# КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики Кафедра теорії та технології програмування

Курсова робота

за спеціальністю 122 Комп’ютерні науки

на тему:

**БРОКЕРИ ПОВІДОМЛЕНЬ: ПРИКЛАДИ ВИКОРИСТАННЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ РЕАЛІЗАЦІЙ**

Виконав студент 3-го курсу

Ходаков Максим Олегович

Науковий керівник:

асистент,

Поліщук Наталія Володимирівна

(підпис)

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

« » 2024 р. протокол №

Завідувач кафедри М. С. Нікітченко

(підпис)

Київ – 2024

# РЕФЕРАТ

Обсяг роботи \_ сторінка, \_ ілюстрації, \_ таблиці, \_ джерела посилання

та \_ додаток.

БРОКЕРИ ПОВІДОМЛЕНЬ, ВЕБ-ЗАСТОСУНОК, SPRING, KAFKA, RABBITMQ, БАЗА ДАНИХ, JPA, MARIADB, .

Об’єктом дослідження є сфера брокерів повідомлень та їх реалізацій.

Мета кваліфікаційної роботи: дослідити популярні брокери повідомлень та розробити Spring-застосунок аудиту та обліку зоопарків по всьому світу з використанням Kafka та RabbitMQ використовуючи мікросервісну архітектуру.

Методи розроблення: розробка програмного продукту на основі поєднання окремих сервісів Producer, Consumer та, зрештою, Zoo сервіс для управління основними сутностями предметної області. Інструменти розроблення: інтегроване середовище розробки IntelliJ IDEA, мова програмування Java, Spring фреймворк, брокери повідомлень: Kafka, RabbitMQ, СКБД MariaDB.

Результати курсової роботи: було проведено огляд та аналіз існуючих рішень на ринку, досліджено основні етапи розробки мікросервісів та застосування брокерів повідомлень, програмні засоби для ефективної інтеграції брокерів повідомлень в мікросервіси. Також було спроектовано та розроблено веб-застосунок «Zoo». Веб-застосунок «Zoo» можна використовувати як і сервіс для внутрішнього аудиту зоопарків так і організації відмовостійкої до високих навантажень структури цілої трансконтинентальної мережі.

# ЗМІСТ

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programming Interface; СКБД – Система Керування Базами Даних; SQL – Structured Query Language;

HTTP – HyperText Transfer Protocol; ORM – Object-Relational Mapping;

CSS – Cascading Style Sheets;

HTML – HyperText Markup Language; JS – JavaScript;

IDE – Інтегроване Середовище Розробки; JPA - Java Persistence API;

CRUD – Create, Read, Update, Delete БД – База Даних;

# ВСТУП

**Оцінка сучасного стану об’єкта розробки.** В наш час існує багато різних систем аудиту та ще більша кількість даних. Надійність, розподіленість та асинхронність виконання – це ті основні властивості аналітичної системи що надають можливість ефективно опрацьовувати велику кількість даних. Саме застосування брокерів повідомлень в веб-застосунку забезпечить виконання всіх перелічених властивостей ефективних аналітичних систем.

Інтернет як інструмент комунікації, допомагає вирішувати питання з електронного обліку. Але дедалі частіше зустрічаються вимоги по продуктивності систем, що в свою чергу підіймають питання щодо розподілення відповідальності різних сервісів систем. Тому розробка такої системи з використанням брокерів повідомлень є затребуваною роботою.

**Актуальність роботи та підстави для її виконання**. В наш час глобальних кліматичних проблем, війн та індустріального забруднення задача збереження природної флори та фауни є однією з найбільш пріоритетною. Саме тому трансконтинентальний облік та аудит тварин, зоопарків та країн перебування є затребуваною системою. А поєднання надійної, розподіленої мікросервісної веб-системи для цієї задачі буде цікавим продуктом яким зацікавляться як великі корпорації так і міжнародні організації.

**Мета й завдання роботи.** Метою курсової роботи є розробка веб-застосунку «Zoo», який буде мікросервісом з трьома різними сервісами для два з яких призначенні для обробки тварин, та використанням таких брокерів повідомлень як: Kafka, RabbitMQ. Для досягнення цієї мети було поставлено такі завдання:

* дослідити існуючі аудиторські сервіси для зоопарків та тварин
* розглянути, дослідити можливості інтеграції брокерів повідомлень та Spring-фреймворку
* побудувати модель бази даних для потреб предметної області
* розробити вимоги до сервісів
* розробити основні сторінки до «Zoo» та «Producer» сервісів
* реалізація та покриття тестами основних сервісів

**Об’єкт, методи й засоби розроблення.** Об’єктом дослідження веб-сервісу

«Zoo» є процес створення файлами тварин, створення та ведення обліку зоопарків. Для «Producer» веб-сервісу об’єктом дослідження є використання брокерів повідомлень для опрацювання тварин, також, для «Consumer» веб-сервісу об’єктом дослідження є валідація та завантаження в систему даних.

Предметом роботи є власний розроблений застосунок.

В якості інструменту створення програмного засобу було обрано IntelliJ IDEA IDE – інтегроване середовище розробки мовою програмування Java від компанії JetBrains [1]. В якості СКБД було обрано MariaDB, а в якості генерації сторінок було обрано бібліотеку Thymeleaf.

**Сфера застосування**. Програмний продукт «Zoo» можна використовувати в cфері обліку та аудиту зоопарків та тварин: для оновлення та зберігання актуальних даних про тварин, зоопарків та країн. Перегляд зоопарків та їх вмісту в різних країнах та на різних континентах світу.

# РОЗДІЛ 1. ОГЛЯД ТА ПОРІВНЯННЯ НАЯВНИХ НА РИНКУ РІШЕНЬ

На сьогодні існує велика кількість веб-додатків, які пропонують послуги перегляду, та управління тваринками в зоопарках. Давайте розглянемо найпопулярніші з них, які надають ту чи іншу послугу.

## Central Park Zoo

Central Park Zoo - веб-сайт для центрального парку міста Нью-Йорк, США. На даному сайті розміщена актуальна інформація щодо складу зоопарку також додаткова інформація про тваринок наявна для навчальних цілей. Хоча сам зоопарк працює з 1934 року, сайт працює тільки з 2004 використовуючи мову PHP.

A seal in a zoo exhibit

Description automatically generated

Рисунок 1.1 - Головна сторінка centralparkzoo.com

Однією з головних особливостей Central Park Zoo є те, що редагувати інформацію в базі даних не може будь-який користувач Інтернету, навіть якщо він увійшов в систем. Cеред переваг можна виділити простий у використанні і гарний інтерфейс, список всіх тваринок та доступна інформація для них.

## species360

species360 - це найбільша некомерційна організація-мережа захисту дикої природи в світі. Основна діяльність цієї організації це Зоологічна система управління даними. Окрім цього мережа має власний сайт, де можна в залежності від вибраного типу запитуваних даних надає бажану інформацію. Сайт має мінімалістичний і простий в користуванні інтерфейс.

A screenshot of a computer

Description automatically generated

Рисунок 1.2 – Головна сторінка species360

## Деякі інші рішення та їх порівняння

Також існує багато інших конкурентів, які надають обидві приблизно такі самі послуги. Наприклад «Київський зоопарк» надає як послуги перегляду тваринок та інформацію про них, покупки білетів та основних новин. Але наданий функціонал не надає можливості для користувачів отримувати інформацію за межами міста, очевидно що і за межами країни. Також відсутній функціонал масового завантаження сутностей системи. San Diego Zoo Wildlife Alliance це теж некомерційна міжнародна мережа в сфері дикої природи. Ця система не надає доступу для управління і не надає надійний при високих навантаженнях API.

Розглянувши ці застосунки, можна побачити, що вони надають один з двох варіантів послуг: або мережа перегляду, або ж сервіси бронювання квитків. Але не надають доступ для управління сутностями та масовому відмовостійкому завантаженню файлів з валідацією сутностей. Задля забезпечення безперебійної роботи функціонування сервісу. Ось наведені основні проблеми з трансконтинентальними зоопарками:

Культурні різниці: Управління зоопарками в різних країнах може варіюватися в залежності від культурних традицій, економічних умов та законодавства. Підходи до догляду за тваринами, експозиційним дизайном та відношення до охорони природи можуть значно відрізнятися.

Законодавство та регулювання: Різні країни мають власні норми і правила, що стосуються утримання та догляду за тваринами у зоопарках. Наявність та рівень обмежень, які ставляться перед зоопарками, можуть різнитися від країни до країни.

Біорізноманіття: Зоопарки мають велику відповідальність щодо збереження біорізноманіття. Проблеми, такі як вимирання видів, зміна клімату та знищення середовища мешкання, можуть впливати на тваринний склад зоопарків. Координація зусиль між зоопарками по всьому світу для збереження видів може бути складною.

Транспортування та логістика: Пересування тварин між зоопарками на різних континентах може бути складним і дорогим процесом. Вимагається дотримання високих стандартів безпеки та добробуту тварин під час перевезень.

Мовні бар'єри та комунікація: Управління зоопарками з різних країн може стикатися з проблемами в комунікації через різні мови та культурні особливості. Це може призвести до недорозумінь і ускладнити спільні проекти та ініціативи.

Фінансування: Операція та розвиток зоопарків потребують значних інвестицій. Фінансування може бути проблемою для трансконтинентальних систем управління зоопарками, особливо у випадку, коли існують великі різниці в економічному стані різних країн.

Соціальні та етичні питання: Управління зоопарками включає в себе різноманітні етичні питання, такі як права тварин, експлуатація та навіть моральність самої концепції зоопарків. Думки щодо цих питань можуть значно розрізнятися в різних країнах і серед різних груп населення

# РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА СИСТЕМ

## Spring-фреймворк

Spring - один з найбільш широко використовуваних фреймворків для розробки застосунків ентерпрайз рівня, що забезпечує продуману модель конфігурації та реалізації. Метою створення даного фреймворку сприяло бажання спростити розробку додатків на популярному на той час Java EE (Java Platform, Enterprise Edition) [5] стеку технологій, який на той момент був дуже складний і не завжди зручний у використанні та підтримки.

На відміну від інших фреймворків, Spring зосереджується на кількох областях програми та надає широкий спектр функцій. Одні з головних особливостей фреймворка Spring є: ін'єкція залежностей та інверсія контролю. Це допомагає розроблювати додатки швидше та дозволяє ефективно дотримуватися найкращих практик розробки сучасного програмного забезпечення.

Хоча фреймворк Spring зосереджується на наданні гнучкості, Spring Boot [6] має на меті скоротити кількість коду та надати простіший спосіб розробки додатків. Завдяки аспектно-орієнтованому підходу в обробці конфігураційних анотацій, Spring Boot скорочує час розробки програми. Це допомагає зекономити час на створення конфігурацій проєкту. Тому в якості серверної частини було прийнято рішення використати саме Spring Boot.

**Архітектура Spring Boot.** Spring Boot дотримується багаторівневої архітектури, кожен рівень якої взаємодіє один з одним. Основною метою створення та існування Spring Boot є прибирання XML і параметрів конфігурації на основі анотацій із програми.

Spring Boot складається з наступних чотирьох рівнів [7]:

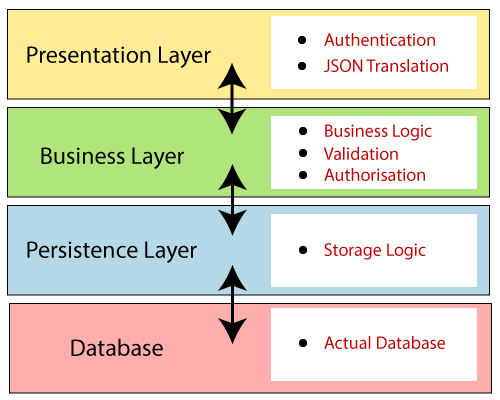


Рисунок 2.1 - Рівні Spring Boot

1. **Presentation Layer.** Рівень презентації є найвищим рівнем архітектури Spring Boot. Він складається з фронтенд частини програми. Він обробляє HTTP- запити та відповідає за сереалізацію та десереалізацію об’єктів Java. Після того, як він виконує аутентифікацію запиту, він передає його на наступний рівень.
2. **Business Layer**. Цей рівень містить бізнес-логіку, яка керує основними функціями програми. Подібно до прийняття рішень, обчислень, оцінок та обробки даних, що передаються між двома іншими рівнями.
3. **Persistence Layer**. Рівень збереження містить всю логіку зберігання бази даних. Він відповідає за перетворення бізнес-об'єктів у рядок бази даних і навпаки.
4. **Database Layer**. Рівень бази даних містить усі бази даних, такі як MySql, PostgreSql тощо. Цей рівень може містити декілька баз даних. Він відповідає за виконання CRUD операцій.

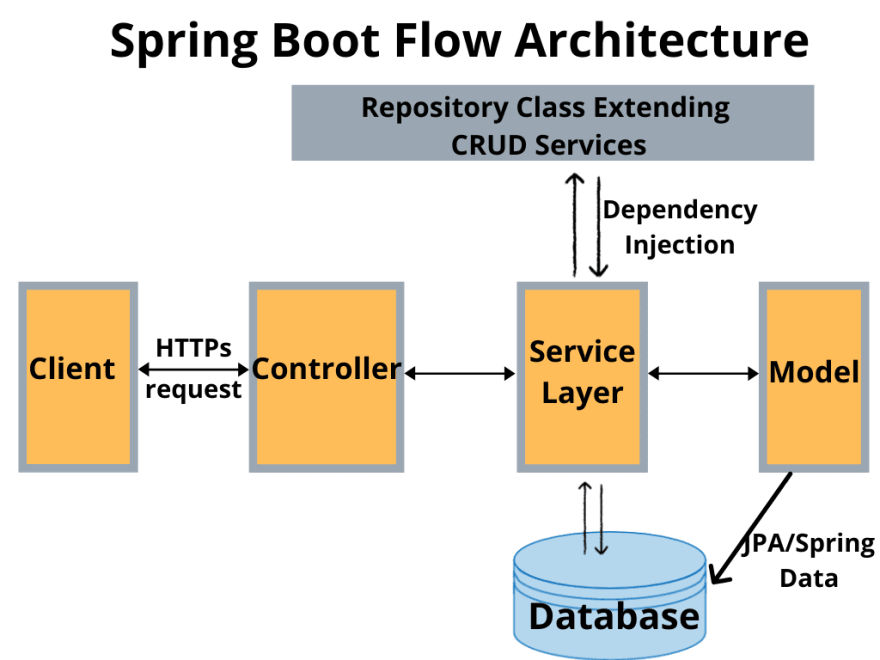


Рисунок 2.2 - Архітектура Spring Boot [7]

* Клієнт робить HTTP-запит (GET, PUT, POST, DELETE тощо)
* HTTP-запит пересилається до контролера. Контролер відображає запит. Він обробляє дескриптори та викликає логіку сервера.
* Бізнес-логіка виконується на рівні сервісу. Spring Boot виконує всю логіку над даними бази даних, які зіставляються з класом моделі Spring Boot через Java Persistence Library(JPA).
* Контролер повертає представлення (JSP, Thymeleaf, JSON тощо) для відтворення у браузері.

## MariaDB

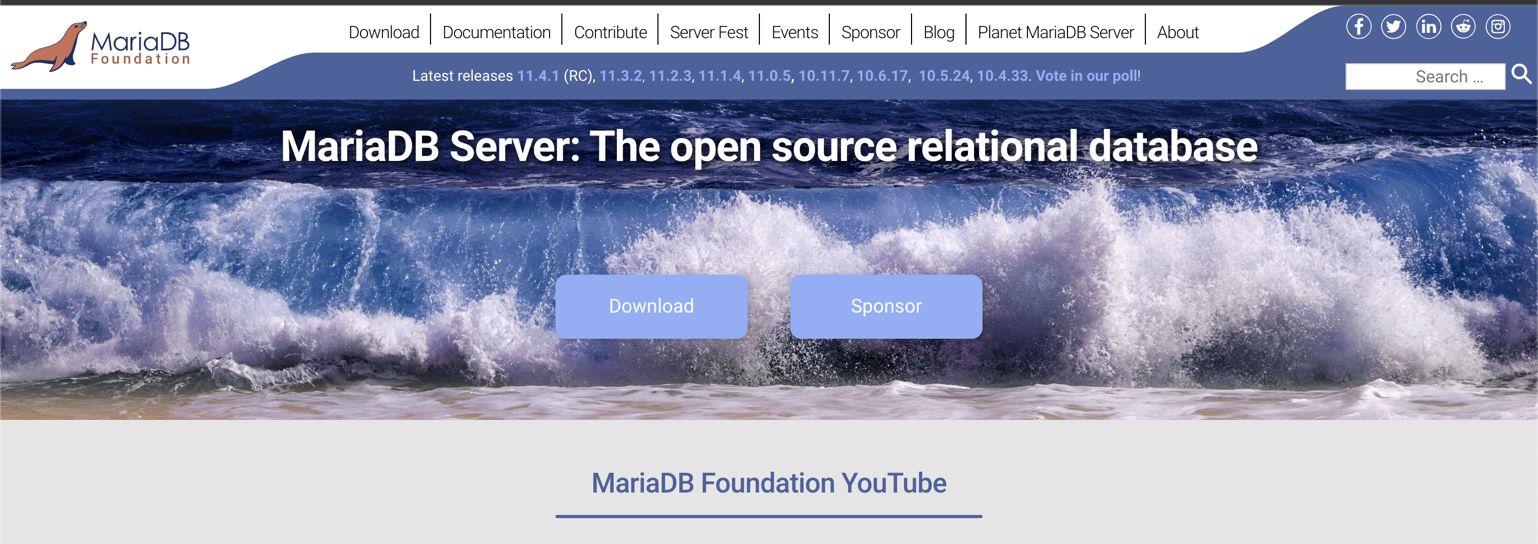
В ролі СКБД було вирішено взяти MariaDB.

MariaDB — це реляційна система управління базами даних (СУБД), яка походить від MySQL і була створена як її вільний відкритий форк. Ця система з'явилась у відповідь на побоювання, що MySQL може перейти у закрите джерело після його купівлі компанією Oracle. Використання MariaDB має кілька переваг, які можуть зробити її більш привабливою в порівнянні з іншими реляційними БД.

**Переваги MariaDB:**

1. Сумісність з MySQL: MariaDB повністю сумісна з MySQL, що означає, що вона може безпроблемно замінити MySQL у більшості випадків. Це робить перехід на MariaDB відносно простим для існуючих систем, що використовують MySQL.
2. Відкрите джерело: MariaDB продовжує залишатися проектом з відкритим джерелом, надаючи гнучкість і контроль над кодом, якого може не бути в комерційних продуктах.
3. Розширена функціональність: MariaDB включає додаткові можливості, які не містяться в стандартному MySQL, включаючи нові типи індексів, підтримку додаткових сховищ даних (наприклад, Aria, ColumnStore) і розширені можливості оптимізації запитів.
4. Підтримка спільноти та інновації: MariaDB підтримується великою спільнотою розробників, яка постійно працює над покращенням продукту і додаванням нових функцій.
5. Краща продуктивність та оптимізації: MariaDB часто демонструє кращу продуктивність в порівнянні з MySQL, особливо у випадках, що використовують новіші оптимізації та технології.
6. Функції для великих навантажень: MariaDB має багато властивостей, спрямованих на оптимізацію для великих обсягів даних і високих навантажень, включаючи покращені реплікації та кластеризації.

**Порівняння з іншими реляційними СКБД.** В порівнянні з PostgreSQL: PostgreSQL часто вважається більш потужним у плані функціональних можливостей, особливо для складних і великих додатків, що потребують глибокої транзакційної логіки та обробки. Водночас, MariaDB може виявитися кращим вибором для додатків, що потребують високої сумісності з MySQL та швидкої та ефективної впроваджуваності. В порівнянні з Oracle або Microsoft SQL Server: Ці комерційні продукти надають розширені функції безпеки та управління, які можуть бути необхідні для деяких корпоративних середовищ. Проте, вони є платними, і їх вартість може бути значною. MariaDB є безкоштовною альтернативою, яка добре підходить для багатьох бізнес-сценаріїв, особливо де важлива вартість та відкритість коду. В порівнянні з SQLite: SQLite ідеально підходить для легких додатків та вбудованих систем, де база даних повинна бути компактною та самодостатньою. MariaDB краще підходить для більш масштабних застосунків, що потребують повномасштабної реляційної СКБД.

 Рисунок 2.3 – Офіційний сайт MariaDB

## Amazon S3

В якості збереження зображень тварин, прапорів держав та усіх завантажених файлів було використано Amazon S3 сервіс.

Amazon S3 (Simple Storage Service) — це об'єктне сховище, створене компанією Amazon Web Services (AWS). Цей сервіс надає масштабоване, надійне, безпечне та високодоступне сховище даних в інтернеті. Він дозволяє зберігати та отримувати будь-яку кількість даних з будь-якого місця в інтернеті. S3 використовується різними типами клієнтів, від стартапів до великих корпорацій, для зберігання та обробки даних різного роду, включаючи веб-сайти, мобільні застосунки, резервне копіювання та відновлення, архівування, ентерпрайз-додатки, IoT-пристрої та багато іншого.

**Основні властивості сервісу.** Об'єктне сховище: Amazon S3 зберігає дані у форматі об'єктів. Кожен об'єкт складається з файлу даних (будь-якого типу і розміру до 5 ТБ), метаданих (набір пар ключ-значення, які описують дані) та унікального ідентифікатора (ключа). Бакети: Дані в Amazon S3 зберігаються у контейнерах, які називаються "бакетами". Кожен бакет має унікальну, глобально унікальну назву та може містити необмежену кількість об'єктів. Доступність та надійність: Amazon S3 гарантує високу доступність і надійність зберігання даних, розподіляючи копії об'єктів по різних фізичних місцях в рамках інфраструктури AWS. Безпека: S3 надає розширені можливості управління доступом, включаючи політики бакетів, ACL (списки контролю доступу) та шифрування даних в режимі спокою та під час передачі. Масштабування: Amazon S3 може автоматично масштабуватися для обробки пікових навантажень, дозволяючи зберігати та обробляти необмежену кількість даних. Менеджмент даних: S3 пропонує можливості життєвого циклу даних, що дозволяє автоматично переміщувати дані між різними рівнями сховища (наприклад, від часто використовуваного сховища до архівного) або видаляти їх після закінчення вказаного терміну.

**Переваги Amazon S3 над конкурентами.** Надійність і масштабованість: Amazon S3 забезпечує 99.999999999% (11 дев'яток) стійкості, що означає крайню надійність зберігання. Гнучкість ціноутворення: AWS пропонує конкурентоспроможне ціноутворення з варіантами, які включають різні рівні доступності і витрат на зберігання. Глобальне покриття: Завдяки глобальній інфраструктурі AWS, дані можуть бути збережені географічно близько до кінцевих користувачів для зменшення затримок та збільшення швидкості відповіді. Інтеграція з AWS: S3 ідеально інтегрується з іншими сервісами AWS, такими як Amazon EC2, Amazon RDS, Amazon Redshift та багатьма іншими, що забезпечує єдиний досвід роботи з різними хмарними сервісами. Інноваційні рішення: AWS регулярно вводить нові функції та покращення для S3, наприклад, S3 Glacier для довгострокового архівування даних або S3 Intelligent-Tiering для автоматичного оптимізування витрат на зберігання в залежності від частоти доступу до даних.

A screenshot of a website

Description automatically generated

Рисунок 2.4 – офіційний сайт Amazon щодо S3 сервісу

## Apache POI

Для обробки .xlsx файлів було використано open-source бібліотеку POI.

Apache POI — це відкрита бібліотека, яка надає Java API для роботи з різними форматами файлів Microsoft Office. Хоча вона підтримує кілька офісних форматів, таких як Word і PowerPoint, однією з її найпопулярніших функцій є можливість читання та запису файлів Excel. Apache POI включає кілька компонентів, але два головні, які використовуються для роботи з Excel, — це HSSF (Horrible Spreadsheet Format) і XSSF (XML Spreadsheet Format).

**Можливості Apache POI.** Читання та запис файлів: Apache POI може використовуватися для генерації файлів Excel з нуля, зчитування існуючих файлів та модифікації цих файлів. Обробка формул: Бібліотека може обчислювати формули, вбудовані у клітинки Excel. Маніпуляції з форматуванням: Вона дозволяє налаштовувати стилі клітинок, такі як шрифти, кольори, обрамлення тощо. Підтримка картинок та графіки: Можливість вставляти зображення і створювати діаграми в документах Excel. Інтеграція з багатьма іншими Java бібліотеками: Легко інтегрується з веб-фреймворками для виводу файлів Excel як відповіді на HTTP запити.

**Переваги Apache POI.** Відкрите джерело: Як проект з відкритим джерелом, Apache POI не вимагає ліцензійних витрат і дозволяє повний контроль над кодом. Міцна спільнота: Існує велика спільнота розробників, яка підтримує і постійно оновлює бібліотеку. Платформонезалежність: Як Java-бібліотека, Apache POI може використовуватися на будь-якій платформі, де працює Java.

## RabbitMQ

## RabbitMQ — це популярне рішення брокера повідомлень, яке дозволяє застосункам масштабуватися та інтегруватися за допомогою асинхронної передачі повідомлень. Цей інструмент забезпечує надійний механізм для обміну даними між компонентами системи у вигляді повідомлень. RabbitMQ написаний на Erlang і підтримує багато протоколів, зокрема AMQP (Advanced Message Queuing Protocol).

RabbitMQ використовує складну, але дуже гнучку архітектуру, яка дозволяє реалізувати різноманітні сценарії обміну повідомленнями. Ця архітектура забезпечує високу надійність, масштабованість та гнучкість у маршрутизації повідомлень. Ось основні компоненти та принципи, які лежать в основі архітектури RabbitMQ:

Producer: Виробник (producer) відправляє повідомлення до черг у RabbitMQ. Producer може вказати, в яку чергу відправляти повідомлення, але це зазвичай визначається маршрутизацією на брокері.

Queue (Черга): Черги в RabbitMQ — це місця, де зберігаються повідомлення до того, як їх буде оброблено. Кожна черга ізольована від інших, і повідомлення обробляються в порядку їх надходження.

Consumer: Споживач (consumer) отримує повідомлення з черги. В одній черзі може бути кілька споживачів, які обробляють повідомлення паралельно, забезпечуючи тим самим високу продуктивність обробки.

Exchange (Обмінник): Обмінник приймає повідомлення від producer'ів і вирішує, в які черги вони повинні бути відправлені. Обмінники підтримують різні типи маршрутизації, включаючи пряму, тематичну, вентиляторну (fan-out) та заголовки (headers).

Binding (Прив'язка): Прив'язка визначає правила, за якими повідомлення в обмінниках відправляються до черг. Прив'язка може включати маршрутизаційний ключ або шаблон, який використовується для фільтрації повідомлень.

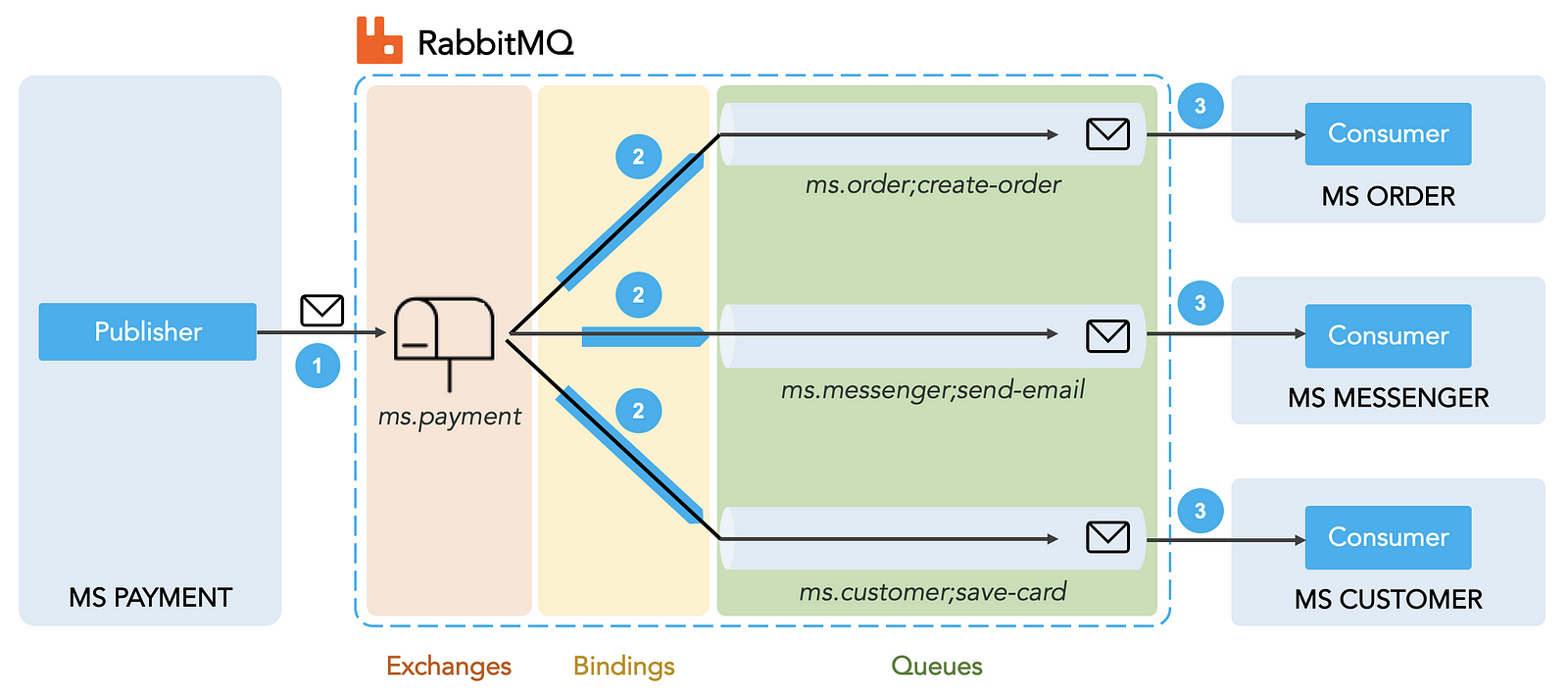
**Архітектурні принципи.**

Масштабованість: RabbitMQ може масштабуватися горизонтально (кластеризація) та вертикально (додавання ресурсів). Кластеризація дозволяє брокерам повідомлень працювати разом, забезпечуючи високу доступність і балансування навантаження.

Надійність: RabbitMQ забезпечує надійність через підтвердження повідомлень, транзакційність, та дзеркальне відображення черг. Це означає, що повідомлення не втрачаються і можуть бути доставлені навіть у разі збоїв.

Багатопротокольна підтримка: RabbitMQ підтримує кілька протоколів для передачі повідомлень, включаючи AMQP, MQTT, STOMP та інші, що робить його сумісним з широким спектром клієнтських застосунків.

**Порівняння з конкурентами**. RabbitMQ часто порівнюють з такими системами, як Kafka, ActiveMQ та інші хмарні послуги (наприклад, AWS SQS). В порівнянні з Kafka, який краще підходить для обробки великих потоків даних, RabbitMQ краще використовувати для гарантованої доставки повідомлень та складної маршрутизації. Порівняно з ActiveMQ, RabbitMQ часто вважається більш продуктивним та легшим у масштабуванні. Таким чином, RabbitMQ є потужним інструментом для реалізації комплексних сценаріїв обміну повідомленнями, який забезпечує високий рівень гнучкості і масштабування для сучасних додатків.



## Рисунок 2.5 – архітектура RabbitMQ

## Apache Kafka

Apache Kafka — це розподілена платформа для обробки потоків даних, яка використовується для побудови високопродуктивних рішень у сфері обробки повідомлень та потоків даних в режимі реального часу. Розроблений спочатку в LinkedIn, Kafka широко використовується для збору та обробки великих обсягів подій.

**Архітектура Apache Kafka.** Kafka має декілька ключових компонентів, які спільно забезпечують його потужність і гнучкість:

Broker: Kafka працює як кластер з одного або декількох серверів, які називаються брокерами. Кожен брокер може обслуговувати тисячі клієнтів та мільярди повідомлень.

Topic: Повідомлення в Kafka зберігаються у категоріях, відомих як теми. Тему можна розділити на декілька "партицій" (сегментів), які можуть бути розподілені між кількома брокерами.

Partition: Кожна тема в Kafka розділена на партиції. Розподіл даних по партиціях дозволяє Kafka масштабуватися горизонтально, зберігаючи дані та забезпечуючи паралельну обробку.

Producer: Виробники відправляють повідомлення до тем Kafka. Вони відповідають за вибір партити в темі, до якої буде відправлено повідомлення.

Consumer: Споживачі читають повідомлення з визначених тем. Вони можуть читати дані з партицій у паралелі з іншими споживачами, що знаходяться в одній споживацькій групі.

Zookeeper: Kafka використовує Zookeeper для управління конфігурацією та координації кластера. Zookeeper відстежує стан брокерів та споживачів, а також зберігає інформацію про теми та партиції.

**Порівняння з конкурентами.**

RabbitMQ: Використання: RabbitMQ широко використовується для забезпечення надійної доставки повідомлень у традиційних веб-додатках, де потрібна складна маршрутизація і транзакційність. Масштабованість: Kafka значно краще масштабується для обробки великих потоків даних завдяки своїй архітектурі з розділенням даних на партити.

Apache ActiveMQ: Функціональність: ActiveMQ пропонує більш традиційні функції JMS і підтримку транзакцій, що є важливим для деяких ентерпрайз-додатків. Продуктивність: Kafka перевершує ActiveMQ у випадках великого обсягу данних і паралельної обробки повідомлень.

Amazon Kinesis: Хмарна інтеграція: Kinesis є хмарним рішенням, яке натісно інтегровано з іншими сервісами AWS, що може спростити розгортання та управління для користувачів AWS. Функціональність: Хоча Kinesis пропонує схожі на Kafka можливості для обробки потоків даних, вона має обмеження на обсяги та швидкість обробки, які можуть бути критичними для деяких застосунків.

Apache Kafka особливо ефективна у сценаріях, де потрібно обробляти величезні обсяги вхідних даних в реальному часі, забезпечуючи високу пропускну здатність і надійність. Вона ідеально підходить для побудови розподілених систем обробки подій, великих логів або агрегації даних з різних джерел. Тому саме використання Apache Kafka та RabbitMQ було обрано для імплементації цього проєкту.

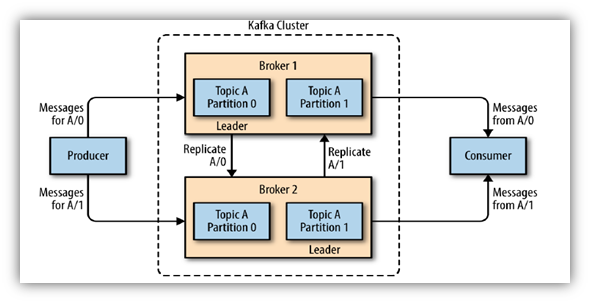


Рисунок 2.6 – архітектура Apache Kafka

# РОЗДІЛ 3. ПРИЗНАЧЕННЯ І ЦІЛІ РОЗРОБКИ ЗАСТОСУНКУ

## Призначення застосунку

Призначенням веб-системи «Zoo» є реалізація сервісу управління трансконтинентальною системою зоопарків та тварин в них, можливість оновлювати видаляти та створювати дані щодо всіх аспектів трансконтинентальної системи, надання інформації про зоопарки, їх тварини та додаткову інформацію про континенти та зоопарків на них відповідно. Також надання функціоналу відмовостійкого завантаження файлів з величезною кількістю інформації про тварини за допомогою розподіленої системи.

Для вирішення поставленої задачі було проведено:

* + аналіз методів і моделей, що застосовуються для розв’язання задач створення комплексних веб-систем
  + аналіз наявних програмних засобів природничо-інформаційних мереж і веб-застосунків популярних зоопарків
  + проектування та програмну реалізацію системи «Zoo»
  + апробація розробленого веб-застосунку

## Цілі створення застосунку

Веб-застосунок «Zoo» розроблено з метою забезпечення:

* + зручного пошуку бажаних зоопарків та інформацію про них на обраному континенті
  + слідкування за новинками та популярними тваринками в межах зоопарку
  + перегляду детальної інформації про тварини та зоопарки
  + управління сутностями зоопарків та наповнення їх тваринами
  + слідкування за статистикою завантаження по брокерам повідомлень
  + розвитку культури зберігання тварин шляхом аналізу наявної кількості певних видів тварин, та привертання уваги до тварин що знаходяться під загрозою вимирання
  + безпечне масове завантаження тварин в спеціальних файлових форматах

## Архітектурні особливості системи

З огляду на поставлені задачі в системі «Zoo» були виділені наступні функціональні підсистеми:

* + підсистема адміністратора, призначена для CRUD операцій з основними сутностями системи та надає функціональний доступ до інших підсистем
  + підсистема продюсера, призначена для запуску асинхронною взаємодії між мікросервісами за допомогою вище згаданих RabbitMQ або Apache Kafka
  + підсистема консюмера, призначена для валідації та завантаження результатів обробки файлів в БД системи

## Вимоги до функцій, які виконуються програмою

Перелік основних функцій адміністратора наведено в табл. 1.

**Таблиця 1**. Перелік функцій та задач адміністратора що підлягають автоматизації

|  |  |
| --- | --- |
| **Функція** | **Задача** |
| Управління країнами | Перегляд країн, інформації про них |
| Редагування країн |
| Видалення країн та всіє інформації що стосується країни |
| Управління зоопарками | Створення, редагування, видалення зоопарків |
| Створення, редагування, видалення зв’язків з країнами |
| Створення, редагування, видалення зв’язків з тваринами |
| Управління тваринами | Створення, редагування та видалення тварин |
| Створення, редагування та видалення зв’язків з зоопарками |
| Створення, редагування та видалення зв’язків з країнами |
| Перегляд всіх тварин за фільтрацією по імені  та інших критеріїв |
| Управління комунікативними елементами | Перегляд результатів валідації завантаження файлів |
| Перегляд результатів асинхронного виклику інших функціональних підсистем |

Перелік основних функцій користувача наведено в табл. 2.

**Таблиця 2.** Перелік функцій та задач користувача системи що підлягають автоматизації

|  |  |
| --- | --- |
| **Функція** | **Задача** |
| Реєстрація | Введення короткої інформації про себе при логіні (логін та пароль) |
| Введення повної інформації про себе при реєстрації |
| Вихід із системи |

## Технічні вимоги до застосунку

В результаті виконання практичної частини курсової роботи було розроблено застосунок який відповідає наступним вимогам:

* + застосунок розроблений використовуючи технології Spring Boot, Spring MVC, Spring Data, RabbitMQ, Apache Kafka, Thymeleaf
  + джерело даних є СКБД MariaDB
  + у застосунку наявна реєстрація облікового запису так і захищенні за протоколом OAuth2 API ендпоїнти
  + наявна пагінація, фільтрація та рядок пошуку
  + на всіх сторінках з формами присутня валідація як на стороні клієнта, так і на стороні сервера
  + застосунок коректно реагує на помилки і виключення різного роду
  + веб-додаток має простий і інтуїтивний у використанні інтерфейс
  + сайт коректно відображається в інтернет-браузерах Google Chome, Microsoft Edge, Mozilla Firefox, Opera.
  + cайт коректно відображається на мобільних пристроях

# РОЗДІЛ 4. ПОРІВНЯЛЬНИЙ АНАЛІЗ БРОКЕРІВ ПОВІДОМЛЕНЬ

## Порівняльний аналіз існуючих реалізацій

Брокери повідомлень відіграють критичну роль у сучасних розподілених системах, де вони виконують функцію посередників у передачі даних між компонентами системи. Ось детальніший аналіз чотирьох популярних рішень: Apache Kafka, RabbitMQ, ActiveMQ, і Redis.

**Apache Kafka**

**Архітектура**: Kafka розроблена як розподілена публікація-підписка на потоки даних, що підтримується на базі зберігання журналів. Це дозволяє їй ефективно обробляти великі потоки даних у реальному часі.

**Особливості**:

1. Підтримка збереження даних на диск, що гарантує високу надійність і дозволяє відновлювати дані після збоїв.
2. Висока пропускна здатність навіть при великих обсягах даних.
3. Широкі можливості для масштабування, включаючи горизонтальне розширення за допомогою додавання більше серверів.

**Використання**: Ідеально підходить для систем, де потрібна швидка обробка великих потоків даних, таких як логування подій, трекінг користувацької активності, реальні аналітичні платформи.

**RabbitMQ**

**Архітектура:** RabbitMQ використовує модель "посередник", яка дозволяє повідомленням між виробниками та споживачами переміщуватися через спеціалізований сервер.

**Особливості:**

1. Підтримка різноманітних протоколів зв'язку, включаючи AMQP, MQTT, STOMP.
2. Підтримує транзакційні повідомлення.
3. Висока надійність і доступність за рахунок кластеризації і дзеркального відображення черг.

**Використання**: Добре підходить для корпоративних додатків, де потрібна надійна передача повідомлень, таких як електронна комерція, фінансові операції.

**ActiveMQ**

**Архітектура:** ActiveMQ також підтримує модель "посередник", з особливим акцентом на підтримку JMS (Java Messaging Service).

**Особливості:**

1. Підтримка JMS сприяє інтеграції з багатьма корпоративними системами.
2. Підтримує клієнти для різних мов програмування.
3. Конфігурація високої доступності та кластеризації.

**Використання**: Хороший вибір для систем, що потребують інтеграції з Java-орієнтованими середовищами, такими як корпоративні веб-сервіси і додатки.

**Redis**

**Архітектура:** Redis служить не лише як брокер повідомлень, а й як структура даних в пам'яті, що дозволяє використовувати його для кешування та інших завдань, що вимагають швидкого доступу до даних.

**Особливості**:

1. Висока швидкість доступу до даних.
2. Підтримка різних типів даних, таких як рядки, списки, множини.
3. Використання публікації-підписки для обміну повідомленнями.

**Використання**: Ідеальний для сценаріїв, де потрібна миттєва реакція, таких як чати, реальні ігри та інші інтерактивні додатки.

## Обґрунтування вибору для реалізації проєкту

Використання Apache Kafka та RabbitMQ в одному додатку може бути виправдане у випадках, коли система вимагає комбінації високої пропускної здатності, гнучкості в управлінні повідомленнями, і здатності до масштабування, які обидва ці брокери повідомлень можуть забезпечити. Ось декілька ключових аргументів на користь одночасного використання обох технологій:

**Комбінація потоків даних у реальному часі та складних патернів повідомлень.** **Apache Kafka** ідеально підходить для: Обробки великих потоків даних: Kafka може обробляти мільйони повідомлень за секунду, що робить її вибором для систем, де потрібна висока пропускна здатність і миттєва обробка, наприклад, для логування подій, моніторингу системи, або обробки потокових даних з IoT пристроїв.

Довгострокове зберігання: Завдяки своїй архітектурі, Kafka дозволяє зберігати повідомлення на диску з можливістю їх відновлення, що важливо для задач, де дані потрібно аналізувати ретроспективно.

**RabbitMQ** використовується для: Гнучкості управління повідомленнями: RabbitMQ підтримує різноманітні патерни повідомлень, такі як request/reply, routing, та topics, що дає можливість тонко налаштовувати доставку повідомлень в залежності від бізнес-логіки.

Надійна доставка: RabbitMQ має вбудовані можливості для забезпечення надійності доставки повідомлень, такі як message acknowledgment і durable queues.

**Масштабування та відмовостійкість.** Використання Kafka та RabbitMQ одночасно може забезпечити розподіл навантаження та підвищити загальну відмовостійкість системи:

Kafka може виступати як основний канал для потоків великих даних, обробляючи і зберігаючи значні об'єми інформації з високою швидкістю.

RabbitMQ може бути використаний для критичних завдань, які вимагають гарантованої доставки та точного управління повідомленнями в складних сценаріях маршрутизації.

**Зменшення точки відмови**. Інтеграція обох систем зменшує ризики залежності від одного брокера повідомлень: Розподілені системи зазвичай виграють від використання багатокомпонентного підходу, де критичні завдання можуть бути ізольовані від потенційних точок відмови.

Саме через ці наведені причини інтеграція Kafka та RabbitMQ дозволяє створити більш надійну, масштабовану та високопродуктивну систему. Kafka може виконувати обробку великих потоків даних, тоді як RabbitMQ забезпечує гнучкість і точність управління меншими, але більш критичними потоками повідомлень. Це робить таку комбінацію ідеальною для складних розподілених систем, які вимагають швидкості, надійності, і адаптивності до мінливих бізнес-вимог.

# РОЗДІЛ 5. РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

## 5.2 Реалізація бази даних

## Логічна структура бази даних

База даних розробленого веб-застосунку складається з 21 основної таблиці, структура яких зображена на діаграмі класів (див. додаток А), та 9 таблиць, які відображають зв’язок «багато до багатьох». Опис кожної з таблиць наведено у табл. 3.

**Таблиця 3.** Короткий опис таблиць бази даних

|  |  |  |
| --- | --- | --- |
| **Номер** | **Таблиця** | **Опис** |
| 1 | user | Таблиця для збереження інформації про  користувачів. |
| 2 | confirmation\_token | Таблиця для збереження інформації про токен  для підтвердження облікового запису по пошті |
| 3 | password\_reset\_token | Таблиця для збереження інформації про токен  для відновлення паролю |
| 4 | company | Таблиця для збереження інформації про  кінокомпанії |
| 5 | movie | Таблиця для збереження інформації про  кінокартини |
| 6 | image\_movie | Таблиця для збереження інформації про  зображення кінокартин |
| 7 | star | Таблиця для збереження інформацію про  персон (акторів чи режисерів) |
| 8 | cinema | Таблиця для збереження інформації про  кінотеатр |
| 9 | image\_cinema | Таблиця для збереження інформації про  зображення кінотеатрів |
| 10 | hall | Таблиця для збереження інформації про  кінозали |
| 11 | row | Таблиця для збереження інформації про ряди  кінозалу |
| 12 | seat | Таблиця для збереження інформації про місця  кінозалу |
| 13 | seance | Таблиця для збереження інформації про сеанси  фільмів в кінотеатрах |

|  |  |  |
| --- | --- | --- |
| 14 | seat\_reserved | Таблиця для збереження інформації про  зарезервовані місця |
| 15 | reservation | Таблиця для збереження інформації про  бронювання |
| 16 | ticket | Таблиця для збереження інформації про  продані квитки |
| 17 | comment | Таблиця для збереження інформації про  коментарі користувачів до кінокартин |
| 18 | contact\_us\_message | Таблиця для збереження інформації про  надіслані запитання від користувачів |
| 19 | rating | Таблиця для збереження інформації про оцінки  користувачів до кінокартин |
| 20 | favourite | Таблиця для збереження інформації про  улюблені кінокартини користувачів |
| 21 | watch\_list | Таблиця для збереження інформації про список  бажаного до перегляду користувача |
| 22 | user\_role | Таблиця для збереження інформації про ролі  користувачів |
| 23 | user\_subscriptions | Таблиця для збереження інформації про  підписки на інших користувачів |
| 24 | movie\_cast | Таблиця для збереження інформації про  акторів фільмів |
| 25 | movie\_genres | Таблиця для збереження інформації жанрів  фільмів |
| 26 | movie\_companies | Таблиця для збереження інформації  кінокомпаній фільмів |
| 27 | movie\_countries | Таблиця для збереження інформації країн  виробників фільмів |
| 28 | movie\_directors | Таблиця для збереження інформації про  режисерів фільмів |
| 29 | comment\_likes | Таблиця для збереження інформації  користувачів, які лайкнули коментар |
| 30 | comment\_dislikes | Таблиця для збереження інформації  користувачів, які дизлайкнули коментар |

## Опис таблиць бази даних програми

Таблиці складають основу бази даних – саме в них зберігаються всі дані. Передусім, було спроектовано структуру кожної таблиці. У таблицях 4-22 наведено детальну інформацію про структуру основних таблиць БД (ім’я, тип та опис поля).

**Таблиця 4.** Опис таблиці user

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор користувача |
| username | varchar | Унікальний юзернейм |
| first\_name | varchar | Ім’я користувача |
| last\_name | varchar | Прізвище користувача |
| email | varchar | Унікальна електрона пошта користувача |
| password | varchar | Пароль користувача |
| birthday | date | Дата народження |
| locked | boolean | Заблокований чи ні |
| active | boolean | Пройшов підтвердження по пошті чи ні |
| created\_at | timestamp | Дата реєстрації |
| updated\_at | timestamp | Дата останнього оновлення облікового  запису |
| image\_name | varchar | Унікальна назва зображення профілю |
| image\_url | varchar | Посилання на зображення |
| gender | varchar | Стать |
| telegram\_id | int | Ідентифікатор телеграм акаунта |

**Таблиця 5.** Опис таблиці confirmation\_token

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор токена |
| token | varchar | Токен |
| created\_at | timestamp | Дата створення |
| user\_id | bigint | Ідентифікатор користувача |

**Таблиця 6.** Опис таблиці password\_reset\_token

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор токена |
| token | varchar | Токен |
| created\_at | timestamp | Дата створення |
| expired\_at | timestamp | Дата закінчення терміну токена |
| user\_id | bigint | Ідентифікатор користувача |

**Таблиця 7.** Опис таблиці company

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор кінокомпанії |
| name | varchar | Назва кінокомпанії |

**Таблиця 8.** Опис таблиці movie

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор кінокартини |
| title | varchar | Назва кінокартини |
| description | varchar | Короткий опис |
| year | int | Рік випуску |
| budget | int | Бюджет (в доларах) |
| box\_office | int | Касові збори (в доларах) |
| release | date | Дата виходу |
| time | int | Час тривалості (в хвилинах) |
| poster\_name | varchar | Унікальна назва зображення постера |
| poster\_url | varchar | Посилання на зображення постера |
| trailer | varchar | Посилання на трейлер |
| status | boolean | Статус видимості |
| mpaa | varchar | Рейтинг кінокартини щодо цільової аудиторії |
| category | varchar | Категорія (фільм чи серіал) |

**Таблиця 9.** Опис таблиці movie\_cast

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| movie\_id | bigint | Ідентифікатор фільму |
| star\_id | bigint | Ідентифікатор актора |

**Таблиця 10.** Опис таблиці image\_movie

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор зображення |
| image\_name | varchar | Унікальна назва зображення |
| image\_url | varchar | Посилання на зображення |
| movie\_id | bigint | Ідентифікатор фільму |

**Таблиця 11.** Опис таблиці star

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор персони |
| first\_name | varchar | Ім’я персони |
| last\_name | varchar | Прізвище персони |
| gender | varchar | Стать |
| birthday | date | Дата дня народження |
| birth\_place | varchar | Опис місця народження |
| death | date | Дата смерті (якщо є) |
| image\_name | varchar | Унікальна назва зображення персони |
| image\_url | varchar | Посилання на зображення |
| status | boolean | Статус видимості |

**Таблиця 12.** Опис таблиці comment

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор коментаря |
| text | varchar | Текст коментаря |
| created\_at | timestamp | Дата створення |
| movie\_id | bigint | Ідентифікатор фільму |
| user\_id | bigint | Ідентифікатор користувача |
| parent\_id | bigint | Ідентифікатор батьківського коментаря |

**Таблиця 13.** Опис таблиці contact\_us\_message

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор повідомлення |
| text | varchar | Текст повідомлення |
| created\_at | timestamp | Дата створення |
| name | bigint | Ім’я відправника |
| email | bigint | Електрона пошта відправника |

**Таблиця 14.** Опис таблиці rating

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор оцінки |
| value | int | Оцінка (від 1 до 10) |

|  |  |  |
| --- | --- | --- |
| movie\_id | bigint | Ідентифікатор кінокартини |
| user\_id | bigint | Ідентифікатор користувача |
| created\_at | timestamp | Дата додання оцінки |
| updated\_at | timestamp | Дата оновлення оцінки |

**Таблиця 15.** Опис таблиці cinema

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор кінотеатру |
| name | varchar | Назва кінотеатру |
| city | varchar | Місто |
| address | varchar | Адреса кінотеатру |
| phone | varchar | Мобільний телефон кінотеатру |
| latitude | double | Координата широти, де знаходиться  кінотеатр |
| longitude | double | Координата довготи, де знаходиться  кінотеатр |
| rating | double | Рейтинг кінотеатру |
| image\_name | varchar | Унікальна назва зображення |
| image\_url | varchar | Посилання на зображення кінотеатру |
| active | boolean | Статус видимості |

**Таблиця 16.** Опис таблиці hall

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор залу |
| name | varchar | Назва залу |
| num\_seats | int | Кількість місць в залі |
| cinema\_id | bigint | Ідентифікатор кінотеатру |
| active | boolean | Статус видимості |

**Таблиця 17.** Опис таблиці row

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор ряду |
| name | varchar | Назва ряду |
| num\_seats | int | Кількість місць в ряді |
| hall\_id | bigint | Ідентифікатор залу |
| type | varchar | Тип ряду |

**Таблиця 18.** Опис таблиці seat

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор місця |
| number | int | Номер місця |
| row\_id | bigint | Ідентифікатор ряду |
| type | varchar | Тип місця |

**Таблиця 19.** Опис таблиці seance

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор сеансу |
| hall\_id | bigint | Ідентифікатор залу |
| movie\_id | bigint | Ідентифікатор фільму |
| date | date | Дата сеансу |
| start\_time | time | Початок показу сеансу |
| end\_time | time | Кінець показу сеансу |
| price | int | Ціна за квиток |
| available | boolean | Статус доступності |

**Таблиця 20.** Опис таблиці reservation

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор бронювання |
| seance\_id | bigint | Ідентифікатор сеансу |
| user\_id | bigint | Ідентифікатор користувача |
| created | timestamp | Дата створення бронювання |
| expired | timestamp | Дата закінчення бронювання |
| session | bigint | Ідентифікатор сесії користувача |
| active | boolean | Статус активності |

**Таблиця 21.** Опис таблиці seat\_reserved

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| id | bigint | Ідентифікатор |
| seat\_id | bigint | Ідентифікатор зарезервованого місця |
| reservation\_id | bigint | Ідентифікатор бронювання |
| price | int | Ціна зарезервованого місця |

**Таблиця 22.** Опис таблиці ticket

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Тип** | **Опис** |
| number | varchar | Унікальний номер квитка |
| created\_at | timestamp | Дата купівлі |
| date | timestamp | Дата сеансу |
| hall\_id | bigint | Ідентифікатор залу |
| movie\_id | bigint | Ідентифікатор фільму |
| price | int | Ціна за квиток |
| seats | varchar | Місця, які було куплено |
| email | varchar | Електрона пошта покупця |
| first\_name | varchar | Ім’я покупця |
| last\_name | varchar | Прізвище покупця |
| phone | varchar | Мобільний телефон покупця |
| user\_id | bigint | Ідентифікатор користувача (можливо  порожнє значення) |
| reservation\_id | bigint | Ідентифікатор бронювання (можливо  порожнє значення) |

## Розробка застосунку

Для підключення Java додатку до бази даних та роботою з нею використовується бібліотека Spring Data JPA [18]. Java Persistence API надає специфікацію для збереження, читання та керування даними з вашого об’єкта Java до реляційних таблиць у базі даних. Spring Data є частиною Spring Framework. Метою абстракції репозиторію Spring Data є суттєве зменшення кількості шаблонного коду, необхідного для реалізації рівнів доступу до даних.

Spring Data JPA не є постачальником JPA. Це бібліотека/фреймворк, який додає додатковий шар абстракції у верхній частині нашого постачальника JPA. В якості реалізації JPA використовується Hibernate.

Hibernate [19] - це інструмент ORM на основі Java, який забезпечує структуру для відображення об’єктів домену програми з таблицями реляційної бази даних і навпаки. Hibernate надає еталонну реалізацію Java Persistence API, що робить його чудовим вибором як інструмент ORM з перевагами слабкого з’єднання.

Для доступу до даних використовуються репозиторії Spring Data JPA. Запити JPA створюються автоматично з імен ваших методів. Для генерації запитів з власною логікою використовується JPQL [20]. JPQL - це потужна мова запитів, яка дозволяє визначати запити до бази даних на основі моделі вашої сутності. Її структура та синтаксис дуже схожі на SQL. JPQL використовує об’єктну модель сутності замість таблиць бази даних для визначення запиту. Це робить його дуже зручним для розробників Java, але БД все ще використовує мову запитів SQL. Тому як раз Hibernate перетворює запити JPQL в SQL.

В якості реалізації бізнес логіки використовуються сервіси. Всього в додатку реалізовано 24 класи сервісів.

В якості об’єктів, які передаються у відображення та приймаються контролерами використовуються DTO [21]. Шаблон проектування об’єкта передачі даних (DTO) – це часто використовуваний шаблон проектування. В основному він використовується для передачі даних із кількома атрибутами за один кадр від клієнта до сервера, щоб уникнути кількох викликів на віддалений сервер. Ще одна перевага використання DTO, полягає в тому, що вони можуть допомогти приховати деталі реалізації об’єктів домену (об’єктів JPA). Відкриття об’єктів через кінцеві точки може стати проблемою безпеки, якщо ми не ретельно обробляємо, які властивості можна змінити за допомогою яких операцій.

Контролери відповідають за обробку запитів користувачів та передачу даних модулю представлення для обробки до сервісів. В наступній таблиці описано всі контролери і за що вони відповідають.

**Таблиця 23.** Опис контролерів

|  |  |  |
| --- | --- | --- |
| **Номер** | **Назва контролеру** | **Опис** |
| 1 | RegistrationController | Відповідає за реєстрацію нового користувача  та підтвердження облікового запису по пошті |
| 2 | PasswordResetController | Відповідає за відновлення паролю по пошті |
| 3 | ProfileController | Відповідає за перегляд та редагування  інформації про себе, зміну паролю, перегляд  власних квитків та функцію підписки та відписки на/від користувачів. |

|  |  |  |
| --- | --- | --- |
| 4 | UserController | Відповідає за відображення профілю користувача: коротку інформацію про нього, його оцінки, улюблені фільми, список  бажаного до перегляду, коментарі, а також список його підписок та підписників. |
| 5 | AdminMainController | Відповідає за відображення головної  сторінки для адміністраторів, основної статистики по сайту |
| 6 | CompanyController | Відповідає за CRUD кінокомпаній |
| 7 | AdminMovieController | Відповідає за CRUD кінокартин |
| 8 | AdminStarController | Відповідає за CRUD персон |
| 9 | AdminCinemaController | Відповідає за CRUD кінотеатрів |
| 10 | AdminHallController | Відповідає за CRUD кінозалів |
| 11 | AdminSeanceController | Відповідає за CRUD сеансів |
| 12 | AdminUserController | Відповідає за перегляд, редагування,  видалення та блокування користувачів з роллю «User». |
| 13 | CommentController | Відповідає за перегляд та видалення  коментарів |
| 14 | AdminTicketController | Відповідає за перегляд та видалення квитків |
| 15 | ContactUsController | Відповідає за перегляд повідомлень,  надісланих від користувачів, їх видалення та можливість відповісти на повідомлення  адміністратором |
| 16 | CinemaController | Відповідає за відображення списку всіх кінотеатрів та сторінки конкретного  кінотеатру з інформацією про нього |
| 17 | SeanceController | Відповідає за відображення сеансу, доступних місць та можливість  зарезервувати місця |
| 18 | ReservationController | Відповідає за перегляд, скасування  бронювання та купівлю квитків |
| 19 | MainController | Відповідає за відображення головної  сторінки, пошуку та «contact us» сторінки |
| 20 | MovieController | Відповідає за відображення списку  кінокартин з можливістю фільтрації та  відображення детальної сторінки фільму з переглядом інформації про нього та можливістю залишати коментарі. |
| 21 | RatingController | Відповідає за виставлення оцінок фільмам, їх  видалення, додання фільмів до улюблених та до бажаних для авторизованих користувачів |

|  |  |  |
| --- | --- | --- |
| 22 | StarController | Відповідає за відображення списку персон та сторінки конкретної персони з її  фільмографією за весь час |

## Сценарії використання веб-додатку

## Сценарій адміністратора

При успішній авторизації в ролі адміністратора, користувач може перейти до адміністративної панелі. Головна сторінка відображає загальну статистику по всьому сайту.

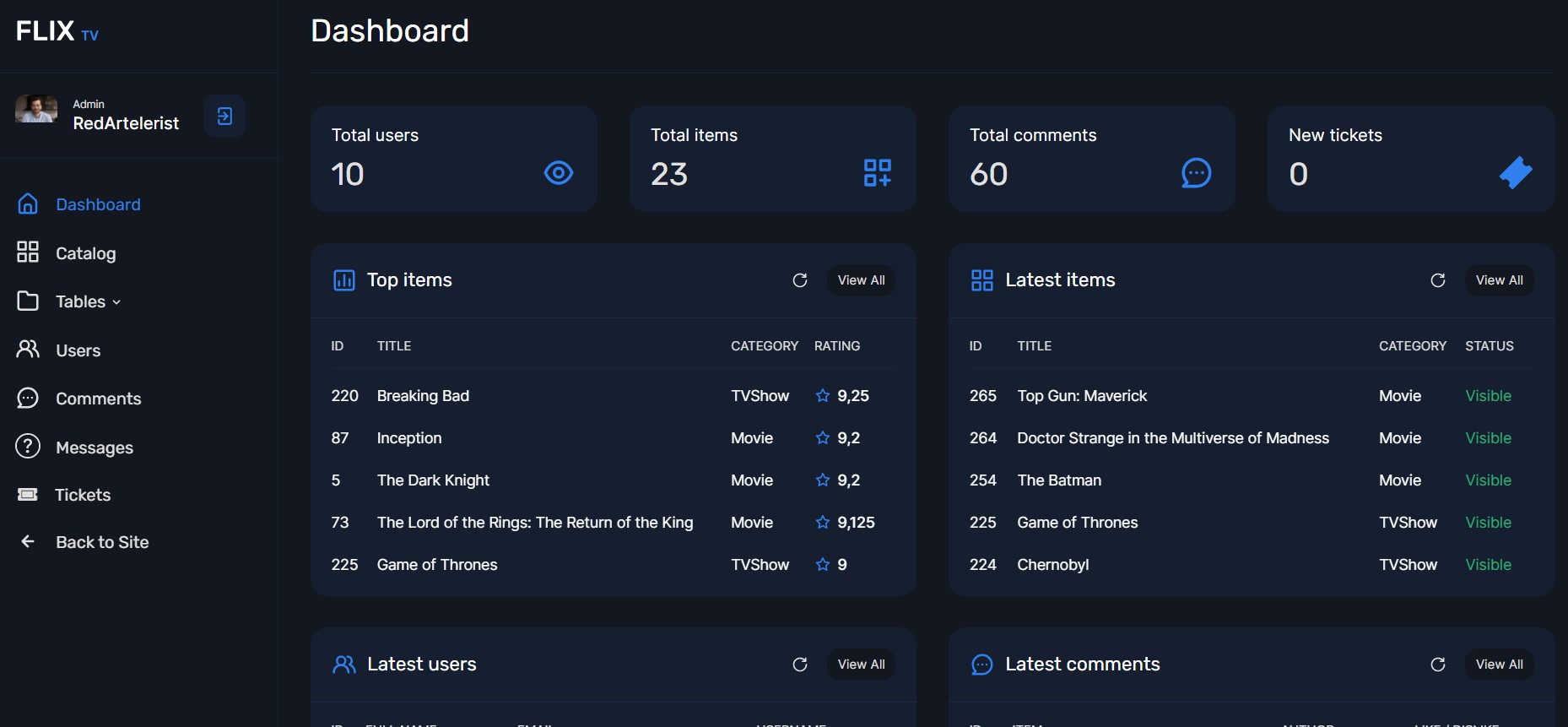


Рисунок 7 - Головна сторінка адміністративної панелі

Далі адміністратор може перейти до каталогу для перегляду всіх фільмів та серіалів. На сторінці відображається коротка інформація про фільми, також є можливість сортувати по деяким полям, наявний пошук по назві та пагінація. Ми можем перейти до конкретного елементу для перегляду повної інформації з можливістю редагування або видалити конкретний елемент чи додати новий.

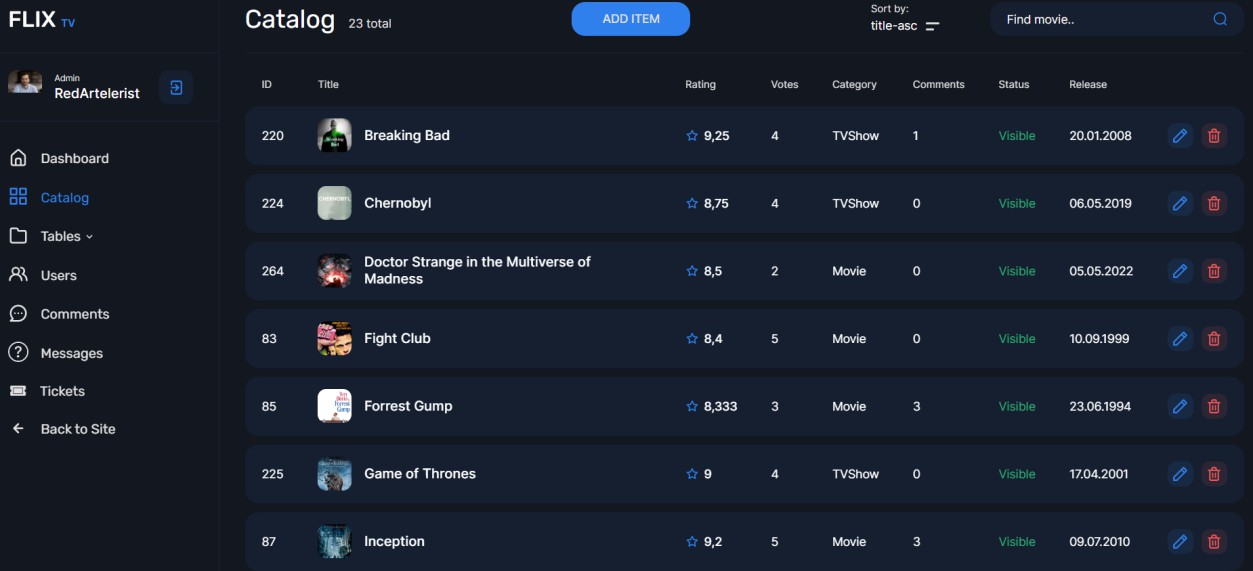


Рисунок 8 - Каталог фільмів та серіалів

При виборі додати новий елемент треба заповнити форму. На всіх формах в адміністративній панелі наявна валідація.

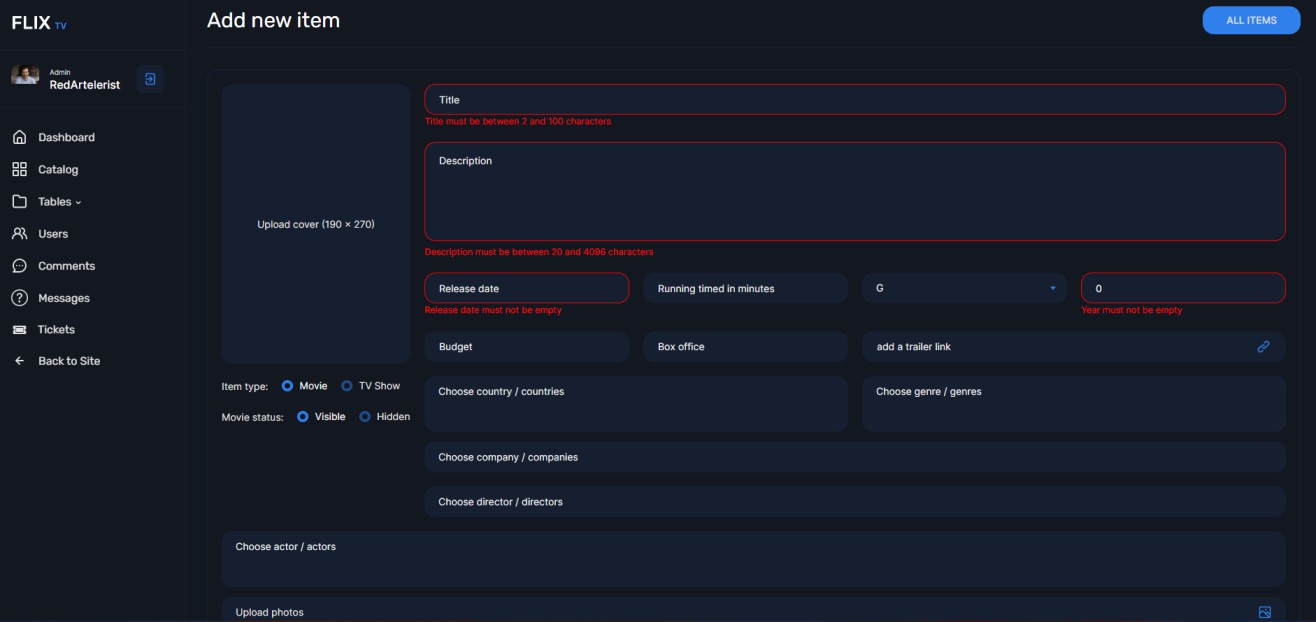


Рисунок 9 - Додавання нового фільму чи серіалу з відповідною валідацією

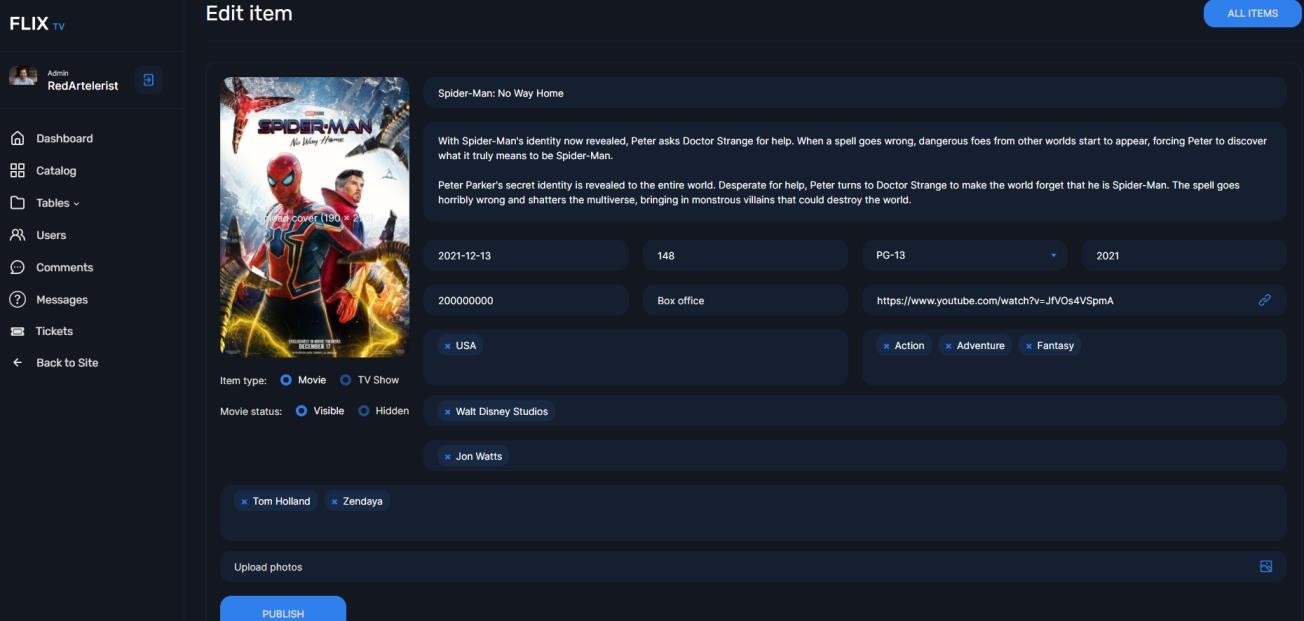


Рисунок 10 - Редагування фільму

Також окремо на сторінці з редагуванням ми можемо видалити будь-яке зображення з галереї.

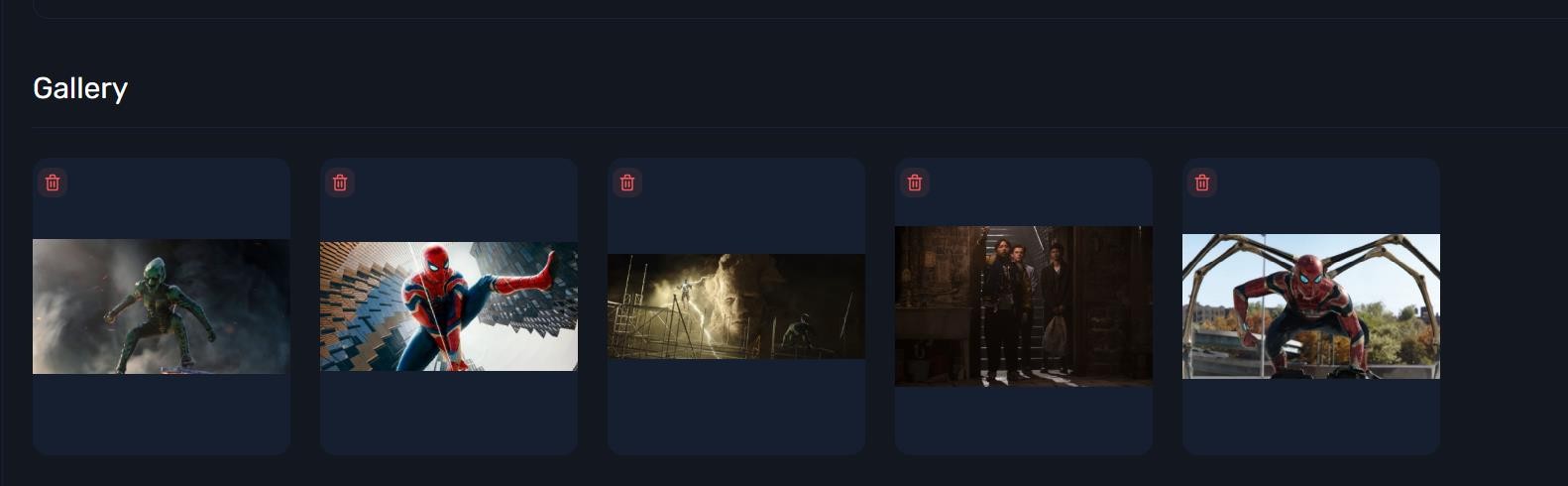


Рисунок 11 - Галерея фільму

Далі можна перейти до списку персон та кінокомпаній з можливістю створення, редагування та видалення аналогічно до кінокартин. Теж саме можна робити з кінотеатрами з одною відмінністю: при створені та редагуванні можна скористатися картою для збереження координат місцезнаходження кінотеатру.

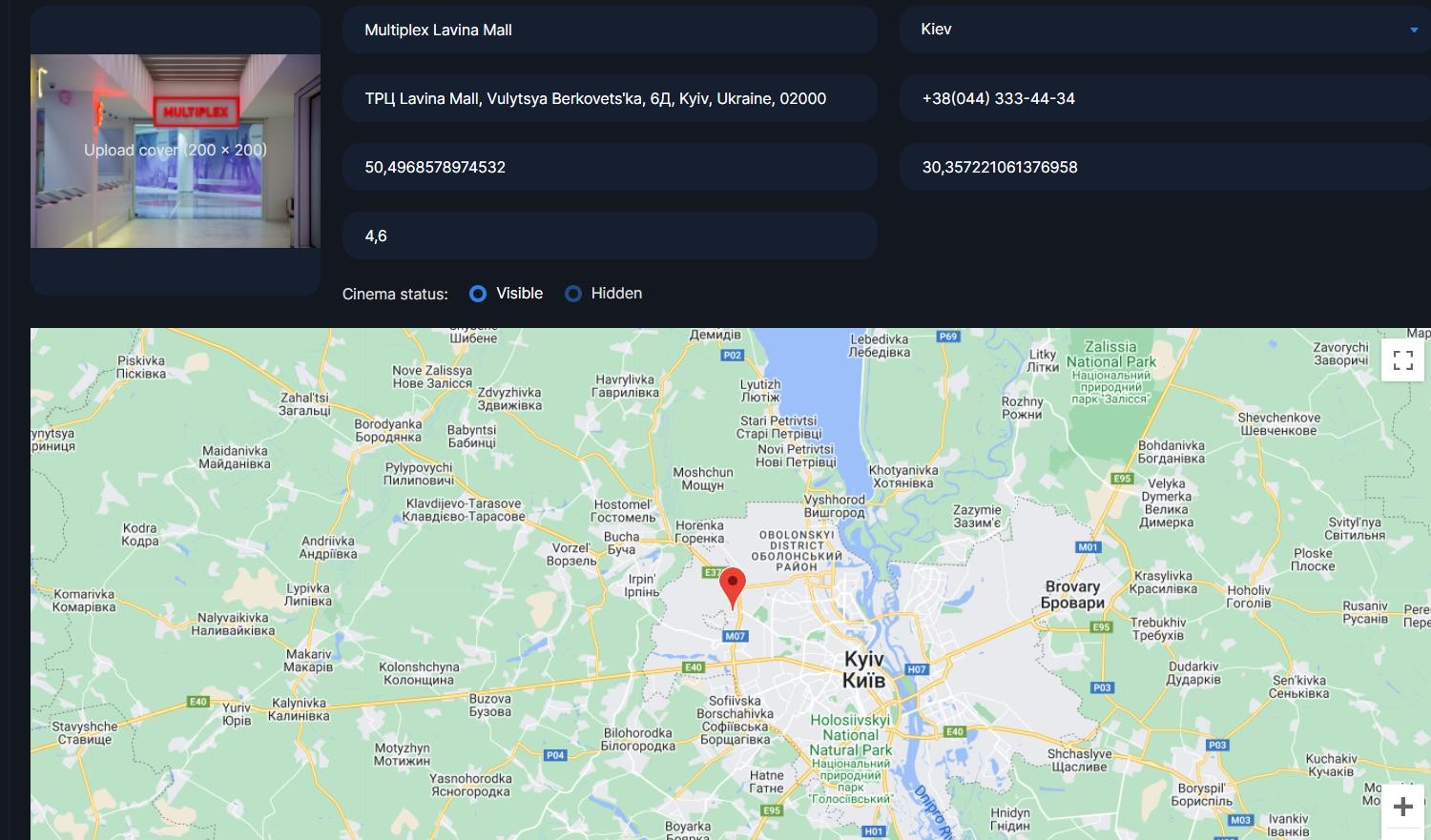


Рисунок 12 - Форма редагування кінотеатру

Також на сторінці редагування кінотеатру наявний список всіх кінозалів з можливістю редагування та видалення їх чи створення нових. Можна переглянути список всіх сеансів в даному кінотеатрі, додати новий сеанс чи редагувати або видалити існуючі. Після закінчення сеансів вони автоматично видаляються із системи.

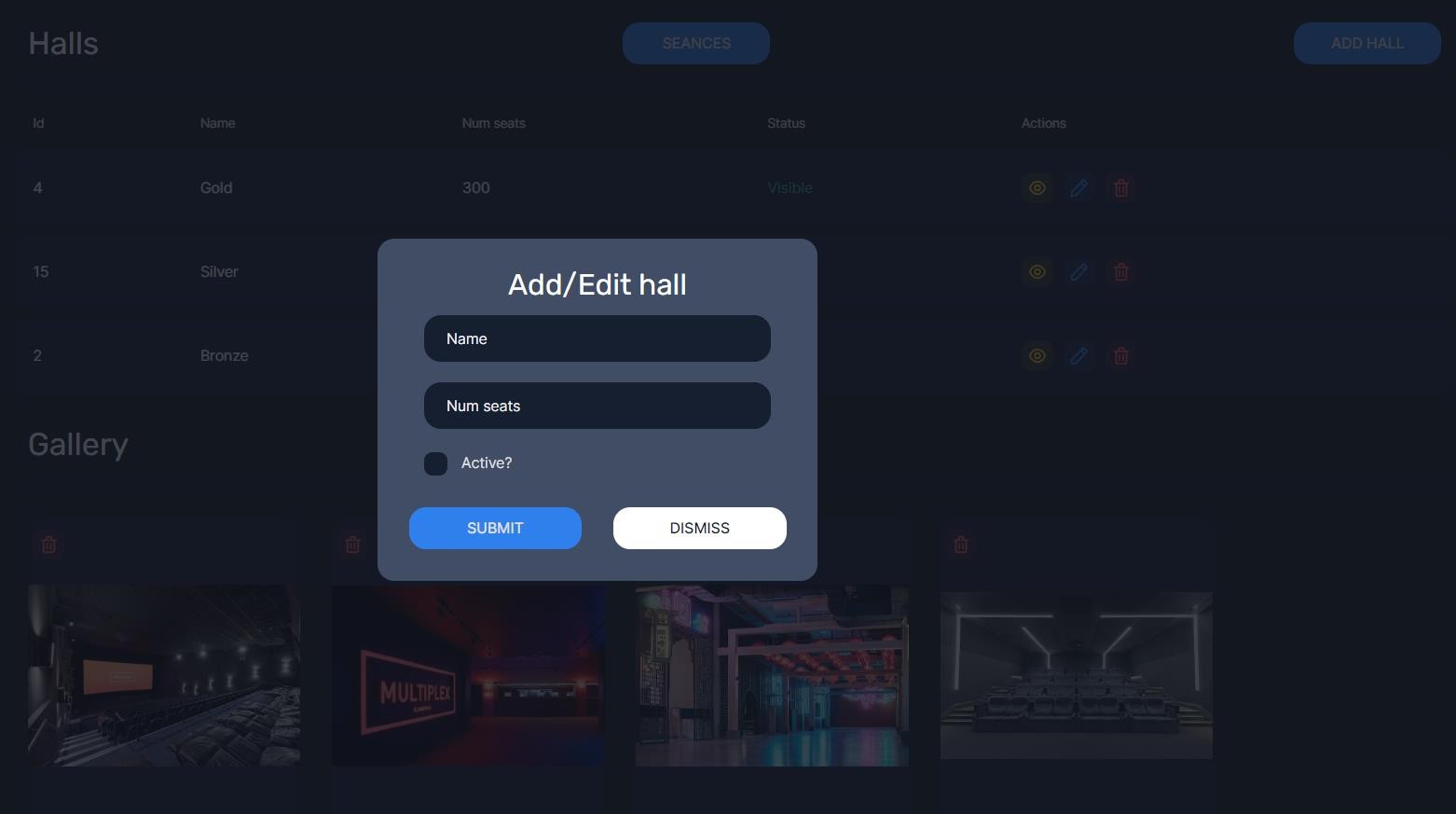


Рисунок 13 - Список кінозалів та форма додання нового залу

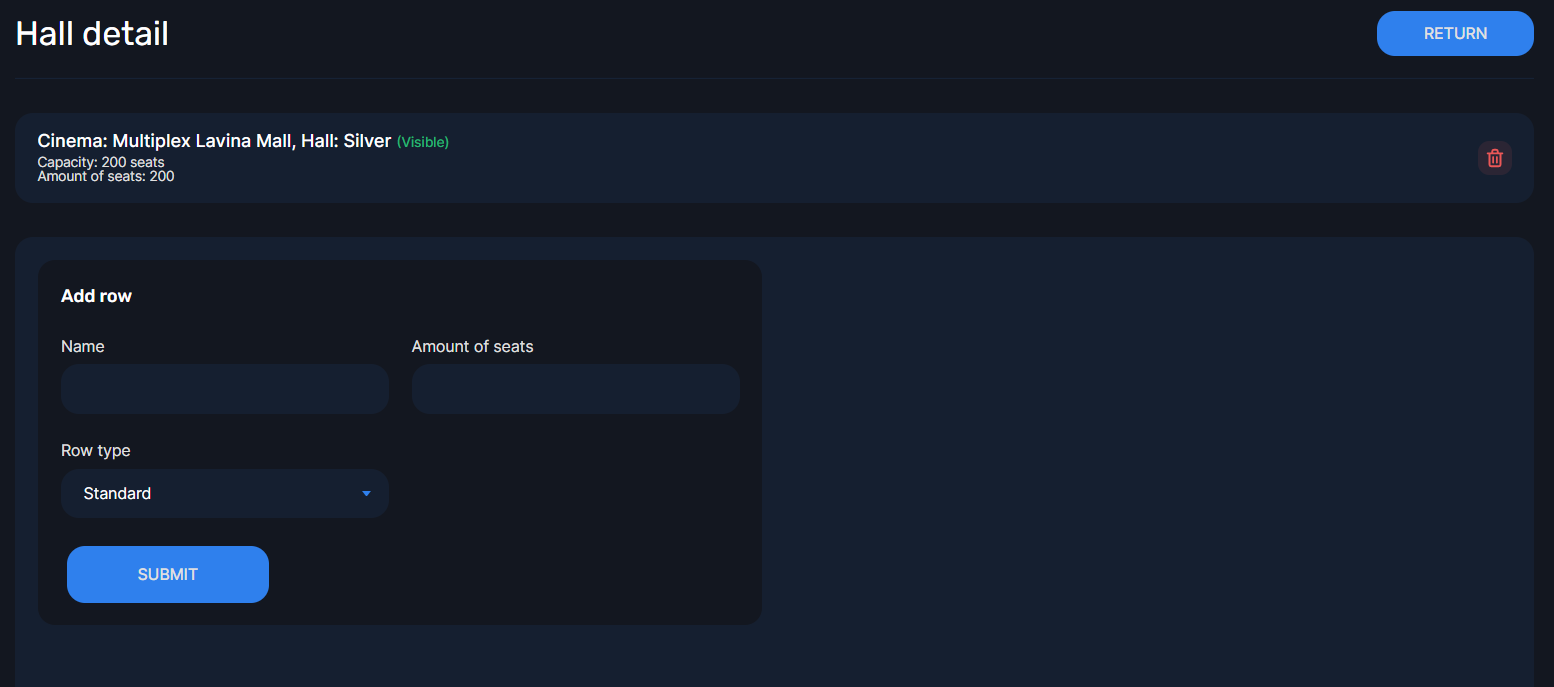
Ми можемо перейти до конкретного кінозалу та переглянути його план відповідно до типу місць. Також можна додати новий ряд, якщо є можливість, або видалити та редагувати вже існуючі.

Рисунок 14 - Форма додавання ряду

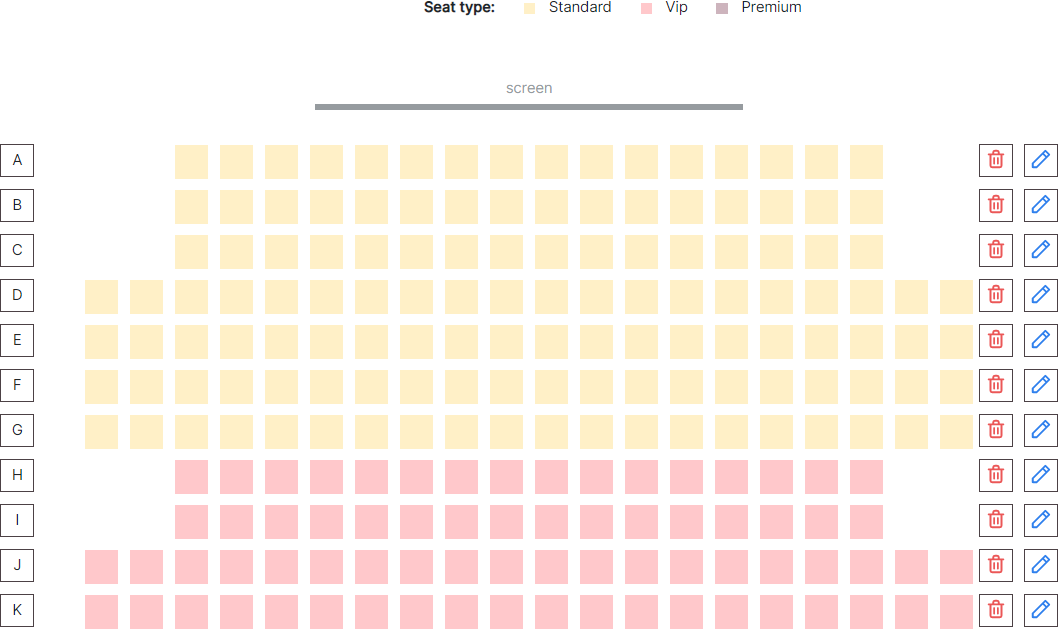


Рисунок 15 - План кінозалу

Далі можна переглянути список користувачів та редагувати їх інформацію, видалити чи заблокувати, якщо вони мають виключно роль «User». Також в системі є роль власника і відповідно єдиний користувач до неї, який може

оперувати адміністраторами. Таким чином він може надавати користувачам або відбирати у них роль адміністраторів.

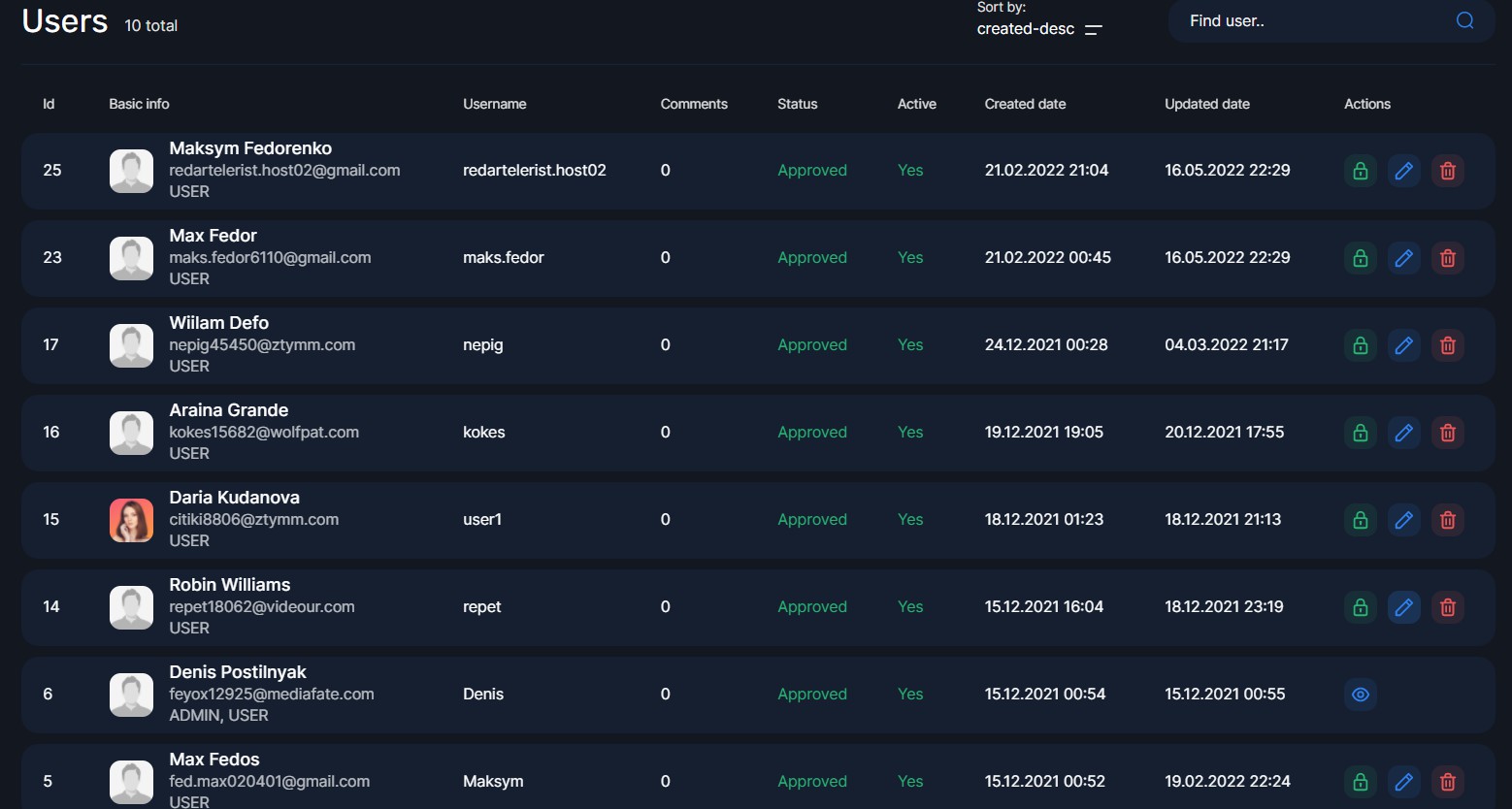


Рисунок 16 - Список користувачів.

Аналогічно до всіх попередніх функцій можна переглядати списки коментарів, запитань та квитків, видаляти їх. Для запитань є можливість надіслати відповідь на пошту після чого запитання автоматично видалиться із системи.

## Сценарій звичайного користувача

При переході на головну сторінку ми можемо побачити фільми, які зараз йдуть у прокаті, або можемо скористатися розширеним пошуком.

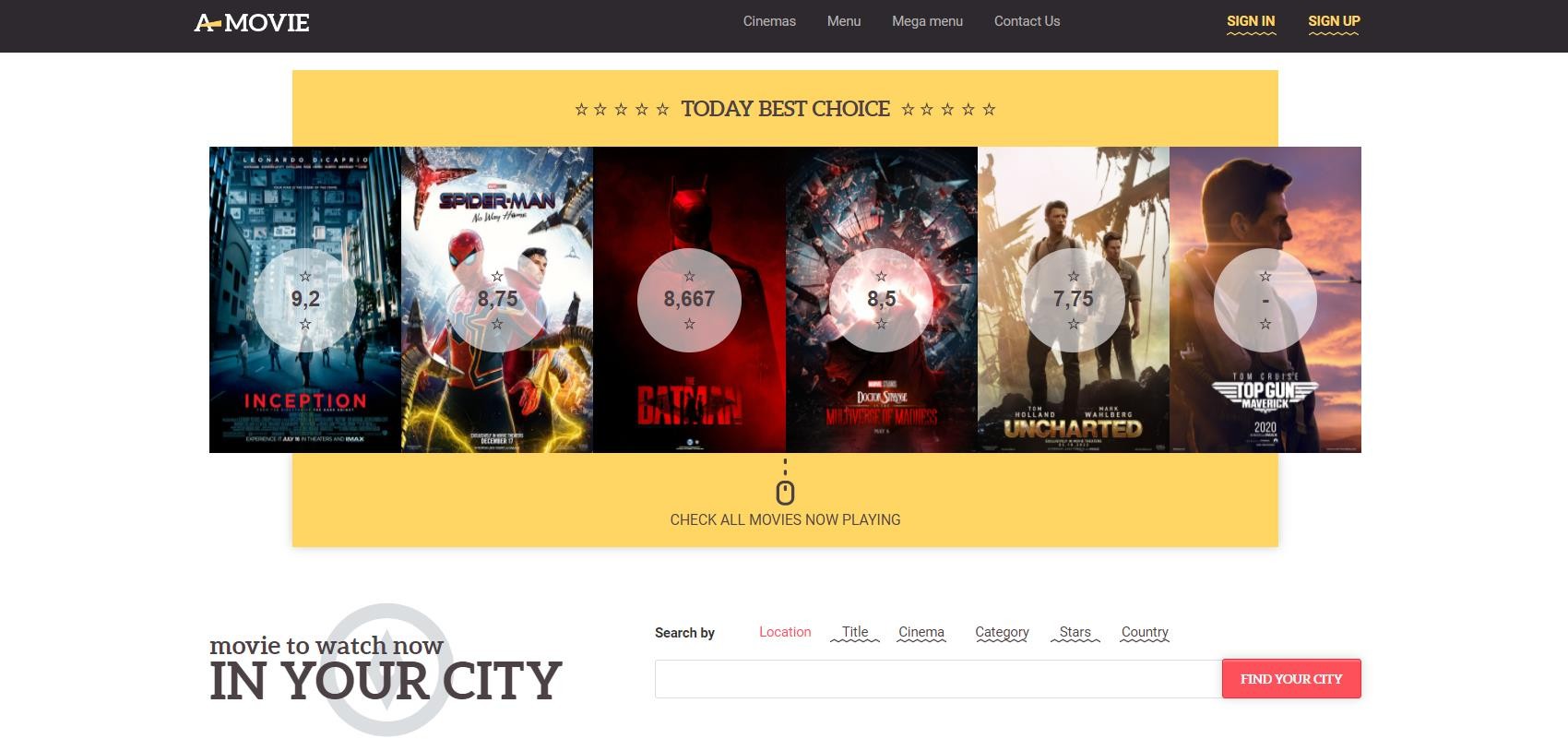


Рисунок 17 - Головна сторінка сайту «A.Movie»

Далі у нас є можливість продовжити як анонімний користувач або авторизуватися чи зареєструвати новий обліковий запис. При реєстрації треба заповнити декілька полів, при чому поля пошти та юзернейму повинні бути унікальними в системі. Після чого вам надійде повідомлення на пошту для підтвердження свого акаунта. І тільки після того, як перейти за ним, ваш обліковий запис стане активованим і буде можливість авторизуватися.

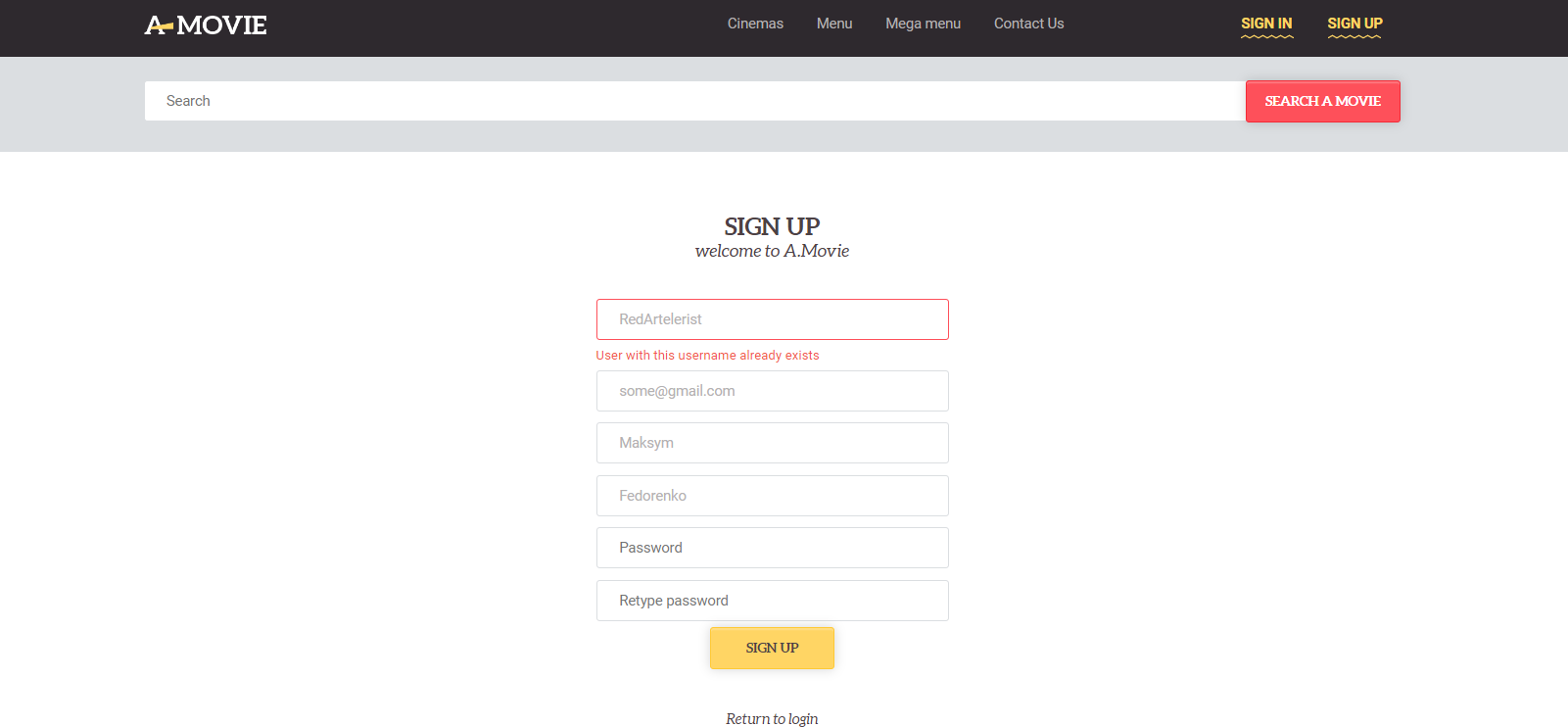


Рисунок 18 - Форма реєстрації

Також наявна можливість увійти за допомогою соціальної мережі Google чи Facebook. Вона під’єднається до існуючого акаунту з такою ж електронною поштою або якщо такого не існує, то система автоматично створить його з тимчасовим паролем, який прийде в якості повідомлення на пошту. Також, якщо користувач забув пароль від свого облікового запису, він може ввести пошту, на яку прийде повідомлення з формою відновлення пароля, яка дійсна півгодини, і змінити свій пароль після чого форма стане недійсною.



Рисунок 19 - Форма авторизації

Далі ми можемо перейти до навігатору для перегляду списку всіх наявних на сайті фільмів та серіалів. На даній сторінці можна відсортувати елементи по рейтингу, даті, популярності та ін. Також наявна розширена форма для фільтрації елементів по категорії, кінокомпанії, жанру, країні та рокам.

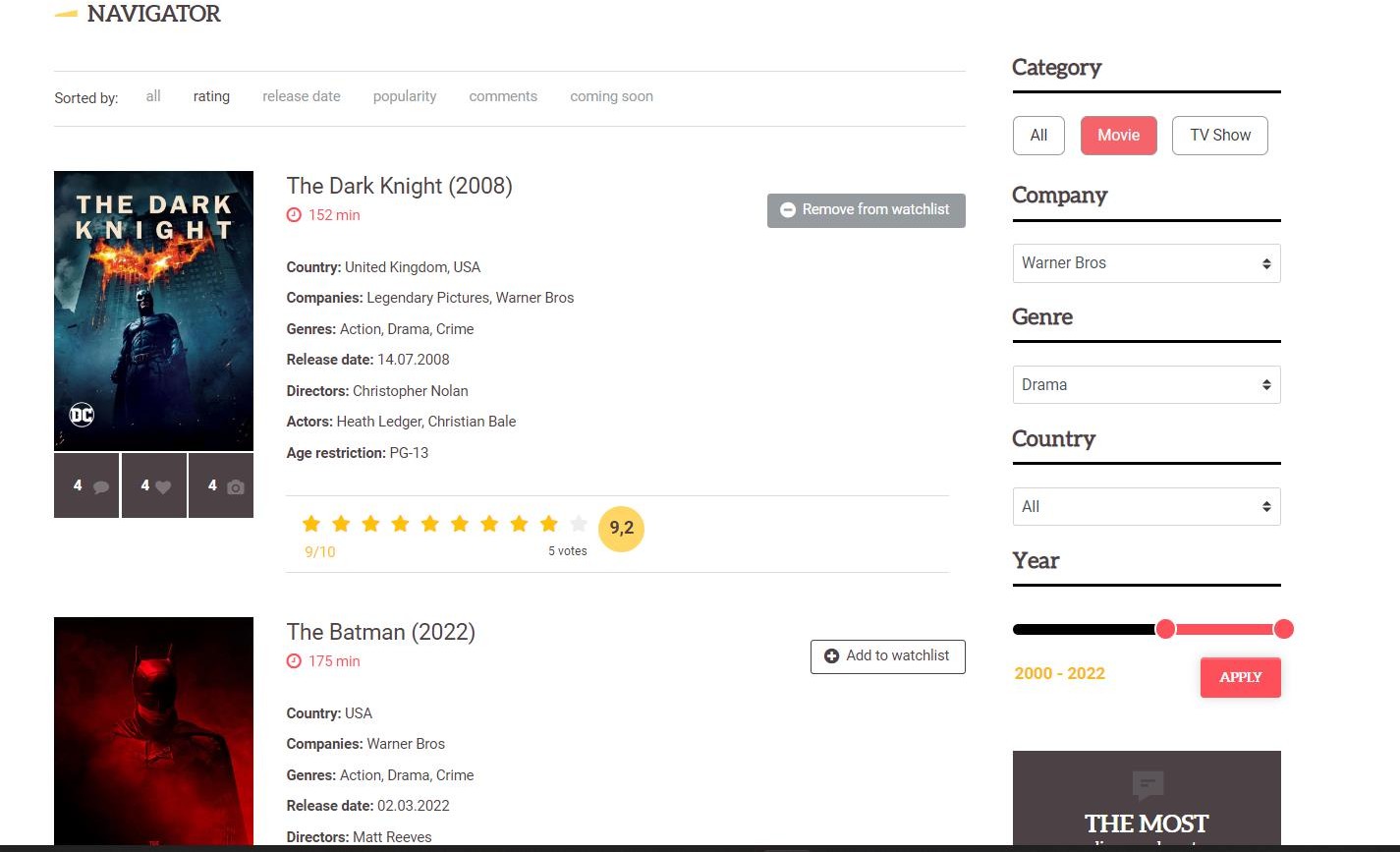


Рисунок 20 - Сторінка навігації по фільмам та серіалам

На сторінці кінокартини можна переглянути детальну інформацію про неї. Також, якщо користувач авторизований, є можливість оцінити фільм, додати його до списку улюблених чи списку бажаного до перегляду. За допомогою Ajax [22] всі ці запити працюють динамічно без перезавантаження сторінки. Також є можливість залишити коментар, відповісти на інший та лайкнути або дизлайкнути коментар.

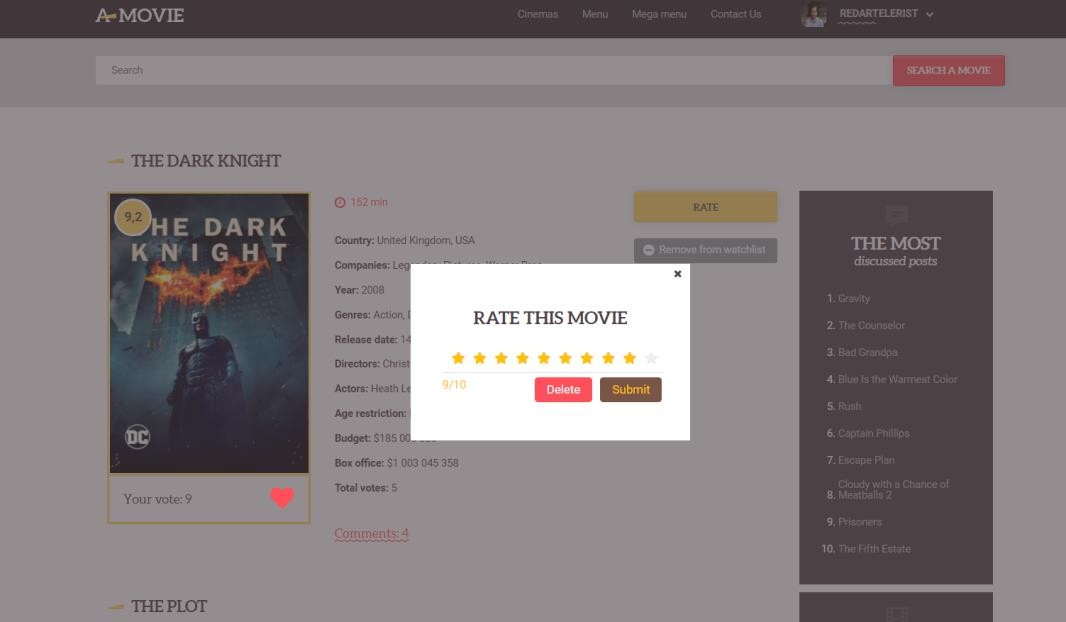


Рисунок 21 - Детальна сторінка фільму та форма виставлення оцінки

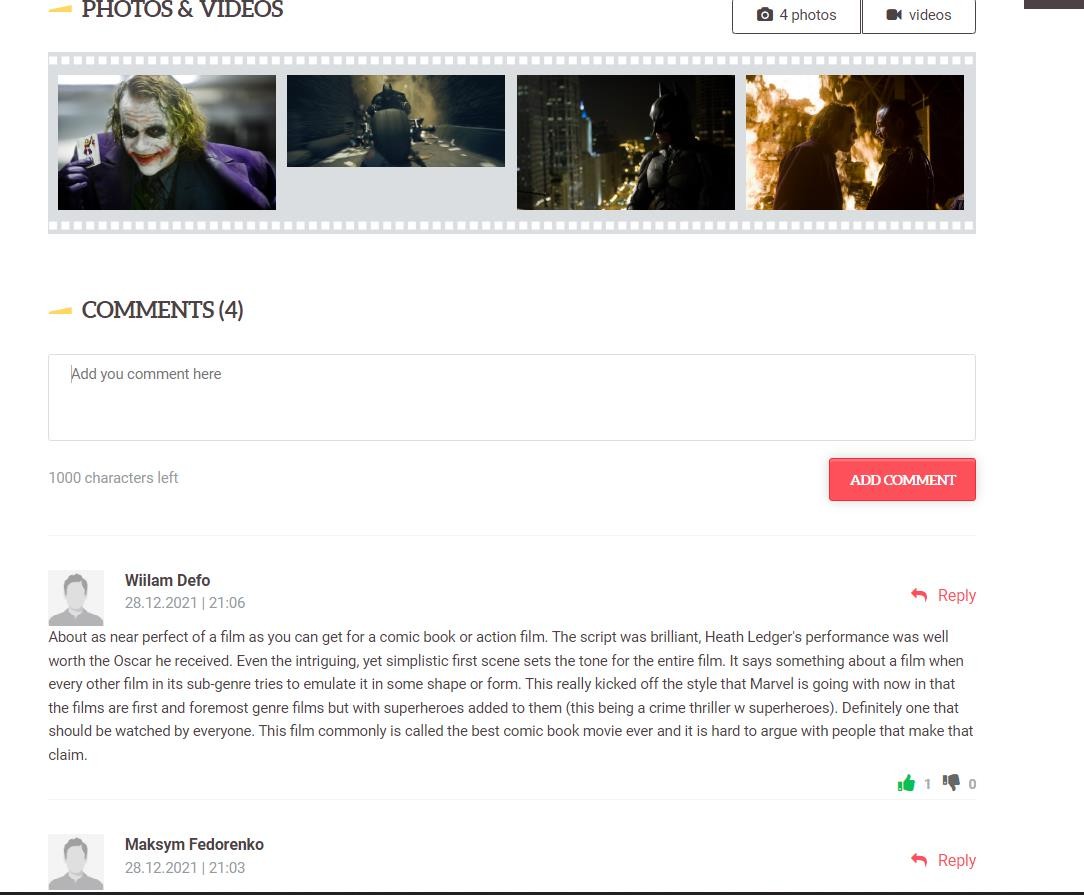


Рисунок 22 - Список коментарів фільму

Можна переглянути профіль користувача, його оцінки, коментарі та ін.

Також, можна підписатися на нього, якщо ви авторизовані.

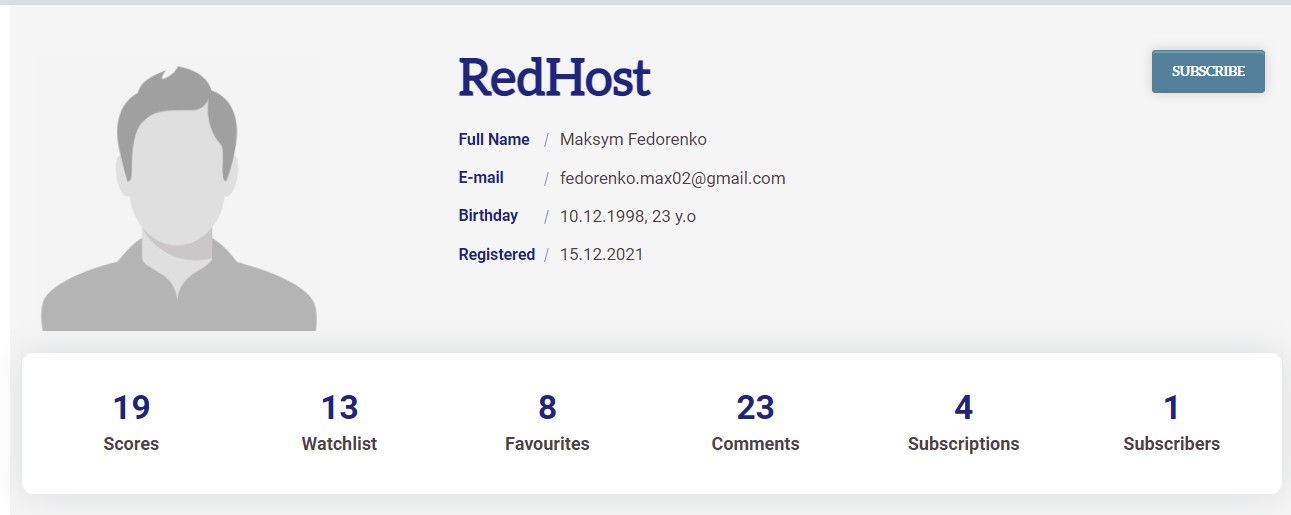


Рисунок 23 - Сторінка користувача

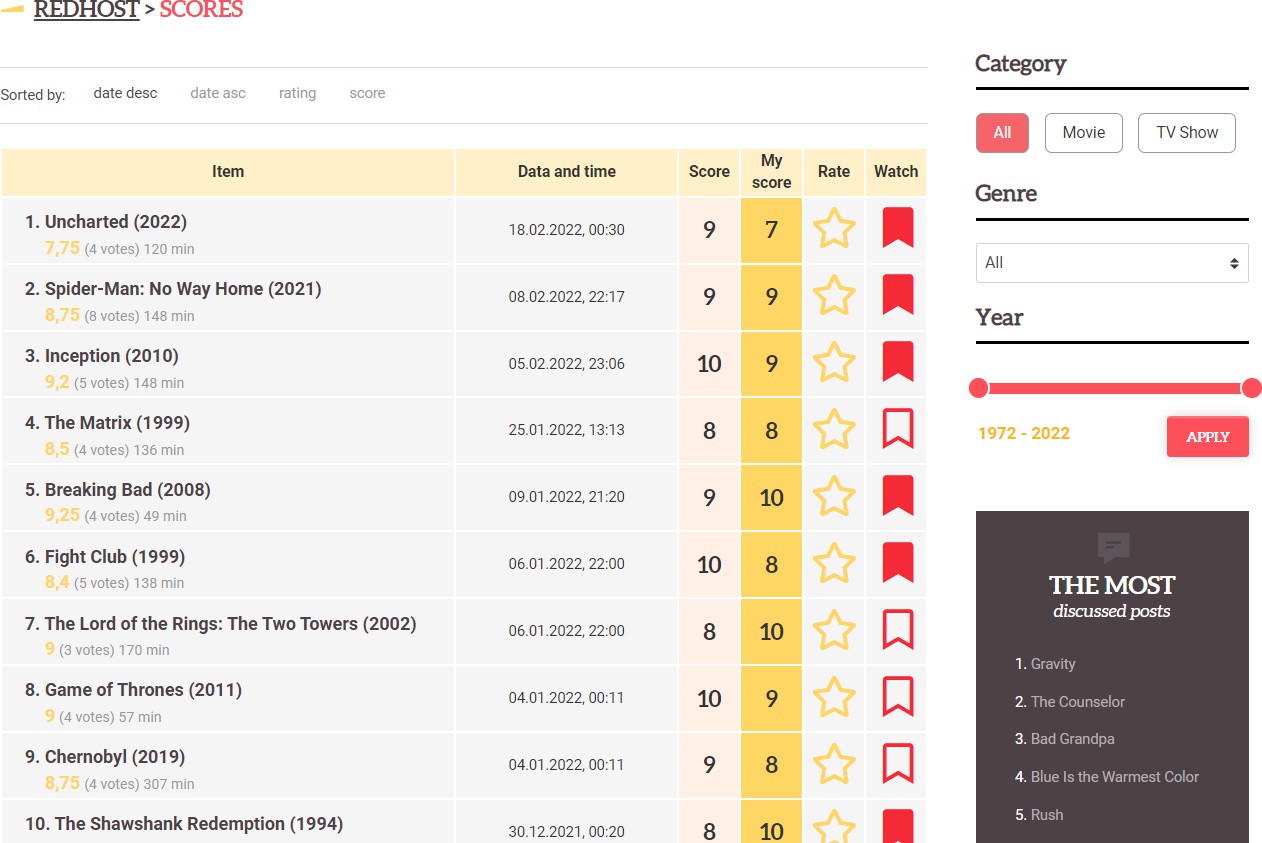


Рисунок 24 - Список оцінок користувача

Зі своїм акаунтом можна робити теж саме, додатково є можливість перегляду куплених білетів за весь час. У своєму профілі можна редагувати інформацію про себе, додати зображення та змінити пароль при необхідності.

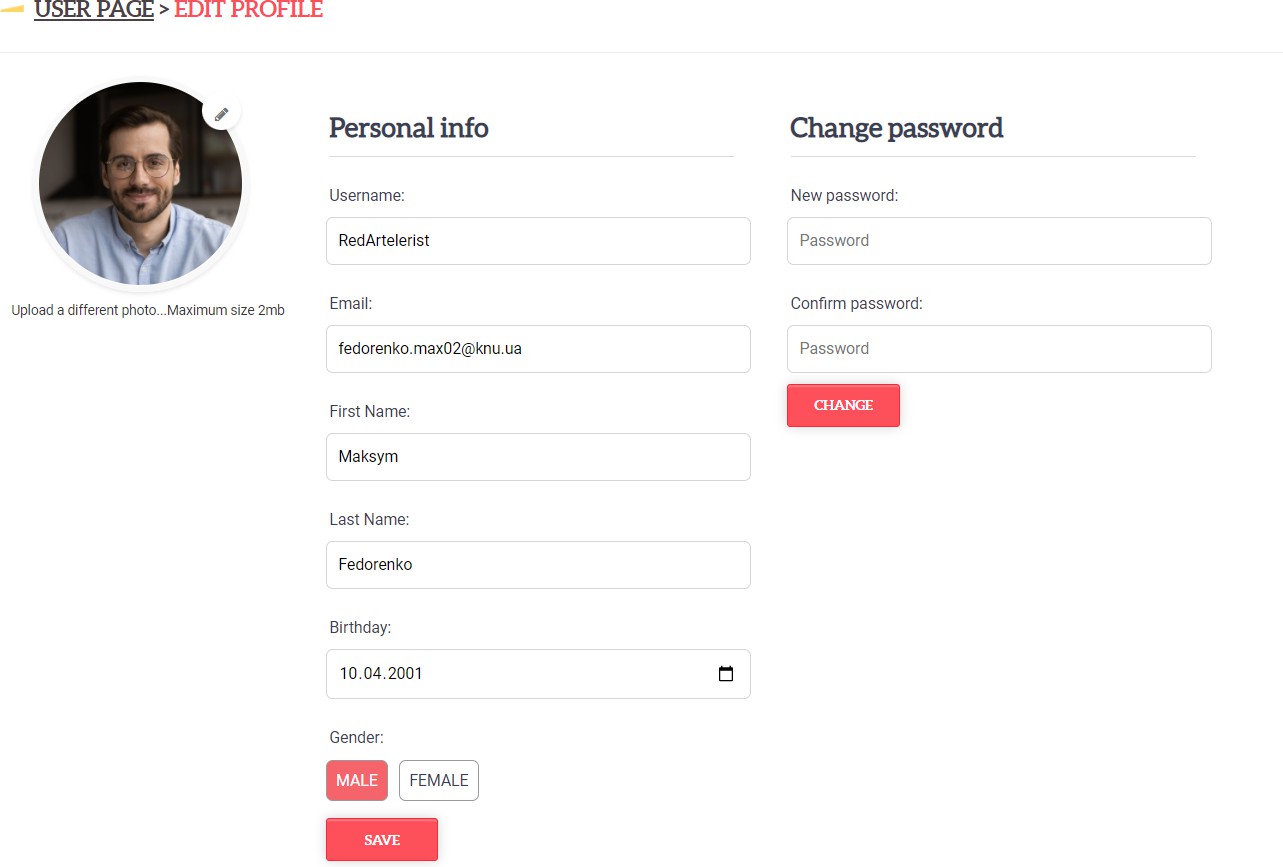


Рисунок 25 - Сторінка власного профілю

Для того щоб замовити квитки, користувач може перейти до списку кінотеатрів, відфільтрувати їх по місту та вибрати потрібний. На головній сторінці відображається інформація про кінотеатр, його місцезнаходження та список сеансів. Можна вибрати дату із можливих, яка вас цікавить, та вибрати сеанс на бажаний фільм.

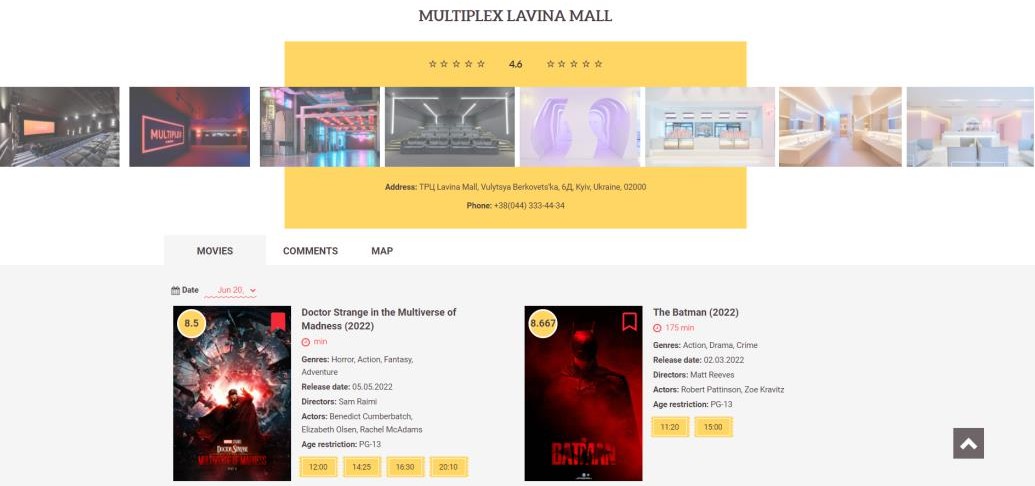


Рисунок 26 - Сторінка кінотеатру

Далі можна вибрати вільні місця і перейти до наступного кроку.

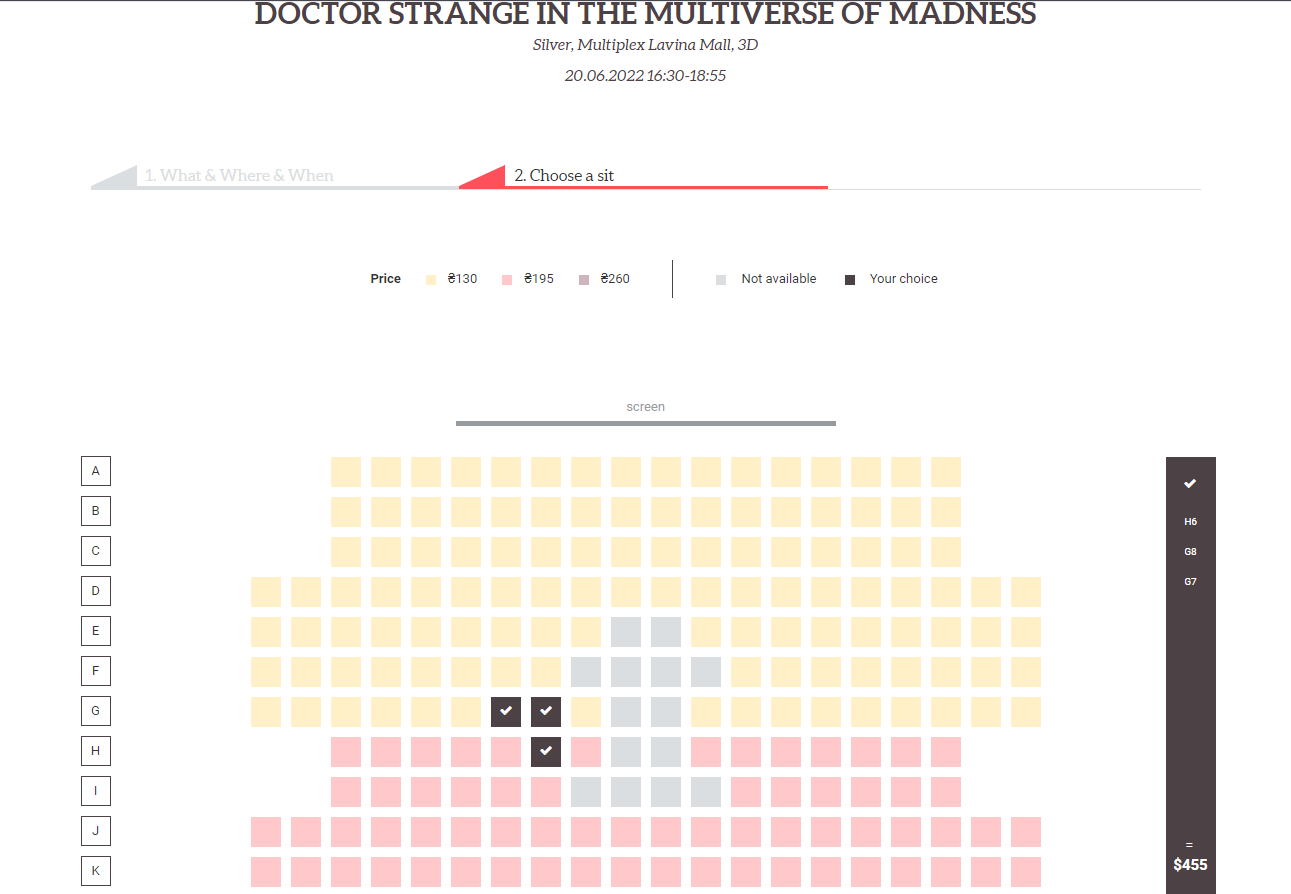


Рисунок 27 - Сторінка сеансу

При чому, коли ви переходите далі, то ваші місця стають недоступними для інших протягом семи хвилин. За цей час вам треба заповнити інформацію про замовлення та оплатити карткою за допомогою сервісу Stripe.

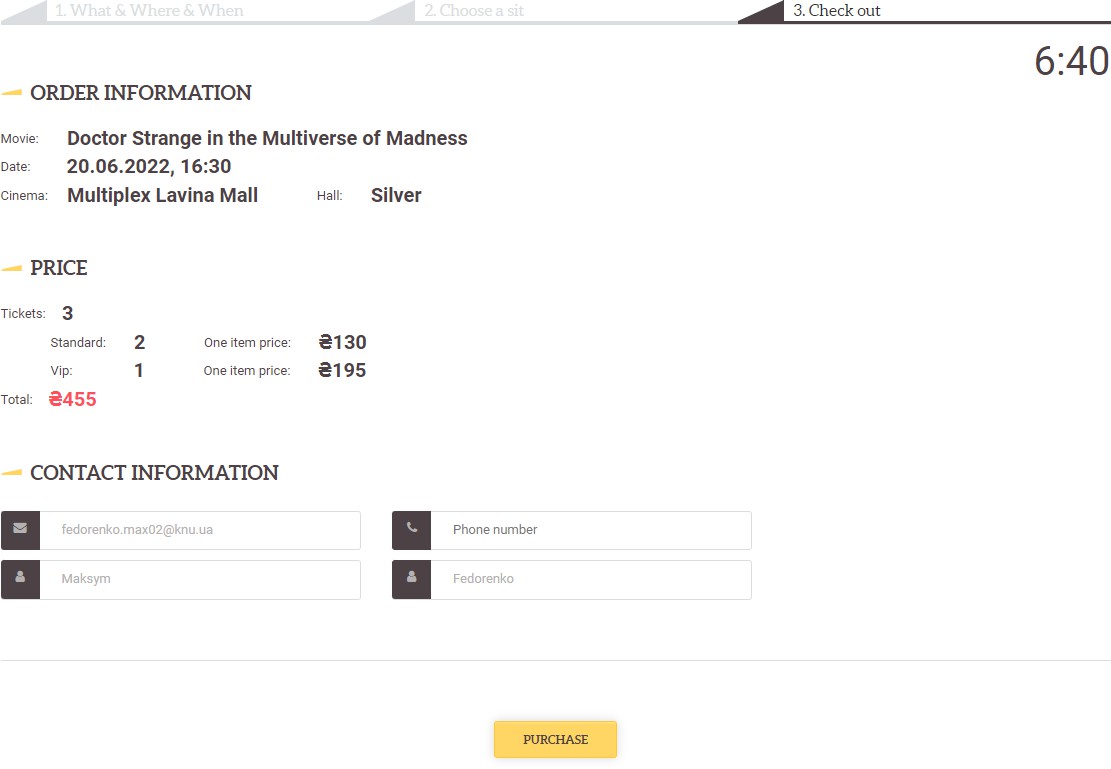


Рисунок 28 - Сторінка купівлі квитків

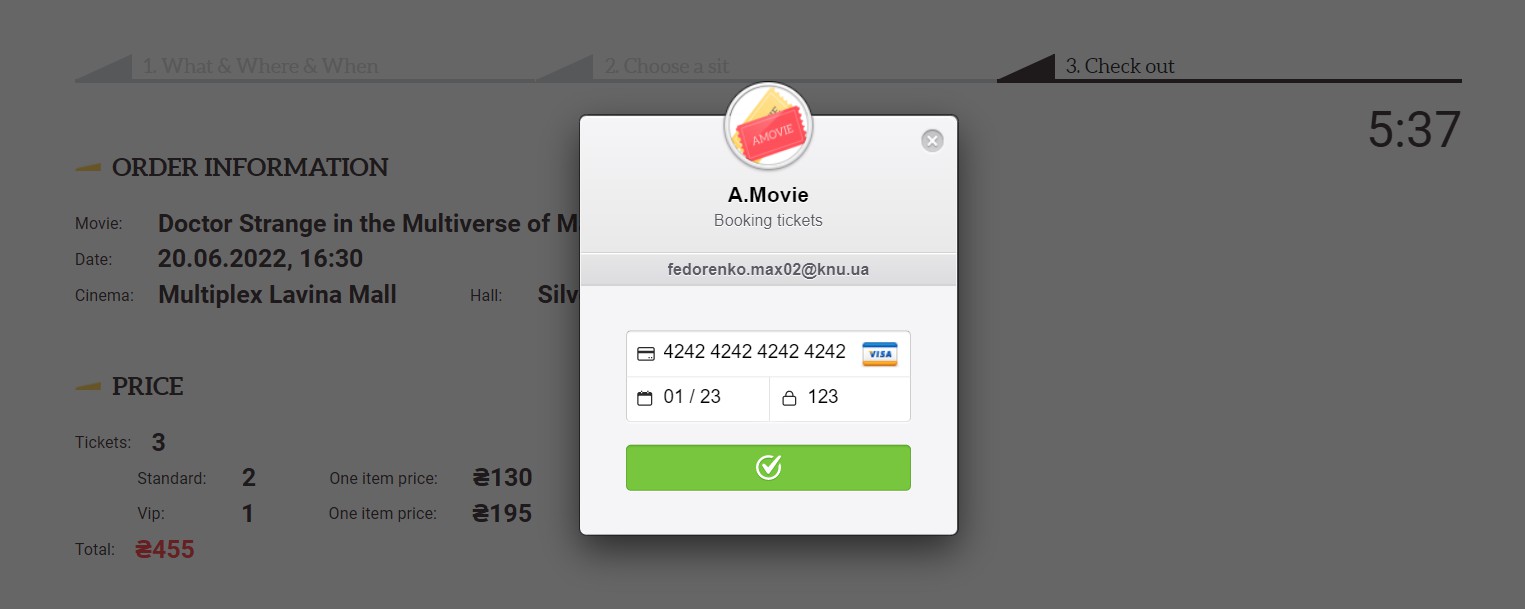


Рисунок 29 - Оплата картою

Після успішної оплати вам на пошту, а також в телеграм бот прийде квиток.

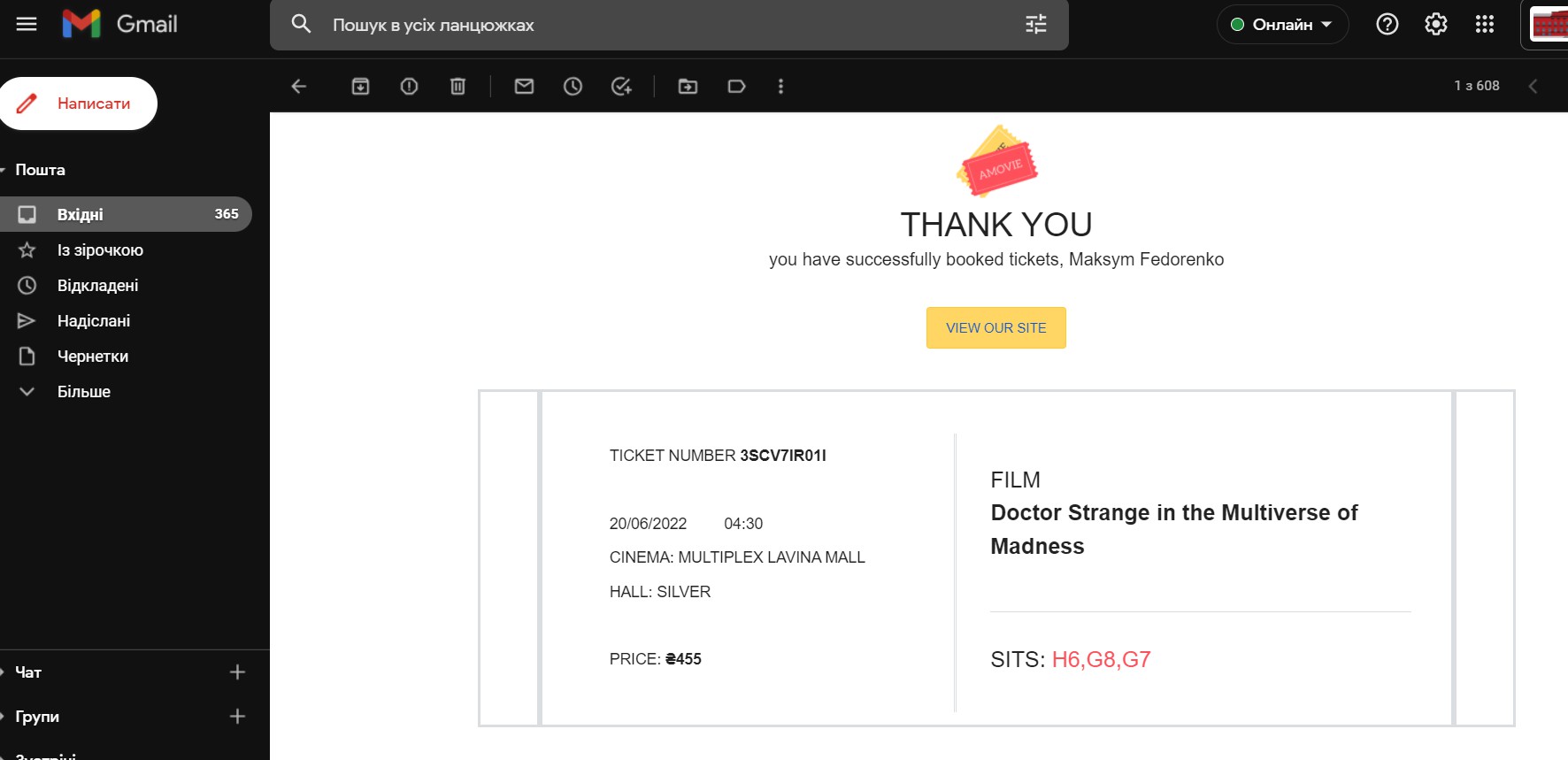


Рисунок 30 - Квиток на пошті

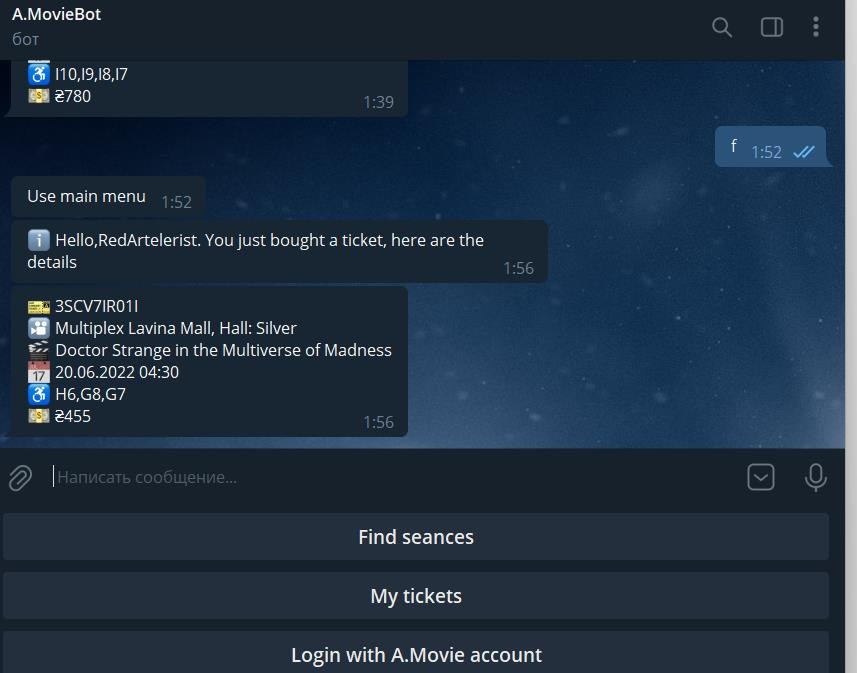


Рисунок 31 - Квиток в телеграм боті

Також користувач може скористатися телеграм ботом від веб-додатка. В ньому можна по юзернейму та паролю приєднатися до облікового запису

«A.Movie».

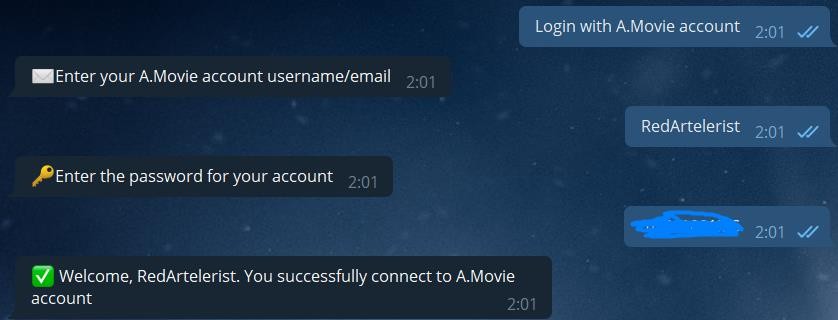


Рисунок 32 - Приєднання до «A.Movie» акаунту в телеграм боті Функціонал бота дозволяє переглядати свої білети куплені за весь час,

знаходити сеанси по кінотеатру, фільму та даті, а також бот має систему сповіщень, яка буде вам надсилати відповідне повідомлення за дві години до вашого сеансу.

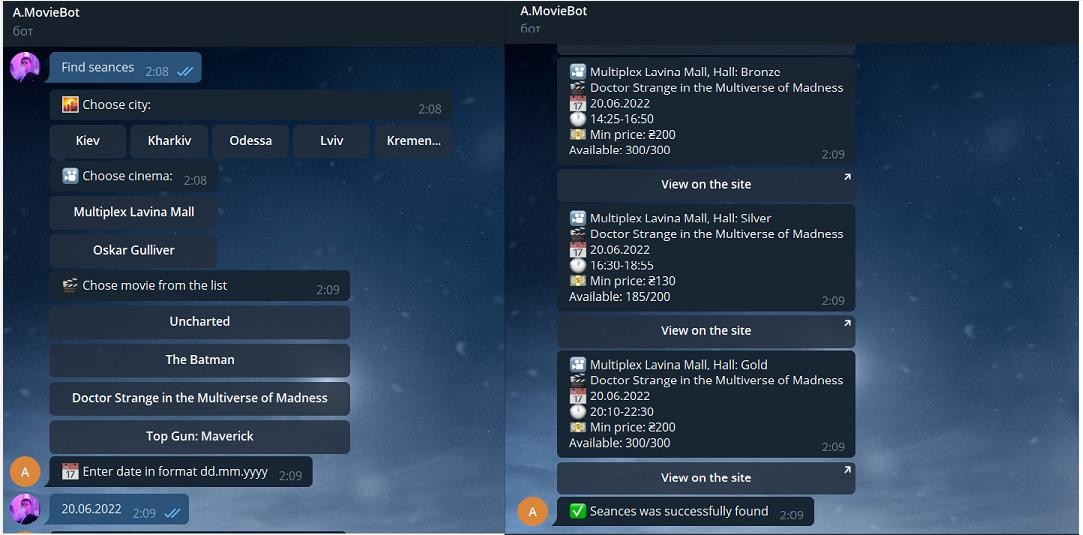


Рисунок 33 - Пошук сеансів в телеграм боті

# ВИСНОВКИ

В ході виконання кваліфікаційної роботи було виконано такі поставлені задачі:

* досліджено готові рішення соціальних кіномереж та сервісів бронювання, купівлі квитків, виявлено їх недоліки та переваги в результаті чого сформовано власні пріоритети для формування функціоналу сайту
* розглянуто та досліджено Spring-фреймворк, його особливості, бібліотеки та архітектуру
* поглиблено навички у розробці веб-застосунків
* впроваджено в проект різні API (Stripe, Firebase, Telegram Bot, Google, тощо). Деякі з них використовувалися вперше (Stripe, Telegram Bot), а з іншими були закріплені та поглиблені знання, отримані раніше
* розроблено технічне завдання, інтерфейс та дизайн веб-застосунку

«A.Movie»

* розроблено власний веб-сайт, який є поєднанням ідеї соціальної мережі для кіноманів з послугами бронювання і купівлі квитків в онлайн кінотеатрах, а також додатково до сайту розроблено телеграм бота, який допомагає у роботі і доповнює веб-застосунок.

В майбутньому дана система може бути розширена шляхом подальшого збагачення функціональних можливостей: наприклад, стрічки кіноновин або навіть онлайн кінотеатру для перегляду кінофільмів та серіалів за окрему підписку.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. JetBrains [Електронний ресурс] – Режим доступу до ресурсу: [https://www.jetbrains.com](https://www.jetbrains.com/).
2. IMDb [Електронний ресурс] – Режим доступу до ресурсу: <https://www.imdb.com/>
3. Інформація про вебсайт IMDb [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/IMDb>
4. Multiplex [Електронний ресурс] – Режим доступу до ресурсу: <https://multiplex.ua/>
5. Java EE [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Java_EE>
6. Understanding the Basics of Spring vs. Spring Boot [Електронний ресурс] – Режим доступу до ресурсу: [https://dzone.com/articles/understanding-the-](https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot) [basics-of-spring-vs-spring-boot](https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot)
7. Spring Boot – Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/spring-boot-architecture/>
8. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/postgresql-vs-mysql>
9. Firebase [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/storage>
10. Google OAuth API [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.google.com/identity/protocols/oauth2>
11. Facebook API [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.facebook.com/docs/facebook-login/web/>
12. Maps JavaScript API [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.google.com/maps/documentation/javascript/overview>
13. Stripe [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Stripe_(company)>
14. Stripe documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://stripe.com/docs/api>
15. Телеграм [Електронний ресурс] – Режим доступу до ресурсу: <https://ru.wikipedia.org/wiki/Telegram>
16. Приклад реалізації телеграм боту на основі webhook [Електронний ресурс] – Режим доступу до ресурсу: [https://javarush.ru/groups/posts/3493-](https://javarush.ru/groups/posts/3493-telegram-bot--napominalka-cherez-webhook-na-java-ili-skazhi-net-google-kalendarju) [telegram-bot--napominalka-cherez-webhook-na-java-ili-skazhi-net-google-](https://javarush.ru/groups/posts/3493-telegram-bot--napominalka-cherez-webhook-na-java-ili-skazhi-net-google-kalendarju) [kalendarju](https://javarush.ru/groups/posts/3493-telegram-bot--napominalka-cherez-webhook-na-java-ili-skazhi-net-google-kalendarju)
17. FreeMarker [Електронний ресурс] – Режим доступу до ресурсу: <https://freemarker.apache.org/>
18. Spring Data JPA [Електронний ресурс] – Режим доступу до ресурсу: [https://thorben-janssen.com/what-is-spring-data-jpa-and-why-should-you-](https://thorben-janssen.com/what-is-spring-data-jpa-and-why-should-you-use-it/) [use-it/](https://thorben-janssen.com/what-is-spring-data-jpa-and-why-should-you-use-it/)
19. Hibernate [Електронний ресурс] – Режим доступу до ресурсу: [https://dzone.com/articles/what-is-the-difference-between-hibernate-and-](https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1) [sprin-1](https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1)
20. JPQL [Електронний ресурс] – Режим доступу до ресурсу: [https://thorben-](https://thorben-janssen.com/jpql/) [janssen.com/jpql/](https://thorben-janssen.com/jpql/)
21. DTO [Електронний ресурс] – Режим доступу до ресурсу: [https://www.javaguides.net/2021/02/spring-boot-dto-example-entity-to-](https://www.javaguides.net/2021/02/spring-boot-dto-example-entity-to-dto.html) [dto.html](https://www.javaguides.net/2021/02/spring-boot-dto-example-entity-to-dto.html)
22. AJAX [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/AJAX>

# ДОДАТКИ

## Додаток А. Діаграма класів

