МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Аліна САВЧЕНКО

«_____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: «Програмний додаток для обліку комерційної діяльності кінотеатру»

Виконав: Резь Владислав Юрійович

Керівник: професор кафедри КІТ Віноградов Микола Анатолійович

Нормоконтролер Ігор РАЙЧЕВ

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Аліна САВЧЕНКО

"___"_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Резь Владислава Юрійовича

(прізвище, ім'я, по батькові)

1.Тема роботи: «Програмний додаток для обліку комерційної діяльності кінотеатру», затверджена наказом ректора від "29"вересня 2023р. за № 1976/ст

2. Термін виконання роботи: : з 02 жовтня 2023р. по 31 грудня 2023р.

3. Вихідні дані до роботи: Розробка програмного продукту за допомогою мови програмування Java.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): Аналіз діяльності кінотеатрів включаючи аналіз комерційної діяльності та дослідження існуючих аналогів систем обліку комерційної діяльності; Формування вимог до застосунку, зокрема, функціональних та нефункціональних вимог**;** опис програмної реалізації проєкту; представлення прототипу застосунку.

5. Перелік обов'язкового графічного матеріалу: інформативні рисунки, графічні скріншоти роботи системи, слайди презентації в MS PowerPoint.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Пошук і дослідження наукових джерел.	02.10.2023 – 08.10.2023	
2.	Розроблення та затвердження календарного плану виконання дипломної роботи.	09.10.2023 – 10.10.2023	
3.	Проведення консультацій з науковим керівником.	11.10.2023 – 13.10.2023	
4.	Написання Розділу 1. Аналіз діяльності кінотеатрів.	14.10.2023 –27.10.2023	
5.	Написання Розділу 2. Формування вимог до застосунку.	28.10.2023 – 16.11.2023	
6.	Написання Розділу 3. Опис програмної реалізації системи.	17.11.2023 – 19.11.2023	
7.	Написання Розділу 4. Прототип застосунку	19.11.2023 – 30.11.2023	
8.	Оформлення пояснювальної записки дипломної роботи.	01.12.2023 – 04.12.2023	
9.	Написання, друк та підписання Рецензії у рецензента та Відгуку керівника у встановленому порядку.	05.12.2022 – 10.12.2023	
10.	Створення Презентації та доповіді.	11.12.2023 – 12.12.2023	
11.	Підготовка до захисту та попередній захист дипломної роботи на випусковій кафедрі.	13.11.2023 – 17.12.2023	

7. Дата видачі завдання: «02» жовтня 2023 р.

Керівник дипломної роботи _____ Віноградов Микола Анатолійович
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Резь Владислав Юрійович
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний додаток для обліку комерційної діяльності кінотеатру» викладена на 68 сторінках і містить 18 рисунків та два додатка. Список бібліографічних посилань складається з 7 найменувань.

Ключові слова: КОМЕРЦІЙНА ДІЯЛЬНІСТЬ, SOLID, КЛІЄНТ-СЕРВЕР, МОВА JAVA, ПРОГРАМНІ ЗАСОБИ, ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ЗАСТОСУТКУ, НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО ЗАСТОСУТКУ, ШАБЛОН MODEL-VIEW-CONTROLLER (MVC).

Об'єкт дослідження – застосунок для обліку комерційної діяльності кінотеатру.

Предмет дослідження – комерційна діяльність кінотеатру.

Мета роботи. Дослідження комерційної діяльності кінотеатру, характеристика основних програмних засобів для створення застосунку. На основі проведеного аналізу створення RESTful застосунку з орієнтацією на застосування сучасних технологій та принципів розробки.

Актуальність дипломної роботи полягає в необхідності створення більш багатоцільового та ефективного застосунку для обліку комерційної діяльності кінотеатру.

У роботі висвітлено:

- Види комерційної діяльності кінотетарів;
- Сучасні напрямки в архітектурі додатків;
- особливості принципів програмування SOLID;

Подальший розвиток розробленої інформаційної системи можливий у напрямку створення більшої кількості функцій для адміністратора кінотеатру, а також нововведеннями для заохочення нових глядачів.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

MVC	Model-view-controller
БД	База даних
Dao	Data access object
JDBC	Java Database Connector
JPA	Java Persistence API
ORM	Object Relational Mapping
ID	Identity document
IoC	Inversion of Control
ПЗ	Програмне забезпечення

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ ДІЯЛЬНОСТІ КІНОТЕАТРІВ 8
1.1. Вплив кіно сьогодення 8
1.1.1. Важливість кіногалузі в Україні 9
1.1.2. Розвиток і популярність сучасних кінотеатрів 10
1.2. Дослідження комерційної діяльності кінотеатру 11
1.3. Розгляд та аналіз існуючих аналогів систем обліку комерційної діяльності 20
Висновки до розділу 1 26
РОЗДІЛ 2. ФОРМУВАННЯ ВИМОГ ДО ЗАСТОСУНКУ 27
2.1. Функціональні вимоги до застосунку 27
2.2. Нефункціональні вимоги до застосунку 30
Висновки до розділу 2 33
РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ 34
3.1. Сучасні напрямки у створенні архітектури програмних додатків 34
3.2. Архітектура системи 36
3.3. Технології розробки 37
3.4. Бази даних 41
Висновки до розділу 3 47
РОЗДІЛ 4. ПРОТОТИП ЗАСТОСУНКУ 48
4.1. Створення бази даних 48
4.2. Створення ключових блоків коду 51
Висновки до розділу 4 59
ВИСНОВКИ 60
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ 61
Додаток А 62
Додаток В 66

ВСТУП

Кіно відіграє важливу роль у сучасному суспільстві та культурі, впливаючи на людей на різних рівнях. Воно служить джерелом розваги, дозволяє відволіктися та відпочити. Культурно та соціально, кіно відображає аспекти суспільства. Як форма мистецтва, воно поєднує кінематографічну майстерність з творчістю, надихаючи та захоплюючи, крім того, кіно може мати освітні та просвітницькі функції, розкриваючи нові знання та історії, впливаючи на громадську думку. Фільми піднімають соціальні питання та сприяють обговоренню. Спільний перегляд фільмів може об'єднувати людей та зміцнювати соціальні зв'язки.

Тому я вирішив дослідити та проаналізувати комерційну діяльність кінотеатрів, щоб посприяти кращому поширенню кінопродукції, зокрема, національного виробництва, оскільки кіно може бути ефективною зброєю в інформаційній війні та грати важливу роль у формуванні національних цінностей та культури українського народу.

Культура – це життєва сила суспільства: історія, традиції, мистецтво, самобутність народу.

Культура підвищує якість життя, концентрує соціальний досвід багатьох поколінь, накопичує знання про світ. Система культури – це живий процес, який постійно рухається, розвивається, видозмінюється. Культура є важливою частиною життя кожної людини і суспільства в цілому.

Підтримуючи культуру, можна показати зовнішньому світові свою систему цінностей. Багата культура, цікава за межами країни, створює її позитивний імідж.

РОЗДІЛ 1
АНАЛІЗ ДІЯЛЬНОСТІ КІНОТЕАТРІВ

1.1. Вплив кіно сьогодення

Кіно завжди було потужним засобом розваг і самовираження. Крім сфери розваг, воно суттєво вплинуло на суспільство, надихаючи на зміни та глибоко формуючи культуру.

З самого початку люди завжди знаходили різні способи відпочинку, і кіно це одне з таких відкриттів.

Кіно служить дзеркалом для суспільства, відображаючи його радості, боротьбу та труднощі. Через фільми ми стаємо свідками історій, які перегукуються з нашим власним досвідом, змушуючи нас почуватися пов'язаними та зрозумілими. Він зображує різні культури, способи життя та проблеми, сприяючи співпереживанню та розумінню аудиторії.

У багатьох фільмах розглядаються політичні та соціальні проблеми, проливаючи світло на такі теми, як перевірка реальності країни, бідність, дискримінація та нерівність. Багато фільмів допомогли підвищити обізнаність і надихнути на зміни, зображуючи історичну несправедливість.

Так само це допомагає людям соціалізуватимя. Це об'єднує людей і допомагає розтопити лід. Люди часто обговорюють кіно, щоб почати розмову.

Культурний обмін: Міжнародне кіно знайомить глядачів з різними культурами, сприяючи почуттю глобальної єдності. Такі фільми пропонують зазирнути у світи за межами нашого.

Вплив на культуру є дужу різностороннім і починається від формування переконань і цінностей, закінчуючи громадськими рухами та ідейними змінами.

Кіно, як відображення та вплив на суспільство, продовжує надихати на зміни та формувати культуру. Він має унікальну здатність розпалювати розмови, кидати виклик сприйняттю та викликати емоції, які призводять до реальних дій. У міру просування вперед вплив кіно на суспільство, безсумнівно, залишатиметься динамічною силою, яка надихатиме майбутні покоління.

1.1.1. Важливість кіногалузі в Україні

Розвиток сучасної української кінематографії розпочався у 2015 році, викликаний трагічними подіями на сході України. Ці події визначили культуру загалом, виявивши, що кіно може бути ефективною зброєю в інформаційній війні та грати важливу роль у формуванні національних цінностей та культури українського народу.

Після ухвалення 2017 року Закону «Про державну підтримку кінематографії в Україні» галузь отримала потужний поштовх для розвитку, про що свідчать показники 2017—2019 років. За цей час не тільки значно збільшилася кількість вироблених українських фільмів, а й істотно поліпшилася їхня якість. Це підтверджується результатами кінотеатральних зборів — дедалі більше українських глядачів обирають вітчизняні фільми і готові сплачувати за їх перегляд.

Важливість показу реалій сьогоднішньої війни очами людей, які це переживають є дуже гострою. До прикладу, у документальному фільмі «20 днів у Маріуполі» режисер Чернов Мстислав Андрійович показував, що відбувалося у місті під час повномасштабного вторгнення росії в Україну.

"У фільмі показано те, що ми відчули, коли були у рідному місті. Саме Маріуполь дав час всій Україні, щоб зібрати сили, затримав колосальні сили супротивника" - волонтер Євген Тузов, який допомагав організувати в блокадному Маріуполі сховища, де рятувалися близько шести тисяч людей.

На думку Євгена, Маріуполь — це нагадування про те, що могло статися з кожним містом України.

1.1.2. Розвиток і популярність сучасних кінотеатрів

Хоча за роки пандемії різко зросла популярність онлайн-кінотеатрів через доступність і широкий асортимент контенту «прямо з дому», проте класичні кінотеатри, все ж мають окреме місце в нашому житті.

Станом на 2023 рік в Україні існує понад двадцять дві кінотеатральні мережі, які представлені такими брендами, як Баттерфляй, Blockbuster cinema, Kinolife Cinema, Оскар, Cinema City(Рис. 1.1), Київкінофільм, Одеса-кіно, Лінія Кіно, Злата, FilMax, KinoLand, Kinolife Cinema, Multiplex та Wizoria. Кожна з цих мереж має різну кількість кінотеатрів та кінозалів, призначених для різної кількості глядачів. Це створює різноманіття вибору для кіноглядачів, а також конкуренційне середовище, що може сприяти покращенню якості послуг та розвитку кінематографії в Україні.

Тенденція розвитку кіноіндустрії в україні не скорочується, і тому слід очікувати ще більшій популяризації і доступності кіно. Тому, розвиток кінотеатрів також не буде знижуватись.

Рис 1.1. Різноманіття кінотеатрів

1.2. Дослідження комерційної діяльності кінотеатру

Одним із основних джерел прибутку є продаж квитків на сеанси. Ціни на квитки можуть варіюватися в залежності від часу доби, днів тижня, виду фільму, формату сеансу (наприклад, 3D, IMAX) та інших факторів. Ось кілька ключових елементів, пов'язаних із продажем квитків:

- **Ціноутворення**

Диференціація цін. Квитки можуть мати різні ціни в залежності від ряду факторів, таких як час доби (ранкові сеанси можуть бути дешевшими), дні тижня (будні проти вихідних), формат фільму (3D, IMAX) та інші.

Онлайн-продаж. Зручність для глядачів полягає в можливості купувати квитки онлайн через веб-сайт кінотеатру чи мобільний додаток. Це не тільки економить час глядачів, але й дозволяє кінотеатрам ефективніше керувати обліком та прогнозувати попит.

- **Абонементи та знижки**

Системи абонементів. Багато кінотеатрів пропонують абонементи або «клубні» картки, що дають право на знижки чи безкоштовні квитки після певної кількості відвідувань.

Знижки. Організація різноманітних знижок, таких як студентські, пенсійні, сімейні та інші, може привертати різні групи глядачів.

- **Управління місцями**

Резервація місць. Можливість глядачів заздалегідь забронювати конкретні місця на сеансі стає все популярнішою. Це особливо важливо для популярних фільмів та пікових часів.

- **Лояльність та бонуси**

Запровадження програм лояльності, де глядачі накопичують бонуси чи знижки при купівлі квитків, може стимулювати повторні відвідування.

Продаж подарункових карток або сертифікатів може бути ефективним засобом привертання нових глядачів.

Продаж квитків — це не лише джерело прибутку, але й важливий інструмент для взаємодії з аудиторією та забезпечення зручності та задоволення для глядачів. Ефективне управління цим процесом може позитивно впливати на економічні показники кінотеатру і задовольняти потреби різних категорій глядачів.

- **Продаж продуктів харчування**

Багато кінотеатрів отримують додатковий прибуток від продажу закусок, напоїв, попкорну та інших продуктів харчування. Часто ці товари мають високий маржинальний рівень, що забезпечує додатковий дохід.

Рис. 1.2. Продаж продуктів харчування в кінотеатрі

Розглянемо деталі цього аспекту:

Попкорн: Як символ кінозалу, попкорн є однією з основних страв у кінотеатрах. Кінотеатри можуть пропонувати різні смакові варіанти, такі як солоний, карамельний, сирний і навіть екзотичні смаки.

Напої: Продаж безалкогольних напоїв, таких як газовані напої, соки, молокошоколадні коктейлі та кава, є стандартними в кінотеатрах. Деякі кінотеатри також пропонують алкогольні напої в окремих зонах або VIP-зонах.

Інші закуски: Меню може включати хот-доги, піццу, попкорн з начинкою, плюшки, цукерки, печиво, снеки та інші закуски.

Комбіновані пропозиції: кінотеатри пропонують комбіновані пропозиції типу "кіно+попкорн+напій" за зниженими цінами, стимулюючи глядачів замовляти комплексно.

Здорові страви: Зростаюча увага до здорового способу життя може призвести до включення у меню кінотеатрів більше здорових альтернатив, таких як суші, фрукти, овочі або горіхи.

Безглютенові або веганські опції: Для того, щоб задовольнити різні дієтичні потреби глядачів, кінотеатри можуть пропонувати безглютенові або веганські варіанти.

Технологічні Інновації такі як:

Самообслуговування: Деякі кінотеатри впроваджують системи самообслуговування, де глядачі можуть замовляти їжу та напої через автоматизовані термінали або мобільні додатки.

Онлайн-замовлення: Можливість замовляти їжу онлайн перед відвідуванням кінотеатру дозволяє глядачам ефективно управляти часом та зручно отримувати свої замовлення.

- **Маркетинг та продаж:**

Рекламні кампанії: Ефективні рекламні кампанії можуть підвищити усвідомленість глядачів щодо доступних продуктів харчування та створити збільшену попит на них.

Сезонні акції: Акції в стилі "купуй попкорн, отримай напій безкоштовно" чи інші сезонні спецпропозиції можуть привертати увагу та стимулювати продажі певних товарів.

- **Оренда приміщень**

Кінотеатри можуть отримувати додатковий дохід, орендуючи свої приміщення для проведення різноманітних подій, таких як прес-конференції, презентації, корпоративні заходи.

Конференції: Кінотеатри можуть пропонувати приміщення для проведення конференцій, прес-конференцій чи бізнес-презентацій.

Корпоративні заходи: Оренда приміщень для корпоративних заходів, які включають в себе вечірки, тренінги або командні збори.

Презентації та Покази: Приміщення кінотеатру може бути використане для презентації нових продуктів, виставок, фотографій чи мистецьких проектів.

Тематичні вечори: Оренда приміщення для тематичних вечорів, таких як кінопокази, кінопанорами, презентації короткометражок тощо.

Концерти та Вистави: Великі зали кінотеатрів можуть бути використані для проведення концертів, мюзиклів чи театральних вистав.

Фестивалі та Конкурси: Кінотеатри можуть стати місцем проведення різноманітних фестивалів, конкурсів чи тематичних подій.

Майстер-класи та Лекції: Простори кінотеатру можуть використовуватися для проведення освітніх заходів, таких як майстер-класи, лекції або навчальні сесії.

Оренда приміщень у кінотеатрі може стати вигідною не лише для самого кінотеатру, але й для організаторів різноманітних заходів, які використовують унікальний простір кінозалів для своїх потреб.

- **Спонсорські угоди та реклама**

Кінотеатри можуть укладати угоди з партнерами та спонсорами, які бажають рекламувати свої продукти або послуги серед аудиторії глядачів.

Розглянемо ці аспекти докладніше:

Рекламні Партнерства:

- *Екранна Реклама:* Рекламні ролики та анонси партнерів можуть відтворюватися перед початком сеансу або під час перерв.
- *Спеціальні Рекламні Пакети:* Кінотеатри можуть пропонувати спеціальні рекламні пакети, які включають в себе різноманітні площі для розміщення реклами: на екранах, плакатах, білбордах тощо.

Спонсорські Угоди на Платформі:

- *Логотипи та Згадки:* Спонсори можуть мати свої логотипи та згадки на екранах, афішах, білбордах, квитках та інших пов'язаних з кінотеатром площах.
- *Партнерські Акції:* Рекламні партнери можуть співпрацювати з кінотеатром для організації спільних акцій, конкурсів чи подарунків для глядачів.

Ексклюзивні Продукти та Брендовані Рішення:

- *Брендовані Попкорн-Відділення:* Угоди можуть передбачати брендовані зони для продажу продуктів, таких як попкорн, напої та інші снеки.

- *Ексклюзивні Продукти:* Спонсори можуть пропонувати ексклюзивні продукти або акції, що додає унікальність досвіду відвідування кінотеатру.

Джерела Додаткового Доходу:

- *Партнерські Програми:* Спонсорські угоди можуть включати програми, за якими кінотеатр отримує комісійні за продажі продуктів чи послуг партнерів.
- *Продажі на Лобі:* Продажі продуктів чи сервісів спонсорів в лобі кінотеатру.

Інтерактивні Технології:

- *Розумні Зони*: Використання розумних технологій для створення інтерактивних зон, де глядачі можуть взаємодіяти з рекламним контентом.
- *Мобільні Додатки*: Спонсори можуть розміщати рекламу чи спеціальні пропозиції у мобільних додатках кінотеатру.

Збільшення Увідомленості та Взаємодії:

- *Соціальні Мережі*: Рекламні партнери можуть сприяти популяризації кінотеатру через соціальні мережі та збільшення кількості підписників.
- *Брендовані Заходи*: Організація заходів з партнерами, що включають в себе брендовані елементи, може створити великий інтерес серед глядачів.

Спонсорські угоди та реклама надають кінотеатру можливість забезпечити фінансову стабільність, а глядачам — розширений та цікавий досвід відвідування.

- **Прокат фільмів**

Комерційна діяльність кінотеатру включає угоди з дистриб'юторами фільмів для отримання прав на їх прокат. Кінотеатр отримує частину виручки від продажу квитків, а решта йде на користь дистриб'ютора.

Цей процес включає в себе кілька ключових етапів:

Домовленості з Фільмовими Студіями:

- *Вибір Фільму:* Кінотеатр укладає угоди з фільмовими студіями чи дистриб'юторами для прокату конкретного фільму. Цей етап може включати переговори про умови прокату, такі як тривалість показу, умови відшкодування, територія прокату тощо.

Визначення Умов Прокату:

- *Вартість Прокату:* Угода визначає вартість прокату, яку кінотеатр повинен сплатити за право показу фільму.
- *Терміни Прокату:* Домовленості включають в себе терміни прокату, тобто період, протягом якого кінотеатр може показувати фільм.

Підготовка до Прем'єри:

- *Маркетингові Заходи:* Кінотеатр розпочинає маркетингові заходи для просування майбутньої прем'єри. Це може включати в себе рекламні кампанії, створення афіш, трейлерів, анонсів тощо.

Прем'єра та Період Першоденного Прокату:

- *Першоденний Прокат:* Фільм має свою прем'єру, і період першоденного прокату розпочинається. Протягом цього часу фільм відображається в кінотеатрах для широкого глядацького потоку.

Організація Додаткових Показів:

- *Спеціальні Покази:* Кінотеатр може організовувати спеціальні покази фільму, такі як нічні сеанси, покази для певних груп глядачів чи тематичні вечори.

Завершення Прокату:

- *Підведення Підсумків:* Після завершення термінів прокату кінотеатр підводить підсумки, визначає успішність фільму в їхньому залі та вирішує, чи буде продовжено показ фільму.

Прокат фільмів є важливим аспектом фінансової діяльності кінотеатру, і успіх цього процесу в значній мірі залежить від правильної стратегії маркетингу та вибору популярних і цікавих фільмів для показу.

- **Розваги та додаткові послуги**

Розваги та додаткові послуги в кінотеатрі допомагають створювати унікальний та комфортний досвід для глядачів, привертаючи їхню увагу та роблячи відвідування кінотеатру приємним та запам'ятовуваним. Докладніше деякі з можливих розваг та додаткових послуг:

VIP-зони та Послуги:

- *VIP-Зали:* Створення преміальних VIP-зон або кінозалів з комфортабельними кріслами, обслуговуванням персоналу та іншими ексклюзивними можливостями для важливих клієнтів.
- *VIP-Обслуговування:* Надання послуг VIP-обслуговування, таких як швидке обслуговування, приватні ліжка чи індивідуальний персонал.

Тематичні Вечори та Заходи:

- *Тематичні Кінопокази:* Організація тематичних кінопоказів, наприклад, кінопоказів класичних фільмів, нічних сеансів, фестивалів чи спеціальних подій.
- *Тематичні Вечори:* Проведення тематичних вечорів, де кінотеатр може оформлюватися під певну тему, а також надавати вечірні розваги.

Гастрономічні Варіанти:

- *Ресторани та Кафе:* Встановлення ресторанів або кафе в лобі кінотеатру, де глядачі можуть насолоджуватися їжею перед або після сеансу.
- *Доставка Їжі:* Послуга доставки їжі прямо до крісел глядачів під час перерв чи під час перегляду фільму.

Інтерактивні Технології:

- *Віртуальна Реальність:* Використання технологій віртуальної реальності для створення інтерактивних вражень перед сеансом чи під час перерв.

Дитячі Майданчики:

- *Ігрові Зони:* Створення ігрових зон або кімнат для дітей, де вони можуть розважатися до початку фільму або під час перерви.
- *Мультфільмові Сеанси:* Проведення спеціальних сеансів для дітей з мультфільмами та іншими розважальними заходами.

Ці розваги та додаткові послуги розширюють можливості кінотеатру, забезпечуючи глядачам різноманітні інтереси та створюючи унікальний досвід відвідування.

1.3. Розгляд та аналіз існуючих аналогів систем обліку комерційної діяльності

Ознайомившись з комерційною діяльністю кінотеатру, стає зрозуміло, що для популяризації та розповсюдження послуг кіно, недостатньо лише різноманітної реклами та хороших фільмів. Не менш важливим аспектом є зручність придбання послуг.

Одним з ключових аспектів є розповсюдження та продаж квитків на події. Зараз оптимальним вважається продаж квитків на концерти через Інтернет. Хоча каси з квитками біля входу залишаються популярними, нові технології з часом замінюють їх. Інтернет-маркетинг та онлайн-системи продажу квитків стають дедалі популярнішими завдяки їхній ефективності та зручності для клієнтів.

Ось кілька стратегій, які можуть сприяти ефективному продажу квитків на концерти:

1. **Створення веб-сайту або використання онлайн-систем продажу квитків.**
2. **Використання соціальних медіа-платформ для просування концерту та продажу квитків.** Створення сторінки події на Twitter Facebook, або ж Instagram рекламування концерту в спеціалізованих групах та проведення спонсорованих оголошень.
3. **Використання електронних розсилок для просування концерту та продажу квитків.** Створення списку розсилки для попередніх покупців та тих, хто підписався на новини.
4. **Використання партнерських програм та співпраця з іншими компаніями для просування концерту та продажу квитків.** Залучення музичних магазинів, радіостанцій та відомих людей для реклами концерту.
5. **Забезпечення зручності та безпеки для покупців квитків.** Включення безкоштовної доставки, захисту від шахрайства та можливості повернення коштів у разі скасування концерту.

Сервіс онлайн-продажу квитків у кіно “Вкіно”

Ви можете обрати місто проживання, а далі вже продовжувати роботу з обраним кінотеатром (Рис. 1.3)

Представлений сервіс має такий функціонал:

1) Можливість провести пошук за містом.
2) Функція пошуку за назвою фільму.
3) Можливість знаходження кінотеатрів.
4) Опція бронювання квитка.
5) Можливість покупки квитка.
6) Функція повернення квитка.
7) Можливість перегляду всієї інформації про фільм.

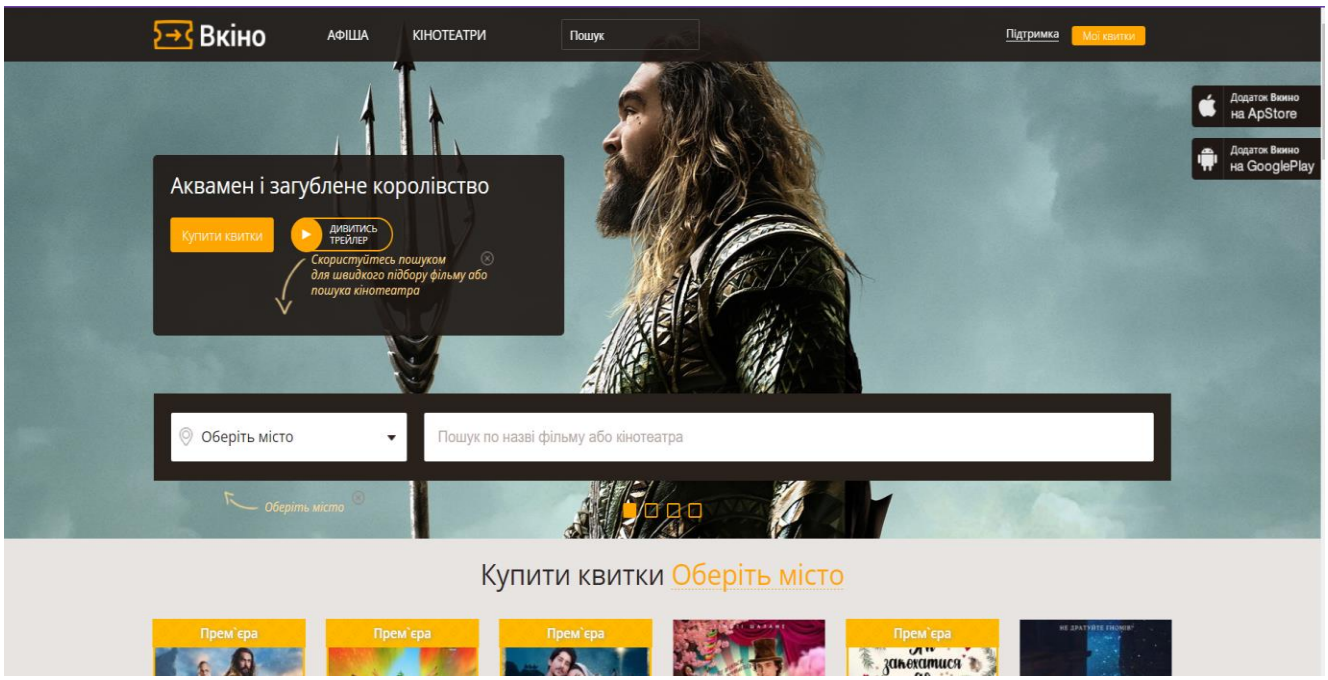

Рис. 1.3. Вигляд готовної сторінки додатку «Вкіно»

Цей сервіс також представлений додатками для мобільних операційних систем iOS та Android, які зберегли всі функції основного сайту. Завдяки додатку можна:

- Здійснити пошук фільму.
- Знайти кінотеатр.
- Купити чи забронювати квиток.

Мобільний додаток “Планета Кіно

Ще одним великим сервісом, представленим в тому числі на Android, є “Планета кіно” (Рис 1.4)

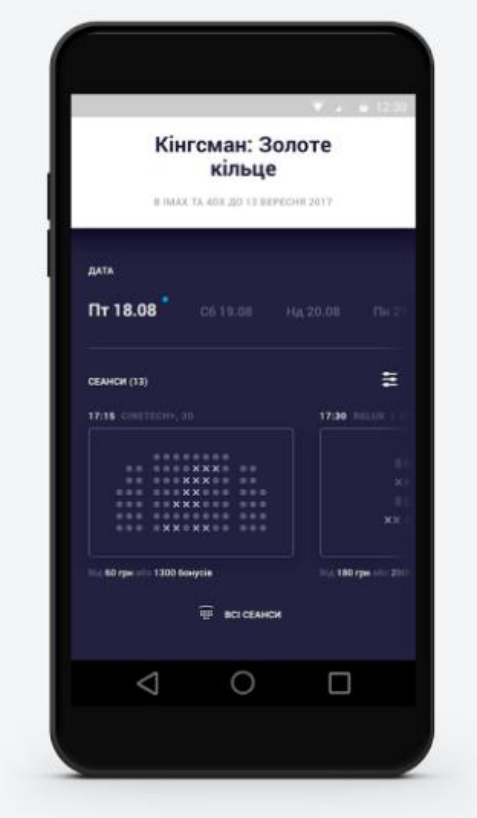

Рис. 1.4. Мобільний додаток «Планета кіно»

Сервіс "Планета кіно" пропонує наступний функціонал:

- Можливість вибору міста, фільму та сеансу.
- Перегляд повної інформації про фільм.
- Опція купівлі квитка.

Сервіс онлайн-продажу квитків у кіно «Прем’єра»

Однак крім дуже відомих сервісів онлайн-продажу квитків є і локальні сервіси з не менш цікавими функціями, зроблені під окремі кінотеатри. До прикладу онлайн-сервіс продажу квитків «Прем’єра», яке представляє лише три міста і три кінотеатри.

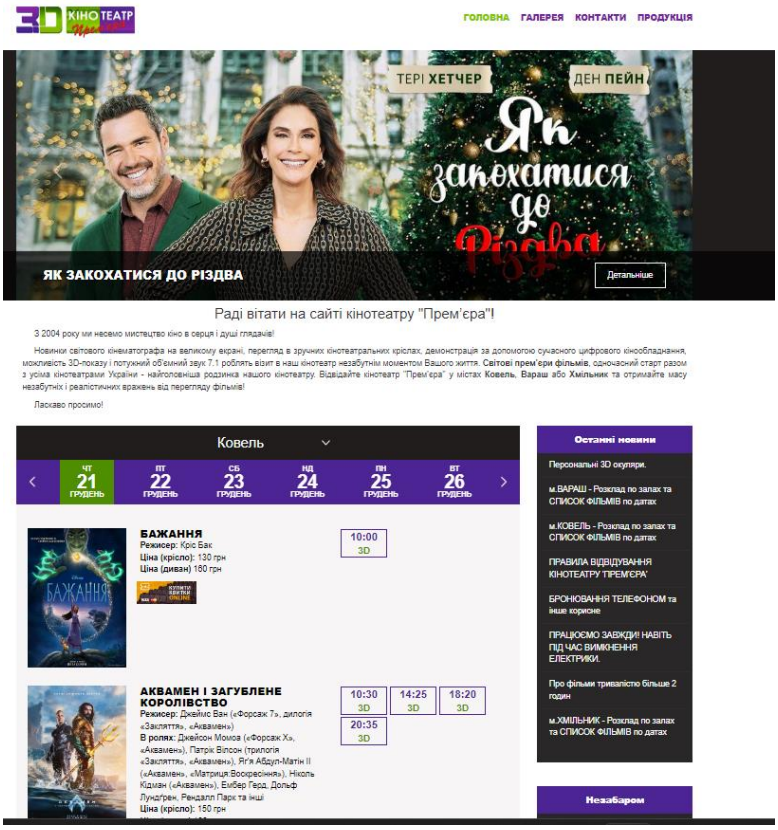

Рис 1.5. Головна сторінка онлайн-сервісу продажу квитків «Прем’єра»

Проте, сервіс має всі необхідні функції для повноцінної роботи, до того ж, окрім функцій продажу квитків, можливості перегляду інформації про фільм, надає змогу переглянути асортимент представленої продукції даного кінотеатру, та деяких спеціальних пропозицій. (Рис. 1.6)

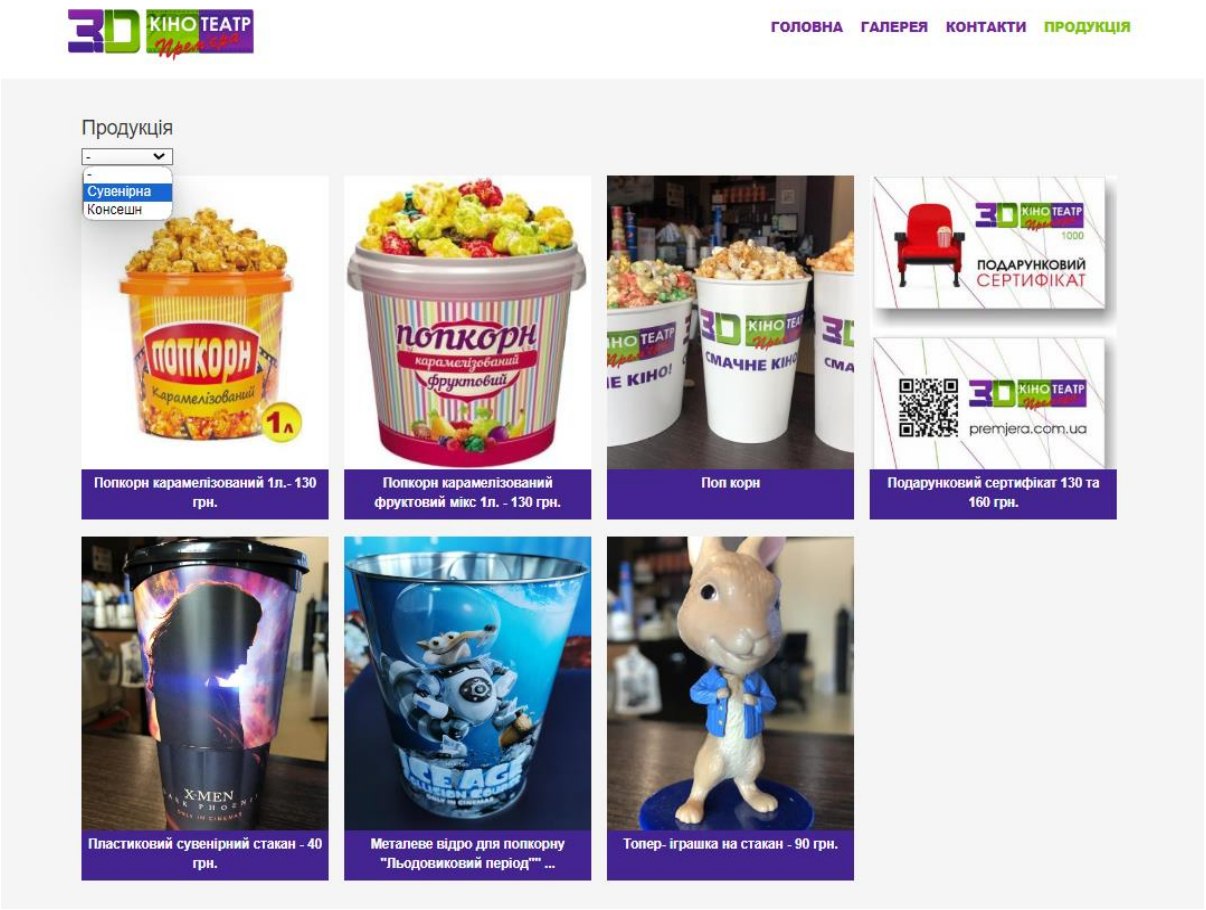

Рис. 1.6. Розділ «Продукція» у сервісі кінотеатр «Прем’єра»

Представлені сервіси є необхідними для існування на життя кінотеатру, оскільки використання має численні можливості для продажу квитків. До того ж, це ще надає:

1. **Широке охоплення ринку:**
    - Співпраця з різноманітними організаторами розширює аудиторію та забезпечує більше можливостей для залучення покупців.
2. **Зручність та доступність:**
    - Зручний функціонал для пошуку та відбору заходів дозволяє користувачам легко купувати квитки онлайн з будь-якого місця світу, а цілодобова робота сервісу забезпечує доступність для користувачів у будь-який час.
3. **Маркетингова підтримка:**
    - Допомога в просуванні та маркетингу заходів дозволяє організаторам привертати більше уваги та збільшувати продажі квитків.
4. **Аналітика:**
    - Надання детальної аналітики продажів та поведінки користувачів допомагає організаторам розуміти свою аудиторію та адаптуватись до її потреб.
5. **Безпека:**
    - Використання надійних систем безпеки та захисту даних забезпечує безпечність покупок у середовищі сервісу.
6. **Різноманітність способів оплати:**
    - Наявність різноманітних способів оплати сприяє зручності користувачів та підвищує ймовірність покупки квитків.
7. **Мобільний додаток:**

- Мобільний додаток дозволяє швидко та зручно купувати квитки та отримувати інформацію про події з будь-якого місця та в будь-який час.

8. **Підтримка клієнтів:**

   - Професійна команда підтримки клієнтів готова допомагати вирішувати будь-які питання щодо покупок та роботи сервісу.

Висновок до розділу 1

У цьому, розділі було розглянуто важливість кіно у житті сучасної людини, також необхідність просування національного продукту. Була проведене аналітика комерційної діяльності кінотеатрів, зокрема, досліджено такі питання як: продаж квитків, продаж продуктів харчування, оренду приміщень, спонсорські угоди та реклама, прокат фільмів, розваги та додаткові послуги. І було розглянуто та проаналізовано декілька аналогічних програм.

РОЗДІЛ 2.
ФОРМУВАННЯ ВИМОГ ДО ЗАСТОСУНКУ

Функціональні та нефункціональні вимоги є основними категоріями вимог у розробці програмного забезпечення. Кожен тип відіграє важливу роль у визначенні характеристик і функціонування рішення.

2.1. Функціональні вимоги до застосунку

Функціональні вимоги — це властивості або функції продукту, які розробники повинні реалізувати, щоб користувачі могли виконувати свої завдання. Тому важливо зробити їх зрозумілими як для команди розробників, так і для зацікавлених сторін. Загалом функціональні вимоги описують поведінку системи за конкретних умов.

Функціональні вимоги будуть різними для різних типів програмного забезпечення. Наприклад, функціональні вимоги до веб-сайту або мобільного додатку повинні визначати потоки користувачів і різні сценарії взаємодії.

Реєстрація та Авторизація:

- Реєстрація передбачає заповнення форми з основними особистими даними. Авторизація включає в себе введення ідентифікаційних даних (логін, пароль).
- Аутентифікація учасників з можливістю відновлення забутого пароля.
- Керування профілями користувачів, що включає в себе можливість зміни особистої інформації, налаштувань, приватності та паролю.
- Визначення ролей учасників (організатор, відвідувач) з можливістю делегування прав доступу та рівнів відповідальності.

Адміністративна Панель:

- Можливість створення нового залу для проведення подій.
- Можливість прикріплення опису до кожного залу, вказання його характеристик та місткості.
- Можливість редагування вже введеної інформації про кожен зал.
- Можливість видалення залу, а також архівування та відновлення його даних.
- Можливість призначення кінозалу для конкретної події.

Пошук та Вибір Фільмів:

- Користувач може використовувати різні параметри для пошуку фільмів: назва, жанр, актори, режисери. Результати пошуку повинні відображати відповідні фільми.

Вибір Місць та Сеансів:

- Після вибору фільма користувач обирає конкретний сеанс та місця в кінозалі на інтерактивній карті.

Інтерактивна Карта Залу:

- Графічне відображення залу, де користувач може бачити вільні та обрані місця.

Детальна Інформація про Фільми:

- Сторінка фільму повинна містити трейлер, опис сюжету, інформацію про акторів, режисера, рейтинг та рецензії.

Бронювання та Покупка Квитків:

- Після вибору сеансу та місць, користувач може обрати між бронюванням та покупкою квитків.

Множинні Варіанти Оплати:

- Інтеграція з відомими платіжними системами для прийому оплати за квитки.
- Керування транзакціями, включаючи відображення історії платежів та відслідковування статусу оплати.

Електронні Квитки:

- Створення та відображення електронних квитків у формі QR-кодів.

Знижки та Промокоди:

- Введення знижок для різних категорій глядачів або застосування промокодів під час покупки.

Сповіщення та Підтримка:

- Відсилання автоматичних сповіщень про підтвердження покупки
- Зміну сеансів та надання підтримки через чат або електронну пошту.

Історія та Облік Покупок:

- Зберігання історії покупок у особистому кабінеті користувача.
- Інтерфейс для адміністраторів для додавання та редагування фільмів, сеансів, налаштування цін та керування іншими параметрами системи.

Аналітика та Звіти:

- Статистика кількості подій за різними критеріями (дата проведення, ціна на квиток, жанр і т. д.).
- Звіти щодо продажу квитків, обсягу прибутку та заповненості залу на кожній події.
- Аналіз проведених заходів, включаючи уподобані жанри відвідувачів, популярність цінових сегментів та зал, який привертає найбільше глядачів за конкретний період.
- (Опціонально) Можливість експорту звітів у форматах CSV, Excel, PDF для подальшого аналізу та обробки.

Ці деталізовані функціональні вимоги допоможуть у створенні повноцінного та ефективного застосунку. Наступним етапом буде розгляд нефункціональних вимог до проекту.

2.2. Нефункціональні вимоги до застосунку

Нефункціональні вимоги не пов’язані з функціональністю системи, а радше визначають, як система повинна працювати. Вони мають вирішальне значення для забезпечення зручності використання, надійності та ефективності системи, часто впливаючи на загальний досвід користувача. Основні категорії нефункціональних вимог будуть описані нижче.

Специфікація нефункціональних вимог:

Продуктивність:

- Забезпечення миттєвої реакції програми на користувацькі дії для ефективного використання.
- Оптимізація запитів до бази даних для швидкого отримання інформації без зайвання навантаження.
- Оптимізація завантаження контенту, включаючи відео та аудіо, під час додавання опису події.

Масштабованість:

- Здатність платформи безперешкодно росту кількості користувачів, подій та інших об'єктів за умови ефективного використання ресурсів.
- Гнучка архітектура для легкої інтеграції нових функцій та розширень.

Сумісність:

- Підтримка різних веб-браузерів, таких як Chrome, Firefox, Safari та Edge.
- Сумісність з операційними системами Android і iOS для роботи на мобільних пристроях.

Безпека:

- Захист конфіденційності даних та транзакцій у відповідності зі стандартами безпеки.
- Реалізація рівнів доступу для різних ролей користувачів з метою запобігання несанкціонованому доступу.

- Забезпечення захисту даних без значного впливу на основні функціональності системи.

Надійність:

- Забезпечення високої доступності та мінімізації часу простою системи.
- Реалізація системи резервного копіювання та відновлення для уникнення втрати інформації.
- Надійне виконання процесів адміністрування системи.

Ергономіка та технічна естетика:

- Забезпечення зручного інтерфейсу користувача з використанням спокійної та недомінуючої кольорової палітри.
- Застосування однакового стилю та шрифту для опису кожної події/залу з метою створення прийнятного вигляду.

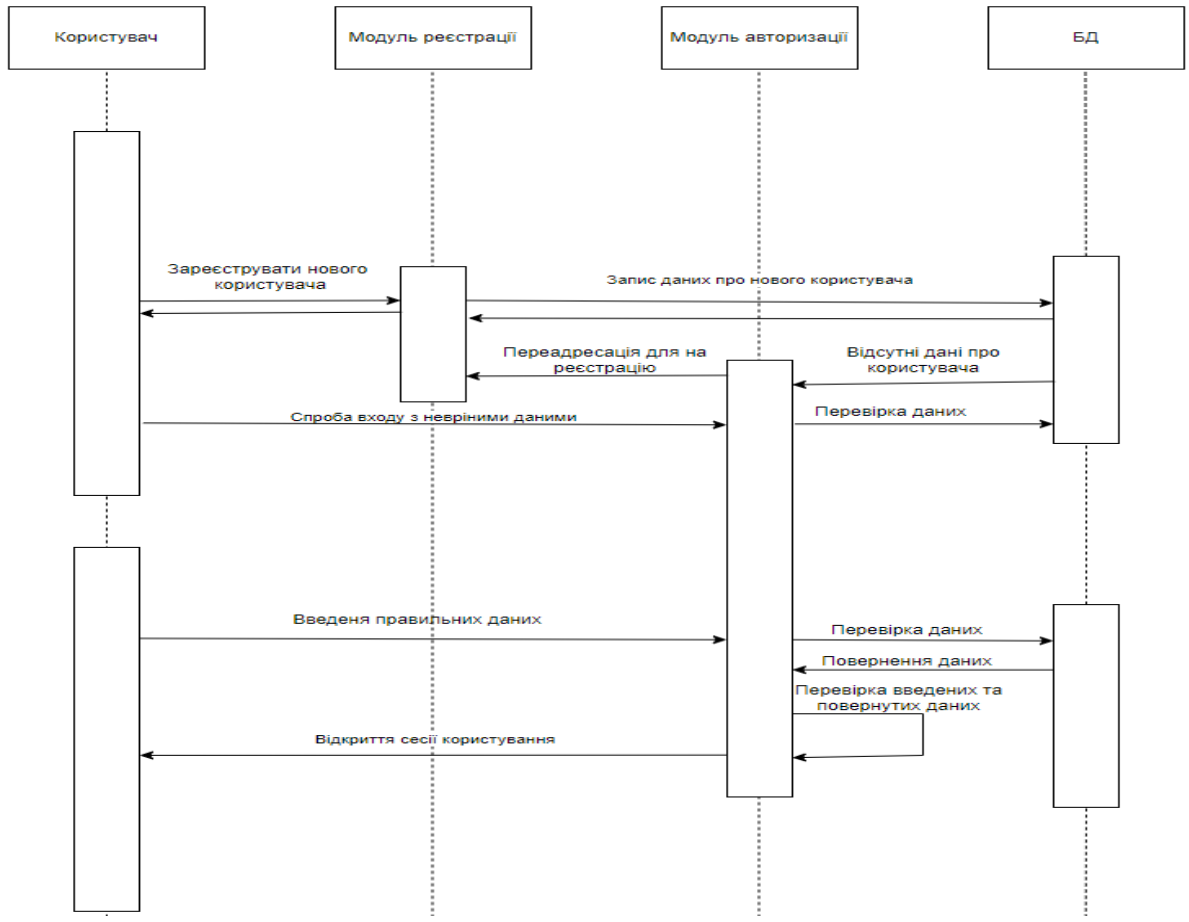

Рис. 2.1. Діаграма послідовностей (сценарій входу в систему)

Після визначення всіх характеристик для розроблюваного продукту, буде створена діаграма послідовностей. (рис. 2.1).

На першій діаграмі розглянуто сценарій входу в систему, що описує процес входження користувача з різних початкових точок, таких як зареєстрований користувач, новий користувач або користувач, який ввів неправильні дані.

Наступною буде діаграма, яка відображатиме можливості користувачів (відвідувача та організатора) після успішного входження в систему.

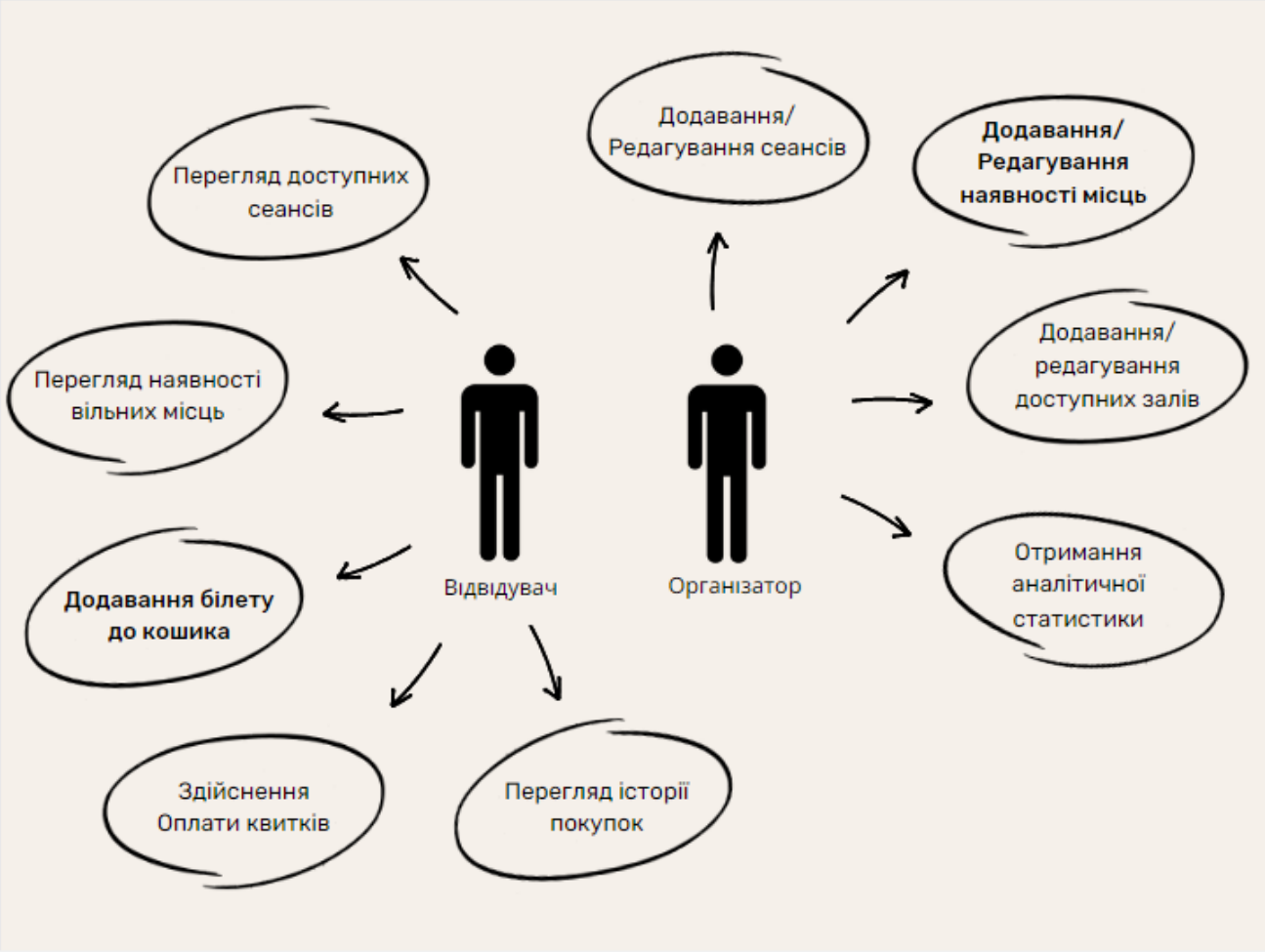

Рис 2.2. Діаграма розподілу можливостей користувачів

Висновок до розділу 2

Під час обробки цього розділу було розказано про можливі функціональні та нефункціональні вимоги до застосунку. Крім того, були уточнені основні можливості ролей, які будуть доступні у застосунку, надана діаграма розподілу можливостей користувачів. Була створена діаграма послідовностей.

РОЗДІЛ 3
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

3.1. Сучасні напрямки у створенні архітектури програмних додатків.

Архітектура додатків — це структурна карта того, як компонуються програмні додатки організації та як ці додатки взаємодіють один з одним, щоб відповідати вимогам бізнесу чи користувача. Архітектура додатків гарантує, що додатки масштабовані та надійні, а також допомагає підприємствам виявляти прогалини у функціональності.

Загалом, архітектура програми визначає, як програми взаємодіють з такими об’єктами, як проміжне програмне забезпечення, бази даних та інші програми. Архітектури додатків зазвичай дотримуються принципів проектування програмного забезпечення, які є загальноприйнятими серед прихильників, але можуть не відповідати формальним галузевим стандартам.

Однією з найвідоміших архітектур додатків є сервісно-орієнтована архітектура (SOA), яка виникла в 1990-х роках, коли інтеграція додатків і операції спільного використання компонентів стали пов’язані з пулами хостингових ресурсів і розподіленими базами даних. За останні 20 років SOA розвинулась у кілька інших архітектур, особливо в архітектуру мікросервісів.

Переваги архітектури програми

Загалом архітектура додатків допомагає IT-спеціалістам і спеціалістам з бізнес-планування працювати разом, забезпечуючи доступність правильних технічних рішень для досягнення бізнес-цілей. Точніше, архітектура програми:

Зменшує витрати шляхом виявлення надмірностей, таких як використання двох незалежних баз даних, які можна замінити однією;

Підвищує ефективність шляхом виявлення прогалин, таких як основні служби, до яких користувачі не можуть отримати доступ через мобільні програми;

Створює корпоративну платформу для доступності програм та інтеграції сторонніх розробників;

Дозволяє використовувати сумісні модульні системи, які прості у використанні та обслуговуванні;

Допомагає архітектору «побачити загальну картину» та узгодити програмні стратегії із загальними бізнес-цілями організації. Для вирішення різноманітних завдань у розробці архітектури використовуються патерни, відомі також як шаблони.

Патерн описує конкретну задачу, і коли ця задача виникає знову в певному контексті, використання патерну дозволяє уникнути і скористатися готовим рішенням. Шаблон – це опис взаємодії різних елементів та класів, що утворюють архітектуру програмного рішення, адаптований до конкретного контексту задачі та вимог проектування. Завдяки можливості ідентифікувати ключові об'єкти структури та використовувати їх повторно, патерни спрощують процес побудови архітектури.

Кожен шаблон має свої переваги та недоліки. Ось кілька прикладів готових архітектурних шаблонів та короткий опис принципів їх роботи:

1. Багаторівневий шаблон: Розділяє систему на рівні, які взаємодіють за певними правилами. Це спрощує модифікацію окремих компонентів, не зачіпаючи інші, але може ускладнити структуру та вплинути на продуктивність.
2. Шаблон посередника: Використовується для спрощення взаємодії модулів, знижуючи їх пряму взаємодію. Це полегшує функціональну сумісність, але може створити слабку ланку у системі.
3. Клієнт-серверний патерн: Застосовується для обмеження прав доступу до ресурсів, роблячи систему більш масштабованою та зрозумілою.

4. Паттерн «Фабричний метод»: Дозволяє додавати нові об'єкти різних типів, полегшуючи процес їхнього введення та зберігаючи систему незалежною.
5. Модель-Подання-Контролер (MVC): Відділяє інтерфейс від даних, дозволяючи змінювати його без порушення принципів роботи системи.

У сфері розробки програмного забезпечення існує значна кількість шаблонів, які використовуються в архітектурі.

3.2. Архітектура системи

Для створення застосунку зобліку комерційної діяльності планується використовувати шаблон Model-View-Controller (MVC), який широко відомий у сфері веб-розробки. Цей шаблон передбачає, що програма або додаток має складатися з трьох ключових компонентів: моделі даних, представлення та контролера. Кожна з цих компонентів представлена окремим об'єктом, і вони чітко розділені між собою.

Шаблон MVC включає наступні шари:

1. **Модель (Model):**
    - Включає в себе дані програми та повний функціональний набір.
    - Містить бізнес-логіку та взаємодіє з базою даних.
2. **Представлення (View):**
    - Містить дані для відображення користувачеві.
    - Відповідає за взаємодію з користувачем.
3. **Контролер (Controller):**
    - Діє як посередник між моделлю та представленням.
    - Відслідковує зміни стану та обробляє вхідні дані від користувача.

Взаємодія між цими шарами має такий вигляд:

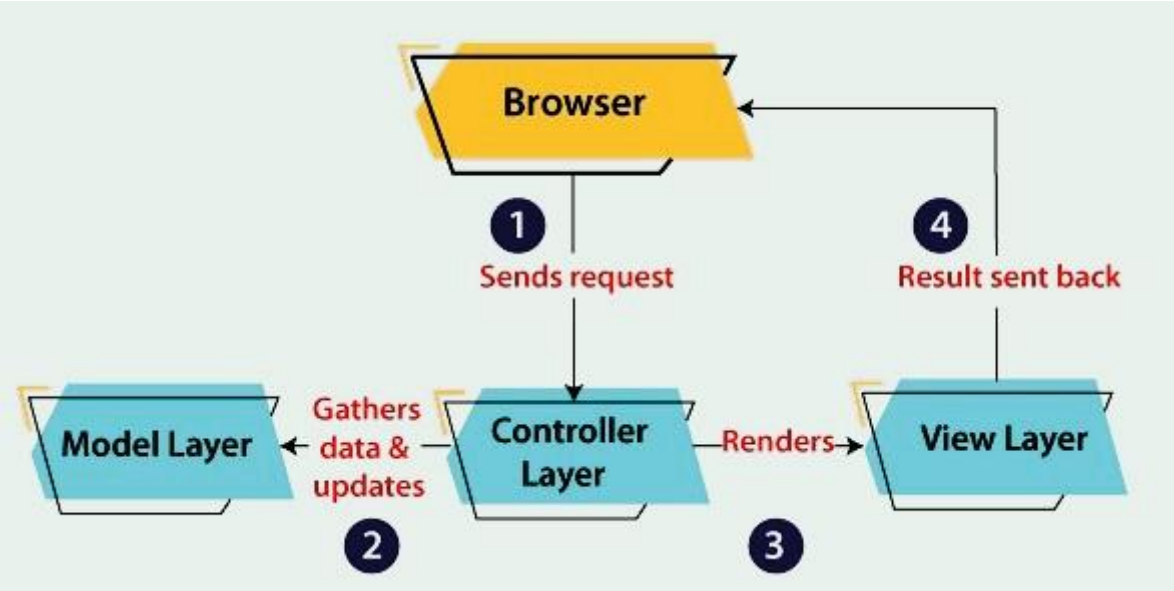

Рис. 3.1. Модель взаємодії шарів MVC

Модель є найнижчим рівнем, куди поступає інформація для обробки, зберігання та подальшого використання. Контролер виконує роль посередника, обробляючи дані від користувача та взаємодіючи з моделлю. Представлення відповідає за обробку та відображення інформації для користувача.

Застосування шаблону MVC дозволяє розподілити розробку проєкту на окремі блоки, які можуть працювати паралельно та незалежно, що сприяє ефективності та можливості розробки різними командами.

3.3. Технології розробки

Після вибору структури проєкту було визначено ключові технології та принципи, які будуть використані в розробці. Основною мовою програмування для написання застосунку обрано Java. Під час написання коду буде застосовано принципи програмування SOLID та підходи RESTful.

Для спрощення взаємодії з веб-частиною та базою даних використовуватимуться відомі фреймворки, зокрема Spring та Hibernate. В якості системи управління базами даних обрано реляційну систему MySQL.

- **Java**: Об'єктно-орієнтована мова програмування, яка була випущена компанією "Sun Microsystems" у 1995 році та в даний час розвивається "Oracle". Java є платформо-незалежною мовою, компільованою в байт-код, який виконується на віртуальній машині Java. Мова є основним компонентом платформи Java та є популярною для розробки бізнес-додатків.
- **SOLID**: Це абревіатура, що включає п'ять основних принципів об'єктно-орієнтованого програмування (ООП). Кожна літера в SOLID представляє один принцип, спрямований на створення гнучких, масштабованих та легко змінюваних програм.
- **RESTful**: Архітектурний стиль для взаємодії між комп'ютерними системами в Інтернеті, забезпечуючи стандартизацію та простоту взаємодії. RESTful системи характеризуються відсутністю стану та розділенням проблем клієнта та сервера.
- **Spring та Hibernate**: Популярні фреймворки для розробки веб-додатків та роботи з базами даних відповідно. Spring дозволяє легко створювати робочі веб-додатки, а Hibernate спрощує взаємодію з реляційними базами даних.
- **MySQL**: Реляційна система управління базами даних, обрана для зберігання інформації у проекті.

Звідно із принципом єдиної відповідальності, кожен клас повинен бути відповідальний лише за одну конкретну функцію чи аспект роботи програми, що робить його гнучким і легко змінюваним. Принцип відкритості/закритості вказує на те, що класи повинні дозволяти розширення функціональності шляхом додавання нових класів, а не змінювати існуючий код. Принцип підстановки Лісков стверджує, що об'єкти підтипів повинні бути замінюваними екземплярами базових класів без втрати коректності програми.

Принцип розділення інтерфейсу наголошує на тому, що клієнти повинні залежати лише від тих інтерфейсів, які вони використовують, уникайте непотрібних залежностей. Принцип інверсії залежностей вказує, що модулі високого та низького рівня повинні залежати від абстракцій, а не від конкретних реалізацій, що сприяє створенню більш гнучкої та розширюваної структури програми.

REST, або REpresentational State Transfer, представляє собою архітектурний стиль, розроблений Роєм Філдінгом у 2000 році, який сприяє встановленню стандартів між комп'ютерними системами в Інтернеті, спрощуючи їх взаємодію. Системи, сумісні із REST, відомі як RESTful, характеризуються відсутністю збереження стану та розділенням проблем клієнта та сервера. Основні принципи RESTful включають:

1. **Клієнт-сервер:** Розділення інтерфейсу користувача та зберігання даних покращує переносимість інтерфейсу та сприяє масштабованості за рахунок спрощення компонентів сервера.
2. **Без збереження стану:** Кожен запит від клієнта повинен містити всю необхідну інформацію, і стан сеансу повністю зберігається на клієнті.
3. **Кешування:** Визначення кешованих даних у відповіді для подальшого використання клієнтом під час еквівалентних запитів.
4. **Уніфікований інтерфейс:** Спрощення архітектури системи через встановлення чотирьох обмежень інтерфейсу, включаючи ідентифікацію ресурсів, маніпулювання ресурсами через уявлення, використання самоописових повідомлень та гіпермедіа як засобу керування станом програми.
5. **Багаторівнева система:** Ієрархічна структура компонентів, де кожен рівень обмежує поведінку компонентів, підтримуючи ієрархічну взаємодію.
6. **Код на вимогу (опціонально):** Можливість розширення функціональності клієнта завдяки завантаженню та виконанню коду у вигляді аплетів або сценаріїв для спрощення роботи клієнтів.

RESTful API — це інтерфейс, який забезпечує безпечний обмін інформацією між двома комп'ютерними системами через Інтернет. Багато бізнес-додатків потребують взаємодії з іншими програмами, і тут важливо використовувати надійні та ефективні стандарти програмної взаємодії. RESTful API відповідає цим вимогам, слідуючи принципам безпеки та ефективності.

Spring є одним із найпопулярніших фреймворків з відкритим кодом для розробки корпоративних додатків на мові програмування Java. Його розширена підтримка для розробки додатків дозволяє ефективно використовувати можливості мови Java. Spring Framework вирішує проблеми організації основних компонентів додатку через принцип Inversion of Control (IoC), спрощуючи компонування елементів у готовий до використання програмний продукт.

Hibernate, у свою чергу, є фреймворком, спрощуючим взаємодію програм на Java з реляційними базами даних. Використовуючи підхід ORM (Object Relational Mapping), Hibernate дозволяє відображати об'єкти та дані бази даних, реалізуючи специфікацію JPA (Java Persistence API). JPA, як стандартна специфікація Java, надає функціональні можливості та інтерфейси для інструментів ORM, і Hibernate виступає як його реалізація.

MySQL є однією з найбільш поширених систем управління базами даних (СУБД) на сьогоднішній день. Це пояснюється її відносною простотою використання, широким функціоналом та високим рівнем безпеки за замовчуванням. MySQL є реляційною базою даних з відкритим програмним забезпеченням, що дозволяє її використовувати безкоштовно та змінювати за необхідністю.

Однією з ключових переваг MySQL є легка інтеграція з різними мовами програмування через API. Розробник може легко отримати доступ до СУБД зі своєї програми, використовуючи MySQL Java Database Connector (JDBC) для з'єднання. Цей драйвер, написаний на Java, не потребує встановлення додаткових сторонніх бібліотек. Підключення до сервера бази даних є досить простим:

достатньо мати URL-адресу сервера, ім'я користувача, пароль і назву бази даних. Таким чином, ця СУБД ідеально підходить для реалізації застосунків і є легкою в застосуванні.

3.4. Бази даних

База даних — це організований набір структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді в комп’ютерній системі. База даних зазвичай контролюється системою керування базами даних (СУБД). Разом дані та СУБД разом із додатками, які з ними пов’язані, називають системою баз даних, яку часто скорочують до просто бази даних.

Дані в найпоширеніших типах баз даних, що працюють сьогодні, зазвичай моделюються в рядках і стовпцях у серії таблиць, щоб зробити обробку та запити даних ефективними. Потім можна легко отримати доступ до даних, керувати ними, змінювати, оновлювати, контролювати та організовувати. Більшість баз даних використовують структуровану мову запитів (SQL) для запису та запиту даних.

Що таке мова структурованих запитів (SQL)?

SQL — це мова програмування, яка використовується майже всіма реляційними базами даних для запитів, обробки та визначення даних, а також для забезпечення контролю доступу. SQL вперше було розроблено в IBM у 1970-х роках за участю Oracle як основного учасника, що призвело до впровадження стандарту SQL ANSI, SQL стимулював багато розширень від таких компаній, як IBM, Oracle і Microsoft. Хоча SQL все ще широко використовується сьогодні, починають з’являтися нові мови програмування.

Еволюція бази даних

Бази даних різко еволюціонували з моменту їх появи на початку 1960-х років. Навігаційні бази даних, такі як ієрархічна база даних (яка спиралася на

деревоподібну модель і допускала лише зв’язок «один-до-багатьох») і мережева база даних (більш гнучка модель, яка допускала численні зв’язки), були оригінальними системами, які використовувалися для зберігання і маніпулювати даними. Хоча ці ранні системи були простими, вони були негнучкими. У 1980-х роках стали популярними реляційні бази даних, а в 1990-х роках – об’єктно-орієнтовані бази даних. Зовсім недавно бази даних NoSQL з’явилися як відповідь на зростання Інтернету та потребу у вищій швидкості й обробці неструктурованих даних. Сьогодні хмарні бази даних і автономні бази даних відкривають нові шляхи, коли йдеться про те, як дані збираються, зберігаються, керуються та використовуються.

Рис. 3.2. Різноманіття Бази даних

Яка різниця між базою даних і електронною таблицею?

Бази даних і електронні таблиці (такі як Microsoft Excel) є зручними способами зберігання інформації. Основні відмінності між ними:

- Як дані зберігаються
- Хто може отримати доступ до даних
- Скільки даних можна зберігати

Електронні таблиці спочатку були розроблені для одного користувача, і їх характеристики відображають це. Вони чудово підходять для одного користувача або невеликої кількості користувачів, яким не потрібно робити багато неймовірно складних маніпуляцій даними. Бази даних, з іншого боку, призначені для зберігання значно більших колекцій упорядкованої інформації — іноді величезних обсягів. Бази даних дозволяють кільком користувачам одночасно отримувати швидкий і безпечний доступ до даних і запитувати їх за допомогою дуже складної логіки та мови.

Типи баз даних

Існує багато різних типів баз даних. Найкраща база даних для конкретної організації залежить від того, як організація збирається використовувати дані.

- **Реляційні бази даних**

  Реляційні бази даних стали домінуючими у 1980-х роках. Елементи в реляційній базі даних організовані як набір таблиць зі стовпцями та рядками. Технологія реляційної бази даних забезпечує найбільш ефективний і гнучкий спосіб доступу до структурованої інформації.

- **Об'єктно-орієнтовані бази даних**

  Інформація в об'єктно-орієнтованій базі даних представлена у вигляді об'єктів, як в об'єктно-орієнтованому програмуванні.

- **Розподілені бази даних**

  Розподілена база даних складається з двох або більше файлів, розташованих на різних сайтах. База даних може зберігатися на кількох комп’ютерах, розташованих в одному фізичному місці або розкиданих по різних мережах.

- **Сховища даних**

  Центральне сховище даних, сховище даних — це тип бази даних, спеціально розроблений для швидкого запиту та аналізу.

- **Бази даних NoSQL**

  NoSQL, або нереляційна база даних, дозволяє зберігати та маніпулювати неструктурованими та напівструктурованими даними (на відміну від реляційної бази даних, яка визначає, як мають бути складені всі дані, вставлені в базу даних). Бази даних NoSQL ставали популярними, оскільки веб-додатки ставали все більш поширеними та складнішими.

- **Графові бази даних**

  База даних графів зберігає дані в термінах сутностей і зв'язків між сутностями.

  Бази даних OLTP. База даних OLTP — це швидка аналітична база даних, призначена для великої кількості транзакцій, які виконуються кількома користувачами.

Це лише деякі з кількох десятків типів баз даних, які використовуються сьогодні. Інші, менш поширені бази даних пристосовані до дуже конкретних наукових, фінансових або інших функцій. На додаток до різних типів баз даних, зміни в підходах до розробки технологій і драматичні досягнення, такі як хмара та автоматизація, рухають бази даних у абсолютно нових напрямках. Деякі з останніх баз даних включають.

Що таке програмне забезпечення баз даних?

Програмне забезпечення бази даних використовується для створення, редагування та підтримки файлів і записів бази даних, що полегшує створення файлів і записів, введення даних, редагування даних, оновлення та звітування. Програмне забезпечення також підтримує зберігання даних, резервне копіювання та звітування, керування множинним доступом і безпеку. Сильна безпека баз даних особливо важлива сьогодні, оскільки крадіжки даних стають

все більш частими. Програмне забезпечення для баз даних іноді також називають «системою керування базами даних» (СУБД).

Програмне забезпечення для баз даних спрощує керування даними, дозволяючи користувачам зберігати дані в структурованій формі та отримувати до них доступ. Зазвичай він має графічний інтерфейс, який допомагає створювати та керувати даними, а в деяких випадках користувачі можуть створювати власні бази даних за допомогою програмного забезпечення для баз даних.

Що таке база даних MySQL?

MySQL — це система керування реляційною базою даних з відкритим кодом, заснована на SQL. Він був розроблений і оптимізований для веб-додатків і може працювати на будь-якій платформі. Оскільки з Інтернетом з’явилися нові й інші вимоги, MySQL стала платформою вибору для веб-розробників і веб-додатків. Оскільки MySQL розроблено для обробки мільйонів запитів і тисяч транзакцій, це популярний вибір для компаній електронної комерції, яким потрібно керувати кількома грошовими переказами. Гнучкість на вимогу є основною особливістю MySQL.

MySQL — це СУБД, що стоїть за деякими з найкращих веб-сайтів і веб-додатків у світі, включаючи Airbnb, Uber, LinkedIn, Facebook, Twitter і YouTube.

Проблеми з базою даних

Сучасні великі корпоративні бази даних часто підтримують дуже складні запити, і очікується, що вони надаватимуть майже миттєві відповіді на ці запити. У результаті адміністратори баз даних постійно змушені застосовувати широкий спектр методів, щоб допомогти підвищити продуктивність. Деякі загальні проблеми, з якими вони стикаються, включають:

- Поглинання значного збільшення обсягу даних. Велика кількість даних, що надходять із датчиків, підключених машин і десятків інших джерел, змушує адміністраторів баз даних намагатися ефективно керувати та впорядковувати дані своїх компаній.
- Забезпечення безпеки даних. Сьогодні витоки даних відбуваються всюди, і хакери стають винахідливішими. Зараз як ніколи важливо забезпечити безпеку даних, а також легкий доступ для користувачів.
- Ідучи в ногу з попитом. У сучасному швидкозмінному бізнес-середовищі компаніям потрібен доступ до своїх даних у реальному часі, щоб підтримувати своєчасне прийняття рішень і використовувати нові можливості.
- Управління та підтримка бази даних та інфраструктури. Адміністратори бази даних повинні постійно стежити за базою даних на наявність проблем і виконувати профілактичне обслуговування, а також застосовувати оновлення програмного забезпечення та виправлення. Оскільки бази даних стають складнішими, а обсяги даних зростають, компанії стикаються з витратами на наймання додаткових спеціалістів для моніторингу та налаштування своїх баз даних.
- Зняття обмежень на масштабованість. Щоб вижити, бізнес повинен розвиватися, а разом з ним має розвиватися й управління даними. Але адміністраторам баз даних дуже важко передбачити, скільки потужностей знадобиться компанії, особливо з локальними базами даних.
- Забезпечення постійності даних, суверенітету даних або вимог до затримки. Деякі організації мають варіанти використання, які краще підходять для локальної роботи. У таких випадках ідеально підходять розроблені системи, які попередньо налаштовані та оптимізовані для роботи з базою даних. Згідно з нещодавнім аналізом Wikibon (PDF), клієнти досягають вищої доступності, більшої продуктивності та до 40% нижчої вартості з Oracle Exadata..

Висновок до розділу 3

Під час праці над розділом, було розглянуто сучасні напрямки у створенні архітектури програмних додатків, бази даних, архітектуру системи майбутнього додатку, а також було вибрано Java основною мовою написання застосунку.

РОЗДІЛ 4
ПРОТОТИП ЗАСТОСУНКУ

4.1. Створення бази даних

База даних конструюється за допомогою таблиць, які включають в себе дані та їх взаємозв'язки. Оскільки для управління цими взаємозв'язками буде використовуватися фреймворк Hibernate, кожній моделі (сутності) повинна відповідати відповідна таблиця в базі даних. Проаналізувавши предметну область, виокремлено наступні сутності:

Event (подія)
EventHall (зал, де проходитиме подія)
EventSession (запланована подія)
Order (куплені квитки)
Role (роль користувача)
ShoppingCart (кошик побажань)
Ticket (квиток)
User (користувач)

Створення таблиць бази даних: Крім Java-моделей, буде створено додаткові таблиці, які не відображаються у вигляді Java-сутностей. Ці таблиці використовуються як зв'язкові для взаємодії між сутностями. Опис кожної таблиці наведено нижче:

- **Event:** Зберігає дані про події, такі як назва (title), опис (description) та ідентифікатор (id). Ідентифікатор проставляється автоматично для кожного нового запису у таблиці.

- **EventHall:** Зберігає інформацію про зали, призначені для проведення подій. Містить дані про вмістимість залу (capacity), опис залу (description) та ідентифікатор (id), який автоматично проставляється.

- **EventSession:** Зберігає дані про заплановані та невдовзі відбудуться події. Містить інформацію про ID події (event_id), яка посилається на таблицю Event, та ID залу (eventHall_id), де ця подія планується (посилається на таблицю EventHall). Також містить колонки для вказання ціни (price) та часу проведення події (showTime). Має власний ідентифікатор (id), який автоматично проставляється.
- **Order_tickets:** Служить як зв'язкова таблиця для відношення між Order та Ticket. Тут зберігаються ID квитків (tickets_id), які були куплені в межах певного Order (Order_id).
- **Role:** Представляє таблицю для зберігання ролей. У поточній конфігурації зберігає два значення: ADMIN та USER. Містить колонки ідентифікатора (id) та назви ролі (roleName).
- **ShoppingCart:** Створена для присвоєння корзинки побажань для кожного користувача. Має лише колонку ідентифікатора користувача (user_id).
- **ShoppingCart_tickets:** Зберігає ідентифікатор користувача (ShoppingCart_user_id) та ідентифікатори квитків (tickets_id), які він бажає придбати.
- **Ticket:** Містить інформацію про квиток, таку як ідентифікатор події (eventSession_id) та ідентифікатор користувача (user_id), а також власний ідентифікатор (id).
- **User:** Зберігає дані про зареєстрованих користувачів, включаючи їхній email та пароль, який зберігається у хешованому вигляді для безпеки. Містить ідентифікатор користувача (id).
- **User_role:** Таблиця, яка містить інформацію про ролі всіх користувачів. Включає дві колонки: User_id (ідентифікатор користувача) та roles_id (ідентифікатор ролі), яку він має в системі.

Після визначення усіх таблиць та зв’язків між ними отримуємо схему, вигляд якої зображено на рисунку 4.1:

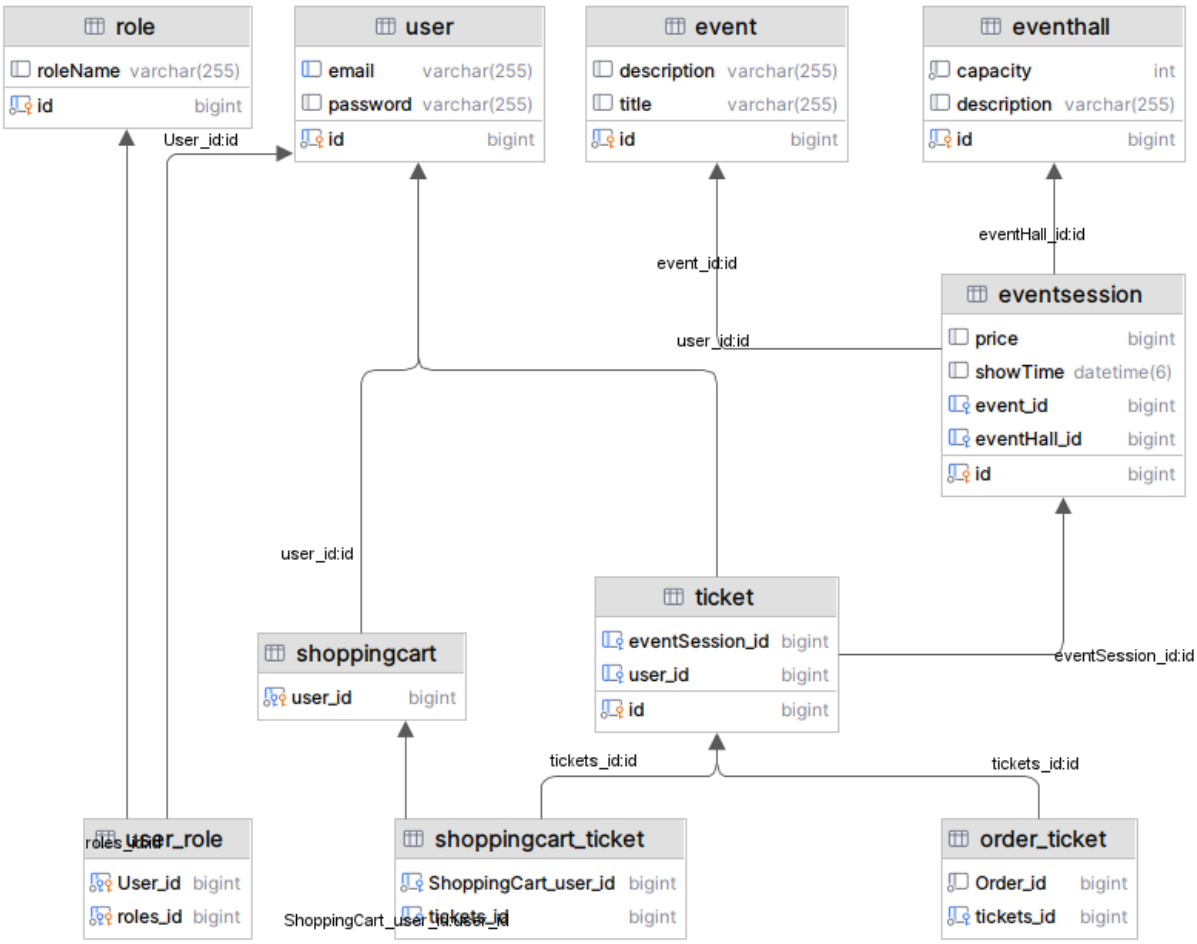

Рис. 4.1. ER - діаграма бази даних

Інші важливі аспекти:

- **Типи даних:** Використання типів даних varchar для рядкових даних, bigint для цілих числових значень та datetime для дати та часу є логічним та оптимальним вибором, забезпечуючи ефективне використання пам'яті та відповідність структурі даних.

- **Автоматичне створення та видалення таблиць:** Використання Hibernate і стратегії видалення-створення (create-drop) для автоматичного створення та видалення таблиць є зручним для розробки та тестування. Цей підхід дозволяє легко регенерувати базу даних під час змін у моделях та спрощує роботу з продуктом на етапі розробки.
- **Стандарти найменувань:** Всі назви таблиць відповідають загальноприйнятим стандартам, що покращує читабельність та зрозумілість структури бази даних.
- **Спрощення розробки:** Автоматичне створення таблиць і використання Hibernate спрощує розробку та тестування продукту, а також полегшує процес впровадження змін в базу даних.
- **Стратегії використання:** Вказано, що для подальшого використання можуть бути застосовані інші стратегії (наприклад, validate або create), що вказує на гнучкість системи при переході в режим експлуатації.

Враховуючи ці аспекти, розроблена база даних має оптимальну структуру для забезпечення ефективної та надійної роботи системи.

4.2. Створення ключових блоків коду

Пакет resources:

У директорії resources розміщено файл, який містить критичні конфігураційні дані для взаємодії з фреймворком Hibernate та базою даних. Основні елементи цього файлу:

1. **Посилання на схему бази даних:** Вказано ім'я файлу або URL, яке визначає структуру бази даних. Це важливо для коректного збереження та взаємодії з даними.

2. **Драйвер бази даних:** У випадку MySQL використовується драйвер **com.mysql.cj.jdbc.Driver**. Це необхідно для встановлення з'єднання між програмою та базою даних.

3. **Логін та пароль:** Вказані дані для автентифікації при доступі до бази даних. Це забезпечує безпеку взаємодії з базою даних.

4. **Конфігурація Hibernate:** Тут вказана стратегія видалення-створення (create-drop), яка означає, що Hibernate автоматично створює таблиці при запуску та видаляє їх при завершенні роботи. Це особливо зручно для розробки та тестування.

Загальна структура проекту має такий вигляд:

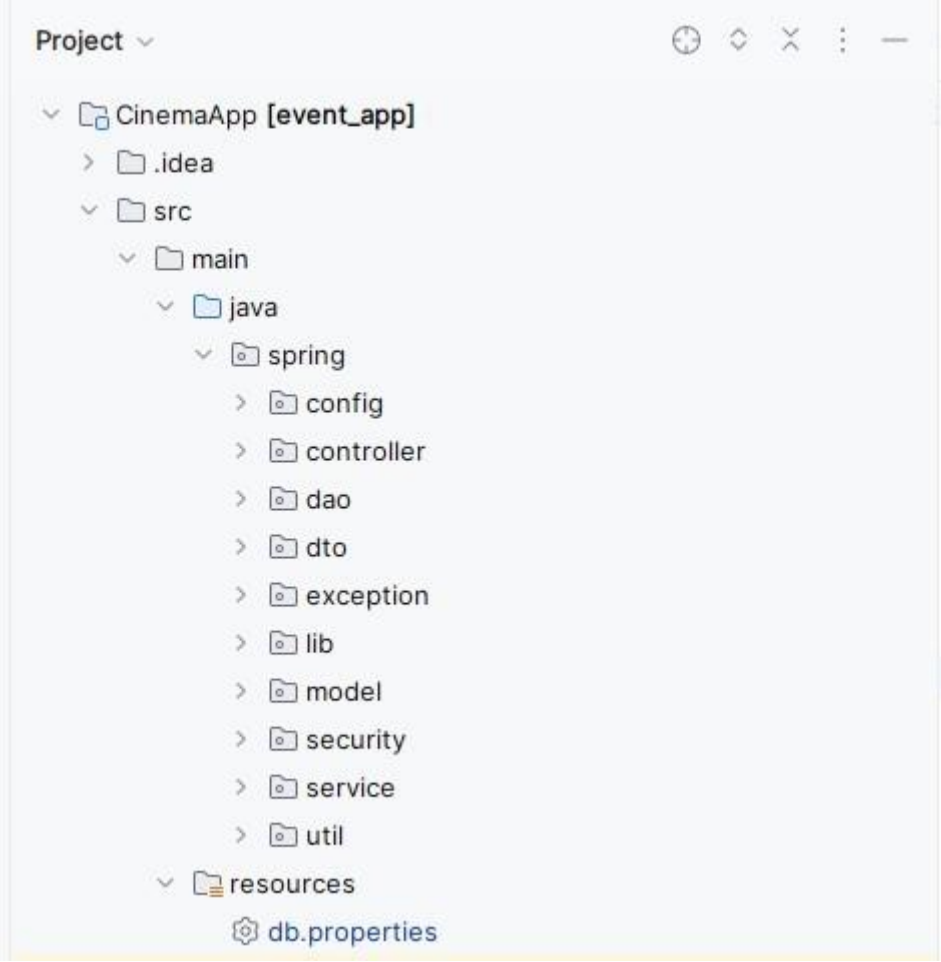

Рис. 4.2. Структура проєкту

Пакет config:

У пакеті config знаходяться ключові класи, які відповідають за конфігурацію фреймворку Spring. Ці класи можуть включати налаштування бінів, компонентів, аспектів, а також інших аспектів конфігурації. Наприклад:

1. **Конфігурація Spring Security:** Якщо у вас є клас, що розширює **WebSecurityConfigurerAdapter**, це може бути конфігурація для забезпечення безпеки вашого додатку.
2. **Конфігурація Spring Data JPA:** У разі використання JPA для роботи з базою даних, тут можуть бути налаштування для цього.
3. **Інші налаштування Spring:** Це може включати будь-які інші налаштування, пов'язані з контекстом додатка, обробкою помилок, локалізацією тощо.

Загалом, пакет config містить конфігураційні класи, які допомагають Spring фреймворку правильно налаштувати та ініціалізувати компоненти додатку.

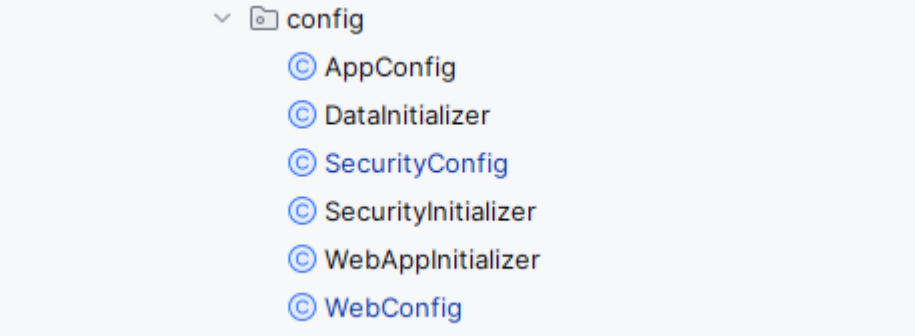

Рис. 4.3. Вміст пакету config

AppConfig:

AppConfig - це клас, де зосереджена основна конфігурація роботи Spring. Тут визначаються ключові компоненти, такі як:

1. **Отримання даних для створення зв’язку з базою даних:** Ймовірно, використовується **DataSource**, який вказує, як отримувати дані з бази даних.
2. **Створення SessionFactory: SessionFactory** є важливим об'єктом у Hibernate, який використовується для створення **Session**, яка дозволяє взаємодіяти з базою даних. Ймовірно, використовується конфігурація Hibernate для цього.
3. **Інші ключові компоненти:** Може включати інші налаштування та біни, які будуть використовуватися в додатку.

DataInitializer:

DataInitializer - опціональний клас, який може використовуватися для початкового заповнення бази даних або налаштування початкових умов для тестування. У вашому випадку він створює базових користувачів (ADMIN та USER). Такий підхід є зручним для тестування без необхідності додавання користувачів вручну.

SecurityConfig та SecurityInitializer:

SecurityConfig та **SecurityInitializer** відповідають за забезпечення безпеки вашого додатку, зокрема за захист запитів від несанкціонованого доступу.

Основні елементи:

1. **Настройка безпеки:** Метод **configure()** містить налаштування Spring Security. Ви вказуєте, хто має доступ до конкретних ресурсів за допомогою ролей.
2. **Захист ресурсів:** Це важливо для того, щоб обмежити доступ до деяких функціональностей, наприклад, щоб заборонити звичайному користувачеві виконувати адміністративні завдання.

Ці класи використовуються для забезпечення захищеності додатку та контролю доступу до різних функцій в залежності від ролі користувача.

У моєму випадку це метод має такий вигляд:

```
protected void configure(HttpSecurity http) throws Exception {
    http
            .authorizeRequests() ExpressionInterceptUrlRegistry
            .antMatchers(HttpMethod.GET,
                    "/event-halls",
                    "/events",
                    "/event-sessions/available",
                    "/event-sessions/{id}").hasAnyRole( ...roles: "USER", "ADMIN")
            .antMatchers(HttpMethod.GET, "/users/by-email").hasRole("ADMIN")
            .antMatchers(HttpMethod.GET,
                    "/orders",
                    "/shopping-carts/by-user").hasRole("USER")
            .antMatchers(HttpMethod.POST, "/register").permitAll()
            .antMatchers(HttpMethod.POST,
                    "/event-halls",
                    "/events",
                    "/event-sessions").hasRole("ADMIN")
            .antMatchers(HttpMethod.POST, "/orders/complete").hasRole("USER")
            .antMatchers(HttpMethod.PUT, "/event-sessions/{id}").hasRole("ADMIN")
            .antMatchers(HttpMethod.PUT, "/shopping-carts/event-sessions").hasRole("USER")
            .antMatchers(HttpMethod.DELETE, "/event-sessions/{id}").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and() HttpSecurity
            .formLogin().permitAll() FormLoginConfigurer<HttpSecurity>
            .and() HttpSecurity
            .httpBasic() HttpBasicConfigurer<HttpSecurity>
            .and() HttpSecurity
            .csrf().disable();
}
```

Рис. 4.4. Відображення прав доступу для кожної ролі в системі.

WebAppInitializer та WebConfig - це технічні компоненти, які забезпечують належну функціональність фреймворку Spring з контролерами. Наступною областю для розгляду є пакет Model, який становить основу для реалізації шаблону MVC. У цьому пакеті розташовані класи-моделі, що виступають прототипами таблиць у базі даних.

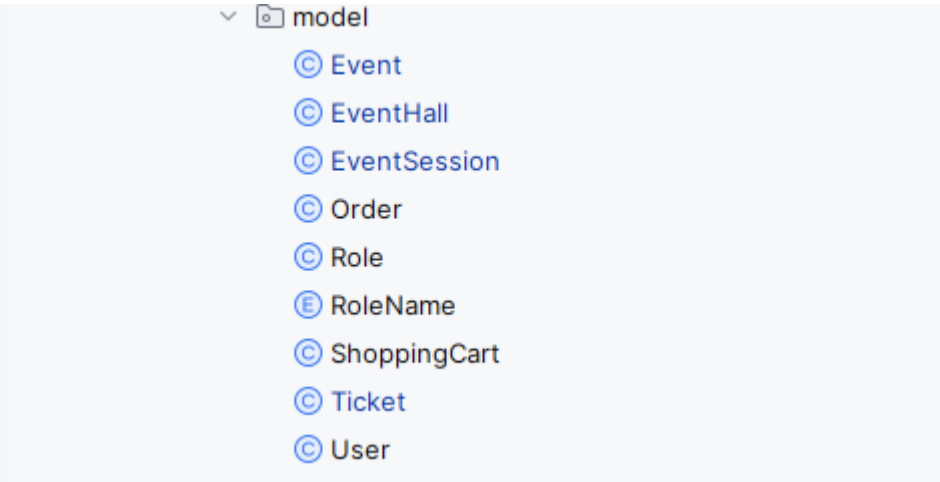

Рис. 4.5. Вміст пакету model

Оскільки всі класи цього пакету мають аналогічну структуру, розглянемо в деталях лише клас Event. Цей клас представляє подію і включає в себе назву, опис та ідентифікатор.

```
@Entity
public class Event  {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    6 usages
    private String title;
    6 usages
    private String description;
```

Рис. 4.6. Клас Event

1. Класи пакету Model служать для визначення моделей, які використовуються для створення таблиць у базі даних. У кожному класі використовується анотація @Entity, що вказує на його роль у процесі створення таблиці.

2. Використання анотацій в Java, запроваджених у версії 5.0, дозволяє надавати багато контексту в декларації класу, що спрощує конфігурацію Spring Framework.
3. Пакет Dao містить класи, які відповідають за внесення та отримання даних з бази даних. Основний функціонал для роботи з таблицями надає клас AbstractDao.
4. Пакет Service містить бізнес-логіку проєкту і діє як зв'язковий шар між Dao та контролерами. Кожен клас сервісу відповідає за взаємодію з конкретною моделлю.
5. Пакет Dto містить класи-обгортки для моделей, що використовуються у RESTful застосунках. Це забезпечує передачу тільки необхідних даних під час обміну інформацією між користувачем і системою.

Такий підхід до використання анотацій та розподілу функціоналу за окремі пакети сприяє ефективному розробленню та управлінню проєктом.

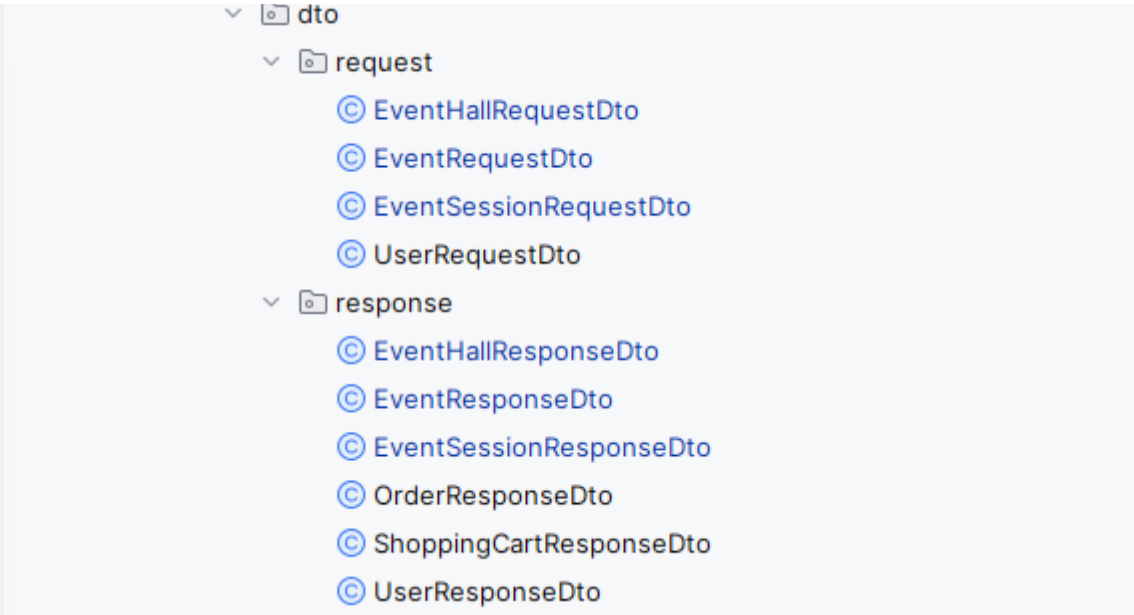

Рис.4.7. Вміст пакету dto

Отже, класи з пакету mapper містять методи, які використовуються для перетворення об'єктів моделі в об'єкти Dto і навпаки. Деякі з цих методів продемонстровані в додатку А.

Наступним є пакет Controller, який представляє наступний рівень у шаблоні MVC - рівень контролерів. Тут розташовані класи, які безпосередньо взаємодіють з браузером, обмінюючись запитами (request - запит, який браузер (або інший view) відправляє контролеру, та response - відповідь контролера, зазвичай у форматі JSON). Цей пакет є ключовою частиною реалізації RESTful підходу. Завдяки цим запитам (REST Api`s) можлива інтеграція з різними проєктами. Все, що потрібно для проєктів, - це надавати один одному запити та мати знання щодо очікуваних відповідей. Таким чином, розроблюваний продукт може легко інтегруватися з іншими проєктами в залежності від їхніх потреб.
Класи цього пакету мають схожу структуру. Розглянемо приклад EventController, де реалізовані запити для управління подіями.

```
@RestController
@RequestMapping("/events")
public class EventController {
    3 usages
    private final EventService eventService;
    4 usages
    private final EventMapper eventMapper;

    new *
    public EventController(EventService eventService, EventMapper eventMapper) {...}

    new *
    @PostMapping
    public EventResponseDto add(@RequestBody @Valid EventRequestDto requestDto) {...}


    @GetMapping
    public List<EventResponseDto> getAll() {...}
}
```

Рис.4.8. Структура класу EventController

Основними компонентами в цьому контексті є анотації. Наприклад, анотація RestController служить показником того, що клас є контролером. Анотація RequestMapping визначає назву запиту та його виклик. Анотації PostMapping та GetMapping вказують на типи запитів (POST та GET), а також визначають параметри, які вони приймають, і типи даних, які повертаються браузеру. Запит GET використовується для отримання певної інформації, тоді як

запит POST використовується для додавання інформації в базу даних. Більш детальні приклади контролерів можна знайти в додатку В.

Щодо інших пакетів, які не були розглянуті, вони є службовими. Серед них пакети Exception, Lib, Security, Util. Коротко про кожен з них:

- **Exception**: містить клас із стилізованою помилкою, яка буде викинута у випадку введення невірних даних.
- **Lib**: містить класи, які використовуються для перевірки вхідних даних, включаючи перевірку електронної пошти та інших. Це запобігає введенню некоректних даних в систему.
- **Util**: містить клас, де розташовані патерни дати та часу, які використовуються в системі. Цей клас вкрай важливий для уніфікації інформації про час, оскільки невідповідності в уніфікації можуть викликати неочікувані проблеми у роботі системи.

Оскільки у цьому дослідженні була розроблена лише back-end частина, тестування та демонстрація функціоналу здійснюються за допомогою програми Postman. Її функціонал створений для тестування сторонніх API і ідеально підходить для розробленого проєкту.

Висновок до розділу 4

У цьому розділі було висвітлено прототип розроблюваного додатку. Було розкрито та проілюстровано ключові компоненти, фрагменти коду та загальний стиль розробки. Також було пояснено, чому створений додаток відповідає принципам RESTful та вважається актуальним і зручним для інтеграції з іншими системами. Такий підхід є одним з найпоширеніших в сучасності і надає можливість легко масштабувати та розширювати систему.

ВИСНОВКИ

У ході дослідження комерційної діяльності кінотеатрів було проведено аналіз основних програмних засобів, необхідних для створення застосунку. При цьому було надано характеристику цих засобів, враховуючи їхню релевантність та потенціал у сучасному розробленні програмного забезпечення.

Основною метою було розроблення RESTful застосунку з використанням сучасних технологій та принципів розробки для ефективної комунікації між компонентами системи та забезпечення гнучкості та масштабованості. Впровадження RESTful архітектури дало можливість створювати легкодоступні та стандартизовані інтерфейси для взаємодії з іншими системами.

Аналіз показав, що використання сучасних технологій та принципів розробки, таких як контейнеризація, мікросервісна архітектура та автоматизація процесів розгортання, значно підвищило ефективність розробки та підтримки застосунку.

Отже, результати дослідження підкреслюють важливість використання сучасних технологій та принципів розробки при створенні програмного забезпечення, зокрема RESTful застосунків. Це сприяє більшій функціональності, гнучкості та ефективності системи, а також поліпшує взаємодію з іншими системами та користувацький досвід.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Планета Кіно в Києві [Інтернет-ресурс] / Режим доступу: https://planetakino.ua/
2. Кінотеатр прем'єра [Інтернет-ресурс] / Режим доступу: https://premjera.com.ua/
3. Wezom [Інтернет-ресурс]. Режим доступу: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya Scientific
4. Вікіпедія [Інтернет-ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Java
5. Amazon [Інтернет-ресурс]. Режим доступу: https://aws.amazon.com/what-is/restful-api
6. Functional and Nonfunctional Requirements: Specification and Types [Інтернет-ресурс]. Режим доступу: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/
7. What is database? [Інтернет-ресурс]. Режим доступу: https://www.oracle.com/database/what-is-database/

Додаток А

```
package spring.config;

import java.util.Properties;
import javax.sql.DataSource;
import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@PropertySource("classpath:db.properties")
@ComponentScan(basePackages = {
        "spring"
})
public class AppConfig {
   private final Environment environment;

   public AppConfig(Environment environment) {
      this.environment = environment;
   }

   @Bean
   public DataSource getDataSource() {
      BasicDataSource dataSource = new BasicDataSource();
      dataSource.setDriverClassName(environment.getProperty("db.driver"));
      dataSource.setUrl(environment.getProperty("db.url"));
      dataSource.setUsername(environment.getProperty("db.user"));
      dataSource.setPassword(environment.getProperty("db.password"));
      return dataSource;
   }

   @Bean
   public LocalSessionFactoryBean getSessionFactory() {
      LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();
      factoryBean.setDataSource(getDataSource());
```

```
        Properties properties = new Properties();
        properties.put("hibernate.show_sql", environment.getProperty("hibernate.show_sql"));
        properties.put("hibernate.dialect", environment.getProperty("hibernate.dialect"));
        properties.put("hibernate.hbm2ddl.auto", environment.getProperty("hibernate.hbm2ddl.auto"));

        factoryBean.setHibernateProperties(properties);
        factoryBean.setPackagesToScan("spring.model");
        return factoryBean;
    }

    @Bean
    public PasswordEncoder getEncoder() {
        return new BCryptPasswordEncoder();
    }
}
package spring.config;

import java.util.Set;
import javax.annotation.PostConstruct;
import org.springframework.stereotype.Component;
import spring.model.Role;
import spring.model.RoleName;
import spring.model.User;
import spring.service.RoleService;
import spring.service.UserService;

@Component
public class DataInitializer {
    private final RoleService roleService;
    private final UserService userService;

    public DataInitializer(RoleService roleService, UserService userService) {
        this.roleService = roleService;
        this.userService = userService;
    }

    @PostConstruct
    public void inject() {
        Role adminRole = new Role();
        adminRole.setRoleName(RoleName.ADMIN);
        roleService.add(adminRole);
```

```
        Role userRole = new Role();
        userRole.setRoleName(RoleName.USER);
        roleService.add(userRole);
        User firstUser = new User();
        firstUser.setEmail("anna@gmail.com");
        firstUser.setPassword("12345678");
        firstUser.setRoles(Set.of(adminRole));
        userService.add(firstUser);
        User secondUser = new User();
        secondUser.setEmail("bohdan@gmail.com");
        secondUser.setPassword("87654321");
        secondUser.setRoles(Set.of(userRole));
        userService.add(secondUser);
    }
}
package spring.config;

import org.springframework.http.HttpMethod;
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.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    public SecurityConfig(UserDetailsService userDetailsService, PasswordEncoder passwordEncoder) {
        this.userDetailsService = userDetailsService;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
    }

    protected void configure(HttpSecurity http) throws Exception {
        http
```

```
                .authorizeRequests()
                .antMatchers(HttpMethod.GET,
                        "/cinema-halls",
                        "/movies",
                        "/movie-sessions/available",
                        "/movie-sessions/{id}").hasAnyRole("USER", "ADMIN")
                .antMatchers(HttpMethod.GET, "/users/by-email").hasRole("ADMIN")
                .antMatchers(HttpMethod.GET,
                        "/orders",
                        "/shopping-carts/by-user").hasRole("USER")
                .antMatchers(HttpMethod.POST, "/register").permitAll()
                .antMatchers(HttpMethod.POST,
                        "/cinema-halls",
                        "/movies",
                        "/movie-sessions").hasRole("ADMIN")
                .antMatchers(HttpMethod.POST, "/orders/complete").hasRole("USER")
                .antMatchers(HttpMethod.PUT, "/movie-sessions/{id}").hasRole("ADMIN")
                .antMatchers(HttpMethod.PUT, "/shopping-carts/movie-sessions").hasRole("USER")
                .antMatchers(HttpMethod.DELETE, "/movie-sessions/{id}").hasRole("ADMIN")
                .anyRequest().authenticated()
                .and()
                .formLogin().permitAll()
                .and()
                .httpBasic()
                .and()
                .csrf().disable();
    }
}
package spring.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = {"spring.controller"})
public class WebConfig implements WebMvcConfigurer {
}
```

Додаток Б

```
package spring.dao;

import java.util.List;
import java.util.Optional;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import spring.exception.DataProcessingException;

public abstract class AbstractDao<T> {
    protected final SessionFactory factory;
    private final Class<T> clazz;

    public AbstractDao(SessionFactory factory, Class<T> clazz) {
        this.factory = factory;
        this.clazz = clazz;
    }

    public T add(T t) {
        Transaction transaction = null;
        Session session = null;
        try {
            session = factory.openSession();
            transaction = session.beginTransaction();
            session.save(t);
            transaction.commit();
            return t;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            throw new DataProcessingException("Can't insert "
                    + clazz.getSimpleName() + " " + t, e);
        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
```

```
public Optional<T> get(Long id) {
    try (Session session = factory.openSession()) {
        return Optional.ofNullable(session.get(clazz, id));
    } catch (Exception e) {
        throw new DataProcessingException("Can't get "
                + clazz.getSimpleName() + ", id: " + id, e);
    }
}

public List<T> getAll() {
    try (Session session = factory.openSession()) {
        return session.createQuery("from " + clazz.getSimpleName(), clazz).getResultList();
    } catch (Exception e) {
        throw new DataProcessingException("Can't get all "
                + clazz.getSimpleName() + "s from db", e);
    }
}

public T update(T t) {
    Transaction transaction = null;
    Session session = null;
    try {
        session = factory.openSession();
        transaction = session.beginTransaction();
        session.update(t);
        transaction.commit();
        return t;
    } catch (Exception e) {
        if (transaction != null) {
            transaction.rollback();
        }
        throw new DataProcessingException("Can't update "
                + clazz.getSimpleName() + " " + t, e);
    } finally {
        if (session != null) {
            session.close();
        }
    }
}

public void delete(Long id) {
    Transaction transaction = null;
```

```
        Session session = null;
        try {
            session = factory.openSession();
            transaction = session.beginTransaction();
            T movieSession = session.get(clazz, id);
            session.delete(movieSession);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            throw new DataProcessingException("Can't delete "
                    + clazz.getSimpleName() + " with id: " + id, e);
        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}
```

```
package spring.dao;

import java.util.List;
import java.util.Optional;
import spring.model.CinemaHall;

public interface CinemaHallDao {
    CinemaHall add(CinemaHall cinemaHall);

    Optional<CinemaHall> get(Long id);

    List<CinemaHall> getAll();
}
```