ (дата звернення: 13.05.2025).
8. Express - Node.js web application framework. *Express - Node.js web application framework*. URL: https://expressjs.com/ (дата звернення: 13.05.2025).
9. FFmpag. *FFmpag*. URL: https://ffmpeg.org/ (дата звернення: 13.05.2025).
10. Help:two-factor authentication - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Help:Two-factor_authentication (дата звернення: 13.05.2025).
11. JavaScript with syntax for types. *TypeScript: JavaScript With Syntax For Types.* URL: https://www.typescriptlang.org/ (дата звернення: 13.05.2025).
12. MongoDB: the world’s leading modern database. *MongoDB*. URL: https://www.mongodb.com/ (дата звернення: 13.05.2025).
13. Mongoose ODM v8.14.3. *Mongoose ODM v8.14.3*. URL: https://mongoosejs.com/ (дата звернення: 13.05.2025).

14.*nginx-rtmp*. URL: https://hub.docker.com/r/tiangolo/nginx-rtmp/ (дата звернення: 13.05.2025).

15.*nginx*. URL: https://nginx.org/ (дата звернення: 13.05.2025).

16.React. *React*. URL: https://react.dev/ (дата звернення: 13.05.2025).

17. GitHub Репозиторій з роботою. *GitHub*. URL: https://github.com/NureVovkDmytro/B_PI_PZPI-21-6_Vovk_D_A (дата звернення: 07.06.2025).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

StrikePlagiarism.com

Дата звіту 6/5/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ-21-6_Вовк_Д_А_скорочений
Автор
Вовк Дмитро Андрійович
Науковий керівник / Експерт
Євген Кардаш
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.29% 0.29% КП 1

0.92% 0.92% КЦ

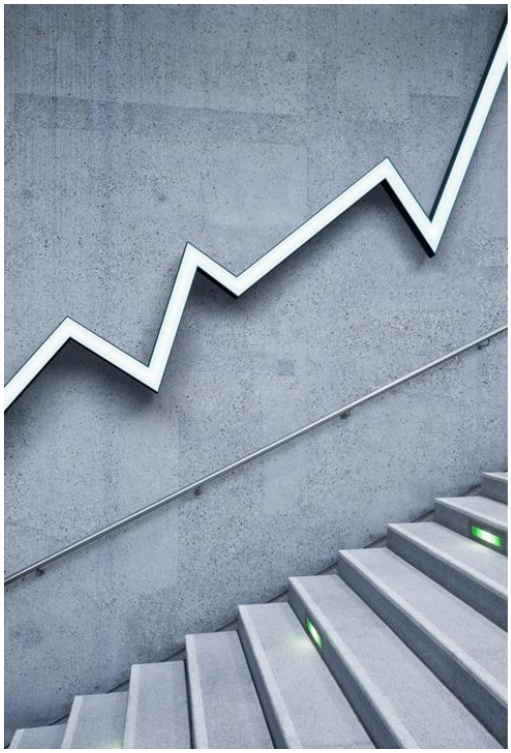
25
Довжина фрази для коефіцієнта подібності 2

6199
Кількість слів

50043
Кількість символів

ДОДАТОК Б

Слайди презентації



Your Stream

Виконав:
ст. гр. ПЗПІ-21-6
Вовк Д.А.

Науковий керівник:
к.т.н., доц.
Ворочек О.Г.

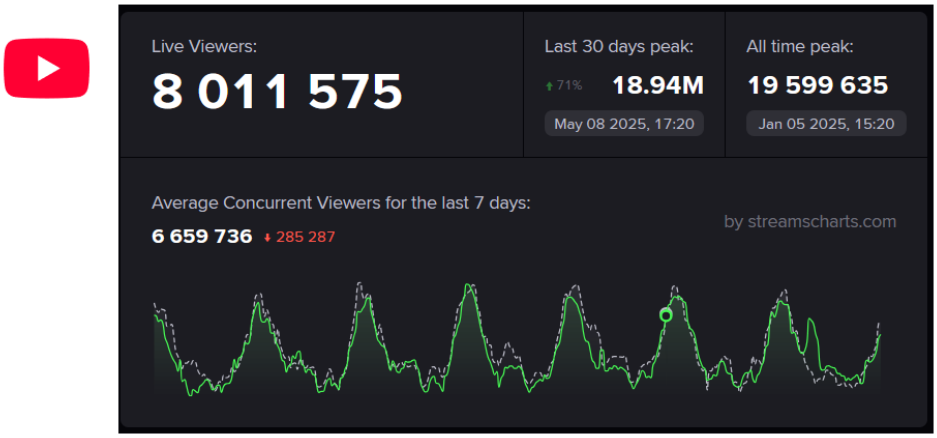
Актуальність теми

- Онлайн-відеотрансляції стали важливою частиною цифрової інфраструктури
- Зростання популярності стримінгових платформ (Twitch, YouTube Live тощо)
- Потреба у передачі відео в реальному часі з мінімальними затримками
- Високі вимоги до якості відео, інтерактивності та безпеки користувачів
- Необхідність створення гнучких, масштабованих та захищених рішень

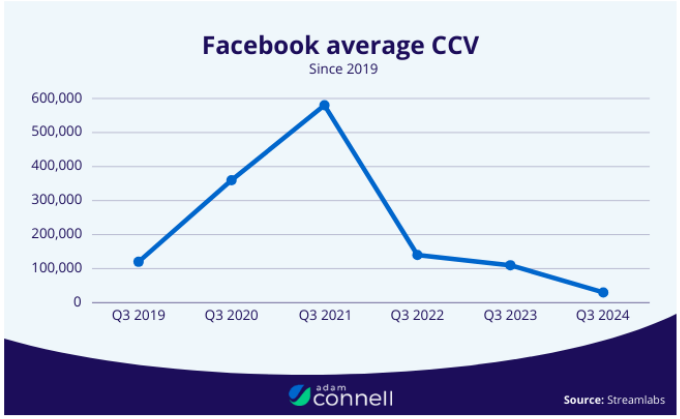
Основні завдання

- Побудова мікросервісної архітектури
- Реалізація адаптивного стримінгу відео
- Створення інтерактивного чату з модерацією
- Впровадження системи авторизації з 2FA
- Розробка функціоналу пошуку трансляцій за тегами
- Створення зручного веб-інтерфейсу на React

Аналіз предметної області

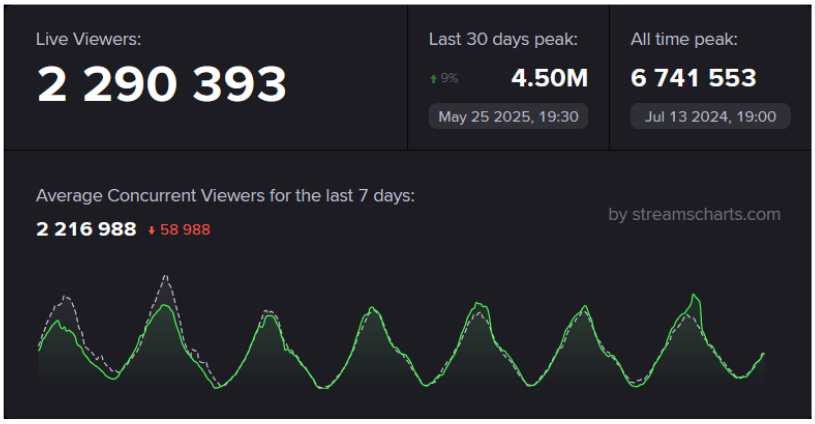


Аналіз предметної області



5

Аналіз предметної області



6

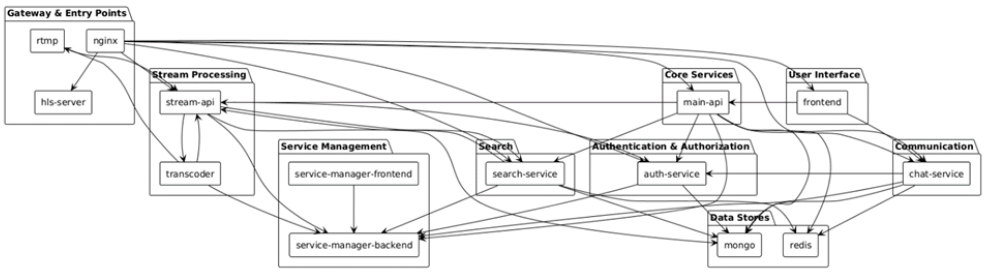
Цільова аудиторія

- Стрімер
  - трансляції, чат, модерація
- Глядач
  - вибір якості, пошук, чат
- Модератор
  - блокування, фільтрація, контроль

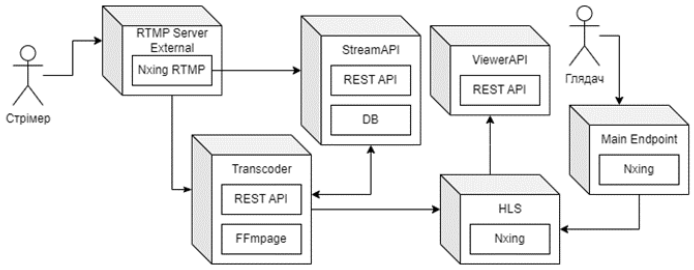
Ключові проблеми та рішення

🔴 Проблема	✅ Рішення
Високе навантаження під час стримів	Мікросервісна архітектура
Нестабільна якість відео	Адаптивне потокове мовлення (HLS)
Затримки та спам у чаті	Інтерактивний чат з модерацією
Небезпека для облікових записів	2FA, захист даних, авторизація через JWT
Складність пошуку трансляцій	Теги, фільтрація, пошуковий сервіс

Архітектура платформи



Архітектура платформи



Архітектура платформи

- Кожен сервіс — окремий контейнер
- Взаємодія — через REST/WebSocket
- Безпечна комунікація (JWT, Docker Network)

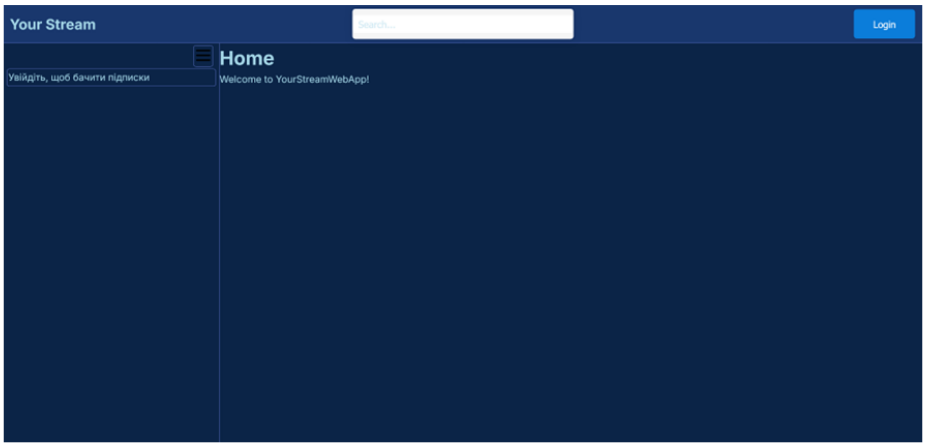
Ключова функціональність платформи

- Прямі трансляції — у реальному часі, з вибором якості
- Чат — інтерактивний, з модерацією
- Авторизація — реєстрація, вхід, 2FA
- Пошук — за ключовими словами та тегами
- Масштабування — завдяки мікросервісам
- Вебінтерфейс — зручний і швидкий (React)

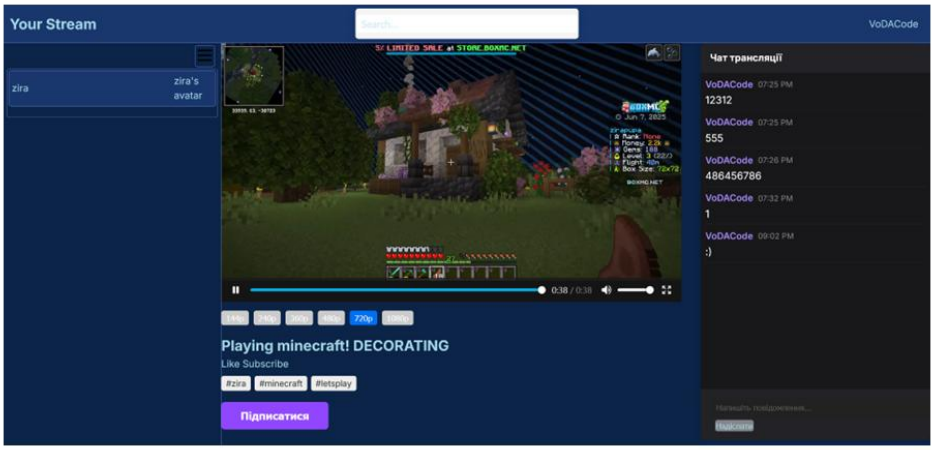
Інтерфейс користувача (UI) Технології

- React + Next.js
- Хуки для авторизації, чату, користувача

Інтерфейс користувача (UI) Головна сторінка

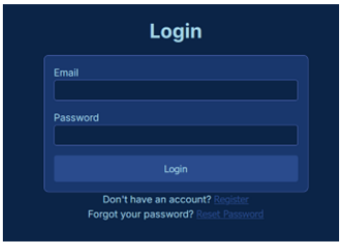


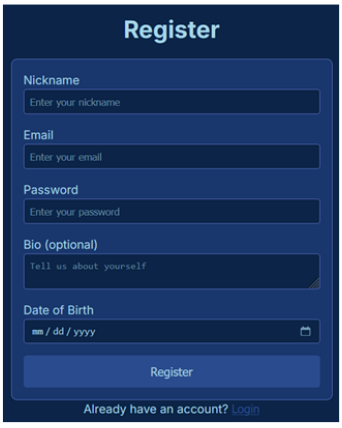
Інтерфейс користувача (UI) Сторінка трансляції



15

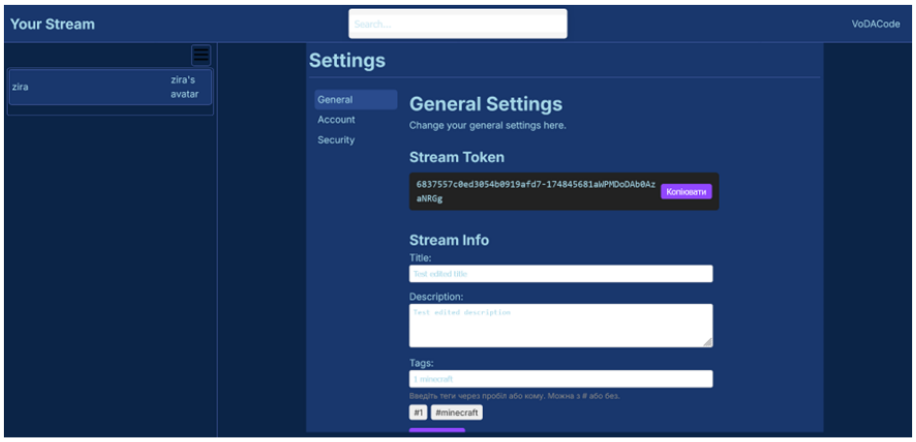
Інтерфейс користувача (UI) Логін/Реєстрація



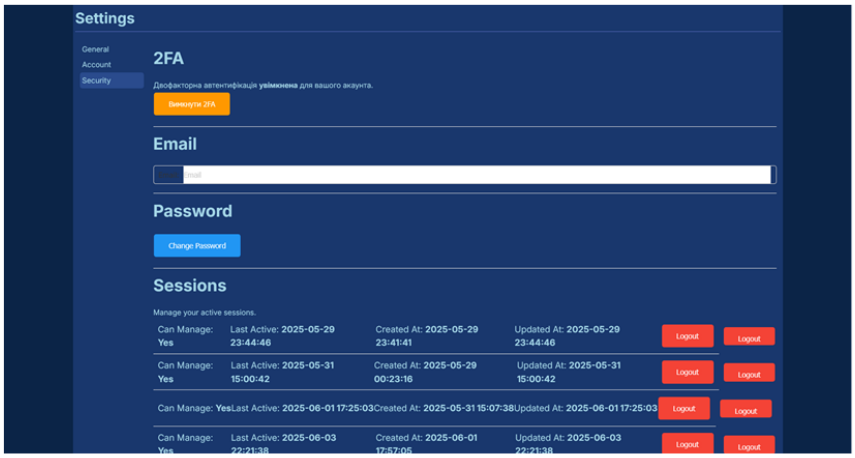


16

Інтерфейс користувача (UI) Налаштування



Інтерфейс користувача (UI) Налаштування



Безпека системи

Рівні безпеки:

- **Користувачі**
  - Авторизація з JWT
  - 2FA (одноразові коди)
  - Захист персональних даних
- **Сервіси**
  - Взаємна автентифікація через токени
  - Обмеження доступу (мережа Docker)
- **Комунікації**
  - HTTPS / TLS
  - Ізоляція баз даних (MongoDB, Redis)
  - Короткоживучі токени

19

Розгортання платформи

- **Docker + Docker Compose** — ізоляція кожного сервісу
- **Єдина внутрішня мережа** (your-stream-net)
- **Вхідні точки**:
  - Nginx (HTTP/HTTPS)
  - RTMP-сервер (відео потоки)
  - HLS-сервер (доставка відео)

20

Висновки

- ✅ Розроблено повнофункціональну стримінгову платформу
- ✅ Застосовано мікросервісну архітектуру
- ✅ Забезпечено безпеку: JWT, 2FA, TLS
- ✅ Реалізовано адаптивне відео, чат, пошук
- ✅ Платформа легко масштабується та підтримує навантаження
- ✅ Інтерфейс — зручний, на React