МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

**КУРСОВА РОБОТА**

з курсу: «Об’єктно-орієнтоване програмування»

на тему: «Розроблення програмного забезпечення з використанням

об’єктно-орієнтованої парадигми»

Виконав: студент (ка) 2 курсу групи 621п

Спеціальність 121 – Інженерія програмного забезпечення

(код та найменування)

Бажанов В.О.

(прізвище й ініціали студента(ки))

Керівник: Лучшев П.О.

(прізвище й ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_ Лучшев П.О.

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_ Дем’яненко В.А.

(підпис) (прізвище та ініціали)

Харків – 2024

**ТИПОВЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

Варіант 1. Кінотеатр

|  |  |  |
| --- | --- | --- |
| **Критерій оцінювання** | **Бали** | **Штрафи** |
| **Покрокове виконання курсової роботи:** | | |
| Розділ 1(крайній термін – **3 тиждень**) | 0..15 | -3 |
| Розділ 2(крайній термін – **6 тиждень**) | 0..15 | -3 |
| Розділ 3(крайній термін – **9 тиждень**) | 0..15 | -3 |
| Розділ 4(крайній термін – **12 тиждень**) | 0..15 | -3 |
| Розділ 5(крайній термін – **13 тиждень**) | 0..5 | -1 |
| Оформлення пояснювальної записки (термін – **14 тиждень**) | 0..3 | -1 |
| **Підготовка до захисту курсової роботи:** | | |
| доповідь з презентацією (термін – **15 тиждень**) | 0..1 |  |
| відеоролик з демонстрацією роботи програми  (термін – **15 тиждень**) | 0..1 |  |
| **Захист курсової роботи:** | | |
| доповідь з презентацією | 0..5 |  |
| демонстрація роботи програми і github-репозиторію | 0..5 |  |
| відповіді на питання | 0..5 |  |
| використання\*:   1. інкапсуляції 2. статичних членів класів 3. інтерфейсів 4. абстрактних класів 5. спадкування 6. поліморфізму 7. .NET-delegates і events 8. сереалізації/десереалізації об’єктів 9. Collections.Generic 10. LINQ   \* мають бути представленні на захисті курсової роботи у презентації | 0..15 |  |
| **Всього за курсову роботу:** | **0..100** | **0..14** |

**Зміст**

[СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 2](#_Toc166323718)

[ВСТУП 2](#_Toc166323719)

[1 ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 2](#_Toc166323720)

[1.1 Аналіз функціональності програмних аналогів 2](#_Toc166323721)

[1.2 Побудова Mind Map для заданої предметної області 2](#_Toc166323722)

[1.3 Виділення ролей користувачів і формулювання функціональних вимог 2](#_Toc166323723)

[1.4 Проєктування інтерфейсу користувача 2](#_Toc166323724)

[1.5 Розроблення функціональних тестів 2](#_Toc166323725)

[1.5.1 Функціональні тести для ролі користувача «Адміністратор» 2](#_Toc166323726)

[1.5.2 Функціональні тести для ролі користувача «Гість» 2](#_Toc166323727)

[2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 2](#_Toc166323728)

[2.1 Виділення і опис класів предметної області 2](#_Toc166323729)

[2.2 Встановлення зв’язків між класами 2](#_Toc166323730)

[3 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ПРЕДМЕТНОЇ ОБЛАСТІ І ЇХ ТЕСТУВАННЯ 2](#_Toc166323731)

[3.1 Структура проєкту з реалізацією класів предметної області 2](#_Toc166323732)

[3.2 Реалізація інтерфейсів і каркасів класів предметної області 2](#_Toc166323733)

[3.3 Розроблення unit-тестів для класів предметної області 2](#_Toc166323734)

[3.4 Повна реалізація класів предметної області 2](#_Toc166323735)

[3.5 Результати unit-тестування класів предметної області 2](#_Toc166323736)

[4 ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ІНТЕРФЕЙСУ КОРИСТУВАЧА 2](#_Toc166323737)

[4.1 Структура проєкту з реалізацією класів інтерфейсу користувача 2](#_Toc166323738)

[4.2 Виділення класів для реалізації інтерфейсу користувача 2](#_Toc166323739)

[4.3 Програмна реалізація класів інтерфейсу користувача 2](#_Toc166323740)

[5 ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 2](#_Toc166323741)

[5.1 Функціональне тестування для ролі користувача «Адміністратор» 2](#_Toc166323742)

[5.2 Функціональне тестування для ролі користувача «Зареєстрований користувач» 2](#_Toc166323743)

[5.3 Функціональне тестування для ролі користувача «Гість» 2](#_Toc166323744)

[ВИСНОВКИ 2](#_Toc166323745)

[ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ 2](#_Toc166323746)

[ДОДАТОК А. Лістинг класів предметної області 2](#_Toc166323747)

[ДОДАТОК Б. Лістинг класів інтерфейсу користувача 2](#_Toc166323748)

[ДОДАТОК В. Лістинг класів unit-тестів 2](#_Toc166323749)

[ДОДАТОК Г. Назва додатку 2](#_Toc166323750)

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

|  |  |  |
| --- | --- | --- |
| CRUD | – | (*англ*. Create Read Update Delete) основні функції управління даними «створення, читання, оновлення і вилучення». |
| ООП | – | об’єктно-орієнтоване програмування |
| ORM | – | (*англ*. Object-Relational Mapping) підхід до програмування, який дозволяє взаємодіяти з базою даних у формі об'єктів програмної мови програмування, замість написання запитів мовою структурованих запитів бази даних |

# ВСТУП

В сучасному світі кінотеатри залишаються важливим культурним центром, де глядачі можуть насолоджуватися найновішими фільмами на великих екранах. З ростом популярності кінематографії, кінотеатри стають популярним місцем відпочинку для різних груп людей.

В кіно театрах часто можна побачити менше черг, порівняно з іншими місцями перегляду фільмів, що забезпечує зручніше відвідування. Крім того, атмосфера кінозалу та можливість насолоджуватися фільмом на великому екрані роблять кінотеатри привабливими для широкого кола глядачів.

Потенційні глядачі кінотеатрів - це люди, які цінують якість відтворення фільмів та хочуть насолоджуватися кіно на великому екрані. Це можуть бути кіномани, які завжди стежать за новинками у світі кінематографу, або просто любителі хорошого кіно.

Серед кінотеатрів в Україні можна виділити такі заклади "Multiplex", або "Планета Кіно".

Під час роботи над курсовою роботою, буде використано такий стек-технологій, як: ASP.NET Core MVC - фреймворк для побудови веб-додатків на платформі .NET Core; Entity Framework - ORM, який надає доступ до бази даних у формі об'єктів, дозволяючи взаємодіяти з базою даних через об'єктно-орієнтований підхід; База даних - я використав SQL Server для зберігання даних мого проекту.

# ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Аналіз функціональності програмних аналогів

1. Multiplex [[2](#_ПЕРЕЛІК_ДЖЕРЕЛ_ТА)]

Сервіс дозволяє передивитись інформацію про фільми які йдуть в прокаті. Присутня можливість авторизації на сайту.

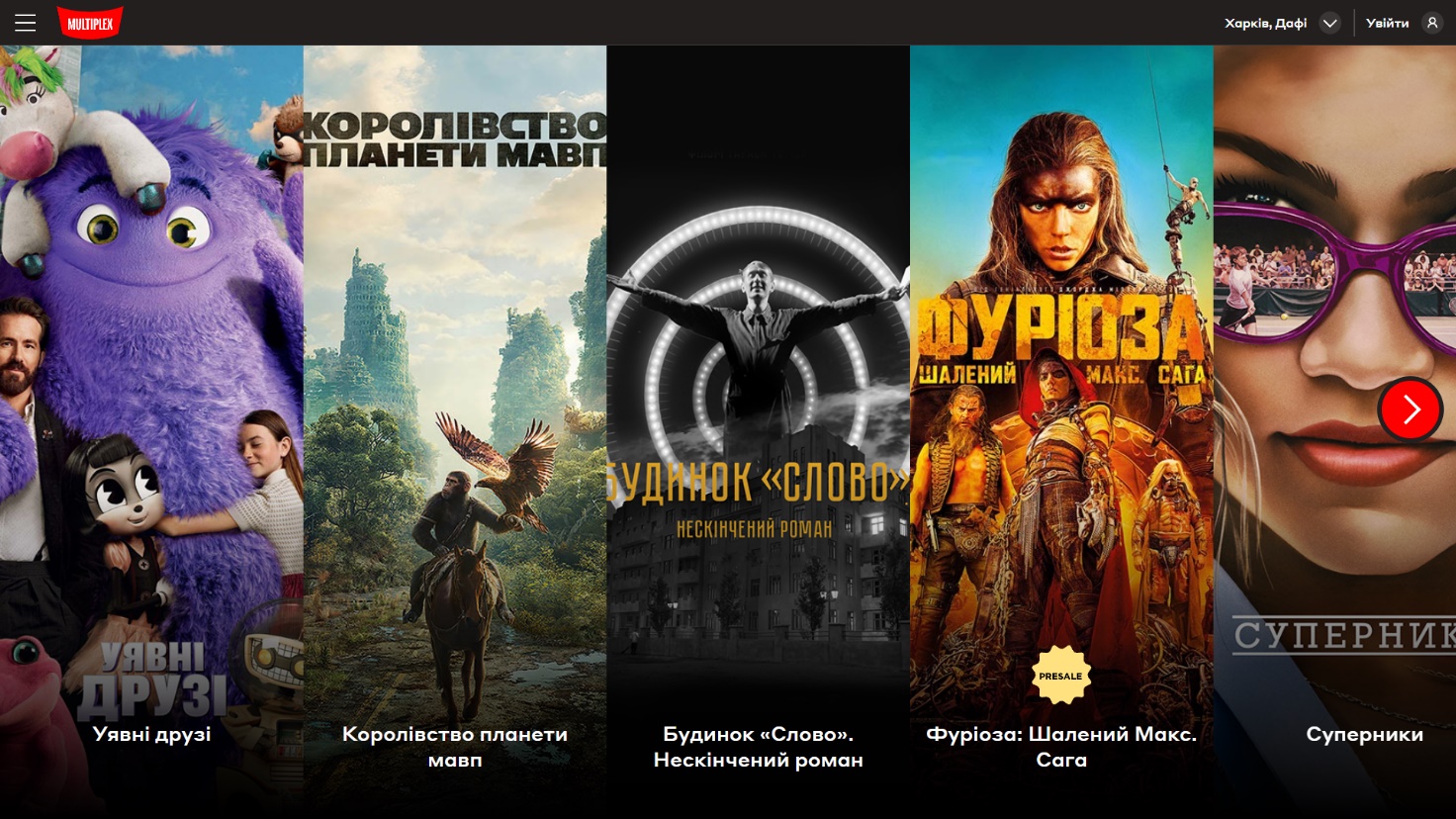


Рисунок 1.1 – Головна сторінка сайту «Multiplex»

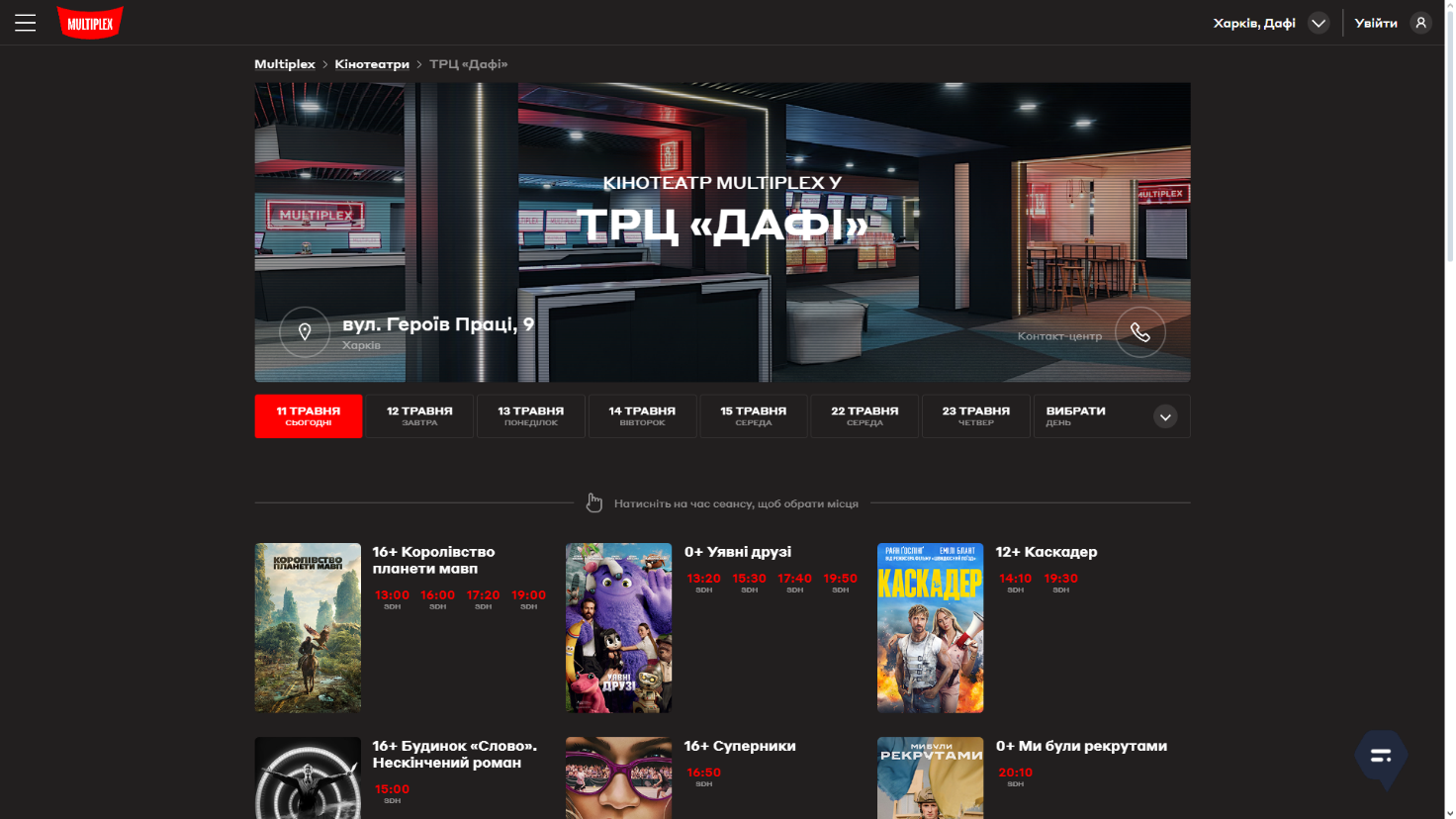


Рисунок 1.2 – Сторінка із розкладом сеансів

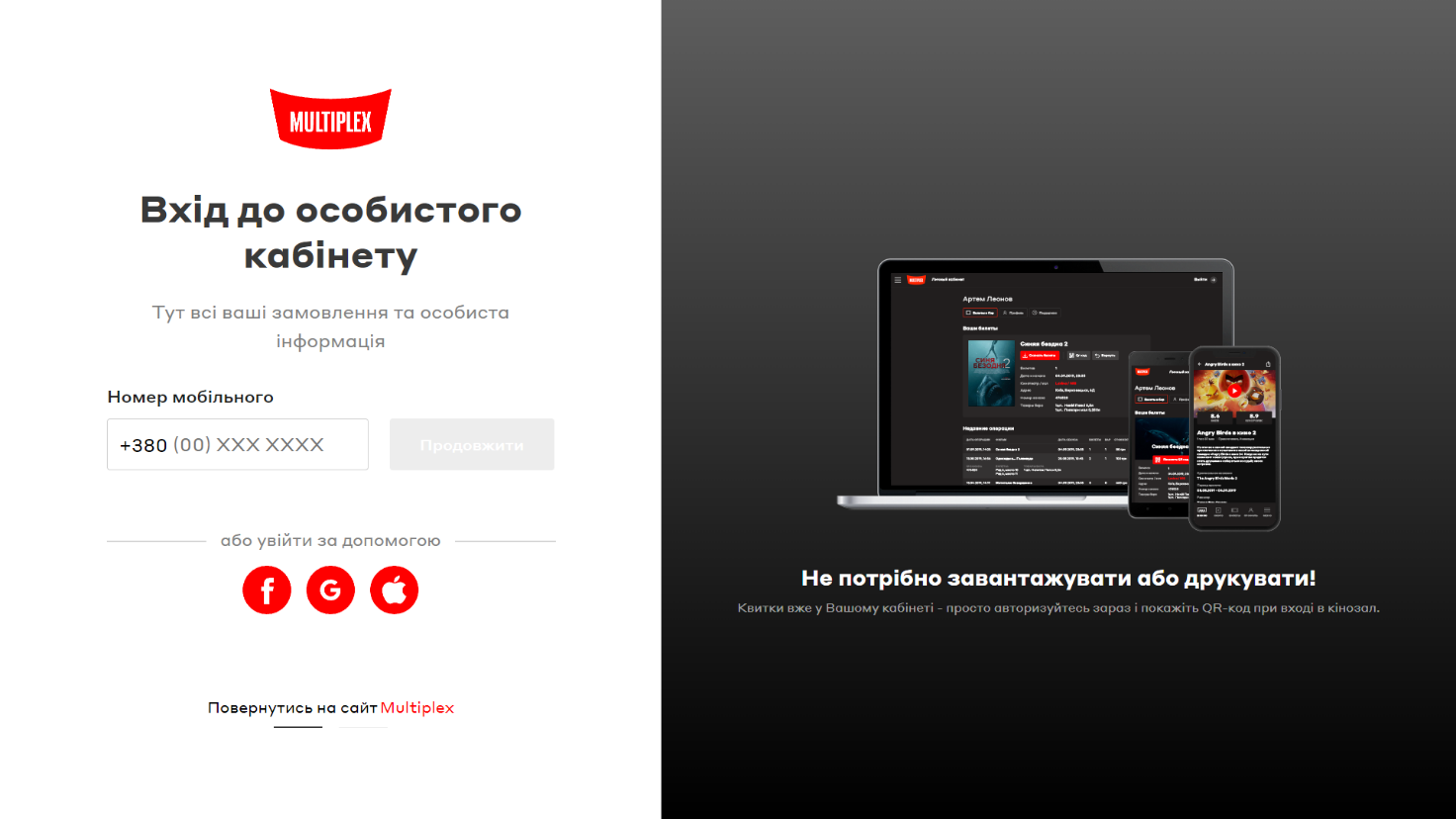


Рисунок 1.3 – Авторизація

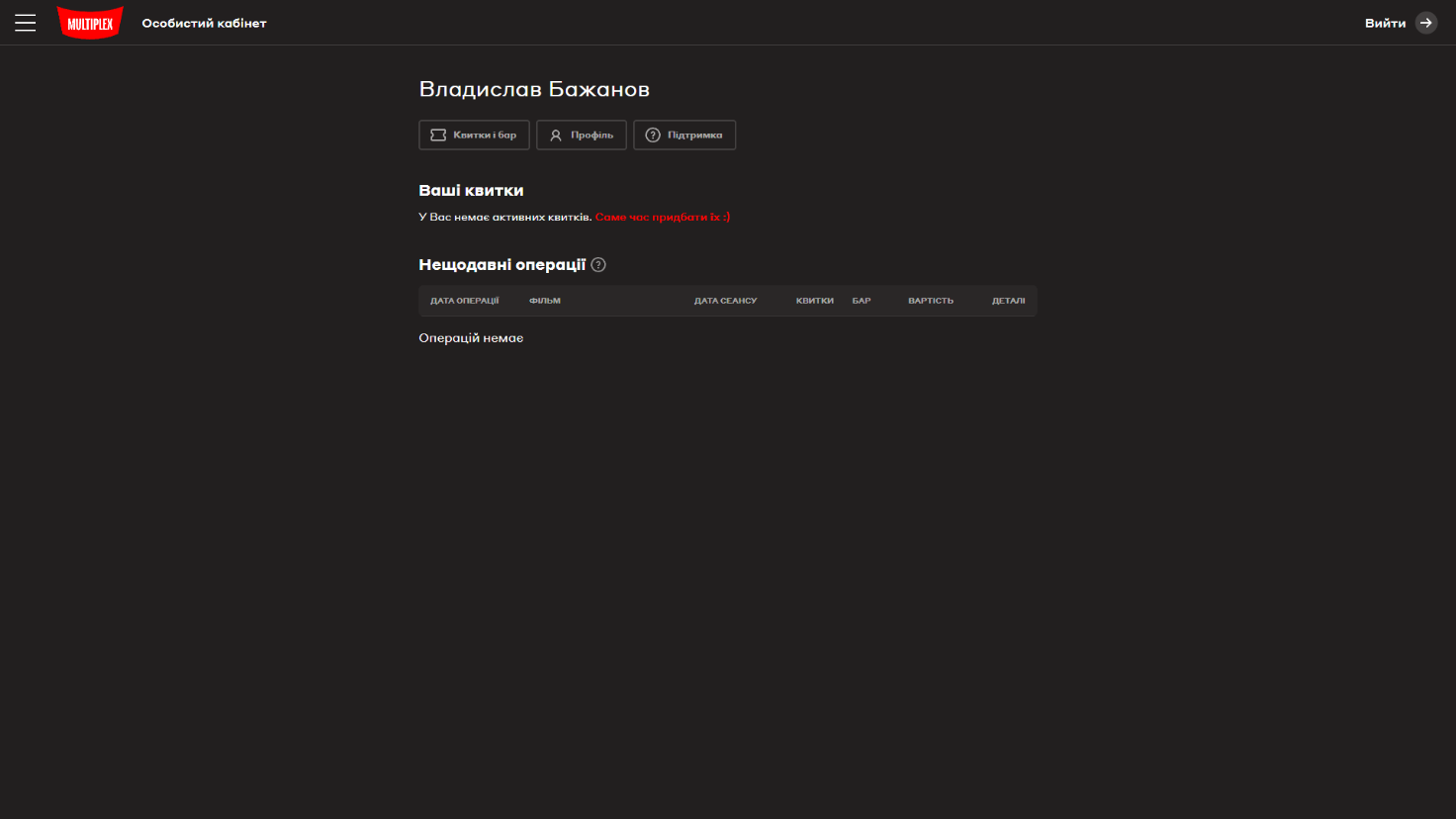


Рисунок 1.4 – Особистий профіль

1. Планета кіно [[2](#_ПЕРЕЛІК_ДЖЕРЕЛ_ТА)]

Сервіс дозволяє передивитися інформацію про фільми які йдуть в прокаті. Також доступна можливість авторизуватись/зареєструватись.

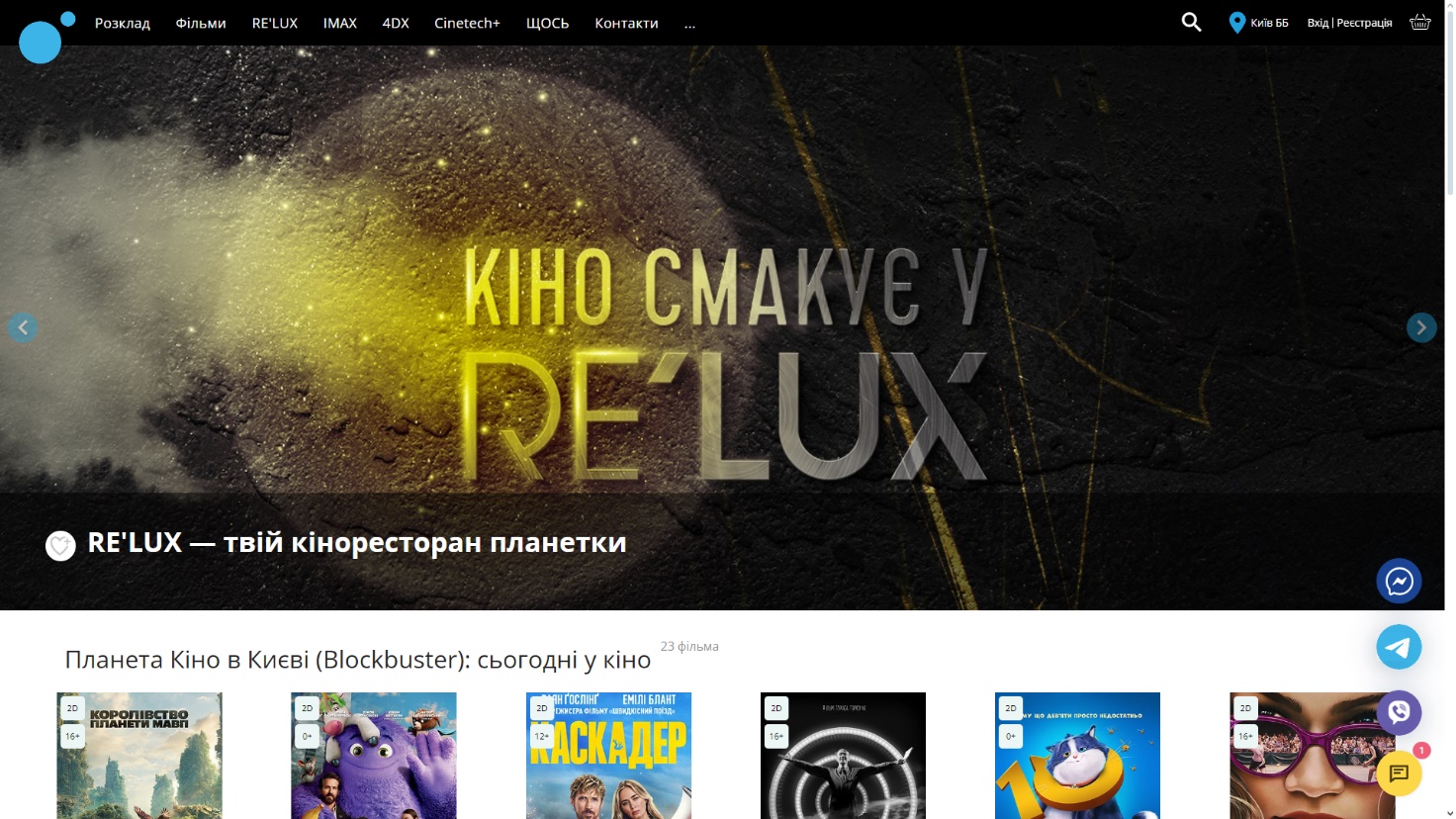


Рисунок 1.5 – Головна сторінка сайту «Планета кіно»

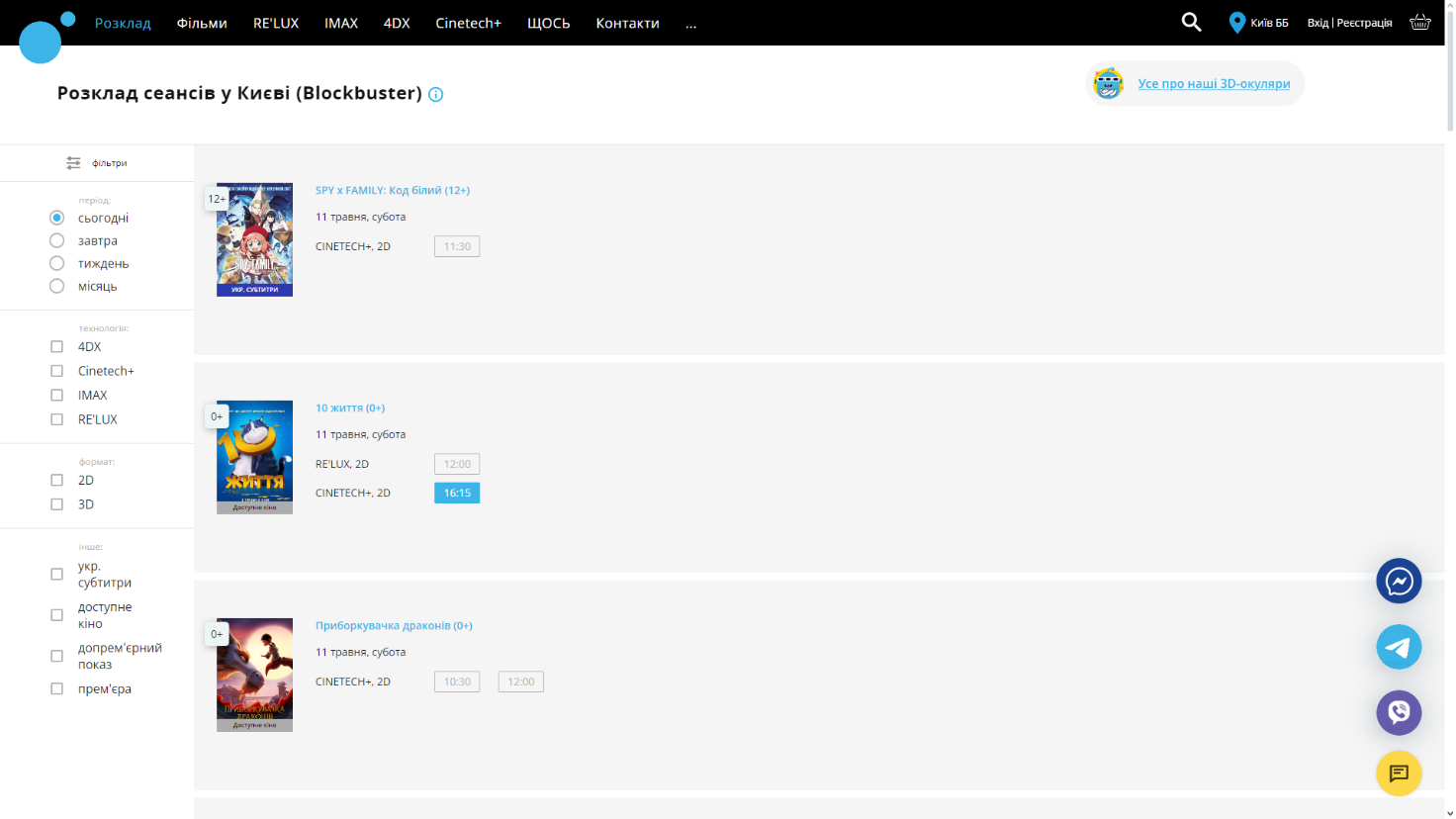


Рисунок 1.6 – Сторінка із розкладом сеансів

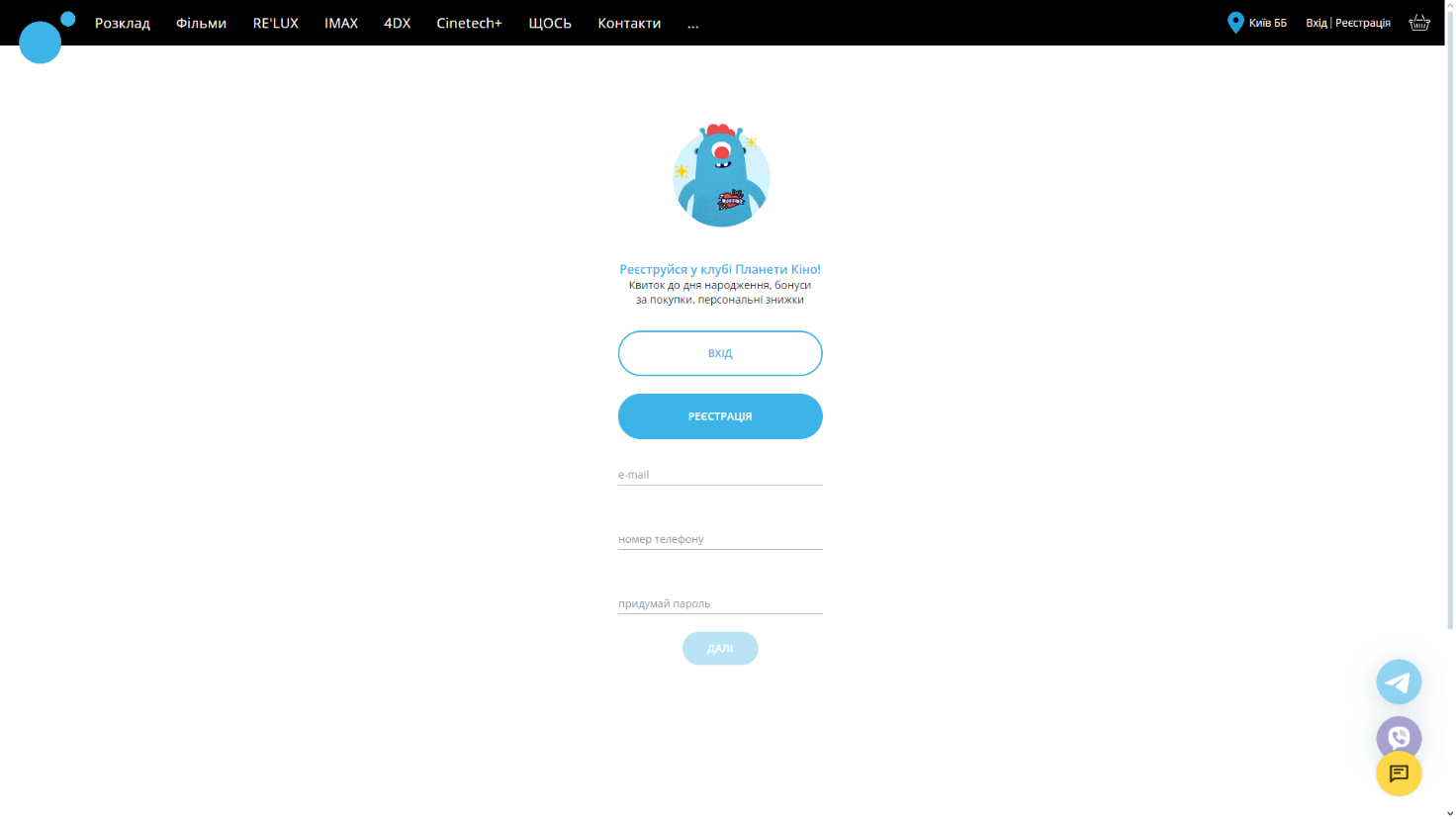


Рисунок 1.7 – Сторінка авторизації

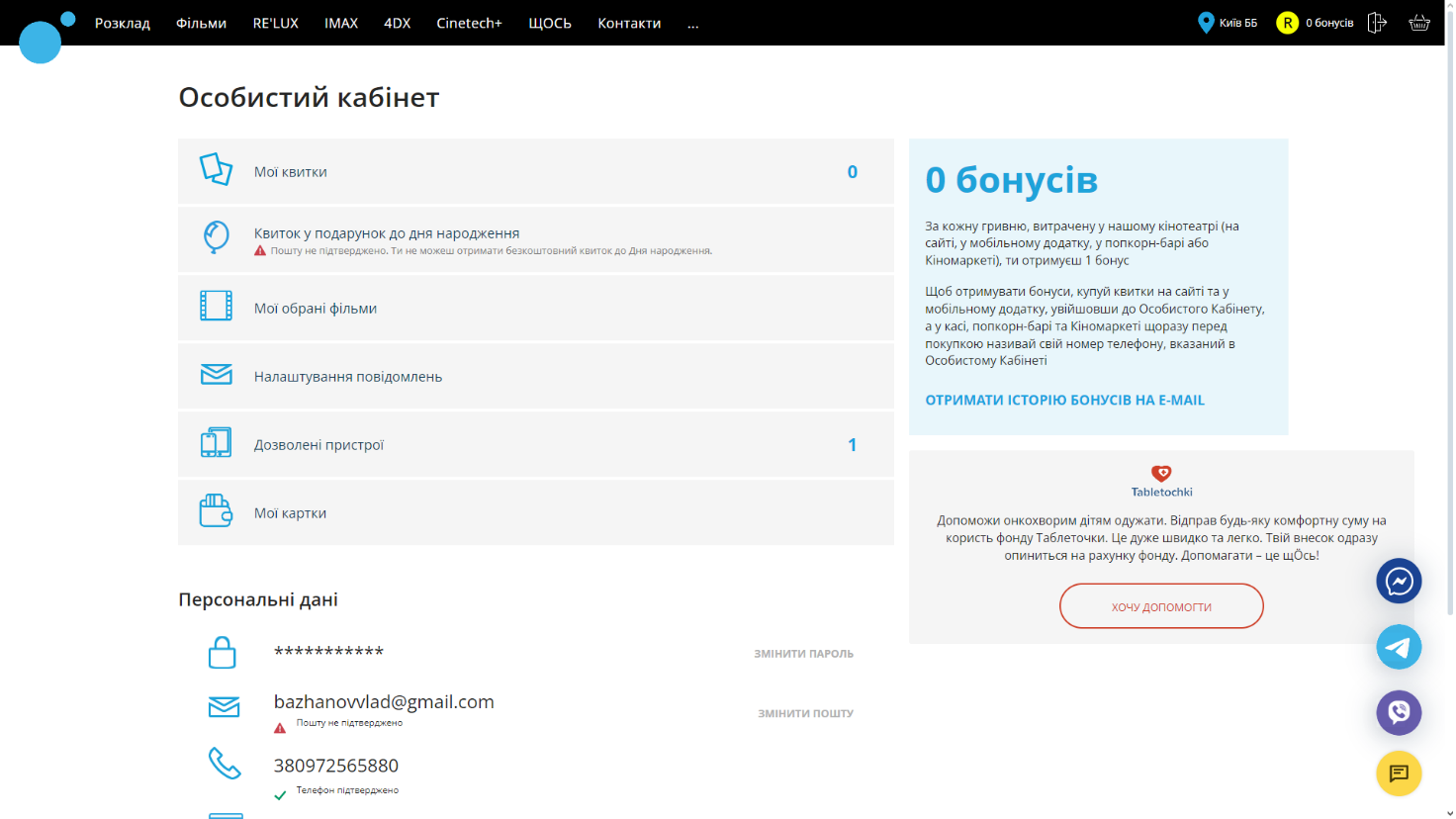


Рисунок 1.7 – Особистий кабінет

## Побудова Mind Map для заданої предметної області



Рисунок 1.8 – Mind map

## Виділення ролей користувачів і формулювання функціональних вимог

Були виділені три ролі користувачів: «Адміністратор», «Зареєстрований користувач», «Гість».

Опишемо функціональні вимоги кожного типу користувачів.

Таблиця 1.1 – Функціональні вимоги користувача з роллю «Адміністратор»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| А.1. | Адміністратор може додавати, редагувати та видаляти фільми. |
| А.2. | Адміністратор може додавати та редагувати та видаляти користувачів. |
| А.3. | Адміністратор може додавати, видаляти або редагувати статті із питаннями. |

Таблиця 1.2 – Функціональні вимоги користувача з роллю «Зареєстрований користувач»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| ЗК.1. | Зареєстрований користувач може купити квитки. |
| ЗК.2. | Зареєстрований користувач може редагувати особистий профіль.  ***Примітка:***   1. можлива зміна пароля; 2. можлива зміна номеру телефону; 3. можлива зміна імені. |

Таблиця 1.3 – Функціональні вимоги користувача з роллю «Гість»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| Г.1. | Гість може подивитися фільми в прокаті а також найчастіші запитання. |

## Проєктування інтерфейсу користувача

Для реалізації замовником було обрано web-застосунок.

На основі сформульованих функціональних вимог були розроблені екранні форми, які наведені на рис. 1.0 – 1.2).

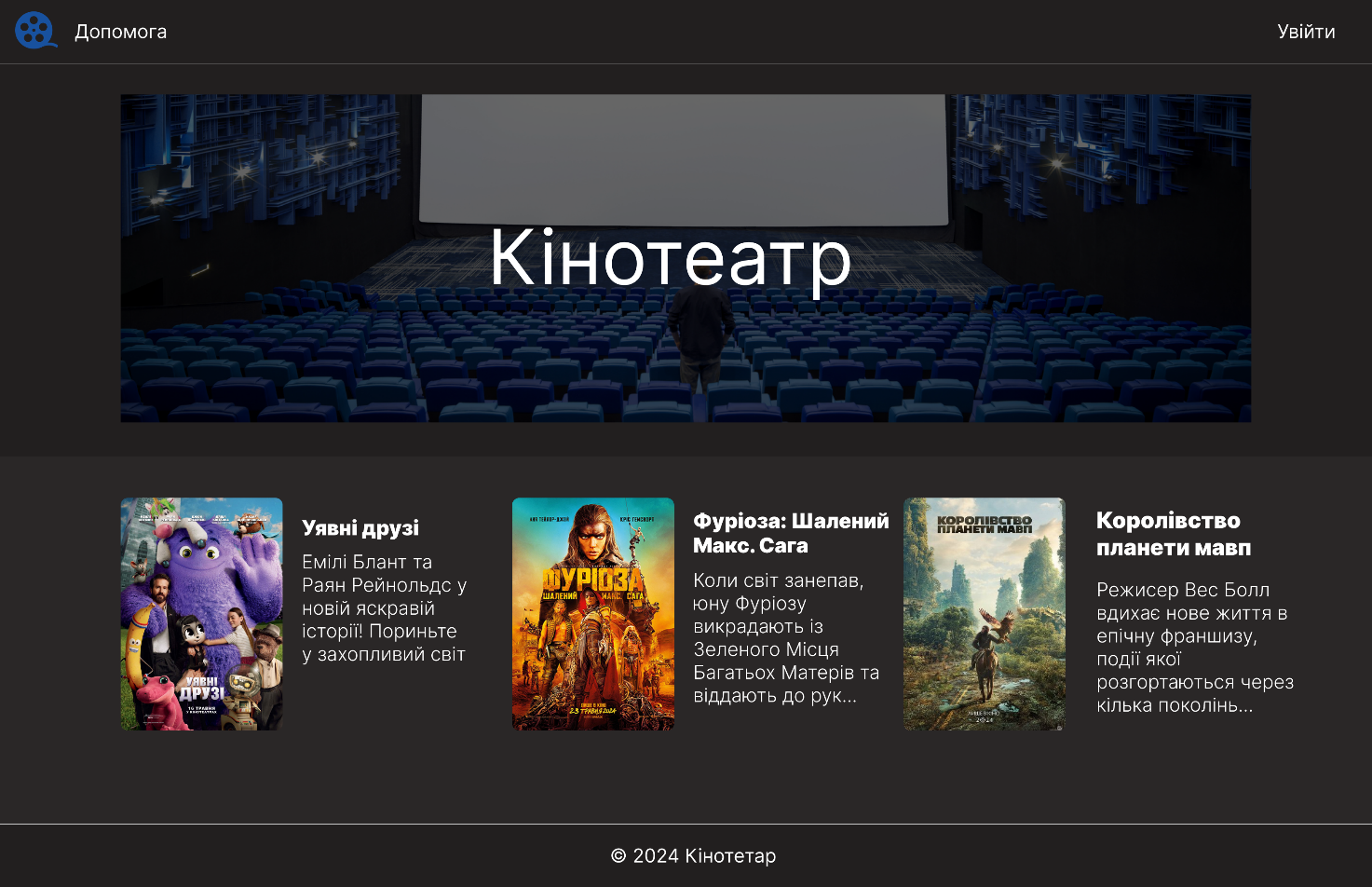


Рисунок 1.9 – Макет екранної форми «Головна сторінка»

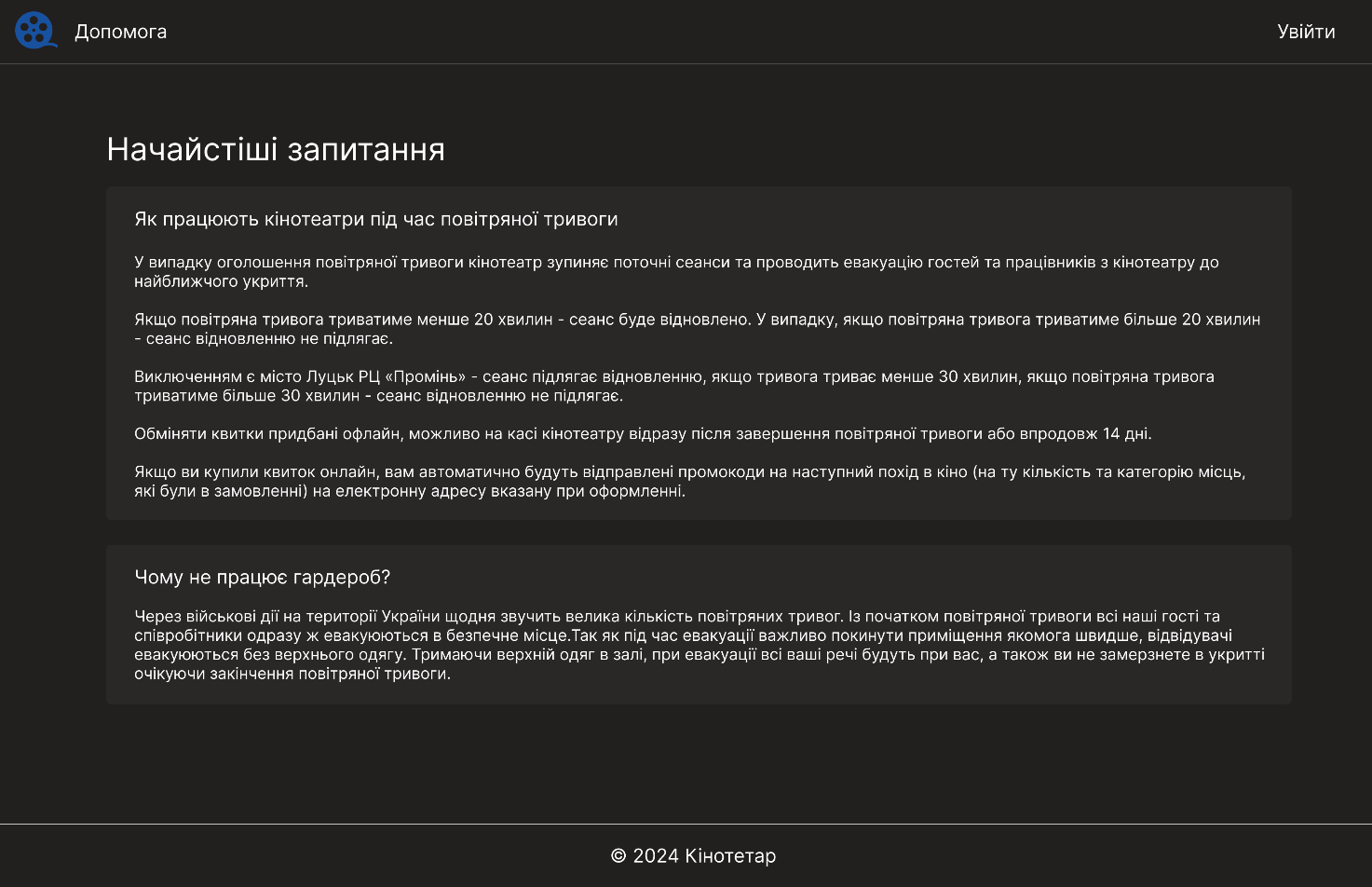


Рисунок 1.10 – Макет екранної форми «Найчастіші запитання»

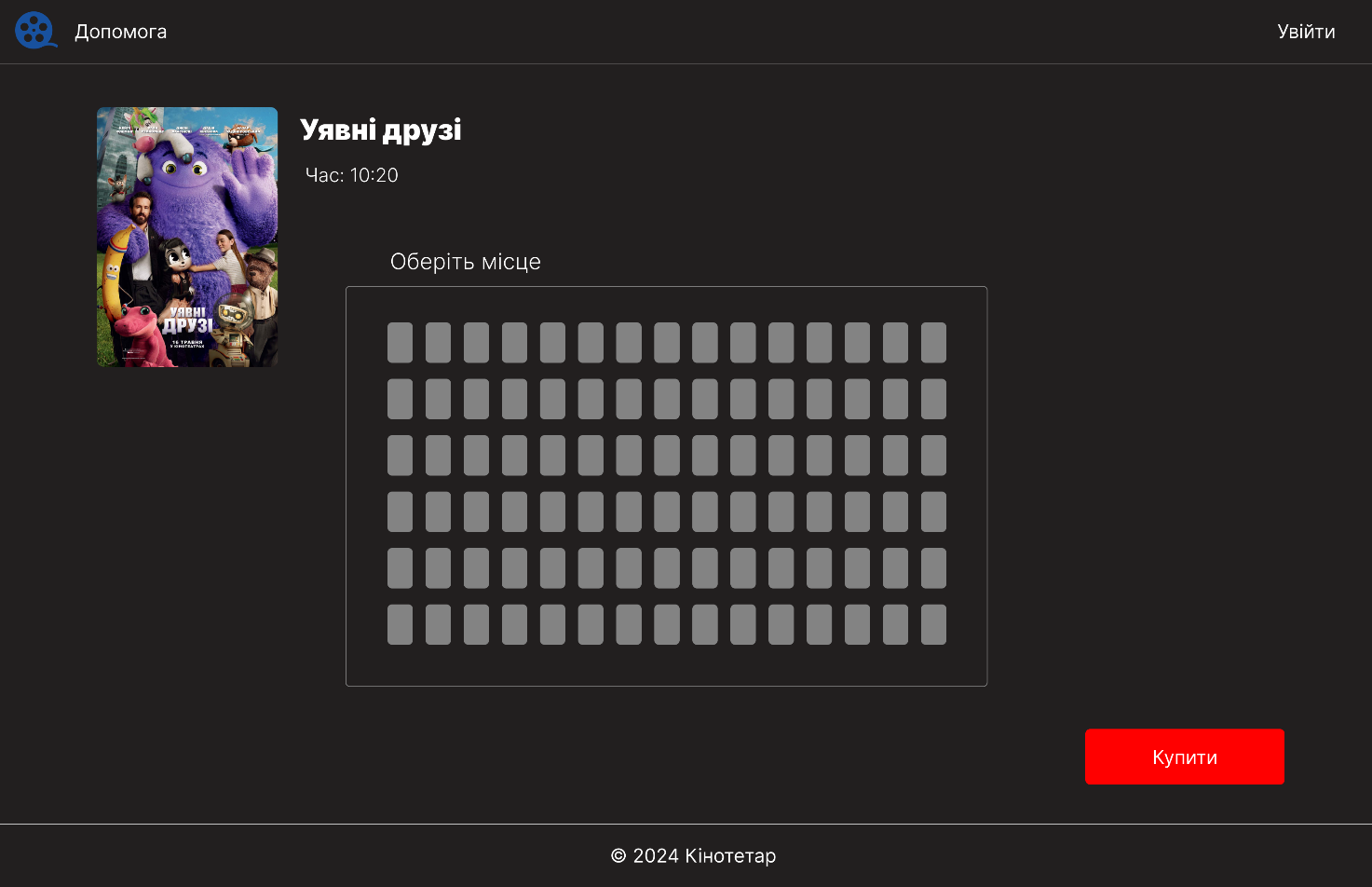


Рисунок 1.12 – Макет екранної форми «Придбання квитків»

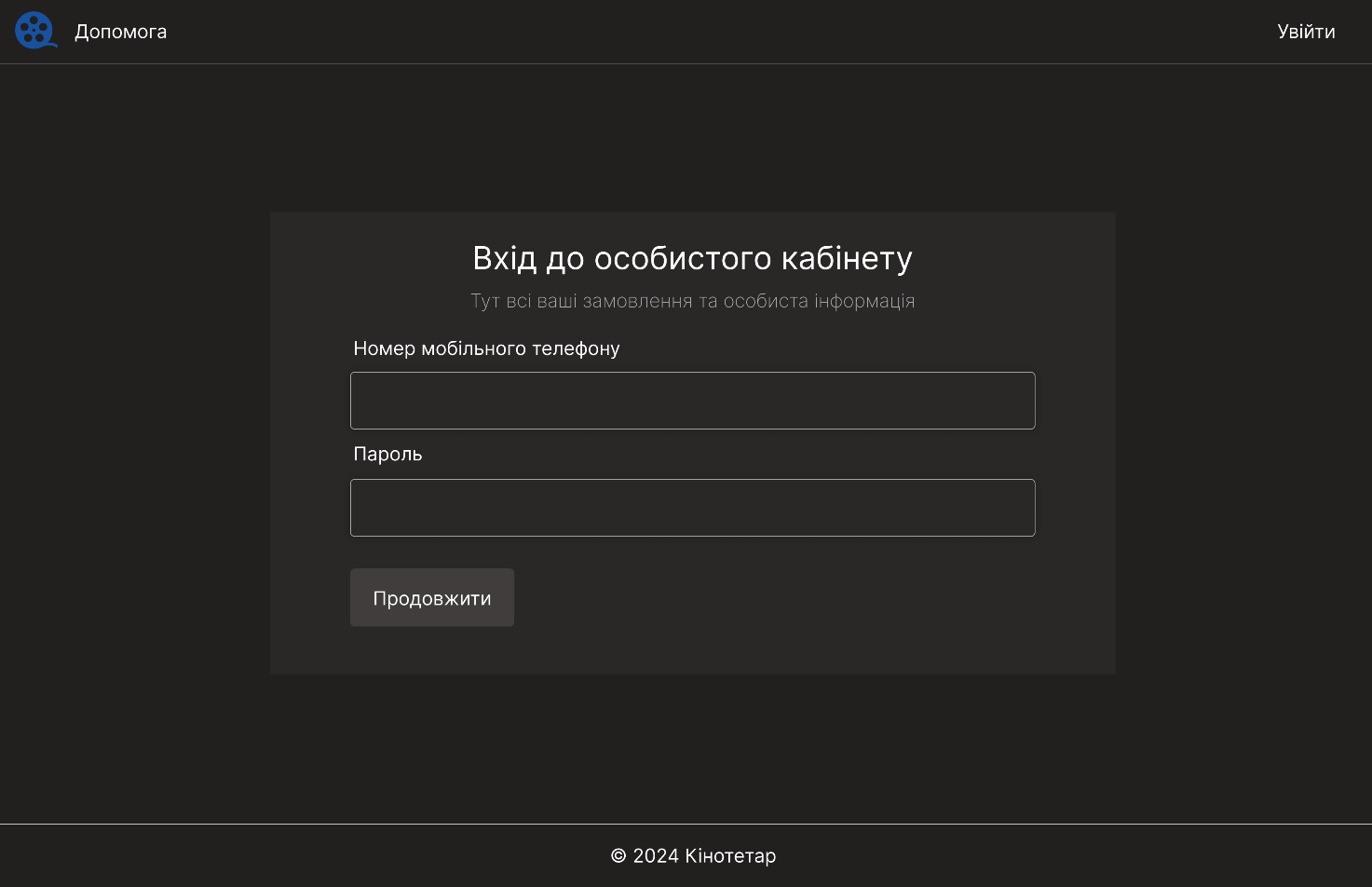


Рисунок 1.13 – Макет екранної форми «Авторизація»

## Розроблення функціональних тестів

### Функціональні тести для ролі користувача «Адміністратор»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Адміністратор» (див. табл. 1.4).

Таблиця 1.4 – Функціональні тести для ролі користувача «Адміністратор»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *А.1. Адміністратор може додавати/редагувати/видаляти фільми.* | | |
| А.1.1 | 1. Ввести назву. 2. Вставити зображення. 3. Написати опис. 4. Додати сеанси. 5. Натиснути кнопку «Додати». | Успішне додавання фільму |
| А.1.2 | 1. Натиснути кнопку «Редагувати». 2. Змінити назву. 3. Змінити опис. 4. Натиснути кнопку «Зберегти». | Успішне редагування фільму |
| А.1.3 | Натиснути кнопку «Видалити». | Успішне видалення фільму |
| А.1.4 | 1. Не ввести назву. 2. Вставити зображення. 3. Написати опис. 4. Додати сеанси. 5. Натиснути кнопку «додати». | Фільм не був доданий. Користувач отримав повідомлення «Не були введені всі данні!» |
| А.1.5 | 1. Натиснути кнопку «Редагувати». 2. Стерти назву. 3. Змінити опис. 4. Натиснути кнопку «Зберегти». | Фільм не був відредагований. Користувач отримав повідомлення «Не були введені всі данні!» |
| *А.2. Адміністратор може додавати/редагувати або видаляти статті.* | | |
| А.2.1 | 1. Ввести заголовок. 2. Ввести текст після заголовку. 3. Натиснути кнопку «Зберегти». | Успішне додавання статті |
| А.2.2 | 1. Змінити заголовок. 2. Змінити текст після заголовку. 3. Натиснути кнопку «Зберегти». | Успішне редагування статті |
| А.2.3 | 1. Натиснути кнопку «Видалити». | Успішне видалення статті |
| А.2.4 | 1. Не вводити текст заголовка. 2. Ввести текст після заголовку. 3. Натиснути кнопку «Зберегти». | Стаття не була додана. Користувач отримав повідомлення «Не були введені всі данні!» |
| А.2.5 | 1. Стерти текст заголовка. 2. Змінити текст після заголовку. 3. Натиснути кнопку «Зберегти». | Стаття не була відредагована. Користувач отримав повідомлення «Не були введені всі данні!» |
| *А.3. Адміністратор може додавати/редагувати або видаляти користувачів.* | | |
| А.3.1 | 1. Ввести ім’я та фамілію. 2. Ввести номер телефона. 3. Ввести пошту. 4. Ввести пароль. 5. Натиснути кнопку «Зберегти». | Успішне додавання користувача |
| А.3.2 | 1. Натиснути кнопку «Редагувати». 2. Змінити ім’я та фамілію. 3. Змінити номер телефона. 4. Змінити пошту. 5. Змінити пароль. 6. Натиснути кнопку «Зберегти». | Успішне редагування користувача |
| А.3.3 | 1. Натиснути кнопку «Видалити». | Успішне видалення користувача |
| А.3.4 | 1. Ввести ім’я та фамілію. 2. Не ввести номер телефона. 3. Ввести пошту. 4. Ввести пароль. 5. Натиснути кнопку «Зберегти». | Користувач не був доданий. Користувач отримав повідомлення «Не були введені всі данні!» |
| А.3.5 | 1. Ввести ім’я та фамілію. 2. Стерти номер телефона. 3. Ввести пошту. 4. Ввести пароль. 5. Натиснути кнопку «Зберегти». | Користувач не був відредагований. Користувач отримав повідомлення «Не були введені всі данні!» |

### Функціональні тести для ролі користувача «Зареєстрований користувач»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Зареєстрований користувач» (див. табл. 1.5).

Таблиця 1.5 – Функціональні тести для ролі користувача «Зареєстрований користувач»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор**  **тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *ЗК.1. Зареєстрований користувач може змінити особисті дані.* | | |
| ЗК.1.1 | 1. Змінити номер телефону. 2. Змінити ім’я користувача. 3. Натиснути кнопку «Зберегти». | Успішна зміна особистих даних |
| ЗК.1.2 | 1. Змінити номер телефону. 2. Стерти ім’я користувача. 3. Натиснути кнопку «Зберегти». | Дані не були зміненні. Користувач отримав повідомлення «Не були введені всі данні!» |
| *ЗК.2. Зареєстрований користувач може купити квитки.* | | |
| ЗК.2.1 | 1. Перейти в розділ з фільмами. 2. Вибрати фільм. 3. Вибрати час сеансу 4. Натиснути кнопку «Купити». | Успішна покупка квитка |

### Функціональні тести для ролі користувача «Гість»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Гість» (див. табл. 1.6).

Таблиця 1.6 – Функціональні тести для ролі користувача «Гість»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор**  **тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *Г.1. Гість подивитись сеанси та найчастіші запитання.* | | |
| Г.1.1 | 1. Перейти в розділ «Найчастіші запитання» 2. Перейти в розділ «Сеанси» | Користувач успішно зміг передивитись наявну інформацію |

# МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

## Виділення і опис класів предметної області

**Клас «User» – користувач**

Таблиця 2.1 – Виділення характеристик класу «User»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Електронна пошта | Рядок | Не пустий рядок, наявність символу ‘@’ |
| Ім’я | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Телефон | Рядок | Не пустий рядок, довжина: 14 символів |
| Роль | Enum | User, Admin |
| Пароль | Рядок | Не пустий рядок, довжина: від 6 символів |

Таблиця 2.2 – Виділення поведінки класу «User»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Бізнес-правила** |
| Оформити купівлю білетів | Фільм,  дата | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то оформити замовлення. |
| Зміна особистих даних | Номер телефону, пароль, ім’я | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то змінити дані користувача. |
| Додавання нового користувача | Електронна пошта, Ім’я, Телефон, Роль, Пароль | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то додати користувача. |
| Редагування користувача | Електронна пошта, Ім’я, Телефон, Роль | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то змінити користувача. |
| Видалення користувача | Ідентифікаційний номер | IActionResult | Якщо існує такий користувач, то він буде видалений |

**Клас «Film» – фільм**

Таблиця 2.3 – Виділення характеристик класу «Film»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Назва фільма | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Постер | Рядок (посилання) | Рядок який містить посилання |
| Опис | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Сеанси | Список | Не пустий список із датами, довжина: хоча б 1 символ |

Таблиця 2.4 – Виділення поведінки класу «Film»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Додання фільму | Назва фільму, постер, опис, сеанси | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то додати фільм в БД |
| Редагування фільму | Назва фільму, опис | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то зберегти відредагований фільм |
| Видалення фільму | Ідентифікаційний номер фільму | IActionResult | Якщо існує такий фільм, то він буде видалений |

**Клас «ModelHelp» – стаття**

Таблиця 2.5 – Виділення характеристик класу «ModelHelp»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Заголовок | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Опис | Рядок | Не пустий рядок, довжина: хоча б 1 символ |

Таблиця 2.6 – Виділення поведінки класу «ModelHelp»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Створення статті | Заголовок, опис | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то додати статтю в БД |
| Редагування статті | Заголовок, опис | IActionResult | 1. Перевірити чи дійсні вхідні дані. 2. Якщо пройдені перевірки, то зберегти відредаговану статтю |
| Видалення статті | Ідентифікаційний номер статті | IActionResult | Якщо така стаття існує то вона видаляється |

## Встановлення зв’язків між класами

На рисунку 0.3 наведена діаграма класів, яка була побудова з використанням сервісу <https://www.lucidchart.com>.

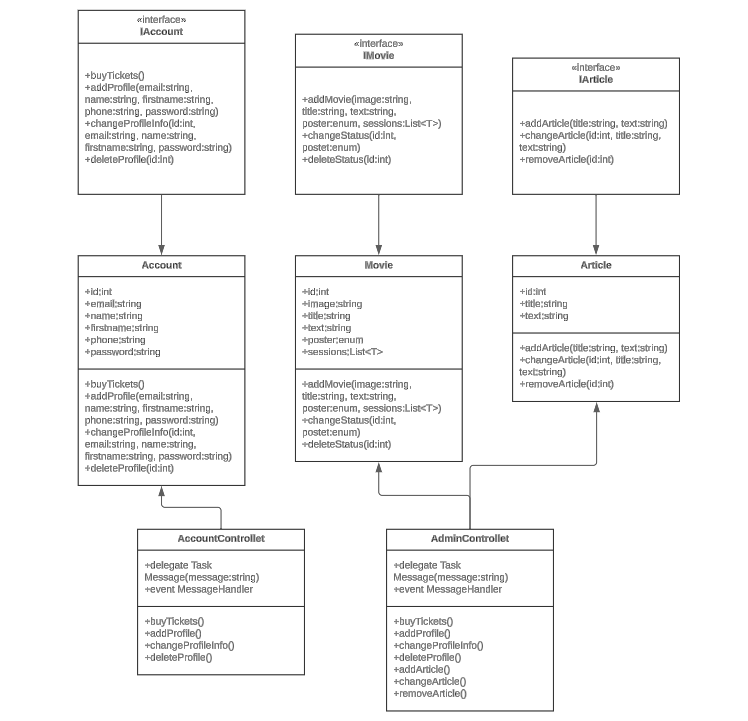


Рисунок 2.1 – Діаграма класів

# ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ПРЕДМЕТНОЇ ОБЛАСТІ І ЇХ ТЕСТУВАННЯ

## Структура проекту з реалізацією класів предметної області

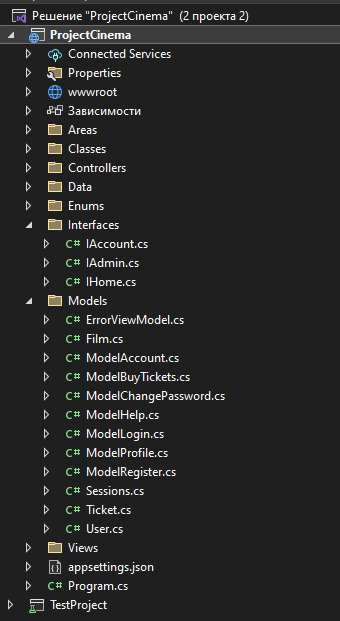


Рисунок 3.1 – Структура проекту з реалізацією предметної області

## Реалізація інтерфейсів і каркасів класів предметної області

**Інтерфейс «IAccount» – користувач**

using Microsoft.AspNetCore.Mvc;

namespace ProjectCinema.Interfaces

{

public interface IAccount

{

Task<IActionResult> Login();

Task<IActionResult> Register();

Task<IActionResult> Logout();

Task<IActionResult> Profile();

Task<IActionResult> ChangePassword();

Task<IActionResult> History();

}

}

**Інтерфейс «IAdmin» – адміністратор**

using Microsoft.AspNetCore.Mvc;

namespace ProjectCinema.Interfaces

{

public interface IAdmin

{

Task<IActionResult> ManageAccounts();

Task<IActionResult> ManageFilms();

Task<IActionResult> ManageArticles();

Task<IActionResult> AddAccount();

Task<IActionResult> AddFilm();

Task<IActionResult> AddArticle();

Task<IActionResult> EditAccount(string id);

Task<IActionResult> EditFilm(int id);

Task<IActionResult> EditArticle(int id);

Task<IActionResult> DeleteAccount(string id);

Task<IActionResult> DeleteFilm(int id);

Task<IActionResult> DeleteArticle(int id);

}

}

**Інтерфейс «IHome» – користувач**

using Microsoft.AspNetCore.Mvc;

namespace ProjectCinema.Interfaces

{

public interface IHome

{

IActionResult Index();

Task<IActionResult> Help();

Task<IActionResult> Movie(int id);

Task<IActionResult> BuyTickets(int Id);

IActionResult ErrorTickets();

}

}

**Клас «User» – користувач**

using Microsoft.AspNetCore.Identity;

using ProjectCinema.Enums;

namespace ProjectCinema.Models

{

public class User : IdentityUser

{

}

}

**Клас «Film» – квиток**

namespace ProjectCinema.Models

{

public class Film

{

}

}

**Клас «ModelHelp» – квиток**

namespace ProjectCinema.Models

{

public class ModelHelp

{

}

}

**Клас «Ticket» – квиток**

namespace ProjectCinema.Models

{

public class Ticket

{

}

}

## Розроблення unit-тестів для класів предметної області

Розроблені unit-тести наведені у [Додатку В](#_ДОДАТОК_В._Лістинг).

Таблиця 3.1 – Покриття unit-тестами класу «User»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| User | 1 |

Таблиця 3.2 – Покриття unit-тестами класу «Film»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| Film | 1 |

Таблиця 3.3 – Покриття unit-тестами класу «ModelHelp»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| ModelHelp | 1 |

Таблиця 3.4 – Покриття unit-тестами класу «Ticket»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| Ticket | 1 |

## Повна реалізація класів предметної області

У [Додатку А](#_ДОДАТОК_А._Лістинг) наведено повну реалізацію програмного коду розроблених класів предметної області.

## Результати unit-тестування класів предметної області

На рисунках 0.5 – 0.7 наведено скріншоти проходження розроблених unit-тестів для класів предметної області.



Рисунок 3.2 – Скріншот проходження unit-тестів для класу «User»



Рисунок 3.3 – Скріншот проходження unit-тестів для класу «Film»



Рисунок 3.4 – Скріншот проходження unit-тестів для класу «ModelHelp»



Рисунок 3.4 – Скріншот проходження unit-тестів для класу «Ticket»

# ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ІНТЕРФЕЙСУ КОРИСТУВАЧА

## Структура проекту з реалізацією класів інтерфейсу користувача

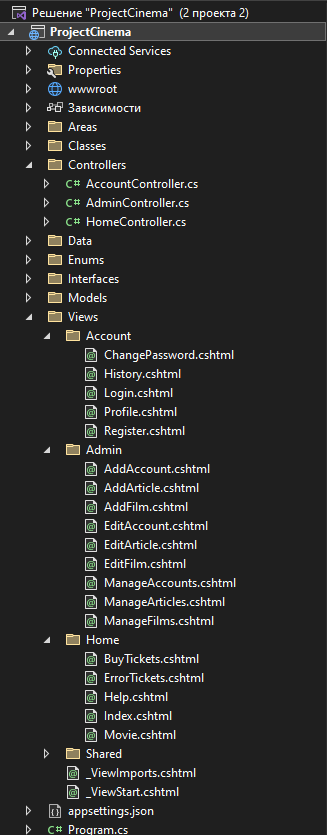


Рисунок 4.1 – Структура проекту з інтерфейсом користувача

## Виділення класів для реалізації інтерфейсу користувача

Класи HomeController, AccountController, AdminController успадковують базовий клас Controller. Це дає можливість використовувати всі функціональні можливості, які надає Controller.

Методи класу HomeController:

1. Index() (IActionResult):

* Метод для обробки GET-запитів до головної сторінки.
* Отримує список фільмів з бази даних та передає його до представлення (View).

1. Help() (Task<IActionResult>):

* Асинхронний метод для обробки GET-запитів до сторінки допомоги.
* Отримує список статей з бази даних та передає його до представлення (View).

1. Movie(int id) (Task<IActionResult>):

* Асинхронний метод для обробки GET-запитів до сторінки конкретного фільму.
* Отримує фільм з бази даних за ідентифікатором та передає його до представлення (View).

1. BuyTickets(int Id) (Task<IActionResult>):

* Асинхронний метод для обробки GET-запитів до сторінки покупки квитків.
* Отримує фільм та його сеанси з бази даних за ідентифікатором та передає їх до представлення (View).
* Включає логування, яке пише "Not null", якщо модель ModelBuyTickets не є null.

1. BuyTickets(ModelBuyTickets modelBuyTickets) (Task<IActionResult>) [HttpPost, Authorize]:

* Асинхронний метод для обробки POST-запитів до сторінки покупки квитків.
* Використовує атрибут [Authorize], щоб дозволити доступ лише аутентифікованим користувачам.
* Отримує дані користувача та фільму з бази даних, створює новий квиток та зберігає його в базі даних.
* Після успішного збереження квитка перенаправляє користувача на сторінку історії покупок.
* Якщо фільм не знайдено, перенаправляє на сторінку помилки ErrorTickets.

1. ErrorTickets() (IActionResult):

* Метод для обробки запитів до сторінки помилки при покупці квитків.
* Повертає представлення ErrorTickets.

Методи класу AccountController:

1. Login() (HttpGet, AllowAnonymous):

* Метод для обробки GET-запитів до сторінки входу.
* Повертає представлення для входу.

1. Login(ModelLogin model) (HttpPost, AllowAnonymous, ValidateAntiForgeryToken):

* Метод для обробки POST-запитів до сторінки входу.
* Перевіряє валідність введених даних, аутентифікує користувача та генерує подію UserEventOccurred у разі успішної авторизації.
* Повертає представлення з моделлю в разі помилки.

1. Register() (HttpGet, AllowAnonymous):

* Метод для обробки GET-запитів до сторінки реєстрації.
* Повертає представлення для реєстрації.

1. Register(ModelRegister model) (HttpPost, AllowAnonymous, ValidateAntiForgeryToken):

* Метод для обробки POST-запитів до сторінки реєстрації.
* Перевіряє валідність введених даних, реєструє нового користувача та генерує подію UserEventOccurred у разі успішної реєстрації.
* Повертає представлення з моделлю в разі помилки.

1. Logout() (HttpPost, ValidateAntiForgeryToken):

* Метод для обробки POST-запитів до сторінки виходу.
* Виконує вихід користувача та перенаправляє на головну сторінку.

1. Profile() (Authorize):

* Метод для обробки запитів до сторінки профілю користувача.
* Отримує поточного користувача та повертає представлення з його даними.

1. Profile(ModelProfile model) (HttpPost, Authorize):

* Метод для обробки POST-запитів до сторінки профілю користувача.
* Оновлює дані користувача, зберігає їх та генерує подію UserEventOccurred у разі успішного оновлення.
* Повертає представлення з моделлю в разі помилки.

1. ChangePassword() (Authorize):

* Метод для обробки запитів до сторінки зміни пароля.
* Повертає представлення з моделлю для зміни пароля.

1. ChangePassword(ModelChangePassword model) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Метод для обробки POST-запитів до сторінки зміни пароля.
* Перевіряє валідність введених даних, змінює пароль користувача та генерує подію UserEventOccurred у разі успішної зміни пароля.
* Повертає представлення з моделлю в разі помилки.

1. History() (Authorize):

* Метод для обробки запитів до сторінки історії квитків користувача.
* Отримує поточного користувача та його квитки з бази даних, повертає представлення з цими даними.

Методи класу AdminController:

1. ManageAccounts() (Authorize):

* Відповідає за обробку запитів до сторінки управління обліковими записами.
* Отримує поточного користувача і, якщо він є адміністратором, повертає список інших користувачів для управління.

1. ManageFilms() (Authorize):

* Відповідає за обробку запитів до сторінки управління фільмами.
* Отримує поточного користувача і, якщо він є адміністратором, повертає список фільмів для управління.

1. ManageArticles() (Authorize):

* Відповідає за обробку запитів до сторінки управління статтями.
* Отримує поточного користувача і, якщо він є адміністратором, повертає список статей для управління.

1. AddAccount() (Authorize):

* Відповідає за обробку GET-запитів до сторінки додавання облікового запису.
* Повертає представлення для введення даних нового облікового запису.

1. AddAccount(ModelAccount account) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до сторінки додавання облікового запису.
* Додає нового користувача до системи та перенаправляє на сторінку управління обліковими записами у разі успіху.

1. AddFilm() (Authorize):

* Відповідає за обробку GET-запитів до сторінки додавання фільму.
* Повертає представлення для введення даних нового фільму.

1. AddFilm(Film film, IFormFile Image) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до сторінки додавання фільму.
* Додає новий фільм до системи разом із зображенням та перенаправляє на сторінку управління фільмами у разі успіху.

1. AddArticle() (Authorize):

* Відповідає за обробку GET-запитів до сторінки додавання статті.
* Повертає представлення для введення даних нової статті.

1. AddArticle(ModelHelp article) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до сторінки додавання статті.
* Додає нову статтю до системи та перенаправляє на сторінку управління статтями у разі успіху.

1. EditAccount(string Id) (HttpGet, Authorize):

* Відповідає за обробку GET-запитів до сторінки редагування облікового запису.
* Отримує дані облікового запису за заданим ідентифікатором та повертає представлення для редагування.

1. EditAccount(ModelAccount account) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до сторінки редагування облікового запису.
* Оновлює дані облікового запису та перенаправляє на сторінку управління обліковими записами у разі успіху.

1. EditFilm(int Id) (Authorize):

* Відповідає за обробку GET-запитів до сторінки редагування фільму.
* Отримує дані фільму за заданим ідентифікатором та повертає представлення для редагування.

1. EditFilm(Film currfilm) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до сторінки редагування фільму.
* Оновлює дані фільму та перенаправляє на сторінку управління фільмами у разі успіху.

1. EditArticle(int id) (Authorize):

* Відповідає за обробку GET-запитів до сторінки редагування статті.
* Отримує дані статті за заданим ідентифікатором та повертає представлення для редагування.

1. EditArticle(ModelHelp article) (HttpPost, Authorize):

* Відповідає за обробку POST-запитів до сторінки редагування статті.
* Оновлює дані статті та перенаправляє на сторінку управління статтями у разі успіху.

1. DeleteAccount(string id) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Відповідає за обробку POST-запитів до сторінки видалення облікового запису.
* Видаляє обліковий запис користувача та перенаправляє на сторінку управління обліковими записами у разі успіху.

1. DeleteFilm(int id) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Відповідає за обробку POST-запитів до сторінки видалення фільму.
* Видаляє фільм та пов'язані сесії, а також видаляє зображення фільму з серверу.

1. DeleteArticle(int id) (HttpPost, Authorize, ValidateAntiForgeryToken):

* Відповідає за обробку POST-запитів до сторінки видалення статті.
* Видаляє статтю з бази даних та перенаправляє на сторінку управління статтями у разі успіху.

## Програмна реалізація класів інтерфейсу користувача

У [Додатку Б](#_ДОДАТОК_Б._Лістинг) наведено повну реалізацію програмного коду розроблених класів інтерфейсу користувача.

# ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Функціональне тестування для ролі користувача «Адміністратор»

Для проведення функціонального тестування для ролі «Адміністратор» були використані розроблені у п. 1.4.1. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:** А.1.1.

**Вхідні дані:**

1. Ввести дату сеансу: «20.01.2023».

2. Ввести час сеансу: «10:00».

3. Обрати зал у кінотеатрі: «2».

4. Обрати фільм: «Аватар: шлях води».

5. Натиснути кнопку «Додати».

**Очікуваний результат:** Успішне додавання сеансу

**Отриманий результат:**

Рисунок 5.1 – Скриншот виконання тесту А.1.1

**Ідентифікатор тесту:**

**Вхідні дані:**

**Очікуваний результат:**

**Отриманий результат:**

Рисунок 5.2 – Скриншот виконання тесту \_\_\_\_\_

## Функціональне тестування для ролі користувача «Зареєстрований користувач»

Для проведення функціонального тестування для ролі «Зареєстрований користувач» були використані розроблені у п. 1.4.2. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:**

**Вхідні дані:**

**Очікуваний результат:**

**Отриманий результат:**

Рисунок 5.3 – Скриншот виконання тесту \_\_\_\_\_

## Функціональне тестування для ролі користувача «Гість»

Для проведення функціонального тестування для ролі «Гість» були використані розроблені у п. 1.4.3. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:**

**Вхідні дані:**

**Очікуваний результат:**

**Отриманий результат:**

Рисунок 5.4 – Скриншот виконання тесту \_\_\_\_\_

# ВИСНОВКИ

У результаті виконання курсової роботи було розроблено програмне забезпечення з використанням об’єктно-орієнтованої парадигми для вказати предметну область.

В ході роботи було…. (коротко описати що саме було зроблено під час виконання курсової роботи).

…

….

**Примітка:**

Посилання на github-репозиторій: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Посилання на відеоролик: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Воробйов, Ю. А. Правила оформлення навчальних і науково-дослідних документів [Текст] : навч. посіб. / Ю. А. Воробйов, Ю. О. Сисоєв. – 4-те вид., випр. і доп. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2019. – 88 с.
2. Автоматичне оформлення джерел по ВАК України [Електронний ресурс] – Режим доступу до ресурсу: <http://vak.in.ua/do.php>.
3. Кінотеатр Multiplex [Електронний ресурс] – Режим доступу до ресурсу: <https://multiplex.ua>.
4. Планета кіно - мережа кінотеатрів [Електронний ресурс] – Режим доступу до ресурсу: <https://planetakino.ua>.

# ДОДАТОК А. Лістинг класів предметної області

Клас «Film»

namespace ProjectCinema.Models

{

public class Film

{

public int Id { get; set; }

public string? Image { get; set; }

public double Price { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public List<Sessions> sessions { get; set; } = new List<Sessions>();

}

}

Клас «ModelAccount»

using ProjectCinema.Enums;

namespace ProjectCinema.Models

{

public class ModelAccount

{

public string Id { get; set; } = "";

public string Name { get; set; }

public string UserName { get; set; }

public string Email { get; set; }

public string PhoneNumber { get; set; }

public Roles Role { get; set; }

public string Password { get; set; }

}

}

Клас «ModelBuyTickets»

namespace ProjectCinema.Models

{

public class ModelBuyTickets

{

public string Data { get; set; }

public Film Film { get; set; } = new Film();

}

}

Клас «ModelChangePassword»

using System.ComponentModel.DataAnnotations;

namespace ProjectCinema.Models

{

public class ModelChangePassword

{

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Ваш пароль")]

public string CurrentPassword { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[StringLength(100, ErrorMessage = "Довжина пароля менше 8 символів!", MinimumLength = 8)]

[DataType(DataType.Password)]

[Display(Name = "Новий пароль")]

public string NewPassword { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Підтвердіть новий пароль")]

[Compare("NewPassword", ErrorMessage = "Паролі не співпадають!")]

public string ConfirmPassword { get; set; }

}

}

Клас «ModelHelp»

namespace ProjectCinema.Models

{

public class ModelHelp

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

}

}

Клас «ModelLogin»

using System.ComponentModel.DataAnnotations;

namespace ProjectCinema.Models

{

public class ModelLogin

{

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Display(Name = "Логін")]

public string UserName { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password, ErrorMessage = "Неправильний формат пароля!")]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запам'ятати мене")]

public bool RememberMe { get; set; } = false;

}

}

Клас «ModelProfile»

namespace ProjectCinema.Models

{

public class ModelProfile

{

public string Name { get; set; }

public string PhoneNumber { get; set; }

}

}

Клас «ModelRegister»

using System.ComponentModel.DataAnnotations;

namespace ProjectCinema.Models

{

public class ModelRegister

{

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Display(Name = "Ім'я користувача")]

public string Name { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Display(Name = "Логін")]

public string UserName { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[EmailAddress]

[Display(Name = "Пошта")]

public string Email { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[Phone]

[Display(Name = "Мобільний телефон")]

public string PhoneNumber { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Required(ErrorMessage = "Поле обов'язкове для заповнення!")]

[DataType(DataType.Password)]

[Display(Name = "Підтвердіть пароль")]

[Compare("Password", ErrorMessage = "Паролі не збігаються.")]

public string ConfirmPassword { get; set; }

}

}

Клас «Sessions»

namespace ProjectCinema.Models

{

public class Sessions

{

public int Id { get; set; }

public string Date { get; set; }

public int FilmId { get; set; }

}

}

Клас «Ticket»

namespace ProjectCinema.Models

{

public class Ticket

{

public int Id { get; set; }

public string FilmName { get; set; }

public double Price { get; set; }

public string? SenderId { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public string Time { get; set; }

}

}

Клас «User»

using Microsoft.AspNetCore.Identity;

using ProjectCinema.Enums;

namespace ProjectCinema.Models

{

public class User : IdentityUser

{

public string Name { get; set; }

public Roles Role { get; set; } = 0;

}

}

# ДОДАТОК Б. Лістинг класів інтерфейсу користувача

Клас «HomeController»

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using ProjectCinema.Data;

using ProjectCinema.Models;

using System.Diagnostics;

namespace ProjectCinema.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly ApplicationDbContext \_context;

private readonly UserManager<User> \_userManager;

public HomeController(ILogger<HomeController> logger, UserManager<User> userManager, ApplicationDbContext context)

{

\_logger = logger;

\_userManager = userManager;

\_context = context;

}

public IActionResult Index()

{

List<Film> films = \_context.Films.ToList();

return View(films);

}

public async Task<IActionResult> Help()

{

List<ModelHelp> help = \_context.Articles.ToList();

return View(help);

}

public async Task<IActionResult> Movie(int id)

{

var movie = \_context.Films.FirstOrDefault(film => film.Id == id);

return View(movie);

}

public async Task<IActionResult> BuyTickets(int Id)

{

var film = \_context.Films.FirstOrDefault(film => film.Id == Id);

ModelBuyTickets modelBuyTickets = new ModelBuyTickets()

{

Film = film

};

modelBuyTickets.Film.sessions = \_context.Sessions.Where(s => s.FilmId == Id).ToList();

if (modelBuyTickets != null)

await Console.Out.WriteLineAsync("Not null");

return View(modelBuyTickets);

}

[Authorize]

[HttpPost]

public async Task<IActionResult> BuyTickets(ModelBuyTickets modelBuyTickets)

{

var user = await \_userManager.GetUserAsync(User);

var film = \_context.Films.FirstOrDefault(f => f.Id == modelBuyTickets.Film.Id);

if (film != null)

{

var ticket = new Ticket

{

FilmName = modelBuyTickets.Film.Name,

Price = modelBuyTickets.Film.Price,

Name = user.Name,

SenderId = user.Id,

Email = user.Email,

Time = modelBuyTickets.Data

};

\_context.Add(ticket);

await \_context.SaveChangesAsync();

return RedirectToAction("History", "Account");

}

return RedirectToAction("ErrorTickets");

}

public IActionResult ErrorTickets()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

Клас «AccountController»

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using ProjectCinema.Data;

using ProjectCinema.Interfaces;

using ProjectCinema.Models;

namespace ProjectCinema.Controllers

{

public class AccountController : Controller, IAccount

{

private readonly UserManager<User> \_userManager;

private readonly SignInManager<User> \_signInManager;

private readonly ApplicationDbContext \_context;

public delegate Task UserEventHandler(string userId, string message);

public event UserEventHandler UserEventOccurred;

public AccountController(UserManager<User> userManager, SignInManager<User> signInManager, ApplicationDbContext context)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_context = context;

UserEventOccurred += async (userId, message) =>

{

TempData["UserNotification"] = message;

};

}

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> Login()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(ModelLogin model)

{

if (ModelState.IsValid)

{

var user = await \_userManager.FindByNameAsync(model.UserName);

if (user != null && await \_userManager.CheckPasswordAsync(user, model.Password))

{

await \_signInManager.SignInAsync(user, model.RememberMe);

await UserEventOccurred?.Invoke(user.Id, "Успішна авторизація!");

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError(string.Empty, "Помилка авторизації!");

return View(model);

}

return View(model);

}

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> Register()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(ModelRegister model)

{

if (ModelState.IsValid)

{

var user = new User { Name = model.Name, UserName = model.UserName, Email = model.Email, PhoneNumber = model.PhoneNumber };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, isPersistent: false);

await UserEventOccurred?.Invoke(user.Id, "Користувач успішно зареєстрований!");

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError(string.Empty, "Помилка реєстрації!");

return View(model);

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> Profile()

{

var user = await \_userManager.GetUserAsync(User);

var model = new ModelProfile

{

Name = user.Name,

PhoneNumber = user.PhoneNumber

};

return View(model);

}

[Authorize]

[HttpPost]

public async Task<IActionResult> Profile(ModelProfile model)

{

var user = await \_userManager.GetUserAsync(User);

if (ModelState.IsValid)

{

user.Name = model.Name;

user.PhoneNumber = model.PhoneNumber;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

await UserEventOccurred?.Invoke(user.Id, "Дані успішно зміненні!");

return View(model);

}

}

return View(model);

}

[Authorize]

public async Task<IActionResult> ChangePassword()

{

ModelChangePassword model = new ModelChangePassword();

return View(model);

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ChangePassword(ModelChangePassword model)

{

if (ModelState.IsValid)

{

var user = await \_userManager.GetUserAsync(User);

if (user == null)

{

return RedirectToAction(nameof(Login));

}

var changePasswordResult = await \_userManager.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);

if (!changePasswordResult.Succeeded)

{

foreach (var error in changePasswordResult.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

return View(model);

}

await \_signInManager.RefreshSignInAsync(user);

await UserEventOccurred?.Invoke(user.Id, "Успішна зміна пароля!");

return View(model);

}

return View(model);

}

public async Task<IActionResult> History()

{

var user = await \_userManager.GetUserAsync(User);

List<Ticket> tickets = \_context.Ticket.Where(t => t.SenderId == user.Id).ToList();

return View(tickets);

}

}

}

Клас «AdminController»

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ProjectCinema.Data;

using ProjectCinema.Enums;

using ProjectCinema.Interfaces;

using ProjectCinema.Models;

using System;

using System.Net.Sockets;

namespace ProjectCinema.Controllers

{

public class AdminController : Controller, IAdmin

{

private readonly UserManager<User> \_userManager;

private readonly ApplicationDbContext \_context;

private readonly IWebHostEnvironment \_webHostEnvironment;

public AdminController(UserManager<User> userManager, ApplicationDbContext context, IWebHostEnvironment webHostEnvironment)

{

\_userManager = userManager;

\_context = context;

\_webHostEnvironment = webHostEnvironment;

}

[Authorize]

public async Task<IActionResult> ManageAccounts()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var users = \_userManager.Users.ToList();

List<User> list = users.Where(u => u.Id != user.Id).ToList();

return View(list);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> ManageFilms()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

List<Film> list = \_context.Films.ToList();

return View(list);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> ManageArticles()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

List<ModelHelp> list = \_context.Articles.ToList();

return View(list);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> AddAccount()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

ModelAccount account = new ModelAccount();

return View(account);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddAccount(ModelAccount account)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

if (ModelState.IsValid)

{

var adduser = new User

{

Name = account.Name,

UserName = account.UserName,

Email = account.Email,

PhoneNumber = account.PhoneNumber,

Role = account.Role

};

var result = await \_userManager.CreateAsync(adduser, account.Password);

if (result.Succeeded)

{

return RedirectToAction("ManageAccounts");

}

}

return View(account);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> AddFilm()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

Film film = new Film();

return View(film);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddFilm(Film film, IFormFile Image)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

if (Image != null && Image.Length > 0)

{

var uploadsFolder = Path.Combine(\_webHostEnvironment.WebRootPath, "images");

var uniqueFileName = Guid.NewGuid().ToString() + "\_" + Image.FileName;

var filePath = Path.Combine(uploadsFolder, uniqueFileName);

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

await Image.CopyToAsync(fileStream);

}

film.Image = "/images/" + uniqueFileName;

if (ModelState.IsValid)

{

\_context.Films.Add(film);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageFilms");

}

}

return View(film);

}

return RedirectToAction("Index", "Home");

}

public async Task<IActionResult> AddArticle()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

ModelHelp article = new ModelHelp();

return View(article);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddArticle(ModelHelp article)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

if (ModelState.IsValid)

{

\_context.Articles.Add(article);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageArticles");

}

return View(article);

}

return RedirectToAction("Index", "Home");

}

[HttpGet]

[Authorize]

public async Task<IActionResult> EditAccount(string Id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var selecteduser = await \_userManager.FindByIdAsync(Id);

if (selecteduser != null)

{

ModelAccount account = new ModelAccount()

{

Id = selecteduser.Id,

Name = selecteduser.Name,

UserName = selecteduser.UserName,

Email = selecteduser.Email,

PhoneNumber = selecteduser.PhoneNumber,

Role = selecteduser.Role,

Password = selecteduser.PasswordHash

};

return View(account);

}

return RedirectToAction("ManageAccounts");

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> EditAccount(ModelAccount account)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

if (ModelState.IsValid)

{

var editedUser = await \_userManager.FindByIdAsync(account.Id);

editedUser.Name = account.Name;

editedUser.UserName = account.UserName;

editedUser.Email = account.Email;

editedUser.PhoneNumber = account.PhoneNumber;

editedUser.Role = account.Role;

var result = await \_userManager.UpdateAsync(editedUser);

if (result.Succeeded)

{

return RedirectToAction("ManageAccounts");

}

}

return View(account);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> EditFilm(int Id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var film = await \_context.Films.FindAsync(Id);

if (film != null)

return View(film);

return RedirectToAction("ManageFilms");

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> EditFilm(Film currfilm)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

if (ModelState.IsValid)

{

var film = \_context.Films.FirstOrDefault(f => f.Id == currfilm.Id);

if (film != null)

{

film.Name = currfilm.Name;

film.Description = currfilm.Description;

film.Price = currfilm.Price;

\_context.Films.Update(film);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageFilms");

}

}

return View(currfilm);

}

return RedirectToAction("Index", "Home");

}

public async Task<IActionResult> EditArticle(int id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var article = await \_context.Articles.FindAsync(id);

if (article != null)

return View(article);

return RedirectToAction("ManageArticles");

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> EditArticle(ModelHelp article)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

if (ModelState.IsValid)

{

\_context.Articles.Update(article);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageArticles");

}

return View(article);

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteAccount(string id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var account = await \_userManager.FindByIdAsync(id);

if (account != null)

{

var result = await \_userManager.DeleteAsync(account);

if (result.Succeeded)

{

return RedirectToAction("ManageAccounts");

}

}

else

{

return RedirectToAction("ManageAccounts");

}

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteFilm(int id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var film = \_context.Films.FirstOrDefault(f => f.Id == id);

if (film != null)

{

var sessions = \_context.Sessions.Where(s => s.FilmId == id).ToList();

\_context.Sessions.RemoveRange(sessions);

var currentDirectory = Directory.GetCurrentDirectory();

var imagePath = Path.Combine(currentDirectory, "wwwroot", film.Image.TrimStart('/'));

if (System.IO.File.Exists(imagePath))

{

try

{

System.IO.File.Delete(imagePath);

Console.WriteLine("Файл успішно видвалений: " + imagePath);

}

catch (Exception ex)

{

Console.WriteLine("Помилка при видаленні файла: " + ex.Message);

}

}

\_context.Remove(film);

await \_context.SaveChangesAsync();

}

return RedirectToAction("ManageFilms");

}

return RedirectToAction("Index", "Home");

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteArticle(int id)

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Role == Roles.Admin)

{

var article = \_context.Articles.FirstOrDefault(a => a.Id == id);

if (article != null)

{

\_context.Articles.Remove(article);

await \_context.SaveChangesAsync();

return RedirectToAction("ManageArticles");

}

else

{

return RedirectToAction("ManageArticles");

}

}

return RedirectToAction("Index", "Home");

}

}

}

# ДОДАТОК В. Лістинг класів unit-тестів

Тест клас «TestUser»:

using ProjectCinema.Models;

using ProjectCinema.Enums;

namespace TestProject

{

public class TestUser

{

[Fact]

public void User()

{

User user = new()

{

Name = "User",

Role = Roles.User,

Id = "lqjsdafh27fh2ehfq9wf7",

PhoneNumber = "1234567890",

Email = "user@gmail.com",

UserName = "user",

};

Assert.Equal("User", user.Name);

Assert.Equal(Roles.User, user.Role);

Assert.Equal("lqjsdafh27fh2ehfq9wf7", user.Id);

Assert.Equal("1234567890", user.PhoneNumber);

Assert.Equal("user@gmail.com", user.Email);

Assert.Equal("user", user.UserName);

}

}

}

Тест клас «TestFilm»:

using ProjectCinema.Models;

namespace TestProject

{

public class TestFilm

{

[Fact]

public void Film()

{

Sessions session = new()

{

Id = 1,

Date = "2024-05-26T20:40",

FilmId = 1

};

List<Sessions> sessions = [session];

Film film = new()

{

Id = 1,

Image = "/images/photo1.png",

Price = 80,

Name = "Deadpool 3",

Description = "Deadpool is back!",

sessions = sessions

};

Assert.Equal(1, film.Id);

Assert.Equal("/images/photo1.png", film.Image);

Assert.Equal(80, film.Price);

Assert.Equal("Deadpool 3", film.Name);

Assert.Equal("Deadpool is back!", film.Description);

Assert.Equal(sessions, film.sessions);

}

}

}

Тест клас «TestModelHelp»:

using ProjectCinema.Models;

namespace TestProject

{

public class TestModelHelp

{

[Fact]

public void ModelHelp()

{

ModelHelp article = new()

{

Id = 1,

Name = "Test",

Description = "description",

};

Assert.Equal(1, article.Id);

Assert.Equal("Test", article.Name);

Assert.Equal("description", article.Description);

}

}

}

Тест клас «TestTicket»:

using ProjectCinema.Models;

namespace TestProject

{

public class TestTicket

{

[Fact]

public void Ticket()

{

Ticket ticket = new()

{

Id = 1,

FilmName = "Deadpool 3",

Price = 80,

SenderId = "lqjsdafh27fh2ehfq9wf7",

Name = "User",

Email = "user@gmail.com",

Time = "2024-05-26T20:40"

};

Assert.Equal(1, ticket.Id);

Assert.Equal("Deadpool 3", ticket.FilmName);

Assert.Equal(80, ticket.Price);

Assert.Equal("lqjsdafh27fh2ehfq9wf7", ticket.SenderId);

Assert.Equal("User", ticket.Name);

Assert.Equal("user@gmail.com", ticket.Email);

Assert.Equal("2024-05-26T20:40", ticket.Time);

}

}

}