МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН ЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ КІНО З ВИКОРИСТАННЯМ REACT, NESTJS, POSTGRESQL»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.С. Глоба
(ініціали та прізвище)
Керівник викладач кафедри програмної інженерії, PhD Климчук А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Глобі Артему Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн застосунку для перегляду кіно з використанням React, Nestjs, PostgreSQL

керівник роботи Климчук Анастасія Олександрівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування архітектури онлайн-застосунку.
3. Програмна реалізація компонентів системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
		,	

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	12.01.2026	
3.	Обробка методичних та теоретичних	19.01.2026	
	джерел.		
4.	Розробка першого та другого розділу.	09.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ А.С. Глоба
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Климчук
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн застосунку для перегляду кіно з використанням React, Nestjs, PostgreSQL»: 50 с., 22 рис., 1 табл., 11 джерел.

АВТОРИЗАЦІЯ, БАЗА ДАНИХ, ВІДГУКИ, КІНО, ОНЛАЙН-ЗАСТОСУНОК, ПЛАТЕЖІ, ФІЛЬТРАЦІЯ, API, NESTJS, POSTGRESQL, REACT, STRIPE.

Об’єкт дослідження – процеси програмної інженерії (визначення вимог, проєктування, створення та забезпечення якості) при розробленні клієнт-серверного ПЗ медіаплатформ стрімінгового відео.

Мета роботи – підвищення модульності, масштабованості та надійності архітектури ПЗ онлайн-кінотеатрів, а також зменшення часу відгуку системи й скорочення витрат серверних ресурсів на керування даними шляхом розроблення відмововитривалого вебзастосунку «StreamWave» на базі стеку React, NestJS та СУБД PostgreSQL.

Методи дослідження – аналіз предметної області, об’єктно-орієнтоване проєктування (UML), моделювання бізнес-процесів (BPMN, EPC), триланкова архітектура, проєктування реляційних БД (ERD), розроблення REST API, декларативна валідація.

У роботі сформульовано функціональні та нефункціональні вимоги до архітектури ПЗ. Обґрунтовано вибір бібліотеки React для фронтенду, фреймворку NestJS для серверного API та СУБД PostgreSQL для збереження взаємозв’язаних сутностей. Розроблено комплекс UML, BPMN та EPC діаграм, що формалізують динаміку системи, а також схему БД, яка відповідає вимогам третьої нормальної форми.

SUMMARY

Bachelor's qualifying paper "Development of an Online Application for Movie Streaming Using React, NestJS, and PostgreSQL": 50 pages, 22 figures, 1 table, 11 references.

AUTHORIZATION, DATABASE, REVIEWS, CINEMA, ONLINE APPLICATION, PAYMENTS, FILTERING, API, NESTJS, POSTGRESQL, REACT, STRIPE.

The object of research is the software engineering processes (requirements definition, design, creation, and quality assurance) in the development of client-server software for video streaming media platforms.

The objective of the thesis is to increase the modularity, scalability, and reliability of online cinema software architecture, as well as to reduce system response time and cut server resource costs for data management by developing a fault-tolerant web application «StreamWave» based on the React, NestJS, and PostgreSQL DBMS stack.

The method of research includes domain analysis, object-oriented design (UML), business process modeling (BPMN, EPC), 3-tier architecture, relational database design (ERD), REST API development, and declarative validation.

The thesis formulates functional and non-functional requirements for the software architecture. The choice of the React library for the frontend, the NestJS framework for the server API, and the PostgreSQL DBMS for storing interconnected entities is justified. A set of UML, BPMN, and EPC diagrams formalizing the system dynamics has been developed, along with a database schema that meets the requirements of the third normal form.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та постановка задачі 9
1.1 Аналіз онлайн-застосунку для перегляду кіно 9
1.2 Постановка задачі та вимоги до застосунку..... 10
1.3 Обґрунтування вибору технологій..... 12
2 Проєктування архітектури онлайн-застосунку 14
2.1 Архітектурне проєктування та структурне моделювання системи ... 14
2.2 Поведінкове моделювання та інтеракція об’єктів 16
2.3 Проєктування бази даних 22
3 Програмна реалізація компонентів системи 30
3.1 Обґрунтування кодової бази та програмна архітектура застосунку.. 30
3.2 Програмна реалізація серверних сервісів та REST API..... 32
3.3 Валідація вхідних даних та обробка виняткових ситуацій..... 40
3.4 Розробка клієнтського інтерфейсу користувача..... 44
Висновки 48
Перелік посилань..... 50

ВСТУП

Розвиток вебтехнологій змінив спосіб доступу користувачів до відеоконтенту. Якщо раніше перегляд кіно переважно був пов’язаний з локальним зберіганням файлів або телевізійним ефіром, то сучасні онлайн-сервіси дають змогу отримувати доступ до каталогу фільмів через браузер, швидко здійснювати пошук, обирати фільми за жанрами, рейтингами та іншими параметрами, а також купувати доступ до окремого контенту. Для такого сервісу важливо поєднати зручний інтерфейс, стабільну серверну частину, надійну базу даних та безпечну роботу з оплатою.

Тема кваліфікаційної роботи пов’язана з розробкою онлайн-застосунку для перегляду кіно з використанням React, NestJS та PostgreSQL. Обрана тема є актуальною, оскільки клієнт-серверні вебзастосунки залишаються основою більшості сучасних сервісів, а розробка відеосервісу охоплює одразу кілька важливих напрямів програмної інженерії: побудову інтерфейсу користувача, створення REST API, проєктування бази даних, роботу з авторизацією, інтеграцію платіжної системи, обробку медіафайлів і тестування основних сценаріїв.

У межах роботи розроблено готовий застосунок «StreamWave», який реалізує функції перегляду каталогу фільмів, реєстрації та входу користувачів, фільтрації контенту, перегляду детальної інформації про фільм, створення відгуків, формування рейтингу, купівлі фільмів через Stripe та збереження інформації про транзакції. Клієнтська частина побудована як односторінковий застосунок на React і TypeScript, серверна частина реалізована на NestJS, а дані зберігаються у PostgreSQL через TypeORM.

Метою роботи є розробка онлайн-застосунку для перегляду кіно, опис його архітектури, функціональних можливостей, структури даних, програмної реалізації та підходу до тестування.

Для досягнення мети необхідно:

– проаналізувати предметну область;
– сформувати вимоги, обґрунтувати вибір технологій;
– спроєктувати клієнт-серверну архітектуру;
– визначити структуру бази даних;
– описати реалізацію клієнтської і серверної частин;
– перевірити роботу основних сценаріїв користувача.

Об’єктом дослідження є процес розробки онлайн-застосунку для перегляду кіно. Предметом дослідження є програмні засоби та архітектурні рішення, які забезпечують роботу клієнтської частини на React, серверної частини на NestJS і реляційної бази даних PostgreSQL. Практична цінність роботи полягає у створенні готового застосунку, який може бути використаний як навчальний або демонстраційний проєкт для подальшого розвитку відеосервісу.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. У першому розділі розглянуто предметну область, вимоги та технології. У другому розділі описано архітектуру, базу даних, API та інтерфейс. У третьому розділі наведено реалізацію основних модулів, підхід до тестування та запуску застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз онлайн-застосунку для перегляду кіно

Онлайн-застосунок для перегляду кіно є вебсистемою, у якій користувач отримує доступ до структурованого каталогу відеоконтенту. З погляду користувача такий сервіс повинен забезпечувати швидкий перегляд списку фільмів, пошук потрібної назви, фільтрацію за характеристиками, відкриття сторінки фільму, перегляд опису, оцінки, тривалості, вікового обмеження та іншої інформації. Якщо частина контенту є платною, система повинна відокремлювати безкоштовні та платні фільми і надавати доступ до платного контенту лише після успішної купівлі.

З погляду програмної реалізації сервіс складається з клієнтської частини, серверної частини та бази даних. Клієнтська частина відповідає за взаємодію з користувачем: відображення сторінок, форм, модальних вікон, повідомлень, списків фільмів і результатів фільтрації. Серверна частина опрацьовує запити, перевіряє дані, виконує бізнес-логіку, працює з базою даних і забезпечує інтеграцію із зовнішніми сервісами. База даних зберігає користувачів, фільми, жанри, акторів, відгуки, покупки та платіжні транзакції.

Ключова особливість відеосервісу полягає у великій кількості взаємопов’язаних даних. Фільм може належати до кількох жанрів, бути пов’язаним з акторами, мати відгуки користувачів, рейтинг, вікове обмеження, вартість та шляхи до медіафайлів. Користувач може мати профіль, історію покупок, власні відгуки та аватар. Тому для реалізації такої системи доцільно використовувати реляційну базу даних, яка підтримує зв’язки між сутностями і дозволяє будувати контрольовані запити до даних.

У готовому застосунку «StreamWave» передбачено сценарій роботи з каталогом фільмів, реєстрацією та входом, фільтрацією за жанром, актором, рейтингом, віковим обмеженням, платністю, роком випуску та пошуковим рядком. Також реалізовано сценарії перегляду детальної сторінки фільму,

створення та редагування відгуку, перерахунку рейтингу, купівлі фільму, збереження транзакції та відображення інформації в профілі користувача. Ці функції відповідають типовому набору можливостей онлайн-кінотеатру навчального рівня.

Окреме місце займає віковий контроль. Для сервісу, що містить фільми з різними віковими обмеженнями, важливо не лише показати позначку вікового рейтингу, а й обмежити виведення контенту відповідно до зазначеного віку користувача. У застосунку такий сценарій реалізовано через модальне вікно введення віку та збереження значення у базу даних для конкретного користувача. Після цього фільтр не дозволяє обрати вікове обмеження вище за збережене значення.

Ще одним важливим аспектом є платіжна логіка. Для платного контенту застосунок повинен відокремити створення платіжного наміру на сервері від підтвердження платежу на клієнті. Такий підхід дозволяє не передавати секретний ключ платіжної системи у браузер і зберігати інформацію про транзакції на сервері. У проєкті для цього використовується Stripe: сервер створює PaymentIntent, клієнт підтверджує оплату карткою, а після успіху зберігається транзакція і створюється запис про покупку.

Таким чином, предметна область поєднує задачі каталогізації контенту, керування користувачами, перевірки доступу, фільтрації, платіжної інтеграції, роботи з відгуками та зберігання медіаданих.

1.2 Постановка задачі та вимоги до застосунку

Постановка задачі полягає у створенні онлайн-застосунку для перегляду кіно, який забезпечує взаємодію користувача з каталогом фільмів та основними сервісними функціями. Застосунок повинен працювати через браузер, мати клієнтську частину з маршрутизацією між сторінками, серверний API для доступу до даних і базу даних для збереження інформації.

Особливістю задачі є необхідність узгодити інтерфейс, серверну логіку та структуру даних так, щоб усі основні сценарії були пов’язані між собою.

Головною функціональною вимогою є перегляд каталогу фільмів. Користувач повинен бачити список фільмів, мати можливість шукати фільми за назвою, фільтрувати їх за жанром, актором, рейтингом, віковим обмеженням, платністю та роком випуску. Список повинен підтримувати посторінкове завантаження, оскільки каталог може містити велику кількість записів. Сервер має приймати параметри фільтрації та повертати структуровану відповідь із даними та інформацією для пагінації.

Другою вимогою є робота з обліковими записами. Користувач повинен мати можливість зареєструватися, увійти в систему, переглядати та оновлювати профіль. Для реєстрації необхідно перевіряти коректність email, пароля та імені користувача. На сервері потрібно перевіряти унікальність email і захищати пароль шляхом хешування. Для авторизованих дій застосунок повинен ідентифікувати користувача, оскільки відгуки, покупки та профіль прив’язані до конкретного облікового запису.

Третьою вимогою є підтримка відгуків. Авторизований користувач повинен мати можливість залишити оцінку та коментар до фільму, оновити або видалити власний відгук. Після кожної зміни відгуку середній рейтинг фільму має перераховуватися на сервері, щоб список фільмів і детальна сторінка відображали актуальну оцінку. Важливо, що редагування та видалення повинні бути дозволені лише автору відгуку.

Четвертою вимогою є інтеграція платежів. Для платних фільмів користувач має отримати доступ після успішної оплати. Сервер повинен створювати платіжний намір, клієнт повинен підтверджувати оплату карткою, а після успішного платежу має створюватися запис про транзакцію і покупку. Додатково передбачено формування PDF-чека, що підвищує завершеність платіжного сценарію.

П’ятою вимогою є підтримка вікового контролю. Застосунок повинен зберігати вік користувача у браузері та використовувати його для обмеження фільтра.

Нефункціональні вимоги включають зрозумілий інтерфейс, адаптивність до різних екранів, розділення відповідальності між клієнтом і сервером, модульність серверного коду, валідацію вхідних даних, контроль помилок, можливість локального запуску через npm-скрипти та використання міграцій бази даних. Також важливими є безпечне зберігання конфігурацій через змінні середовища та відсутність секретних ключів у клієнтському коді.

1.3 Обґрунтування вибору технологій

Для реалізації клієнтської частини обрано React, оскільки він дозволяє будувати інтерфейс із повторно використовуваних компонентів і добре підходить для односторінкових застосунків. У проєкті React [1] використовується разом із TypeScript [3], що дає змогу описувати типи даних, зменшувати кількість помилок під час розробки та спростити супровід коду. Для локального запуску і збірки використано Vite [2], який забезпечує швидкий старт.

Для маршрутизації між сторінками використано React Router DOM [9]. Це дає змогу організувати переходи між головною сторінкою, сторінкою фільму, сторінками реєстрації та входу, профілем і платіжним сценарієм без перезавантаження всієї сторінки. Для форм застосовано Formik [10], а для їх валідації – Yup [11]. Така комбінація дозволяє відокремити опис форми, правила перевірки та реакцію на помилки користувача.

Серверна частина побудована на NestJS [4]. Цей фреймворк підтримує модульну структуру, контролери, сервіси, dependency injection і добре підходить для створення REST API. У проєкті виділено модулі авторизації, користувачів, фільмів, акторів, жанрів, платежів і роботи з S3. Завдяки модульності кожна група функцій має окрему відповідальність, що спрощує розуміння та подальше розширення коду.

Для роботи з базою даних використано PostgreSQL [5] і TypeORM [6]. PostgreSQL є реляційною системою керування базами даних, яка підходить

для збереження пов’язаних сутностей. TypeORM дозволяє описувати структуру таблиць через сутності TypeScript, будувати зв’язки між ними, виконувати міграції та працювати з репозиторіями. Для даного застосунку це зручно, оскільки сутності Movie, User, Review і PaymentTransaction безпосередньо відповідають логіці предметної області.

Для платежів обрано Stripe [7], оскільки у проєкті потрібен сценарій створення платіжного наміру на сервері та підтвердження карткової оплати на клієнті. Клієнтська частина використовує @stripe/react-stripe-js і CardElement, серверна – пакет stripe. Секретний ключ зберігається лише на сервері, а клієнт отримує clientSecret для підтвердження платежу.

Для роботи з медіафайлами передбачено використання AWS SDK S3 [8]. У серверному package.json є залежність @aws-sdk/client-s3. Це рішення дозволяє відокремити зберігання файлів від бази даних: у таблицях зберігаються шляхи до постерів, прев’ю та відео, а безпосередньо медіафайли можуть розміщуватися в об’єктному сховищі.

Таким чином, стек React + NestJS + PostgreSQL відповідає задачі розробки повного вебзастосунку. React забезпечує інтерактивний інтерфейс, NestJS – структуровану серверну логіку, PostgreSQL – надійне зберігання даних, а додаткові бібліотеки розширюють функціональність застосунку відповідно до вимог проєкту.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ОНЛАЙН-ЗАСТОСУНКУ

Другий розділ присвячено аналітичному, архітектурному та структурному проєктуванню онлайн-застосунку «StreamWave». На цьому етапі виконується системна декомпозиція вимог до вебплатформи за допомогою методології об’єктно-орієнтованого моделювання. У межах розділу обґрунтовано вибір структурних елементів системи, розроблено діаграми прецедентів та поведінкових інтеракцій об’єктів (UML-моделі), а також спроєктовано схему збереження даних і взаємозв’язків сутностей для інтеграції з реляційною базою даних PostgreSQL.

Системне проєктування є обов’язковим інженерним кроком, який дозволяє зафіксувати інтерфейсні контракти, мінімізувати зв’язність між модулями та сформувати чітку архітектурну базу перед безпосередньою програмною реалізацією сервісу.

2.1 Архітектурне проєктування та структурне моделювання системи

Для забезпечення високої модульності, масштабованості та низької зв’язності компонентів вебзастосунку «StreamWave» на етапі проєктування було розроблено серію структурних діаграм Unified Modeling Language (UML).

Першим етапом декомпозиції вимог є побудова діаграми прецедентів (Use Case Diagram), яка визначає межі системи та взаємодію актора з ключовими сервісними функціями.

Головним актором системи виступає користувач (User). Взаємодія клієнта з онлайн-кінотеатром охоплює три магістральні сценарії: «Перегляд кіно» (базовий функціонал), «Написання коментарів» (інструмент зворотного зв’язку) та «Купівля платного фільму» (транзакційна бізнес-логіка), детально продемонстровано на рисунку 2.1.

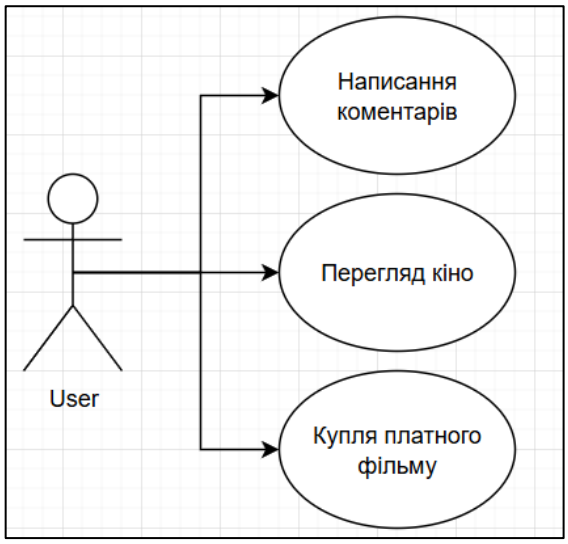


Рисунок 2.1 – Діаграма варіантів використання

Для відображення статичної структури кодової бази та фізичного розподілу модулів розроблено діаграму компонентів (Component Diagram). Вебзастосунок реалізує класичну триланкову архітектуру (3-tier architecture). Компонент WebApp (клієнтський SPA-додаток на React + TypeScript) ініціює HTTP-запити через REST API до компонента Backend API (серверний застосунок на NestJS). Backend API відповідає за виконання бізнес-логіки та здійснює операції читання/запису у компонент Database (реляційна СУБД PostgreSQL). Для обробки транзакцій Backend API асинхронно взаємодіє з ізольованим зовнішнім компонентом Stripe Payment. Загальну схему взаємодії цих структурних одиниць наочно представлено на рисунку 2.2.

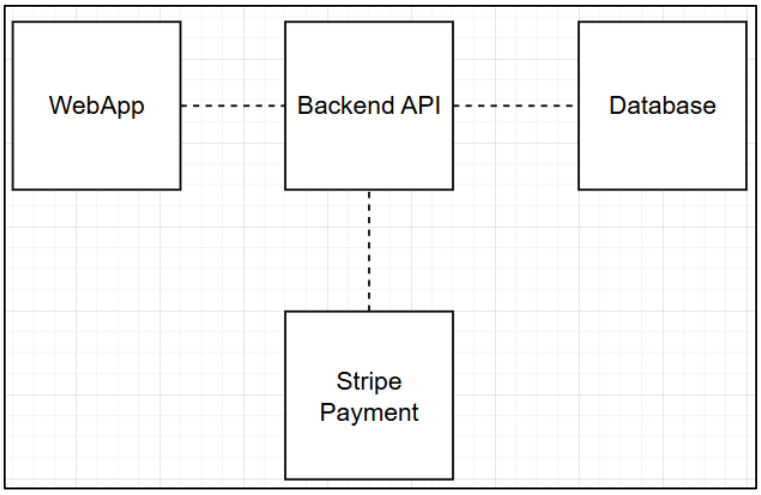


Рисунок 2.2 – Діаграма компонентів

Фізичне розгортання та топологія мережевої інфраструктури відображені на діаграмі розгортання (Deployment Diagram). Клієнтська частина представлена кінцевими пристроями користувачів (Computer та Smartphone), на яких у середовищі веббраузера виконується скомпільована статика фронтенду. Взаємодія з сервером додатків (Server (API, Database)) здійснюється через протокол HTTP. Сервер додатків інкапсулює виконання NestJS-сервісів та локальний екземпляр PostgreSQL. Для верифікації банківських карток сервер додатків за протоколом TLS взаємодіє з віддаленою хмарною інфраструктурою Stripe Server. Топологію розміщення вузлів системи та протоколи їхнього зв'язку наочно представлено на рисунку 2.3.

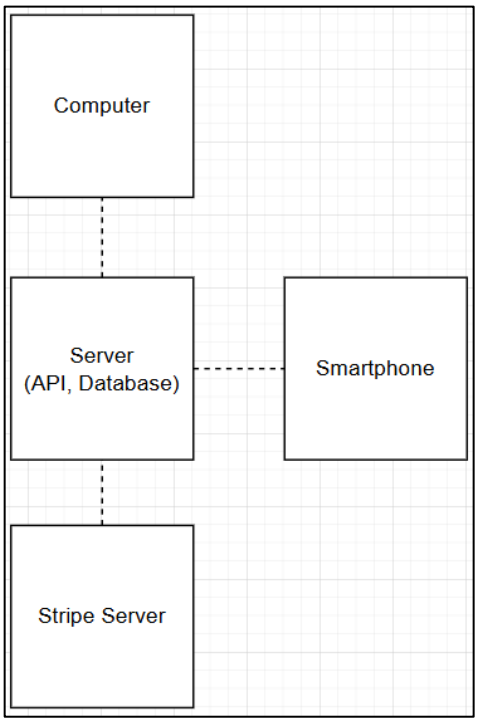


Рисунок 2.3 – Діаграма розгортання

2.2 Поведінкове моделювання та інтеракція об'єктів

Для деталізації динаміки виконання бізнес-процесів та часової послідовності обміну повідомленнями було застосовано нотації BPMN (Business Process Model and Notation) та поведінкові діаграми UML.

Діаграма BPMN ілюструє наскрізний процес взаємодії користувача з платформою, розділений на дві доріжки: «Користувач» та «Онлайн-кінотеатр». Процес стартує з події входу на сайт. Ексклюзивний шлюз перевіряє стан авторизації. Якщо користувач новий, активується підпроцес «Введення даних» та «Створити акаунт/авторизувати». Після успішного переходу на головний екран користувач виконує дію «Обрав фільм».

Наступний шлюз аналізує тип контенту. Для безкоштовного фільму доступ відкривається миттєво. Для платного контенту ініціюється операція «Оплата фільму».

У разі виникнення помилки на стороні банку чи шлюзу, процес завершується генерацією «Повідомлення про помилку», інакше генерується успішна транзакція. Повну послідовність кроків та розгалужень цієї логіки відображено на рисунку 2.4.

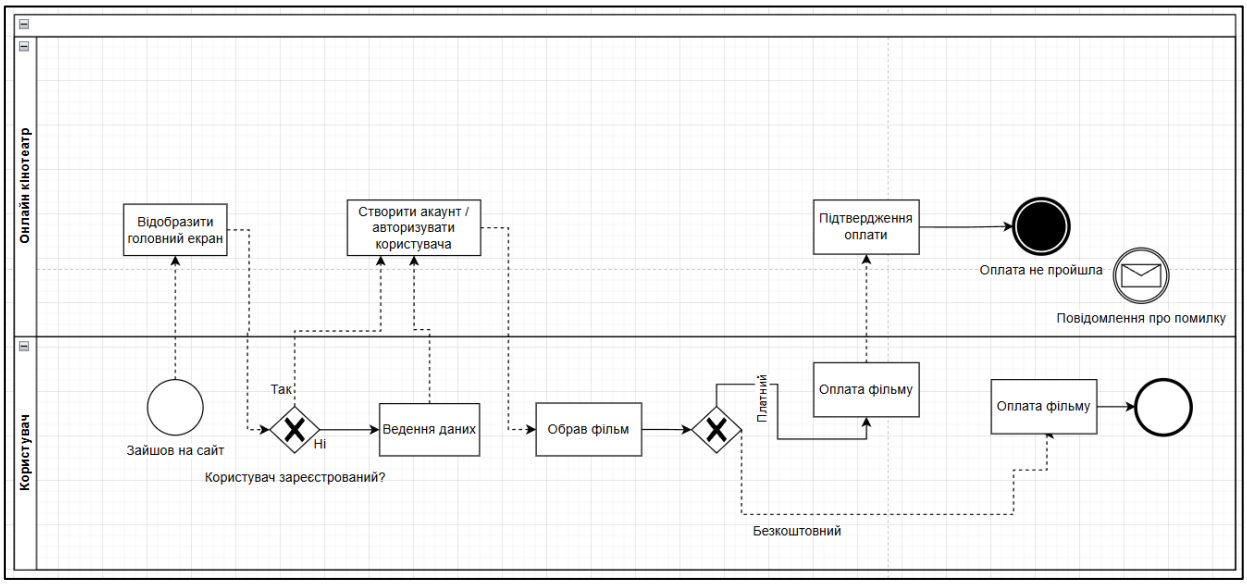


Рисунок 2.4 – Діаграма бізнес-процесу (BPMN)

Функціональна модель у нотації EPC формалізує бізнес-логіку застосунку через оператори розгалуження OR. Вона послідовно описує етапи автентифікації (вхід або реєстрація), перевірки типу контенту (безкоштовний чи платний) та валідації транзакцій (надання доступу до відео або ініціація оплати). Повну логічну структуру цієї моделі представлено на рисунку 2.5.

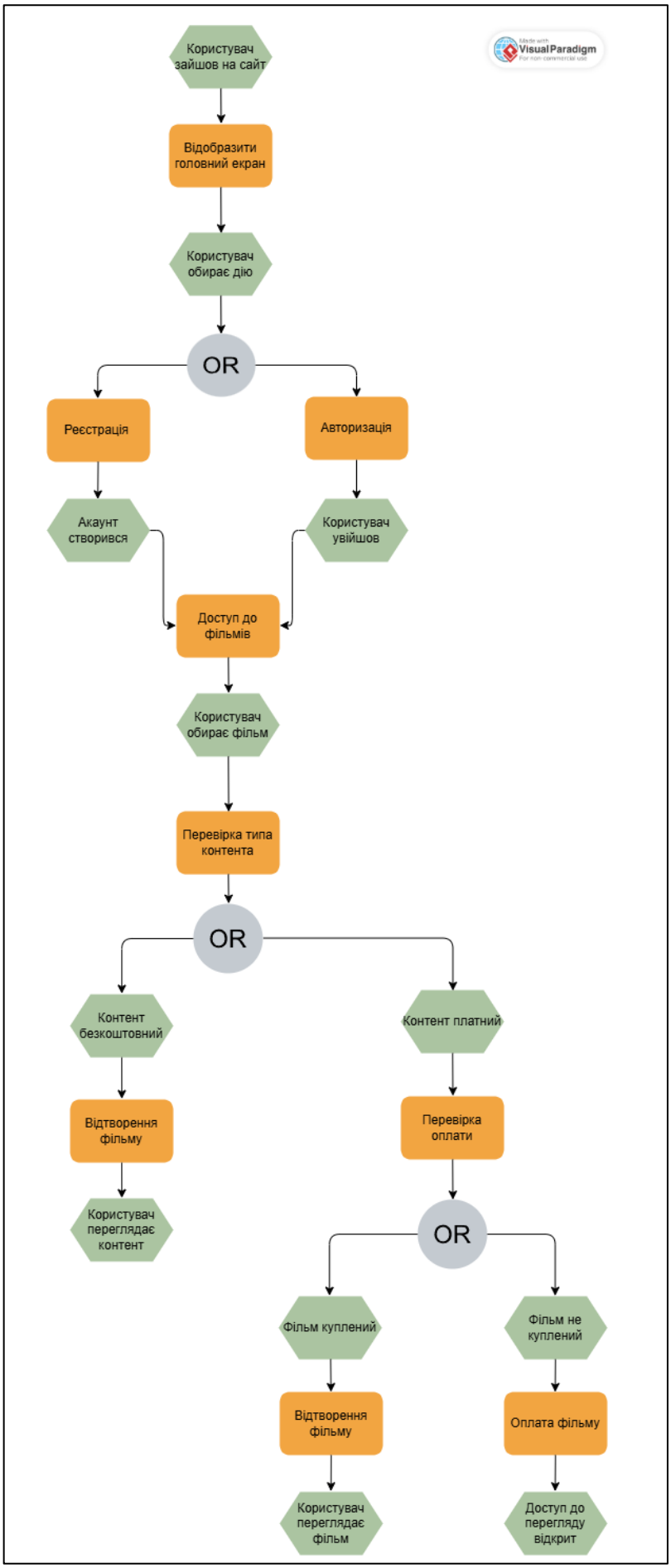


Рисунок 2.5 – Розширений ланцюг бізнес-процесів сервісу

Логіка навігації та переходу станів інтерфейсу формалізована у вигляді алгоритмічної схеми (Flowchart). Схема чітко розмежовує потоки керування. Важливим елементом архітектури є циклічне повернення (loopback): у разі виникнення стану Помилка оплати, система не перериває сесію, а повертає потік керування на етап Вибір фільму з каталогу, запобігаючи блокуванню інтерфейсу. При успішному виконанні умов (безкоштовний тип або успішний статус Stripe) викликається термінальний стан Доступ до перегляду відкрито. Алгоритмічний граф розгалуження цих станів та логіку умовних переходів наочно представлено на рисунку 2.6.

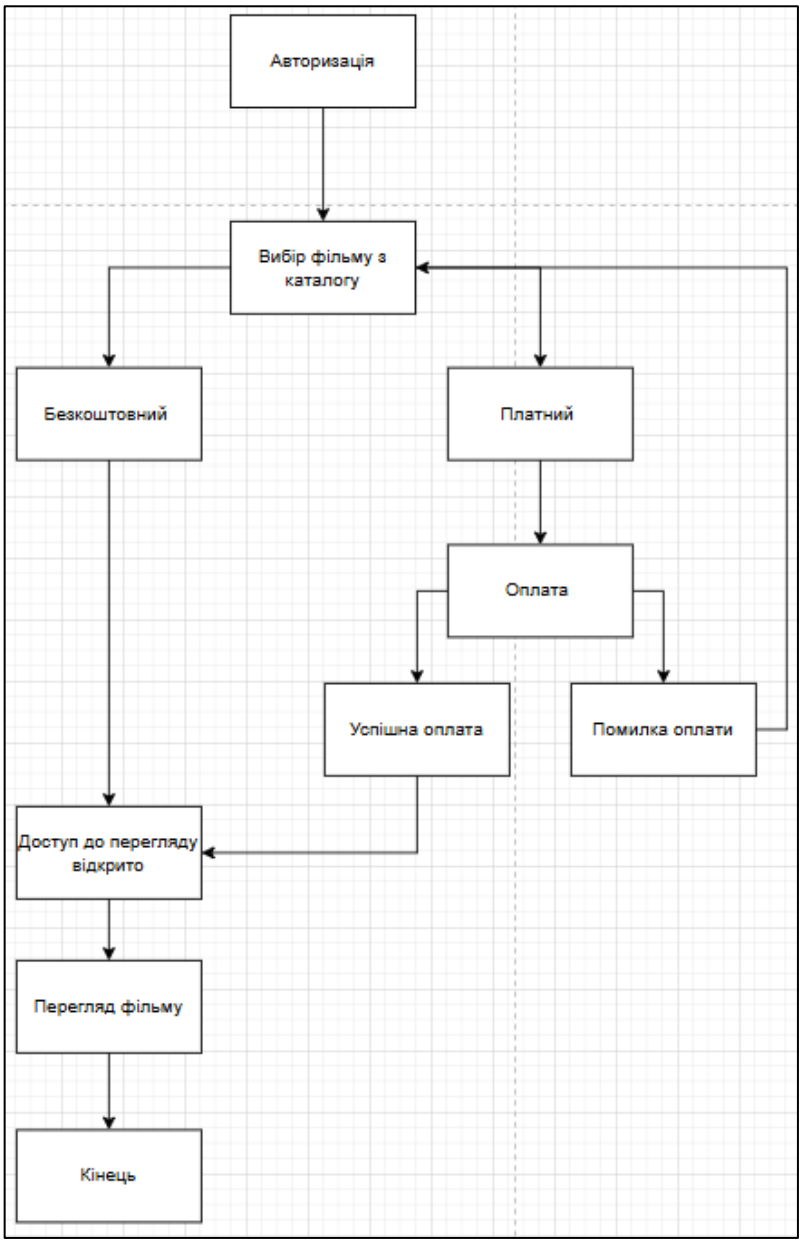


Рисунок 2.6 – Алгоритм вибору та доступу до медіаконтенту

Діаграма послідовності (Sequence Diagram) відображає часову шкалу обміну викликами між об’єктами «Клієнт» (Frontend) та «Онлайн-кінотеатр» (Backend REST API). Вона фіксує хронологічний порядок виконання операцій: від початкових запитів реєстрації до відтворення медіапотоку та деавторизації (Вихід з акаунта).

Хронологію обміну повідомленнями між клієнтом та сервером наочно представлено на рисунку 2.7.



Рисунок 2.7 – Діаграма послідовності

Діаграма кооперації (Collaboration Diagram) доповнює динамічну модель, фокусуючись на структурній взаємодії між трьома ключовими мета-об’єктами: Клієнт, Онлайн-кінотеатр та Платіжна система. На відміну від діаграми послідовності, тут акцент зміщено на архітектурні зв’язки, а черговість викликів задається індексами 1: Повідомлення про вхід, 2: Запит на оплату та 3: Перегляд фільму. Специфіку міжоб’єктних зв’язків та хронологію обміну повідомленнями у такому форматі представлено на рисунку 2.8.

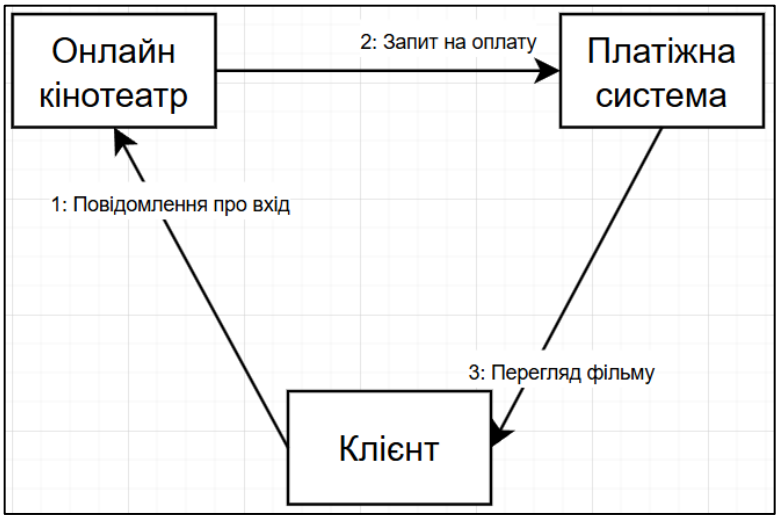


Рисунок 2.8 – Діаграма кооперації інтеграції софтверних об’єктів

Для опису поведінки сутностей контенту під впливом зовнішніх подій розроблено діаграму станів (UML Statechart Diagram). Вона формалізує життєвий цикл фільму в контексті користувацького доступу та прогресу перегляду.

Початковим станом об’єкта є Доступний фільм (контент знаходиться в загальному каталозі). Подія успішної транзакції (через Stripe API) переводить об’єкт у стан Придбаний фільм, що фіксує зміну прав доступу користувача в базі даних. Ініціація відтворення у відеоплеєрі (подія Play) трансформує стан у Переглядається (активація механізму збереження прогресу в watch_history). Після досягнення термінального таймкоду відео (подія End), об’єкт переходить у стан Переглянутий фільм. Повну карту тригерів, подій та умовних переходів між цими станами детально ілюструє рисунок 2.9.

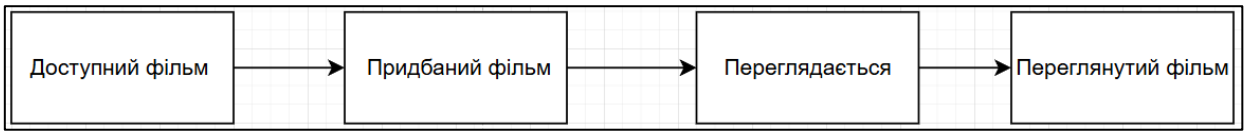


Рисунок 2.9 – Діаграма станів життєвого циклу контенту

2.3 Проєктування бази даних

База даних онлайн-застосунку для перегляду кіно побудована на основі реляційної моделі даних. Такий підхід є доцільним для даного проєкту, оскільки система працює з чітко структурованими сутностями: користувачами, фільмами, жанрами, акторами, відгуками, покупками, історією перегляду та платіжними транзакціями. Реляційна модель дозволяє зберігати дані у вигляді окремих таблиць, встановлювати між ними логічні зв’язки та забезпечувати цілісність інформації за допомогою первинних і зовнішніх ключів.

Для реалізації бази даних у проєкті використовується PostgreSQL. Робота з базою даних на серверній частині здійснюється через TypeORM, що дозволяє описувати таблиці у вигляді сутностей, встановлювати зв’язки між ними та виконувати запити до бази даних без прямого написання SQL-коду для кожної операції. Такий підхід спрощує підтримку проєкту, оскільки структура бази даних безпосередньо пов’язана з кодом серверної частини.

Повну логічну структуру бази даних сервісу, склад її реляційних атрибутів та зв’язків між сутностями продемонстровано на рисунку 2.10.

База даних складається з основних таблиць `user`, `movie`, `genre`, `actor`, `review`, `purchase`, `favourite`, `watch_history`, `payment_transaction` та проміжної таблиці `movie_actor`. Кожна таблиця відповідає окремій сутності предметної області або окремому процесу, який виконується у системі.

Основною таблицею для зберігання інформації про користувачів є таблиця `user`. Вона містить дані, необхідні для реєстрації, авторизації та ідентифікації користувача в системі. До основних полів цієї таблиці належать

`id`, `username`, `email`, `passwordHash`, `role`, `createdAt`, `updatedAt` та `birthDate`. Поле `id` є первинним ключем і використовується для однозначної ідентифікації кожного користувача. Поле `email` застосовується для входу до системи, тому воно повинно бути унікальним. Пароль користувача не зберігається у відкритому вигляді, а записується у вигляді хешу в полі `passwordHash`. Це підвищує безпеку системи та зменшує ризик розкриття паролів у разі несанкціонованого доступу до бази даних. Поле `role` дозволяє розділяти користувачів за правами доступу, наприклад, на звичайних користувачів і адміністратора. Поле `birthDate` використовується для контролю вікових обмежень під час перегляду фільмів.

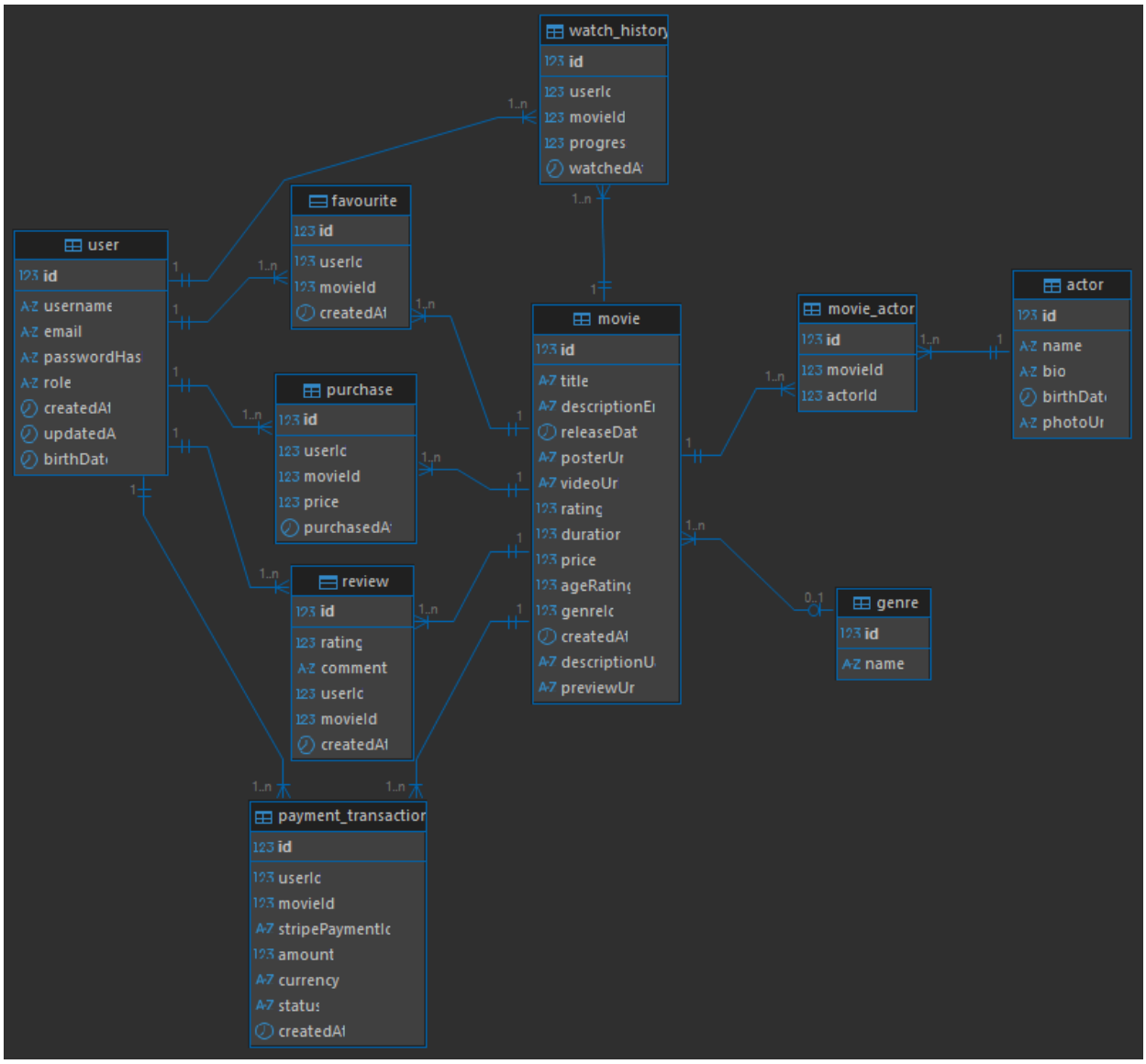


Рисунок 2.10 – Схема бази даних онлайн-застосунку для перегляду кіно

Центральною таблицею бази даних є таблиця `movie`, яка зберігає інформацію про фільми. Вона містить такі поля: `id`, `title`, `descriptionEn`, `releaseDate`, `videoUrl`, `rating`, `duration`, `price`, `ageRating`, `genreId`, `createdAt`, `descriptionUa` та `previewUrl`. Поле `id` є первинним ключем таблиці. Поле `title` зберігає назву фільму, а поля `descriptionEn` та `descriptionUa` дозволяють відображати опис різними мовами. Поле `releaseDate` використовується для збереження дати виходу фільму, `duration` – для тривалості, `rating` – для середньої оцінки, `price` – для вартості доступу до платного фільму. Поле `ageRating` використовується для вікового обмеження. Поле `genreId` є зовнішнім ключем і пов'язує фільм із відповідним жанром. Поля `videoUrl` та `previewUrl` зберігають шляхи до відеофайлу та прев'ю, що дозволяє відокремити метадані фільму від самих медіафайлів.

Таблиця `genre` використовується для зберігання жанрів фільмів. Вона має поля `id` та `name`. Поле `id` є первинним ключем, а поле `name` містить назву жанру. Зв'язок між таблицями `genre` та `movie` реалізований таким чином, що один жанр може бути пов'язаний з багатьма фільмами, але кожен фільм у межах цієї моделі має один основний жанр. Такий підхід спрощує фільтрацію фільмів у каталозі.

Для зберігання інформації про акторів використовується таблиця `actor`. Вона містить поля `id`, `name`, `bio`, `birthDate` та `photoUrl`. Поле `name` зберігає ім'я актора, поле `bio` – коротку біографічну інформацію, `birthDate` – дату народження, а `photoUrl` – шлях до фотографії. Оскільки один фільм може мати багато акторів, а один актор може зніматися у багатьох фільмах, між таблицями `movie` та `actor` реалізовано зв'язок багато-до-багатьох.

Для реалізації зв'язку багато-до-багатьох використовується проміжна таблиця `movie_actor`. Вона містить поля `id`, `movieId` та `actorId`. Поле `movieId` є зовнішнім ключем на таблицю `movie`, а поле `actorId` – зовнішнім ключем на таблицю `actor`. Завдяки цій таблиці система може зберігати інформацію про акторський склад кожного фільму без дублювання даних про акторів у таблиці фільмів.

Окрема частина бази даних пов’язана з діями користувача. Для збереження обраних фільмів використовується таблиця `favourite`. Вона містить поля `id`, `userId`, `movieId` та `createdAt`. Поле `userId` вказує, який користувач додав фільм до обраного, а поле `movieId` визначає конкретний фільм. Така структура дозволяє кожному користувачу мати власний список обраних фільмів.

Для збереження інформації про придбані фільми використовується таблиця `purchase`. Вона містить поля `id`, `userId`, `movieId`, `price` та `purchasedAt`. Після успішної оплати в цій таблиці створюється запис, який підтверджує, що користувач отримав доступ до конкретного платного фільму. Поле `price` зберігає вартість фільму на момент покупки. Це важливо, оскільки ціна фільму в майбутньому може змінитися, але історія покупки повинна залишатися коректною.

Таблиця `watch_history` використовується для збереження історії перегляду. Вона містить поля `id`, `userId`, `movieId`, `progress` та `watchedAt`. Поле `progress` дозволяє зберігати прогрес перегляду фільму, а поле `watchedAt` – дату та час останнього перегляду. Завдяки цій таблиці користувач може продовжити перегляд з того місця, на якому він зупинився.

Для реалізації системи оцінювання та коментування фільмів використовується таблиця `review`. Вона містить поля `id`, `rating`, `comment`, `userId`, `movieId` та `createdAt`. Поле `rating` зберігає оцінку фільму, поле `comment` – текстовий відгук, а поля `userId` і `movieId` визначають, який користувач залишив відгук і до якого фільму він належить. Після створення, редагування або видалення відгуку серверна частина перераховує середній рейтинг фільму та оновлює відповідне поле в таблиці `movie`.

Таблиця `payment_transaction` використовується для збереження інформації про платіжні операції. Вона містить поля `id`, `userId`, `movieId`, `stripePaymentId`, `amount`, `currency`, `status` та `createdAt`. Поле `stripePaymentId` зберігає ідентифікатор платежу, отриманий від Stripe. Поле `amount` містить суму платежу, `currency` – валюту, а `status` – стан транзакції.

Ця таблиця потрібна для збереження історії оплат, перевірки успішності транзакцій і формування PDF-чека для користувача.

Між таблицями бази даних встановлено логічні зв’язки. Таблиця `user` пов’язана з таблицями `favourite`, `purchase`, `watch_history`, `review` та `payment_transaction` зв’язком один-до-багатьох. Це означає, що один користувач може мати багато обраних фільмів, багато покупок, багато записів історії перегляду, багато відгуків і багато платіжних транзакцій.

Таблиця `movie` також має зв’язок один-до-багатьох з таблицями `favourite`, `purchase`, `watch_history`, `review` та `payment_transaction`. Один фільм може бути доданий до обраного багатьма користувачами, придбаний багатьма користувачами, переглянутий багатьма користувачами та мати багато відгуків. При цьому кожен окремий запис у цих таблицях стосується конкретного користувача і конкретного фільму.

Зв’язок між таблицями `movie` та `genre` реалізовано через поле `genreId` у таблиці `movie`. Один жанр може належати багатьом фільмам. Така структура дозволяє виконувати фільтрацію каталогу за жанром без дублювання назви жанру в кожному записі фільму.

Зв’язок між таблицями `movie` та `actor` реалізовано через таблицю `movie_actor`. Це необхідно, тому що між фільмами та акторами існує зв’язок багато-до-багатьох. Наприклад, один фільм може містити декілька акторів, а один актор може бути пов’язаний з декількома фільмами. Якби список акторів зберігався безпосередньо в таблиці `movie`, це призвело б до дублювання даних і ускладнило б пошук фільмів за конкретним актором. Тому використання проміжної таблиці є правильним рішенням для реляційної бази даних.

Для забезпечення цілісності даних у базі використовуються первинні та зовнішні ключі. Первинний ключ `id` є у кожній таблиці та дозволяє однозначно визначити кожен запис. Зовнішні ключі `userId`, `movieId`, `genreId` та `actorId` забезпечують зв’язки між таблицями. Завдяки цьому база даних не допускає ситуації, коли, наприклад, відгук існує без користувача або

без фільму, а запис про оплату не пов’язаний з конкретним користувачем і конкретним фільмом.

Окрему увагу при проєктуванні бази даних було приділено нормалізації. Нормалізація потрібна для того, щоб уникнути дублювання даних, зменшити ризик помилок під час оновлення інформації та зробити структуру бази даних логічною і зручною для подальшої підтримки. Розроблена база даних відповідає вимогам третьої нормальної форми.

Перша нормальна форма передбачає, що всі поля таблиць мають атомарні значення, тобто в одному полі зберігається лише одне значення. У розробленій базі даних ця умова виконується. Наприклад, у таблиці `user` окремо зберігаються `username`, `email`, `passwordHash`, `role` та `birthDate`. У таблиці `movie` окремо зберігаються назва, опис, дата виходу, рейтинг, тривалість, ціна та вікове обмеження. Список акторів не зберігається в одному полі таблиці `movie`, а винесений в окрему таблицю `actor` і проміжну таблицю `movie_actor`. Це підтверджує відповідність першій нормальній формі.

Друга нормальна форма передбачає, що база даних уже перебуває у першій нормальній формі, а всі неключові поля залежать від усього первинного ключа. У цій базі даних кожна таблиця має власний первинний ключ `id`, а інші поля описують саме той об’єкт, який представлений у таблиці. Наприклад, у таблиці `movie` поля `title`, `releaseDate`, `rating`, `duration`, `price` та `ageRating` залежать від конкретного фільму. У таблиці `user` поля `email`, `passwordHash`, `username`, `role` та `birthDate` залежать від конкретного користувача. У таблиці `review` поля `rating` і `comment` належать конкретному відгуку, який пов’язаний з користувачем і фільмом. Таким чином, часткових залежностей у структурі бази даних немає.

Третя нормальна форма передбачає, що база даних перебуває у другій нормальній формі, а неключові поля не мають транзитивних залежностей від первинного ключа. Іншими словами, неключові поля не повинні залежати одне від одного, а мають залежати тільки від ключа таблиці. У розробленій базі даних ця вимога також виконується.

Наприклад, у таблиці `movie` зберігається тільки `genreId`, а назва жанру не дублюється в цій таблиці. Назва жанру зберігається окремо в таблиці `genre`. Це означає, що дані про жанр не залежать транзитивно від фільму, а зберігаються у власній таблиці. Якщо назву жанру потрібно змінити, достатньо змінити її один раз у таблиці `genre`, а не оновлювати всі записи фільмів.

Так само інформація про акторів не дублюється в таблиці `movie`. Ім'я, біографія, дата народження та фото актора зберігаються в таблиці `actor`, а зв'язок з фільмами реалізовано через `movie_actor`. Це усуває дублювання даних про акторів і відповідає третій нормальній формі.

У таблиці `review` не зберігаються email користувача, назва фільму або інші дані, які вже є в таблицях `user` і `movie`. Замість цього використовуються зовнішні ключі `userId` та `movieId`. Завдяки цьому відгук містить тільки ті дані, які безпосередньо належать до нього: оцінку, коментар і дату створення.

У таблиці `payment_transaction` також не дублюються дані користувача або фільму. Вона зберігає тільки ідентифікатори `userId` та `movieId`, а також дані, які безпосередньо стосуються платежу: `stripePaymentId`, `amount`, `currency`, `status` та `createdAt`. Поле `amount` не порушує третю нормальну форму, оскільки воно фіксує суму конкретної транзакції на момент її виконання. Це історичне значення, яке може відрізнятися від поточної ціни фільму в таблиці `movie`.

Поле `price` у таблиці `purchase` також є допустимим, оскільки воно зберігає ціну покупки на конкретний момент часу. Якщо в майбутньому ціна фільму зміниться, інформація про вже виконану покупку залишиться правильною. Тому це поле не є дублюванням поточної ціни фільму, а виконує роль історичного запису.

Таким чином, база даних відповідає третій нормальній формі, оскільки всі таблиці мають атомарні поля, не містять повторюваних груп даних, усі неключові атрибути залежать від первинного ключа, а транзитивні залежності між неключовими атрибутами відсутні. Така структура зменшує дублювання

інформації, спрощує оновлення даних і забезпечує логічну цілісність бази даних.

Проєктування бази даних у такому вигляді дозволяє ефективно реалізувати основні функції онлайн-застосунку: реєстрацію та авторизацію користувачів, перегляд каталогу фільмів, фільтрацію за жанрами, перегляд інформації про акторів, додавання фільмів до обраного, збереження історії перегляду, створення відгуків, покупку платного контенту та збереження платіжних транзакцій. Обрана структура є достатньо гнучкою для подальшого розширення, наприклад, додавання підписок, декількох жанрів для одного фільму, персональних рекомендацій або розширеної аналітики переглядів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

Третій розділ присвячено програмній реалізації онлайн-застосунку. На цьому етапі визначено загальну архітектуру системи, основні API-маршрути та логіку взаємодії користувача з інтерфейсом.

3.1 Обґрунтування кодової бази та програмна архітектура застосунку

Застосунок побудовано за клієнт-серверною архітектурою. Клієнтська частина виконується у браузері користувача, відображає інтерфейс і надсилає HTTP-запити до серверного API. Серверна частина приймає запити, виконує бізнес-логіку, працює з базою даних, створює платіжні наміри, зберігає транзакції та повертає клієнту структуровані відповіді. PostgreSQL використовується як основне сховище даних.

Клієнтська частина не повинна напряму звертатися до бази даних або зберігати секретні ключі. Вона працює лише з відкритими конфігураційними значеннями, наприклад VITE_API_URL і VITE_STRIPE_PUBLISHABLE_KEY. Усі операції, що потребують захищеної логіки або секретних параметрів, виконуються на сервері. Це стосується створення платежу, перевірки унікальності email, хешування пароля та запису інформації про покупки.

Серверна частина має модульну структуру NestJS. Кожен модуль відповідає за окрему групу функцій: Auth – за реєстрацію і вхід; User – за профіль, відгуки та покупки; Movie – за каталог фільмів; Actor і Genre – за довідникові сутності; Payment – за платіжну логіку; AwsS3 – за роботу з файлами. Такий поділ зменшує зв’язність коду і робить систему зрозумілішою.

Архітектурно система також відокремлює бізнес-логіку від контролерів.

Контролери приймають HTTP-запити і передають їх у сервіси. Сервіси виконують основні операції, взаємодіють із репозиторіями TypeORM і повертають результат. Такий підхід відповідає типовій структурі NestJS і дозволяє тестувати окремі частини логіки незалежно від маршрутизації. Приклад програмної реалізації такого розділення шарів та опрацювання вхідного HTTP-запиту наведено на рисунку 3.1.

```
@Controller('movies')
export class MovieController {

constructor(private readonly movieService: MovieService) {}

@Get(':id')
findOne(@Param('id', ParseIntPipe) id: number) {
return this.movieService.findOne(id);
}
}
```

Рисунок 3.1 – Програмний код обробника маршруту GET /movies/:id у NestJS

Клієнтська частина організована навколо сторінок і компонентів. Сторінки відповідають основним маршрутам застосунку: головна сторінка, сторінка фільму, реєстрація, вхід, профіль, платіжний модал. Компоненти забезпечують повторне використання елементів інтерфейсу, наприклад картки фільму, форми, кнопки, повідомлення і модальні вікна.

Обмін даними між клієнтом і сервером відбувається через REST API. Клієнт формує запити з параметрами фільтрації або даними форми, сервер повертає JSON-відповідь. Для помилок сервер використовує стандартні HTTP-статуси, а клієнт показує повідомлення користувачу. Наприклад, при повторній реєстрації з уже використаним email сервер повертає конфлікт, а клієнт відображає відповідну помилку. Програмну логіку відправки асинхронного запиту до бекенду та отримання статусу відповіді наочно продемонстровано на рисунку 3.2.

```
const API_URL = ‘http://localhost:3000/auth’;

export const register = async (data: { email: string; username: string; password: string }) => {
  const res = await fetch(`${API_URL}/register`, {
    method: ‘POST’,
    headers: { ‘Content-Type’: ‘application/json’ },
    body: JSON.stringify(data),
  });
  return { ...(await res.json()), _status: res.status };
};
```

Рисунок 3.2 – Реалізація взаємодії клієнтської частини з REST API модуля Auth

3.2 Програмна реалізація серверних сервісів та REST API

API онлайн-застосунку для перегляду кіно побудовано за принципами REST-архітектури. Кожна функціональна частина системи має власну групу маршрутів, які відповідають окремим модулям серверної частини. Такий підхід дозволяє логічно розділити відповідальність між модулями, спростити підтримку коду та зробити взаємодію між клієнтською і серверною частинами зрозумілою.

Серверна частина застосунку реалізована на NestJS. У цій технології кожен модуль зазвичай складається з контролера, сервісу та допоміжних класів. Контролер приймає HTTP-запити від клієнта, сервіс виконує основну бізнес-логіку, а робота з базою даних здійснюється через TypeORM. Завдяки цьому код серверної частини не змішує логіку маршрутизації, обробку даних і доступ до бази даних.

Клієнтська частина, створена на React, звертається до API за допомогою HTTP-запитів. Для цього використовуються окремі функції, які відповідають за взаємодію з сервером. Такий підхід дозволяє не дублювати запити в різних компонентах інтерфейсу та зберігати логіку роботи з API в одному місці.

Основними модулями API є `Auth`, `Movie`, `User`, `Payment`, `Genre`, `Actor` та `Upload`. Кожен із цих модулів відповідає за окрему частину функціональності застосунку. Модуль авторизації обробляє реєстрацію та вхід користувачів, модуль фільмів відповідає за каталог, модуль користувача – за профіль, відгуки, покупки, обране та історію переглядів. Модуль платежів забезпечує взаємодію зі Stripe, а модуль завантаження файлів використовується для збереження зображень і відео.

Модуль `Auth` відповідає за реєстрацію та авторизацію користувача. Під час реєстрації клієнт передає на сервер email, ім’я користувача та пароль. Сервер перевіряє, чи не використовується вже такий email, після чого хешує пароль і створює новий запис користувача в базі даних. Якщо користувач із таким email вже існує, сервер повертає помилку. Це дозволяє уникнути дублювання облікових записів.

Під час входу користувач передає email і пароль. Сервер знаходить користувача за email, перевіряє правильність пароля та у разі успішної перевірки повертає основні дані профілю. До таких даних належать ідентифікатор користувача, email, ім’я користувача та роль. Надалі ці дані використовуються клієнтською частиною для відображення профілю, перевірки доступу до функцій і роботи з персональними діями користувача.

Модуль `Movie` відповідає за роботу з каталогом фільмів. Основним маршрутом цього модуля є `GET /movies`, який повертає список фільмів. Цей маршрут підтримує фільтрацію, сортування та пагінацію. Клієнт може передавати параметри `genreId`, `actorId`, `isPaid`, `minRating`, `ageRating`, `search`, `sortBy`, `showPurchased`, `userId`, `yearFrom`, `yearTo`, `page` та `limit`. Завдяки цьому користувач може шукати фільми за назвою, фільтрувати їх за жанром, актором, рейтингом, віковим обмеженням, роком випуску, платністю або статусом покупки.

Важливою особливістю маршруту `GET /movies` є підтримка пагінації. Це означає, що сервер не повертає одразу всі фільми з бази даних, а віддає лише частину записів відповідно до параметрів `page` і `limit`. Такий підхід

зменшує навантаження на сервер і прискорює роботу клієнтської частини, особливо якщо кількість фільмів у базі даних буде збільшуватися.

Для отримання детальної інформації про конкретний фільм використовується маршрут `GET /movies/:id`. Він повертає дані про фільм, його жанр, акторів, а також відгуки користувачів. Цей маршрут використовується на сторінці детального перегляду фільму, де користувач може побачити повний опис, рейтинг, тривалість, вартість, вікове обмеження, список акторів і коментарі інших користувачів.

Окремо в модулі `Movie` передбачено маршрути для отримання популярних і останніх фільмів. Маршрут `GET /movies/popular` повертає фільми, відсортовані за рейтингом, а `GET /movies/latest` – останні додані фільми. Це дозволяє формувати окремі блоки на головній сторінці застосунку.

Модулі `Genre` та `Actor` використовуються для роботи з жанрами та акторами. Жанри потрібні для класифікації фільмів, а актори – для відображення акторського складу та фільтрації каталогу. Для жанрів передбачено маршрути отримання всіх жанрів, отримання одного жанру, створення, редагування та видалення. Аналогічно для акторів реалізовано отримання списку акторів, отримання конкретного актора, створення, редагування та видалення. Це дозволяє підтримувати каталог фільмів у структурованому вигляді.

Модуль `User` відповідає за функції, пов’язані з персональними даними та діями користувача. Через цей модуль користувач може отримати та оновити свій профіль, змінити пароль, переглянути історію переглядів, додати фільм до історії, працювати з обраними фільмами, створювати відгуки та переглядати список придбаних фільмів.

Для профілю користувача використовуються маршрути `GET /user/:userId/profile` та `PUT /user/:userId/profile`. Перший маршрут повертає основні дані користувача, а другий дозволяє оновити профіль. Окремо передбачено маршрут `PUT /user/:userId/password`, який використовується для зміни пароля. При зміні пароля користувач передає старий і новий пароль,

після чого сервер перевіряє правильність старого пароля та зберігає новий пароль у хешованому вигляді.

Для роботи з історією переглядів використовується маршрут `GET /user/:userId/history`, який повертає список фільмів, переглянутих користувачем. Також є маршрут `POST /user/:userId/watch-history/:movieId`, який додає фільм до історії або оновлює час останнього перегляду. Завдяки цьому користувач може бачити, які фільми він уже переглядав.

Для роботи з обраними фільмами передбачені маршрути `GET /user/:userId/favourites`, `POST /user/:userId/favourites/:movieId` та `DELETE /user/:userId/favourites/:movieId`. Перший маршрут повертає список обраних фільмів користувача, другий додає фільм до обраного, а третій видаляє його зі списку. Така функціональність робить застосунок зручнішим, оскільки користувач може зберігати фільми, які планує переглянути пізніше.

Відгуки реалізовані через маршрути `GET /user/:userId/reviews`, `POST /user/:userId/reviews/:movieId`, `PATCH /user/:userId/reviews/:reviewId` та `DELETE /user/:userId/reviews/:reviewId`. Користувач може створити відгук до фільму, змінити власний відгук або видалити його. Після створення, редагування або видалення відгуку сервер перераховує середній рейтинг фільму. Це дозволяє підтримувати рейтинг фільмів в актуальному стані.

Для покупок передбачено маршрути `GET /user/:userId/purchases`, `GET /user/:userId/purchases/:movieId/check` та `POST /user/:userId/purchases/:movieId`. Перший маршрут повертає список придбаних фільмів, другий перевіряє, чи купив користувач конкретний фільм, а третій створює запис про покупку. Ці маршрути важливі для розмежування доступу до безкоштовного та платного контенту.

Модуль `Payment` відповідає за оплату платних фільмів. Оплата реалізована через Stripe. На першому етапі клієнтська частина надсилає на сервер суму платежу, ідентифікатор фільму та ідентифікатор користувача. Сервер створює платіжний намір через Stripe та повертає клієнту `clientSecret`. Після цього клієнтська частина підтверджує оплату через Stripe. Якщо платіж

проходить успішно, клієнт надсилає запит на збереження транзакції в базі даних. Структуру об’єкта передачі даних, що ініціює цей транзакційний процес на сервері, разом із правилами його типізації та валідації наведено на рисунку 3.3.

```
export class CreatePaymentIntentDto {
  @Type(() => Number)
  @IsNumber()
  @Min(0)
  amount: number;
  @Type(() => Number)
  @IsInt()
  @IsPositive()
  movieId: number;
  @Type(() => Number)
  @IsInt()
  @IsPositive()
  userId: number;
}
```

Рисунок 3.3 – Схема валідації та типізації DTO-класу для створення платіжного наміру Stripe

Для роботи з оплатою використовуються маршрути `POST /payment/create-payment-intent`, `POST /payment/save-transaction`, `GET /payment/user/:userId/transactions` та `GET /payment/transaction/:id/receipt`. Перший маршрут створює платіжний намір, другий зберігає успішну транзакцію, третій повертає історію транзакцій користувача, а четвертий формує PDF-квитанцію за конкретною транзакцією.

Модуль `Upload` використовується для завантаження файлів. Він підтримує завантаження зображень, постерів фільмів, прев’ю фільмів, фото акторів, прев’ю акторів і відеофайлів. Для цього використовуються маршрути `POST /upload/image`, `POST /upload/movie-poster`, `POST /upload/movie-preview`, `POST /upload/actor-photo`, `POST /upload/actor-preview` та `POST

/upload/movie-video`. Для завантаження файлів застосовується перевірка типу файлу та обмеження розміру. Це потрібно для того, щоб сервер приймав тільки допустимі файли та не зберігав некоректні дані.

Окрім опису окремих маршрутів, важливо розглянути основні сценарії роботи застосунку. Вони показують, як між собою взаємодіють клієнтська частина, сервер, база даних і зовнішні сервіси.

Першим основним сценарієм є реєстрація користувача. Користувач відкриває сторінку реєстрації та вводить email, ім’я користувача і пароль. Клієнтська частина виконує базову перевірку введених даних і надсилає запит `POST /auth/register`. Сервер перевіряє, чи не зайнятий email, хешує пароль і створює запис у таблиці користувачів. Якщо реєстрація пройшла успішно, користувач може перейти до входу в систему. Якщо email уже використовується, сервер повертає помилку, а клієнтська частина відображає відповідне повідомлення.

Другим сценарієм є вхід користувача. Користувач вводить email і пароль, після чого клієнт надсилає запит `POST /auth/login`. Сервер перевіряє наявність користувача та правильність пароля. У разі успіху сервер повертає дані користувача, які клієнтська частина зберігає та використовує для подальшої роботи. Після входу користувач отримує доступ до персональних функцій: профілю, обраних фільмів, історії переглядів, покупок і відгуків.

Третім важливим сценарієм є перегляд каталогу фільмів. Коли користувач відкриває головну сторінку, клієнтська частина надсилає запит `GET /movies`. Якщо користувач використовує фільтри, до запиту додаються відповідні параметри. Наприклад, користувач може вибрати жанр, актора, мінімальний рейтинг, рік випуску або ввести пошуковий запит. Сервер обробляє ці параметри, формує запит до бази даних і повертає список фільмів з урахуванням фільтрації та пагінації. Після цього клієнтська частина відображає отримані фільми у вигляді карток.

Четвертим сценарієм є відкриття детальної сторінки фільму. Користувач натискає на фільм у каталозі, після чого клієнт надсилає запит `GET

/movies/:id`. Сервер повертає повну інформацію про фільм, включаючи жанр, акторів і відгуки. На цій сторінці користувач може переглянути опис фільму, його рейтинг, тривалість, вікове обмеження, ціну, акторський склад і коментарі інших користувачів.

Окремим сценарієм є додавання фільму до обраного. Користувач натискає відповідну кнопку на картці або сторінці фільму. Клієнт надсилає запит `POST /user/:userId/favourites/:movieId`. Сервер створює запис у таблиці обраних фільмів. Якщо користувач хоче видалити фільм з обраного, клієнт надсилає запит `DELETE /user/:userId/favourites/:movieId`, після чого сервер видаляє відповідний запис. Список обраних фільмів користувач може отримати через `GET /user/:userId/favourites`.

Ще одним сценарієм є збереження історії переглядів. Коли користувач переглядає фільм, клієнтська частина може надіслати запит `POST /user/:userId/watch-history/:movieId`. Сервер створює новий запис в історії або оновлює час останнього перегляду. Завдяки цьому користувач може переглянути список фільмів, які він дивився раніше, через маршрут `GET /user/:userId/history`.

Найбільш складним сценарієм є купівля платного фільму. Цей процес охоплює клієнтську частину, серверну частину, базу даних і зовнішній платіжний сервіс Stripe. Спочатку користувач відкриває сторінку платного фільму та натискає кнопку оплати. Клієнт надсилає на сервер запит `POST /payment/create-payment-intent`, передаючи суму платежу, ідентифікатор фільму та ідентифікатор користувача. Сервер створює платіжний намір у Stripe та повертає `clientSecret`. Після цього клієнтська частина використовує отриманий `clientSecret` для підтвердження платежу через Stripe. Якщо оплата проходить успішно, клієнт надсилає запит `POST /payment/save-transaction`, щоб зберегти інформацію про транзакцію в базі даних. Далі створюється запис про покупку фільму через `POST /user/:userId/purchases/:movieId`. Після цього користувач отримує доступ до платного фільму, а інтерфейс оновлює стан кнопки оплати або перегляду.

Також користувач може переглянути історію своїх платежів. Для цього клієнт надсилає запит `GET /payment/user/:userId/transactions`. Сервер повертає список транзакцій користувача, відсортований за датою створення. Якщо користувачу потрібна квитанція, клієнт надсилає запит `GET /payment/transaction/:id/receipt`, після чого сервер формує PDF-файл квитанції.

Сценарій створення відгуку починається з того, що користувач відкриває сторінку фільму та вводить оцінку і коментар. Клієнт надсилає запит `POST /user/:userId/reviews/:movieId`. Сервер створює відгук у базі даних і після цього перераховує середній рейтинг фільму. Якщо користувач змінює відгук, використовується маршрут `PATCH /user/:userId/reviews/:reviewId`, а якщо видаляє – `DELETE /user/:userId/reviews/:reviewId`. У кожному з цих випадків рейтинг фільму оновлюється, щоб відображати актуальну середню оцінку.

Для адміністрування контенту в API передбачено маршрути створення, редагування та видалення фільмів, жанрів і акторів. Наприклад, через `POST /movies` можна створити новий фільм, через `PUT /movies/:id` – оновити його дані, а через `DELETE /movies/:id` – видалити. Аналогічні маршрути реалізовані для жанрів і акторів. Завантаження постерів, прев'ю, фото акторів і відео виконується через модуль `Upload`.

У процесі проєктування API важливо було забезпечити зрозумілу структуру маршрутів. Назви маршрутів відповідають сутностям, з якими вони працюють. Наприклад, маршрути, що починаються з `/movies`, пов'язані з фільмами, `/user` – з діями користувача, `/payment` – з оплатою, `/genres` – з жанрами, `/actors` – з акторами, а `/upload` – із завантаженням файлів. Це робить API логічним і зручним для використання на клієнтській частині.

Розроблена структура API дозволяє реалізувати всі основні функції онлайн-застосунку для перегляду кіно: реєстрацію та вхід користувача, перегляд каталогу, фільтрацію фільмів, перегляд детальної сторінки фільму, роботу з жанрами та акторами, додавання фільмів до обраного, збереження історії переглядів, створення відгуків, купівлю платного контенту, збереження

транзакцій і формування PDF-квитанцій. Завдяки поділу API на окремі модулі система залишається зрозумілою, розширюваною та зручною для подальшої підтримки.

3.3 Валідація вхідних даних та обробка виняткових ситуацій

Під час розробки онлайн-застосунку для перегляду кіно важливо було передбачити не лише успішні сценарії роботи, а й ситуації, коли запит користувача не може бути виконаний. До таких випадків належать неправильні дані авторизації, спроба зареєструвати користувача з уже зайнятим email, звернення до неіснуючого фільму, жанру або актора, повторна покупка фільму, некоректна сума платежу чи завантаження файлу неправильного формату.

Для обробки таких ситуацій у серверній частині використовуються стандартні HTTP-коди відповідей. Вони дозволяють клієнтській частині правильно визначити причину помилки та показати користувачу зрозуміле повідомлення. Наприклад, якщо користувач вводить неправильний пароль, сервер повертає помилку авторизації, а якщо користувач намагається відкрити фільм, якого немає в базі даних, сервер повертає помилку про те, що ресурс не знайдено.

У NestJS для цього використовуються стандартні класи винятків, зокрема `BadRequestException`, `UnauthorizedException`, `ForbiddenException`, `NotFoundException`, `ConflictException` та `InternalServerErrorException`. Завдяки цьому серверна частина повертає не випадкові повідомлення, а структуровані відповіді з відповідним HTTP-статусом. Перелік основних HTTP-кодів помилок, які генерує API застосунку, та типові сценарії їх виникнення наведено у таблиці 3.1. Приклад програмної реалізації механізму перевірки наявності запису в базі даних та генерації відповідного HTTP-винятку у сервісному шарі NestJS представлено на рисунку 3.4.

```
async findOne(id: number) {
  const movie = await this.movieRepository.findOne({
    where: { id },
    relations: ['genre', 'actors', 'reviews', 'reviews.user'],
  });

  if (!movie) {
    throw new NotFoundException(`Movie with id ${id} not found`);
  }

  return movie;
}
```

Рисунок 3.4 – Приклад генерації стандартного винятку NotFoundException у сервісному шарі NestJS

Таблиця 3.1 – Основні HTTP-коди помилок API застосунку

HTTP-код	**Назва помилки**	**Загальне значення**	**Приклад використання в застосунку**
400	Bad Request	Некоректний запит або неправильні вхідні дані	Некоректна сума платежу, відсутній файл, неправильний тип файлу
401	Unauthorized	Користувач не пройшов логін	Неправильний email або пароль
403	Forbidden	Дія заборонена для користувача	Спроба змінити або видалити чужий відгук
404	Not Found	Запитуваний ресурс не знайдено	Не знайдено фільм, жанр, актора, користувача або транзакцію
409	Conflict	Конфлікт даних або повторна дія	Email уже використовується, фільм уже куплено, відгук уже існує
500	Internal Server Error	Внутрішня помилка сервера	Помилка під час формування PDF-квитанції

Найчастіше помилки виникають у тих частинах системи, де користувач передає дані на сервер. Це реєстрація, вхід, оновлення профілю, створення відгуку, покупка фільму, створення платіжної транзакції та завантаження файлів. Для таких запитів сервер виконує перевірку отриманих даних і лише після цього переходить до основної логіки.

Для модуля авторизації основними помилками є зайнятий email під час реєстрації та неправильні дані під час входу. Якщо користувач намагається зареєструватися з email, який уже є в базі даних, сервер повертає код `409 Conflict`. Якщо під час входу email або пароль неправильні, повертається код `401 Unauthorized`.

У модулі фільмів найчастіше використовується помилка `404 Not Found`. Вона виникає тоді, коли користувач або адміністратор звертається до фільму, якого не існує. Також така помилка може виникати під час створення або оновлення фільму, якщо переданий жанр не знайдено.

У модулі користувача обробляються помилки, пов’язані з профілем, обраними фільмами, відгуками, покупками та історією перегляду. Наприклад, якщо користувач намагається оновити email на той, який уже використовується іншим користувачем, сервер повертає код `409 Conflict`. Якщо користувач намагається змінити або видалити не свій відгук, сервер повертає код `403 Forbidden`. Програмну реалізацію багаторівневої верифікації існування ресурсу та перевірки прав доступу поточного автора із генерацією відповідного архітектурного винятку наочно продемонстровано на рисунку 3.5.

Окрему увагу приділено покупкам. Якщо користувач намагається купити фільм, якого не існує, сервер повертає `404 Not Found`. Якщо фільм є безкоштовним або вже був куплений цим користувачем, повертається `409 Conflict`. Це дозволяє уникнути некоректних записів у базі даних і повторних покупок одного й того самого фільму.

У модулі оплати перевіряється коректність суми платежу. Якщо сума неправильна, сервер повертає `400 Bad Request`. Також під час отримання

PDF-квитанції сервер перевіряє, чи існує відповідна транзакція. Якщо транзакцію не знайдено, повертається `404 Not Found`, а якщо виникла технічна проблема під час генерації квитанції, може бути повернуто `500 Internal Server Error`.

```
async deleteReview(userId: number, reviewId: number) {
  const review = await this.reviewRepository.findOne({
    where: { id: reviewId },
    relations: [‘user’, ‘movie’],
  });
  if (!review) {
    throw new NotFoundException(‘Review not found’);
  }
  if (review.user.id !== userId) {
    throw new ForbiddenException(‘You can delete only your own review’);
  }
  const movieId = review.movie.id;
  await this.reviewRepository.remove(review);
  await this.recalculateMovieRating(movieId);
  return { success: true };   }
```

Рисунок 3.5 – Реалізація перевірки прав доступу користувача з генерацією винятку ForbiddenException

Модуль завантаження файлів також має власну систему перевірок. Якщо файл не передано, сервер повертає `400 Bad Request`.

Для зображень перевіряється формат файлу, оскільки дозволені тільки JPEG, PNG та WebP. Також обмежується розмір зображення.

Для відеофайлів дозволені формати MP4 та MOV, а максимальний розмір файлу обмежено. Такі перевірки потрібні для того, щоб на сервер не потрапляли некоректні або занадто великі файли.

3.4 Розробка клієнтського інтерфейсу користувача

Інтерфейс користувача має бути побудований так, щоб основні дії виконувалися без зайвих переходів. Головна сторінка повинна показувати каталог фільмів, фільтри, пошуковий рядок і доступ до сторінок входу або профілю. Картка фільму повинна містити постер, назву, рейтинг, позначку платності або ціни та коротку інформацію, достатню для переходу до детальної сторінки.

Детальна сторінка фільму повинна відображати повний опис, постер або прев'ю, рейтинг, тривалість, дату випуску, вікове обмеження, жанри, акторів, кнопку перегляду або оплати, а також блок відгуків. Якщо фільм платний і користувач його ще не купив, замість доступу до перегляду має пропонуватися оплата. Якщо фільм уже куплений, інтерфейс повинен показати доступ до перегляду без повторного платежу. Програмну реалізацію цієї логіки розмежування доступу на рівні декларативного опису інтерфейсу React-компонента наведено на рисунку 3.6.

```
<div className={styles.actions}>
  {Number(movie.price) > 0 && !hasPurchased ? (
    <button className={styles.purchaseBtn} onClick={handlePurchase}>
      <FontAwesomeIcon icon={faShoppingCart} /> {t('movie.purchase')} $
      {movie.price}
    </button>
  ) : (
    <button className={styles.watchBtn} onClick={handleWatch}>
      <FontAwesomeIcon icon={faPlay} /> {t('common.watchNow')}
    </button>
  )}
  <button className={styles.favoriteBtn} onClick={handleToggleFavorite}>
    <FontAwesomeIcon icon={faHeart} />
    {isFavorite
      ? t('movie.removeFromFavorites')
      : t('movie.addToFavorites')}
  </button>
</div>
```

Рисунок 3.6 – Фрагмент React-компонента умовного відображення кнопок дії

Форма реєстрації повинна бути простою та містити мінімально необхідні поля: ім’я користувача, email і пароль. Для зручності користувача реалізовано кнопку повернення, іконку перегляду пароля, підказку для email і повідомлення про помилки. Валідація має працювати на клієнті до відправлення даних, а сервер додатково перевіряє унікальність email.

Платіжне модальне вікно повинно бути окремим елементом інтерфейсу, який відкривається з детальної сторінки фільму. Клієнт запитує clientSecret, після чого обгортає платіжну форму у Stripe Elements. Для простого сценарію карткової оплати використано CardElement. Після успішного платежу користувач отримує повідомлення про успіх, а доступ до фільму оновлюється.

Сторінка профілю повинна відображати інформацію про користувача, придбані фільми або пов’язані з користувачем дані. Для такого застосунку профіль важливий, оскільки він показує персоналізацію сервісу та дозволяє пов’язати користувача з покупками й відгуками.

Колірна схема інтерфейсу у проєкті орієнтована на темний фон і червоний акцент. Такий підхід є типовим для відеосервісів, оскільки темний фон не відволікає від постерів і відеоконтенту, а акцентний колір виділяє кнопки дії. Приклади практичної програмної реалізації згаданих візуальних модулів, сторінок навігації та елементів інтерфейсу користувача з урахуванням обраної стилістики вебзастосунку наведено на рисунках 3.7 – 3.12.

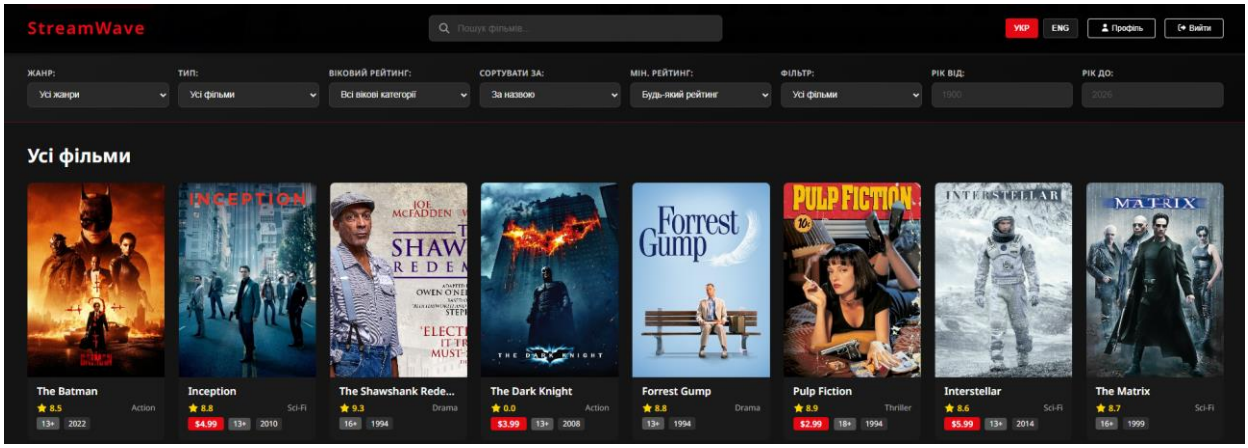


Рисунок 3.7 – Скриншот головної сторінки

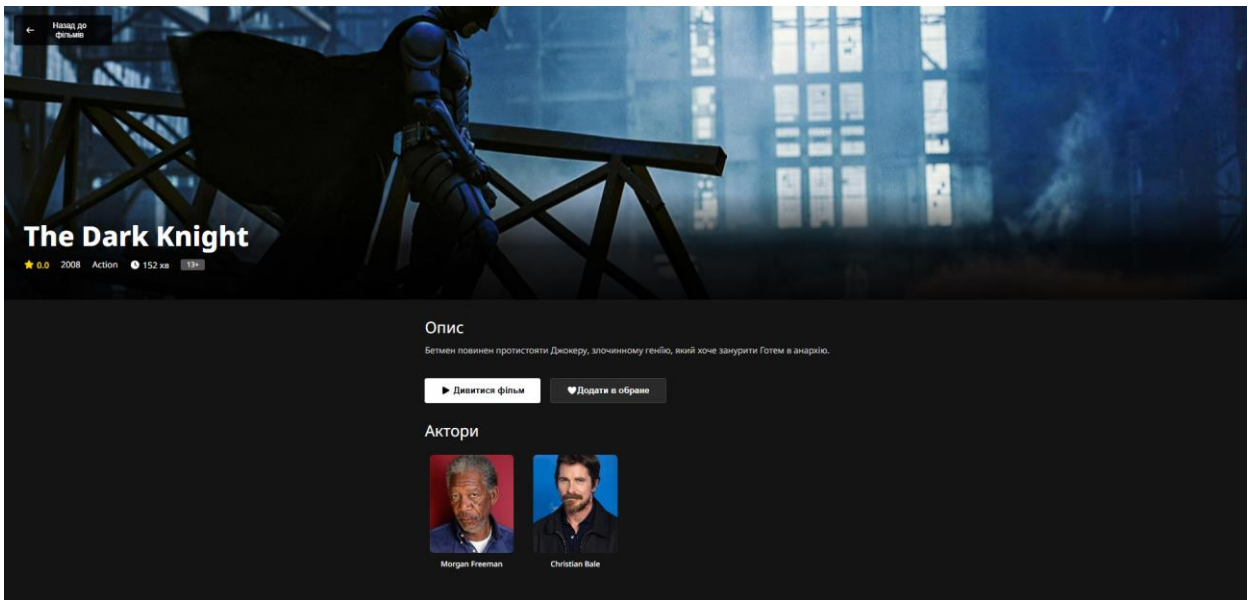


Рисунок 3.8 – Скриншот сторінки фільму

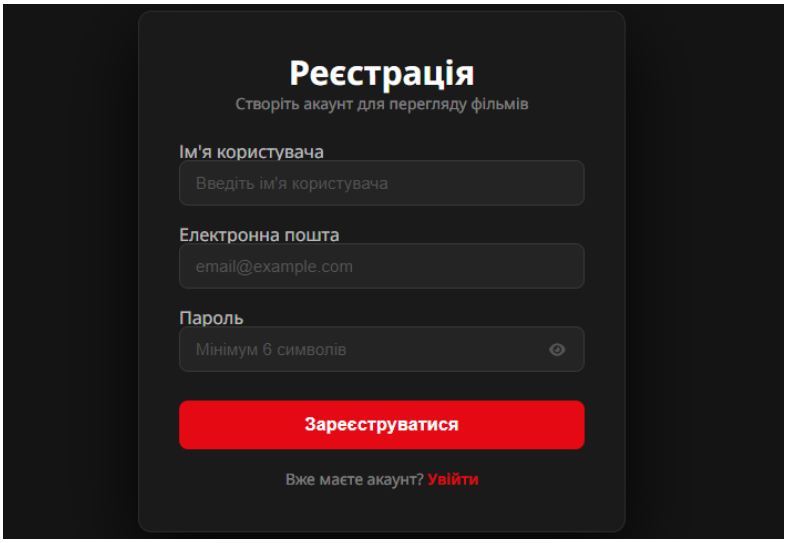


Рисунок 3.9 – Скриншот сторінки реєстрації користувача

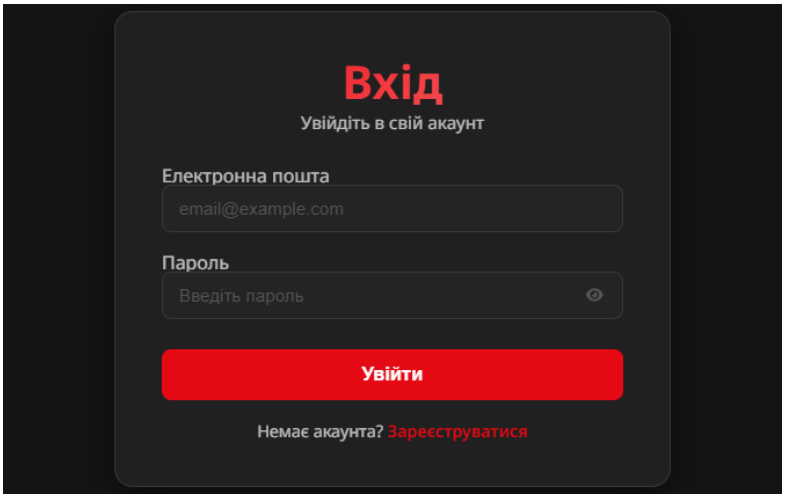


Рисунок 3.10 – Скриншот сторінки логіну користувача

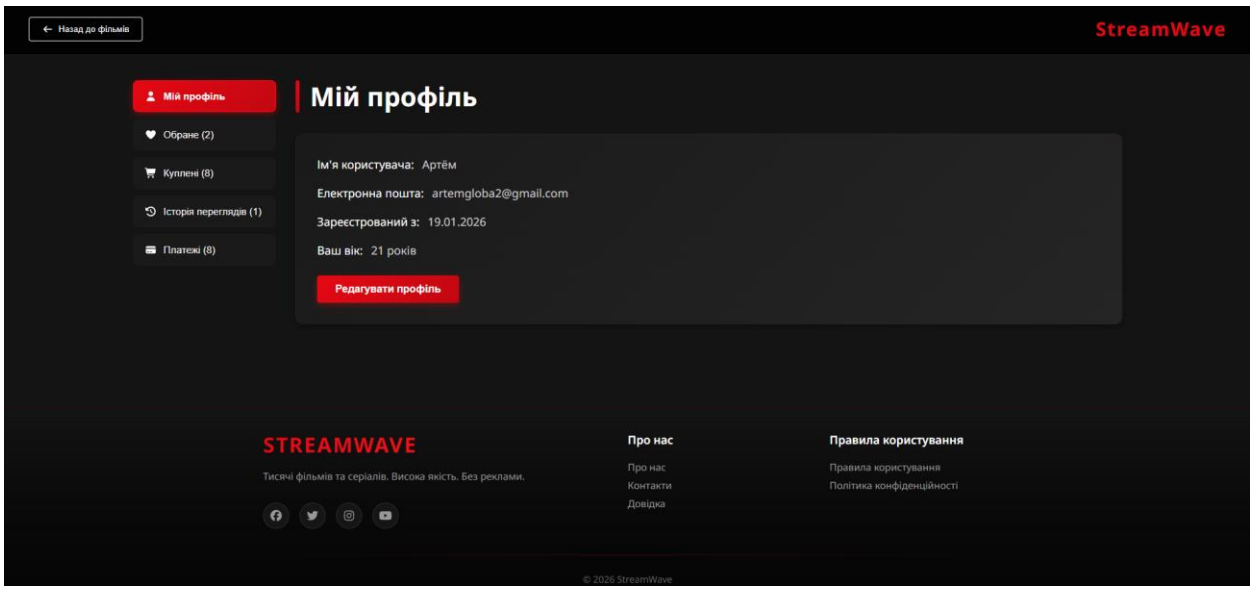


Рисунок 3.11 – Скриншот профілю користувача

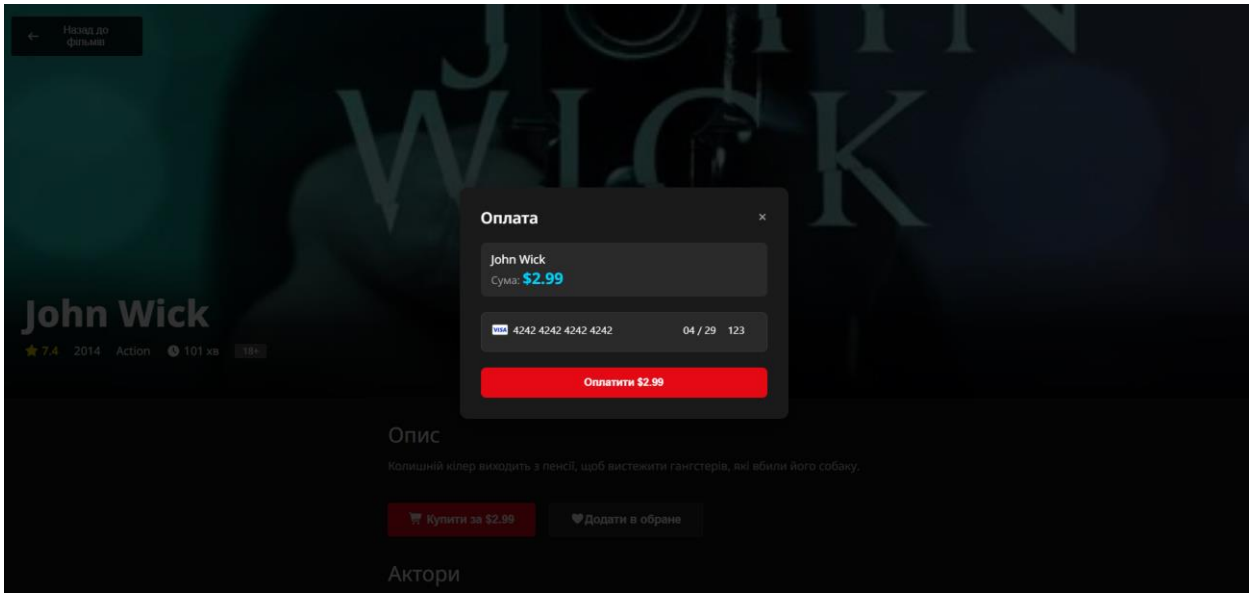


Рисунок 3.12 – Скриншот модального вікна покупки фільму

Таким чином у третьому розділі реалізовано програмну частину застосунку: створено захищений REST API (NestJS, PostgreSQL) з інтеграцією платежів Stripe та розроблено інтерактивний клієнтський інтерфейс (React) із маршрутизацією і логікою розмежування доступу до медіаконтенту.

ВИСНОВКИ

У кваліфікаційній роботі досліджено та реалізовано повний цикл розробки вебзастосунку для перегляду кіно – від аналізу концепції та предметної області до програмного коду. Результатом роботи став працездатний клієнт-серверний сервіс «StreamWave». Проєкт об’єднує в собі каталог фільмів, систему користувацьких профілів, гнучкі інструменти фільтрації, модулі модерації відгуків, механізми захисту контенту за віком та інтеграцію з міжнародною платіжною системою Stripe.

У першому розділі детально проаналізовано архітектуру та принципи функціонування сучасних стримінгових платформ, на основі чого сформульовано вимоги до сервісу – як функціональні (каталог контенту, авторизація, купівля фільмів), так і технічні (швидкість відгуку інтерфейсу та безпека збереження даних). Також обґрунтовано вибір технологічного стеку: обрано бібліотеку React для клієнтської частини, фреймворк NestJS для сервера, СУБД PostgreSQL для бази даних та мову TypeScript для надійної типізації коду.

Другий розділ присвячено проєктуванню системи. Виконано архітектурне розділення застосунку на клієнтську та серверну частини, базу даних і зовнішні сервіси. Розроблено схему бази даних (ERD) із таблицями для користувачів, фільмів, жанрів, акторів, відгуків та транзакцій. Повністю описано логіку RESTful API, створено класи DTO з декларативною валідацією через class-validator, а також продумано архітектуру інтеграції з платіжним сервісом Stripe.

У третьому розділі описано безпосередню програмну реалізацію проєкту. Фронтенд побудовано на базі збирача Vite та React, де за маршрутизацію відповідає React Router DOM, за асинхронні запити до сервера – Axios, а за роботу з формами та валідацію даних – Formik та Yup. Інтерфейс отримав оптимізовану темну тему та підтримку багатомовності за допомогою

i18next. Бекенд реалізовано на фреймворку NestJS із використанням TypeORM. Безпеку доступу забезпечено через безсесійний механізм JWT-токенів та хешування паролів за допомогою алгоритму bcryptjs. Додатково налаштовано сервіс автоматичної генерації PDF-чеків через pdfkit та архітектуру завантаження медіафайлів у хмарне сховище AWS S3.

Практична цінність проєкту «StreamWave» полягає в тому, що створено готову, робочу основу для реального відеосервісу. Проєктна архітектура дозволяє легко масштабувати застосунок у майбутньому: наприклад, додати адміністративну панель керування контентом, підключити рекомендаційні алгоритми, реалізувати повноцінний стримінг відео високої якості (HLS/DASH) або впровадити Refresh токени для підвищення рівня безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Documentation. URL: https://react.dev/learn (дата звернення: 12.01.2026).

2. Vite Guide. URL: https://vite.dev/guide/ (дата звернення: 12.01.2026).

3. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 12.01.2026).

4. NestJS Documentation. URL: https://docs.nestjs.com/ (дата звернення: 13.01.2026).

5. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 13.01.2026).

6. TypeORM Documentation. URL: https://typeorm.io/ (дата звернення: 13.01.2026).

7. Stripe Docs. Accept a payment. URL: https://docs.stripe.com/payments/accept-a-payment (дата звернення: 19.01.2026).

8. AWS SDK for JavaScript v3 Documentation. URL: https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/ (дата звернення: 15.01.2026).

9. React Router Documentation. URL: https://reactrouter.com/ (дата звернення: 18.01.2026).

10. Formik Documentation. URL: https://formik.org/docs/overview (дата звернення: 18.01.2026).

11. Yup Documentation. URL: https://yup-docs.vercel.app/docs/intro (дата звернення: 18.01.2026).