КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)
«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Веб-сервіс із підтримки діяльності кінотеатру»

Виконав ___ _____
(Підпис)
Хасанов Богдан Маратович
(прізвище, ім'я, по батькові)

Унікальність тексту 78,3 %
Автор _____ Хасанов Б.М.
(Підпис) (Прізвище, ініціали)

Керівник к.т.н., доцент Бойко Юлія Петрівна
(прізвище, ім'я, по батькові)
_ _____
(Резолюція «До захисту»)

Попередній захист:

_____***23.05.2022***_____
_____
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри __ _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали)
(Дата)

Київ – 2022

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «Веб-сервіс із підтримки діяльності кінотеатру»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Хасанов Богдан Маратович	

Назва роботи українською та англійською мовами:

Веб-сервіс із підтримки діяльності кінотеатру
Web-service for supporting cinema activities

Мета бакалаврської роботи: Підвищення ефективності діяльності кінотеатру шляхом розробки та впровадження сучасного веб-сервісу
План роботи: 1. Сучасні підходи для розроблення і впровадження веб-сервісів 2. Аналіз архітектурних рішень і вибір технологій для реалізації веб-систем 3. Програмна реалізація веб-сервісу із підтримки діяльності кінотеатру

ПІБ, ступінь, звання наукового керівника роботи:

К.т.н., доцент Бойко Ю.П.

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Номер	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	виконано
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	виконано
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2022	виконано
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2022	виконано
9.	Подання роботи у першому варіанті	28.04.2022	виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**23.05.2022**	виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	
14.	Захист кваліфікаційної роботи бакалавра	23.06.2022	

Здобувач вищої освіти _____ ___

(підпис)

Керівник _____

(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	21
2	12
3	11
Висновки	2
Перелік використаних джерел	3
Додатки	26

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Хасанов Б.М.			Відомість дипломної роботи	Лист	Листів
Керівн.	Бойко Ю.П.					
Н/контр.	Базилюк А.М.					
Зав.каф.	Плескач В.Л.					

АНОТАЦІЯ

Дипломна робота: 84 с., 23 рис., 25 джерел, 2 дод.

Ця дипломна робота присвячена проектуванню та розробленню веб-сервісу із підтримки діяльності кінотеатру.

Метою дипломної роботи є підвищення ефективності діяльності кінотеатру шляхом розробки веб-сервісу.

Для досягнення поставленої мети треба вирішити такі **завдання**:

- дослідити теоретичні аспекти побудови веб-сайту, веб-сервісу та їх взаємодії у мікросервісній архітектурі;
- дослідити архітектуру побудови веб-системи;
- розробити веб-сервіс із підтримки діяльності кінотеатру;

Об’єкт дослідження.

Процеси автоматизації організаційної діяльності кінотеатру.

Предмет дослідження.

Засоби та принципи розробки веб-застосунків для забезпечення ефективної діяльності кінотеатру.

Методи дослідження.

Методи теоретичного рівня: абстрагування, ідеалізація, формалізація, аналіз і синтез, аксіоматика, узагальнення – використовуються для проведення логічного дослідження наявної інформації щодо веб-розробки, вироблення понять, думок, виведення висновків.

Методи емпіричного рівня: спостереження, порівняння, розрахунок, тести – використовуються для формулювання результатів розробки; метод проб і помилок – у процесі розробки.

Ключові слова: веб-система, е-сервіс, мікросервісна архітектура, розподілені обчислення у хмарі, AWS DynamoDb.

ABSTRACT

Thesis: 84 pages, 23 figures, 25 sources, 2 appendix.

This thesis is devoted to the design and development of a web service to support the activities of the cinema.

The purpose of the thesis is to increase the efficiency of the cinema by developing a web service.

To achieve this goal, following tasks should be solved:

• explore the theoretical aspects of building a website, web service and their interaction in microservice architecture;

• explore the architecture of building a web system and develop a website to support the activities of the cinema;

• Conduct functional testing of the web site and develop installation documentation.

Object of study.

Processes of automation of organizational activity of cinema.

Subject of study.

Tools and principles for developing web applications to ensure the effective operation of the cinema.

Research methods.

Methods of theoretical level: abstraction, idealization, formalization, analysis, and synthesis, axiomatics, generalization - are used to conduct a logical study of available information on web development, development of concepts, opinions, conclusions.

Empirical level methods: observation, comparison, calculation, tests - are used to formulate the results of development; trial and error method - in the process of development.

Keywords: software system, e-commerce, handmade products ASP.NET technology.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ОС – операційна система

ДКД – дизайн керований доменом

SQL – структурна мова запитів

HATEOAS - гіпермедіа як рушій стану програми

ІТ – інформаційні технології

ВСТУП

За останні пару років фактично кожен бізнес був вимушений розглянути можливість переведення діяльності частково або повністю в онлайн формат. Розробка власного веб-сайту є найефективнішим шляхом покращити якість взаємодії з клієнтами. Залучення нових клієнтів також найкраще досягається шляхом просування веб-сайт, адже користувач може отримати інформацію будь де і будь коли. Але з розвитком технологій, надання інформації вже не є ключовою особливістю веб-сайту. Саме надання електронних послуг призводить до ключових змін у якості взаємовідносин між надавачем послуг та споживачем.

В наш час кінотеатр може взаємодіяти з клієнтами багатьма шляхами: оффлайн-спілкування, телефонний дзвінок, листування електронною поштою, чат-бот у мессенджері, веб-сервіс, тощо.

З точки зору зручності, безперебійності та кількості можливих послуг що можуть бути надані, саме веб-сервіс є найкращим варіантом. Надання можливості клієнту замовити квиток на сесію кінотеатру будь-коли та при будь яких обставинах буде відігравати ключову ролі в якості організації дозвілля

Актуальність дослідження. Переведення діяльності в онлайн формат є ключовим питанням для переважної більшості організації. Але для кінотеатру зараз немає зручнішого способу організувати взаємодію ніж веб-сервіс. Можливість створити унікальне візуальне оформлення сайту допоможе привабити клієнта. Це рішення також є актуальним бо дозволить зменшити кількість найманих робітників що будуть залучені до організаційних процесів

Метою кваліфікаційної роботи бакалавра є переведення організаційної діяльності кінотеатру у веб-формат за допомогою веб-сервісу із підтримки діяльності кінотеатру.

Для досягнення поставленої мети треба вирішити такі **завдання**:

- дослідити теоретичні аспекти побудови веб-сайту, веб-сервісу та їх взаємодії у мікросервісній архітектурі;
- дослідити архітектуру побудови веб-системи та розробити веб-сайт із підтримки діяльності кінотеатру;
- провести тестування роботи та розробити настановчу документацію.

Об’єктом дослідження є процес переведення організаційної діяльності кінотеатру у веб формат.

Предмет дослідження.

Сучасні технології розробки веб-систем з використанням розподіленої архітектури розгорнених на хмарній інфраструктурі.

Методи дослідження.

Методи теоретичного рівня: абстрагування, ідеалізація, формалізація, аналіз і синтез, аксіоматика, узагальнення – використовуються для проведення логічного дослідження наявної інформації щодо веб-розробки, вироблення понять, думок, виведення висновків.

Методи емпіричного рівня: спостереження, порівняння, розрахунок, тести – використовуються для формулювання результатів розробки; метод проб і помилок – у процесі розробки.

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. ТЕОРЕТИЧНЕ ПІДҐРУНТЯ ДЛЯ РОЗРОБКИ ТА ІМПЛЕМЕНТАЦІЇ ПРОГРАМНОЇ СИСТЕМИ ПІДТРИМКИ ДІЯЛЬНОСТІ КІНОТЕАТРУ

1.1 Аналіз роботи веб-сервісу у взаємодії з прикладними системами

Для розробки програмної системи потрібно розглянути принципи побудови веб-сервісу, що є складовою клієнт-серверної архітектури.

Веб-сервіс - це будь-яка частина програмного забезпечення, яка робить себе доступною через Інтернет і використовує стандартизовану систему обміну повідомленнями XML. XML використовується для кодування всіх повідомлень веб-служби. Наприклад, клієнт викликає веб-сервіс, відправивши повідомлення XML, а потім чекає на відповідну XML-відповідь. Оскільки всі комунікації в XML, веб-служби не прив'язані до жодної операційної системи або мови програмування, Java може спілкуватися з Perl; Програми Windows можуть спілкуватися з додатками Unix.

Веб-сервіси - автономні, модульні, розподілені, динамічні програми, які можна описати, публікувати, знаходити або викликати через мережу для створення продуктів, процесів та ланцюгів постачання. Ці програми можуть бути локальними, розподіленими або веб-based. Веб-сервіси побудовані на основі відкритих стандартів, таких як TCP / IP, HTTP, HTML та XML.

Веб-сервіси - це системи обміну інформацією на базі XML, які використовують Інтернет для взаємодії прямого застосування до програми. Ці системи можуть включати програми, об'єкти, повідомлення або документи.

Веб-сервіс - це набір відкритих протоколів та стандартів, які використовуються для обміну даними між додатками або системами. Програмні додатки, написані на різних мовах програмування та запущені на різних платформах, можуть використовувати веб-сервіси для обміну даними через комп'ютерні мережі, такі як Інтернет, у спосіб, подібний до

міжпроцесного зв'язку на одному комп'ютері. Ця взаємодія (наприклад, між Java і Python або програмами Windows та Linux) пов'язана з використанням відкритих стандартів

1.1.1 Мета веб-сервісу

- Доступний через Інтернет або приватні (інтрамережі) мережі
- Використовує стандартизовану систему обміну повідомленнями XML
- Не прив'язаний до жодної операційної системи або мови програмування
- Самоописується через загальну граматику XML
- Виявляється за допомогою простого механізму пошуку

1.1.2 Сценарії застосування веб-сервісів

Веб-сервіси як реалізація логіки програми (бізнес-логіки). Тобто, створення нової прикладної програми бізнес-логіка, яка реалізується у веб-сервісі. На рисунку 1.1 зображено сценарій застосування веб-сервісу де клієнт звільнений від бізнес-логіки.

Рисунок 1.1. – Веб-сервіс що реалізує бізнес логіку

Веб-сервіси як засіб інтеграції. Тобто, використання веб-сервісу як способу доступу віддалених клієнтів до внутрішньої ІС компанії, або для організації взаємодії компонента (наприклад, EJB, COM-компонента) з різними віддаленими клієнтами. На рисунку 1.2 зображено схему роботи веб-сервісу що виконує функції проксювання інтерфейсу декількох компонентів

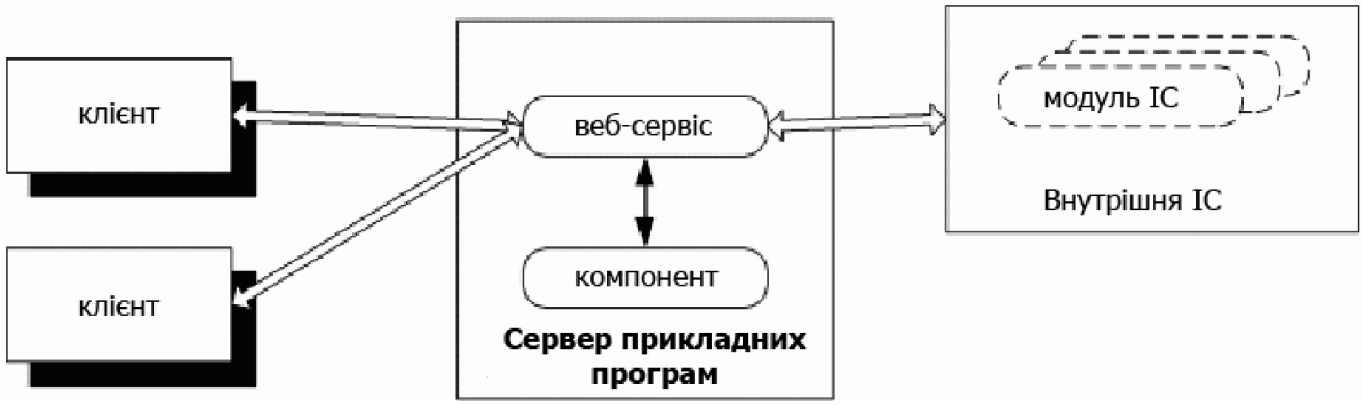


Рисунок 1.2 – Веб-сервіс проксі для інтеграції декількох клієнтів

● Використання веб-сервісу як будівельного блоку при створенні програми. Програма може використовувати веб-сервіси як вилучені компоненти, які надають певну функціональність. Існують різні сервіси, які надають якісне рішення таких задач як аутентифікація, ведення календаря, відправлення повідомлень, пошук, переклад і т. п. На рисунку 1.3 зображено спосіб побудови веб-системи де програма застосовує композитний шаблон проектування.

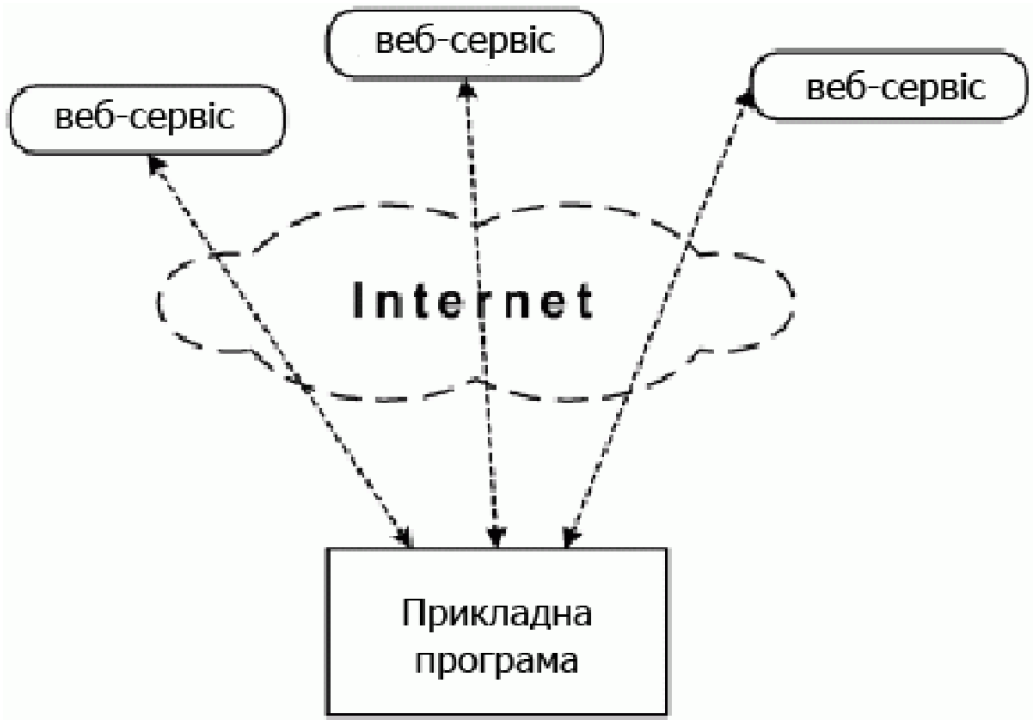


Рисунок 1.3 – Функціональний веб-сервіс для забезпечення правильної роботи прикладної програми

Переваги використання веб-сервісів:

Викриття існуючої функціональності в мережі

Веб-сервіс є одиницею керованого коду, який може бути віддалено викликаний за допомогою HTTP[1], тобто його можна активувати за допомогою HTTP-запитів. Веб-сервіси дозволяють виявляти функціональність існуючого коду через мережу. Після виявлення в мережі інша програма може використовувати функціональність вашої програми.

Взаємодія

Веб-сервіси дозволяють різним програмам спілкуватися одна з одною та обмінюватися даними та службами між собою. Інші програми також можуть використовувати веб-служби. Наприклад, програма VB або .NET може спілкуватися з веб-службами Java і навпаки. Веб-сервіси використовуються для незалежної роботи платформи та технології.

Стандартизований протокол

Веб-сервіси використовують стандартний галузевий протокол для зв'язку. Всі чотири шари (сервісний транспорт, XML-повідомлення, опис сервісу та шари виявлення служб) використовують чітко визначені протоколи у стеку протоколів веб-сервісів. Ця стандартизація стек протоколу дає бізнесу безліч переваг, таких як широкий вибір варіантів, зменшення витрат за рахунок конкуренції та підвищення якості.

Низька вартість спілкування

Веб-сервіси використовують протокол SOAP за протоколом HTTP, тому ви можете використовувати свій існуючий недорогий інтернет для впровадження веб-служб. Це рішення набагато менш дорого, ніж у фірмових рішеннях, таких як EDI / B2B. Крім SOAP через HTTP, веб-сервіси також можуть бути реалізовані на інших надійних транспортних механізмах, таких як FTP.

1.1.3 Ролі веб-сервісу

Постачальник послуг

Це постачальник веб-сервісу. Постачальник послуг реалізує цю послугу та робить її доступною в Інтернеті.

Замовник послуги

Це будь-який споживач веб-сервісу. Замовник використовує наявну веб-службу, відкриваючи мережеве з'єднання та надсилаючи запит XML.

Служба реєстру

Це логічно централізований каталог послуг. Реєстр забезпечує центральне місце, де розробники можуть публікувати нові сервіси або знаходити існуючі. Таким чином, він є центральним кліринговим центром для компаній та їх послуг.

Загальна архітектура веб-сервісів зображена на рисунку 1.4

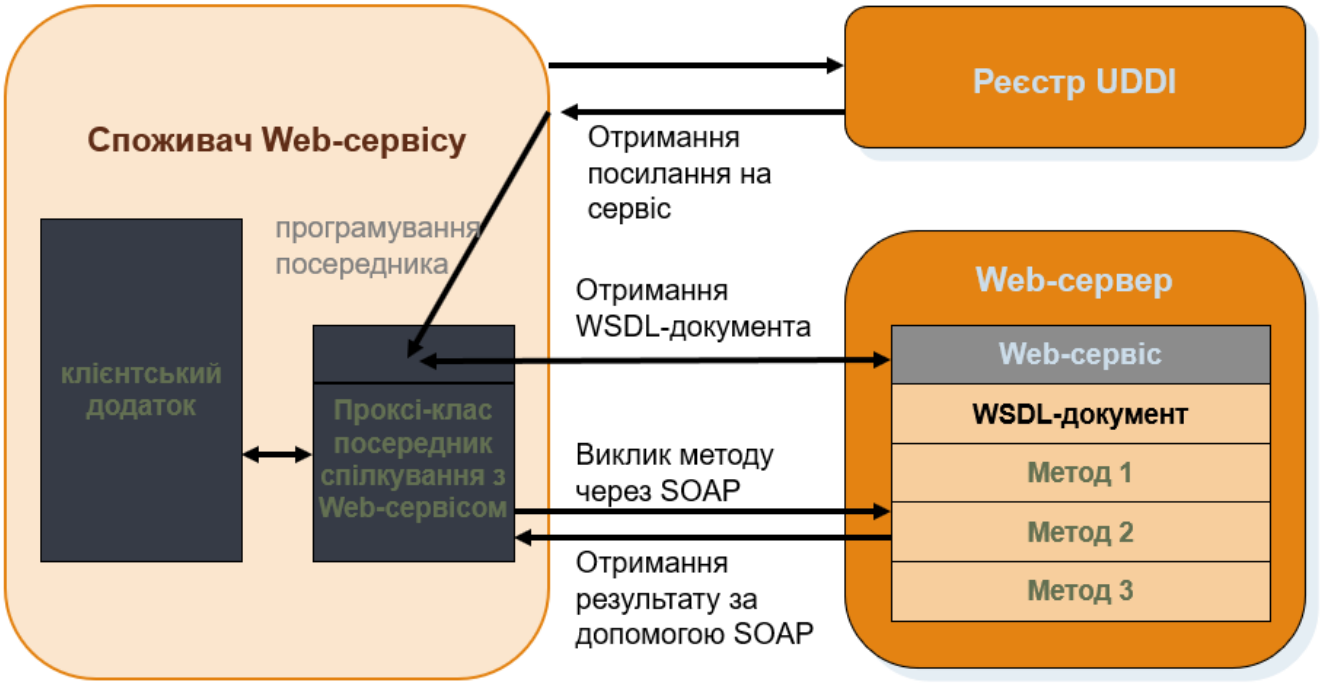


Рисунок 1.4 – Взаємодія споживача з веб-сервісом

1.2 Застосування принципів REST при розробці веб-сервісу

Передача / зміни стану через уявлення. REST - це архітектурний стиль, деяка безліч обмежень, для побудови розподілених додатків[7].

«Слова - Representational State Transfer повинні викликати в голові появу картинки роботи грамотного веб додатку: мережа з веб-сторінок (назвемо їх

станами), в якій користувач переміщається клікаючи по посиланнях (зміна станів) для переходу на наступну сторінку (що представляє собою чергове стан додатку) », - Рой Філдінг[8].

На рисунку 1.5 зображено стандартний підхід застосування принципу передачі стану через сутності.

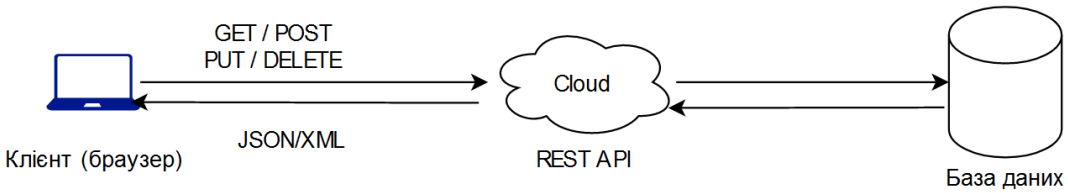


Рисунок 1.5 – Схема роботи клієнт-серверної архітектури за принципами REST

Обмеження REST

Архітектура, клієнт - сервер. Відокремлюємо логіку додатки від різних клієнтів, робимо їх код більш стерпним, а структуру сервера простіший і масштабованої. Розробка клієнтів і сервера може вестися абсолютно незалежно.

Stateless - сервер. Стан клієнта не зберігається на сервері, ні в якому вигляді, цим займається виключно сам клієнт. Це спрощує доопрацювання і супровід сервера, робить його більш стабільним.

Кешованість[6]. Повинна бути розроблена чітка система кешування запитів до сервера, що дозволяє значно поліпшити продуктивність.

Багатошарова структура. Наявність / відсутність проміжних серверів кешування, балансування навантаження, додаткового проксінг має. залишатися абсолютно непоміченим з боку клієнтів

Єдиний інтерфейс.

Ідентифікація ресурсів. Кожен ресурс має унікальний ідентифікатор URI (Uniform Resource Identifier - універсальний ідентифікатор ресурсу). наприклад:

/ Users / ae25b8.

Взаємодія з ресурсами через уявлення. Кожен ресурс має свою унікальну адресу URI (подання), і кілька дієслів для управління цим поданням.

Самодостатні повідомлення. Кожна відповідь має містити всю необхідну інформацію, щоб його можна було правильно обробити, не звертаючись до допомоги інших ресурсів. Наприклад який парсер викликати по MIME type

Код на вимогу. Опціональний елемент структури. Дозволяє отримувати програмний код для подальшого його виконання на клієнті.

Таким чином, якщо ваш додаток відповідає всім вимогам (обмеженням), описаним вище, він сміливо може називатися RESTful. Виняток становить лише код на вимогу - цей параметр опціональний.

1.2.1 HATEOAS

Суть HATEOAS полягає в підході до опису ресурсів нашого API. Замість простого перерахування набору ресурсів, зі списком усіх можливих операцій, які клієнт може викликати, керуючись певною внутрішньою логікою, ми проводимо інверсію контролю - тепер за стан ресурсу відповідає сервер і він диктує клієнту, які операції над ресурсом можна зробити в поточний момент. Ця інформація повинна бути присутнім в самому поданні ресурсу, який отримує клієнт. Таким чином, уявлення ресурсу саме себе описує в достатній мірі, щоб клієнт зрозумів, що з ним можна робити.

Застосування такого підходу зазвичай означає, що клієнт знає деякий кінцевий набір "точок входу" (можете вважати їх аналогами стартових сторінок на сайтах), з яких він починає свою взаємодію з API, використовуючи надану в поданні ресурсу інформацію для навігації до інших ресурсів і здійснення дій .

Для досягнення цього завдання якраз і використовуються гіперпосилання (hypermedia):

Всі ресурси адресованих за допомогою посилань, причому посилання на інші ресурси присутні всередині самих повідомлень для їх зв'язку між собою. Клієнт замість орієнтації на формат URI керується ідентифікаторами, за якими він вибирає посилання, розташовані прямо в поданні ресурсу. Якщо раніше ми вказували в документації що потрібно взяти деякий ID і на його основі побудувати спеціальний URL, тим самим роблячи набори URL частиною нашого API, то тепер деталі формування URL є просто особливостями реалізації сервера і клієнта не хвилюють. Зрештою, клієнту важливо отримати доступ до ресурсу, а не генерувати URL по шаблонах з документації.

Доступні операції над ресурсом теж представимо у вигляді посилань.

Відсутність посилання як на пов'язані ресурси, так і на доступні дії означає, що дана операція недоступна в поточному стані ресурсу.

1.2.2 Ресурс

Ресурс - унікальний об'єкт, доступний за унікальним URL.

Основні URL повинні бути зрозумілі без документації.

Для початку, бажано все URL адреси вашого API починати з префікса, наприклад,

“/Api/”

це допоможе спростити супровід API в майбутньому. Хорошим варіантом може виявитися префікс в імені домена:

“http://api.example.com/users/ae25b8”

Існує 2 основних типи ресурсу в архітектурі REST: колекція і елемент колекції.

Колекція представляє собою набір самостійних і самодостатніх елементів.

Приклад посилання на колекцію користувачів:

“/Api/users”

Елемент колекції користувачів, або конкретний користувач, в такому

випадку, може бути представлений у вигляді:

“/Api/users/ae25b8”

Іменники - добре, дієслова - погано.

Імена колекцій повинні представляти сутності (іменники у множині), і вони повинні бути максимально конкретними і зрозумілими (самодокументірующіміся). Якщо мова йде про собак, то це повинні бути собаки, а не просто тварини.

Кожним ресурсом в RESTful API управляє кілька певних мінімально необхідних дієслів. Для більшості випадків достатньо 4 основних дієслова HTTP методу:

- GET - отримати
- POST - створити
- PUT - змінити
- DELETE - видалити

Дієслова GET, PUT, DELETE - ідемпотентні.

1.3 Поняття серверної віртуалізації

Серверна віртуалізація – це алгоритм утворення та абстрагування певного числа віртуальних зразків запущених на виділеному сервері. Серверна віртуалізація у ході роботи змінює інформацію про ресурси серверу, також змінюються кількість та ідентифікатор кожного фізичного серверу, число обчислювальних процесорів та тип виконуваної ОС (який саме дистрибутив Linux використовується).

Традиційна конфігурація апаратного забезпечення та програмної архітектури у загальному випадку надавала підтримку єдиного програмного додатку. В результаті, цей факт дозволяв виділеному серверу проводити запуск єдиного об’єму роботи (набору програм), головним чином витрачаючи ресурси системи впусту – незадіяні процесори, фіксовану оперативну пам’ять (commit memory) та інші апаратні ресурси використовувані запущеними програмами.

Кількість апаратних серверів зростала пропорційно кількості запущених

програм та сервісів. В результаті такого підходу зростали потреби до обслуговування серверів. Зростала необхідна площа приміщень у зв'язку зі збільшенням кількості апаратних серверів. Витрати на охолодження, живлення та зв'язок серверів надмірно зростали.

Розробка підходу серверної віртуалізації усуває описані проблеми. Це веде до наступних змін:

- Додається шар програмного забезпечення що має назву гіпервізор (hypervisor). Його основною функцією є перетворення апаратних ресурсів в абстракції, що повинні використовуватись програмним забезпеченням
- Гіпервізор налаштовує взаємодію таким чином, щоб абстрактні (віртуалізовані) ресурси були згруповані в логічні одиниці названі віртуальними машинами. В результаті кожна віртуальна машина може виконувати функції виділеного серверу, що не використовує спільні ресурси.

Цей підхід віртуалізації дозволяє одному комп'ютеру виконувати ту саму кількість роботи що виконує декілька окремих комп'ютерів застосовуючи всі доступні обчислювальні ресурси для виконання декількох об'ємів роботи (набору виконуваних програм). Використання всіх доступних ресурсів збільшує ефективність їх витрати, таким чином показник коефіцієнту корисної дії обчислювальних ресурсів досягає свого максимуму. Зі зменшення кількості запущених серверів відпадає потреба постійно збільшувати робочу площу, кількість енергії. Витрати зменшуються за рахунок гнучкості налаштування та розгортки програмного забезпечення.

У підходу віртуалізації обчислень є недоліки які потрібно пом'якшувати. Серед яких:

- Ліцензування програмного продукту
- Складність керування
- Проблеми доступності

Еволюція підходу віртуалізації відбувалась в декілька етапів:

- У 1960-х роках комп'ютерні мейнфрейми розподілялись таким чином, щоб ресурси одного мейнфрейма були спрямовані для виконання

кількох об’ємів робіт (набору виконуваних даних). Причиною застосування саме такого підходу є те що у зв’язку з рекордною вартістю утримання одного мейнфрейму, компанія що його використовувала повинна була отримати максимум виконаної роботи в межах однієї одиниці. Так з’явилась віртуалізація в середовищі мейнфреймів

- У 1980-х роках було випущено стандарт набору процесорних інструкцій типу x86. В результаті з’явилась значно дешевша і ефективна архітектура обчислювального пристрою. Яка стала легкодоступною і простішою для застосування у порівнянні з мейнфреймною архітектурою. Власники програмних систем здійснили перехід на нову архітектуру і таким чином почали застосовуватись окремі виділені комп’ютерні системи, що могли незалежно розміщувати додатки, що використовувались відносно великою кількістю користувацьких або клієнтських комп’ютерів. Проте використання операційних систем на x86 архітектурі накладало обмеження на розмір підтримуваного додатку, у зв’язку з тим що комп’ютери були порівняно простими і мали дуже обмежені обчислювальні ресурси, пам’ять і об’єм даних що могли би бути збереженими. Такий перехід до малих за розміром комп’ютерів усунув потребу в віртуалізації та гіпервізорах що були реалізовані в мейнфреймах

- На початку 2000-х стались зміни в підходах до побудови корпоративних веб-систем, ці зміни були спричинені двома факторами:

1) Апаратне забезпечення комп’ютера зазнало стрімкого розвитку - показники швидкості разом з іншим показниками покращувались. Кінцева продукція знижувалась у ціні і ставала доступнішою. Звичайний процесор у корпоративній веб-системі мав декілька процесорів разом зі значно збільшеними обсягами оперативної пам’яті та жорстокого диску. Додатки що проводили обчислювальні операції не потребували такої великої потужності. В результаті матеріальні інвестиції мали низькі показники ефективності витрат на впровадження серверів. Тобто значні фінансові ресурси витрачались впусту по причині того що надмірні потужності апаратного забезпечення не

використовувались до кінця. Частим випадком було використання корпоративним сервером лише 15-20% своїх потужностей. Виникало питання розподілу ресурсів системи між більш ніж одним програмним застосунком.

2) Обмеження площі на якій могли бути розміщені сервери змушувало замислитись над доцільністю організації багатьох серверів в одному місці. З додаванням нових об'ємів робіт, зазвичай, просто відбувалось збільшення кількості серверів що були задіяні у роботі. Така тенденція могла спричинити загрозу перенавантаження фізичного простору, систем охолодження та забезпечення живлення. Тому рух у напрямку постійного збільшення кількості серверів мав би бути замінений альтернативою яка ефективніше застосовувала ресурси.

- Відновлення розвитку серверної віртуалізації відбулось у кінці 1990-х з першими продуктами які містили в собі реалізацію цього підходу. Але перше готове до використання рішення було випущене компанією VMWare. Саме з цього моменту корпоративні веб-системи почали свій перехід до серверної віртуалізації. Продукт був названий ESX 1.0 Server. Почали з'являтись і інші реалізації, а саме Windows Server 2008 with Hyper-V, Xen Project. Операційні системи надавали додаткам доступ до ресурсів так само - за допомогою гіпервізорів. Було досягнуто необхідний рівень стабільності системи. Випуск Docker'у в 2013 остаточно завершив перехід до віртуалізованих контейнерів. Даний продукт дозволив досягти високих показників маштабованості та породив новий вид архітектури додатку – мікросервісний[4].

Ті самі підходи що застосовувались 30 років назад до мейнфреймних систем наявні в сучасній серверній архітектурі. Так само гіпервізор розподіяє ресурси, такі як процесорні ядра, оперативна пам'ять сектори файлової системи жорсткого диску. В результаті один фізичний сервер з фізичними процесорами є перетворений в певну абстракцію. Ця логічна абстракція має лише частину ресурсів, розподілених в межах потреб. Додаток буде запущений в необхідній операційній системі із використанням вже

віртуальних процесорів (vCPU). Де достатня кількість ядр буде виділена гіпервізором. В результаті один фізичний сервер розміщує декілька запущених додатків що знаходяться кожен в своєму ізольованому середовищі.

Віртуалізація вирішила дві вкрай важливі проблеми описані вище. Проблема нестачі фізичної площі для розміщення серверів вирішується зменшенням кількості фізичних серверів[22]. Більше наборів програм може виконуватись на тій самій кількості фізичних серверів. Цей підхід має назву ущільнення серверів. Проблема забезпечення фізичних показників середовища у приміщенні де працюють сервери також вирішується. Зменшення кількості серверів дозволяє без великих труднощів налаштувати кондиціонування, подачу живлення та підтримку мережевого зв'язку. Наявні платформи віртуалізації надають додаткові можливості щодо управління, такі як:

- панель керування всіма запущеними віртуальними машинами
- стандартизований підхід щодо реорганізації віртуальних машин у процесі зміни фізичного серверу. Таким чином образи віртуальної машини можна переносити між фізичними серверами
- також шляхи збереження стану віртуальної машини у файлову систему і забезпечення цілісності збережених даних

Серверна віртуалізація працю шляхом видозміни наявного фізичного апаратного забезпечення у форму інтерфейсу. Кожен додаток запущений на цьому сервері буде отримувати доступ до ресурсів саме через цей інтерфейс. Цю роботу виконує окремий шар програмного забезпечення, названий гіпервізором. Кожна операційна система має свою власну реалізацію цього шару. Наявними рішення є Microsoft Hyper-V та інші. Під капотом гіпервізор визначає наявні фізичні ресурси: процесорні ядра, оперативну пам'ять, жорсткий диск та мережеві підключення, і створює програмний інтерфейс доступу до них. В результаті такого перетворення, програма звертається до vCPU замість звичайного фізичного ядра. Об'єми ресурсів визначаються згідно потреб додатку, але обмеження можуть бути налаштовані

користувачем. У випадку мережевих підключень, може створюватись аналог локальної мережі – але на рівні віртуальних машин. Це дозволяє додаткам обмінюватись пакетами з використанням протоколів:

- TCP (зі створенням з'єднання)
- UDP (без створення з'єднань)

Вся взаємодія програмного забезпечення з апаратним відбувається крізь гіпервізор.

Перевагою застосування віртуалізації є те що додатки можуть виконуватись на окремих операційних системах. Це може бути як дистрибутив Linux, так і Windows. І це все відбуватиметься на єдиному фізичному сервері.

Результатом створення цього інтерфейсу взаємодії є віртуальна машина. Для додатку не має різниці чи це фізична машина на якій запущена операційна система чи це віртуальна машина. Також запущена віртуальна машина ізольована таким чином, що відсутній шлях отримати інформацію про фізичний комп'ютер або інші логічні комп'ютери.

Основною функцією гіпервізора є не виділення ресурсів, а керування віртуальними комп'ютерами, забезпечення їх функціонування як цілісного комп'ютера. Таким чином додаток що підтримує єдину операційну систему може бути запущений у віртуальній машині разом з усіма необхідними бібліотеками. Гіпервізор також може надавати доступ до апаратного забезпечення з використанням окремих драйверів для кожної віртуальної машини. Таким чином фізична обчислювальна машина може розміщувати будь який набір додатків.

Надана гнучкість є основною перевагою віртуалізації. Для випадків коли потрібно зберегти налаштовану операційну систему для запуску у середовищі віртуальної машини разом з драйверами, бібліотеками та наборами для виконання є можливість зберегти всю віртуальну машину у виді образу. Цей образ можна перенести на іншу фізичну машину. Для цього потрібно перенести файл з файлової системи однієї фізичної машину на файлову систему іншої.

Але кількість віртуальних машин запущених на фізичному сервері є обмеженою апаратними ресурсам. Незважаючи на те що серверні процесори в даний час мають велике число віртуальних ядр, це досягається за допомогою технологій Hyperthreading від Intel та Simultaneous Multithreading від AMD, одне таке ядро може бути включеним в пул ресурсів лише однієї віртуальної машини. Хоча адміністратор може змінити обсяг виділеної пам’яті або віртуальних ядр, це обмеження продовжує діяти. Є спосіб обійти це обмеження, таким чином що декілька віртуальних машин мають доступ до ресурсів, але такий підхід вважається неефективним, бо частина процесорного часу буде виділена на те щоб розподіляти надмірно зайняті ресурси.

1.3.1 Переваги серверної віртуалізації

Перехід серверів до використання серверної віртуалізації вирішив тогочасні проблеми масштабування. Окрім вирішення проблем, цей підхід має певні переваги:

- З введенням серверної віртуалізації з’явилось поняття консолідації серверів. Це означає що задля скорочення кількості фізичних серверів що розміщують набори програм, ми можемо кожен додаток розмістити в окремому середовищі. Це середовище надаватиме додатку доступ до тих самих ресурсів що і фізичний сервер. За допомогою віртуалізації ми можемо створити декілька таких середовищ на одному фізичному сервері. Враховуючи, що додаток зазвичай не потребує усіх ресурсів системи, то поділ фізичного серверу на віртуальні машини не вплине на показники продуктивності цього додатку. В результаті масштабування не вимагатиме постійного збільшення кількості фізичних серверів.
- В результаті скорочення числа фізичних серверів, їх обслуговування можна проводити значно ефективніше. Проведення процесу налагодження серверу після помилок буде може відбуватись значно швидше. Для того щоб провести оновлення системи або запуск додатку потрібно менше зусиль.
- Так як віртуальна машина може відтворювати середовище

фізичного серверу включно з операційною системою. В результаті запуску декількох віртуальних машин на одному фізичному сервері може бути запущено кілька абсолютно різних операційних систем.

- Так як запуск віртуальної машини абсолютно не схожий з запуском фізичного серверу. Програми для керування віртуальними машинами надають можливості з перенесення запущеного комп'ютера на інший фізичний сервер. В результаті таких дій, зменшується ризик втрати даних при перенесенні з вимкненням і повторним запуском віртуальної машини. Операційна система як запущена на віртуальній машині може також раптово припиняти свою роботи, для зменшення ризиків є можливість створювати образи системи в даний момент часу. Таку дію варто робити постійно з певним інтервалом, що залежить від критичності даних що зберігаються.

Для досягнення описаних переваг потрібно виконувати додаткові дії, але в довгостроковій перспективі це зменшує вартість обслуговування та збільшує стійкість системи до помилок.

1.3.2 Недоліки серверної віртуалізації

Здійснюючи перехід на серверну віртуалізацію потрібно врахувати недоліки.

У зв'язку з тим що на одному фізичному сервері запущено декілька додатків, у випадку несправності апаратного забезпечення відбудуться проблеми у роботі багатьох додатків. Таким чином вплив на роботу всієї системи значно сильніший. Висока ймовірність перебої у роботі цілого ланцюгу процесів. Для уникнення таких проблем, архітекторам серверів потрібно розмістити критичні набори додатків на окремих фізичних серверах. Також додатково потрібно проводити стрес тестування окремої віртуальної машини на ймовірність виведення з ладу всього фізичного серверу.

Проблема ефективного розподілу обчислювальних ресурсів може бути нівельована створенням віртуальних машин що з часом перестають бути задіяними в обчисленнях. Один великий фізичний сервер може мати запущеними декілька віртуальних машин. Ці машини не застосовують

виділені на них ресурси, які можна розподілити між задіяними віртуальними машинами. Для пом’якшення ризиків потрібно налаштовувати моніторинг використання ресурсів віртуальними машинами щоб знаходити запущені віртуальні машини з великим відсотком незадіяних виділених ресурсів. У випадку відсутності дій з пом’якшення цього ризику, в результаті можуть відбуватись непрораховані витрати на розгортку додаткових потужностей.

Для розгортання віртуальних машин потрібні також кваліфіковані інженери. Так як цей процес вимагає опрацювання ризиків, спроби зекономити на рівні компетенції працівників може зашкодити роботи всієї системи. В результаті неправильної конфігурації кластер може бути виведеним з ладу, що принесе збитки.

1.3.3 Цілі серверної віртуалізації

Головною ціллю віртуалізації є проведення консолідації фізичних серверів. В результаті перенесення програм на віртуальні машини знижується кількість необхідних фізичних серверів. Тому з меншою кількістю серверів відбувається більше обчислень. Ефективність використання ресурсів стає значно вищою.

Покращення методології безперервної розгортки. Так як за допомогою віртуальної машини легше відтворити виробниче середовище під час тестування роботи програмного забезпечення, результативність проведення операцій тестування якості значно збільшується. Окрім тестування можна проводити пошук оптимальної конфігурації виділених ресурсів для програмного застосунку.

Для досягнення критеріїв доступності запущеного програмного забезпечення віртуалізація надає можливості перенесення віртуальних машин між фізичними серверами без їх зупинки. Це дозволяє уникнути простою серверів та відновлювати роботу після системних переривань.

1.4 Контейнеризація додатків

Контейнери це легший спосіб застосовувати віртуалізацію. Замість запуску віртуальної машини, запускається контейнер. Перша відмінність в

тому що не використовується додатковий шар програмного забезпечення – гіпервізор. Як наслідок виділення ресурсів відбувається швидше та доступність контейнера підвищується.

Контейнер має схожі можливості, також виділяються ресурси і інший контейнер не має уявлення про існування іншого. Їх схема роботи відрізняється, замість розгортання нової віртуальної машини, застосовується можливості операційної системи апаратного пристрою для виділення, розміщення та розмежування ресурсів. Але операційна не включається до образу контейнера. Тож в результаті контейнер містить лише програмне забезпечення та бібліотеки потрібні для запуску.

Переваги що надають контейнери схожі з перевагами віртуальної машини. Апаратні ресурси розподіляються ефективніше. Але контейнери дозволяють застосовувати мікросервісну архітектуру, це коли додаток ділиться на компоненти що можуть бути масштабовані індивідуально. Цей альтернативний підхід дозволяє уникнути масштабування всього додатку у випадку перевантаження одного з його компонентів. Також перевагою є те, що розгортання контейнерів у хмарному середовищі відбувається стандартизовано. Це кожен надавач хмарних послуг забезпечує цілковиту підтримку цього процесу[15].

1.5 Визначення вимог до програмного продукту веб-система підтримки діяльності кінотеатру

Чітке визначення вимог до продукту дозволяє окреслити бізнес-цінність діяльності. Пріоритизація вимог зазвичай виконується відносно їх впливу на артефакти цінності у продукті. Тому проведення специфікації вимог дозволяє вирішити питання максимальної ефективності виділення ресурсів.

Результати визначення специфікації вимог:

- отримання точної оцінки вартості, ризиків і витрат часу;
- клієнт зможе більш чітко сформувати власне бачення проекту;
- замовник і виконавець матимуть однакове уявлення про продукт;
- специфікація вимог допоможе виявити оптимальний набір функцій;

- специфікація вимог служить основою для формування іншої технічної документації;
- процес розробки буде оптимізований – мінімізовані витрати часу;
- ніякого дублювання завдань;
- дозволяє структурувати проблеми, щоб вирішувати їх простіше і швидше;
- вона допомагає зрозуміти, які результати вважаються оптимальними при тестуванні[24].

При описі Software Requirements Specification потрібно дотримувати ключових вимог:

- Опис повинен бути настільки коротким і чітким настільки це можливо. Надлишкові роз'яснення можуть ускладнити визначення завершеності цілі.
- Людина, що читатиме SRS повинна розуміти саме те, що написано, а не щось інше. Закон Мерфі: Якщо Вас можуть зрозуміти неправильно, Вас зрозуміють неправильно. Будь яка двозначність повинна бути усунута
- Простота і «читабельність». Використання ускладнених виразів спричинить затримку в обробці інформації
- Використання Схеми потоку Даних (Data flow diagram). Специфікація не може бути повною якщо ми не знаємо що на вході в описуваний програмний продукт, а що на виході. Все повинно бути чітко відображено.
- Висока ступінь деталізації. Це параметр є евристичною мірою. Його можна визначити так: якщо я, як автор, можу вільно викласти інформацію про функціонал і написане не викликає збентеження, якщо вимоги однозначні і не підлягають ніякому двоякому розумінню після прочитання, якщо вимоги в достатній для мене мірі описують поведінку функціоналу, то результат опрацювання вимог програмного продукту можна вважати завершеним

Висновки до розділу. Таким чином, у результаті проведеного

дослідження у першому розділі було проведено аналіз роботи веб-сервісу з використанням сучасних технологій. Було розглянуто основні проблеми що вирішує створення веб-сервісу, його роль в взаємодії компонентів в рамках веб-системи. Для ефективної роботи клієнт-серверної архітектури було розглянуто застосування принципів REST та їх дотримання. Також для визначення відповідності цим принципам було проаналізовано застосування моделі зрілості веб-системи імені Richardson'a (Richardson maturity model). Для визначення типу серверу на якому буде розгорнуто додаток було досліджено недоліки та переваги серверної віртуалізації разом з можливістю застосування контейнеризації компонентів системи. На основі обраних підходів до реалізації було проведено визначення вимог до програмного продукту та ключові критерії правильності опису специфікації вимог програмного забезпечення (Software Requirements Specification).

РОЗДІЛ 2. ОБРАНІ ПРОЕКТНІ ТА ТЕХНІЧНІ РІШЕННЯ

1.1 Архітектура програмної системи підтримки діяльності кінотеатру

Архітектура ПЗ передбачає декілька видів подань, які служать різним цілям:

1. зображенню функціональних можливостей системи;
2. відображенню логічної організації системи;
3. опису фізичної структури програмних компонентів в середовищі реалізації;
4. відображенню структури потоків управління та аспектів паралельної роботи;
5. опису фізичного розміщення програмних компонентів на базовій платформі.

Подання архітектури:

Зображення архітектури – є описом у спрощеному виді (абстракція) системи дослідженим в рамках вузької області знань, який опускає певні предмети, несуттєві із заданої точки зору. В той час охоплює певне коло інтересів. Зображує елементи, які мають цінність з архітектурної точки зору.

Архітектурно–значущим елементом називають предмет, що значно впливає на структурну складову системи та її продуктивність, міцність і здатність покращуватись в ході розробки[18]. Наприклад, для архітектурно–цінних об’єктів об'єктно–орієнтованого дизайну можуть бути включені основні класи предметної області, підсистеми та їх інтерфейси, ключові процеси або потоки керування.

Під час вибору архітектурного рішення було проведено зіставлення двох класифікацій архітектури – монолітну та мікросервісну

Перевагою монолітної архітектури є її швидкість розробки та легкість дотримання загальних вимог продуктивності всього продукту. Проведення оцінки показників швидкості виконання основних функції проводиться

точніше у зв’язку з наявністю відносно невеликої кількості компонентів, і деяких випадках це може бути однокомпонентна архітектура що матиме найвищі показники ефективності взаємодії. Як результат обсяг використаної пам’яті є меншим. Причиною цього є виділення операційною системою незастосованої пам’яті для кожного робочого процесу. Також час виконання задач буде меншим, бо проведення оптимізації алгоритмічних процесів може бути лише в межах єдиного програмного компонента.

Але з іншої точки зору, зі збільшенням масштабу програмного продукту побудованого на монолітній архітектурі складність внесення будь яких змін без порушення цілісності взаємодії буде лише зростати. Як результат кожна зміна потребувати більших трудових ресурсів і як результат більших фінансових витрат. Постійне збільшення кількості розробників що працюють над продуктом не зможе покращити ситуацію. Детально ця проблема описана у книзі Роберта Мартіна «Чиста Архітектура»[21]. Тому поділ системи на вузько направлені компоненти, що будуть розташовані окремо, але виконуватимуть лише певну частину задачі системи, буде кращим рішенням. У випадку потреби у внесенні змін до специфікацій вимог, дії будуть направлені лише на зміну тих компонентів що безпосередньо беруть участь у розв’язанні цих задач.

Враховуючи всі описані наслідки вибору архітектури для побудови веб-системи підтримки діяльності кінотеатру було обрано мікросервісне архітектурне рішення. Проблема використання більшої кількості ресурсів окремими компонентами буде розв’язана шляхом розміщення їх в абстракції названі контейнерами. Таким чином масштабування здійснюватиметься не шляхом збільшенням кількості обчислювальних машин, а розгортанням додаткових контейнерів з тими самими компонентами. Контейнери можуть бути розміщені як і в окремих операційних системах, так і на одній обчислювальній машині з однією операційною системою з використанням віртуалізації процесорних потоків. Для побудови веб-сайту використовуватиметься бібліотека React зі збереженням стану за допомогою

Redux[20].

Структура та вигляд веб інтерфейсу повинен повністю відповідати темі кваліфікаційної роботи, тому було розроблено тільки необхідні елементи управління, які в повному обсязі реалізують основний функціонал застосунку:

- пошук кінофільму або телесеріалу за назвою
- перегляд стислої інформації про кінопродукт
- вибір сесії перегляду в кінотеатрі
- бронювання місця
- можливість входу в систему
- можливість адміністратору додавати кіносеанси за датою

Інтерфейс складається з таких модулів – зображено на рисунку 2.1:

- Поле для пошуку
- Поля для вводу інформації для входу в систему
- Область відображення фільмів або телесеріалів для оцінки

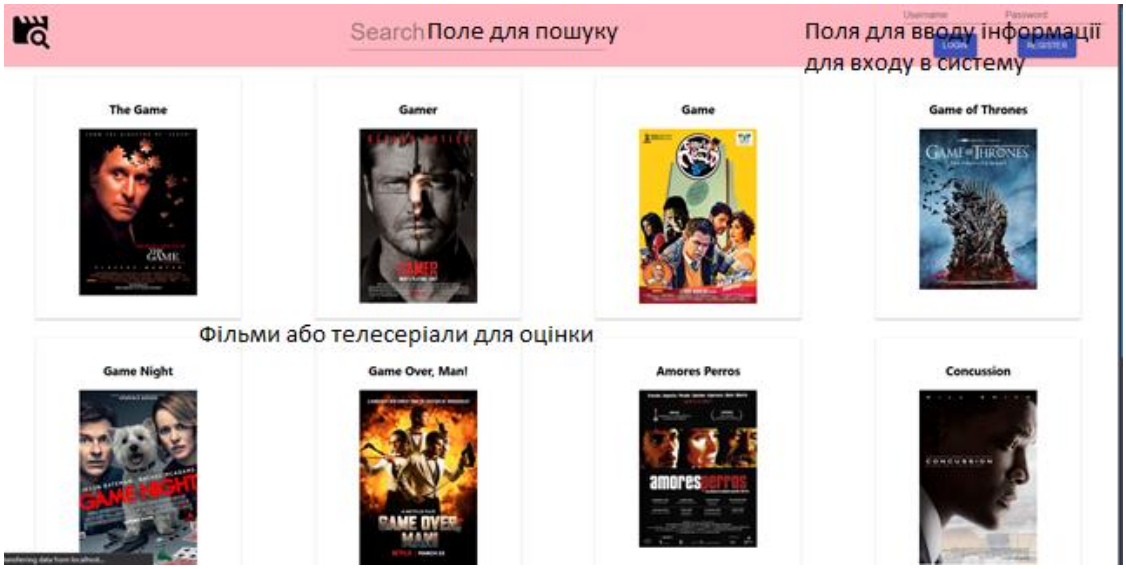


Рисунок 2.1 – Структура інтерфейсу веб-сайту

Для роботи веб-сервісу потрібно обрати відповідне програмне забезпечення для обробки http-запитів. Найсучаснішою програмною реалізацією веб-сервера для роботи з asp.net core фреймворком є Kestrel.

Kestrel - це багатоплатформовий веб-сервер для розміщення веб-додатку написаного з використанням фреймворку ASP.NET Core[5]. Kestrel зазвичай використовується при створенні проектів ASP.NET Core MVC, ASP.NET Core WebApi та Blazor[17]. Його рекомендовано застосовувати в середовищі розробки у зв'язку з легкістю розгортки на локальній машині. Але при застосуванні його в виробничому середовищі потрібно враховувати наявні вразливості при отриманні пакетів з мережі Internet. Дані вразливості є виправленні в реалізації веб-серверу Nginx.

Kestrel підтримує наступні сценарії:

- HTTPS
- HTTP / 2 (крім macOS)
- Для використання WebSockets потрібно налаштувати окремий компонент
- Розгортання Unix сокетів що проводять трансфорнування заголовків пакетів до серверу Nginx

Kestrel успішно запускається в усіх середовищах згідно з вказаними як підтримування для фреймворку .NET Core, в даний час це Microsoft Windows, Linux дистрибутиви та MacOS. З кожним інкрементом версії Asp.Net Core відбувається новий випуск Kestrel.

Потреба у Kestrel веб-сервері виникла при переході на кросплатформову розробку. Наявний IIS та IIS Express може працювати лише в ОС Windows.

При застосування Kestrel як веб-серверу застосунку, для проксювання запитів є можливість обрати з-поміж IIS, Apache та Nginx. В результаті названі сервери будуть взаємодіяти з мережею Internet і проводити перетворення отримувача пакетів на локальний веб-сервер Kestrel. Такий підхід дозволяє нам використовувати функціональні можливості що підтримуються цими серверами.

Програмна реалізація веб-серверу Kestrel застосовується для виконання функцій веб-серверу що не має прив'язки до середовища. Є надбудовою до бібліотеки парального вводу і виводу libuv. Фреймворк Asp.Net Core

проводить розгортання хоста у методі Main класу Program.cs[25]. Для запуску саме веб-серверу Kestrel викликається метод UseKestrel() але неявно, під час створення об'єкту WebHost.

Для більш детального налаштування веб-серверу Kestrel можна налаштувати параметри викликавши цей метод вручну. Метод приємає делегат як параметр що повертає об'єкт з усіма доступними значеннями. Наприклад, для вибору порту який буде прослуховуватись сервером, можна задати число у виді параметру у вигляді твердження (рядка коду).

Для встановлення обмежувальних значень потрібно задати змінні об'єкту Limit. Кількість з'єднань що можуть виконуватись додатком в момент часу задається у змінну MaxConcurrentConnections.

Вибір максильного розміру тіла запиту відбувається шляхом призначення числа змінній MaxRequestBodySize.

Вибір мінімальної швидкості передачі інформації відбувається шляхом призначення числа змінній MinRequestBodyDataRate.

Вибір мінімальної швидкості передачі інформації до клієнта відбувається шляхом призначення числа змінній MinResponseDataRate.

При розгортанні на Windows Kestrel може застосовувати IIS в якості проксі-сервера, а при розгортанні на Linux як проксі-сервери можуть використовуватися Apache і Nginx. Але також Kestrel може працювати самостійно всередині свого процесу без IIS.

Так, за замовчуванням в Visual Studio доступні дві можливості для запуску: з проксі через IIS і безпосередньо.

При виборі можливостей варіантів запуску ми можемо за замовчуванням зустріти два профілі:

IIS Express (запуск з проксюванням через IIS Express)

Профіль, який збігається з назвою проекту (в моєму випадку це HelloApp) - той пункт, який дозволяє запускати додаток в окремому процесі без всякого проксінг через IIS. Так як в файлі Program.cs встановлений в якості сервера Kestrel (методом UseKestrel ()), то в даному випадку додаток буде

запускати саме Kestrel. Причому за замовчуванням додаток буде запускатися на 5000-порту.

Запуск програми за допомогою Kestrel досить простий, нам не треба встановлювати і налаштовувати інші веб-сервери. Однак фахівці Microsoft рекомендують такий спосіб запуску переважно в рамках локальної внутрішньої мережі. А якщо у відкритій програмі для глобальної мережі інтернет, то рекомендованим способом запуску для забезпечення більшої безпеки є саме проксінг через IIS, Apache, Nginx. Подібний метод має ряд переваг у порівнянні зі звичайним запуском програми на Kestrel у вигляді самохостуючого процесу. Зокрема, проксі-сервери дозволяє приховати додатки, якщо вони не повинні бути доступні безпосередньо. Крім того, веб-сервери дозволяє управляти навантаженням до всіх програм, і надають інші функції з управління додатками.

Власне при створенні проекту ASP.NET Core в Visual Studio такий спосіб використовується за замовчуванням.

1.2 Опис специфікації OAuth 2.0

Платформа авторизації OAuth 2.0 дозволяє стороннім додаткам отримати обмежений доступ до служби HTTP, або від імені власника ресурсу шляхом організації взаємодії з утвердженням дозволів між власником ресурсу і службою HTTP, або дозволивши сторонньому додатоку отримати доступ від свого імені. Ця специфікація замінює і скасовує описаний протокол OAuth 1.0.

У традиційній клієнт-серверній моделі аутентифікації клієнт надсилає запит на ресурс з обмеженим доступом (захищений ресурс) шляхом аутентифікації на сервері з використанням реквізитів для входу.

Щоб надати стороннім додаткам доступ до обмежених ресурсів, власник ресурсу ділиться своїми обліковими даними з третьою стороною. Це створює ряд проблем і обмежень:

- Сторонні додатки повинні зберігати облікові дані власника ресурсу для майбутнього використання, зазвичай це пароль у вигляді відкритого тексту.

- Сервери повинні підтримувати аутентифікацію за паролем, незважаючи на недоліки безпеки, властиві паролям.
- Сторонні додатки отримують занадто широкий доступ до ресурсу
- захищених ресурсів власника, залишаючи власників ресурсів без будь-яких можливість обмежити тривалість або доступ до обмеженого набору ресурсів.
- Власники ресурсів не можуть відкликати доступ для окремої третьої особи, таким чином, щоб не анулювати доступ для всіх третіх осіб, і повинні зробити це, змінивши пароль третьої особи.

Злом будь-якого стороннього додатка призводить до компрометації пароля кінцевого користувача і всіх даних, захищених цим паролем.

OAuth вирішує ці проблеми, вводячи рівні авторизації і відділення ролі клієнта від ролі власника ресурсу. Згідно з OAuth, клієнт запитує доступ до ресурсів контрольованих власником ресурсу і розміщених на сервері ресурсів, і йому видається інший набір облікових даних, ніж у власника ресурсу.

Замість використання облікових даних власника ресурсу для доступу до захищених ресурсів клієнт отримує токен - рядок, що позначає конкретну область доступу, час життя та інші атрибути доступу. Токени доступу видаються стороннім клієнтам сервером авторизації зі схвалення власника ресурсу. Клієнт використовує токен доступу для доступу до захищених ресурсів, розміщених на сервері ресурсів.

Наприклад, кінцевий користувач (власник ресурсу) може надати службі друку (клієнту) доступ до своїх захищеним фотографій, що зберігаються в службі обміну фотографіями (сервер ресурсів), що не повідомляючи своє ім'я користувача та пароль службі друку. Замість цього вона проводить автентифікацію безпосередньо з сервером, якому довіряє служба обміну фотографіями (сервер авторизації), який видає облікові дані, пов'язані з делегуванням служби друку (токен доступу).

Ця специфікація призначена для використання з HTTP. Використання OAuth з будь-якого протоколу, крім HTTP, виходить за рамки специфікації OAuth.

1.2.1 Застосування OAuth 2.0 для отримання доступу до HTTP-сервісу

Для використання у межах фреймворку Asp.Net.Core доступний набір класів Microsoft.AspNetCore.Identity, Microsoft.AspNetCore.Authentication. JwtBearer [10]. Використання цих двох наборів класів дозволяє проводити валідацію токена, його генерацію на базі секретного ключа[16]. Токен генерується при надсиланні логіну і пароля необхідних для авторизації і надсилається клієнту, щоб той його надсилав у виді хедеру запиту.

Приклад згенерованого токену:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6ImJvZ2RhbjIiLCJlbWFpbCI6ImJvZ2RhbjJnbWFpbC5jb20iLCJuYmYiOjE2MjExOTk5NTMsImV4cCI6MTYyMTIwNTk1MywiaWF0IjoxNjIxMTk5OTUzLCJpc3MiOiJodHRwczovL2xvY2FsaG9zdDo1MDAxIiwiYXVkIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMSJ9.gLM91S_pUSwDnIg7iOF0qxvriyXPqS0M9ByuCUc6pw4

Після його розшифровки з використанням публічного ключа[14] маємо дані (рис.2.2.):

Рисунок 2.2 – Розшифрування

Кожен з цих полів є підтвердженим третьою стороною, що дозволяє уникнути підробки токену[3].

Веб-сайт для пошуку та оцінки фільмів для автентифікації використовує алгоритм отримання токену під назвою «Implicit flow»[2]. Сценарій покрокового застосування цього алгоритму зображено на рисунку 2.3.

Рисунок 2.3 – Алгоритм "Implicit flow" для отримання токену

Послідовність кроків відбувається таким чином[23]:

1. Користувач натискає на кнопку логіну
2. Проходить автентифікацію використовуючи логін і пароль надсилаючи його на точку доступу /oauth/token.
3. OAuth-орендатор проводить валідацію логіну і паролю.
4. Далі надсилає токен доступу до клієнтського додатку.
5. Клієнт надсилає дані користувача разом з токеном доступу
6. Веб-сервер надсилає відповідь

1.3 DynamoDB як NoSQL рішення для збереження даних

Amazon DynamoDB - це повністю керована служба бази даних NoSQL, яка забезпечує швидку і передбачувану продуктивність з безшовним маштабуванням[9]. DynamoDB дозволяє зняти навантаження з адміністрування по експлуатації і масштабування розподіленої бази даних, так що вам не доведеться турбуватися про виділення обладнання, налаштування та організацію реплікації, установці виправлень програмного забезпечення або масштабування кластера[11]. DynamoDB також пропонує шифрування в стані

спокою, що усуває операційну навантаження і складність, пов'язані із захистом конфіденційних даних. Дані зберігаються у JSON форматі у виді пари ключ-значення.

Веб-сайт для пошуку та оцінки фільмів зберігає дані про оцінені фільми в DynamoDb створюючи документ для кожного користувача

Приклад збереження даних про оцінений фільм (рис.2.4.):

Рисунок 2.4 – Збереження даних про оцінений фільм

Унікальним ключем – Primary Key в цьому випадку є Email користувача. А значенням документа з даним ключем може бути будь який JSON об’єкт, що дозволяє досягти гнучкості в процесі дизайну БД. На рисунку 2.5 зображено фінальну версію архітектури додатку.

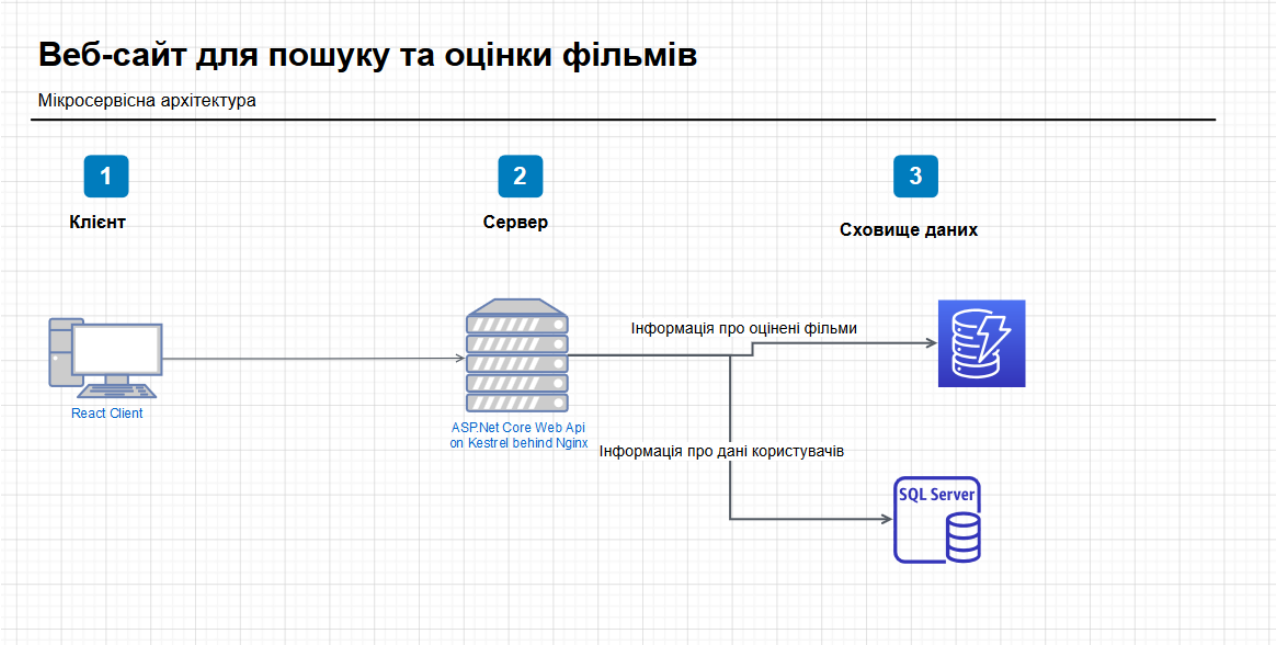


Рисунок 2.5 Взаємодія програмної системи в межах мікросервісної архітектури

Дотримання принципів RESTful при дизайні API дозволяє з мінімальними труднощами проводити масштабування всієї системи. Це можливо завдяки принципу stateless, тобто проміжний стан не зберігається на сервері, а відправляється одразу до бази даних. Це дозволяє уникнути роботи з неактуальними даними.

Висновки до розділу. Отже, в результаті проведення другої частини дослідження, було обрано архітектуру програмної системи підтримки діяльності кінотеатру, а саме мікросервісну архітектуру побудови веб-системи. Для дотримання вимог авторизації та автентифікації було досліджено специфікацію OAuth2.0 та алгоритми надання доступу до HTTP-сервісу. Також було проаналізовано переваги NoSQL рішень для збереження даних та програмний продукт компанії AWS - DynamoDb.

РОЗДІЛ 3. ОПИС РОБОТИ ВЕБ-САЙТУ

2.1 Функціональні можливості веб-сайту

Даний веб-сайт дозволяє користувачу проводити пошук та оцінку кінофільмів та серіалів. Веб-сайт побудований за принципом SPA – Single Page Application – односторінковий додаток. Дотримання даного принципу дозволяє переходити по сторінкам веб сайту без завантаження всього HTML файлу.

Побудова інтерфейсу відбувалась з врахуванням популярних технік верстки – зображено на рисунку 3.1. В результаті інтерфейс користувача є зручним та інтуїтивно зрозумілим, що дозволяє користувачеві отримати найкращий досвід використання веб-сайту.

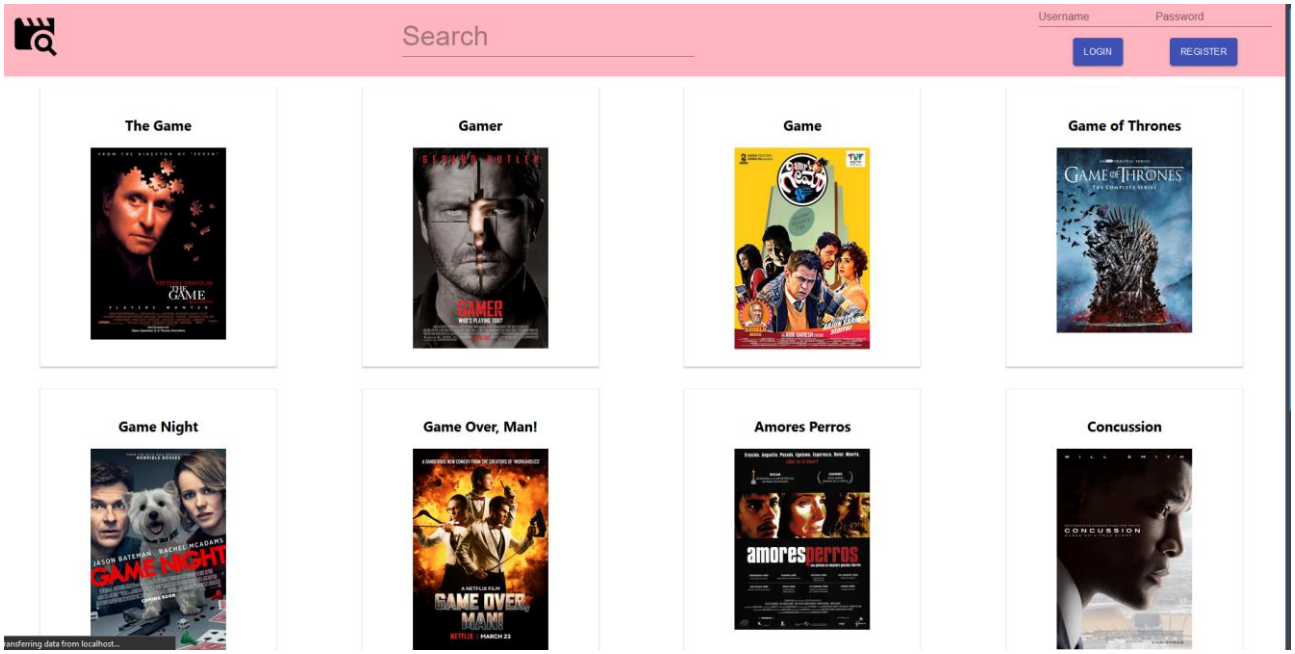


Рисунок 3.1 − Головна сторінка веб-сайту

Дані про фільми надходять з IMDb API, що забезпечує актуальність даних. Відбувається показ не тільки фільмів, а й телесеріалів.

Мобільна версія головної сторінки[19] є також зручною та інтуїтивно зрозумілою, зображена на рисунку 3.2.

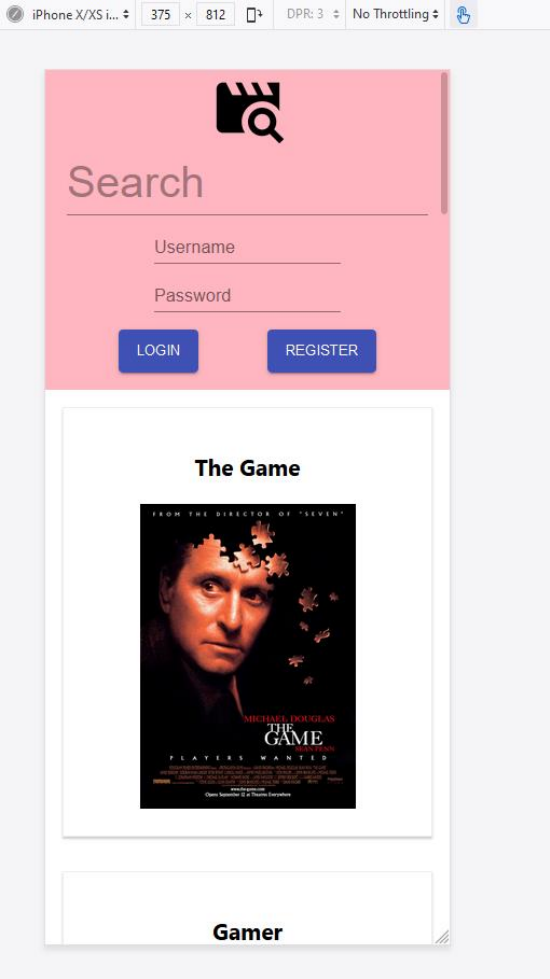


Рисунок 3.2 – Мобільна версія головної сторінки

Великі розміри елементів для взаємодії дозволяють уникнути випадкових натискань[12].

Для того щоб отримати можливість обрати місце потрібно пройти процес авторизації. Для реєстрації необхідно обрати незайнятий логін і пароль. Так само і для логіну потрібно ввести логін і пароль вказані при реєстрації.

При успішній авторизації у верхній частині сайту (рисунок 3.3) відображено логін користувача що успішно увійшов до системи.

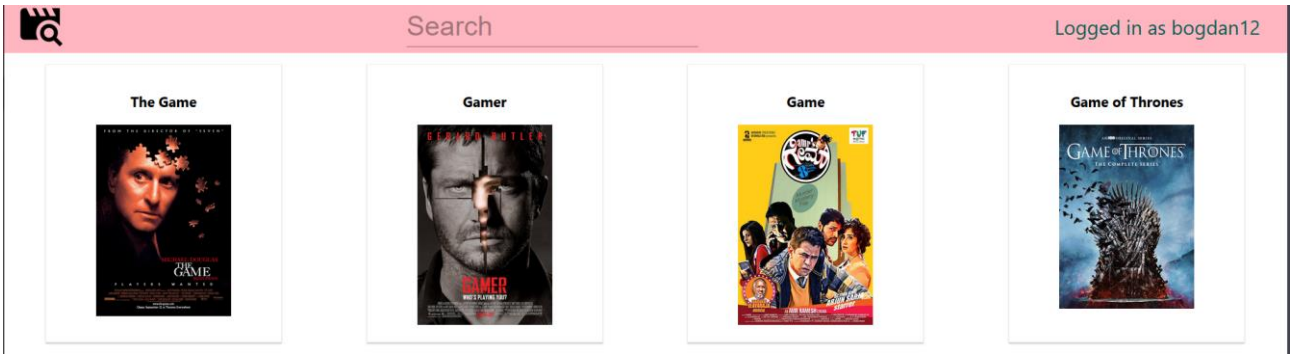


Рисунок 3.3 – Користувач увійшов в систему

У мобільній версії хід процесу входу користувача також зображено вірно. Емуляція дизайну сайту на мобільному пристрої зображена на рисунку 3.4.

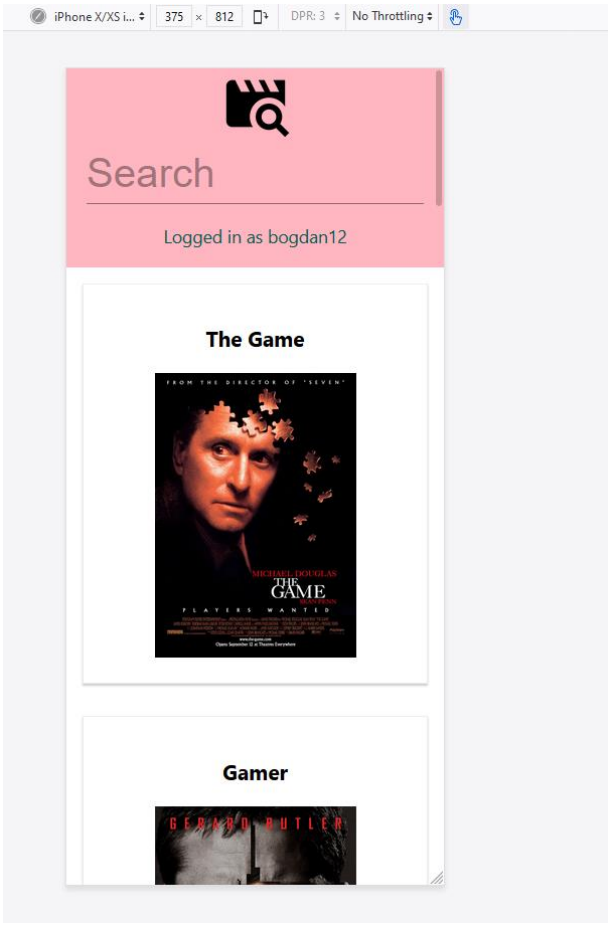


Рисунок 3.4 – Користувач увійшов на сайт з мобільного додатку

Результатом успішного входу в систему є отримання клієнтом токену доступу до веб-серверу. Інструменти браузеру дозволяються переглянути тіло

відповіді веб-серверу. Результат виконання запиту для отримання токену зображено на рисунку 3.5

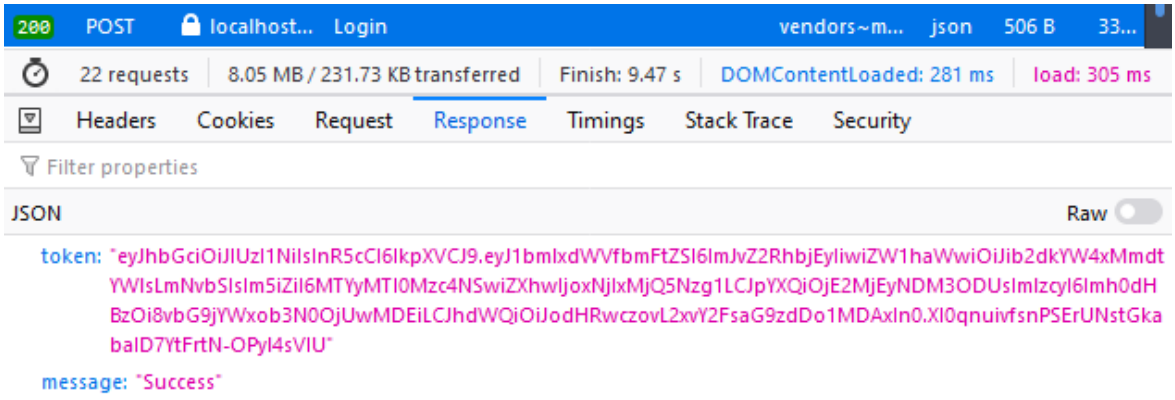


Рисунок 3.5 – Токен доступу згенерований після проходження аутентифікації

Для кожного запиту також надсилається Токен доступу у формі заголовка запиту. Список заголовків запиту клієнта та відповіді веб-серверу зображено на рисунку 3.6

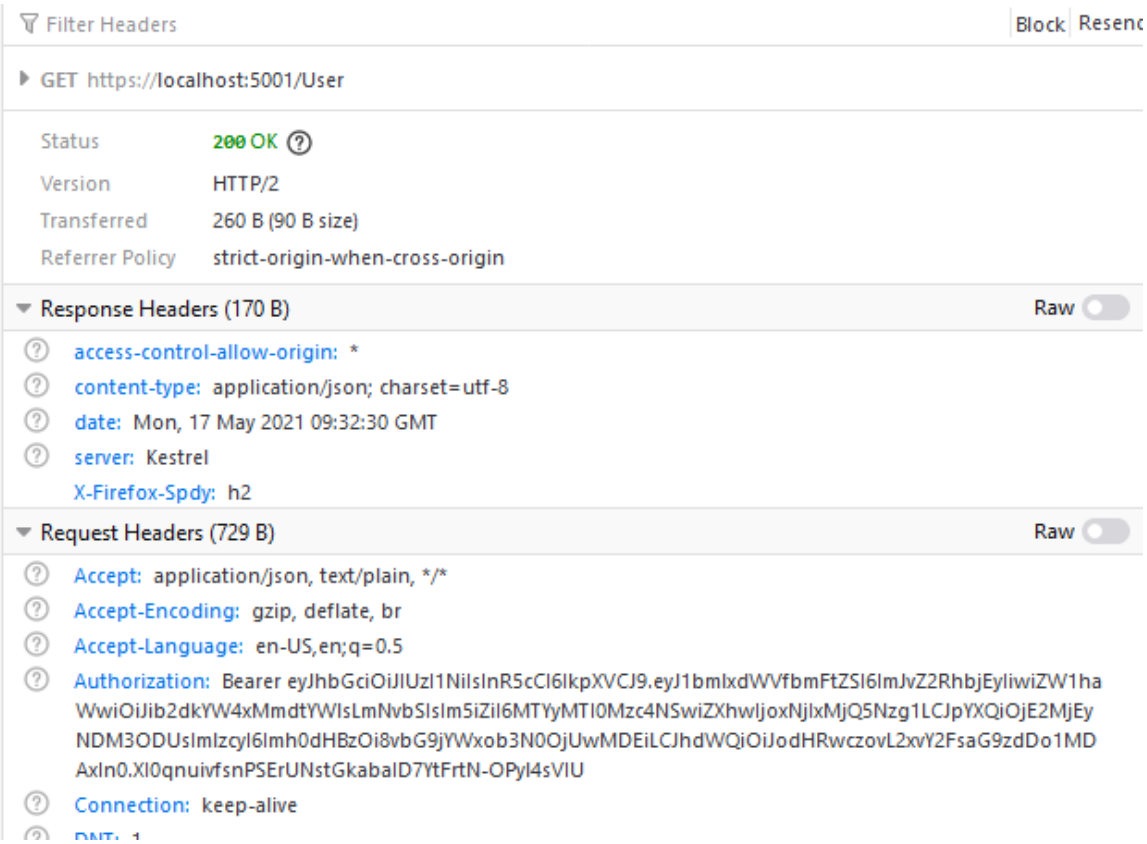


Рисунок 3.6 – Токен надсилається для кожного запиту

При натисканні на зображення фільму або телесеріалу відкривається зображення у повному розмірі разом з інформацією про рік, щоб уникнути

плутанини для фільмів з однаковою назвою. Вікно детальної інформації про фільм зображено на рисунку 3.7

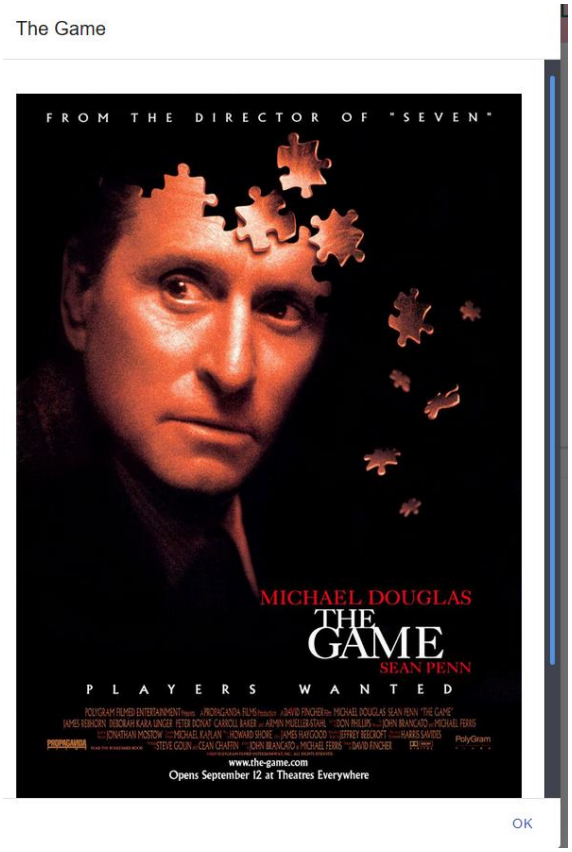


Рисунок 3.7 – Модальне вікно з інформацією про кінострічку

При виборі місця в залі, веб-сайт надсилає запит на збереження інформації про сесію. І щоразу при відкритті модального вікна надсилається запит на отримання актуальних сесій. Сервер зберігає обрані місця для кожної сесії. Клієнт не зберігає інформацію в браузері, натомість надсилає їх напряму на сервер.

Для того щоб додавати сесії в кінотеатру було розроблено роль адміністратора. Задля безпеки реєстрація такого користувача відбувається вручну. Таким чином лише отримувач послуг з перегляду фільмів в кінотеатрі може створити для себе аккаунт.

Відмінністю ролі адміністратора є можливість створення кіносеансу. Для цього за допомогою функціоналу пошуку можна обрати фільм для якого буде створено кіносеанс. Графічний інтерфейс функціоналу створення кіносеансу

зображено на рисунку 3.8.

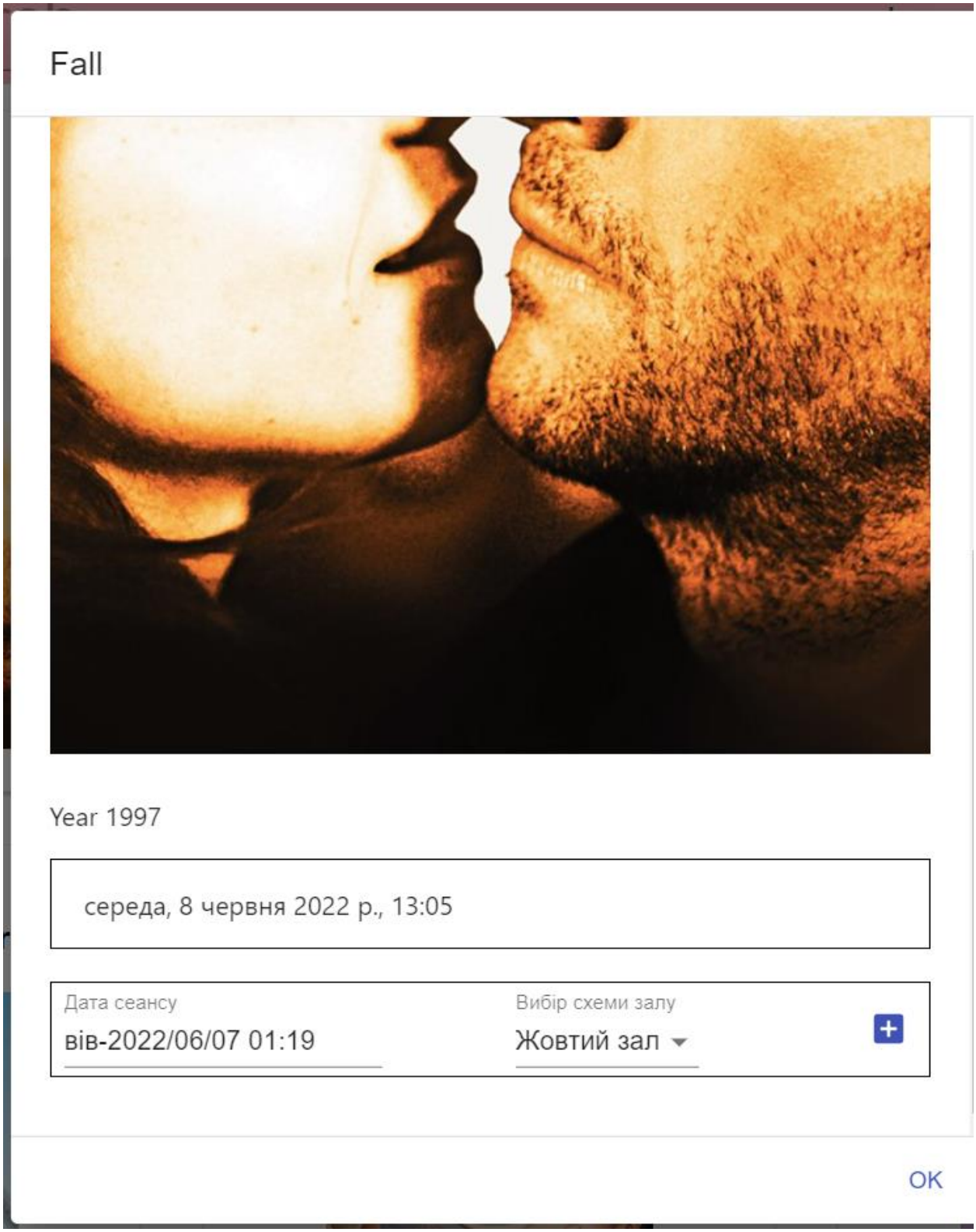


Рисунок 3.8 – Функціонал додавання сеансу до фільму

При відкритті модального вікна перегляду інформації про фільм адміністратор має можливість створити новий кіносеанс шляхом вибору дати та схеми залу. Для вибору дати використовується компонент DatePicker що

входить в пакет MaterialUI Design. Вибір дати відбувається шляхом вибору року, місяця та дня в окремому вікні компоненту, цей компонент зображено на рисунку 3.9.

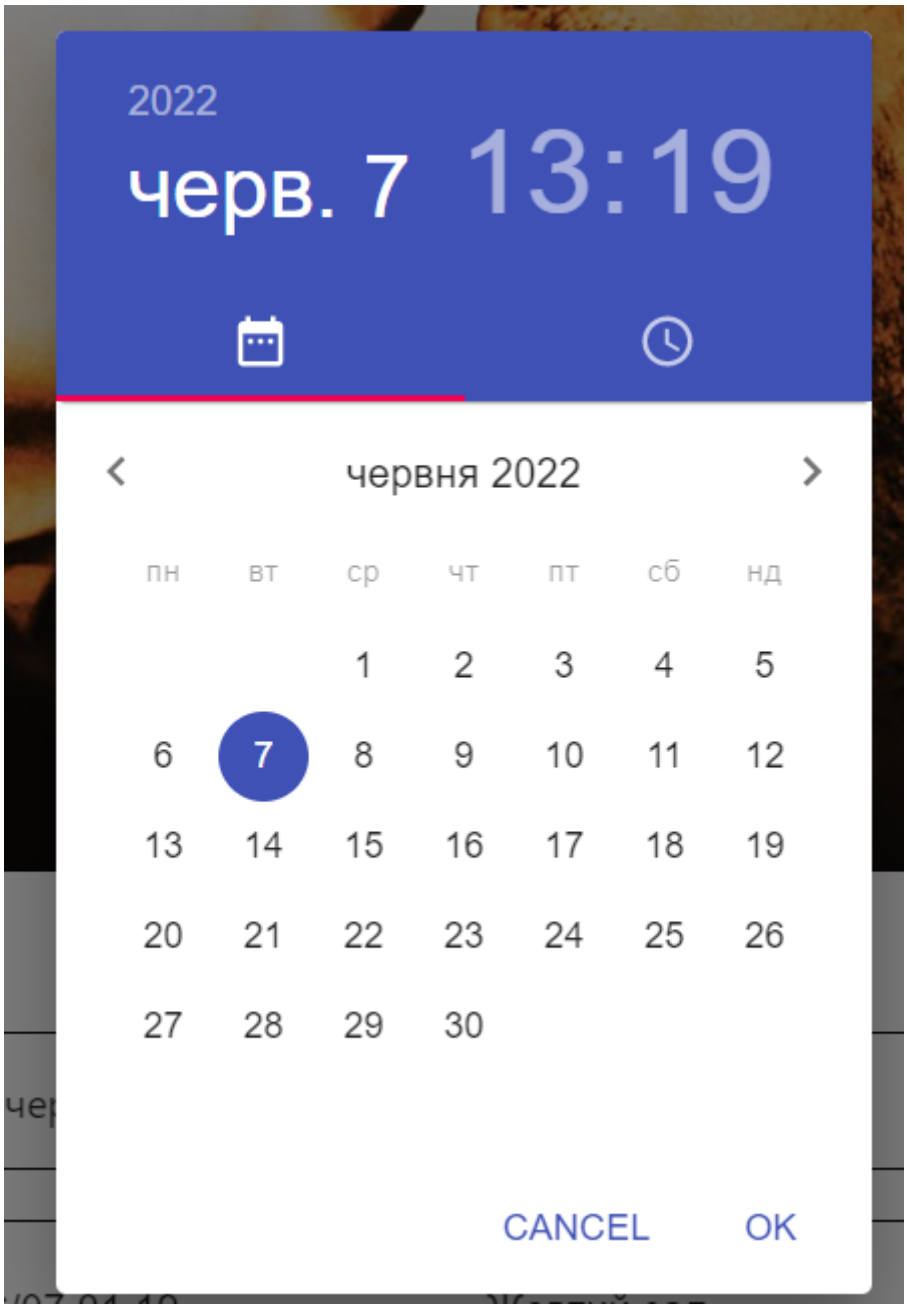


Рисунок 3.9 – Вибір дати кіносеансу

Для вибору часу сеансу потрібно змінити вкладку. Графічний дизайн функціоналу вибору часу зображено на рисунку 3.10.

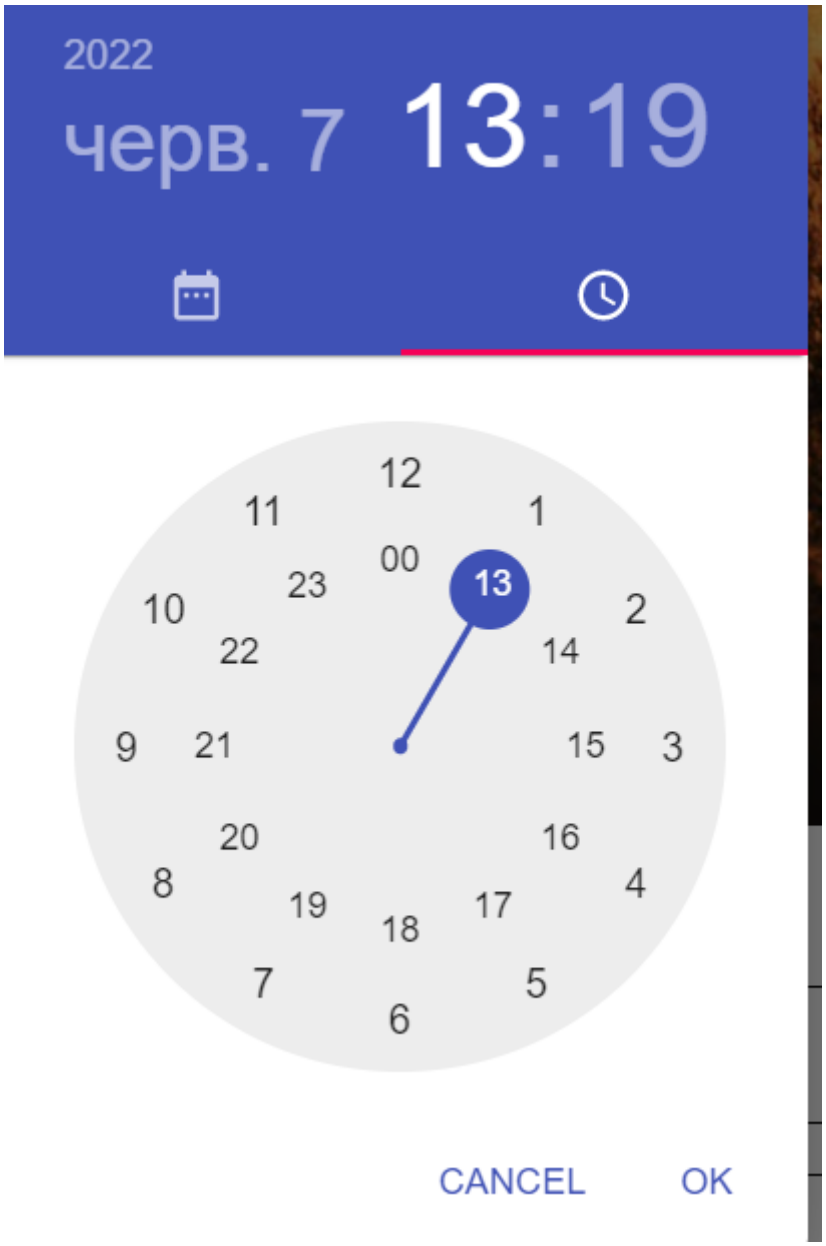


Рисунок 3.10 – Вибір часу кіносеансу

Після того як було обрано дату, потрібно обрати зал в якому буде відбуватись кіносеанс. Для цього потрібно обрати елемент зі списку залів. Функціонал вибору залу для кіносеансу зображено на рисунку 3.11

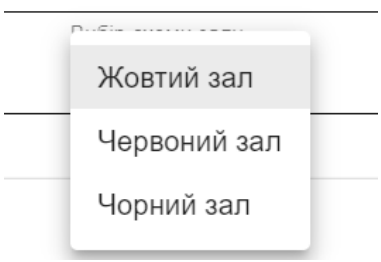


Рисунок 3.11 – Список залів для вибору місця проведення кіносеансу

Для створення кіносеансу потрібно натиснути на синю кнопку що

знаходиться поруч з обраним залом. Вигляд кнопки зображено на рисунку 3.8. В результаті виконання цих дій буде створено новий кіносеанс і він з’явиться в списку.

Користувач веб-сайт може обрати місце у залі для бажаної сесії. Для цього після вибору фільму, у вікні детальної інформації потрібно натиснути на елемент списку сесій для вибору сесії перегляду фільму. Після натискання буде показано схему залу на якій різними кольорами буде позначено стан всіх сидінь. Якщо місце позначено сірим кольором, значить воно вільне і користувач може його обрати натиснувши на іконку з номером. Дизайн графічного інтерфейсу функціоналу вибору місця в залі зображено на рисунку 3.12.

Рисунок 3.12 – Схема залу для обраної сесії

Обрані місця позначаються зеленим кольором. У випадку якщо інший користувач забронював вже місце, то воно буде зафарбоване синім кольором. Схема залу з місцями недоступними для вибору зображена на рисунку 3.13

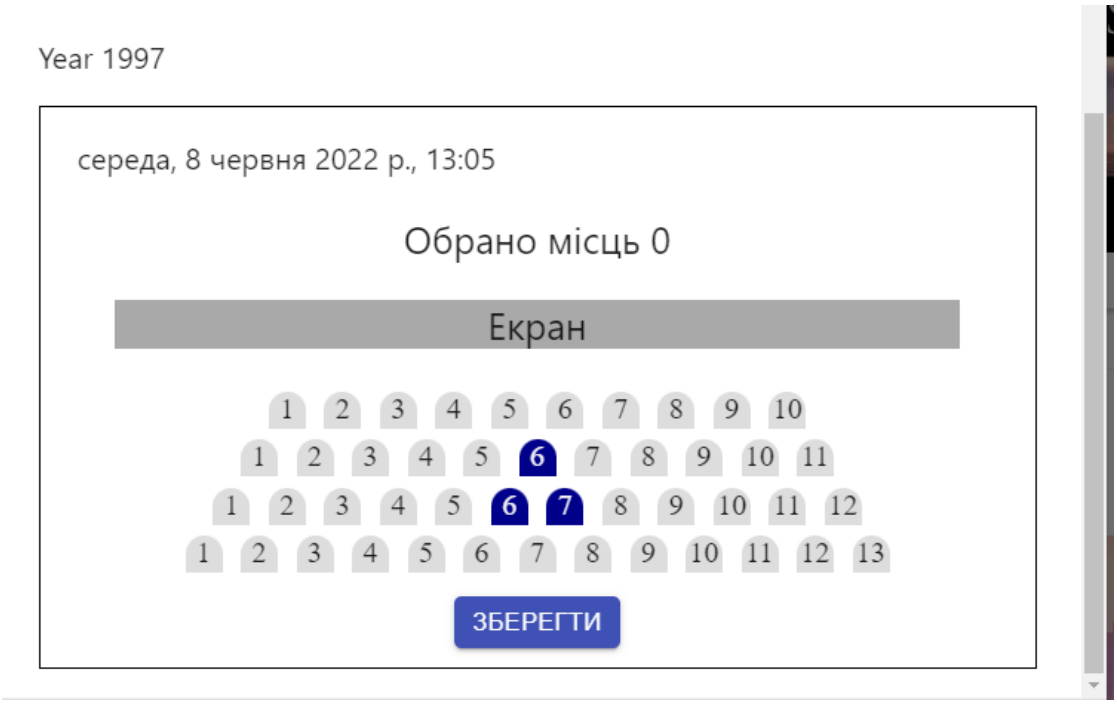


Рисунок 3.13 – Місця заброньовані іншими користувачами позначаються синім кольором

Веб система побудована на мікросервісній архітектурі. Для цього використовується віртуалізаційний інтерфейс процесора. Кожен мікросервіс має виділені ресурси і не може використовувати ресурси іншого контейнера. На відміну від віртуальних машин де додатки використовуватимуть окрему операційну систему, контейнери мають єдину операційну систему до якої надходять інструкції. Управлінням контейнерів займається Docker, для надсилання команд використовується консольний інтерфейс або графічний інтерфейс застосунку Docker Desktop. Список контейнерів зображено на рис 3.15

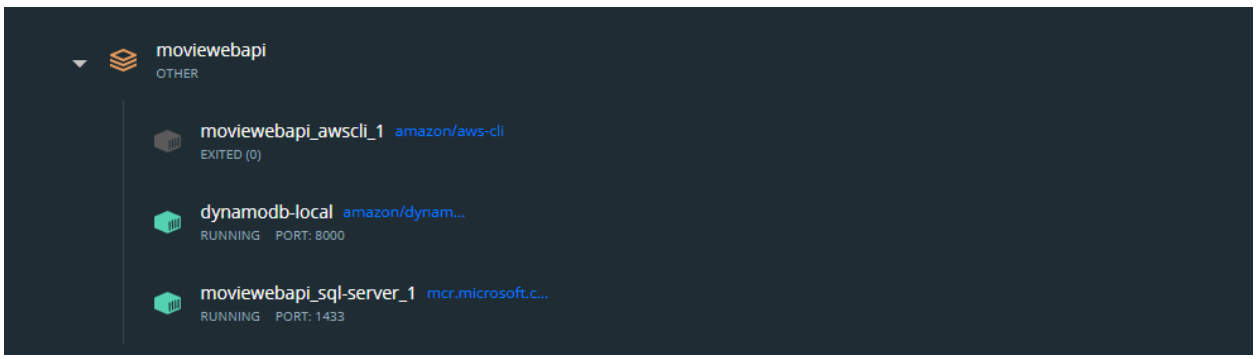


Рисунок 3.15 – Список контейнерів в мікросервісній архітектурі

Висновки до розділу. Таким чином, в результаті виконання третього розділу кваліфікаційної роботи було проведено проектування, розробку та тестування веб-системи. Також в цьому розділі було розглянуто всі ключові функціональні можливості веб-сайту та користувацький досвід що отримує відвідувач сайту.

ВИСНОВОК

У результаті виконаної кваліфікаційної роботи було спроектовано та розроблено програмну систему на мікросервісній архітектурі для підтримки діяльності кінотеатру

Під час виконання дипломної було проведено дослідження й узагальнення теоретичних аспектів побудови веб-сайту, веб-сервісу та їх взаємодії. Також була розглянута розгортка програм у ізольованих контейнерах. Для дослідження теоретичних засад було взято на розгляд відповідні стандарти та протоколи. Було використано документацію для бібліотек та фреймворків.

Також було досліджено архітектурні підходи для побудови веб-системи. Система реалізована з використанням React фреймворку для побудови клієнтського веб-додатку. З програмного забезпечення для розробки було використано програмне забезпечення компанії JetBrains – Rider та WebStorm. Дані про сесії кінотеатру зберігаються на IaaS (інфраструктура як сервіс) продукті корпорації Amazon – AWS. Використання Cloud рішень дозволило уникнути розгортки на локальному середовищі.

Для збереження даних використовується реляційна база даних розроблена компанією Microsoft – MS SQL Server 2019[13] та документо-орієнтована база даних що належить компанії Amazon – DynamoDb.

Головною особливістю програмного рішення є низький вплив веб-сайту на продуктивність браузеру, також варто відзначити підтримку мобільних пристроїв, що дозволяє охопити більшу кількість користувачів. Тому, було проведено тестування роботи сайту у браузері персонального комп’ютера та на мобільному пристрої

На web-сайті передбачені наступні функції та можливості:

- Авторизація за допомогою JWT Токену отриманого з веб-серверу з дотримання протоколу OAuth 2.0.
- Застосування Responsive design для зручності інтерфейсу користувача.

- Функціонал з пошуку фільмів та перегляд запланованих сеансів в кінотеатрі.
- Можливість бронювання місць користувачу що увійшов в систему за допомогою відображення актуальної схеми залу на сайті

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Протокол HTTP/2 *Internet Engineering Task Force, IETF* : веб-сайт. URL: https://tools.ietf.org/html/rfc7540 (переглянуто 3 травня 2022 року).
2. Прокотол доступу до даних реєстрації з використанням OpenID Connect *Internet Engineering Task Force, IETF* : веб-сайт. URL: https://tools.ietf.org/html/draft-ietf-regext-rdap-openid-06 (переглянуто 3 травня 2022 року).
3. Протокол OAuth 2.0 *Internet Engineering Task Force, IETF* : веб-сайт. URL: https://tools.ietf.org/html/rfc6749 (переглянуто 3 травня 2022 року).
4. Microservices on the Edge: The Infrastructure Impact *Internet Engineering Task Force, IETF* : веб-сайт. URL: https://www.ietf.org/proceedings/98/slides/slides-98-nfvrg-sessa-01-microservices-on-the-edge-the-infrastructure-impact-00.pdf (переглянуто 3 травня 2022 року).
5. Kestrel web server *Microsoft Software Developers Network* : веб-сайт. URL:https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-5.0 (переглянуто 3 травня 2022 року).
6. ASP.NET Cache *Web Training Room*: веб-сайт. URL: https://www.webtrainingroom.com/aspnetmvc/caching (переглянуто 3 травня 2022 року)
7. Введение в REST API — RESTful веб-сервисы - *Хабр* : веб-сайт. URL: https://habr.com/ru/post/483202/ (переглянуто 3 травня 2022 року).
8. What is REST – *REST API Tutorial*: веб-сайт. URL: https://restfulapi.net/ (переглянуто 3 травня 2022 року).
9. AWS DynamoDb getting started guide – *AWS Documentation*: веб-сайт. URL:

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html (переглянуто 3 травня 2022 року).

10. Introduction to Identity on ASP.NET Core – *Microsoft Docs*: веб-сайт. URL: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio (переглянуто 3 травня 2022 року).

11. AWS DynamoDb Optimistic locking – *AWS Documentation*: веб-сайт. URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.OptimisticLocking.html (переглянуто 3 травня 2022 року).

12. Sass — *Sass Lang*: веб-сайт. URL: https://sass-lang.com (переглянуто 3 травня 2022 року).

13. LINQ to SQL — *Microsoft*: веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/sql/linq/ (переглянуто 5 травня 2022 року).

14. SHA-1 — *Вікіпедія*: веб-сайт. URL: https://uk.wikipedia.org/wiki/SHA-1 (переглянуто 7 травня 2022 року).

15. Z. H. Wu. Cloud Computing: Analysis of the Core Technology. Posts & Telecom Press — Esri: веб-сайт. URL: http://www.esri.com/news/arcwatch/0110/feature.html (переглянуто 7 травня 2022 року).

16. Добавление Identity в проект с нуля– *Metanit*: веб-сайт. URL: https://metanit.com/sharp/aspnet5/16.2.php (переглянуто 3 травня 2022 року).

17. Troelsen A., Japikse P. Pro C# 9 with .NET 5 — Apress, 2021. – 1353p.

18. Соммервіл І. Інженерія програмного забезпечення, 6-е видання Вільямс, 2002. — 455с.

19. David C. Full-Stack React, TypeScript, and Node — Packt, 2020. – 648p.

20. Redux: веб-сайт. URL: https://redux.js.org/ (переглянуто 3 травня 2022 року).

21. Роберт М. Чистая архитектура – Питер, 2021. – 352p.

22. Плескач В.Л., Затонацька Т.Г. Електронна комерція: Підручник. — Київ: Знання, 2007. — 535 с.

23. Diagrams of All The OpenID Connect Flows– *Medium*: веб-сайт. URL: https://darutk.medium.com/diagrams-of-all-the-openid-connect-flows-6968e3990660 (переглянуто 3 травня 2022 року).

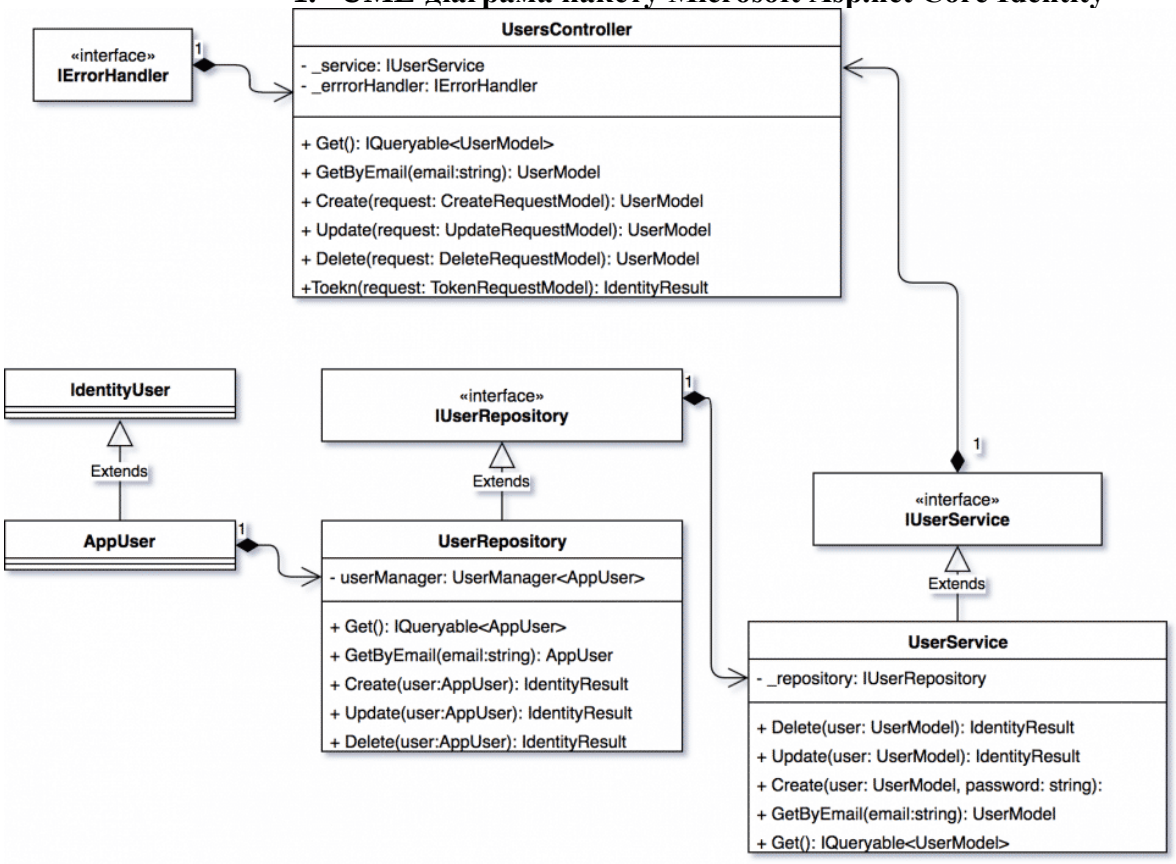
24. Unit testing ASP.NET Core Identity– *Samueleresca*: веб-сайт. URL: https://samueleresca.net/unit-testing-asp-net-core-identity (переглянуто 3 травня 2022 року).

25. Фримен А. Entity Framework Core 2 для ASP.NET Core MVC для профессионалов – Диалектика, 2019. – 626с.

ДОДАТКИ

ДОДАТОК А

1. UML-діаграма пакету Microsoft Asp.net Core Identity



ДОДАТОК Б

```
using System.IdentityModel.Tokens.Jwt;
using System.Text;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using MovieWebApi.Configuration;
using MovieWebApi.DataAdapters;
using MovieWebApi.DbContext;
using MovieWebApi.Options;
using MovieWebApi.Repositories;
using MovieWebApi.Services;
using SendGrid.Extensions.DependencyInjection;

namespace MovieWebApi
```

```
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "MovieWebApi",
Version = "v1" });
            });

            services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("SqlConnection")));
            services.AddScoped<DynamoDbDataAdapter>();

            services.AddScoped<UserInfoRepository>();
            services.AddScoped<SchemeRepository>();
            services.AddScoped<SessionRepository>();
            services.AddSendGrid(options => options.ApiKey =
"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");
            services.AddScoped<EmailSenderService>();
            services.AddIdentity<IdentityUser, IdentityRole>(options =>
                {
                    options.SignIn.RequireConfirmedAccount = true;
                    options.Password.RequireNonAlphanumeric = false;
                    options.Password.RequireDigit = false;
                    options.Password.RequireUppercase = false;
                })
                .AddRoles<IdentityRole>()
                .AddEntityFrameworkStores<ApplicationDbContext>();

            JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();

            var jwtSection =
Configuration.GetSection("JwtBearerTokenSettings");
            services.Configure<JwtBearerTokenSettings>(jwtSection);
            var jwtBearerTokenSettings =
jwtSection.Get<JwtBearerTokenSettings>();
            var key =
Encoding.ASCII.GetBytes(jwtBearerTokenSettings.SecretKey);
            services.AddAuthentication(cfg =>
            {
                cfg.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
                cfg.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
            }).AddJwtBearer(options =>
            {
                options.RequireHttpsMetadata = false;
                options.SaveToken = true;
                options.TokenValidationParameters = new
TokenValidationParameters()
                {
                    ValidIssuer = jwtBearerTokenSettings.Issuer,
```

```
                    ValidAudience = jwtBearerTokenSettings.Audience,
                    IssuerSigningKey = new SymmetricSecurityKey(key),
                };
            });

            services.AddScoped<ImdbApiDataAdapter>();

services.Configure<ImdbApiOptions>(Configuration.GetSection(ImdbApiOptions.ImdbApi));
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "MovieWebApi v1"));
            }

            app.UseCors(builder => builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());

            app.UseRouting();

            app.UseAuthentication();

            app.UseAuthorization();

            app.UseEndpoints(endpoints => { endpoints.MapControllers(); });
        }
    }
}
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using MovieWebApi.DbContext;
using Serilog;

namespace MovieWebApi
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateHostBuilder(args).Build();
            using (var scope = host.Services.CreateScope())
            {
                var applicationDbContext= scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
                applicationDbContext.Database.EnsureCreated();
            }

            host.Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .UseSerilog((context, services, loggerConfiguration) =>
```

```
                {
                    loggerConfiguration
                        .Enrich.FromLogContext()
                        .WriteTo.Console()
                        .WriteTo.File("logs/log.txt", rollingInterval:
RollingInterval.Day);
                })
                .ConfigureWebHostDefaults(webBuilder => {
webBuilder.UseStartup<Startup>(); });
    }
}
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;

namespace MovieWebApi
{
    public class NonUniqueIntListPropertyMapper : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            if (value == null)
            {
                return DynamoDBNull.Null;
            }

            if (value is IEnumerable<int> objects)
            {
                return DynamoDBList.Create(objects);
            }

            return DynamoDBNull.Null;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            if (entry.GetType() == typeof(DynamoDBList))
            {
                return entry.AsDynamoDBList().Entries.Select(primitive =>
primitive.AsInt()).ToList();
            }

            return null;
        }
    }
}
using System.Collections.Generic;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Newtonsoft.Json;

namespace MovieWebApi
{
    public class DictionaryOfListsPropertyMapper: IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
           return JsonConvert.SerializeObject(value as Dictionary<int,
List<int>>);
        }

        public object FromEntry(DynamoDBEntry entry)
```

```
        {
            if (entry is DynamoDBNull)
            {
                return null;
            }
            return JsonConvert.DeserializeObject<Dictionary<int,
List<int>>>(entry.AsString());
        }
    }
    public class DictionaryOfDictionariesPropertyMapper: IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            return JsonConvert.SerializeObject(value as
Dictionary<string,Dictionary<int, List<int>>>);
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            if (entry is DynamoDBNull)
            {
                return null;
            }
            return
JsonConvert.DeserializeObject<Dictionary<string,Dictionary<int,
List<int>>>>(entry.AsString());
        }
    }
}
using System.Collections.Generic;
using System.Threading.Tasks;
using MovieWebApi.DataAdapters;
using MovieWebApi.Models;

namespace MovieWebApi.Repositories
{
    public class UserInfoRepository
    {
        private readonly DynamoDbDataAdapter dynamoDataAdapter;

        public UserInfoRepository(DynamoDbDataAdapter dynamoDataAdapter)
        {
            this.dynamoDataAdapter = dynamoDataAdapter;
        }

        public async Task<bool> Save(UserInfo entity)
        {
            await this.dynamoDataAdapter.Save(entity);
            return true;
        }

        public async Task<UserInfo> Single(string id)
        {
            return await this.dynamoDataAdapter.Single<UserInfo>(id);
        }
    }
}
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using MovieWebApi.DataAdapters;
using MovieWebApi.Models;
```

```
namespace MovieWebApi.Repositories
{
    public class SessionRepository
    {
        private readonly DynamoDbDataAdapter dynamoDataAdapter;

        public SessionRepository(DynamoDbDataAdapter dynamoDataAdapter)
        {
            this.dynamoDataAdapter = dynamoDataAdapter;
        }

        public async Task<MovieSession> Single(string id)
        {
            return await this.dynamoDataAdapter.Single<MovieSession>(id);
        }

        public async Task Create(MovieSession movieSession)
        {
            await this.dynamoDataAdapter.Save(movieSession);
        }

        public async Task Update(MovieSession movieSession)
        {
            await this.dynamoDataAdapter.Save(movieSession);
        }

        public async Task<List<MovieSession>> All()
        {
            return (await
this.dynamoDataAdapter.All<MovieSession>()).ToList();
        }
    }
}
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using MovieWebApi.DataAdapters;
using MovieWebApi.Models;

namespace MovieWebApi.Repositories
{
    public class SchemeRepository
    {
        private readonly DynamoDbDataAdapter dynamoDataAdapter;

        public SchemeRepository(DynamoDbDataAdapter dynamoDataAdapter)
        {
            this.dynamoDataAdapter = dynamoDataAdapter;
        }

        public async Task<Scheme> Single(string id)
        {
            return await this.dynamoDataAdapter.Single<Scheme>(id);
        }

        public async Task Create(Scheme scheme)
        {
            await this.dynamoDataAdapter.Save(scheme);
        }

        public async Task Update(Scheme scheme)
        {
            await this.dynamoDataAdapter.Save(scheme);
```

```
        }

        public async Task<List<Scheme>> All()
        {
            return (await this.dynamoDataAdapter.All<Scheme>()).ToList();
        }
    }
}

namespace MovieWebApi.Options
{
    public class ImdbApiOptions
    {
        public const string ImdbApi = "ImdbApi";
        public string ApiKey { get; set; } = string.Empty;
        public string BaseUrl { get; set; } = string.Empty;
        public string Host { get; set; } = string.Empty;
    }
}
using System.Collections.Generic;
using Amazon.DynamoDBv2.DataModel;
using Newtonsoft.Json;

namespace MovieWebApi.Models
{
    [DynamoDBTable("UserInfo")]
    public class UserInfo
    {
        [DynamoDBProperty("userId")]
        [JsonProperty("userId")]
        public string UserId { get; set; }
        [JsonProperty("movieMarks")]
        [DynamoDBProperty("movieMarks")]
        public Dictionary<string,int> MovieMarks { get; set; }
        [DynamoDBProperty("selectedSeats",
typeof(DictionaryOfDictionariesPropertyMapper))]
        public Dictionary<string,Dictionary<int, List<int>>> SelectedSeats {
get; set; }
        [JsonProperty("role")]
        public string Role { get; set; }
    }
}
using System.ComponentModel.DataAnnotations;

namespace MovieWebApi.Models
{
    public class UserDetails
    {
        [Required]
        public string UserName { get; set; }

        [Required]
        public string Password { get; set; }

        [Required]
        public string Email { get; set; }
    }
}
using Newtonsoft.Json;

namespace MovieWebApi.Models
{
    public class SearchResult
```

```
    {
        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("image")]
        public Image Image { get; set; }

        [JsonProperty("title")]
        public string Title { get; set; }
    }
}
using System.Collections.Generic;
using Amazon.DynamoDBv2.DataModel;

namespace MovieWebApi.Models
{
    [DynamoDBTable("Schemes")]
    public class Scheme
    {
        public string Id { get; set; }
        public string Name { get; set; }
        [DynamoDBProperty(typeof(NonUniqueIntListPropertyMapper))]
        public List<int> Seats { get; set; }
    }
}
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2.DataModel;
using Newtonsoft.Json;

namespace MovieWebApi.Models
{
    [DynamoDBTable("MovieSessions")]
    public class MovieSession
    {
        [JsonProperty("id")]
        public string Id { get; set; }
        [JsonProperty("startDate")]
        public DateTime StartDate { get; set; }
        [JsonProperty("endDate")]
        public DateTime EndTime { get; set; }
        [JsonProperty("schemeId")]
        public string SchemeId { get; set; }
        [JsonProperty("movieId")]
        public string MovieId { get; set; }
        [JsonProperty("occupiedPlaces")]
        [DynamoDBProperty(typeof(DictionaryOfListsPropertyMapper))]
        public Dictionary<int, List<int>> OccupiedPlaces { get; set; }
    }
}
using Newtonsoft.Json;

namespace MovieWebApi.Models
{
    public class MovieDetails
    {
        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("image")]
        public Image Image { get; set; }

        [JsonProperty("title")]
```

```
        public string Title { get; set; }

        [JsonProperty("titleType")]
        public string TitleType { get; set; }

        [JsonProperty("year")]
        public string Year { get; set; }

        [JsonProperty("seriesEndYear")]
        public string SeriesEndYear { get; set; }

        [JsonProperty("seriesStartYear")]
        public string SeriesStartYear { get; set; }

        [JsonProperty("numberOfEpisodes")]
        public string NumberOfEpisodes { get; set; }
    }
}
namespace MovieWebApi.Models
{
    public class LoginCredentials
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
using Newtonsoft.Json;

namespace MovieWebApi.Models
{
    public class Image
    {
        [JsonProperty("url")]
        public string Url { get; set; }
    }
}
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using MovieWebApi.DbContext;

namespace MovieWebApi.Migrations
{
    [DbContext(typeof(ApplicationDbContext))]
    partial class ApplicationDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("Relational:MaxIdentifierLength", 128)
                .HasAnnotation("ProductVersion", "5.0.6")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole",
b =>
                {
                    b.Property<string>("Id")
                        .HasColumnType("nvarchar(450)");
```

```
                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("Name")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique()
                        .HasDatabaseName("RoleNameIndex")
                        .HasFilter("[NormalizedName] IS NOT NULL");

                    b.ToTable("AspNetRoles");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>"
, b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int")
                        .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

                    b.Property<string>("ClaimType")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("ClaimValue")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("RoleId")
                        .IsRequired()
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("Id");

                    b.HasIndex("RoleId");

                    b.ToTable("AspNetRoleClaims");
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUser",
b =>
                {
                    b.Property<string>("Id")
                        .HasColumnType("nvarchar(450)");

                    b.Property<int>("AccessFailedCount")
                        .HasColumnType("int");

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("nvarchar(max)");
```

```
                    b.Property<string>("Email")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<bool>("EmailConfirmed")
                        .HasColumnType("bit");

                    b.Property<bool>("LockoutEnabled")
                        .HasColumnType("bit");

                    b.Property<DateTimeOffset?>("LockoutEnd")
                        .HasColumnType("datetimeoffset");

                    b.Property<string>("NormalizedEmail")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<string>("NormalizedUserName")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<string>("PasswordHash")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("PhoneNumber")
                        .HasColumnType("nvarchar(max)");

                    b.Property<bool>("PhoneNumberConfirmed")
                        .HasColumnType("bit");

                    b.Property<string>("SecurityStamp")
                        .HasColumnType("nvarchar(max)");

                    b.Property<bool>("TwoFactorEnabled")
                        .HasColumnType("bit");

                    b.Property<string>("UserName")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedEmail")
                        .HasDatabaseName("EmailIndex");

                    b.HasIndex("NormalizedUserName")
                        .IsUnique()
                        .HasDatabaseName("UserNameIndex")
                        .HasFilter("[NormalizedUserName] IS NOT NULL");

                    b.ToTable("AspNetUsers");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>"
, b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int")
                        .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);
```

```
                    b.Property<string>("ClaimType")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("ClaimValue")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("Id");

                    b.HasIndex("UserId");

                    b.ToTable("AspNetUserClaims");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>"
, b =>
                {
                    b.Property<string>("LoginProvider")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("ProviderKey")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("ProviderDisplayName")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("AspNetUserLogins");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>",
b =>
                {
                    b.Property<string>("UserId")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("RoleId")
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("UserId", "RoleId");

                    b.HasIndex("RoleId");

                    b.ToTable("AspNetUserRoles");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>"
, b =>
                {
                    b.Property<string>("UserId")
```

```
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("LoginProvider")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("Name")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("Value")
                        .HasColumnType("nvarchar(max)");

                    b.HasKey("UserId", "LoginProvider", "Name");

                    b.ToTable("AspNetUserTokens");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole",
null)
                        .WithMany()
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>",
b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole",
null)
                        .WithMany()
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
```

```
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });
#pragma warning restore 612, 618
        }
    }
}
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using MovieWebApi.DbContext;

namespace MovieWebApi.Migrations
{
    [DbContext(typeof(ApplicationDbContext))]
    [Migration("20210513090434_InitialCreate")]
    partial class InitialCreate
    {
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("Relational:MaxIdentifierLength", 128)
                .HasAnnotation("ProductVersion", "5.0.6")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole",
b =>
                {
                    b.Property<string>("Id")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("Name")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256)
```

```
                        .HasColumnType("nvarchar(256)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique()
                        .HasDatabaseName("RoleNameIndex")
                        .HasFilter("[NormalizedName] IS NOT NULL");

                    b.ToTable("AspNetRoles");
                });


modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>"
, b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int")
                        .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

                    b.Property<string>("ClaimType")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("ClaimValue")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("RoleId")
                        .IsRequired()
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("Id");

                    b.HasIndex("RoleId");

                    b.ToTable("AspNetRoleClaims");
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUser",
b =>
                {
                    b.Property<string>("Id")
                        .HasColumnType("nvarchar(450)");

                    b.Property<int>("AccessFailedCount")
                        .HasColumnType("int");

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("Email")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<bool>("EmailConfirmed")
                        .HasColumnType("bit");

                    b.Property<bool>("LockoutEnabled")
                        .HasColumnType("bit");

                    b.Property<DateTimeOffset?>("LockoutEnd")
```

```
                        .HasColumnType("datetimeoffset");

                    b.Property<string>("NormalizedEmail")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<string>("NormalizedUserName")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.Property<string>("PasswordHash")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("PhoneNumber")
                        .HasColumnType("nvarchar(max)");

                    b.Property<bool>("PhoneNumberConfirmed")
                        .HasColumnType("bit");

                    b.Property<string>("SecurityStamp")
                        .HasColumnType("nvarchar(max)");

                    b.Property<bool>("TwoFactorEnabled")
                        .HasColumnType("bit");

                    b.Property<string>("UserName")
                        .HasMaxLength(256)
                        .HasColumnType("nvarchar(256)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedEmail")
                        .HasDatabaseName("EmailIndex");

                    b.HasIndex("NormalizedUserName")
                        .IsUnique()
                        .HasDatabaseName("UserNameIndex")
                        .HasFilter("[NormalizedUserName] IS NOT NULL");

                    b.ToTable("AspNetUsers");
                });


modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>"
, b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int")
                        .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

                    b.Property<string>("ClaimType")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("ClaimValue")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("Id");
```

```
                    b.HasIndex("UserId");

                    b.ToTable("AspNetUserClaims");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>"
, b =>
                {
                    b.Property<string>("LoginProvider")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("ProviderKey")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("ProviderDisplayName")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("AspNetUserLogins");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>",
b =>
                {
                    b.Property<string>("UserId")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("RoleId")
                        .HasColumnType("nvarchar(450)");

                    b.HasKey("UserId", "RoleId");

                    b.HasIndex("RoleId");

                    b.ToTable("AspNetUserRoles");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>"
, b =>
                {
                    b.Property<string>("UserId")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("LoginProvider")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("Name")
                        .HasColumnType("nvarchar(450)");

                    b.Property<string>("Value")
                        .HasColumnType("nvarchar(max)");
```

```
                    b.HasKey("UserId", "LoginProvider", "Name");

                    b.ToTable("AspNetUserTokens");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole",
null)
                        .WithMany()
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>"
, b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>",
b =>
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole",
null)
                        .WithMany()
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();

                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>"
, b =>
```

```
                {
                    b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser",
null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });
#pragma warning restore 612, 618
        }
    }
}
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.Options;
using MovieWebApi.Models;
using MovieWebApi.Options;
using Newtonsoft.Json.Linq;
using RestSharp;

namespace MovieWebApi.DataAdapters
{
    public class ImdbApiDataAdapter
    {
        private readonly ImdbApiOptions options;

        private readonly RestClient _restClient;

        public ImdbApiDataAdapter(IOptions<ImdbApiOptions> iOptions)
        {
            this.options = iOptions.Value;
            _restClient = new RestClient(options.BaseUrl);
            _restClient.AddDefaultHeaders(new Dictionary<string, string>
            {
                {"x-rapidapi-key", options.ApiKey},
                {"x-rapidapi-host", options.Host}
            });
        }

        public async Task<List<SearchResult>> SearchByName(string keyWord)
        {
            RestRequest restRequest = new RestRequest("/title/find",
Method.GET);
            restRequest.AddQueryParameter("q", keyWord);
            var restResponse = await _restClient.ExecuteAsync(restRequest);
            JObject response = JObject.Parse(restResponse.Content);
            var searchResults =
Newtonsoft.Json.JsonConvert.DeserializeObject<List<SearchResult>>(response["r
esults"]?.ToString() ?? string.Empty);
            return searchResults;
        }

        public async Task<MovieDetails> GetMovieById(string id)
        {
            RestRequest restRequest = new RestRequest("/title/get-details",
Method.GET);
            restRequest.AddQueryParameter("tconst", id);
            var restResponse = await _restClient.ExecuteAsync(restRequest);
            JObject response = JObject.Parse(restResponse.Content);
            var searchResults =
Newtonsoft.Json.JsonConvert.DeserializeObject<MovieDetails>(restResponse.Cont
```

```
ent);
            return searchResults;
        }
    }
}
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;

namespace MovieWebApi.DataAdapters
{
    public class DynamoDbDataAdapter
    {
        private readonly DynamoDBContext context;


        public DynamoDbDataAdapter()
        {
            DynamoDBContextConfig config = new DynamoDBContextConfig();
            var client = new AmazonDynamoDBClient(new
AmazonDynamoDBConfig(){ServiceURL = "http://localhost:8000"});
            context = new DynamoDBContext(client, config);
        }

        public async Task Save<T>(T entity)
        {
            await context.SaveAsync<T>(entity, new
DynamoDBOperationConfig(){IgnoreNullValues = false});
        }

        public async Task<IEnumerable<T>> All<T>(string paginationToken = "")
        {
            var table = context.GetTargetTable<T>();

            var scanOps = new ScanOperationConfig();

            if (!string.IsNullOrEmpty(paginationToken))
            {
                scanOps.PaginationToken = paginationToken;
            }

            var results = table.Scan(scanOps);
            List<Document> data = await results.GetNextSetAsync();

            IEnumerable<T> readers = context.FromDocuments<T>(data);


            return readers;
        }


        public async Task<T> Single<T>(string partitionKey, object sortKey =
null)
        {
            return
                await context.LoadAsync<T>(partitionKey,
                    sortKey);
//.QueryAsync<Reader>(readerId.ToString()).GetRemainingAsync();
        }
    }
}
```

```
using System;
using System.Globalization;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using MovieWebApi.DataAdapters;
using MovieWebApi.Models;
using MovieWebApi.Repositories;
using MovieWebApi.Services;

namespace MovieWebApi.Controllers
{
    [Authorize]
    [Route("[controller]")]
    public class UserController : ControllerBase
    {
        private readonly UserInfoRepository userInfoRepository;
        private readonly UserManager<IdentityUser> userManager;
        private readonly EmailSenderService emailSenderService;
        private readonly SessionRepository sessionRepository;
        private readonly SchemeRepository schemeRepository;
        private readonly ImdbApiDataAdapter imdbApiDataAdapter;

        public UserController(UserInfoRepository userInfoRepository,
UserManager<IdentityUser> userManager,
            EmailSenderService emailSenderService, SessionRepository
sessionRepository, SchemeRepository schemeRepository,
            ImdbApiDataAdapter imdbApiDataAdapter)
        {
            this.userInfoRepository = userInfoRepository;
            this.userManager = userManager;
            this.emailSenderService = emailSenderService;
            this.sessionRepository = sessionRepository;
            this.schemeRepository = schemeRepository;
            this.imdbApiDataAdapter = imdbApiDataAdapter;
        }

        [HttpGet]
        public async Task<UserInfo> GetUserInfo()
        {
            var identityUser =
                await userManager.FindByEmailAsync(User.Claims.Single(claim
=> claim.Type == "email")
                    .Value);
            var rolesAsync = await userManager.GetRolesAsync(identityUser);
            var userInfo = await
userInfoRepository.Single(User.Claims.First(claim => claim.Type ==
"email").Value);
            userInfo.Role = rolesAsync.SingleOrDefault();
            return userInfo;
        }

        [HttpPut]
        public async Task<bool> SaveUserInfo([FromBody] UserInfo userInfo)
        {
            var email = User.Claims.First(claim => claim.Type ==
"email").Value;
            userInfo.UserId = email;
            return await userInfoRepository.Save(userInfo);
        }
```

```
        [HttpPost]
        [Route("session/{sessionId}/sendSelectedSeats/")]
        public async Task SaveSession(string sessionId, [FromBody] UserInfo
userInfo)
        {
            var email = User.Claims.First(claim => claim.Type ==
"email").Value;
            userInfo.UserId = email;
            var movieSession = await sessionRepository.Single(sessionId);
            var scheme = await
schemeRepository.Single(movieSession.SchemeId);
            var titleId = movieSession.MovieId.IndexOf("tt",
StringComparison.Ordinal);
            string result = new string(movieSession.MovieId[(titleId-1)..]
                .SkipWhile(x => !char.IsDigit(x))
                .TakeWhile(char.IsDigit)
                .ToArray());
            var movie = await imdbApiDataAdapter.GetMovieById("tt"+result);
            var seatsChosenEmailNotificationTemplate = new
SeatsChosenEmailNotificationTemplate()
            {
                Email = email,
                SelectedSeats = userInfo.SelectedSeats[sessionId],
                MovieName = movie.Title,
                SchemeName = scheme.Name,
                StartDate = movieSession.StartDate.ToString("f",
CultureInfo.GetCultureInfo("uk"))
            };
            await
emailSenderService.SendSeatsChosenNotification(seatsChosenEmailNotificationTe
mplate);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using Microsoft.AspNetCore.Mvc;
using MovieWebApi.Models;
using MovieWebApi.Repositories;

namespace MovieWebApi.Controllers
{
    [Route("[controller]")]
    // [Authorize]
    public class SessionController : ControllerBase
    {
        private readonly SessionRepository sessionRepository;

        public SessionController(SessionRepository sessionRepository)
        {
            this.sessionRepository = sessionRepository;
        }

        [Route("{id}")]
        [HttpGet]
        public async Task<MovieSession> Single(string id)
        {
            return await this.sessionRepository.Single(id);
        }
```

```
        [HttpPost]
        public async Task<IActionResult> Create([FromBody] MovieSession
movieSession)
        {
            movieSession.Id = Guid.NewGuid().ToString();
            await this.sessionRepository.Create(movieSession);
            return Ok();
        }

        [Route("{id}")]
        [HttpPut]
        public async Task<IActionResult> Update(string id, [FromBody]
MovieSession movieSession)
        {
            if (movieSession.Id != id)
            {
                return BadRequest("Id doesn't match");
            }
            await this.sessionRepository.Update(movieSession);
            return Ok();
        }

        [Route("/Movie/{id}/Sessions")]
        [HttpGet]
        public async Task<List<MovieSession>> ByMovie(string id)
        {
            var all = await this.sessionRepository.All();
            id = HttpUtility.UrlDecode(id);
            return all.Where(session => session.MovieId == id).ToList();
        }

        [HttpGet]
        public async Task<List<MovieSession>> All()
        {
            return await this.sessionRepository.All();
        }
    }
}
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using MovieWebApi.Models;
using MovieWebApi.Repositories;

namespace MovieWebApi.Controllers
{
    [Route("[controller]")]
    // [Authorize]
    public class SchemeController : ControllerBase
    {
        private readonly SchemeRepository schemeRepository;

        public SchemeController(SchemeRepository schemeRepository)
        {
            this.schemeRepository = schemeRepository;
        }

        [Route("{id}")]
        [HttpGet]
        public async Task<Scheme> Single(string id)
```

```
        {
            return await this.schemeRepository.Single(id);
        }

        [HttpPost]
        public async Task<IActionResult> Create([FromBody] Scheme scheme)
        {
            scheme.Id = Guid.NewGuid().ToString();
            await this.schemeRepository.Create(scheme);
            return Ok();
        }

        [Route("{id}")]
        [HttpPut]
        public async Task<IActionResult> Update(string id, [FromBody] Scheme
scheme)
        {
            if (scheme.Id != id)
            {
                return BadRequest("Id doesn't match");
            }
            await this.schemeRepository.Update(scheme);
            return Ok();
        }

        [HttpGet]
        public async Task<List<Scheme>> All()
        {
            return await this.schemeRepository.All();
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using MovieWebApi.DataAdapters;
using MovieWebApi.Models;

namespace MovieWebApi.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class MovieController : ControllerBase
    {
        private readonly ImdbApiDataAdapter _dataAdapter;

        public MovieController(ImdbApiDataAdapter dataAdapter)
        {
            _dataAdapter = dataAdapter;
        }

        [HttpGet]
        [Route("[action]/{keyword}")]
        public async Task<List<SearchResult>> SendSearch(string keyword)
        {
            return (await _dataAdapter.SearchByName(keyword)).FindAll(result
=> result.Id.Contains("title"));
        }

        [HttpGet]
        [Route("[action]/{id}")]
```

```
        public async Task<MovieDetails> GetMovieById(string id)
        {
            var titleId = id.IndexOf("tt", StringComparison.Ordinal);
            string result = new string(id.Substring(titleId-1)
                .SkipWhile(x => !char.IsDigit(x))
                .TakeWhile(char.IsDigit)
                .ToArray());
             Console.WriteLine("tt"+result);
            return (await _dataAdapter.GetMovieById("tt"+result));
        }
    }
}
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using MovieWebApi.Configuration;
using MovieWebApi.Models;
using MovieWebApi.Repositories;

namespace MovieWebApi.Controllers
{
    [Route("[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly JwtBearerTokenSettings jwtBearerTokenSettings;
        private readonly UserManager<IdentityUser> userManager;
        private readonly RoleManager<IdentityRole> roleManager;
        private readonly UserInfoRepository userInfoRepository;

        public AuthController(IOptions<JwtBearerTokenSettings>
jwtTokenOptions, UserManager<IdentityUser> userManager,
RoleManager<IdentityRole> roleManager
        , UserInfoRepository userInfoRepository)
        {
            this.jwtBearerTokenSettings = jwtTokenOptions.Value;
            this.userManager = userManager;
            this.roleManager = roleManager;
            this.userInfoRepository = userInfoRepository;
        }

        [HttpPost]
        [Route("Register")]
        public async Task<IActionResult> Register([FromBody] UserDetails
userDetails)
        {
            if (!ModelState.IsValid || userDetails == null)
            {
                return new BadRequestObjectResult(new { Message = "User
Registration Failed" });
            }

            var identityUser = new IdentityUser() { UserName =
userDetails.UserName, Email = userDetails.Email };
            var result = await userManager.CreateAsync(identityUser,
userDetails.Password);
```

```
            if (!result.Succeeded)
            {
                var dictionary = new ModelStateDictionary();
                foreach (IdentityError error in result.Errors)
                {
                    dictionary.AddModelError(error.Code, error.Description);
                }

                return new BadRequestObjectResult(new { Message = "User
Registration Failed", Errors = dictionary });
            }

            await userManager.AddToRoleAsync(identityUser, Roles.Customer);
            await userInfoRepository.Save(new UserInfo() { UserId =
userDetails.Email });
            var token = GenerateToken(identityUser);
            return Ok(new { Token = token, Message = "Success" });
        }

        [HttpPost]
        [Route("Login")]
        public async Task<IActionResult> Login([FromBody] LoginCredentials
credentials)
        {
            IdentityUser identityUser;

            if (!ModelState.IsValid
                || credentials == null
                || (identityUser = await ValidateUser(credentials)) == null)
            {
                return new BadRequestObjectResult(new { Message = "Login
failed" });
            }

            var token = GenerateToken(identityUser);
            return Ok(new { Token = token, Message = "Success" });
        }

        [HttpPost]
        [Route("Logout")]
        public async Task<IActionResult> Logout()
        {
            // Well, What do you want to do here ?
            // Wait for token to get expired OR
            // Maintain token cache and invalidate the tokens after logout
method is called
            return Ok(new { Token = "", Message = "Logged Out" });
        }

        private async Task<IdentityUser> ValidateUser(LoginCredentials
credentials)
        {
            var identityUser = await
userManager.FindByNameAsync(credentials.Username);
            if (identityUser != null)
            {
                var result =
userManager.PasswordHasher.VerifyHashedPassword(identityUser,
identityUser.PasswordHash,
                    credentials.Password);
                return result == PasswordVerificationResult.Failed ? null :
identityUser;
            }
```

```
            return null;
        }


        private object GenerateToken(IdentityUser identityUser)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key =
Encoding.ASCII.GetBytes(jwtBearerTokenSettings.SecretKey);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new(ClaimTypes.Name, identityUser.UserName.ToString()),
                    new(ClaimTypes.Email, identityUser.Email)
                }),

                Expires =
DateTime.UtcNow.AddSeconds(jwtBearerTokenSettings.ExpiryTimeInSeconds),
                SigningCredentials =
                    new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature),
                Audience = jwtBearerTokenSettings.Audience,
                Issuer = jwtBearerTokenSettings.Issuer
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
namespace MovieWebApi.Configuration
{
    public class JwtBearerTokenSettings
    {
        public string SecretKey { get; set; }
        public string Audience { get; set; }
        public string Issuer { get; set; }
        public int ExpiryTimeInSeconds { get; set; }
    }
}
```