МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проект

здобувача освіти денної форми навчання
РП.08.17.000.ДП

ПЯТНІЧЕНКО
ДМИТРІЯ ОЛЕКСАНДРОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: **«Розробка програмного забезпечення»**

Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка програмного забезпечення для керування підписками користувачів

Проектний матеріал складається з пояснювальної записки на 88 сторінках та графічного (презентаційного) матеріалу на 14 аркушах (слайдах)

Дипломник _____ (Пятніченко Д.О.)

Керівник _____ (Залапін О.І.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «21» 06 2025 р. Протокол ЕК № 1

Оцінка ЕК 5/95

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного зазабезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Пятніченко Дмитрій Олександрович*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка програмного забезпечення для керування підписками користувачів*

затверджена наказом по коледжу від "14" 11 2024 р. № 246

2. Термін здачі закінченого проекту

3. Вихідні данні до проекту *1. Використовувати REST API для взаємодії між фронтендом та бекендом; 2. Реалізувати збереження відеофайлів і зображень на хмарному сховищі (Cloudinary); 3. Програмне забезпечення реалізувати засобами Django та React у середовищі розробки Visual Studio Code, з інтеграцією з СУБД MySQL та платіжною системою Stripe; 4. Забезпечити підтримку ролей користувачів, фільтрації контенту за підпискою, збереження прогресу перегляду та взаємодію з коментарями*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз існуючих рішень у сфері відеоплатформ з підписками; Розробка структури бази даних для зберігання користувачів, відео та коментарів; Алгоритм контролю доступу до відео на основі рівня підписки; Реалізація адміністративної панелі для керування користувачами та підписками; Тестування інтерфейсів користувача, включаючи відеоплеєр; Економічний розділ; Розділ охорони праці і техніки безпеки

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Методологічна схема порівняльного аналізу платформ відеохостингу; ER-діаграма бази даних користувачів, відео та підписок; Блок-схема перевірки доступу до відеоконтенту за підпискою; Блок-схема алгоритму моніторингу прогресу перегляду відео; Алгоритм роботи панелі керування користувачами та підписками; Структурна схема архітектури веб-платформи; Блок-схема процесу оформлення та обробки підписки через Stripe

6.Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Залапін О.І*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник *Залапін О.І.* _____
(підпис)

Завдання прийняв до виконання *Пятніченко Д.О.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Постановка задачі проектування*	15.05.2025	виконано
2	*Аналіз сучасних відеоплатформ з підписками*	16.05.2025	виконано
3	*Визначення структури бази даних*	17.05.2025	виконано
4	*Реалізація системи ролей користувачів та сесійної*	19.05.2025	виконано
	автентифікації		
5	*Розробка алгоритму контролю доступу до відео*	22.05.2025	виконано
	за рівнем підписки		
6	*Інтеграція з Cloudinary для зберігання відео та*	26.05.2025	виконано
	зображень		
7	*Інтеграція з Stripe: створення підписок поповнення*	01.06.2025	виконано
	балансу, webhook		
8	*Реалізація інтерфейсу адміністратора*	06.06.2025	виконано
9	*Розробка користувацького інтерфейсу: профіль,*	10.06.2025	виконано
	плеєр, сторінка підписки		
10	*Збереження прогресу перегляду відео*	11.06.2025	виконано
11	*Аналіз ефективності та продуктивності платформи*	12.06.2025	виконано
12	*Виконання економічних розрахунків*	13.06.2025	виконано
13	*Розробка заходів з охорони праці*	14.06.2025	виконано
14	*Підготовка доповіді та презентації для захисту*	16.06.2025	виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Форма	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 17 000. 00 ДП	Дипломний проект		
А4			РП 08. 17 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 17 000. 00 ДП		
Розробив		Пятніченко Д.О		18.06.25	Розробка програмного забезпечення для керування підписками користувачів	Літ.	Аркуш 4 / Аркушів 88
Перевірив		Залапін О.І		18.06.25		Н Д П	
Н. Контр.		Петрашова В.І.		18.06.25		ВСП "ОТФК ОНТУ" гр. 4РП-08	
Затверд.		Кривченко Ю.В.		18.06			

ЗМІСТ

ВСТУП

Сучасний цифровий ринок динамічно змінюється, і одним із ключових трендів є зростання популярності онлайн-відеоконтенту. Це зумовлює необхідність розробки ефективних веб-рішень, здатних забезпечити стабільний доступ до мультимедійних матеріалів через систему підписок. Критично важливими аспектами таких платформ є безпека даних, стабільність роботи та зручність управління обліковими записами. Особливо це стосується сервісів, які використовують зовнішні хмарні сховища для медіа та інтегровані платіжні шлюзи, оскільки їхня неправильна налаштування може спричинити технічні збої, фінансові ризики та незадоволеність клієнтів. Крім того, автоматизація процесів підписки та контролю доступу до контенту є невід'ємною частиною сучасних відеосервісів, адже ручне керування цими функціями значно знижує ефективність платформи. Додатковою складовою є масштабованість системи, оскільки кількість користувачів і обсяги контенту постійно збільшуються ,тому вимагають гнучкої архітектури.

У даному дипломному проекті пропонується розробка веб-застосунку для управління підписками на відеоконтент, який автоматизує надання доступу відповідно до обраних тарифних планів та забезпечує надійне зберігання відео за допомогою хмарного сервісу Cloudinary. Інтеграція зі Stripe дозволить реалізувати безпечні транзакції, включаючи оплату підписок, поповнення балансу та керування статусом облікового запису. Окрема увага приділяється створенню зручного інтерфейсу для перегляду відео з функцією збереження прогресу, що підвищить комфорт користувачів.

Результатом проекту є вдосконалена адміністративна панель для управління користувачами, аналізу статистики та контролю доступу до контенту залежно від типу підписки. Це рішення зменшує вплив людського фактора, підвищує якість обслуговування та оптимізує функціонування платформи. Впровадження продукту сприяє розвитку сучасних онлайн-сервісів із підписками.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз існуючих платформ для відеохостингу з підтримкою підписок

У сучасних платформах для відеохостингу активно використовують модель підписки як спосіб монетизації та надання доступу до контенту. В теперішніх умовах швидкого розвитку медіа-ресурсів особливо актуальним є створення сервісів, які забезпечуватимуть зручний доступ до відеоматеріалів, а також реалізують ефективні механізми контролю доступу. Серед найпоширеніших рішень сьогодні можна виділити YouTube Premium, Patreon та Twitch, кожна з яких має власні особливості, переваги та недоліки.

YouTube Premium може запропонувати користувачам доступ до ексклюзивних відео, можливість перегляду контенту без реклами та підтримку фонового режиму. Patreon орієнтований на підтримку авторів контенту через підписки, надаючи доступ до ексклюзивних матеріалів, обмежених для широкої аудиторії. Twitch, у свою чергу, пропонує модель щомісячних підписок, яка дозволяє користувачам підтримувати контент-мейкерів.

1.1.1 Огляд основних платформ та їх порівняльний аналіз

У рамках дослідження було розглянуто три найвпливовіші відеосервіси сучасності: YouTube як відеохостинг, Patreon як платформа для фан-підтримки та Twitch у якості лідера сегменту стрімінгу. Вибір саме цих майданчиків не є випадковим - разом вони охоплюють понад 80% ринку відеоконтенту, формуючи основні тенденції галузі.Хоча всі три сервіси по суті є посередниками між творцями та їх аудиторією, кожен з них пропонує унікальний підхід до організації цього процесу.

YouTube Premium розширює функціонал базової платформи YouTube, пропонуючи підписку з функцією “реклама-мінус”, яка вимикає всі рекламні формати (pre-roll, mid-roll, банери) на стороні клієнта, забезпечуючи безперервне відтворення контенту без затримок. Додатково сервіс надає доступ до ексклюзивного контенту та офлайн-перегляду. Інфраструктурно платформа

спирається на розподілену файлову систему Colossus і керується за допомогою Borg-контейнерів, що гарантує гео-реплікацію відеофайлів із рівнем доступності не нижче 99,99 %.

Для доставки даних використовується багатошарова CDN-мережа Google Global Cache із підтримкою протоколу QUIC над UDP, що знижує час встановлення з'єднання (RTT) і адаптує бітрейт у режимі реального часу (MPEG-DASH або HLS), враховуючи показники мережної латентності та втрати пакетів.

Водночас творці контенту змушені дотримуватися суворих правил Partner Program: від технічних вимог до форматів відео (не менше H.264 High Profile Level 4.2 із мінімальною бітовою швидкістю 4 Мбіт/с) до контент-політик (advertiser-friendly). Рекомендаційний алгоритм, побудований на TensorFlow Extended із графами знань і click-stream аналізом, може приглушувати видимість роликів, котрі не відповідають внутрішнім критеріям engagement-rate і session-duration, що змушує творців проводити A/B-тести назв, thumbnail-зображень та тривалості відео для досягнення оптимального ранжування.

Для авторів контенту YouTube Premium відкриває альтернативне джерело доходу у вигляді частки від платіжного потоку підписників. Ця модель доповнює вже існуючі механізми монетизації (AdSense-аукціони, Super Chat, Channel Memberships, Merch Shelf), забезпечуючи більш передбачуваний грошовий потік: частина плати за кожен місяць перераховується прямо в пул творців, пропорційно загальному часу перегляду їхнього контенту.

Перед кожним відтворенням офлайн-контенту клієнт ініціює handshake із сервером авторизації, під час якого перевіряється валідність токена доступу, зокрема його цифровий підпис та термін дії. У разі наближення дати завершення підписки, система автоматично формує запит на поновлення ліцензії, що дозволяє продовжити перегляд без втручання користувача. Така схема забезпечує динамічне оновлення прав доступу та унеможливлює накопичення дешифрувальних ключів у локальному сховищі.

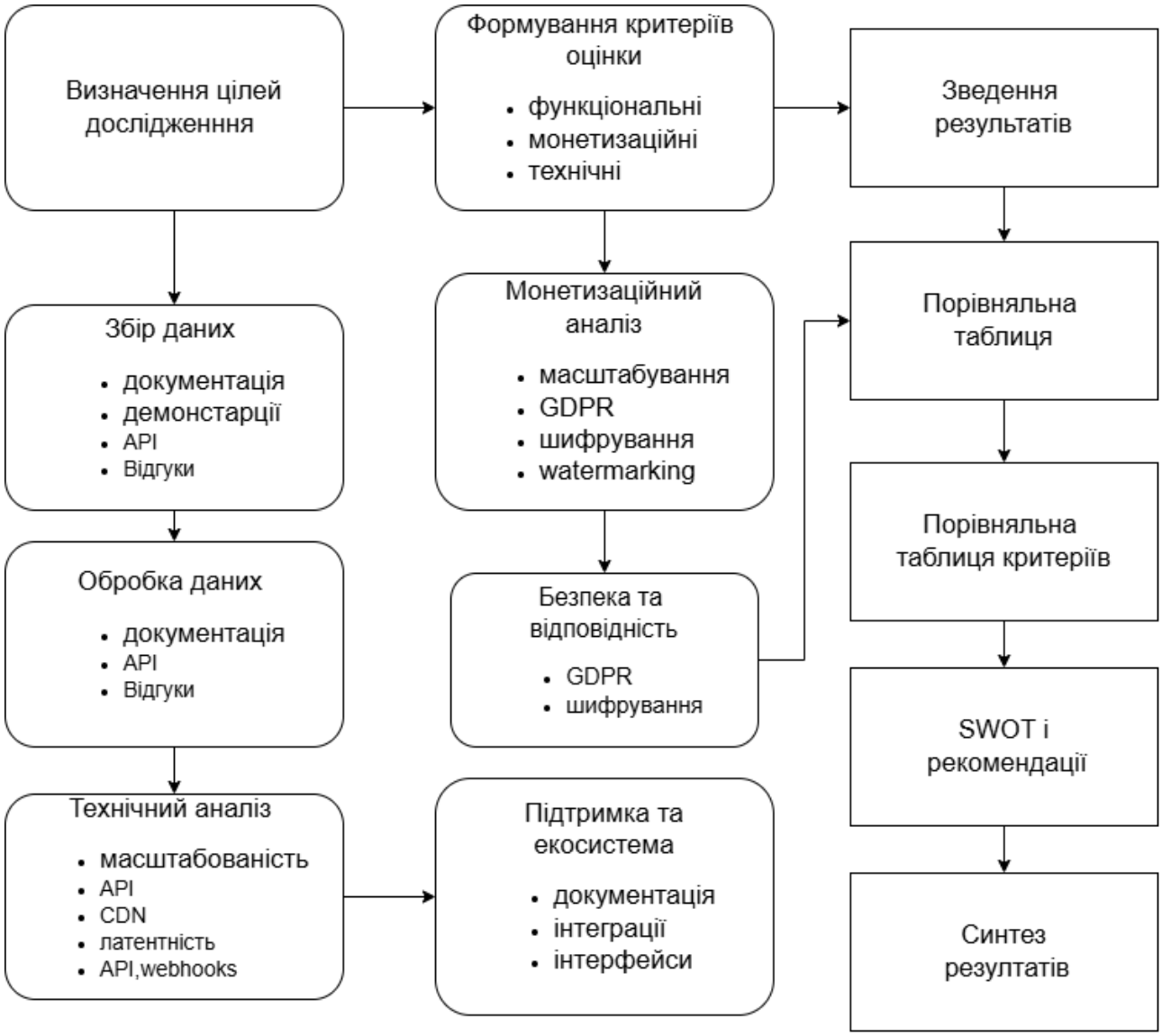


Рисунок 1.1 Методологічна схема порівняльного аналізу платформ відеохостингу

Patreon впроваджує багаторівневу модель патронажу, що сприяє тісній взаємодії між творцями контенту та їхньою аудиторією шляхом вибору рівнів підписки з унікальними привілеями, такими як ексклюзивний контент чи доступ до закритих спільнот, спираючись на мікросервісну архітектуру, поділену на домени, з REST-API для керування підписками, платежами та профілями, а також Webhook-механізм для миттєвих сповіщень про події (pledge.created, pledge.updated, pledge.deleted), що забезпечує інтеграцію з зовнішніми аналітичними системами, CRM-платформами та інструментами автоматизації маркетингу, гарантуючи безпеку транзакцій за стандартами PCI DSS та високу масштабованість (рисунок. 1.1).

Twitch спеціалізується на потокових трансляціях у реальному часі із збереженням записів (VOD), пропонуючи підписникам безрекламний перегляд, ексклюзивні емодзі, бейджі, пріоритет у чаті та можливість миттєвої винагороди стрімерів через донати й валюту Bits. Інфраструктура платформи базується на багаторівневому CDN із POP-вузлами, оптимізованими для низької латентності та стабільності, використовує протокол RTMPS для безпечної передачі відеопотоку та WebSocket API для керування сесіями з автоматичним масштабуванням інстансів (рисунок. 1.1).У межах порівняльного аналізу особливу увагу приділено архітектурним та протокольним аспектам реалізації підписної моделі кожної платформи (табл. 1.1, рисунок. 1.1). YouTube Premium інтегрує механізм підписки безпосередньо на рівні облікового запису Google, використовуючи єдиний протокол OAuth 2.0 для аутентифікації і авторизації користувача. Такий підхід забезпечує наступні технічні переваги:

- Уніфіковане керування правами доступу через Google Identity Platform, що дозволяє сервісу автоматично відновлювати токени і синхронізувати статус підписки між різними сервісами .
- Масштабованість інфраструктури завдяки кластеризації мікросервісів у Kubernetes на базі Borg; кожний мікросервіс відповідає за окремий набір операцій;
- Безперервне оновлення метаданих підписки у розподілених кешах (Memcached, Redis), що мінімізує затримки під час запитів до API для перевірки прав на відтворення.

Цей підхід, хоча й простий та швидкий, не забезпечує повної класифікації додатків. Менш ефективний метод, заснований на аналізі вхідного інтерфейсу трафіку, має обмежену застосовність. Базові методи, що використовують IP-адреси чи протоколи, обмежені в функціональності, оскільки аналізують лише заголовки IP. Більш точні підходи, такі як аналіз шаблонів трафіку або застосування статистичних методів, можуть підвищити ефективність класифікації, проте потребують додаткових обчислювальних витрат.

Patreon використовує кастомізований стек із комбінацією Ruby on Rails та Node.js-сервісів. Ключові технічні особливості:

- Мультирангові рівні доступу, реалізовані через реляційну модель у PostgreSQL: таблиця tiers містить поля tier_id, price_cents, benefits (JSONB), а зв'язок creator_id → tiers дозволяє гнучко налаштовувати політику доступу.
- Webhook-події (pledge.created, pledge.updated, pledge.deleted) передаються через захищений HTTPS-канал із підписом HMAC-SHA256, що гарантує цілісність і достовірність повідомлень.
- Плагінна архітектура клієнтської частини на React із Redux-стором для керування станом авторизації та підписок у реальному часі, що спрощує розробку кастомних інтерфейсів для різних авторів.
- Twitch реалізує модель підписки через шлюз Amazon Prime (Twitch Prime) та власний платіжний сервіс:
- Інтеграція з Amazon Cognito забезпечує єдину точку входу для користувачів Prime, які отримують щомісяця одну безкоштовну преміум-підписку.
- Розподілена CDN-мережа на базі Amazon CloudFront із локальними POP-вузлами, що знижує час встановлення з'єднання (TTFB) та обробляє сотні тисяч одночасних стрімів.
- Сервіс Bits, побудований на AWS Lambda та DynamoDB, підтримує масштабовані мікротранзакції в реальному часі з підтвердженням через WebSocket API.

Аналіз показує, що жодна з платформ не здатна повною мірою задовольнити всі потреби одночасно: YouTube Premium забезпечує стабільність і глибоку інтеграцію з екосистемою Google, але вимагає дотримання суворих правил; Patreon надає творцям гнучкість у монетизації, проте потребує активної участі у просуванні; Twitch пропонує високу інтерактивність і низьку затримку, однак стягує значну комісію з доходів. Отже,вибір платформи передбачає баланс між контролем, витратами, зусиллями автора і потенціалом масштабування.

Таблиця 1.1 Порівняльна характеристика реалізації підписки

Платформа	*Аутентифікація та авторизація*	*Механізм оновлення статусу*
YouTube Premium	OAuth 2.0 (Google Account)	Розподілені кеші + background jobs
Patreon	OAuth / Custom JWT	Webhook HMAC + polling
Twitch	Amazon Cognito	Серверні події через EventBridge

OAuth 2.0 — це відкритий протокол авторизації, який дозволяє стороннім застосункам отримувати обмежений доступ до захищених ресурсів користувача без необхідності передачі його облікових даних. Протокол реалізується на базі системи делегування прав доступу, де користувач виступає як суб'єкт дозволу, а сторонній сервіс діє від його імені у межах заданих повноважень. Найчастіше OAuth 2.0 використовується у зв'язці з сервісами Google, Facebook, Microsoft, де він забезпечує логінізацію.

Протокол підтримує кілька моделей доступу: Authorization Code Flow, Implicit Flow, Resource Owner Password Credentials та Client Credentials Flow. Найбільш безпечним вважається Authorization Code Flow із використанням Proof Key for Code Exchange (PKCE), що дозволяє уникнути атак типу interception у клієнтських застосунках. Усі запити повинні бути захищені через HTTPS, а доступ обмежується через scope — область прав доступу, яку визначає користувач під час авторизації.

JWT - це самодостатній маркер автентифікації, який дозволяє зберігати та передавати інформацію між сторонами у форматі JSON з цифровим підписом. Його структура складається з трьох частин: заголовка (header), тіла (payload) та підпису .У заголовку визначається тип токена (типово "JWT") та алгоритм підпису (наприклад, HMAC SHA256 або RSA). У тілі зберігаються claims — тобто структуровані дані про користувача, тривалість дії токена, призначення, ідентифікатори сесії тощо.

Підпис токена генерується за допомогою секретного ключа (HMAC) або приватного ключа (RSA/ECDSA) і дозволяє приймаючій стороні перевірити автентичність та цілісність токена без запиту до централізованого серверу. JWT використовується у системах мікросервісів, CDN, IoT-пристроях, а також для керування доступом до API. Токени передаються через заголовки типу Bearer або зберігаються у cookie/localStorage. Основним недоліком є складність відкликання вже виданого токена, якщо не реалізований blacklist-механізм.Іншим перспективним підходом є числовий аналіз параметрів. Наприклад, у типовому сценарії запит клієнта може містити повідомлення обсягом 18 байтів, тоді як відповідь сервера — 11 байтів. Аналіз цих показників може потребувати більше часу, адже включає кілька пакетів.

Amazon Cognito — це керована хмарна служба автентифікації, створена Amazon Web Services (AWS) для організації безпечного та масштабованого доступу до веб- і мобільних застосунків. Cognito забезпечує централізоване управління ідентичностями користувачів, підтримуючи такі сучасні стандарти автентифікації, як OAuth 2.0, OpenID Connect (OIDC), SAML 2.0, а також інтеграцію з соціальними провайдерами (Google, Facebook, Apple) і корпоративними каталогами через SAML-автентифікацію.

Фундаментом авторизаційної архітектури Cognito є User Pools та Identity Pools. User Pool — це сервіс для реєстрації, входу та керування профілями користувачів із можливістю налаштування політик паролів, MFA (Multi-Factor Authentication), email-підтвердження та CAPTCHA-захисту. Після успішної автентифікації користувач отримує три типи токенів:

- ID Token (JWT-формат) — містить атрибути користувача, такі як email, ім’я, час створення, призначений для передачі на сторонні сервіси;
- Access Token — використовується для авторизованого доступу до захищених ресурсів або API;
- Refresh Token — дозволяє оновлювати ID та Access токени без повторного входу.

1.1.2 Аналіз моделей монетизації та механізмів контролю доступу до контенту маршрутизаторів

Моделі монетизації у відеохостингових платформах, що підтримують підписку, базуються на принципах прямих та непрямих джерел доходу. Основні механізми включають: платну підписку, мікроплатежі, пряму підтримку контент-мейкерів (донати), вбудовану рекламу, преміальні функції та моделі pay-per-view. Одночасно реалізуються засоби контролю доступу, які гарантують, що перегляд, кешування чи копіювання матеріалів можливе лише в межах дозволених сценаріїв. Нижче розглянуто ключові моделі монетизації та способи реалізації доступу до відеоконтенту.

1. Платна підписка (Subscription-based Access). Це найпоширеніший механізм серед платформ на кшталт YouTube Premium, Patreon, Netflix, Spotify Video та ін. Користувач щомісяця або щорічно оплачує фіксовану суму та отримує доступ до закритого контенту. Технічно реалізується через систему керування підписками, яка асоціює унікальний ідентифікатор користувача (ID або email) з певним рівнем доступу, що зберігається у базі даних або в токені.

Під час запиту на відтворення відео сервер виконує перевірку валідності токена, рівня підписки та її статусу (активна, закінчена, скасована). У разі відповідності вимогам користувач отримує тимчасовий доступ до ресурсів CDN або S3, іноді — із генерацією signed URL на основі HMAC чи RSA-підпису.

2. Модель «Freemium» поєднує безкоштовний доступ до обмеженого контенту з можливістю розблокування додаткових функцій або матеріалів шляхом підписки.Контроль доступу базується на сегментації користувачів за тарифами або рівнями, що часто реалізується через систему ACL .

У таких моделях зберігається кешована версія профілю користувача у Redis або Memcached, що дозволяє мінімізувати час доступу до метаінформації та прискорює авторизацію при запитах до API. Це особливо важливо для масштабованих систем з великою кількістю одночасних сесій. Завдяки цьому зменшується навантаження на основну базу даних.

3. Pay-per-view (одноразовий доступ) передбачає оплату за кожен окремий елемент контенту. Наприклад, деякі освітні або кіноплатформи (Udemy, Google Play Movies) дозволяють разову покупкучиорендувідео.Сервіс генерує одноразовий токен або signed URL із часовим обмеженням дії (наприклад, TTL=3600 сек.). Така URL включає шифрований контрольний підпис, за яким бекенд перевіряє авторизацію, час створення та IP-адресу користувача.

4. Донати та мікроплатежі платформи на кшталт Twitch або Kick реалізують підтримку авторів через механізми пожертвувань (donations) і внутрішню валюту (Bits, Coins). У цьому випадку контент залишається безкоштовним, але глядачі можуть оплачувати інтерфейсні покращення .

Фінансова логіка таких транзакцій реалізується через інтеграцію з платіжними шлюзами (Stripe, PayPal, Amazon Pay) з подальшою фіксацією транзакції в окремій таблиці логів платежів.

5. Інтегрована реклама для безкоштовних моделей (YouTube, Dailymotion) дохід формується на основі перегляду рекламних блоків. Контроль відбувається через сервер вставки реклами (Ad Server), який синхронізується з відеоплеєром через VAST/VPAID протоколи.Після ініціалізації сесії користувача сервіс відслідковує параметри таргетингу: геолокацію, демографічні дані, тип пристрою. Контент відтворюється тільки після прогляду pre-roll або interstitial реклам.

1.1.3 Аналіз технологій збереження мультимедійних файлів

На даний момент системах збереження мультимедійного контенту виступає фундаментальним компонентом інфраструктури, що визначає стабільність роботи, масштабованість та якість доступу до сервісу. Під збереженням розуміється, що не лише фізичне розміщення файлів, а й уся сукупність механізмів обробки, кодування, доставки, кешування і оновлення вмісту.

Для ефективного зберігання мультимедійного контенту є використання контейнерних форматів, які дозволяють агрегувати декілька потоків у межах одного логічного об’єкта.

До найпоширеніших форматів належать MP4 (ISO/IEC 14496-12: ISO Base Media File Format), WebM та MOV.

Ці формати підтримують інкапсуляцію відео-, аудіоданих, субтитрів і службових метаданих (наприклад, timecode, language, track ID). З технічної точки зору, контейнери реалізують структуру атомів або блоків, де кожен блок відповідає за окремий тип потоку, що дозволяє ефективно обробляти дані на рівні стрімінгу та декодування.

Кодування відеопотоків виконується за допомогою спеціалізованих кодеків. Найпоширенішим є H.264/AVC (Advanced Video Coding), що відповідає стандарту ITU-T H.264 / MPEG-4 Part 10.

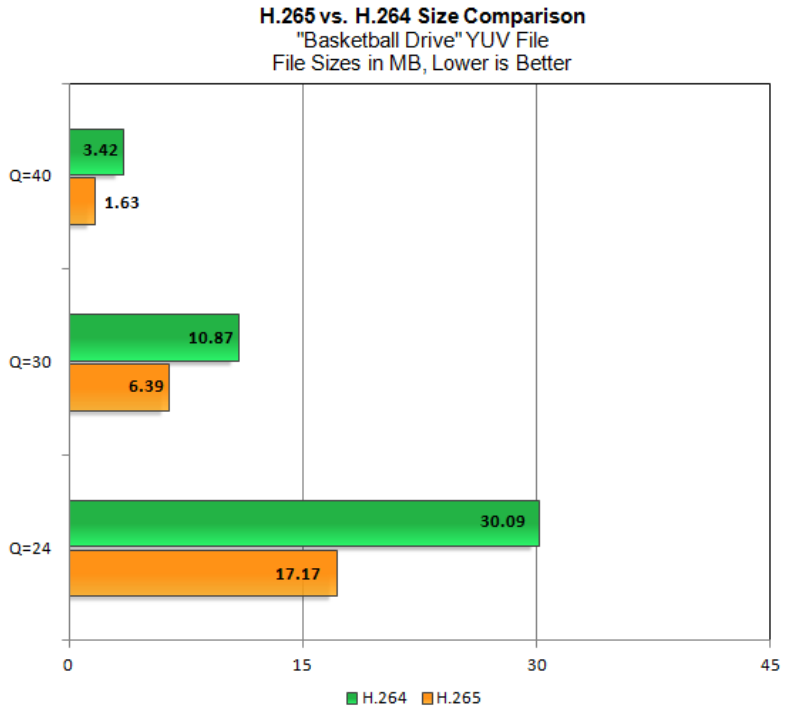


Рисунок 1.2 Порівняння ефективності технологій стиснення відео H.265 та H.264

Новітні кодеки, як-от H.265/HEVC та AV1, дозволяють зменшити бітрейт до 50% без втрати візуальної якості, але потребують потужніших ресурсів для декодування. Формат вибирається залежно від вимог до якості, ліцензування та сумісності з клієнтськими пристроями. При цьому AV1, як відкритий і безкоштовний стандарт, поступово набуває популярності серед платформ з високими вимогами до масштабованої доставки контенту.

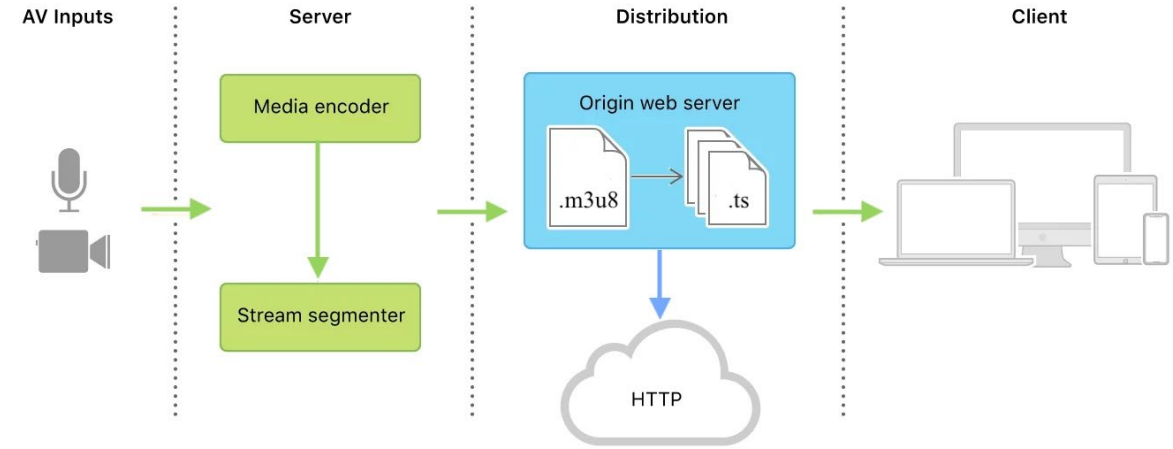


Рисунок 1.3 Архітектура потокової доставки HLS-протоколу

Як зображено на рисунку 1.3, потік мультимедіа спочатку кодується, потім розбивається на сегменти та розміщується на сервері розповсюдження, звідки клієнт завантажує необхідні сегменти через HTTP.

Слід відзначити, що всі зазначені протоколи використовують адаптивну передачу потоку (ABR), яка забезпечує стабільне та безперервне відтворення відео за змінної пропускної здатності мережі. Для визначення оптимального потоку застосовуються адаптаційні алгоритми, зокрема BOLA (Buffer Occupancy-based Lyapunov Algorithm), FESTIVE (Fair Efficient Streaming to Enhance Video Experience) та їхні гібридні варіанти.

Суть механізму полягає у наданні кількох версій відеопотоку з різною роздільною здатністю та бітрейтом, з можливістю динамічного перемикання між ними відповідно до поточних умов на стороні клієнта (рисунок. 1.4).

Таблиця 1.2. Порівняльна характеристика протоколів доставки мультимедійного контенту

Протокол	*Тип передавання*	*Затримка*	*Адаптивність*	*Тип трафіку*
HLS	Сегментований через HTTP	Середня	Підтримується	Відео на запит
MPEG-DASH	Сегментований через HTTP	Середня	Підтримується	Відео на запит
WebRTC	Peer-to-peer (UDP/DTLS)	Низька	Обмежена	Реальний час (чат, стрім)

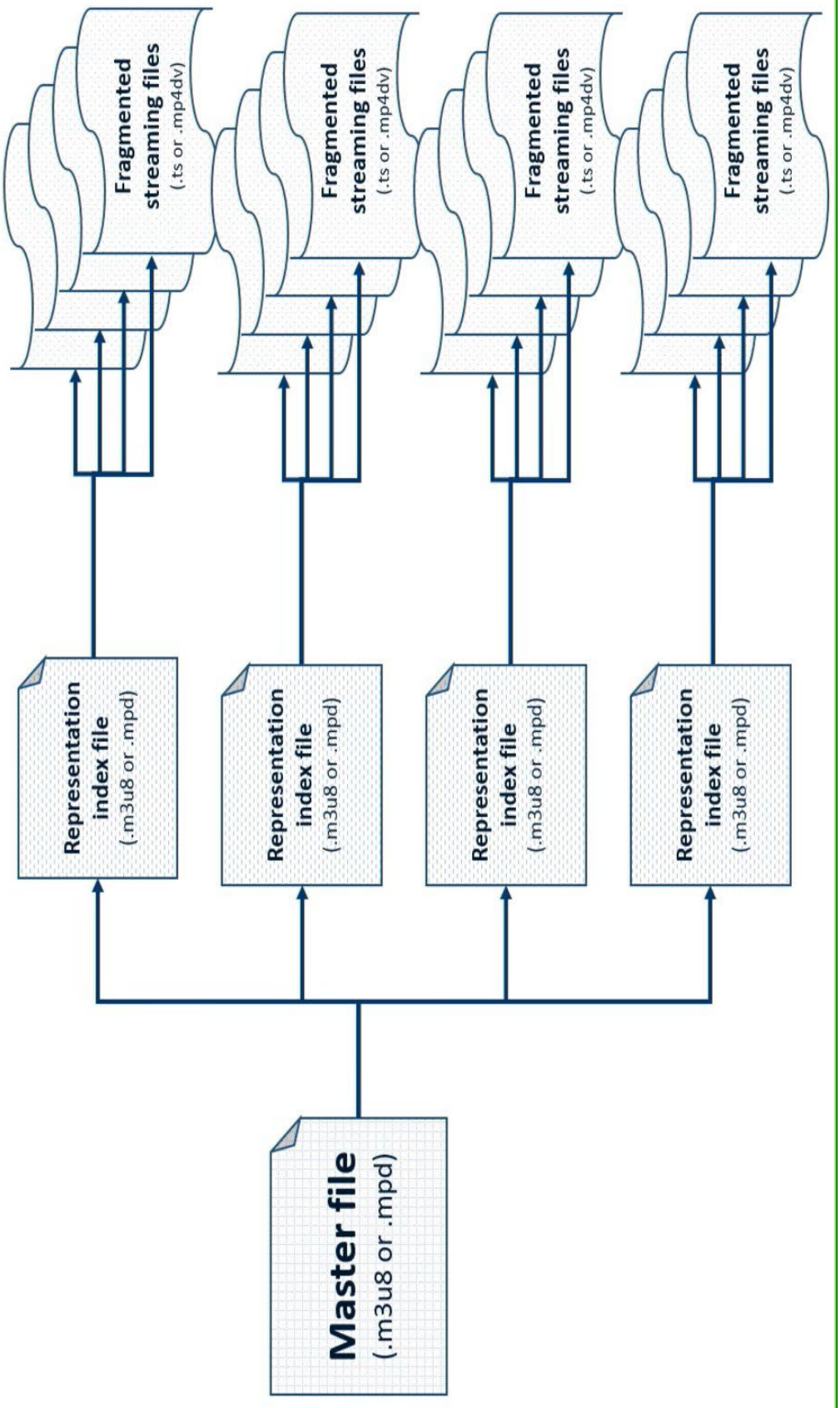


Рис. 1.4 Принцип адаптивної передачі мультимедіа з ABR.

Рисунок 1.5 Формат IP-заголовку за стандартом 802.1p

У хмарних архітектурах зберігання мультимедійних даних реалізується переважно через об’єктно-орієнтовані сховища, які характеризуються горизонтальною масштабованістю, високою доступністю та інтеграцією з механізмами автентифікації й авторизації. Основні хмарні платформи, які надають сервіси зберігання мультимедійного контенту, включають Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Згідно з архітектурною схемою на рисунку 1.4, кожен із провайдерів пропонує декілька спеціалізованих сервісів:

- AWS — S3 для основного зберігання об’єктів, Glacier для архівування, Storage Gateway для гібридного доступу та Snowball для офлайн-міграції великих обсягів даних.
- Azure — Blob Storage як основне сховище, а також рішення для резервного копіювання (Backup), масштабованого зберігання (StorSimple) та фізичної передачі даних (Azure Data Box).
- GCP — надає сервіси через Cloud Storage з інтеграцією з Compute Engine та BigQuery.

Зберігання мультимедійних об’єктів реалізується через контейнерні формати .mp4, .webm, .mov, які інкапсулюють відео, аудіо, субтитри й метадані

Зокрема, одним із поширених варіантів є MySQL, який дозволяє створювати структуровану модель таблиць з індексами по полях file_id, codec, duration, bitrate, resolution, upload_date, access_scope, що дає змогу виконувати SQL-запити з високою продуктивністю. У разі потреби гнучкої схеми збереження, додаткові параметри (наприклад, аудиторські мітки або технічні характеристики потоку) можуть зберігатися у вигляді JSON-структур у відповідних полях (за допомогою типу JSON у MySQL 5.7+).

1.1.4 Аналіз інтеграції платіжних систем

Платіжна система є критичним компонентом будь-якої платформи, що реалізує модель доступу до контенту за підпискою або через разові платежі. Надійна інтеграція з платіжними шлюзами забезпечує безперервність обслуговування, захист транзакцій, гнучкість білінгу, підтримку глобальних методів оплати, відповідність регуляторним нормам (PCI DSS, PSD2, SCA) та масштабованість. Розгортання платіжної інфраструктури в сучасних мультимедійних системах здійснюється через стандартизовані API-інтерфейси та Webhook-механізми для асинхронної обробки подій, що дозволяє оперативно реагувати на зміни статусів платежів і адаптуватися до навантажень.

Для реалізації автоматичного поновлення підписок використовується білінг-модуль, який дозволяє планувати повторювані списання з гнучкою частотою (щомісяця, щороку, по події). Дані карток не зберігаються на стороні платформи, а токенізуються провайдером, що дозволяє обійти складну сертифікацію PCI DSS Level 1.

Платіжна інфраструктура повинна відповідати стандартам PCI DSS (Payment Card Industry Data Security Standard). Це передбачає:

- TLS 1.2+ для передачі даних;
- шифрування карткових даних (AES-256 або RSA 2048);
- ротацію ключів доступу до API;
- логи доступу та аудит;
- реалізацію механізмів запобігання шахрайству (anti-fraud).

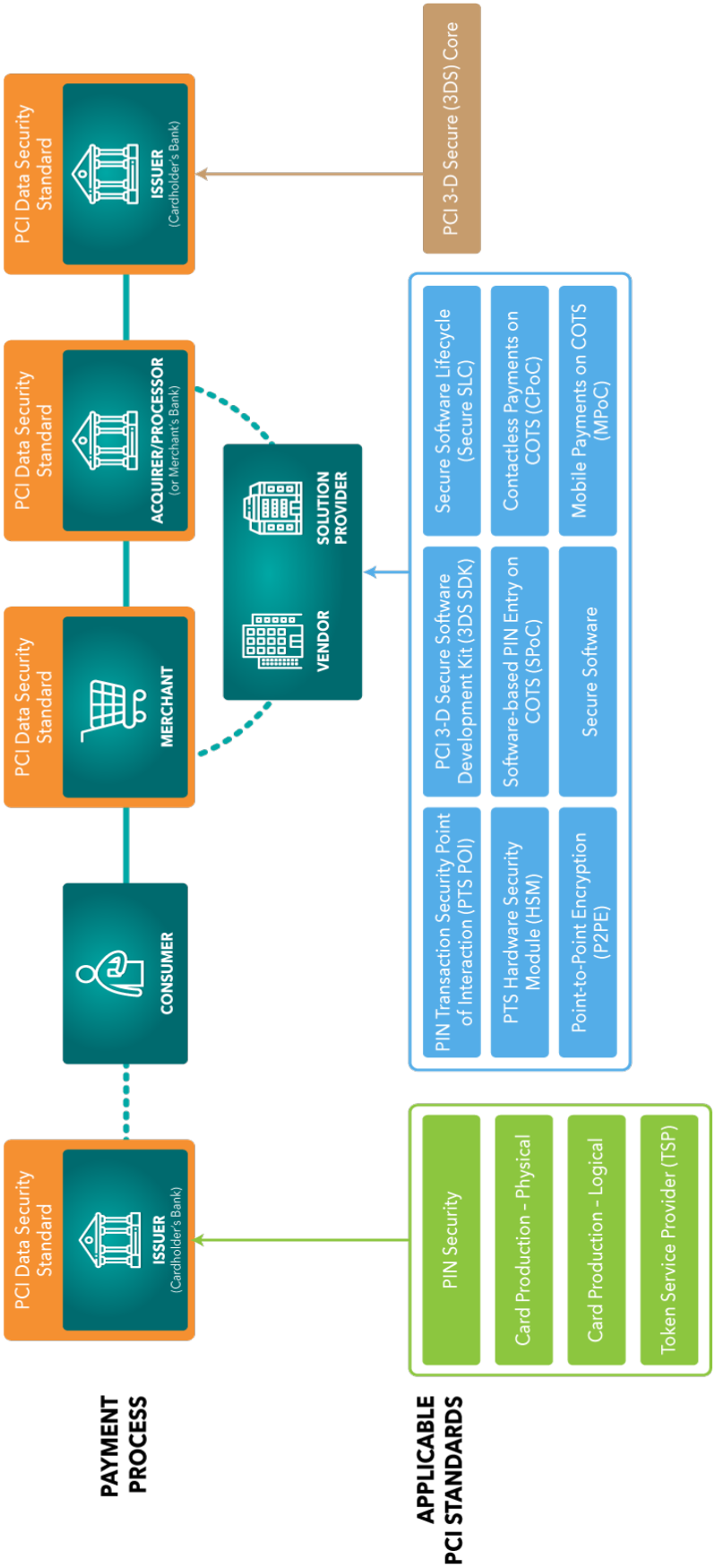


Рисунок 1.6 Структура стандартів безпеки платіжних систем PCI DSS та 3DS

Усі ключові учасники платіжного процесу — користувач (споживач), торговець (merchant), банк-еквайр, банк-емітент, постачальники рішень та інтегратори — зобов'язані дотримуватись відповідних стандартів PCI. Серед них варто виділити PCI DSS (для безпеки зберігання та обробки карткових даних), PCI 3DS SDK (для 3D Secure авторизації), а також специфікації для програмного забезпечення, апаратного забезпечення та контактних/безконтактних платіжних каналів. Ця система дозволяє підтримувати цілісність транзакцій, захищати користувача від шахрайських операцій та забезпечувати надійну обробку на стороні продавця (рисунок. 1.6).

Технології Apple Pay і Google Pay є основою сучасної інфраструктури цифрових платежів для мобільних і веб-застосунків, забезпечуючи нативну інтеграцію з iOS/macOS та Android/Chrome відповідно, що дозволяє впроваджувати підписну модель без зберігання чутливих даних на сервері.

Основою Apple Pay та Google Pay є EMVCo Tokenization Framework, який замінює реальні карткові реквізити токенами з обмеженим терміном дії, прив'язаними до пристрою чи сесії. Кожна транзакція супроводжується одноразовим криптограмним підписом (Dynamic Cryptogram), що гарантує цілісність і автентичність операції, унеможливлюючи повторне використання перехоплених даних зловмисником. Додатковий рівень безпеки забезпечується асиметричною криптографією, де закритий ключ зберігається в захищеному елементі пристрою (SE або TEE). Apple Pay реалізовано через PassKit API для iOS/macOS із підтримкою WKWebView для веб-інтеграцій, де аутентифікація виконується через Face ID, Touch ID або пароль, після чого пристрій генерує підпис, який надсилається шлюзу платіжної системи, а серверна інтеграція обмежується роботою з сертифікатами Apple Pay та endpoint-ами (наприклад, Stripe чи Braintree). Google Pay SDK взаємодіє через Google Payment API (GPA), підтримуючи інтентний метод на Android та Payment Request API для браузерів. Такий підхід сприяє підвищенню довіри користувачів завдяки надійному захисту їхніх фінансових даних.

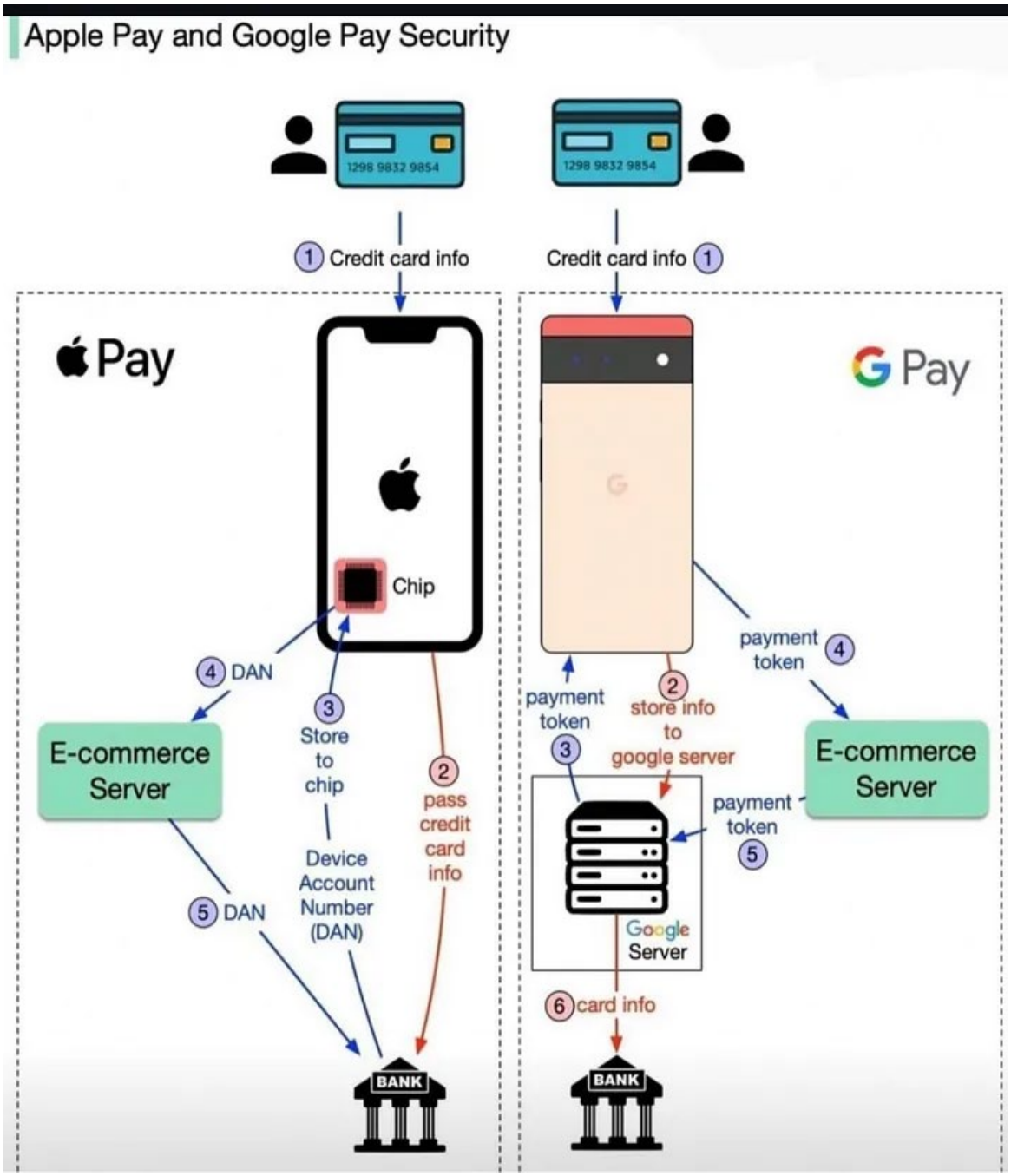


Рисунок 1.7 Порівняльна схема обробки транзакцій Apple Pay та Google Pay

У Google Pay обробка транзакції базується на двох основних компонентах: сторонньому сервері (merchant backend) і Google Server, який виконує функцію Trusted Execution Environment (TEE) при генерації токенів. Передача платіжного токена здійснюється через Google Payment API після локального шифрування запиту.Токен містить Payment Data, зокрема encryptedMessage, ephemeralPublicKey і tag, які декодуються платіжним шлюзом. Сам же процес верифікації супроводжується відправленням на backend Google відповідає

сформованим токеном. Як зображено на (рисунку 1.7), токенізація виконується віддалено, у той час як Apple Pay здійснює її безпосередньо на пристрої за допомогою Secure Element (SE).

Важливим аспектом є те, що в Apple Pay відсутня потреба в передачі або обробці повних реквізитів карти: замість них на пристрої створюється DAN (Device Account Number), який зберігається на апаратному рівні та захищений від несанкціонованого доступу. Кожен запит на оплату супроводжується динамічним криптографічним підписом, який генерується за допомогою приватного ключа, що ніколи не покидає SE. Google Pay, зі свого боку, забезпечує подібний рівень безпеки, однак реалізує логіку токенізації на стороні Google Server, що дозволяє виконувати розподілену верифікацію та масштабну підтримку партнерських банків через Cloud Tokenization Framework.

Для підтвердження транзакцій Apple Pay використовує протокол EMV Payment Tokenisation Specification, реалізований у співпраці з EMVCo, що включає в себе криптографічну автентифікацію з HSM-модулями платіжних процесорів. Google Pay підтримує аналогічну модель, але з опорою на cloud-based token vault, де токени зберігаються в обліковому записі Google Payments. Обидві технології відповідають вимогам PCI DSS Level 1 та підтримують SCA (Strong Customer Authentication) згідно PSD2.

На стороні розробника інтеграція Google Pay реалізується шляхом ініціалізації PaymentsClient у Android або JavaScript API в браузерах, де при створенні запиту вказується merchant ID, public key та параметри платіжної сесії.

Платформа повертає об'єкт PaymentData для подальшої верифікації токена через бекенд-шлюз. Apple Pay використовує аналогічну архітектуру з PKPaymentRequest та PKPaymentAuthorizationViewController, що генерують encrypted payment object із мережевим токеном та SE-криптограмою. Обидві системи забезпечують PCI DSS compliance через використання токенізації та енд-ту-енд шифрування**.** Ключова відмінність полягає в кросплатформенності Google Pay (фізичні пристрої/хмара) проти екосистемної обмеженості Apple Pay.

У контексті безпеки обидві платформи відповідають вимогам PCI DSS Level 1 і EMVCo Tokenization Framework. Дані карток заміщуються токенами, кожен із яких має обмежений термін дії та прив'язку до пристрою. У Apple криптографічні ключі зберігаються в Secure Enclave — мікропроцесорі з ізольованим середовищем виконання, що забезпечує hardware-level захист.

Таблиця 1.3. Класифікація бітів пріоритету

Параметр	Stripe/Braintree	Apple Pay	Google Pay
Тип інтеграції	API + Webhook	PassKit SDK	Google Payment API
Токенізація	На стороні шлюзу	Secure Element (локально)	Google Server (хмарно)
Зберігання картки	Не зберігаються (tokenized via PCIprovider),	Дані не передаються серверу — лише одноразовий криптограмний підпис	Карткові дані зберігаються в Google Vault, передаються через токен із обмеженим TTL
Аутентифікація	3D Secure 2.0, OTP, біометрія (залежно від банку)	Face ID, Touch ID або PIN перед підписом транзакції	з прив'язкою до пристрою, авторизація через PIN або біометрію
Платформи	Кросплатформенно (iOS, Android, Web), через SDK та REST API	Обмежено лише iOS / macOS із Safari	Android 5.0+, Google Chrome WebView з підтримкою GPA
Підтримка підписок	Повна підтримка recurring billing, інтервали, retry-логіка, Webhook для поновлення	Підписка через merchant backend, одноразова генерація payment object	Підтримка підписок через GPA із callback-логікою на стороні продавця

Загальна архітектура платіжної взаємодії в межах цифрових платформ дедалі більше тяжіє до безсерверної моделі авторизації, де обробка чутливих даних делегується спеціалізованим сервісам на основі токенізації та нативних SDK. Використання стандартів PCI DSS, EMVCo Tokenization, TLS 1.2+, P2PE та захищених API шлюзів мінімізує ризики компрометації, дозволяючи платформі зосередитися на бізнес-логіці білінгу, а не на збереженні чи обробці платіжної інформації. Такий підхід є технічно виправданим і водночас відповідним до регуляторних вимог, що дозволяє системам підписок функціонувати з високим рівнем надійності та безпеки навіть за значного зростання навантаження.

1.2 Розробка алгоритмів керування доступом до контенту

Система керування доступом до цифрового контенту є основоположним компонентом платформ із монетизованим розповсюдженням матеріалів. Основна мета — забезпечити контрольоване, регульоване та динамічно адаптоване надання доступу до ресурсів відповідно до рівня підписки, ролі користувача або історії транзакцій.

1.2.1 Розробка структури бази даних для користувачів, відеоматеріалів та підписок

Розробка структури бази даних для користувацьких мультимедійних платформ є критичним етапом при реалізації системи керування доступом до контенту. Основною вимогою до архітектури такої бази є забезпечення узгодженості даних, підтримка гнучкого механізму підписок, а також ефективне зберігання та пошук відеоматеріалів. На рисунку 1.8 зображено загальну Er-діаграму, що ілюструє логічні зв’язки між ключовими сутностями.

Таблиця «Subscription» (Підписка) містить визначення тарифних планів, кожен з яких має унікальні характеристики: `id`, `name`, `price`, `description`, `allowed_roles`. Ці параметри дозволяють розмежовувати доступ до певних категорій відео відповідно до рівня підписки. Взаємозв’язок між користувачем і підпискою реалізовано через таблицю `UserSubscription`, що фіксує активні підписки, їх тривалість (`start_date`, `end_date`) та статус (`status`).

Сутність «User» (Користувач) є центральною у системі, оскільки вона зберігає облікові дані, особисту інформацію, тип ролі та статус активності. Основні атрибути: `id`, `username`, `email`, `password`, `role`, `balance`. Роль користувача визначає доступ до адміністративного функціоналу, тоді як баланс і статус підписки — доступ до платного контенту.Сутність «Video» (Відеоматеріал) включає поля `id`, `title`, `description`, `videoUrl`, `previewUrl`, `uploadDate`, `requiredLevel`, що дозволяють не лише каталогізувати контент, а й реалізувати обмеження на основі рівня підписки.

Наприклад, контент із рівнем доступу `Powerful SEO` не буде доступний для підписників нижчих тарифів. Також передбачено поле `hashtags` для семантичного пошуку.

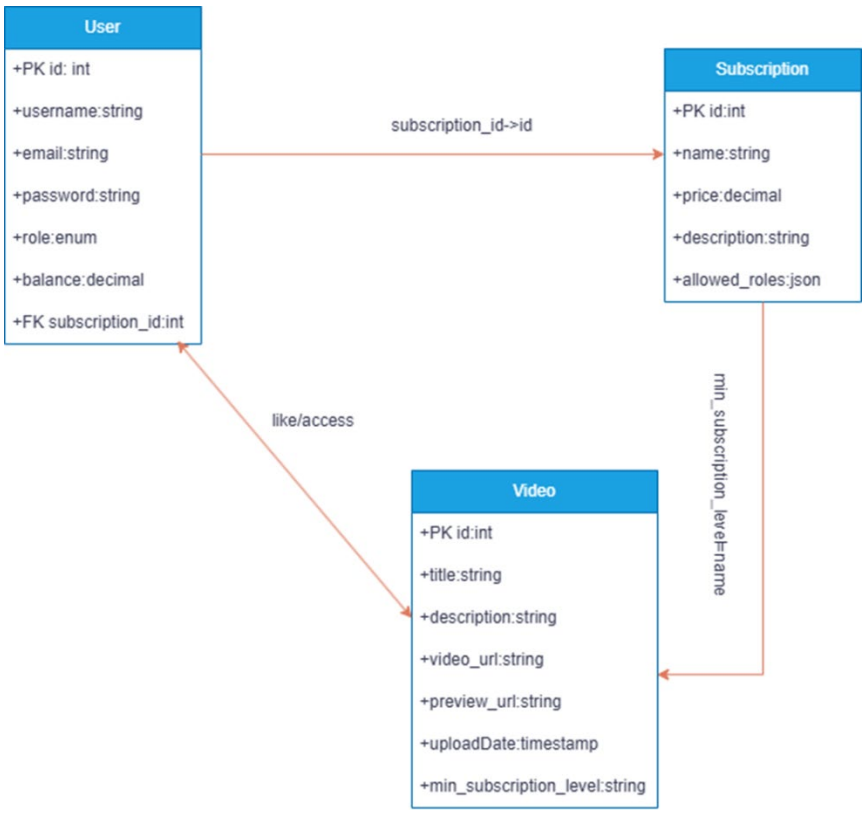


Рисунок 1.8 ER-діаграма бази даних користувачів, відео та підписок

У системі реалізовано зв'язок між користувачем і відео через таблицю `WatchProgress`, що фіксує індивідуальний прогрес перегляду: тривалість перегляду, відсоток завершення, дата останнього перегляду. Це дозволяє реалізувати функціонал продовження перегляду з останнього моменту.

База даних реалізується у MySQL, з використанням типів `JSON`, `ENUM`, індексації по полях `video_id`, `user_id`, `subscription_id` для забезпечення оптимального доступу при великій кількості запитів.Обов'язковим є зовнішнє обмеження цілісності даних (foreign keys), а також каскадне оновлення/видалення (`ON DELETE CASCADE`) для пов'язаних записів.

Додатково до базової діаграми варто врахувати механізм денормалізованих агрегатів, аби мінімізувати «дорогі» JOIN-операції у найчастіших сценаріях читання.

Зокрема, у таблиці video доцільно зберігати comments_count, likes_count та views_count, що оновлюються тригерами після вставки або видалення записів у відповідних таблицях comment та «журналі переглядів». Такий підхід скорочує час вибірки при відображенні списків роликів і одночасно не порушує вимоги 3-ї нормальної форми завдяки тому, що «правдивим» джерелом лишаються дочірні таблиці, а у тригерах реалізовано перевірку цілісності. Для масових операцій (наприклад, міграцій або повторного обчислення метрик) передбачено матеріалізовані представлення із розкладом оновлення через EVENT SCHEDULER, що дозволяє ізолювати аналітичне навантаження від OLTP-шарів.

Таким чином, поєднання строгих зовнішніх ключів, каскадної обробки подій та багато-рівневої стратегії резервування забезпечує цілісність, доступність і масштабованість даних навіть у пікових режимах експлуатації.

1.2.2 Алгоритм контролю доступу до відео залежно від рівня підписки

Контроль доступу до відеоматеріалів у системі реалізується через поєднання ролевої авторизації та перевірки рівня підписки користувача. Основною метою є обмеження доступу до контенту відповідно до умов, що встановлені автором або адміністрацією платформи. Кожен відеозапис має атрибут min_subscription_level, який визначає мінімальний рівень підписки, необхідний для перегляду. Цей атрибут зберігається у вигляді значення з обмеженого переліку (тип ENUM): Free, Junior, Chilli Middle, Powerful SEO.

При кожній спробі доступу до відео користувач ідентифікується через JWT-токен або сесію, після чого виконується запит до бази даних для отримання поточного статусу підписки (subscription_name) з таблиці CustomUser. Порівняння рівнів здійснюється за заздалегідь визначеною ієрархією, де Free має найнижчий пріоритет, а Powerful SEO — найвищий.

Якщо рівень користувача нижчий, ніж вимагає відео, сервер повертає HTTP-відповідь з кодом 403 Forbidden та повідомленням про обмеження доступу. У структурі програми реалізовано проміжний шар (middleware), який перехоплює запити до кінцевих точок відео-контенту.

Цей модуль містить функцію перевірки рівня підписки, яка приймає ID користувача та ID відео як параметри.

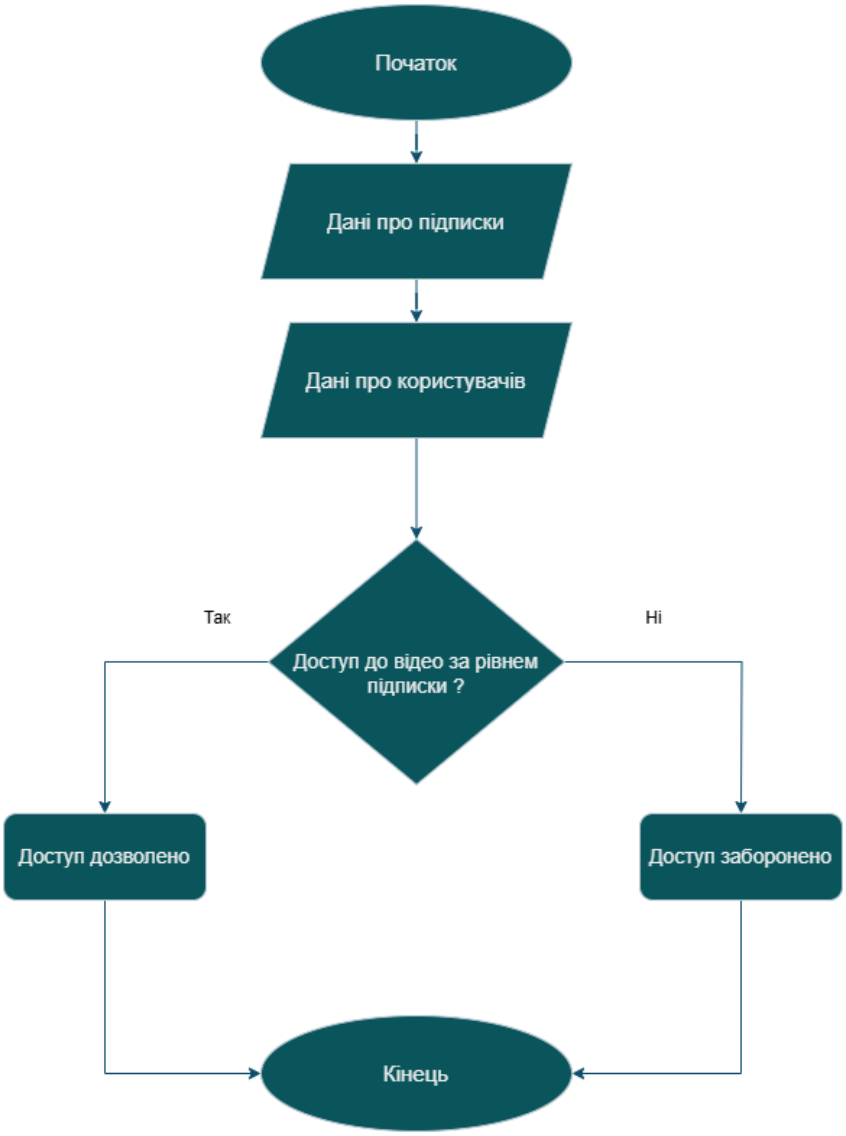


Рисунок 1.9 Блок-схема перевірки доступу до відеоконтенту за підпискою

1.2.3 Реалізація алгоритму збереження прогресу перегляду відео

У системах дистрибуції відеоконтенту одним з ключових функціональних компонентів є збереження прогресу перегляду для кожного користувача. Це дозволяє не лише забезпечити зручність користування платформою, а й реалізувати функції відновлення перегляду з місця зупинки, персоналізованих рекомендацій та аналітики поведінки користувачів.

Алгоритм збереження прогресу перегляду має враховувати технічні аспекти відправки, зберігання і синхронізації даних.

На клієнтській стороні (браузер або мобільний додаток) реалізується механізм трекінгу перегляду у реальному часі.

Кожні N секунд (наприклад, 5 або 10) фронтенд-додаток відправляє POST-запит до API з інформацією про позицію плеєра (в секундах), ідентифікатор відео та ID користувача. У разі завершення перегляду або закриття вкладки спрацьовує подія onbeforeunload, яка ініціює фінальну синхронізацію прогресу перегляду.

На серверному боці Django API отримує ці запити та зберігає відповідні дані у таблиці прогресу (наприклад, VideoProgress). Основними полями цієї таблиці є: user_id (foreign key), video_id (foreign key), last_position (float/int), updated_at (datetime). Для уникнення дублювання кожна нова позиція замінює попереднє значення для відповідної пари user_id + video_id за допомогою унікального індексу або операції update_or_create() у Django ORM.

Після збереження позиції перегляду на сервері система повинна забезпечити її коректне відображення при наступному зверненні користувача до відеоматеріалу. Під час завантаження сторінки відео або ініціалізації плеєра фронтенд виконує GET-запит до API, передаючи параметри user_id і video_id. У відповідь сервер повертає останню зафіксовану позицію перегляду, яка використовується як стартова точка (player.seekTo(position)) у плеєрі. Таким чином, користувач автоматично повертається до моменту, на якому було завершено попередній сеанс перегляду. Для запобігання маніпуляціям або помилкам важливо перевіряти коректність позиції на клієнтській стороні (наприклад, вона не повинна перевищувати тривалість відео).Блок-схема алгоритму моніторингу прогресу перегляду відео рисунок. 1.10.

Інтеграція з Cloudinary як основного сховища відеофайлів дозволяє зменшити навантаження на серверну частину та оптимізувати доставку контенту. Cloudinary забезпечує потокову передачу відео через CDN (Content Delivery Network), автоматичне масштабування та конвертацію форматів.

Завдяки використанню Cloudinary public_id для кожного відеофайлу, можна гарантовано зв’язувати файли у хмарі з відповідними об’єктами у базі даних.

Крім того, Cloudinary API дозволяє динамічно витягувати мета-дані про відео, зокрема загальну тривалість, що може бути використано для валідації збереженого прогресу та обчислення відсотка перегляду.

Таким чином, платформа може будувати графік завершених переглядів або виявляти недивлені відео в особистому кабінеті користувача.

Рисунок 1.10 Блок-схема алгоритму моніторингу прогресу перегляду відео

1.2.4 Опис алгоритму роботи адміністративної панелі керування користувачами та підписками

Адміністративна панель керування користувачами та підписками є ключовим функціональним компонентом системи, який дозволяє адміністраторам здійснювати контроль над обліковими записами користувачів, їхніми ролями, активністю та статусами підписки. Основу алгоритму складає концепція CRUD-операцій (Create, Read, Update, Delete), яка реалізується через авторизований інтерфейс з обмеженим доступом відповідно до ролі користувача в системі. При вході до панелі адміністратор проходить перевірку на відповідність ролі admin, після чого отримує доступ до функціональних розділів: «Користувачі», «Підписки», «Статистика», «Логи дій»..

Основна частина алгоритму полягає в реалізації двосторонньої інтеграції між фронтендом адміністративної панелі та бекенд-сервером через RESTful API або GraphQL, з використанням JWT або session-based авторизації. Коли адміністратор відкриває розділ «Користувачі», відправляється GET-запит до відповідного endpoint'у, який повертає список зареєстрованих облікових записів з полями email, display_name, country, balance, subscription_status, role. Додатково може включатися пагінація, фільтрація за статусом підписки або країною, та сортування. У разі вибору конкретного користувача відбувається запит до /api/users/<id>, що надає повну інформацію про профіль, історію змін та прив’язані підписки. Адміністратор має змогу змінювати роль (user ↔ admin), редагувати контактну інформацію, деактивувати обліковий запис, а також вручну змінювати статус підписки (наприклад, активувати преміум-доступ без оплати).

Оновлення підписки здійснюється через відповідний PUT або PATCH-запит, який змінює поля subscription_name, subscription_status, subscription_price, subscription_description. Для безпеки кожна така операція логуються у таблиці audit_logs, де фіксується ідентифікатор адміністратора, час зміни, тип операції та значення до/після. Для нових користувачів адміністратор може створити обліковий запис вручну, ініціюючи POST-запит до /api/users/, де вказуються

основні атрибути, роль, стартовий баланс та підписка. У межах системи передбачено механізм валідації даних (наприклад, перевірка унікальності email, відповідності типів, правильності JSON-структур) на стороні сервера,показані на блок-схемі «Алгоритм роботи адміністративної панелі керування користувачами та підписками» (рисунок. 1.11).

Особливу увагу в алгоритмі приділено контролю за платіжним статусом підписок. Після оновлення статусу користувача адміністратор може вручну ініціювати запит до Stripe або іншого білінг-сервісу для синхронізації змін. Це реалізується через окрему API-функцію, яка перевіряє актуальний стан платежу, оновлює локальну інформацію в полях stripe_customer_id, subscription_price, subscription_status та реєструє відповідну подію у subscription_history.

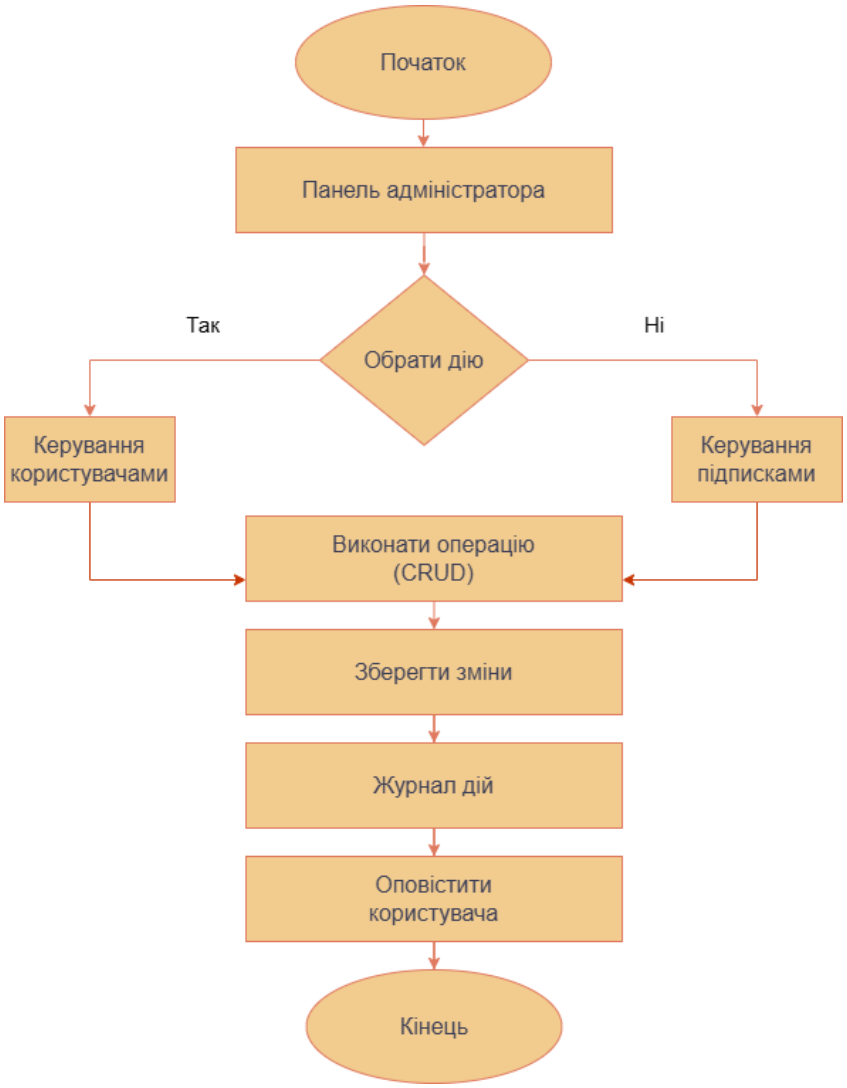


Рисунок 1.11. Алгоритм роботи панелі керування користувачами та підписками

Особливу увагу в алгоритмі приділено контролю за платіжним статусом підписок. Після оновлення статусу користувача адміністратор може вручну ініціювати запит до Stripe або іншого білінг-сервісу для синхронізації змін. Це реалізується через окрему API-функцію, яка перевіряє актуальний стан платежу, оновлює локальну інформацію в полях stripe_customer_id, subscription_price, subscription_status та реєструє відповідну подію у subscription_history. У разі помилки або невідповідності платіжної інформації система повертає помилку, а адміністратор отримує сповіщення з пропозицією перевірити операцію вручну. Такий підхід дозволяє ефективно відслідковувати розбіжності між фактичними платежами та внутрішніми статусами підписок.

Для оптимізації перегляду та адміністрування передбачено візуальні індикатори активності: наприклад, кольорове маркування активних/неактивних користувачів, статусу підписки (безкоштовна, Junior, Chilli Middle, Powerful SEO), автоматичне згортання неактивних записів. Адміністратор може групувати користувачів за роллю, країною, підпискою або датою останньої активності. Також доступна експортна функція у форматах CSV або JSON для створення звітів про активність, популярність підписок, баланси та зміну кількості користувачів у часі.

1.3 Програмна реалізація веб-платформи

Система побудована за логікою «тонкого клієнта + REST / GraphQL API» із суворим розмежуванням шару представлення та бізнес-логіки. Бекенд виконано у вигляді service-oriented-monolith на Django 4 + Django REST Framework (DRF). Фронтенд реалізовано як односторінковий застосунок (React 18, TypeScript, MUI v5).

Первинне персистентне сховище — MySQL 8, оптимізоване індексами за полями user_id, video_id, subscription_id, а також покритими індексами для комбінованих запитів аналітики (дата + рівень підписки). Для управління міграціями використано Alembic-style патерн на базі django-migrations із перевіркою цілісності зовнішніх ключів (foreign keys) у CI-конвеєрі.

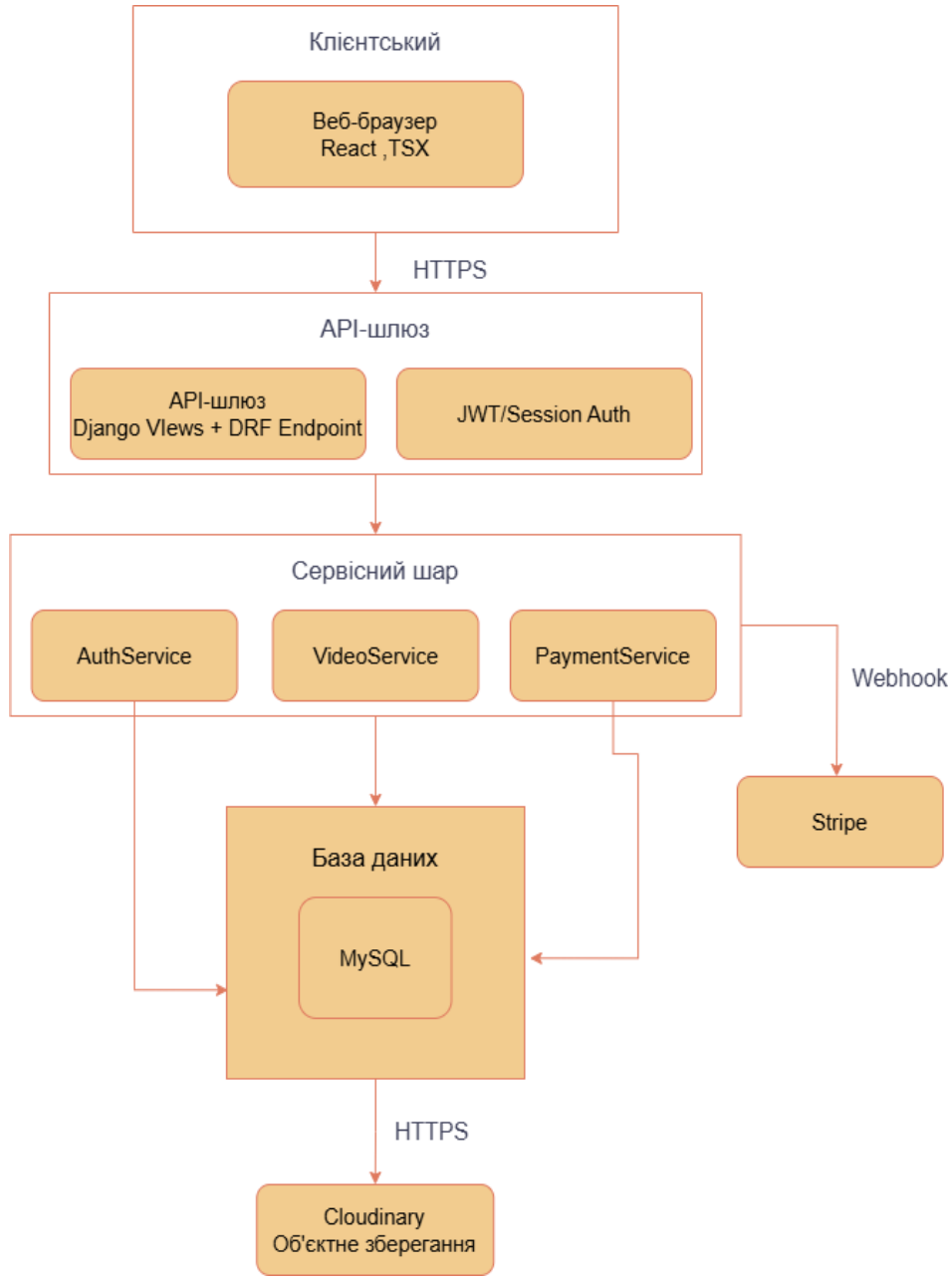


Рисунок 1.12 Структурна схема архітектури веб-платформи

На верхньому рівні (рисунок. 1.12) розташовано веб-браузер, у якому виконується клієнтська логіка, реалізована на React + TypeScript (файли .tsx). Компоненти інтерфейсу відповідають за формування маршрутизації (React Router), керування станом та відтворення медіа (кастомний VideoPlayer). Жодна бізнес-логіка, пов’язана зі зберіганням або платежами, не виконується на клієнті; натомість браузер здійснює лише авторизовані HTTP/HTTPS-запити до API-шлюзу, передаючи JWT-токен або cookie-сесію. Визначення вимог до програмного рішення;

Наступний модуль — API-шлюз на базі Django Views та Django REST Framework. Його завдання:

- уніфікувати точки входу (/api/...),
- виконати попередню аутентифікацію (JWT або Session Auth),
- застосувати крос-запитну валідацію й нормалізувати помилки.

Таким чином, UI ніколи не звертається безпосередньо до сервісного шару, що спрощує введення rate-limiting, CORS-політик та розподіленого логування.

Сервіси мають тонкі REST-контролери й товсті доменні модулі: усі бізнес-правила (перевірка рівня підписки, ліміти на завантаження, розрахунок комісій) розміщено всередині сервісного рівня, зберігаючи контролери «пасивними».Відео-транскодування, відправлення е-майлів та обробка Stripe-веб-хуків виконуються у фонових робітниках Celery, що під’єднані до RabbitMQ. Це дозволяє не блокувати відповідь для клієнта і масштабувати робітники горизонтально.

MySQL — реляційне ядро зі схемою по нормальній формі 3NF.

- Індекси по video_id, user_id, subscription_id мінімізують latency SELECT-запитів при рекомендаціях та побудові стрічки.
- У таблицях video і comment застосовано JSON-поля для напівструктурованих метаданих (timecodes, materials).

Cloudinary Object Storage використовується як зовнішній бакет для оригіналів і трансформованих копій медіа. Записи про URL-адреси зберігаються в MySQL, тому транзакція додавання відео охоплює як базу, так і Cloudinary API за допомогою двофазного commit.

Stripe — єдиний платіжний шлюз систему, під’єднаний до PaymentService.

1.3.1 Вибір і обґрунтування технологічного стек

Проєктована веб-платформа створювалася як багатофункціональна система, що об'єднує роботу з мультимедійним контентом (відео, зображення, аудіо), реалізацію підписної моделі для надання доступу до преміум-функцій, керування ролями користувачів (адміністратор, звичайний користувач, підписник), обробку онлайн-платежів для інтеграції фінансових операцій та масштабоване зберігання

даних для підтримки зростання кількості користувачів і контенту.

Відповідно, до вибору технологічного стеку висувалися суворі вимоги щодо надійності (запобігання збоїв при високому навантаженні), безпеки (захист даних від витоків і несанкціонованого доступу), продуктивності (швидке виконання запитів і обробка великих обсягів даних) та розширюваності (можливість додавання нових функцій без переписування коду). Ці вимоги були визначені на етапі планування для забезпечення довготривалого розвитку платформи та її адаптації до змін у потребах користувачів.

У ролі серверного середовища обрано фреймворк Django, що завдяки своїй модульній архітектурі та вбудованій ORM (Object-Relational Mapping) дозволяє швидко проєктувати ієрархічні моделі даних (наприклад, User, Video, Subscription), реалізовувати контролери доступу для обмеження дій залежно від ролі, маршрути API для обробки запитів, механізми авторизації (наприклад, перевірка токенів), обробку запитів (GET, POST, PATCH, DELETE) і підключення сторонніх сервісів (Stripe, Cloudinary). Цей фреймворк забезпечує вбудовану підтримку middleware для обробки заголовків і валідації даних, що спрощує захист від типових атак, таких як SQL-ін’єкції. Для створення RESTful-інтерфейсів використовується Django REST Framework (DRF), що забезпечує повну підтримку CRUD-операцій (створення, читання, оновлення, видалення), фільтрації даних за параметрами (наприклад, за статусом підписки), пагінації для зручного перегляду великих списків, серіалізації об’єктів у JSON-формат для передачі клієнту та обробки permission-класів для тонкого налаштування доступу. Крім того, DRF надає простий інструментарій для впровадження токен-автентифікації через JWT (JSON Web Token), що забезпечує безпечну й масштабовану авторизацію між клієнтською та серверною частинами, включаючи захист від підробки токенів і підтримку оновлення через refresh-токени.

В якості системи управління базами даних застосовано MySQL, яка підтримує реляційну структуру з можливістю використання транзакцій для забезпечення атомарності операцій (наприклад, оновлення статусу підписки і

списання коштів), складних SELECT-запитів для аналітики (наприклад, вибірка активних користувачів за період), зовнішніх ключів для підтримки цілісності даних між таблицями (наприклад, зв’язок User і Subscription) та індексації для прискорення пошуку. В моделі передбачене використання JSONField для гнучкого зберігання метаданих — таких як часові мітки відео (timecodes для навігації), супровідні матеріали (наприклад, PDF-документи чи текстові примітки) та індивідуальні налаштування користувача (наприклад, обрана тема інтерфейсу). Для забезпечення продуктивності виконання запитів у таблицях встановлено індекси за ключовими полями (user_id для ідентифікації користувача, video_id для зв’язку з контентом, subscription_status для фільтрації за статусом), що оптимізує роботу фільтрів під час генерації персоніфікованої стрічки, зменшуючи час відповіді сервера до десятків мілісекунд навіть при великій кількості записів.

Інтеграція з Cloudinary також включає підтримку вебхуків для сповіщення про завершення завантаження файлів, що дозволяє оновлювати статус контенту в базі даних у реальному часі. Для забезпечення безпеки передачі медіа налаштовано шифрування URL і обмеження доступу через підписані посилання (signed URLs), які дійсні лише протягом заданого часу. Це доповнює механізм авторизації Django, створюючи подвійний рівень захисту. Оптимізація продуктивності додатково включає кешування результатів API-запитів через Redis, що зменшує час відповіді при повторних запитах до популярних ресурсів, і конфігурацію балансувальника навантаження для розподілу трафіку між серверами в разі масштабування. Такий підхід гарантує, що платформа залишиться стабільною навіть при пікових навантаженнях, таких як масовий перегляд контенту під час запуску нової кампанії.

Фронтенд реалізований за допомогою React з використанням с спростити дебагінг интаксису TypeScript (.tsx), що дозволяє реалізувати типізовані інтерфейси, спростити дебагінг та підвищити надійність коду. Для керування станом використовується Redux Toolkit, для маршрутизації — React Router. Інтерфейс підтримує lazy-loading компонентів, умовний рендеринг залежно від

ролі користувача та динамічне оновлення контенту через REST-запити до бекенду.Для обробки платежів інтегровано Stripe API, що дозволяє реалізовувати підписки, автоматичне поновлення (recurring billing), вебхуки та створення інвойсів.Сервер обробляє події типу checkout.session.completed, customer.subscription.updated та оновлює відповідні записи в базі.

Для зберігання прогресу перегляду використовується стандартна таблиця watch_progress, яка оновлюється через PUT-запити з client-side кожні 10 секунд або при виході з відеоплеєра.

1.3.2 Створення об’єктно-орієнтованої моделі застосунку

Об’єктно-орієнтована (ОО) модель, показана на рисунку 1.13, ґрунтується на принципах Domain-Driven Design та чотиришарової архітектури, де схема класів слугує «контрактом» між сервісним шаром і шаром зберігання даних. Усі сутності є агрегатними коренями (aggregate roots) — CustomUser агрегує облікові та підписні дані, Video — медіаресурси та метадані відтворення, Comment — вкладені гілки обговорень, SupportMessage — шаблон одиничної взаємодії «користувач ↔ служба підтримки». Така декомпозиція мінімізує транзакційні кордони: кожен агрегат модифікується у межах однієї атомарної операції ORM / SQL. Завдяки такій структурі зміни всередині кожного агрегата ізольовані від інших, що спрощує масштабування сервісів і полегшує впровадження патернів CQRS.

CustomUser наслідує AbstractUser фреймворку Django, але змінює natural key — полем USERNAME_FIELD виступає email.

Атрибутивна частина поділяється на:

- Особисті дані (display_name, country, language, avatar);
- Фінансовий стан (balance: Decimal(10,2));
- Підписка (subscription_name, subscription_price, subscription_status, subscription_description, stripe_customer_id);
- Контроль доступу (role: Enum(user|admin), перевизначені M2M до Group і Permission);
- Індекси створено на email — UNIQUE та на комбінації (role, subscription_status) для швидкого відбору сегментів під час адміністративних запитів;

- Життєвий цикл об’єкта керується сервісом AuthService, який інкапсулює логіку реєстрації, ресетів пароля, синхронізації з Stripe.

Video моделює streamable entity:

- Медіа-поля (preview: Cloudinary(image), video: Cloudinary(video)) зберігаються поза базою; ORM оперує лише URI-рядками, що повертаються Cloudinary SDK;
- Контент-метадані (title, description, created_at, date_ua, date_us) та семантичні теги (hashtags: JSON), які обробляються як масив строк — така схема виключає надлишкові таблиці для тегів і пришвидшує фільтрацію оператором JSON_CONTAINS;
- Контроль доступу (min_subscription_level — ENUM із чотирьох значень). Логіка перевірки описана в § 1.2.2: перед поверненням URI потокового відео сервіс звіряє рівень підписки із полем CustomUser.subscription_name;
- Статистика (likes: INT, M2M liked_by→CustomUser) використовує проміжну таблицю video_likes з PK (video_id,user_id) для унеможливлення дублювання лайків;
- Додаткові структури: timecodes: JSON (масив об’єктів {time,label}) та materials: JSON (посилання на PDF, Google Doc, Git-репозиторій тощо). JSON обрано згідно з принципом schema-on-read: різнорідний набір матеріалів не потребує жорсткої схеми й легко серіалізується через DRF.

Comment пов’язує користувачів із відео через дві зовнішні ключові залежності user_id → CustomUser(id) (ON DELETE CASCADE) та video_id → Video(id) (ON DELETE CASCADE):

- Атрибут parent — самопосилання для вкладених обговорень що дозволяє формувати N-рівневі дерева без окремих таблиць зв’язків;
- Для прискорення відображення стрічки коментарів використовується комбінований індекс (video_id,date DESC);
- Реакції «like» реалізовані M2M liked_by, що семантично відрізняє «оцінку коментаря» від «оцінки відео»;

- Сервіс CommentService інкапсулює правила: максимальна глибина вкладення, санкції за спам, кешування кількості лайків;

Найменший агрегат, який не містить зовнішніх ключів. Винесення в окрему таблицю із PK id забезпечує:

- фізичну ізоляцію від «сутностей основного графа»;
- можливість витримувати *write-heavy* навантаження без блокувань CustomUser.Колонки created_at: DATETIME(6) автоматично індексуються для SLA-звітів служби підтримки.

Відображення на рівень СУБД:

- ORM-to-SQL mapping: таблиці у форматі InnoDB, кодування utf8mb4, дефолтний *read-committed* рівень ізоляції транзакцій;
- Foreign Keys у MySQL 8.0.27 підтримують ON UPDATE CASCADE, що полегшує масові міграції (наприклад, об'єднання акаунтів);
- JSON-поля індексуються виразами GENERATED COLUMN + INDEX (hash_tag) для пошуку по тегах;
- Partitioning можливе для Video (BY RANGE (created_at)), аби прискорити cold / archive storage.

Транзакційні аспекти:

- Створення відео: одна транзакція, що вставляє запис Video + CloudinaryUpload (async); при помилці SDK іде ROLLBACK.
- Публікація коментаря: транзакція Comment INSERT → UPDATE Video.comment_count;
- Оплата підписки: зовнішня орієнтована ACID-границя – Stripe → webhook → UPDATE CustomUser subscription_*; підтвердження записується в audit_logs.
- Доступ до додаткових матеріалів, в залежності від рівня доступу користувач має право на інформацію та різні мультимедійні дані, зокрема, відео, посилання на матеріали.

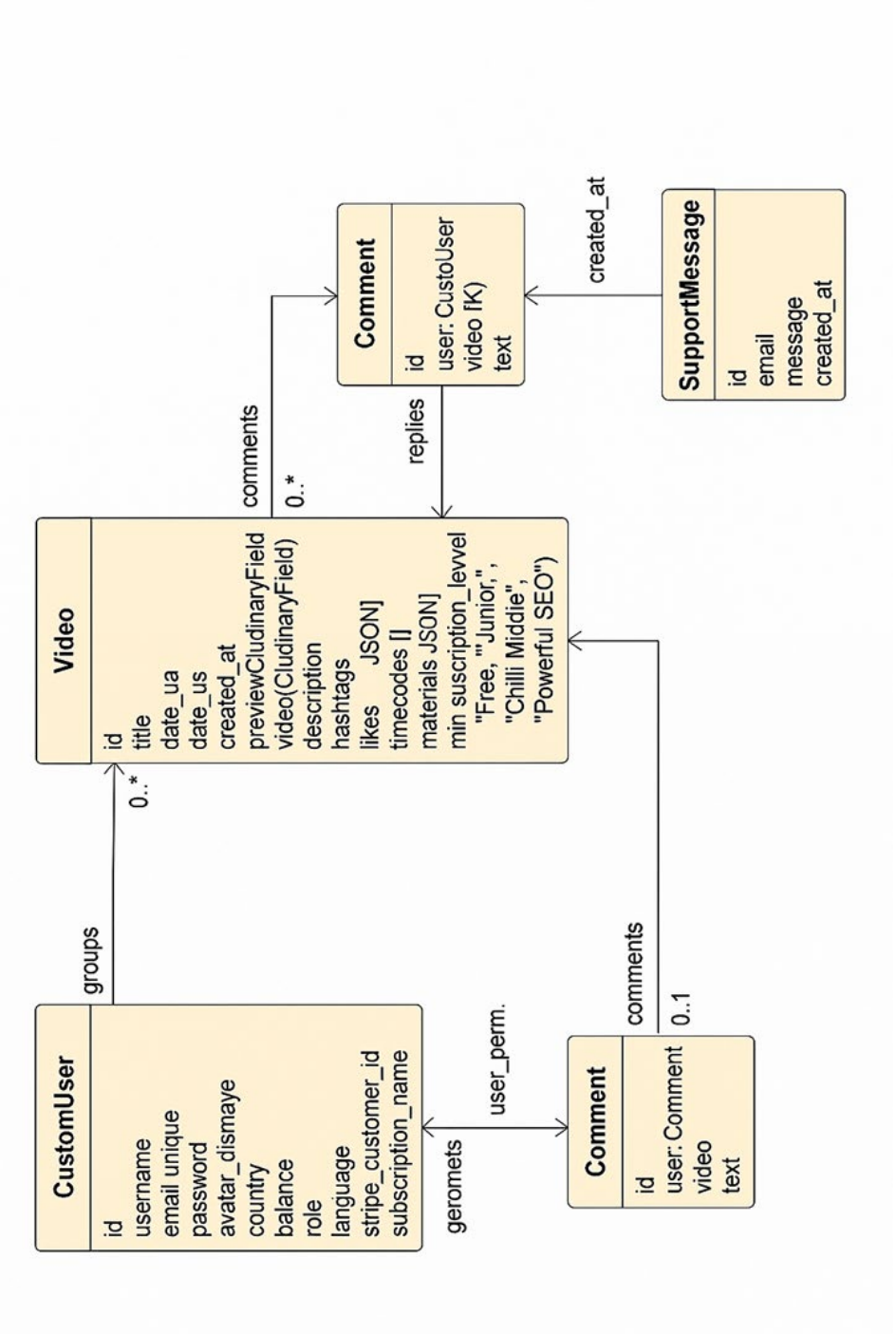


Рисунок 1.13. Загальна об’єктно-орієнтована модель застосунку

Додаткову роль у структурі об’єктної моделі відіграють сервіси бізнес-логіки (service layer), які опосередковують взаємодію між контролерами (API-кінцевими точками) та ORM-моделями.

Наприклад, PaymentService відповідає за створення сесій оплати у Stripe, перевірку webhook-повідомлень, обробку статусів транзакцій та оновлення підписки в моделі CustomUser. Цей сервіс також фіксує журнал операцій у

таблиці audit_logs, в якій зберігається повна інформація про час, користувача, дію, статус відповіді від Stripe API та результат операції.

Щоб гарантувати цілісність зв'язків, було застосовано каскадні операції на рівні бази даних (ON DELETE CASCADE), що особливо важливо для моделей з високим ступенем зв’язаності (наприклад, Comment, Video). Це дозволяє автоматично видаляти коментарі разом із видаленням відео або облікового запису користувача, запобігаючи накопиченню «осиротілих» записів і помилкам цілісності.

Під час розробки також враховано потребу в гнучкому управлінні доступом. Атрибут min_subscription_level у моделі Video співвідноситься із полем subscription_name у CustomUser через логіку сервісного шару. Перевірка відбувається під час кожного запиту до відео, причому кешована відповідь (наприклад, через Redis або Django cache) може використовуватися для зниження навантаження на БД. Така система дозволяє швидко розгортати рівні доступу без необхідності змін у структурі таблиць.

Модель SupportMessage має вузьку спеціалізацію — зберігання звернень у службу підтримки. Вона повністю ізольована від основного об'єктного графа (має лише власний первинний ключ), що дозволяє застосовувати спеціалізовану політику партиціювання або видалення (наприклад, щоквартальне очищення архівних звернень через скрипти cron). Кожне повідомлення зберігається з точністю до мікросекунд (DATETIME(6)), що дозволяє будувати високоточну аналітику навантаження на підтримку, у тому числі у вигляді SLA-метрик.

З технічного боку, реалізовано політику lazy loading для важких полів (наприклад, CloudinaryField у Video) щоб уникати надмірного завантаження даних. Кожне посилання на медіафайл є лише URI, який при потребі довантажується через Cloudinary SDK на стороні клієнта або бекенду.

Цей підхід дозволяє одночасно зменшити обсяг оперативної пам’яті, потрібної для запиту, та забезпечити контроль над терміном дії (TTL) ресурсів.

Загальна модель відповідає патерну «анемічної» доменної моделі, де сутності містять лише дані, а вся бізнес-логіка винесена в сервіси.

Це значно підвищує тестованість застосунку, дозволяє писати модульні тести без підключення до БД (mock-ін'єкція моделей) і розділяє відповідальність між компонентами системи.

Підтримка багатомовності реалізована через поле language у моделі CustomUser, що дозволяє адаптувати інтерфейс, API-повідомлення та email-розсилки згідно з уподобаннями користувача. Локалізація здійснюється за допомогою механізму gettext, з урахуванням мови профілю.

З точки зору відповідності GDPR та іншим стандартам захисту даних, ОО-модель дозволяє сегментувати персональні дані, використовуючи логіку soft-delete або маскування (mask_email, anonymize_name), що може бути легко реалізовано на рівні сервісу або окремого обробника (наприклад, для видалення акаунту користувача після запиту на «право на забуття»).

Усі сервіси використовують транзакційну обгортку (@transaction.atomic), що забезпечує цілісність навіть у разі часткових помилок (наприклад, коли створено коментар, але не оновлено кількість відповідей у відео — система відкотить транзакцію повністю). В деяких випадках застосовується «сагова» стратегія: Stripe оплата → success webhook → CustomUser.subscription_status → запис у log, при цьому кожен крок має fallback-обробку у разі невдачі. Цей підхід дозволяє гарантувати консистентність даних навіть у розподілених сценаріях, забезпечуючи атомарність і контрольований відкат дій на кожному етапі бізнес-процесу. Таким чином, представлена модель поєднує кращі практики об'єктно-орієнтованого проектування: агрегацію, інкапсуляцію, зв'язність та розділення відповідальностей, реалізуючи масштабовану архітектуру, яка легко адаптується до змін у бізнес-вимогах і відповідає сучасним стандартам безпеки, продуктивності та підтримки.

Блок-схема, демонструє етапи процесу оформлення платної підписки користувачем за допомогою інтеграції з платіжною системою Stripe(рисунок 1.14). Кожен елемент схеми відповідає чітко визначеній дії або логічній розвилці, що дозволяє зрозуміло представити послідовність обробки транзакцій — від ініціації запиту на оплату до підтвердження та надання доступу до цифрового

контенту. Далі наведено поетапний опис кожного блоку цієї схеми, його функціонального призначення, взаємодії з іншими компонентами системи та технічних особливостей реалізації.

Початковий етап означає старт сценарію, що ініціюється користувачем. Це може бути натискання кнопки «Оформити підписку» на фронтенді (React). На цьому кроці система перевіряє, чи автентифікований користувач, чи не має активної підписки, а також чи доступна Stripe-сесія. Усі ці перевірки виконує бекенд-сервіс `AuthService` або `PaymentService` в рамках захищеного маршруту API.

Користувач ознайомлюється із доступними рівнями підписок, що попередньо отримані через Stripe Price API. Вибір конкретного плану (наприклад, «Junior», «Chilli Middle», «Powerful SEO») зберігається у стані сесії або у формі запиту до бекенду. Цей крок відповідає за передачу параметра `priceId` до наступного етапу — формування Stripe-сесії. Важливо, що система не дозволяє одночасну активацію кількох підписок, тому логіка контролю унікальності реалізується на рівні БД та Stripe

На цьому етапі відбувається ключова інтеграція з API платформи Stripe. Django-сервер за допомогою бібліотеки `stripe-python` викликає метод `stripe.checkout.Session.create`, передаючи параметри: `customer`, `price`, `success_url`, `cancel_url`, а також опціонально — промокод або підписаний token доступу. Створена сесія відповідає одноразовому платіжному процесу, посилання на який передається фронтенду. Сам об’єкт `Session` є ідемпотентним і живе обмежений час (TTL ~15 хв).

Фронтенд здійснює HTTP-редірект до зовнішньої платіжної форми Stripe. Цей етап виносить відповідальність за збирання карткових із домену застосунку, виключаючи його з PCI DSS scope. Хостинг, валідація та 3D Secure автентифікація відбуваються повністю на стороні Stripe, що гарантує безпеку обробки чутливих даних.

На цьому етапі користувач взаємодіє безпосередньо з інтерфейсом Stripe, де вводить реквізити платіжної картки. Інтерфейс хоститься на стороні Stripe і

відповідає всім вимогам PCI DSS, зокрема обов'язковій автентифікації через 3D Secure (включаючи PSD2-регіони). Після успішного введення даних Stripe створює об'єкт `Invoice`, а також `Subscription`, який є центральним у керуванні рекурентними платежами. У разі схвалення транзакції банком-картоотримувачем статус підписки стає `active`, а користувач миттєво отримує підтвердження в інтерфейсі. Якщо платіж відхиляється (через відсутність коштів, неправильні реквізити, відмову 3DS), Stripe генерує подію `invoice.payment_failed`. Користувач не отримує доступу до ресурсу до моменту повного підтвердження з боку Webhook. Саме це відокремлення обробки платежу і доступу забезпечує додаткову безпеку та запобігає помилковому наданню доступу.

Умовна перевірка «Оплата успішна?» цей логічний елемент вказує на необхідність гілкування залежно від результату оплати. Ромб символізує контрольну точку, на якій перевіряється статус транзакції. Якщо Stripe підтвердив платіж і виконав створення `Subscription`, далі йде обробка webhook; у протилежному випадку — тригериться логіка невдалого платежу. На технічному рівні це реалізується через `event.type`, що містить або `checkout.session.completed`, або `invoice.payment_failed`.

У системі реалізовано fail-safe механізм: навіть якщо користувач спробує обійти фронтенд та отримати доступ до контенту вручну, він не зможе цього зробити без зміни статусу в БД через Webhook. Це жорстке розмежування між оплатою та авторизацією дозволяє запобігти шахрайству або доступу до платного контенту без підтвердженого платежу.

Webhook є єдиним надійним джерелом істини щодо підтвердження платежу. Stripe надсилає запит на endpoint `/stripe/webhook`, який обробляється окремим APIView із декоратором `@csrf_exempt`.

У тілі запиту міститься JSON-представлення події, зокрема унікальний `event.id`, підпис, тип події (`checkout.session.completed`) і посилання на клієнта (`customer`). Перевірка валідності виконується через Stripe SDK з використанням секретного ключа webhook.

Ідемпотентність забезпечується завдяки збереженню `event.id` у локальній таблиці `stripe_events`, а всі транзакції, пов’язані з оновленням CustomUser, виконуються в межах `@transaction.atomic`, щоб уникнути часткових змін. Успішна обробка включає оновлення підписки, запис в `audit_logs` та ініціалізацію надання доступу до захищеного ресурсу.

Після обробки webhook-події checkout.session.completed, отриманої від Stripe, бекенд-сервер виконує транзакційне оновлення профілю користувача. Зокрема, у базі даних змінюються поля, пов’язані з підпискою: subscription_status, subscription_name, subscription_price, subscription_description. Це оновлення фіксується як точка входу в новий рівень доступу і виступає тригером для механізму авторизації до платного контенту.

Далі система передає оновлені дані в кеш (Redis) у вигляді зв’язки user_id → subscription_level. Цей кеш виконує роль швидкої lookup-таблиці, яка використовується всіма сервісами, що обслуговують запити до захищених ресурсів, включаючи відео, файли, документи або API-ендпоінти.

Фізичний доступ до контенту регулюється окремим сервісом — умовно названим ContentGate. Цей сервіс реалізує авторизаційну логіку і перевіряє, чи відповідає поточний рівень підписки користувача вимогам контенту, до якого надсилається запит.

Наприклад, якщо відео має атрибут min_subscription_level="Chilli Middle", то користувач із рівнем Junior не отримає доступу — у відповідь повертається статус 403 Forbidden з поясненням «Недостатній рівень підписки», тобто для отримання матеріалу потрібно підняти свій рівень.

При кожному запиті до відеоплеєра або сторінки матеріалу виконується перевірка прав доступу: спочатку звернення до Redis-кешу для отримання рівня підписки, потім — порівняння з мінімальним рівнем, зазначеним у метаданих ресурсу.

Якщо значення відповідають або перевищують очікуване — доступ дозволено; якщо ні — виконується редирект на сторінку оновлення підписки або повертається помилка з підказкою для користувача.

Рисунок 1.14 Блок-схема процесу оформлення та обробки підписки через Stripe

Крім кешу, система також підтримує механізм миттєвої реакції на зміну підписки. У разі поновлення чи скасування активної підписки через Webhook або вручну з боку адміністратора, Redis-кеш автоматично оновлюється, а клієнтський інтерфейс синхронізується в реальному часі через WebSocket або push-події.Завдяки цьому користувач не стикається з затримками чи конфліктами у відображенні контенту, а зміни прав вступають у дію негайно.

Реалізація логіки доступу дотримується принципу fail-safe: у разі втрати зв’язку з Redis або невизначеного статусу доступ блокується, і лише підтверджена інформація з кешу або БД дозволяє обробити запит. Це забезпечує

безпечну модель контролю, яка не дозволяє обійти обмеження навіть у нестандартних умовах (наприклад, при повторних запитах через API, обхід кешу тощо).

У контексті підписної моделі, де кожна транзакція є критичною для підтримки доступу до преміального контенту, обробка невдалих платежів відіграє ключову роль у підтриманні цілісності системи, а також у підвищенні рівня обслуговування клієнтів. Якщо з будь-якої причини оплата не проходить, Stripe автоматично ініціює подію типу invoice.payment_failed, яка слугує початком реактивного ланцюга у бекенді.

Webhook-процесор у Django негайно приймає цю подію та виконує кілька дій. По-перше, інформація про невдалу транзакцію зберігається в спеціалізовану таблицю payment_failures. До цієї таблиці заносяться такі поля: унікальний ідентифікатор події (event_id), дата і час невдачі (timestamp), тип помилки (failure_reason), ідентифікатор користувача (user_id) та пов’язаний subscription_id. Такий підхід дозволяє не тільки зберігати повну історію помилок для подальшого аналізу, але й реалізовувати повторні спроби або альтернативні сценарії для конкретного користувача.

Після фіксації помилки бекенд запускає Celery-задачу, яка виконує відкладене сповіщення користувача. Якщо увімкнено email-нотифікації — користувач отримує лист із поясненням помилки та рекомендаціями для повторного оформлення підписки.

У випадку, якщо користувач перебуває онлайн, сповіщення з’являється безпосередньо в інтерфейсі за допомогою WebSocket-з’єднання.

При цьому система підтримує обробку повторних спроб оплати відповідно до політики Stripe. Якщо кілька платежів підряд завершуються невдало (наприклад, три рази з інтервалом у добу), система переводить статус підписки у canceled, що автоматично блокує доступ до платного контенту. Важливо, що така логіка реалізується ідемпотентно, тобто навіть якщо Stripe кілька разів надсилає одну й ту ж подію, база даних не створить дублів і не виконає зайвих дій.

1.3.3 Розробка інтерфейсів користувача: відеоплеєр, профіль, сторінка підписок

Фронтенд застосунку реалізований за допомогою React та організований компонентно, що забезпечує гнучку масштабованість та реактивне оновлення даних. Ключовими інтерфейсами користувача є сторінка перегляду відео з інтегрованим плеєром, персональний профіль користувача та сторінка підписок. Всі інтерфейси адаптовані до ролей користувачів, рівнів підписки та інтеграції з backend-сервісами.

Компонент VideoPlayer.tsx відповідає за програвання відео з Cloudinary через ReactPlayer. Плеєр підтримує стандартні функції — відтворення, пауза, прогрес-бар, зміна гучності, вибір швидкості, а також режим постера (light preview). Логіка обмеження доступу реалізована на клієнті через порівняння рівня доступу користувача (subscription_level) із рівнем, встановленим для відео. Якщо рівень недостатній — плеєр блокується з повідомленням.

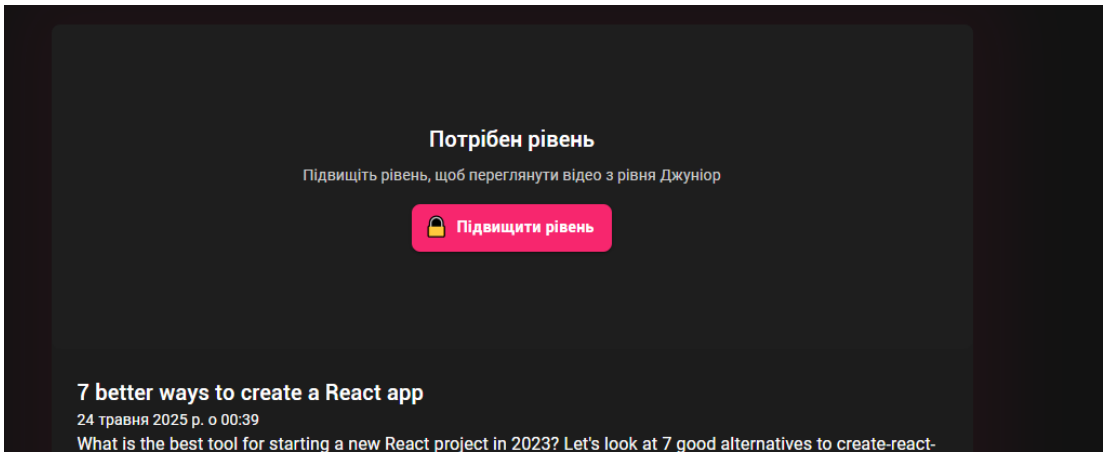


Рисунок 1.15 Інтерфейс відеоплеєра з обмеженим доступом

Компонент HomeScreen.tsx дозволяє переглядати та частково редагувати персональні дані. Інформація включає ім’я користувача, email, країну, аватарку, роль і поточний статус підписки. Усі дані беруться з localStorage після авторизації або запитуються з сервера. Окрім цього, сторінка містить кнопку переходу до оформлення підписки або її зміни. Зміни зберігаються на сервері через API. Email і роль не редагуються — вони лише для перегляду. Статус підписки оновлюється після запиту до сервера. Якщо підписки немає або вона неактивна, показується

кнопка переходу до сторінки оплати. Дані з localStorage синхронізуються з серверними при кожному завантаженні сторінки.

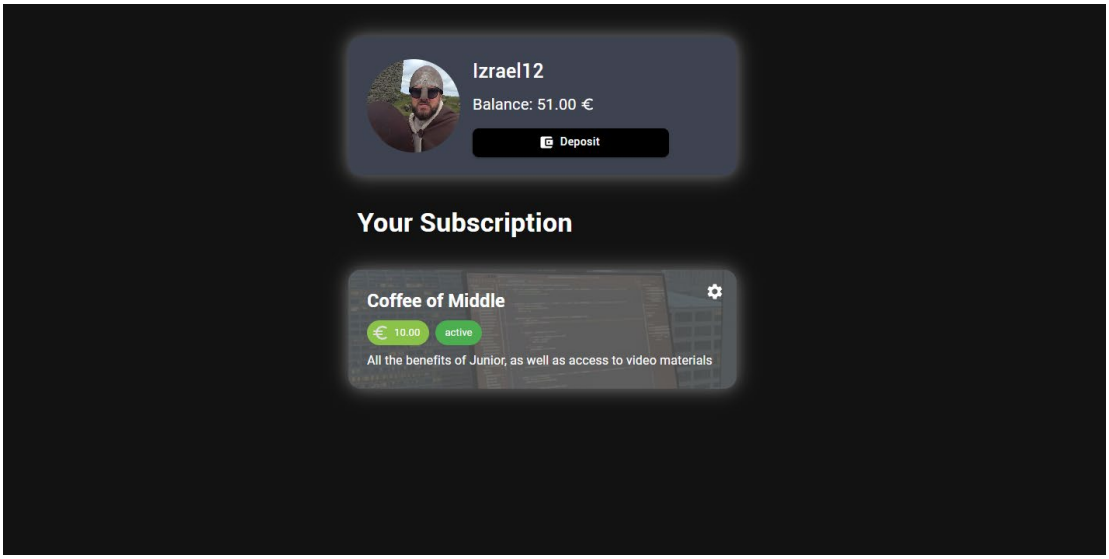


Рисунок 1.16 Сторінка профілю користувача з активною підпискою

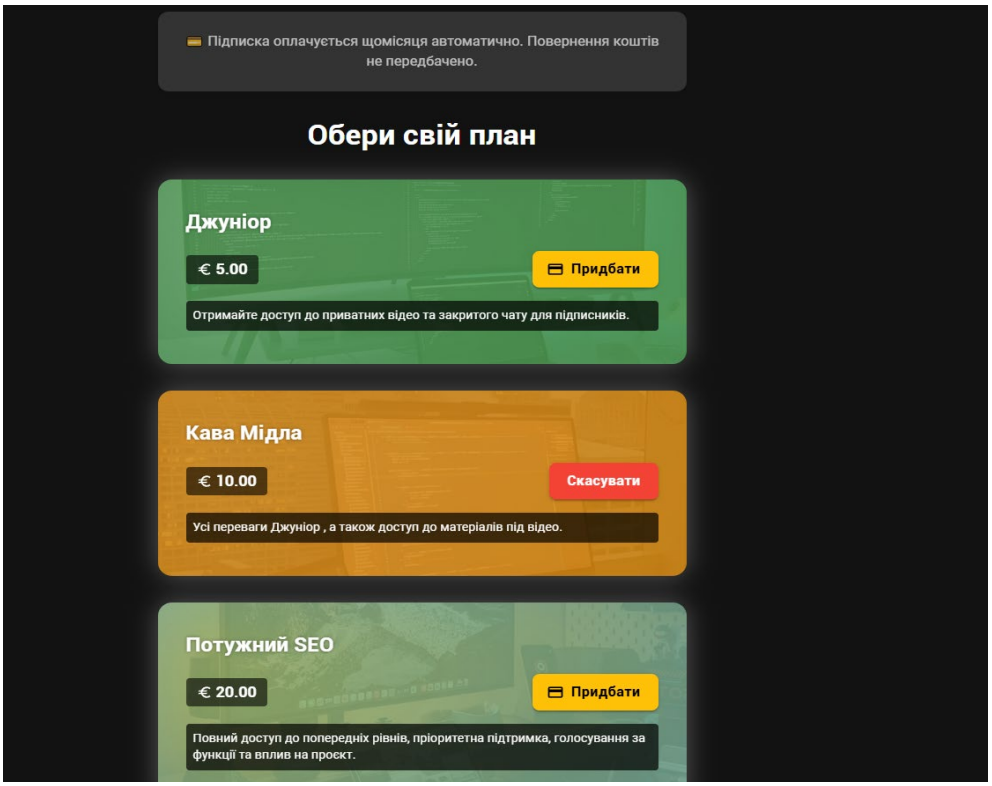


Рисунок 1.17 Вибір тарифного плану підписки

Сторінка SubscriptionPage.tsx реалізує інтерактивний інтерфейс для вибору та оформлення плану підписки. Кожен тариф відображається у вигляді картки з назвою, описом, ціною та кнопкою "Підписатись". При натисканні генерується Stripe Checkout Session, і користувач перенаправляється на сторінку оплати.

Після успішного платежу статус оновлюється як у базі, так і в UI, через відповідний Webhook.

Всі зміни у підписці миттєво відображаються в інтерфейсі, що забезпечується оновленням глобального стану користувача через контекст. Це дозволяє уникати повторних запитів до сервера і зменшує затримки у відображенні оновлених доступів.

1.3.4 Реалізація адміністративної панелі керування підписками та користувачами

Компонент StudioPage.tsx виступає ядром адміністративної панелі керування системою підписок та обліковими записами. Його основною функцією є централізоване візуальне представлення усіх користувачів платформи з можливістю виконання CRUD-операцій (створення, оновлення, видалення, перегляд) для кожного з них. Компонент монтується після авторизації адміністратора та ініціює запит типу GET /api/users/, який передає JWT-токен у заголовках авторизації. У відповідь сервер повертає повну колекцію об’єктів користувачів, кожен з яких містить атрибути: id, email, username, role, subscription_name, subscription_price, subscription_status, balance тощо.

Отримані дані відображаються у вигляді інтерактивної таблиці, реалізованої засобами бібліотеки Material UI (@mui/material) з використанням компонентів Table, TableRow, TableCell, Select, IconButton тощо. Для кожного рядка таблиці рендеряться ключові поля, які є критично важливими для контролю: електронна пошта, ім’я користувача, роль у системі (admin або user), активний тариф (наприклад, Free, Junior, Chilli, Powerful SEO), статус підписки (active, canceled, expired), вартість тарифу, а також баланс рахунку, що відображає доступні кошти користувача у форматі Decimal.

Візуально таблиця адаптується до будь-якої ширини екрану та підтримує прокрутку, що дозволяє масштабувати її під великі. Приклад відображення таблиці керування користувачами з можливістю редагування тарифу та виконання інших дій наведено нижче (рисунку 1.18).

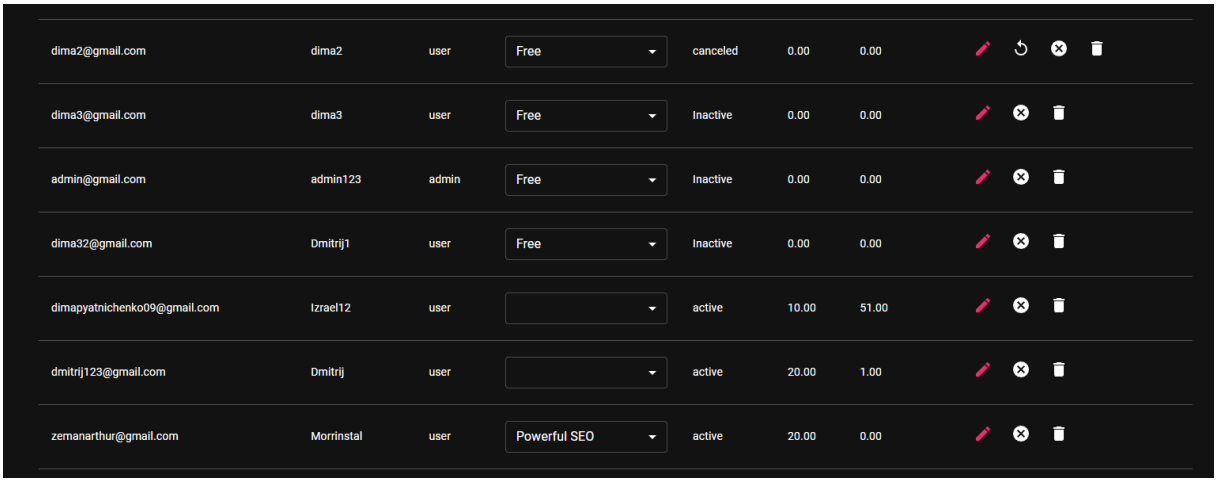


Рисунок 1.18 Фрагмент таблиці з можливістю інлайнового редагування тарифу

Окрім редагування тарифу, таблиця також містить інші кнопки керування: активація, скасування підписки, редагування даних користувача, повне видалення облікового запису. Зокрема, кнопка скасування підписки викликає POST-запит до api/admin/cancel-subscription/{id}/, який змінює статус на canceled. При цьому тариф залишається прив’язаним до користувача, але деактивованим. Для повторного підключення підписки доступна кнопка реактивації, яка викликає PATCH із параметром subscription_status: "active".Детально представлено на (рисунок 1.19)

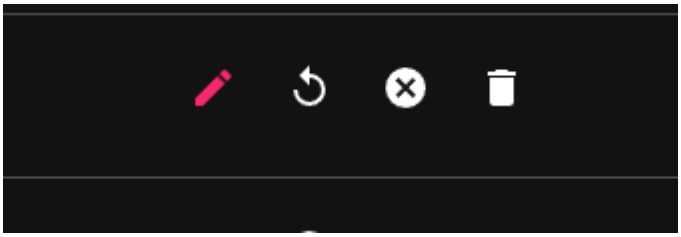


Рисунок 1.19 Елементи керування: редагування, скасування, активація, видалення

Після підтвердження змін надсилається PATCH-запит до відповідного користувача.Якщо оновлюваний користувач є поточним авторизованим (currentUser), то його дані автоматично зберігаються у localStorage, щоб уникнути розсинхронізації при наступному оновленні сторінки.Таким чином, уся взаємодія є реактивною – дані у таблиці оновлюються одразу після успішного запиту, без необхідності ручного перезавантаження.

Редагування персональних даних користувача реалізується через компонент EditUserModal.tsx.

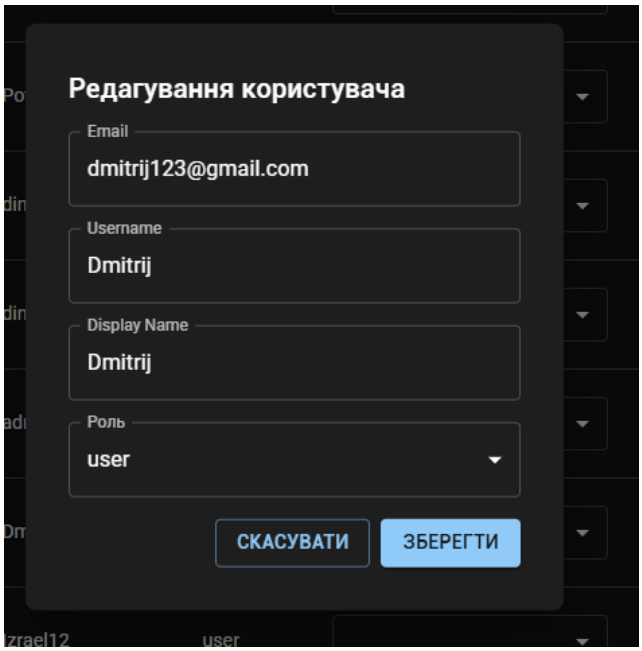


Рисунок 1.20 Модальне вікно редагування атрибутів користувача

Даний компонент забезпечує функціонал повного видалення користувача, який реалізується через HTTP-запит DELETE /api/users/{id}/. Цей запит активується адміністратором для видалення конкретного користувача, ідентифікованого унікальним ідентифікатором {id}, і виконується на сервері для безповоротного видалення відповідного запису з бази даних, виключаючи будь-які резервні копії чи архіви. Процес не передбачає можливості відновлення даних, що забезпечує надійність операції, запобігаючи випадковому поверненню компрометованих акаунтів, і підкріплюється логуванням дії для подальшого аудиту. Завдяки асинхронному оновленню інтерфейсу таблиця користувачів миттєво відображає зміни, усуваючи потребу в повному перезавантаженні сторінки, що включає автоматичне видалення рядка з видаленим користувачем і оновлення пагінації. Такий механізм сприяє підвищенню динамічності інтерфейсу, надаючи адміністраторам інструмент для швидкого реагування на порушення правил використання системи, таких як спам чи зловживання доступом. Крім того, він дозволяє ефективно керувати доступом користувачів у реальному часі, що є критично важливим для підтримки безпеки платформи, особливо під час масових атак чи витоків даних. Ця функціональність підтримує оперативне видалення компрометованих акаунтів (наприклад, після злому),

неактивних профілів (які не використовувалися понад 6 місяців) або тих, що порушують політику (наприклад, порушення авторських прав), зменшуючи потенційні загрози для інфраструктури, такі як перевантаження серверів чи поширення шкідливого контенту.

Для поля email у формі передбачена регулярна валідація формату, яка гарантує коректність введених даних шляхом перевірки відповідності стандартному шаблону електронної пошти (наприклад, наявність символу "@", доменного імені типу .com чи .org, правильної структури адреси з локальною та доменною частинами). Ця валідація виконується на стороні клієнта перед відправленням даних, використовуючи вбудовані JavaScript-регулярні вирази, запобігаючи помилкам під час реєстрації (наприклад, введення некорректного домену) або оновлення профілю (наприклад, зміна на неіснуючу адресу) та забезпечуючи надійність контактної інформації для сповіщень і відновлення доступу. Поле username (ім’я користувача) перевіряється на унікальність за допомогою запиту HEAD /api/users/?username=..., який відправляється до сервера для перевірки наявності ідентичного імені в базі даних через аналіз таблиці користувачів, повертаючи статус 200 (унікально) або 409 (конфлікт). Цей запит дозволяє уникнути дублювання імен, підтримуючи унікальність ідентифікаторів у системі, що критично для ідентифікації користувачів у багатокористувацькому середовищі. Поле display name (відображуване ім’я) обмежене максимум 30 символами, що встановлює чіткий ліміт для довжини імені, зберігаючи його читабельність і запобігаючи надмірному навантаженню інтерфейсу, при цьому спеціальна валідація для цього поля не застосовується, надаючи користувачам свободу у виборі стилю написання, включаючи пробіли, дефіси чи спеціальні символи (наприклад, ©), за умови дотримання встановленого ліміту.

Перед відправленням виконується попередня валідація введених значень на клієнтській стороні, що включає перевірку формату полів (email, username), їх відповідність встановленим критеріям (наприклад, довжина display name до 30 символів) і відсутність порожніх значень, дозволяючи уникнути помилок на етапі

обробки сервером, таких як 400 Bad Request, і зменшуючи навантаження на бекенд.

У разі успішного завершення запиту модальне вікно автоматично закривається, завершуючи процес редагування профілю чи інших даних, очищаючи робочий простір від тимчасових елементів, таких як поля введення чи кнопки підтвердження, і повертаючи користувача до основного інтерфейсу для продовження роботи з таблицею користувачів, перегляду контенту чи переходу до інших розділів, таких як статистика чи налаштування. Цей процес супроводжується анімацією закриття вікна для зручності сприйняття та забезпечує чітке завершення дії, виключаючи плутанину з відкритими елементами. Одночасно таблиця користувачів автоматично оновлюється — зміни відображаються в реальному часі без необхідності перезавантаження всієї сторінки, що включає оновлення рядків із відредагованими даними, коригування пагінації для правильного відображення кількості записів та оновлення активних фільтрів (наприклад, за статусом підписки), завдяки асинхронній синхронізації з API, яка реалізується через періодичні опитування серверу або автоматичне оновлення через HTTP-запити. Такий підхід до оновлення інтерфейсу значно підвищує швидкодію панелі адміністратора, уникаючи зайвих повторних запитів до сервера, оптимізуючи мережеву активність шляхом зменшення обсягу переданих даних і забезпечуючи безперервність роботи навіть при великій кількості одночасних операцій, таких як редагування десятків профілів. Він створює враження роботи з «живими» даними, що покращує загальний досвід користувача через миттєву реакцію системи на дії (наприклад, видимість змін протягом 1-2 секунд), підвищує ефективність управління системою за рахунок швидкого доступу до актуальної інформації без ручного оновлення та дозволяє адміністраторам зосереджуватися на ключових операціях, таких як масове редагування профілів, видалення неактивних акаунтів чи зміна тарифів, без затримок, зберігаючи продуктивність у пікові періоди, коли кількість одночасних користувачів може сягати сотень.

Рисунок 1.21 Інтерфейс панелі керування користувачами та підписками

Якщо ж сервер повертає помилку (наприклад, у разі дублювання імені користувача через конфлікт із вже існуючим username або невірного формату email, що не відповідає стандартам валідації), на інтерфейсі негайно з’являється інформативне повідомлення про помилку, яке включає деталі проблеми, такі як код помилки чи конкретне поле, що потребує виправлення. Це повідомлення відображається у вигляді модального вікна або інлайн-повідомлення, дозволяючи

користувачу оперативно внести коригування без втрати контексту роботи. За допомогою цього механізму всі залежні від ролі або рівня доступу елементи інтерфейсу, як-от бокова панель (Sidebar) з адміністративними функціями, контекстні меню для редагування чи видалення даних, або кнопки доступу до захищених розділів, миттєво адаптуються до нового стану без необхідності перезавантаження сторінки або оновлення сесії. Така адаптація включає приховування чи розблокування елементів залежно від оновлених прав доступу, що забезпечується асинхронною синхронізацією з сервером і локальним оновленням стану додатку. Такий підхід гарантує безперервність роботи та зручність для користувача, особливо в умовах динамічних змін ролей чи статусів.

Тому значно покращує користувацький досвід, оскільки користувач бачить результат своїх дій одразу після їх виконання, що створює відчуття плавної та живої взаємодії з системою. Така миттєва віддача дозволяє уникнути затримок у зворотному зв’язку, що особливо критично для користувачів, які звикли до високої швидкості роботи сучасних додатків. Така поведінка особливо важлива в умовах багатокористувацького середовища з підвищеним навантаженням, де швидкість синхронізації стану між клієнтом і сервером відіграє ключову роль у збереженні стабільності й консистентності даних.І забезпечується асинхронними оновленнями, які мінімізують час очікування та запобігають конфліктам даних між одночасними діями кількох користувачів. Такий підхід також сприяє підвищенню довіри до системи, оскільки користувачі отримують підтвердження своїх дій у реальному часі, що особливо цінно для адміністраторів, які керують великою кількістю операцій.

Окремо варто відзначити інтеграцію із системою повідомлень, яка підтримує ефективну взаємодію між адміністратором і системою. Усі ключові дії адміністратора, такі як оновлення профілю користувача, зміна тарифу підписки або видалення акаунта, супроводжуються візуальними індикаторами — короткі toast-повідомлення з’являються на екрані та інформують про успішність операції чи виникнення помилок при взаємодії з бекендом. Ці повідомлення включають деталі, такі як ідентифікатор дії чи код помилки, що полегшує діагностику

проблем.Особливо у багатокористувацьких інтерфейсах, де адміністратор може виконувати велику кількість дій поспіль, наприклад, масове редагування профілів або обробку скарг, і має бути впевненим у результаті кожної з них. Такий підхід зменшує ймовірність пропуску критичних подій і дозволяє оперативно реагувати на збої, зберігаючи контроль над системою навіть у пікові періоди навантаження.

Список користувачів сортується за замовчуванням за датою створення, що дає змогу одразу бачити останніх зареєстрованих учасників і спрощує моніторинг нових акаунтів. Такий порядок сортування є логічним для адміністративних задач, таких як перевірка активності чи виявлення підозрілих реєстрацій. Також реалізовано можливість сортування за іншими полями — зокрема, email або статусом підписки, що дозволяє адміністраторам швидко знаходити потрібних користувачів залежно від контексту роботи. Кожне поле сортування підтримує як ascending, так і descending порядок, що розширює гнучкість інтерфейсу. Крім цього, у верхній частині таблиці розташоване поле пошуку, яке дозволяє знайти користувача за email або іменем у режимі реального часу, забезпечуючи інтуїтивно зрозумілий спосіб фільтрації даних.

Якщо токен прострочено або втрачено, відповідні дії, такі як редагування чи видалення користувача, блокуються, і користувачу виводиться сповіщення з чіткою пропозицією повторно авторизуватися для відновлення доступу. Ця перевірка реалізована на рівні утиліти authFetch, яка централізовано додає токен до заголовків кожного запиту, перевіряє його валідність і обробляє помилки 401 шляхом виклику silent-refresh для автоматичного оновлення токена або перенаправлення на сторінку логіну у разі невдачі. Такий механізм забезпечує безперервність роботи за умови коректного оновлення сесії та захищає систему від несанкціонованого доступу. Завдяки цьому функціоналу адміністративна панель не лише забезпечує повноцінне керування користувачами та підписками, але й гарантує безпеку, масштабованість та UX, адаптований під сучасні вимоги, включаючи підтримку високого навантаження та захист даних.

1.3.5 Тестування розробленого програмного забезпечення

Тестування розробленої системи здійснювалося з метою підтвердження її стабільної роботи, відповідності заявленим функціональним вимогам, надійності обробки даних у різних сценаріях та забезпечення зручного користувацького досвіду для всіх ключових ролей — адміністратора, звичайного користувача та підписника. Особливу увагу було приділено перевірці ключових компонентів: механізму керування підписками, захисту відеоконтенту від несанкціонованого доступу, функціональності адміністративної панелі для керування користувачами та коректної обробки платіжних подій через інтеграцію з платіжною системою Stripe.

Дані аспекти тестувалися для забезпечення безперебійної роботи системи в реальних умовах експлуатації та відповідності стандартам безпеки та зручності.

Першим етапом тестування стало створення облікових записів із різними ролями: user (звичайний користувач), admin (адміністратор) та тестового користувача з активованою підпискою для імітації повноцінного сценарію використання. Для кожного облікового запису перевірялися ключові сценарії автентифікації, що включали первинну реєстрацію з введенням базових даних (електронна пошта, пароль), авторизацію через введення облікових даних, отримання JWT-токенів для доступу до захищених ресурсів та їх автоматичне оновлення через механізм refresh token, який забезпечує безперервність сесії. Була проаналізована робота механізму збереження даних у localStorage, зокрема коректність зберігання токенів та користувацьких налаштувань, а також перевірено відповідність персоналізованого контенту, що відображається в інтерфейсі, даним, які повертаються з API. Цей етап дозволив підтвердити стабільність автентифікаційного процесу та коректність роботи клієнт-серверної взаємодії.

У разі спроби доступу до відеоконтенту без відповідної підписки інтерфейс коректно відображав обмеження доступу, показуючи користувачу повідомлення про необхідність оформлення підписки, та перенаправляв його на відповідну сторінку з деталями пропозицій. Такий сценарій тестувався для різних типів

контенту з різними рівнями доступу, що дозволило підтвердити правильну реалізацію механізму контролю доступу на клієнтському рівні. Перевірка включала аналіз поведінки системи при спробах обійти обмеження (наприклад, через маніпуляцію URL) та забезпечення, що лише авторизовані користувачі з активною підпискою отримують доступ до захищеного контенту. Цей етап також включав тестування повідомлень про помилки, їх чіткість і зручність для користувача, що сприяло підвищенню якості користувацького досвіду.

Далі було проведено серію функціональних тестів на доступ до різних інтерфейсів залежно від ролі користувача. У випадку користувачів із роллю user система правильно обмежувала доступ до розширених функцій:приховувала кнопки додавання або редагування відео в інтерфейсі, блокувала відображення адміністративного меню та перешкоджала виконанню запитів до захищених маршрутів, таких як панель керування користувачами. Ці обмеження тестувалися в різних сценаріях, включаючи зміну розміру екрана та перехід між розділами, щоб виключити можливі помилки відображення. Додатково перевірялися редиректи після входу в систему, зокрема коректність направлення користувача на відповідний інтерфейс залежно від його ролі, а також відповідність завантажених інтерфейсів ролі користувача після авторизації. Була проаналізована поведінка системи у разі зміни ролі (наприклад, підвищення до admin) або статусу підписки (активація/деактивація) без повторної авторизації, що включало тестування оновлення інтерфейсу в реальному часі та відсутність конфліктів із попередніми налаштуваннями. Цей етап дозволив оцінити гнучкість системи та її здатність адаптуватися до змін у стані користувача.

Окрему увагу приділено тестуванню обробки платіжних подій через Stripe, включаючи створення тестових транзакцій, перевірку коректності відображення статусу підписки після оплати та обробку помилок (наприклад, відмова платежу). Для адміністраторів тестувалася можливість перегляду історії транзакцій у панелі керування, а для користувачів — автоматичне розблокування контенту після успішної оплати. Цей етап включав також перевірку інтеграції з API Stripe,

стабільність обробки webhook-повідомлень та захист даних платіжних карток відповідно до стандартів PCI DSS.

Результати тестування підтвердили надійність платіжного модуля та його інтеграцію з основною системою.

Особливу увагу було приділено інтеграції з платіжною системою Stripe. В режимі тестування (sandbox) виконано перевірку усіх основних сценаріїв:

- оформлення підписки (успішна транзакція);
- відхилення оплати (тестові картки з помилками);
- скасування підписки;
- відновлення активного тарифу.

Реакція сервера на події Stripe (webhooks) перевірялась шляхом моделювання відповідних подій (наприклад, checkout.session.completed, invoice.payment_failed).

Після отримання webhook-даних система оновлювала статус користувача, який у режимі реального часу отримував або втрачав доступ до закритого контенту, відображаючи зміни в інтерфейсі через асинхронне оновлення таблиці користувачів. На рівні бекенду було проведено тестування API за допомогою Postman.

Зокрема, перевірялися:

- Авторизація та refresh-токени.
- Створення та оновлення користувачів через PATCH/DELETE.
- Обробка Stripe Webhooks

Проведено повне тестування адміністративного модуля StudioPage.tsx. У фокусі перевірки — основні CRUD-операції над користувачами: оновлення підписки, зміна ролі, редагування даних та повне видалення облікового запису. Кожна операція супроводжувалась відповідними запитами типу PATCH, DELETE до REST API. Було перевірено:

- коректність оновлення інтерфейсу без перезавантаження сторінки;
- збереження нових тарифів

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

У даному розділі виконується економічне обґрунтування доцільності розробки та впровадження вебзастосунку для керування підписками та мультимедійним контентом, розробленого в межах дипломного проєкту. Розрахунки проводяться з метою оцінки трудомісткості розробки, фінансових витрат на створення програмного продукту, а також формування кінцевої вартості продукту для потенційного споживача або замовника.

Проведення такого аналізу дає змогу визначити доцільність інвестицій у розробку, рівень витрат на оплату праці фахівців, супровідні витрати, а також оцінити рівень прибутковості та конкурентоспроможності розробленого програмного забезпечення. У процесі аналізу використано типові показники трудомісткості, витрат на розробку, нормативи оплати праці та методику формування вартості відповідно до чинних нормативних документів і довідкових матеріалів.

Основною метою даного аналізу є демонстрація економічної доцільності проєкту не лише як навчального кейсу, а й як потенційного комерційного продукту. Визначення загальної трудомісткості, собівартості, рівня рентабельності та кінцевої ціни програмного продукту дає змогу зробити висновки щодо можливості реалізації такого рішення в реальних ринкових умовах.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від обсягу коду, складності та кваліфікації виконавців.

Методом структурної аналогії за відповідними каталогами аналогів програмного забезпечення визначається обсяг програмних засобів у тисячах умовних машинних команд програми-аналога. Підхід ґрунтується на порівнянні функціоналу розроблюваного програмного забезпечення з аналогами.

Таблиця 2.1 Каталог аналогів програмного забезпечення

Найменування ПП	Обсяг функції ПП – V_0, тис. усл. машинних команд
1. ПП СУБД	2,5 – 9,8
2. Комплексні системи ведення БД	0,95 – 7,43
3. ПП введення інформації	1,06 – 5,75
4. ПП оптимізації розрахунків	1,3 – 4,2
5. ПП автоматизації засобів по каталогу	0,68 – 7,0
6. ПП автоматизованих розрахунків	1,3 – 8,6
7. ПП загальної математики і ПП імітаційного моделювання	7,8 – 8,8
8. ПП організації обчислювального процесу	10,2 – 13,0
9. ПП оптимізаційних розрахунків	1,3 – 4,2

Таблиця 2.2 Базові норми часу для розробки ПП

Обсяг ПП, тис. усл. машинних команд	Норма часу, люд/год
1,0	229
2,0	244
3,0	262
4,0	**283**
5,0	306
6,0	330
7,0	357
8,0	385

На підставі каталогу аналогів було вибрано ПП «Введення інформації» (V_0 = 4,0 тис. умовних машинних команд). Згідно з табл. 2.2 базова норма часу становить 283 людино-годин. З урахуванням коефіцієнта застосування сучасної ЕОМ (Кк = 0,75) одержано укрупнену трудомісткість:

Тан = 283 × 0,75 = 212,3 люд-год

Таблиця 2.3 Значення питомих коефіцієнтів трудомісткості стадій

Код стадії	Ступінь новизни А	Ступінь новизни Б	Ступінь новизни В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Таблиця 2.4 Значення поправочного коефіцієнта новизни

Код ступеня новизни	Ступінь новизни	Значення Кн
А	Принципово нові ПП	1,2 – 1,75
Б	ПП – розвиток визначеного параметричного ряду	0,8 – 1,0
В	ПП, що має аналог	0,7

Таблиця 2.5 – Значення коефіцієнта ступеня використання типових програм

Ступінь охоплення реалізованих функцій розроблюваного ПП типовими програмами, %	Значення Кт
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Код новизни – В (аналог), Кн = 0,7. Частка повторного використання модулів ≈ 30 %, тому Кт = 0,8. Питомі коефіцієнти стадій при коді новизни В: $L_1 = 0,12$; $L_2 = 0,11$; $L_3 = 0,61$.

Таблиця 2.6 – Нормативні параметри для оцінки трудомісткості розробки

Параметр	Значення
Тан, люд-год	212,3
Кн	0,7
Кт	0,8
L_1	0,12
L_2	0,11
L_3	0,61

Розрахунок трудомісткості за етапами:

- Трудомісткість технічного завдання: Ттз = 212,3 × 0,12 × 0,7 = 17,9 год
- Трудомісткість технічного проекту: Ттп = 212,3 × 0,11 × 0,7 = 16,4 год

- Трудомісткість робочого проекту: Трп = 212,3 × 0,61 × 0,7 × 0,8 = 72,7 год

Загальна трудомісткість проекту становить Тзаг ≈ 150 люд-год, що еквівалентно 0,085 розробника-року (при 8-годинному робочому дні та коефіцієнті календарного планування 0,73).

2.3 Розрахунок ціни програмного продукту

Даний підрозділ визначає планову собівартість та кінцеву відпускну ціну веб-платформи, розробленої на рівні Junior. Розрахунки виконано нормативним методом із урахуванням чинних тарифів оплати праці, матеріальних витрат, накладних та податкових відрахувань. Калькулювання собівартості здійснюється за статтями витрат з деталізацією прямих та непрямих витрат, що забезпечує точність визначення економічних показників проекту.

Відповідно до ст. 8 Закону України «Про Державний бюджет на 2025 р.», мінімальна погодинна тарифна ставка становить 46 грн. Дана ставка використовується як базова для розрахунку заробітної плати всіх категорій спеціалістів, задіяних у проекті, що відповідає чинному трудовому законодавству України.

У проекті задіяно один розробник рівня Junior і керівник-консультант; нормоконтроль виконується окремим спеціалістом. Загальна трудомісткість (див. п. 2.2) – 150 люд·год. Розподіл ролей передбачає: основну розробку програмного коду виконує Junior-розробник (120 год), технічне керівництво та консультації надає досвідчений спеціаліст (20 год), перевірку відповідності технічним вимогам та стандартам здійснює нормоконтролер (10 год).

Таблиця 2.8 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість, год	Тарифна ставка, грн/год	Сума, грн
Розробка ПП	120	46	5 520
Контроль керівника	20	46	920
Нормоконтроль	10	46	460
Усього Zo	150	—	6 900

Загальний фонд основної заробітної плати: Zo = Σ Tmj · Zгод = 150 год × 46 грн = 6 900 грн.

Таблиця 2.10 Калькуляція планової собівартості програмного продукту

Стаття витрат	Значення, грн	Формула
Матеріали	660	Вм
Основна заробітна плата	6 900	Зо
Додаткова заробітна плата (10 %)	690	0,10 × Зо
Відрахування ЄСВ (22 %)	1 670	0,22 × (Зо + Зд)
Накладні витрати (50 % Зо)	3 450	0,50 × Зо
Повна собівартість Спов	13 370	Σ поз. 1–5

Розрахунок відпускної ціни:

- Планова рентабельність: 15 %
- Плановий прибуток: П = 13 370 × 0,15 = 2 006 грн
- Оптова ціна: Цо = 13 370 + 2 006 = 15 376 грн
- ПДВ (20 %): 15 376 × 0,20 = 3 075 грн
- Ціна реалізації: Цр = 15 376 + 3 075 = 18 451 грн

Розрахункові показники демонструють, що при повній собівартості 13 370 грн і плановій нормі рентабельності 15 % очікуваний прибуток становить 2 006 грн. Додавання прибутку формує оптову ціну програмного продукту на рівні 15 376 грн. Після нарахування податку на додану вартість (20 %) у сумі 3 075 грн кінцева ціна реалізації визначається в обсязі 18 451 грн.

Такі фінансові результати підтверджують економічну доцільність проекту: закладена маржа покриває усі прямі та непрямі витрати, водночас забезпечуючи конкурентну вартість для замовника й прийнятний рівень прибутковості для розробника.

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Оцінка професійних небезпек і шкідливих чинників для фахівця-програміста

У даному підрозділі виконується системний аналіз усіх небезпечних і шкідливих виробничих факторів, що супроводжують діяльність розробника програмного забезпечення у типовому офісному середовищі. Методика аналізу базується на вимогах ДСанПіН 3.3.6.042-99, ДБН В.2.5-28-2006, ПУЕ-2021, ПБЕЕ та інших чинних нормативних актів, а також на рекомендаціях методичних вказівок, наданих кафедрою.

Аналіз шкідливих та небезпечних чинників під час роботи розробника програмного забезпечення:

Класифікація факторів:

- Фізичні: параметри мікроклімату, рівні шуму та вібрації, освітленість, електромагнітні поля ПК, іонізація повітря.
- Хімічні: озон та оксиди азоту від лазерних принтерів, пара та аерозолі засобів очищення, випаровування пластику периферії.
- Біологічні: мікрофлора систем кондиціювання, бактеріальне забруднення клавіатури та миші.
- Ергономічні та психофізіологічні: статична поза, монотонне зорове навантаження, стрес від дедлайнів, нерівномірний робочий ритм.
- Електротехнічні та пожежні: струмоведучі частини, перевантаження мережі, займання кабель-каналів і блоків живлення

3.2 Гігієнічні вимоги до виробничого середовища

Цей розділ визначає гігієнічні вимоги до створення безпечного та здорового виробничого середовища, охоплюючи такі аспекти, як приміщення, освітлення, шум, організація робочого місця, мікроклімат та електробезпека.

3.2.1 Вимоги до приміщення

Приміщення для розробників належить до категорії офісних IIIб і повинно розташовуватися у сухій, опалюваній частині будівлі.Мінімальна висота стелі — 3,2 м, площа на одного працівника не менше 6 м², об'єм — 20 м³.

Стіни виконують із матеріалів класу пожежної небезпеки Г1, підлогу — з антистатичного лінолеуму та піднятою кабель-системою, що спрощує евакуацію та сервісне обслуговування електромереж. Робоча зона відокремлюється від коридорів глухими перегородками висотою 1,5 м, що знижує рівень стороннього шуму й візуальних відблисків.

3.2.2 Освітлення

Освітлення відповідає ДБН В.2.5-28-2018 із мінімальною горизонтальною освітленістю 500 лк при змішаному типі (природне через вікна з КПО ≥ 1,5% та штучне від LED-панелей 4000 K). Заміна люмінесцентних ламп на LED забезпечила 610–630 лк, перевищуючи норму на 10%. Світильники розміщено паралельно лінії погляду, уникаючи відблисків, із контрастом «екран–тло» 1,5–2:1 і різницею яскравостей у полі зору до 3:1 (EN 12464-1).

3.2.3 Шум

Допустимий рівень еквівалентного шуму у приміщеннях з персональними ЕОМ — 50 дБА. Джерелами акустичного навантаження є системні блоки, серверні стійки, системи вентиляції.

Для їхнього демпфування застосовано:

- корпуси із низькошумними вентиляторами (≤ 25 дБА);
- гумові антивібраційні опори під системні блоки;
- шумогасні вставки в каналах вентсистеми.

Контроль здійснюють класичним шумоміром за методикою ГОСТ 12.1.003-83 у п'яти точках робочої зони. Перевищення понад 50 дБА допускається лише епізодично (не більше 15 хв на зміну); у цьому випадку працівнику надають додаткову перерву тривалістю 5 хв.Вимірювання шуму (системні блоки, кондиціонери, комутатори) показало середнє значення 44 дБА, що нижче граничних 50 дБА для IIб категорії робочих місць.

Вібрація (горизонтальна та вертикальна) від посадкового місця не перевищує 70 дБ до 80 Гц і вважається безпечною. ЕМП моніторів (0,05 мкТл при 50 Гц) у 20 разів нижча від норми 1 мкТл, але при щільному розміщенні ноутбуків використовують екрановані кабелі категорії 6А.

3.2.4 Мікроклімат

Робота програміста належить до категорії «1а» за ДСанПіН (мінімальні фізичні зусилля), тому нормативні значення такі: температура повітря 22–24 C взимку та 23–25 C у теплий період; відносна вологість 40–60 %; швидкість руху повітря ≤ 0,1 м/с.

Перевищення температури вище 28 C знижує швидкість психічних реакцій на 9–12 %, а вологість нижче 30 % збільшує ризик офтальмологічних захворювань, спричиняючи сухість очей та зниження працездатності. Вентиляційна система повинна забезпечувати кратність повітрообміну не менше ніж 60 м³/год на одне робоче місце, що дозволяє підтримувати якість повітря на безпечному рівні. У міжсезоння доцільно застосовувати локальні зволожувачі з фільтрами-адсорберами для регуляції вологості. Такі засоби покращують мікроклімат у приміщенні, сприяючи збереженню здоров'я працівників і підвищенню ефективності праці.

Система ОВіК включає приточно-витяжну установку 180 м³/год з рекуператором і касетними фільтрами класу G4 + F7. Вологість стабілізується ультразвуковими зволожувачами з автоматикою PID-контролю, що запобігає пересушуванню повітря та зменшує ризик офтальмологічних проблем.

3.2.5 Організація робочого місця

Кабель-менеджмент виконано в каналах під стільницею з радіусом згину не менше 4 см; це зменшує ризик пошкодження обплетення та спрощує прибирання.Для профілактики зорової втоми запроваджено цикл «50 хв роботи/10 хв відпочинку».

3.2.6 Електробезпека

Робоче місце має III клас небезпеки за ПУЕ. Живлення 230 В виконується по системі TN-S із виділеним захисним провідником PE. Основні заходи:

- Захист від ураження струмом — диференційні автомати типу A, IΔn = 30 мА, час спрацювання ≤ 40 мс.
- Вирівнювання потенціалів — всі металеві частини столів та стелажів приєднані до шини PE.
- Захист від перенапруг — комбіновані SPD T1+T2 зі струмом розряду 10 кА встановлені на вводі, локальні фільтри T3 на робочих розетках.
- Організація електромережі — групові лінії виконані кабелем NYM 3×2,5 мм² Cu у ПВХ-каналі, номінал автоматів — C-16 A. Запас по струму (коеф. завантаження 0,6) запобігає перегріву.

Інструктаж з електробезпеки проводиться щороку, журнали обліку — згідно з НПАОП 40.1-1.21-98. Засоби захисту — діелектричні килимки біля електрощитів, ізолюючі рукавички та вимірювальні кліщі — перевіряються повіркою кожні шість місяців.

Робоче місце належить до класу I за ПУЕ, електроприймачі — категорії III надійності, мережа 220 В — система TN-S із PE-провідником 2,5 мм² Cu, щит оснащено Δ-автоматами 30 мА для захисту від витоків; пожежну безпеку забезпечено автоматами C-16 A та термоіндикаторами, що зменшують ризик замикань і перегріву. ДСТУ EN ISO 9241-5 вимагає відстань «око-екран» 500–700 мм, кут погляду 15–20°, висоту столу 720–750 мм; крісла мають підтримку попереку, регулювання сидіння 420–530 мм і 3D-підлокітники, а 8-годинні зміни організовано за циклом 50+10 хв (НПАОП 0.00-1.28-10). Кондиціонери щомісяця чистять фільтри G-3 і дезінфікують 3%-м розчином перекису водню, тримаючи концентрацію мікроорганізмів

3.3 Пожежна безпека

У цьому підрозділі систематизовано організаційно-технічні заходи, що забезпечують належний рівень пожежної безпеки в офісному приміщенні розробника.

Приміщення призначене для експлуатації ПЕОМ і периферії, з горючим навантаженням від меблів (ДСП), корпусів системних блоків, кабелів і паперу. Сумарна пожежна навантага ≤ 400 МДж/м², що відповідає категорії В4 (спалимі

речовини у невеликій кількості). Клас зони – П-IIа за ПУЕ, підтверджений розрахунком за ДСТУ ISO/TR 13387-3:2002.

Первинні засоби пожежогасіння:

- Вогнегасники: Вуглекислотні ВВК-5 (2 шт.) для електроустаткування ≤ 1 кВ, порошковий ВП-6 (1 шт.) для тліючих матеріалів; відстань до вогнегасника ≤ 15 м, техобслуговування – раз на 12 міс. (НПАОП 0.00-5.13-12).
- Пожежний кран: Ду-50 з рукавом 20 м у коридорі (12 м від входу), тиск 0,6 МПа (норма ≥ 0,2 МПа), перевірка – щоквартально.
- Оснащення доповнено ВВК-2 і ВП-5; евакуаційний шлях відповідає ДБН В.1.1-7:2020 (ширина ≥ 1,2 м, протяжність ≤ 25 м), із реагуванням системи < 5 с при спрацьовуванні ≥ 1 детектора.

Шляхи евакуації та протипожежне планування:

- Основний вихід: двері 0,9 м, відчиняються назовні, евакуація < 30 с.
- Запасний вихід: коридор 1,2 м, без сходинок, з аварійним LED-освітленням 1 лк (UPS-2 кВА на 3 год).
- План евакуації (А3) вивішений біля входу, фотолюмінесцентні вказівники (≥ 3 мкд/м² через 120 хв), максимальна відстань до виходу – 15 м (норма ≤ 25 м).

Організаційні заходи та документація:

- Наказ призначає відповідального за ПБ, інструктажі (вступний, первинний, повторний) – щорічно.
- Інструкція ІН-ПБ-2025-03 переглядається раз на 3 роки.
- Журнал ПБ-6 веде відповідальна особа, контроль маси вогнегасників – щомісяця.
- Договір із ДСНС на обслуговування ППС і систем протидимного захисту діє до 2027 р.

ВИСНОВКИ

У рамках дипломного проєкту розроблено веб-платформу з багаторівневою системою підписок для керування навчальним відеоконтентом, що включає модулі реєстрації, автентифікації, адміністрування користувачів та інтеграцію з платіжною системою Stripe, забезпечуючи користувачам зручний доступ до матеріалів і адміністраторам ефективні інструменти для управління підписками та монетизації.

У процесі розробки проведено аналіз сучасних веб-технологій, порівняння підходів до створення платіжних систем і моделей доступу до контенту. Обґрунтовано вибір технологічного стеку: React з TypeScript для клієнтської частини, Django REST Framework з JWT-автентифікацією для серверної, MySQL як основна СУБД, Cloudinary для зберігання мультимедійних даних та Stripe API для обробки платежів. Реалізовано чотирирівневу архітектуру (презентація, сервіси, домен, персистенція), що забезпечує низьку зв’язність, високу підтримуваність і масштабованість системи.

Реалізовано компоненти для автоматизації підписок, верифікації платежів і оновлення статусів доступу, а також адміністративний інтерфейс StudioPage для керування користувачами, тарифами й підписками без прямого доступу до бази даних.

Проведено тестування системи, яке включало ручне тестування ключових функцій. Результати показали, що:

- користувачі можуть зручно оформлювати підписки, а доступ до контенту регулюється відповідно до активного тарифного плану;
- інтеграція зі Stripe забезпечує стабільну обробку платежів і автоматичне оновлення статусів підписок;
- адміністративна панель дозволяє швидко керувати користувачами, відображаючи зміни в реальному часі;
- система сповіщень коректно повідомляє користувачів про помилки або несанкціоновані дії.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Django REST Framework Official Documentation. – [Електронний ресурс]. – Режим доступу: https://www.django-rest-framework.org – Дата звернення: 12.03.2025.
2. Meta Platforms Inc. React Documentation. – [Електронний ресурс]. – Режим доступу: https://reactjs.org/ – Дата звернення: 13.03.2025.
3. Microsoft. TypeScript Handbook. – [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs – Дата звернення: 13.03.2025.
4. Oracle Corporation. MySQL Documentation. – [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/ – Дата звернення: 13.03.2025.
5. Cloudinary Inc. Офіційна документація Cloudinary. – [Електронний ресурс]. – Режим доступу: https://cloudinary.com/documentation – Дата звернення: 13.03.2025.
6. Stripe Inc. Stripe Developer Documentation. – [Електронний ресурс]. – Режим доступу: https://stripe.com/docs – Дата звернення: 13.03.2025.
7. Офіційна документація MUI (Material UI). – [Електронний ресурс]. – Режим доступу: https://mui.com – Дата звернення: 13.03.2025.
8. ISO/IEC 9126-1:2001. Software Engineering — Product quality — Part 1: Quality model.
9. ISO/IEC 27001:2022 – Information Security Management. – International Organization for Standardization.
10. Саммерфілд М. Python 3. Підручник програміста. – Харків: Фактор, 2021. – 600 с.
11. Фримен Е., Робсон Е. React. Сучасний підхід до розробки інтерфейсів. – Київ: Діалектика, 2023. – 472 с.
12. Миколайчук І. В. Основи веб-розробки з використанням Django. – Київ: Ліра-К, 2022. – 284 с.
13. Литвиненко Р. О. Frontend-розробка з React: навчальний посібник. – Харків: НТУ "ХПІ", 2022. – 312 с.
14. William S. Vincent. Django for Professionals. — WelcomeToCode, 2023.
15. Robin Wieruch. The Road to React — 2024.

ДОДАТОК А. Лістинг ключових компонентів StudioScreen, EditUserModal

```
// Основний компонент модального вікна редагування
const EditUserModal: React.FC<EditUserModalProps> = ({ open, onClose, user, onSave }) => {
  // Стан для полів форми
  const [email, setEmail] = useState("");
  const [username, setUsername] = useState("");
  const [displayName, setDisplayName] = useState("");
  const [role, setRole] = useState("");
  // Заповнення полів при відкритті модалки
  useEffect(() => {
    if (user) {
      setEmail(user.email);
      setUsername(user.username);
      setDisplayName(user.display_name || "");
      setRole(user.role);
    }
  }, [user]);
  // Обробка збереження
  const handleSubmit = () => {
    if (user) {
      onSave({
        id: user.id,
        email,
        username,
        role,
        display_name: displayName,
      });
      onClose();
    }
};
```

```
 return (
    // Модальне вікно
    <Modal open={open} onClose={onClose}>
      <Box sx={style}>
     <Typography variant="h6" sx={{ mb: 2 }}>Редагування користувача</Typography>
        {/* Поле Email */}
        <TextField
          fullWidth
          label="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          sx={{ mb: 2 }}
        />
        {/* Поле Username */}
        <TextField
          fullWidth
          label="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          sx={{ mb: 2 }}
        />
        {/* Поле Display Name */}
        <TextField
          fullWidth
          label="Display Name"
          value={displayName}
          onChange={(e) => setDisplayName(e.target.value)}
          sx={{ mb: 2 }}
        />
       {/* Випадаючий список ролей */}
```

```
        <FormControl fullWidth sx={{ mb: 2 }}>
          <InputLabel>Роль</InputLabel>
        <Select value={role} label="Роль" onChange={(e) => setRole(e.target.value)}>
              <MenuItem value="user">user</MenuItem>
              <MenuItem value="admin">admin</MenuItem>
            </Select>
          </FormControl>
          {/* Кнопки управління */}
          <Box sx={{ display: "flex", justifyContent: "flex-end", gap: 1 }}>
            <Button variant="outlined" onClick={onClose}>Скасувати</Button>
            <Button variant="contained" onClick={handleSubmit}>Зберегти</Button>
          </Box>
        </Box>
      </Modal>
    );
};
const StudioPage: React.FC = () => {
  // Стан для користувачів, змін, модалки
  const [users, setUsers] = useState<User[]>([]);
  const [edited, setEdited] = useState<Record<number, string>>({});
  const [editUserModalOpen, setEditUserModalOpen] = useState(false);
  const [selectedUser, setSelectedUser] = useState<User | null>(null);
  // Завантаження користувачів з API
  const fetchUsers = async () => {
    const res = await
authFetch("https://codecaveback2.onrender.com/api/users/");
    if (!res || !res.ok) return;
    const data = await res.json();
    setUsers(data);
  };
  // Збереження нової підписки
  const handleSaveSubscription = async (id: number) => {
    const subName = edited[id];
    const subData = subscriptionOptions.find((s) => s.name === subName);
    if (!subData) return;
    const res = await
authFetch(`https://codecaveback2.onrender.com/api/users/${id}/`, {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        subscription_name: subName,
        subscription_price: subData.price,
        subscription_status: "active",
      }),
    });
    if (res && res.ok) {
    const updatedUser = await res.json();
      // Оновлення currentUser у localStorage
   const currentUser = JSON.parse(localStorage.getItem("currentUser") || "{}");
      if (currentUser.id === id) {
        localStorage.setItem("currentUser", JSON.stringify(updatedUser));
      }
      // Очистка змін
      setEdited((prev) => {
        const clone = { ...prev };
        delete clone[id];
        return clone;
      });
      fetchUsers();
    } else {
      console.warn("❌ PATCH failed", res?.status);
    }
  };
  // Збереження оновленого користувача
```

```
  const handleSaveUser = async (updated: {
    id: number;
    email: string;
    username: string;
    role: string;
    display_name: string;
  }) => {
    const res = await
authFetch(`https://codecaveback2.onrender.com/api/users/${updated.id}/`, {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        email: updated.email,
        username: updated.username,
        role: updated.role,
        display_name: updated.display_name,
      }),
    });
    if (res && res.ok) {
      const updatedUser = await res.json();
   const currentUser = JSON.parse(localStorage.getItem("currentUser") || "{}");
      if (currentUser.id === updatedUser.id) {
        localStorage.setItem("currentUser", JSON.stringify(updatedUser));
      }
      fetchUsers();
    }
  };
  // Скасування підписки
  const handleCancelSub = async (id: number) => {
    const res = await
authFetch(`https://codecaveback2.onrender.com/api/admin/cancel-
subscription/${id}/`, {
      method: "POST"
    });
    if (res && res.ok) {
      fetchUsers();
    } else {
      console.error("Не вдалось відмінити підписку:", res?.status);
    }
  };
  // Повторна активація підписки
  const handleReactivate = async (id: number) => {
    const res = await
authFetch(`https://codecaveback2.onrender.com/api/users/${id}/`, {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ subscription_status: "active" }),
    });
    if (res && res.ok) {
      fetchUsers();
    }
  };
  // Видалення користувача
  const handleDeleteUser = async (id: number) => {const res = await
authFetch(`https://codecaveback2.onrender.com/api/users/${id}/`, {
      method: "DELETE",
    });
    if (res && res.ok) {
      fetchUsers();
    }
  };
  // Завантаження при монтуванні компонента
  useEffect(() => {
    fetchUsers();
  }, []);
```

```
return (
  <Box sx={{ p: 4 }}>
    <Typography variant="h4" gutterBottom>
      Студія — Керування користувачами
    </Typography>
    {/* Таблиця користувачів */}
    <Table sx={{ mt: 2 }}>
      <TableHead>
        <TableRow>
          <TableCell>Email</TableCell>
          <TableCell>Username</TableCell>
          <TableCell>Роль</TableCell>
          <TableCell>Підписка</TableCell>
          <TableCell>Статус</TableCell>
          <TableCell>Ціна</TableCell>
          <TableCell>Баланс (€)</TableCell>
          <TableCell>Дії</TableCell>
        </TableRow>
      </TableHead>
      <TableBody>
        {users.map((u) => (
          <TableRow key={u.id}>
            <TableCell>{u.email}</TableCell>
            <TableCell>{u.username}</TableCell>
            <TableCell>{u.role}</TableCell>
            {/* Зміна підписки */}
            <TableCell>
              <FormControl size="small" fullWidth>
                <Select
                  value={edited[u.id] ?? u.subscription_name}
                  onChange={(e) =>
                    setEdited((prev) => ({
                      ...prev,
                      [u.id]: e.target.value,
                    }))
                  }
                >
                  {subscriptionOptions.map((opt) => (
                    <MenuItem key={opt.name} value={opt.name}>
                      {opt.name}
                    </MenuItem>
                  ))}
                </Select>
              </FormControl>
            </TableCell>
            <TableCell>{u.subscription_status}</TableCell>
            <TableCell>{u.subscription_price}</TableCell>
            <TableCell>{u.balance}</TableCell>
            <TableCell>
              {/* Зберегти тариф */}
              {edited[u.id] && (
                <Button
                  size="small"
                  variant="contained"
                  onClick={() => handleSaveSubscription(u.id)}
                  sx={{
                    mb: 1, mr: 1,
                    "&:focus": { outline: "none", boxShadow: "none" },
                  }}
                >
                  Зберегти
                </Button>
              )}
              {/* Редагування */}
              <IconButton
```

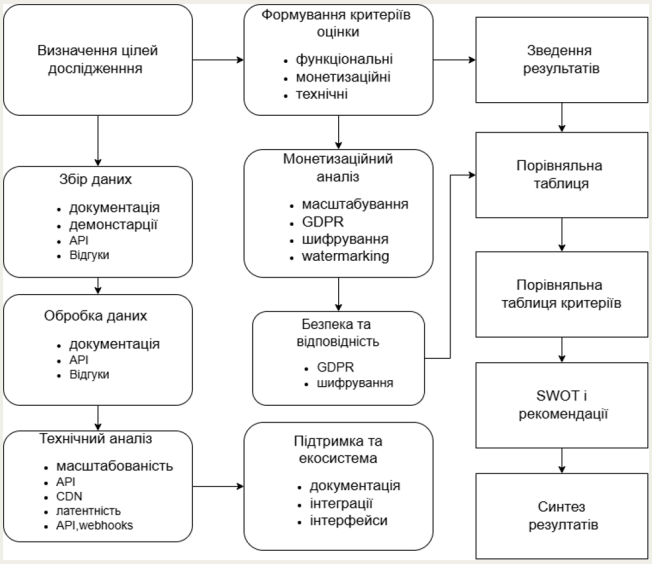
```
                  onClick={() => {
                    setSelectedUser(u);
                    setEditUserModalOpen(true);
                  }}
                  sx={{
                    mb: 1, mr: 1,
                    color: "#f7266e",
                    "&:focus": { outline: "none", boxShadow: "none" },
                  }}
                  title="Редагувати користувача"
                >
                  <EditIcon />
                </IconButton>
                {/* Реактивація */}
                {u.subscription_status === "canceled" && (
                  <IconButton
                    onClick={() => handleReactivate(u.id)}
                    sx={{
                      mb: 1, mr: 1,
                      "&:focus": { outline: "none", boxShadow: "none" },
                    }}
                    title="Активувати знову"
                  >
                    <ReplayIcon />
                  </IconButton>
                )}
                {/* Скасувати підписку */}
                <IconButton
                  onClick={() => handleCancelSub(u.id)}
                  sx={{
                    mb: 1, mr: 1,
                    "&:focus": { outline: "none", boxShadow: "none" },
                  }}
                  title="Скасувати підписку"
                >
                  <CancelIcon />
                </IconButton>

                {/* Видалити користувача */}
                <IconButton
                  onClick={() => handleDeleteUser(u.id)}
                  title="Видалити користувача"
                  sx={{
                    mb: 1, mr: 1,
                    "&:focus": { outline: "none", boxShadow: "none" },
                  }}
                >
                  <DeleteIcon />
                </IconButton>
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
      {/* Модалка редагування */}
      <EditUserModal
        open={editUserModalOpen}
        onClose={() => setEditUserModalOpen(false)}
        user={selectedUser}
        onSave={handleSaveUser}
      />
    </Box>
  );
};
export default StudioPage;
```

ДОДАТОК Б. Слайди мультимедійної презентації

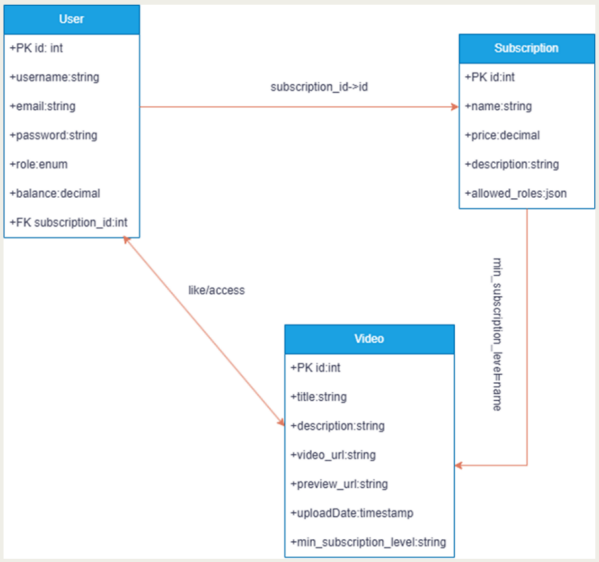
Розробка програмного забезпечення для керування підписками користувачів

Пятніченко Дмитрій 4РП-08

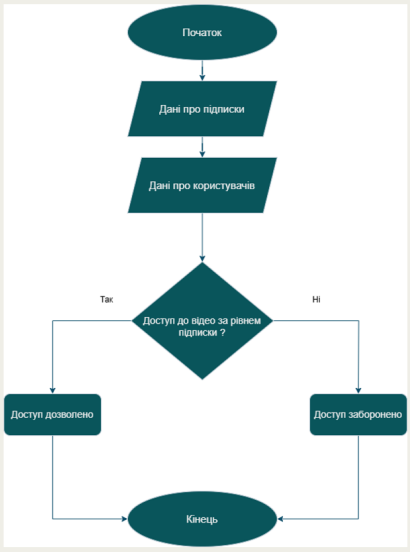
Методологічна схема порівняльного аналізу платформ відеохостингу



ER-діаграма бази даних користувачів, відео та підписок

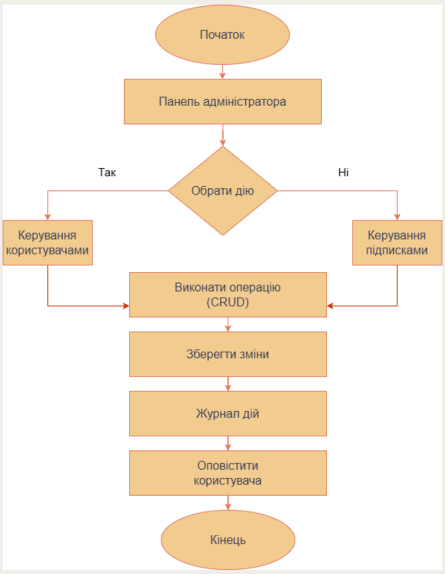


Блок-схема перевірки доступу до відеоконтенту за підпискою

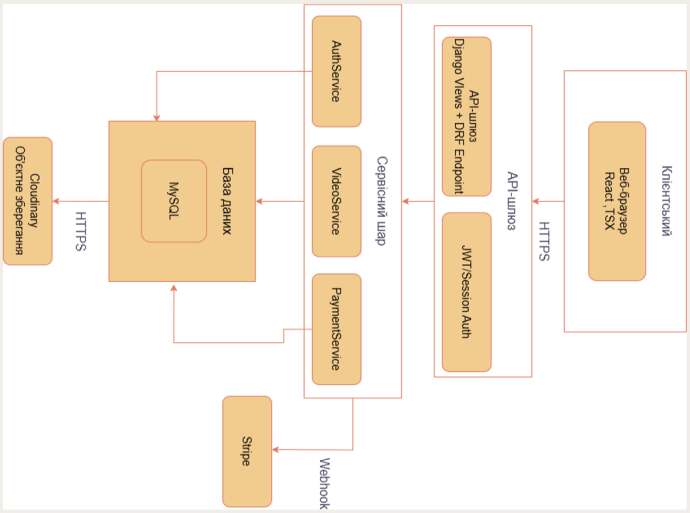


Блок-схема алгоритму моніторингу прогресу перегляду відео

Алгоритм роботи панелі керування користувачами та підписками

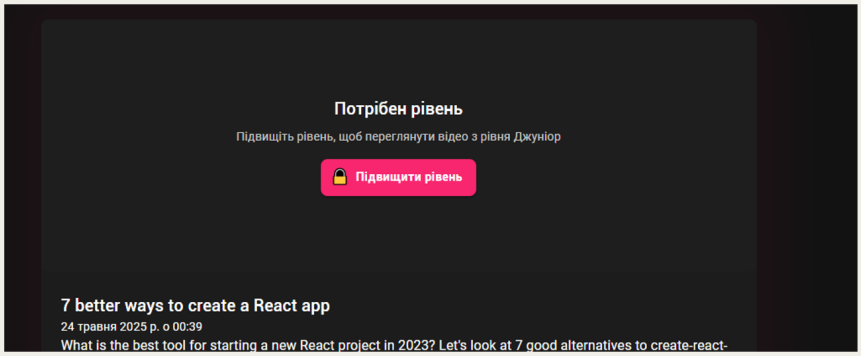


Алгоритм роботи панелі керування користувачами та підписками

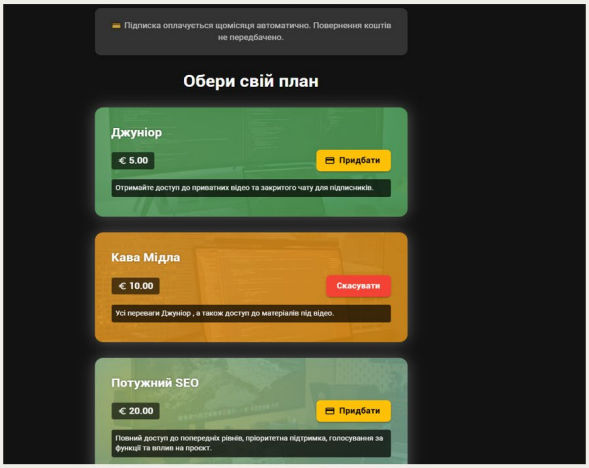


Блок-схема процесу оформлення та обробки підписки через Stripe

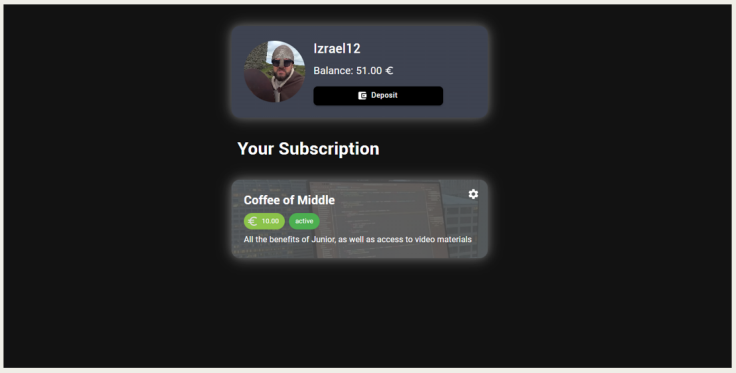
Інтерфейс відеоплеєра з обмеженим доступом



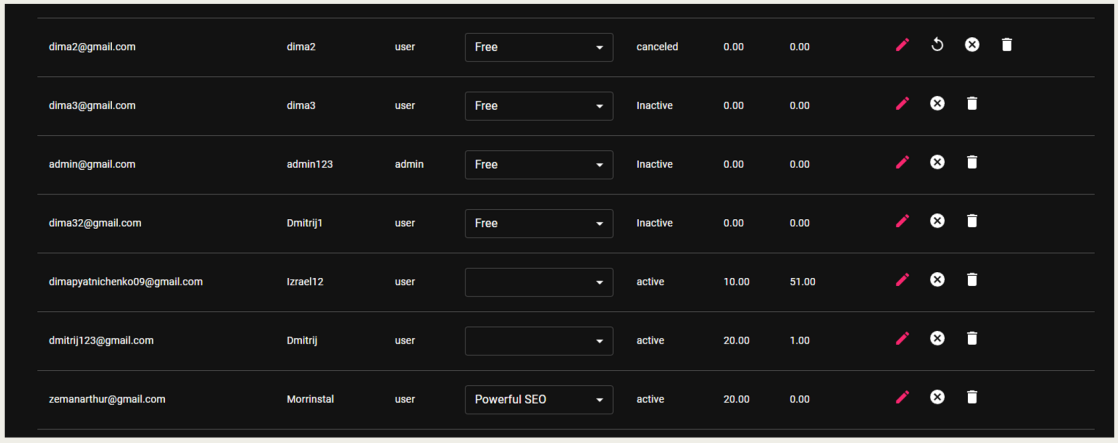
Вибір тарифного плану підписки



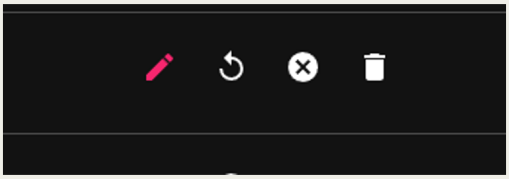
Сторінка профілю користувача з активною підпискою



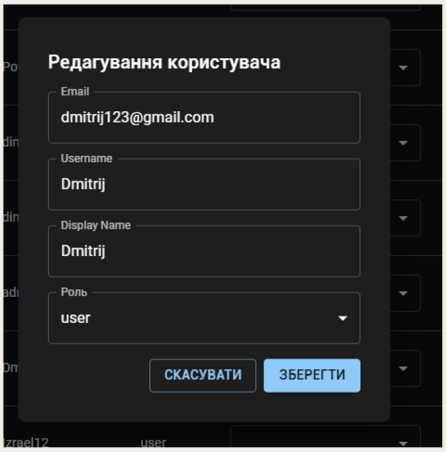
Фрагмент таблиці з можливістю інлайнового редагування тарифу



Елементи керування: редагування, скасування, активація, видалення



Модальне вікно редагування атрибутів користувача



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Пятніченко Дмитрій Олександрович

(прізвище, ім`я та по батькові)

Спеціальність *121 " Інженерія програмного забезпечення "*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Залапін Олексій Ігорович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка програмного забезпечення для керування підписками користувачів*

Обсяг розрахунково-пояснювальної записки *88* сторінок

Обсяг графічної (презентаційної) частини *14* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню *Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення продукту для керування підписками користувачів у web-інтерфейсі.*

б) характеристика виконання кожного розділу дипломного проекту (роботи) *Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті питання проблематики розробки програмного забезпечення мовою засобами Django та React, сформовано інтеграцію з СУБД MySQL згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляємої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені роботи з ігровим процесом.*

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) *Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено*

г) перелік позитивних якостей дипломного проекту (роботи)

Детально описано процес виконання розробки програмного забезпечення;

Виконано проектування елементів web-серверу з поясненнями на схемах та за допомогою коду;

Для інтеграції фронтенду та бекенду було використано REST API.

д) основні недоліки дипломного проекту (роботи)

Підтримка ролей, що була забезпечена, є дуже простою.

Мультимедійні дані, що отримує користувач дуже обмежені.

Недостатньо деталізовано описано, як вирішуються питання можливих затримок або розбіжностей між зовнішніми подіями (webhook Stripe) та локальними оновленнями бази даних.

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Рудніченко Микола Дмитрович*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис

« *20* » *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Пятніченко Дмитрія Олександровича

(прізвище, ім`я та по батькові)

Спеціальність: *121 "Комп'ютерна інженерія"*

Освітньо-професійна програма: *«Розробка програмного забезпечення»*

Тема дипломного проекту: *Розробка програмного забезпечення для керування підписками користувачів стану*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 88 сторінки. У пояснювальній записці наведено етапи розробки програмного забезпечення для керування підписками користувачів. Графічна частина складається з 14 слайдів мультимедійної презентації, які також містять схеми, ER-діаграми та скріншоти веб-інтерфейсу, що передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.*

б) самостійність роботи над проектом: *Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Пятніченко Д.О. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Пятніченко Д.О. під час роботи над дипломним проектом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою. Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проекту*

г) вміння розв'язувати виробничі та конструкторські питання *Під час дипломного проектування здобувач освіти Пятніченко Д.О. мав змогу самостійно приймати окремі рішення з розробки програмного забезпечення для керування підписками користувачів ПК та показав вміння організовано працювати над поставленим завданням, розробляти структурні схеми та програмні зв'язки із застосуванням сучасних комп'ютерних програмних засобів, таких як Visual Studio Code, HTML, CSS та JavaScript, а також готувати презентаційні та звітні матеріали.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту *Залапін Олексій Ігорович*

Місце роботи і посада керівника дипломного проекту *ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії.*

Підпис

«16» червня 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Пятніченко Дмитрій Олександрович
здобувач освіти гр. 4РП-08, та

Залапін Олексій Ігорович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка програмного забезпечення для керування підписками користувачів» (автор роботи – Пятніченко Д.О., керівник роботи – Залапін О.І.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Пятніченко Д.О. /

Керівник _____ / Залапін О.І. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Пятніченка Дмитра Олександровича

на тему *Розробка програмного забезпечення для керування підписками користувачів*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з некритичними порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 15.06.2025 р. значення коефіцієнту подібності в роботі становить 4,14%, коефіцієнт цитування – 0,92%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Пятніченка Д.О.* (П.І.Б.)

проведена «*16*» *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/15/2025
Дата редагування 6/15/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка програмного забезпечення для керування підписками користувачів

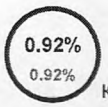
Автор
Пятніченко Дмитрій Олександрович
Науковий керівник / Експерт
Залапін Олексій Ігорович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

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.

КП 1



КЦ

25	13977	111463
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		28
Інтервали		2
Мікропробіли		2
Білі знаки		0
Парафрази (SmartMarks)	a	29

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	51 0.36 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	34 0.24 %
3	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	32 0.23 %
4	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	28 0.20 %
5	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	25 0.18 %

1	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**188 (12) 1.35 %**
2	**https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download**	**69 (3) 0.49 %**
3	**https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download**	**62 (3) 0.44 %**
4	**https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download**	**49 (2) 0.35 %**
5	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	37 (4) 0.26 %
6	https://bibliotekanauki.pl/articles/208960.pdf	17 (1) 0.12 %
7	https://zp.edu.ua/sites/default/files/konf/met-vkaz_dr_fret_125.pdf	14 (1) 0.10 %
8	https://card-file.ontu.edu.ua/server/api/core/bitstreams/c63b91ba-d04f-4715-890d-b16277695c7e/content	14 (1) 0.10 %
9	https://redstone.media/intehratsiia-khmarnykh-servisiv-u-vashi-veb-dodatky-mozhlyvosti-ta-perevahy	12 (1) 0.09 %
10	https://card-file.ontu.edu.ua/bitstreams/907a4053-091b-44b2-bcae-b4601346673b/download	11 (1) 0.08 %
11	https://card-file.ontu.edu.ua/bitstreams/8f088d70-9465-490c-8fa6-2eb74516c620/download	10 (1) 0.07 %
12	http://lib.pnu.edu.ua:8080/bitstream/123456789/8063/1/ATSU2020_kuz_kozlenko.pdf	5 (1) 0.04 %
13	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	5 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121«Інженерія програмного забезпечення»
Освітньо-професійна програма: «Розробка **програмного забезпечення»**
Група: 4 **РП-** 08

Дипломний проект
здобувач освіти денної форми навчання
РП.08.17.000.ДП

ПЯТНІЧЕНКО
ДМИТРІЙ ОЛЕКСАНДРОВИЧ

м. Одеса
2025 р. МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення_____Комісія_____
Спеціальність_____
Освітньо-професійна **програма**_____

ЗАТВЕРДЖУЮ:
Заст. дир. з НВР_____
_____
"_____"_____ **2025 р.**

ЗАВДАННЯ
на дипломний проект
Здобувачеві освіти _____ **(прізвище, ім'я, по батькові) 1. Тема** проекту_____
_____
_____
затверджена **наказом по коледжу від** "____"_____202___**р. No**_____ **2. Термін здачі**

6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	24 0.17 %
7	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	21 0.15 %
8	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	21 0.15 %
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	20 0.14 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	19 0.14 %

з домашньої бази даних (0.34 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка програмного забезпечення для відстеження ступеня готовності виконання проекту [redacted] 6/15/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	31 (4) 0.22 %
2	Розробка системи авторизації користувача на web-сервері за допомогою nrf-модулю [redacted] 6/15/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	6 (1) 0.04 %
3	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	6 (1) 0.04 %
4	Розробка web-застосунку для генерації повідомлень із використанням технологій штучного інтелекту [redacted] 6/14/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	5 (1) 0.04 %

з програми обміну базами даних (0.26 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	2024_АПЗ_ПЗПІ_21_4_Павленко_П_О_скорочений [redacted] 6/4/2024 Kharkiv National University of Radio Electronics (Харківський національний університет радіоелектроніки)	14 (1) 0.10 %
2	Hryhorenko_Yesina_Diplom_2025 [redacted] 6/4/2025 V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра кібербезпеки інформаційних систем, мереж і технологій)	13 (1) 0.09 %
3	БКР Явір Р. КН-420 [redacted] 6/8/2025 Educational and Research Institute of Spatial Planning and Advanced Technologies of Lviv Polytechnic National University (Educational and Research Institute of Spatial Planning and Advanced Technologies of Lviv Polytechnic National University)	10 (1) 0.07 %

з Інтернету (3.53 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)