**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

|  |  |
| --- | --- |
|  |  |

**Курсова робота**

**з дисципліни «Програмування»**

**на тему: «Веб-сайт замовлення квитків у кінотеатр»**

Виконав:

студент 1 курсу, групи ІА-31

Губар Богдан Олександрович \_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник:

асистент кафедри ІСТ

Тюляков Дмитро Ігорович \_\_\_\_\_\_\_\_\_\_

(підпис)

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

Студент \_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2024 року

ЗМІСТ

[ВСТУП 3](#_Toc158383261)

[1 ВИМОГИ ДО СИСТЕМИ 4](#_Toc158383262)

[1.1 Функціональні вимоги до системи 4](#_Toc158383263)

[1.2 Нефункціональні вимоги до системи 4](#_Toc158383264)

[2 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ 5](#_Toc158383265)

[2.1 Діаграма прецедентів 6](#_Toc158383266)

[2.2 Опис сценаріїв використання системи 7](#_Toc158383267)

[3 АРХІТЕКТУРА СИСТЕМИ 24](#_Toc158383268)

[4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ 26](#_Toc158383269)

[4.1 Загальна структура проекту 27](#_Toc158383270)

[4.2 Компоненти рівня доступу до даних 27](#_Toc158383271)

[4.3 Компоненти рівня бізнес-логіки 29](#_Toc158383272)

[4.4 Компоненти рівня інтерфейсу користувача 30](#_Toc158383273)

[ВИСНОВКИ 31](#_Toc158383274)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 33](#_Toc158383275)

[ДОДАТОК А Лістінг програми 36](#_Toc158383276)

[ДОДАТОК Б Назва додатку Б (за наявності) 37](#_Toc158383277)

[ДОДАТОК В Назва додатку В (за наявності) 38](#_Toc158383278)

# ВСТУП

## На сьогоднішній день зростає тенденція в онлайн-сервісах, таких як: інтернет-магазини, сайти аптек, онлайн-банкінги, бронювання готелів та подорожей в Інтернеті, тощо. Це пов’язано насамперед через їх зручність та відсутність необхідності фізичної наявності в магазині.

## Актуальність створення веб-сайту для замовлення квитків полягає в зростаючому попиті на онлайн-сервіси в різних галузях, зокрема у галузі культурних подій та розважальної індустрії. З підвищенням інтересу до кіно, театральних вистав, музичних концертів та інших подій, користувачі шукають зручний і надійний спосіб замовлення квитків без необхідності особистої присутності в касі або на місці події.

## Метою роботи є створення системи, яка дозволяє користувачам зручно та швидко замовляти квитки на різноманітні події через інтернет. Для цього система повинна мати такі можливості та функціональність:

- Зручний інтерфейс для перегляду різних подій, вистав та концертів.

- Можливість обрати місце та кількість квитків для замовлення.

- Нагадування та інформаційна підтримка щодо подій, на які були придбані квитки.

- Система повинна бути надійною та безпечною з точки зору обробки особистих даних користувачів та фінансових операцій.

## Такий веб-сайт забезпечить користувачам зручний спосіб отримання квитків на різні події, сприяючи розвитку онлайн-культурної індустрії та задоволенню потреб сучасного споживача.

# 1 ВИМОГИ ДО СИСТЕМИ

## 1.1 Функціональні вимоги до системи

Система має відповідати наступним функціональним вимогам:

* Незареєстрований користувач(далі - гість) має право переглядати інформацію про сеанси
* Гість має право замовлювати квитки, встановлявати ціну та кількість квитків для конкретного сеансу, а також місце та дату;
* Адміністратор має право редагувати інформацію про квитки та сеанси а також видаляти та додавати сеанси;
* Адміністратор може бачити деталі замовлення квитків користувачами;

## 1.2 Нефункціональні вимоги до системи

Система має відповідати наступним функціональним вимогам:

– система повинна мати архітектуру MVC;

– система повинна мати веб-інтерфейс;

– інтерфейс користувача має бути зручним та інтуїтивно-зрозумілим;

– система повинна бути крос-платформною.

# 2 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ

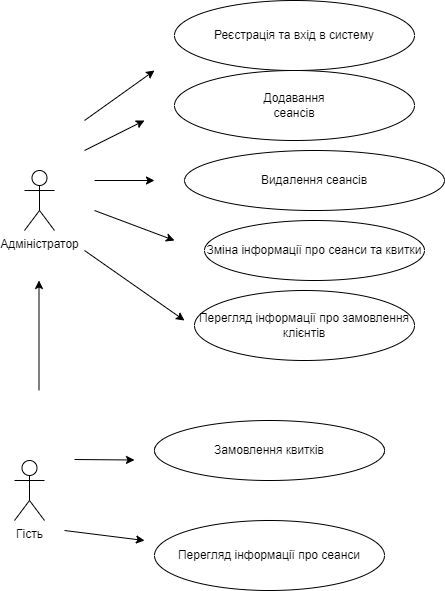
## 2.1 Діаграма прецедентів

Діаграма прецедентів системи представлена на рис. 2.1.

Акторами є користувачі системи: незареєстрований (гість) та зареєстрований (адміністратор).

Адміністратору доступна інформація про замовлення, можливість додавати, видаляти сеанси та редагувати їх, також він може переглядати замовлення клієнтів. Гість – незареєстрований користувач, може лише замовити квиток, та переглянути інформацію про сеанси, які є на сайті, на які можна замовити квиток. Детально усі сценарії використання описані у наступному підрозділі.

Рисунок 2.1 – Діаграма прецедентів



## 2.2 Опис сценаріїв використання системи

Детальні описи сценаріїв використання наведено у таблицях 2.1 – 2.17.

Таблиця 2.1 – Сценарій використання «Реєстрація та вхід»

|  |  |
| --- | --- |
| Назва | Реєстрація та вхід |
| ID | 1 |
| Опис | Адміністратор може зареєструватись в сервісі, і надалі керувати процесами на сайті, додавати та змінювати інформацію на сайті |
| Актори | Адміністратор |
| Вигоди компанії | Сайт буде мати постійні оновлення та актуальну інформацію про сеанси, завдяки роботі адміністраторів |
| Частота користування | Часто |
| Тригери | Користувач переходить на сторінку реєстрації або входу |
| Передумови | Правильні дані, активний обліковий запис(для входу) |
| Постумови | Невдалий або успішний вхід або реєстраціія |
| Основний розвиток | Користувач вводить запит у пошукову строку, натискає на кнопку пошуку чи Enter |
| Альтернативні розвитки | - |
| Виняткові ситуації | Заблокований обліковий запис |

В таблиці 2.2 представлений сценарій використання «Додавання сеансів»

Таблиця 2.2 – Сценарій використання «Додавання сеансів»

|  |  |
| --- | --- |
| Назва | Додавання сеансів |
| ID | 2 |
| Опис | Адміністратор додає новий сеанс на сторінку перегляду сеансів |
| Актори | Адміністратор |
| Вигоди компанії | - |
| Частота користування | Часто |
| Тригери | Адміністратор натискає кнопку «Додати новий сеанс» |
| Передумови | Введені необхідні дані(назва, опис сеансу) |
| Постумови | Додається новий сеанс |
| Основний розвиток | Адміністратор натиснув «Додати сеанс»  На сайті з’явився новий сеанс, на який можна замовити квитки |
| Альтернативні розвитки | – |
| Виняткові ситуації | Після натискання кнопки сеанс не додається або неправильно додається через неправильно вказані дані при додаванні |

В таблиці 2.3 представлений сценарій використання «Видалення сеансів»

Таблиця 2.3 – Сценарій використання «Видалення сеансів»

|  |  |
| --- | --- |
| Назва | Видалення сеансів |
| ID | 3 |
| Опис | Адміністратор видаляє обраний сеанс |
| Актори | Адміністратор |
| Вигоди компанії | Сайт не буде містити неактуальні сеанси, тому користувачі ресурсу завжди матимуть актуальну інформацію щодо нових фільмів |
| Частота користування | Часто |
| Тригери | Адміністратор натискає кнопку «Видалити сеанс» |
| Передумови | Було перейдено на сторінку перегляду інформації про сеанс |
| Постумови | Сеанс видаляється з сторінки |
| Основний розвиток | Сеанс зникає з головної сторінки |
| Альтернативні розвитки | – |
| Виняткові ситуації | – |

В таблиці 2.4 представлений сценарій використання «Зміна інформації про сеанси»

Таблиця 2.4 – Сценарій використання «Зміна інформації про сеанси»

|  |  |
| --- | --- |
| Назва | Зміна інформації про сеанси |
| ID | 4 |
| Опис | Адміністратор змінює інформацію про сеанс |
| Актори | Адміністратор |
| Вигоди компанії | Завдяки цьому користувачі сервісу в динаміці бачитимуть зміни і завжди розумітимуть точно інформацію про кожен сеанс на даний момент |
| Частота користування | Невідомо, як часто ця опція буде використовуватися на практиці |
| Тригери | Натиснута кнопка «Змінити інформацію» на сторінці перегляду сеансу |
| Передумови | Адміністратор перейшов на сторінку сеансу |
| Постумови | Змінюється відповідно інформація про конкретний сеанс |
| Основний розвиток | На сайті змінюється певна інформація щодо того сеансу, який редагував адміністратор |
| Альтернативні розвитки | – |
| Виняткові ситуації | – |

В таблиці 2.5 представлений сценарій використання «Перегляд інформації про замовлення клієнтів»

Таблиця 2.5 – Сценарій використання «Перегляд інформації про замовлення клієнтів»

|  |  |
| --- | --- |
| Назва | Перегляд інформації про замовлення клієнтів |
| ID | 5 |
| Опис | Адміністратор переглядає замовлення клієнтів |
| Актори | Адміністратор |
| Вигоди компанії | – |
| Частота користування | Напряму залежить від потоку клієнтів |
| Тригери | Адміністратор перейшов на вкладку «Останні замовлення» |
| Передумови | Було створене замовлення гостями |
| Постумови | Адміністратор бачить інформацію про останні замовлення на сайті та їх деталі |
| Основний розвиток | В вкладці перегляд замовлень видно останні замовлення |
| Альтернативні розвитки | – |
| Виняткові ситуації | – |

В таблиці 2.6 представлений сценарій використання «Замовлення квитків»

Таблиця 2.6 – Сценарій використання «Замовлення квитків»

|  |  |
| --- | --- |
| Назва | Замовлення квитків |
| ID | 6 |
| Опис | Гість (рідко – адміністратор) може замовити квитки на будь який актуальний сеанс |
| Актори | Гість, Адміністратор |
| Вигоди компанії | Завдяки замовленням, кошти йтимуть в бюджет компанії |
| Частота користування | Залежить від потоку клієнтів але зазвичай часто  Адміністратор – рідко |
| Тригери | Було натиснуто кнопку «Замовити квиток» |
| Передумови | Користувачі знаходяться на сторінці сеансу |
| Постумови | Було створено нове замовлення |
| Основний розвиток | Було створено нове замовлення на конкретний сеанс, місце і час |
| Альтернативні розвитки | Через відсутність вільних місць користувач може не потрапити на бажаний час місце і навіть сеанс |
| Виняткові ситуації | – |

В таблиці 2.7 представлений сценарій використання «Перегляд інформації про сеанси»

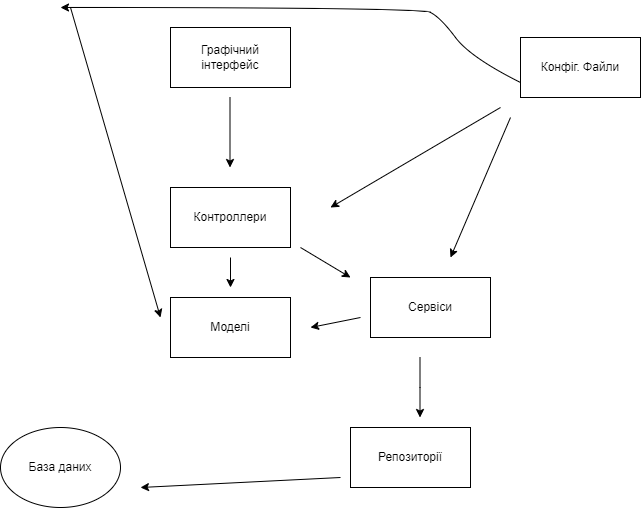
Таблиця 2.7 – Сценарій використання «Перегляд інформації про сеанси»

|  |  |
| --- | --- |
| Назва | Перегляд інформації про сеанси |
| ID | 7 |
| Опис | Користувачі можуть переглянути сторінку з сеансами та конкретно інформацію про кожен сеанс |
| Актори | Адміністратор, Гість |
| Вигоди компанії | На сторінці буде активність клієнтів, і це може призвести до покупок квитків |
| Частота користування | Дуже часто |
| Тригери | Було перейдено на сайт ресурсу або на сторінку перегляду інформації про конкретний сеанс |
| Передумови | Було перейдено на веб-ресурс |
| Постумови | Користувачі можуть переглянути потрібну їм інформацію про сеанси |
| Основний розвиток | Користувачі переходять на сторінку сайту, бачать перелік сеансів, або якщо переходять на сторінку конкретного сеансу, то бачать інформацію про сеанс |
| Альтернативні розвитки | – |
| Виняткові ситуації | Через відсутність будь яких сеансів або технічних збоїв користувачі не бачать ніяких сеансів та інформації |

# 3 АРХІТЕКТУРА СИСТЕМИ

Загальна архітектура системи наведена на рис. 3.1.

Рисунок 3.1 – Загальна архітектура системи



Система складається з наступних елементів:

– графічний інтерфейс;

– серверна частина;

– база даних.

Графічний інтерфейс необхідний для взаємодії з користувачем. HTTP запит надходить до серверної частини, де оброблюється і повертається відповідь. На серверній частині виконується основна логіка системи. Дані, отриманні з графічного інтерфейсу валідуються, конвертуються. Також, серверна частина формує запит до бази даних та оброблює відповідь і передає її до графічного інтерфейсу. База даних зберігає дані, які були сформовані на серверній частині та повертає їх у разі запиту. Структурна схема зображена в Додатку Б

До серверної частини належать наступні елементи:

– контролер;

– модель та вигляд;

– сервіс;

– репозиторій;

– конфігураційні файли для авторизації та безпеки.

**Графічний інтерфейс:**

Користувач взаємодіє з графічним інтерфейсом, вводячи дані або взаємодіючи з елементами інтерфейсу.

**Контролери:**

Контролери отримують дані від графічного інтерфейсу та визначають, які дії потрібно виконати на основі цих даних.

Також вони створюють об'єкти та дані для відображення на графічному інтерфейсі.

**Сервіси:**

Контролери передають отримані дані в сервіси для виконання бізнес-логіки та обробки даних.

Сервіси виконують різні операції з даними в залежності від вимог.

**Моделі:**

Моделі в свою чергу використовуються для збереження та обробки інформації у сервісах та репозиторіях.

**Репозиторії:**

Репозиторії взаємодіють з моделями для збереження та отримання даних з бази даних. Є своєрідним мостом між БД та системою

Сервіси використовують репозиторії для роботи з даними, включаючи створення, оновлення, видалення та отримання даних.

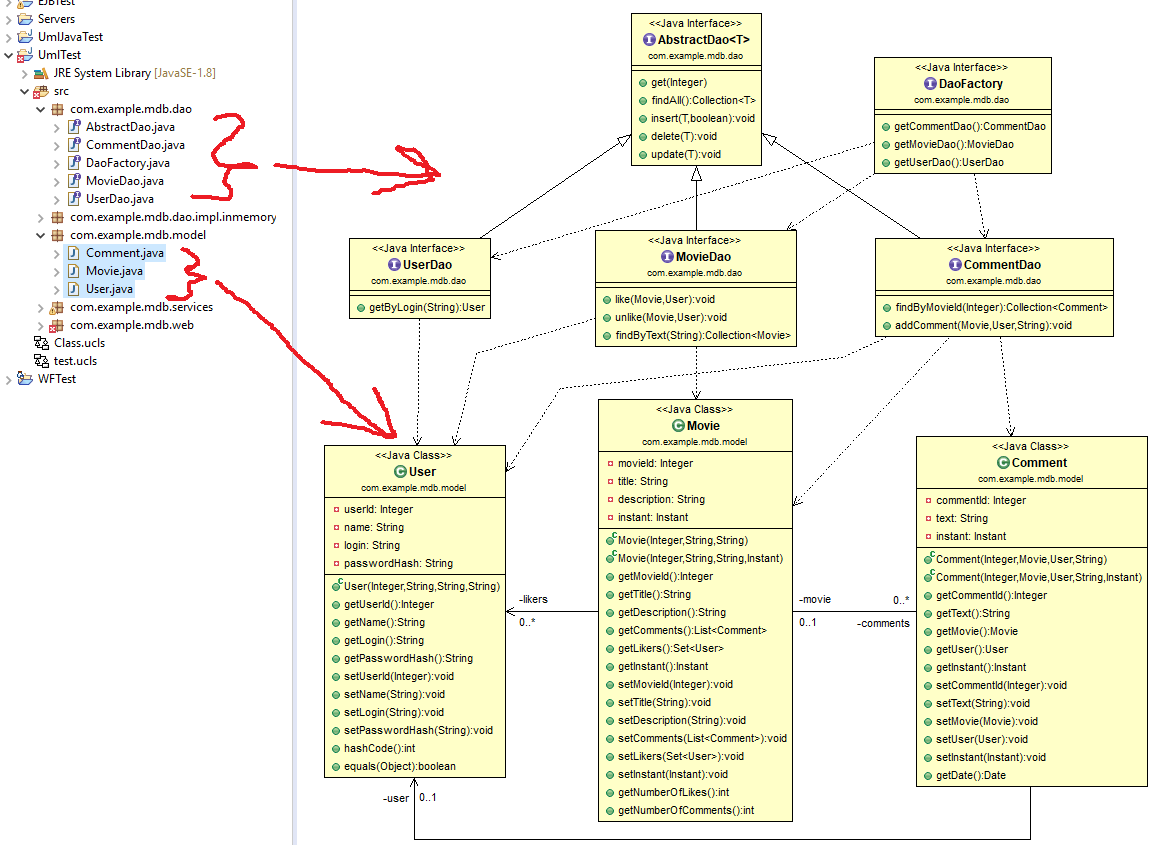
**Конфігураційні файли:**

Конфігураційні файли визначають налаштування та параметри сервісу, наприклад, права для відвідування певних сторінок, можливість редагування тощо.

# 4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

Обсяг: 10-20 сторінок, включаючи фрагменти коду, рисунки з UML-діаграмами класів або іншими типами діаграм.

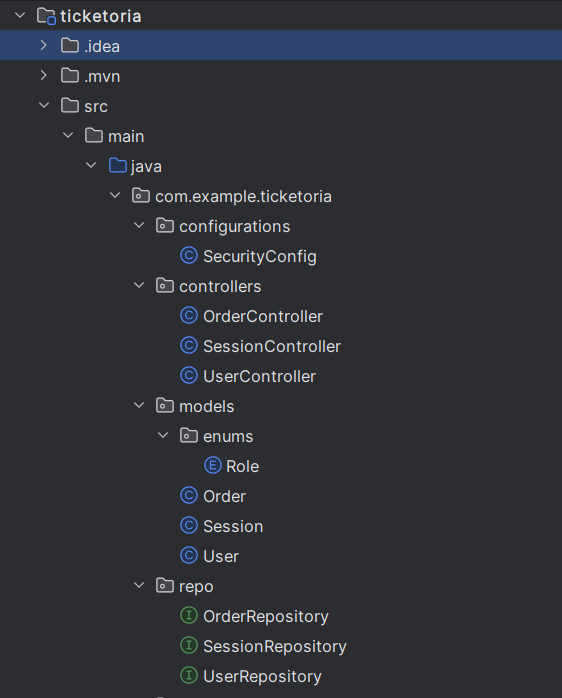
Для економії часу рекомендується не малювати діаграми класів «руками», а генерувати їх на основі написаного коду. Наприклад, це вміє робити компонент plug-in компонент ObjectAid для Eclipse (<https://www.objectaid.com/>). Достатньо просто перетягти необхідні елементи з дерева проекту на полотно діаграми, і усі елементи з’являться автоматично:

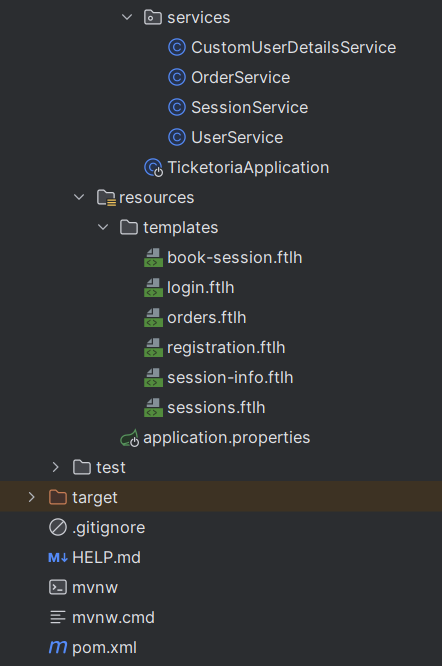


Рекомендується розбити цей розділ на підрозділи відповідно до архітектури системи

## 4.1 Загальна структура проекту

Загальна структура проекту представлена на рисунках 4.1.1, 4.1.2





Рисунки 4.1.1, 4.1.2 – Загальна структура проекту

Проект складається з веб-ресурсів, бази даних, та вихідного коду, який в свою чергу можна поділити на компоненти рівня доступу до даних, компоненти бізнес-логіки та веб-компоненти.

## 4.2 Компоненти рівня доступу до даних

Основні сутності та інтерфейси рівня доступу до даних наведені на рис. 4.2

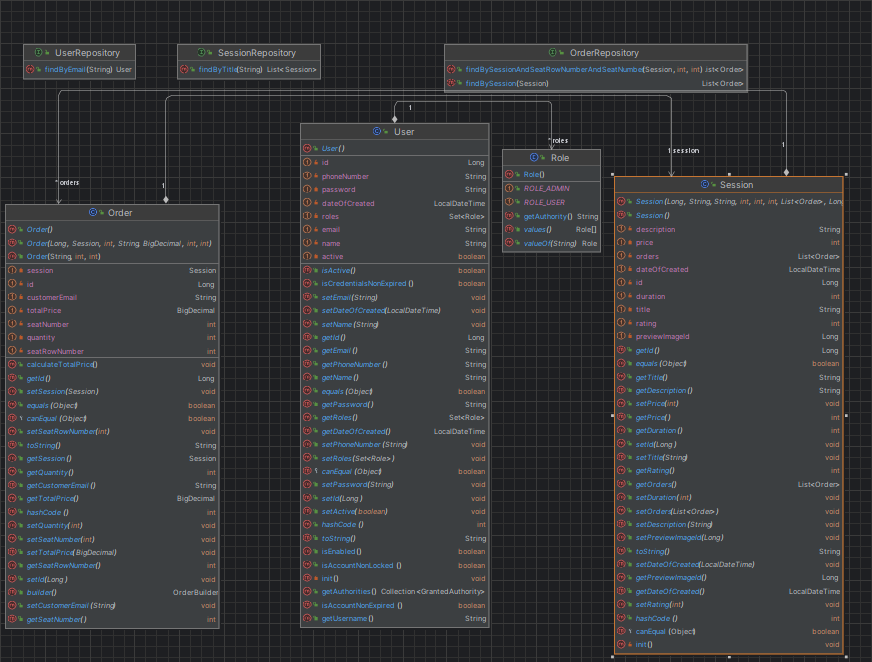


Рисунок 4.2 – Основні сутності та інтерфейси рівня доступу до даних

Репозиторії – JPA Repository

UserRepository – Знаходить юзера за його Email, інтерфейс.

SessionRepository - Знаходить колекцію об’єктів типу Sessions за заголовком, інтерфейс.

OrderRepository – Знаходить замовлення за номером ряду і місця а також з’єднується з таблицею sessions для прив’язки до конкретного типу фільму

Моделі - Models

Session – представляє собою Entity Table для збереження інформації в базі даних (MySQL)

Містить поля: опис(description), ціна(price) , дата створення(dateOfCreated), айді(ключ)- (id), тривалість(duration), заголовок(title) та оцінка(rating).

Методи: - гетери та сетери для відповідних полів та їх використання в бізнес логіці.

Order – Модель для збереження замовлень

Містить поля: Айді(ключ)- (id), totalPrice – ціна, ряд і місце(SeatNumber та SeatRowNumber), емейл замовника (customerEmail) та quantity – кількість 1 штука, бо не можна замовити більше 1 квитка на 1 і те саме місце.

Методи – гетери та сетери а також метод calculateTotalPrice для виведення інформації про ціну квитка.

User – Модель юзера для збереження інформації про користувачів.

Містить поля – id(ключ) , phoneNumber – номер телефону, e-mail, role – роль, та ім’я (name).

Методи – гетери та сетери для використання під час авторизації і бізнес логіки.

Ролі – Роль ADMIN – адміністратор, USER – звичайний користувач (гостьовий режим).

## 4.3 Компоненти рівня бізнес-логіки

## Компоненти рівня бізнес логіки представлені на рисунку 4.3

Рис. 4.3 Компоненти рівня бізнес логіки

Services – сервіси реалізують бізнес логіку.

SessionService – Сервіс для сеансів та фільмів, містить зв'язок з репозиторіями сеансу та замовлень а також логування.

Методи:

* saveSession – зберігає новий сеанс фільму;
* delete – видаляє сеанс фільму
* getSessionById – метод для доступу до сеансу за ключем;
* listSessions – відображає всі сеанси які є на сайті.

OrderService – сервіс для обробки бізнес логіки пов’язаної з замовленням квитків. Має логування, зв'язок з сервісом сеансів(попередній сервіс), юзера(наступний сервіс) а також з репозиторієм OrderRepository.

Методи:

* getUserOrders – отримує інформацію про наявні замовлення які є в юзера;
* saveOrder – зберігає нове замовлення в базі даних;
* getOrderById – метод для отримання замовлення по ключу( id);

UserService – сервіс для управління юзерами та сценаріями, пов’язаними з доступом і авторизацією(виходом).

Містить кодувальник паролів, лог-вікна а також зв'язок з репозиторієм.

Містить методи:

* createUser – Створення нового юзера(з роллю адміна при реєстрації);
* logout – вихід з облікового запису(адміністратора);
* getUserById – доступ до юзера за ключем;

CustomUserDetailsService

Використовує UserRepository, щоб здійснювати доступ до даних користувачів з бази даних.

Реалізація CustomUserDetailsService:

Клас CustomUserDetailsService реалізує інтерфейс UserDetailsService, який визначає метод loadUserByUsername. Цей метод викликається Spring Security під час аутентифікації користувача.

У методі loadUserByUsername йде пошук користувача за його email в базі даних і повертається об'єкт, який реалізує інтерфейс UserDetails. Цей об'єкт містить інформацію про користувача, необхідну для проведення аутентифікації і авторизації.

Цей код є частиною бізнес-логіки, яка відповідає за ідентифікацію та авторизацію користувачів у вашому додатку. Його основна функція - забезпечення здійснення безпечного доступу до додатку шляхом перевірки облікових записів користувачів.

4.4 Компоненти рівня інтерфейсу користувача

На рисунку 4.4 показано компоненти рівня інтерфейсу користувача

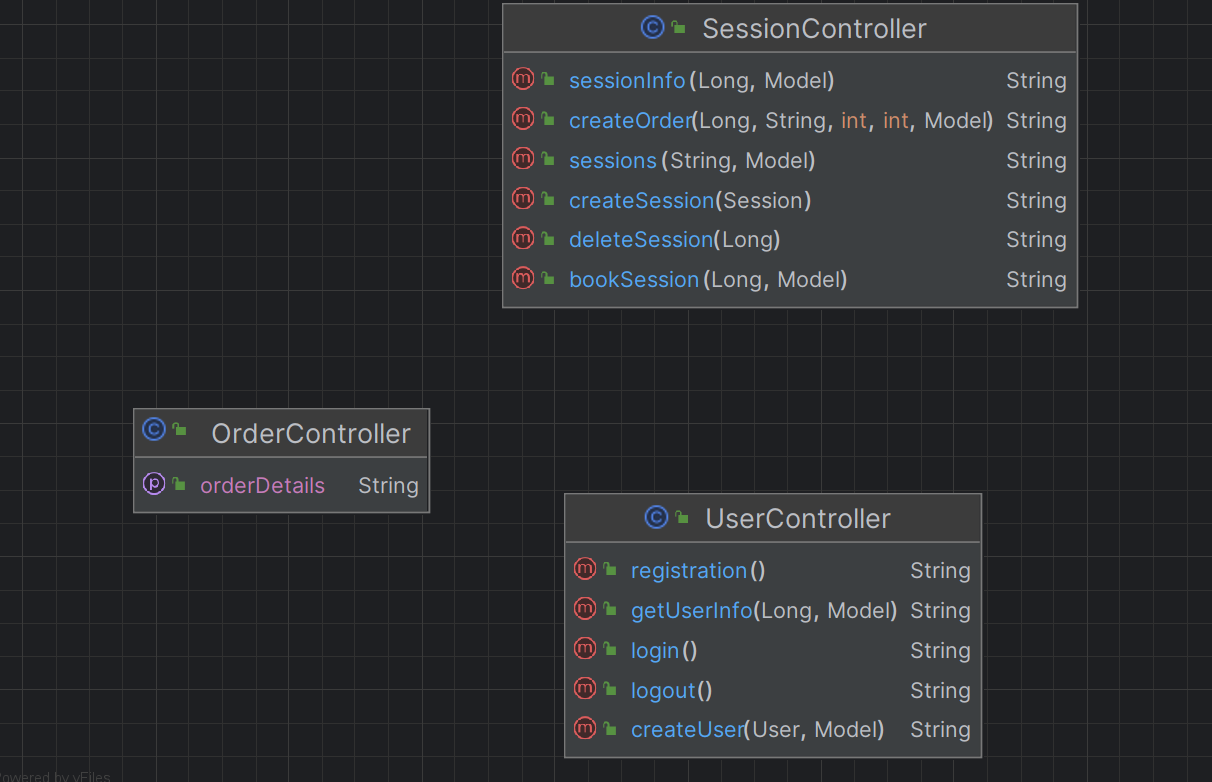


Рис. 4.4 Компоненти рівня інтерфейсу користувача

Controllers – контроллери призначені для обробки http – запитів та взаємодією з користувачем.

SessionController – Відповідає за переходи на сторінки замовлень, перегляду, видалення сеанасів. Також відповідає за завантаження головної сторінки (sessions).

SessionInfo – GET – запит – відповідає за перехід на сторінку з відображенням інформації про сеанс. Працює лише за наявних сеансів на сайті.

createSession, deleteSession – POST запити, відповідають за додавання і видалення сеансів на сайті відповідно.

bookSession – GET запит , призначений для переходу на сторінку замовлення квитка, пов'язаний з моделлю Order.

createOrder – POST запит, призначений для створення замовлення після натискання кнопки створення замовлення. Створює новий екземпляр класу Order.

Sessions – GET запит, відображає головну сторінку (мапінг: «/»);

OrderController – контроллер для відображення інформації про змаовлення.

Мість методи

getOrderDetails – GET запит – призначений для відображення сторінки замовлень(orders).

UserController – контроллер, який керує запитами, пов’язаними з авторизацією та користувачами.

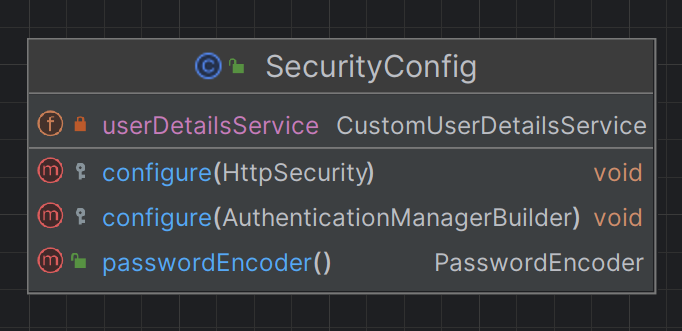
Registration – 2 запити: GET – для переходу на сторінку реєстрації зі сторінки логіну;

POST(createUser) - – для створення нового користувача після реєстрації.

Login, logout – GET запити , призначені для входу в акаунт або виходу з нього(авторизація).

Spring Security – призначений для управління ролями, та доступом до певних розділів чи сторінок сайту для різних користувачів.

SecurityConfig – містить всю логіку пов’язаною з використанням безпеки.



userDetailsService – зв'язок з сервісом юзера,

configure – методи для управління доступом

passwordEncoder – Bean для шифрування паролю.

# ВИСНОВКИ

Під час написання роботи було проаналізовано актуальність онлайн сервісів замовлення квитків. Було вирішено розробити систему, яка дозволить користувачу замовляти квитки на фільми онлайн.

Першим кроком було сформулювати функціональні та нефункціональні вимоги до системи, що визначило очікувану поведінку системи. Наступним, обрано технології на яких буде написана система. Java була обрана в якості мови програмування, середа розробки – IntelliJ IDEA за її можливості та потужність. В якості системи управління базами даних було вирішено використовувати MySQL за легкість її налаштування та підтримки. Для роботи з запитами та налаштуванням з’єднання між системою та базою даних було вирішено використати Hibernate та Spring Data. Величезна перевага фреймворку Spring Data, за яку його було обрано, в можливості не писати запити на SQL, а будувати їх за допомоги назв методів. Для налаштування безпеки системи було вирішено використовувати Spring Security. Цей фреймворк дозволяє власноруч зробити необхідні налаштування, такі як дозволенні ресурси, url-адреси та як саме система визначає ці права доступу. Система повинна мати графічний інтерфейс, тож вирішено було використовувати HTML та Thymeleaf за їх легкість у впровадженні.

Наступним кроком були описані сценарії використання. Система підтримує функціонал замовлення квитків а також управління сторінками з інформацією про квитки досвідченими адміністраторами які можуть додавати та видаляти сеанси на різні фільми та робити інформацію на сайті актуальною .

Далі, було розроблено загальну архітектуру системи. Систему було розділено на логічні рівні. Серед них – контроллери, сервіси, моделі, репозиторії, ресурси та конфігураціні файли.

Далі, була розроблена ER-модель. Завдяки Hibernate, для встановлення відповідності між базою даних та Java-кодом було створено класи-сутності, що прискорило розробку системи. Для кожної сутності були визначенні необхідні поля та зв’язок з іншими сутностями, якщо в ньому була необхідність.

Отже, сумуючи наведене вище, можна сказати, що основі переваги створеної системи в простоті її налаштування та оновлення функціоналу. Завдяки тому, що система має відкриту архітектуру, її розширення не є проблемою. Також графічний інтерфейс системи є легким в налаштуванні і простий для розуміння юзерами. Система є конкурентоспроможною та має потенціал до розвинення.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SpringFramework. Spring Security - [Електронний ресурс] <https://spring.io/projects/spring-security>
2. A Simple E-Commerce Implementation with Spring - [Електронний ресурс] <https://www.baeldung.com/spring-angular-ecommerce>
3. Bootstrap – Режим доступу до ресурсу - [Електронний ресурс] <https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.bundle.min.js>
4. Draw.io - [Електронний ресурс] <https://app.diagreams.net/>
5. Html Documentation - [Електронний ресурс] - <https://developer.mozilla.org/en-US/docs/Web/HTML>

# ДОДАТОК А Лістінг програми

В першому рядочку – посилання на github

Далі – код системи (інтерфейси, класи, шаблони сторінок, конфігураційні файли, файли з тестовими даними і т.д.)

Враховуючи загальний обсяг коду системи, в додатках допускається використовувати шрифт, що відрізняється від стандартного Times New Roman 14 міжрядковий 1.5 (наприклад, розмір – 10, міжрядковий – одинарний).

Проект на Github:

Код Системи:

**Контроллери**

SessionController

package com.example.ticketoria.controllers;  
  
import com.example.ticketoria.models.Order;  
import com.example.ticketoria.models.Session;  
import com.example.ticketoria.services.OrderService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.ui.Model;  
import com.example.ticketoria.services.SessionService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
  
import java.io.IOException;  
  
@Controller  
@RequiredArgsConstructor  
public class SessionController {  
 private final SessionService sessionService;  
 @Autowired  
 private final OrderService orderService;  
  
 @GetMapping("/")  
 public String sessions(@RequestParam(name = "title", required = false) String title, Model model) {  
 model.addAttribute("sessions", sessionService.listSessions(title));  
 return "sessions" ;  
 }  
  
 @GetMapping("/session/{id}")  
 public String sessionInfo(@PathVariable Long id, Model model){  
 Session session = sessionService.getSessionById(id);  
 model.addAttribute("session", session);  
 return "session-info";  
  
 }  
  
  
  
 @PostMapping("/session/create")  
 public String createSession( Session session) throws IOException {  
 sessionService.saveSession(session);  
 return "redirect:/";  
 }  
  
 @PostMapping("/session/delete/{id}")  
 public String deleteSession(@PathVariable Long id){  
 sessionService.delete(id);  
 return "redirect:/";  
  
 }  
  
 @GetMapping("/session/{id}/book")  
 public String bookSession(@PathVariable Long id, Model model) {  
 Session session = sessionService.getSessionById(id);  
 Order order = new Order();  
 order.setSession(session);  
 order.setQuantity(1);  
 order.calculateTotalPrice();  
 model.addAttribute("order", order);  
 model.addAttribute("session", session);  
 return "book-session";  
 }  
  
  
  
 @PostMapping("/session/{id}/order")  
 public String createOrder(@PathVariable Long id,  
 @RequestParam("customerEmail") String customerEmail,  
 @RequestParam("seatRowNumber") int seatRowNumber,  
 @RequestParam("seatNumber") int seatNumber,  
 Model model) {  
 Order order = new Order(customerEmail, seatRowNumber, seatNumber);  
 model.addAttribute("order", order);  
  
  
 Order savedOrder = orderService.saveOrder(order);  
  
 return "redirect:/orders/" + savedOrder.getId();  
 }  
  
}

**OrderController**

package com.example.ticketoria.controllers;  
  
  
  
  
import lombok.RequiredArgsConstructor;  
  
import org.springframework.stereotype.Controller;  
  
import org.springframework.web.bind.annotation.\*;  
  
  
  
@Controller  
@RequiredArgsConstructor  
public class OrderController {  
  
  
 @GetMapping("/orders")  
 public String getOrderDetails() {  
  
 return "orders";  
 }  
  
}

**UserController**

package com.example.ticketoria.controllers;  
  
import com.example.ticketoria.models.User;  
import com.example.ticketoria.services.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
  
import org.springframework.web.bind.annotation.PostMapping;  
  
  
@Controller  
@RequiredArgsConstructor  
  
public class UserController {  
 private final UserService userService;  
  
 @GetMapping("/login")  
 public String login() {  
 return "login";  
 }  
  
 @GetMapping("/registration")  
 public String registration() {  
 return "registration";  
  
 }  
  
 @PostMapping("/registration")  
 public String createUser(User user, Model model) {  
 if (!userService.createUser(user)) {  
 model.addAttribute("errorMessage", "Користувач з e-mail: " + user.getEmail() + " вже існує");  
 return "registration";  
 }  
 userService.createUser(user);  
 return "redirect:/login";  
 }  
  
 @GetMapping("/logout/")  
 public String logout() {  
 userService.logout();  
 return "redirect:/";  
 }  
  
  
  
}

**Моделі(Models)**

**Session**

package com.example.ticketoria.models;  
  
import javax.persistence.\*;  
import lombok.\*;  
  
import java.time.LocalDateTime;  
import java.util.ArrayList;  
import java.util.List;  
  
@Entity  
@Table(name = "sessions")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Session {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 private Long id;  
  
 @Column(name = "title")  
 private String title;  
  
 @Column(name = "description", columnDefinition = "text")  
 private String description;  
  
 @Column(name = "price")  
 private int price;  
  
 @Column(name = "duration")  
 private int duration;  
  
 @Column(name = "rating")  
 private int rating;  
  
 @OneToMany(mappedBy = "session", cascade = CascadeType.*ALL*, orphanRemoval = true)  
 private List<Order> orders = new ArrayList<>();  
  
  
  
 private Long previewImageId;  
 private LocalDateTime dateOfCreated;  
  
 @PrePersist  
 private void init() {  
 dateOfCreated = LocalDateTime.*now*();  
 }  
  
  
  
 protected boolean canEqual(final Object other) {  
 return other instanceof Session;  
 }

**Order**

package com.example.ticketoria.models;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
import java.math.BigDecimal;  
import javax.validation.constraints.Max;  
import javax.validation.constraints.Min;  
@Entity  
@Table(name = "orders")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Order {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @ManyToOne  
 @JoinColumn(name = "session\_id", nullable = false)  
 private Session session;  
  
 @Column(name = "quantity")  
 private int quantity = 1;  
  
 @Column(name = "customer\_email")  
 private String customerEmail;  
  
 @Column(name = "total\_price")  
 private BigDecimal totalPrice;  
  
 @Column(name = "seat\_row\_number")  
 @Min(value = 1, message = "Row number must be at least 1")  
 @Max(value = 20, message = "Row number cannot exceed 20")  
 private int seatRowNumber;  
  
 @Column(name = "seat\_number")  
 @Min(value = 1, message = "Seat number must be at least 1")  
 @Max(value = 20, message = "Seat number cannot exceed 20")  
 private int seatNumber;  
  
 public Order(String customerEmail, int seatRowNumber, int seatNumber) {  
 this.customerEmail = customerEmail;  
 this.seatNumber = seatNumber;  
 this.seatRowNumber = seatRowNumber;  
 }  
  
 @PrePersist  
 public void calculateTotalPrice() {  
 this.totalPrice = BigDecimal.*valueOf*(this.session.getPrice()).multiply(BigDecimal.*valueOf*(this.quantity));  
 }

**User**

package com.example.ticketoria.models;  
  
import javax.persistence.\*;  
  
import com.example.ticketoria.models.enums.Role;  
import lombok.Data;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.time.LocalDateTime;  
import java.util.Collection;  
import java.util.HashSet;  
import java.util.Set;  
  
@Entity  
@Table(name = "users")  
@Data  
  
public class User implements UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private Long id;  
  
 @Column(name = "email", unique = true)  
 private String email;  
  
 @Column(name = "phone\_number")  
 private String phoneNumber;  
  
 @Column(name = "name")  
 private String name;  
  
 @Column(name = "active")  
 private boolean active;  
  
  
 @Column(name = "password", length = 1000)  
 private String password;  
  
  
 @ElementCollection(targetClass = Role.class, fetch = FetchType.*EAGER*)  
 @CollectionTable(name = "user\_role",  
 joinColumns = @JoinColumn(name = "user\_id"))  
 @Enumerated(EnumType.*STRING*)  
 private Set<Role> roles = new HashSet<>();  
  
 private LocalDateTime dateOfCreated;  
  
 @PrePersist  
 private void init(){  
 dateOfCreated = LocalDateTime.*now*();  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return roles;  
 }  
  
 @Override  
 public String getUsername() {  
 return email;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return active;  
 }  
}

**Enums 🡪 Role**

package com.example.ticketoria.models.enums;  
  
import org.springframework.security.core.GrantedAuthority;  
  
public enum Role implements GrantedAuthority {  
 *ROLE\_ADMIN*, *ROLE\_USER*;  
  
 @Override  
 public String getAuthority() {  
 return name();  
 }  
}

**Services**

**SessionService**

package com.example.ticketoria.services;  
  
import com.example.ticketoria.models.Order;  
import com.example.ticketoria.models.Session;  
import com.example.ticketoria.repo.SessionRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
  
  
import javax.persistence.EntityNotFoundException;  
import java.io.IOException;  
import java.util.List;  
import com.example.ticketoria.repo.OrderRepository;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class SessionService {  
  
 private final SessionRepository sessionRepository;  
 private final OrderRepository orderRepository;  
  
 public List<Session> listSessions(String title) {  
 if (title != null) return sessionRepository.findByTitle(title);  
 return sessionRepository.findAll();  
 }  
  
 public void saveSession(Session session) throws IOException {  
  
  
 *log*.info("Saving new Session. Title: {}; ", session.getTitle());  
 Session sessionFromDb = sessionRepository.save(session);  
 }  
  
 public void delete(Long id) {  
 Session session = sessionRepository.findById(id)  
 .orElseThrow(() -> new EntityNotFoundException("Session not found"));  
  
 List<Order> orders = orderRepository.findBySession(session);  
 for (Order order : orders) {  
 orderRepository.delete(order);  
 }  
  
 // Тепер видаляємо сесію  
 sessionRepository.deleteById(id);  
 }  
  
 public Session getSessionById(Long id) {  
 return sessionRepository.findById(id).orElse(null);  
 }  
}

**OrderService**

package com.example.ticketoria.services;  
import com.example.ticketoria.models.User;  
import lombok.AllArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
import com.example.ticketoria.models.Order;  
import com.example.ticketoria.repo.OrderRepository;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
  
import javax.persistence.EntityNotFoundException;  
  
  
@Service  
@AllArgsConstructor  
@Slf4j  
public class OrderService {  
 private final OrderRepository orderRepository;  
 private final SessionService sessionService;  
 private final UserService userService;  
  
 public Order saveOrder(Order order) {  
 return orderRepository.save(order);  
 }  
  
 public Order getOrderById(Long id) {  
 return orderRepository.findById(id).orElseThrow(() -> new EntityNotFoundException("Order not found"));  
 }  
  
  
 @GetMapping("/user/{userId}/orders")  
 public String getUserOrders(@PathVariable Long userId, Model model) {  
 User user = userService.getUserById(userId);  
 if (user != null) {  
 model.addAttribute("user", user);  
 }  
 return "orders";  
 }  
}

**UserService**

package com.example.ticketoria.services;  
  
import com.example.ticketoria.models.User;  
import com.example.ticketoria.models.enums.Role;  
import com.example.ticketoria.repo.UserRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class UserService {  
 private final UserRepository userRepository;  
 private final PasswordEncoder passwordEncoder;  
  
 public boolean createUser(User user){  
 String email = user.getEmail();  
 if (userRepository.findByEmail(email) != null) return false;  
 user.setActive(true);  
 user.setPassword(passwordEncoder.encode(user.getPassword()));  
 user.getRoles().add(Role.*ROLE\_ADMIN*);  
 *log*.info("Saving new user with email: {}", email);  
 userRepository.save(user);  
 return true;  
 }  
  
 public void logout() {  
 SecurityContextHolder.*clearContext*();  
 }  
  
  
  
 public User getUserById(Long id) {  
 return userRepository.findById(id).orElse(null);  
 }  
  
  
}

**CustomUserDetailsService**

package com.example.ticketoria.services;  
  
  
import com.example.ticketoria.repo.UserRepository;  
import lombok.RequiredArgsConstructor;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
@Service  
@RequiredArgsConstructor  
public class CustomUserDetailsService implements UserDetailsService {  
 private final UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {  
 return userRepository.findByEmail(email);  
 }  
}

**Repositories**

**SessionRepository**

package com.example.ticketoria.repo;  
  
import com.example.ticketoria.models.Session;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface SessionRepository extends JpaRepository<Session, Long> {  
 List<Session> findByTitle(String title);  
}

**UserRepository**

package com.example.ticketoria.repo;  
  
import com.example.ticketoria.models.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
 User findByEmail(String email);  
}

**OrderRepository**

package com.example.ticketoria.repo;  
  
import com.example.ticketoria.models.Session;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import com.example.ticketoria.models.Order;  
  
import java.util.List;  
  
@Repository  
public interface OrderRepository extends JpaRepository<Order, Long> {  
  
 List<Order> findBySession(Session session);  
 List<Order> findBySessionAndSeatRowNumberAndSeatNumber(Session session, int seatRowNumber, int seatNumber);  
}

**Configurations**

**SecurityConfig**

package com.example.ticketoria.configurations;  
  
import com.example.ticketoria.services.CustomUserDetailsService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.context.annotation.Bean;  
  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
  
  
  
@EnableWebSecurity  
@RequiredArgsConstructor  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
 private final CustomUserDetailsService userDetailsService;  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable()  
 .authorizeRequests()  
 .antMatchers("/session/create", "/session/delete").authenticated()  
 .antMatchers("/", "/login", "/images/\*\*", "/registration", "/session/{id}", "/session/{id}/book", "/session/{id}/order", "/order/{id}", "/logout")  
 .permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .loginPage("/login")  
 .and()  
 .logout()  
 .permitAll()  
 .logoutSuccessUrl("/");  
 }  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService)  
 .passwordEncoder(passwordEncoder());  
 }  
 @Bean  
 public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder(8);  
 }  
}

# ДОДАТОК Б

**Ftlh-сторінки**

**Sessions.ftlh**

<!doctype html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">  
 <link rel="stylesheet" href="/static/css/style.css">  
 <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>  
 <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-s15y6DxywsA9hU4d53pH+Ng7qzT7wC09h6Xn+tbVx+VDDVu2k5ZITu1tGrKJXK6A" crossorigin="anonymous"></script>  
 <title>Ticketoria</title>  
 <style>  
 body {  
 background-color: #f0f0f0;  
 color: #333333;  
 font-family: 'Roboto', sans-serif;  
 }  
  
 span{  
 color: white;  
 }  
 h1 {  
 margin-top: 20px;  
 text-align: center;  
 font-weight: bold;  
 color: #ff5722;  
 }  
 .container {  
 margin-top: 30px;  
 }  
 .search-bar, .new-session-form {  
 background-color: #ffffff;  
 padding: 20px;  
 border-radius: 10px;  
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
 }  
 .search-bar input, .new-session-form input, .new-session-form select, .new-session-form textarea {  
 margin-bottom: 15px;  
 background-color: #ffffff;  
 border: 1px solid #ff5722;  
 color: #333333;  
 }  
 .search-bar input:focus, .new-session-form input:focus, .new-session-form select:focus, .new-session-form textarea:focus {  
 background-color: #ffffff;  
 color: #ff5722;  
 border-color: #ff5722;  
 }  
 .session-item {  
 margin-bottom: 15px;  
 transition: background-color 0.3s;  
 }  
 .session-item p {  
 font-size: 18px;  
 }  
 .session-item a {  
 color: #ff5722;  
 text-decoration: none;  
 transition: color 0.3s, transform 0.3s;  
 }  
 .session-item a:hover {  
 color: #ffccbc;  
 transform: scale(1.1);  
 }  
 .btn-custom {  
 background-color: #ff5722;  
 border: none;  
 color: #ffffff;  
 transition: background-color 0.3s, color 0.3s;  
 }  
 .btn-custom:hover {  
 background-color: #ffccbc;  
 color: #333333;  
 }  
  
  
  
 .session-list {  
 display: flex;  
 flex-wrap: wrap;  
 gap: 10px;  
 max-height: 400px;  
 overflow-y: auto;  
 }  
  
 .session-item {  
 background-color: #ffffff;  
 border-radius: 10px;  
 padding: 10px;  
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
 flex-grow: 1;  
 }  
  
 .session-item a.btn-primary {  
 background-color: #f44336;  
 border-color: #f44336;  
 }  
  
 .session-item a.btn-primary:hover {  
 background-color: #e57373;  
 border-color: #e57373;  
 }  
  
 .btn-danger {  
 background-color: #f44336;  
 border: none;  
 color: #ffffff;  
 transition: background-color 0.3s, color 0.3s;  
 }  
 .btn-danger:hover {  
 background-color: #e57373;  
 color: #ffffff;  
 }  
 </style>  
</head>  
<body>  
<div class="container">  
 <h1>TICKETORIA | <a href="/login" class="text-muted" style="font-style: italic">Вхід</a></h1> <hr>  
 <div class="text-right mb-4">  
 <form action="/logout" method="get">  
 <button type="submit" class="btn btn-danger">Logout</button>  
 </form>  
  
  
 </div>  
 <h4 class="h4">Квитки на будь який фільм</h4>  
  
 <div class="search-bar mb-4">  
 <form action="/" method="get">  
 <div class="form-group">  
 <label for="title">Пошук по назві сеансу:</label>  
 <input type="text" id="title" name="title" class="form-control">  
 </div>  
 <button type="submit" class="btn btn-custom">Шукати</button>  
 </form>  
 </div>  
 <div class="session-list">  
 **<#list** sessions as session**>** <div class="session-item rounded p-3 mb-3" style="width: 25%; box-sizing: border-box;">  
 <p><b>**${**session.title**}**</b></p>  
 <p>**${**session.price**}** грн. **${**session.rating**}**</p>  
 <a href="/session/**${**session.id**}**" class="btn btn-primary"> <span>Детальніше</span></a>  
 </div>  
 **<#else>** <h3>Сеансів нема</h3>  
 **</#list>** </div>  
 <hr>  
 <h3>Створити новий сеанс</h3>  
 <div class="new-session-form">  
 <form action="/session/create" method="post">  
 <div class="form-group">  
 <label for="title">Назва фільму:</label>  
 <input type="text" id="title" name="title" class="form-control" required>  
 </div>  
 <div class="form-group">  
 <label for="description">Опис:</label>  
 <input type="text" id="description" name="description" class="form-control" required>  
 </div>  
 <div class="form-group">  
 <label for="price">Ціна квитка на сеанс:</label>  
 <input type="number" id="price" name="price" class="form-control" required>  
 </div>  
 <div class="form-group">  
 <label for="duration">Тривалість(хв):</label>  
 <input type="number" id="duration" name="duration" class="form-control" required>  
 </div>  
 <div class="form-group">  
 <label for="rating">Оцінка фільму:</label>  
 <input type="number" id="rating" name="rating" class="form-control" required>  
 </div>  
 <button type="submit" class="btn btn-custom">Додати новий сеанс</button>  
 </form>  
 </div>  
  
 </div>  
</body>  
</html>

**Session-info.ftlh**

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport"  
 content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <link rel="stylesheet" type="text/css" href="/static/css/style.css">  
 <title>Ticketoria</title>  
 <style>  
 body {  
 background-color: #f0f0f0;  
 color: #333333;  
 font-family: 'Roboto', sans-serif;  
 }  
 h1.session\_h1 {  
 text-align: center;  
 font-weight: bold;  
 color: #ff5722;  
 margin-top: 20px;  
 }  
 h4 {  
 color: #333333;  
 margin-top: 20px;  
 }  
 hr {  
 border: 0;  
 height: 1px;  
 background-color: #cccccc;  
 margin: 20px 0;  
 }  
 b {  
 color: #ff5722;  
 }  
 .btn-primary {  
 background-color: #ff5722;  
 border: none;  
 color: #ffffff;  
 padding: 10px 20px;  
 text-align: center;  
 text-decoration: none;  
 display: inline-block;  
 font-size: 16px;  
 margin: 10px 2px;  
 transition: background-color 0.3s, color 0.3s;  
 cursor: pointer;  
 border-radius: 5px;  
 }  
 .btn-primary:hover {  
 background-color: #ffccbc;  
 color: #333333;  
 }  
 input[type="submit"] {  
 background-color: #ff5722;  
 border: none;  
 color: #ffffff;  
 padding: 10px 20px;  
 text-align: center;  
 text-decoration: none;  
 display: inline-block;  
 font-size: 16px;  
 margin: 10px 2px;  
 transition: background-color 0.3s, color 0.3s;  
 cursor: pointer;  
 border-radius: 5px;  
 }  
 input[type="submit"]:hover {  
 background-color: #ffccbc;  
 color: #333333;  
 }  
 </style>  
</head>  
<body>  
<h1 class="session\_h1">TICKETORIA</h1>  
<hr>  
<h4>Детальна інформація про сеанс</h4>  
  
<p><b>Назва сеансу: </b> **${**session.title**}**</p>  
<p><b>Опис сеансу: </b> **${**session.description**}**</p>  
<p><b>Ціна: </b> **${**session.price**}** грн.</p>  
<p><b>Тривалість сеансу (у хв): </b> **${**session.duration**}**</p>  
<p><b>Оцінка від наших експертів: </b> **${**session.rating**}**</p>  
<hr>  
<form action="/session/delete/**${**session.id**}**" method="post">  
 <input type="submit" value="Видалити сеанс"/>  
</form>  
<br>  
<form action="/session/**${**session.id**}**/book" method="get">  
 <input type="submit" class="btn btn-primary" value="Замовити квиток"/>  
</form>  
</body>  
</html>

**Login.ftlh**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Ticketoria</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #F0F0F0;  
 color: #333;  
 margin: 0;  
 padding: 0;  
 }  
  
 h1 {  
 text-align: center;  
 color: #007BFF;  
 }  
  
 h4 {  
 color: #FF6B6B;  
 }  
  
 form {  
 max-width: 400px;  
 margin: 20px auto;  
 padding: 20px;  
 background-color: #FFF;  
 border-radius: 8px;  
 box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="email"],  
 input[type="password"] {  
 width: 100%;  
 padding: 8px;  
 margin: 8px 0;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 margin-top: 10px;  
 background-color: #007BFF;  
 color: #FFF;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #0056B3;  
 }  
  
 a {  
 display: block;  
 text-align: center;  
 color: #007BFF;  
 text-decoration: none;  
 margin-top: 20px;  
 }  
  
 a:hover {  
 color: #0056B3;  
 }  
 </style>  
</head>  
<body>  
<h1>TICKETORIA</h1>  
<hr>  
<h4>Авторизація</h4>  
<form action="/login" method="post">  
 Ел.Пошта: <input type="email" name="username"><br><br>  
 Пароль: <input type="password" name="password"><br><br>  
 <input type="submit" value="Вхід"/>  
</form>  
<a href="/registration"> Зареєструватися</a>  
<a href="/"> Залишитись в гостьовому режимі</a>  
</body>  
</html>

**Registration.ftlh**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Ticketoria</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #F0F0F0;  
 color: #333;  
 margin: 0;  
 padding: 0;  
 }  
  
 h1 {  
 text-align: center;  
 color: #007BFF;  
 }  
  
 h4 {  
 color: #FF6B6B;  
 }  
  
 form {  
 max-width: 400px;  
 margin: 20px auto;  
 padding: 20px;  
 background-color: #FFF;  
 border-radius: 8px;  
 box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="email"],  
 input[type="password"] {  
 width: 100%;  
 padding: 8px;  
 margin: 8px 0;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 margin-top: 10px;  
 background-color: #007BFF;  
 color: #FFF;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #0056B3;  
 }  
  
 .error-message {  
 color: red;  
 text-align: center;  
 margin-top: 10px;  
 }  
 </style>  
</head>  
<body>  
<h1>TICKETORIA</h1>  
<hr>  
<h4>Реєстрація</h4>  
<form action="/registration" method="post">  
 Ім'я користувача: <input type="text" name="name"><br><br>  
 Ел.Пошта: <input type="email" name="email"><br><br>  
 Номер телефону: <input type="text" name="phoneNumber"><br><br>  
 Пароль: <input type="password" name="password"><br><br>  
 <input type="submit" value="Зареєструватись"/>  
</form>  
**<#if** errorMessage??**>** <div class="error-message">**${**errorMessage**}**</div>  
**</#if>**</body>  
</html>

**Book-session.ftlh**

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Book Session</title>  
 <style>  
 body {  
 background-color: #f0f0f0;  
 color: #333333;  
 font-family: 'Roboto', sans-serif;  
 padding: 20px;  
 }  
 h1 {  
 text-align: center;  
 margin-bottom: 20px;  
 color: #ff5722;  
 }  
 form {  
 max-width: 400px;  
 margin: 0 auto;  
 padding: 20px;  
 background-color: #ffffff;  
 border-radius: 10px;  
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
 }  
 label {  
 display: block;  
 margin-bottom: 10px;  
 }  
 input[type="email"],  
 select {  
 width: 100%;  
 padding: 8px;  
 margin-bottom: 15px;  
 background-color: #ffffff;  
 border: 1px solid #ff5722;  
 color: #333333;  
 box-sizing: border-box;  
 }  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #ff5722;  
 border: none;  
 color: #ffffff;  
 border-radius: 4px;  
 cursor: pointer;  
 transition: 0.5s;  
 }  
 input[type="submit"]:hover {  
 background-color: #ffccbc;  
 color: #333333;  
 }  
 img {  
 display: block;  
 margin: 20px auto;  
 max-width: 100%;  
 height: auto;  
 }  
 </style>  
</head>  
<body>  
<h1>Замовити квиток на фільм: **${**session.title**}**</h1>  
<form action="/session/**${**session.id**}**/order" method="post">  
 <input type="hidden" th:field="\*{id}" />  
 <input type="hidden" th:field="\*{id}" />  
 <label for="quantity">Кількість квитків: 1</label>  
 <br>  
 <label for="customerEmail">Email:</label>  
 <input type="email" id="customerEmail" name="customerEmail" />  
 <br>  
 <label for="row">Виберіть ряд:</label>  
 <select id="row" name="seatRowNumber" th:field="\*{seatRowNumber}">  
 <script th:inline="javascript">  
 for (let i = 1; i <= 20; i++) {  
 *document*.write("<option value='" + i + "'>" + i + "</option>");  
 }  
 </script>  
 </select>  
  
 <label for="seat">Виберіть місце:</label>  
 <select id="seat" name="seatNumber" th:field="\*{seatNumber}">  
 <script th:inline="javascript">  
  
 for (let i = 1; i <= 20; i++) {  
 *document*.write("<option value='" + i + "'>" + i + "</option>");  
 }  
 </script>  
 </select>  
 <label for="totalPrice">Price: **${**session.price**}** UAH</label>  
 <br>  
 <input type="submit" class="btn btn-primary" value="Create Order" />  
  
  
</form>  
</body>  
</html>

**Orders.ftlh**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Order Details</title>  
 <style>  
 body {  
 background-color: #f0f0f0;  
 color: #333333;  
 font-family: 'Roboto', sans-serif;  
 padding: 20px;  
 }  
 h1 {  
 text-align: center;  
 margin-bottom: 20px;  
 color: #ff5722;  
 }  
 p {  
 margin-bottom: 10px;  
 }  
 h3 {  
 text-align: center;  
 margin-top: 20px;  
 color: #ff5722;  
 }  
 </style>  
</head>  
<body>  
<h1>Order Details</h1>  
**<#list** orders as order**>** <div style="background-color: #ffffff; padding: 20px; margin-bottom: 20px; border-radius: 10px; box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);">  
 <p>Order ID: **${**order.id**}**</p>  
 <p>Customer Email: **${**order.customerEmail**}**</p>  
 <p>Seat Row Number: **${**order.seatRowNumber**}**</p>  
 <p>Seat Number: **${**order.seatNumber**}**</p>  
 </div>  
**<#else>** <h3>Нема замовлень</h3>  
**</#list>**</body>  
</html>