МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Савченко А.С.

«___»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА СПЕЦІАЛІЗАЦІЄЮ «ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)»

Тема: «Web-сервіс для автоматизації та підтримки роботи кінотеатру»

Виконав: Озімай Дар'я Олександрівна

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер: Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕСИТЕТ

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Галузь 12 «Інформаційні технології»

Спеціальність 122 «Комп’ютерні науки»

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Савченко А.С.

“ ” _____2019р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Озімай Дар'ї Олександрівни

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи): «Web-сервіс для автоматизації та підтримки роботи кінотеатру» затверджена наказом ректора №2175/ст. від 25.10.2019р..
2. Термін виконання проекту (роботи): з 14.10.2019р. по 03.02.2020р.
3. Вихідні данні до проекту (роботи): готове рішення у вигляді web-сервісу.
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): детальне ознайомення з темою та постановка задач, вибір архітектури програмного забезпечення, проектування та розробка web-сервісу.
5. Перелік обов’язкового графічного матеріалу: схеми архітектур сервісу, скрsншоти структури XML, JSON, скріншоти виконання запитів до web-сервісу.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Аналіз літератури та джерел за темою дипломного проекту.	14.10.19р.– 20.10.19	
2.	Розроблення та затвердження плану дипломного проекту.	21.10.19– 22.10.19	
3.	Проведення консультації з науковим керівником щодо створення першого розділу.	23.10.19 – 27.10.19	
4.	Розробка розділу 1: Види архітектур програмного забезпечення.	30.10.19 – 10.11.19	
5.	Розробка розділу 2: Обмін даними з web-сервісами.	11.11.19 – 22.11.19	
6.	Розробка розділу 3: Проектування системи.	23.11.19 – 08.12.19	
7	Розробка розділу 4: Застосування технологій.	09.12.19 – 22.12.19	
8.	Висновки та оформлення пояснювальної записки дипломного проекту.	25.12.19 – 29.12.19	
9.	Підписання необхідних документів у встановленому порядку.	15.01.20-19.01.20	
10.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	22.01.20 – 31.01.20	

7. Дата видачі завдання: 14.10.2019р.

Керівник дипломного проекту _____ Холявкіна Т.В.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Озімай Д.О.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту роботи «Web-сервіс для автоматизації та підтримки роботи кінотеатру» викладена на с. 123, містить рис. 17, табл. 1, літературних джерел 7.

Ключові слова: ІНТЕРНЕТ, WEB-СЕРВІС, АРХІТЕКТУРА, HTTP,

***Об'єкт дослідження*:** WEB-сервіс.

Предмет дослідження: розробка WEB-сервісу.

Мета роботи: порівняти види архітектури програмного забезпечення та реалізувати обраний.

Методи дослідження: порівняння видів архітектури програмного забезпечення, аналіз протоколу обміну і передачі даних, формалізація однієї з архітектур.

Отримані результати: у процесі створення дипломної роботи був спроектован та розроблен web-сервіс для автоматизації бізнес-процесів кінотеатру.

Результати дипломної роботи використовується на підприємстві.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 7

ВСТУП 8

РОЗДІЛ 1. ВИДИ АРХІТЕКТУР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 9

1. Архітектура програмного забезпечення 9

1.1. Клієнт-серверна архітектура 10

1.1.1. Обов'язки та взаємодія 11

1.1.2. Трирівнева архітектура 11

1.2. Аспектно-орієнтована архітектура 12

1.3. Сервісно-орієнтована архітектура 14

1.4. REST 16

1.4.1. Клієнт-сервер 17

1.4.2. Відсутність стану 18

1.4.3. Кешування 18

1.4.4. Шари абстракції 19

1.4.5. Елементи даних 19

1.4.6. Ресурс 20

1.4.7. Ідентифікатор ресурсу 20

1.4.8. Представлення 21

1.4.9. Конектори 21

1.4.10. Семантика протоколу HTTP 23

ВИСНОВОК ДО РОЗДІЛУ 1 25

РОЗДІЛ 2. ОБМІН ДАНИМИ З WEB-СЕРВІСОМ 26

2.1. Протокол обміну і передачі даних 26

2.1.1. SOAP 28

2.1.2. XML-RPC 29

2.1.3. WSDL 29
2.1.4. UDDI 32
2.2. WEB-інтерфейс 34
2.2.1. XML 35
2.2.2. JSON 36
ВИСНОВОК ДО РОЗДІЛУ 2 39
РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ 40
3.1. Діаграми варіантів використання (use case diagrams) 41
3.1.1. Аналіз діаграми використання 41
3.2. Створення діаграми класів 45
ВИСНОВОК ДО РОЗДІЛУ 3 49
РОЗДІЛ 4. ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ 50
4.1. .Net Core 50
4.2. ASP.Net Core 51
4.3. Архітектура сервісу 52
4.3.1. Монолітна архітектура 52
4.3.2. Багаторівнева архітектура 53
4.3.3. Onion-архітектура 54
4.4. Entity Framework 56
ВИСНОВОК ДО РОЗДІЛУ 4 57
ВИСНОВКИ 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59
Додаток А 60
Додаток Б 61

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних;

ІТ – інформаційні технології;

API – програмний інтерфейс програми;

WEB – всесвітня павутина;

JSON –JavaScript Object Notation;

HTTP – HyperText Transfer Protocol.

REST - Representational State Transfer\

CMS - Content Management System

SOA – Service-oriented architecture

ПЗ – програмне забезпечення

RPC - Remote Procedure Call

ВСТУП

Нещодавно всі ми користувалися додатками, які доводилося встановлювати на свої пристрої. Зараз, зі швидким розвитком інтернету, дуже популярні у використанні різні веб сайти або додатки. В основному зараз активно використовуються веб сервіси. Це дуже зручно, тому що дає можливість взаємодії такого типу програм не тільки між собою, а й з іншими сторонніми додатками за допомогою повідомлень, заснованих на певних протоколах і угодах.

Однією з переваг є те, що до web-сервісів ми маємо можливість звертатись з будь якої точки світу, достатньо мати інтернет і ім’я хоста, на який потрібно звертатись.

Також, завдяки тому, що ми маємо один веб сервіс, що працює з даними, то для одного web-сервісу ми маємо можливість створювати різні інтерфейси, та надавати доступ стороннім програмам.

На сьогодні, для покращення роботи підприємств людство намагається автоматизувати якомога більше бізнес-процесів. І це дуже впливає на розвиток web програмування. З’являються нові архітектурні стилі; найбільшу популярність набрав стиль під назвою REST.

РОЗДІЛ 1

ВИДИ АРХІТЕКТУР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Архітектура програмного забезпечення

Архітектура програмного забезпечення - це процес перетворення таких характеристик програмного забезпечення, як: гнучкість, масштабованість, можливість реалізації, багаторазовість використання і безпеку - в структуроване рішення, яке відповідає як технічним, так і бізнес вимогам. Звідси вже є важливим дізнатися які характеристики програмного забезпечення можуть вплинути на проектування архітектури програмного забезпечення. Крім технічних особливостей, також існує безліч параметрів, які в основному відповідають вимогам бізнесу або функціональності.

Сама архітектура включає в себе:

- обирання елементів структури і їх інтерфейсів, з яких складена обрана система, а також поведінка в рамках співпраці структурних елементів;
- об'єднання обраних елементів і їх поведінки у все більш крупні системи;
- архітектурний стиль, який направляє всю організацію - всі елементи, інтерфейси, їх співпраця і з'єднання

Високий рівень абстракції визначає спосіб реалізації вибору архітектурних вимог. Архітектура значно впливає і на зручність використання і ефективність ПЗ, які визначаються реалізацією окремих компонентів. Значно менше вплив архітектури на функціональність – зазвичай для реалізації заданої функціональності можна використовувати

різні архітектури. Тому вибір між тією або іншою архітектурою визначається насамперед саме нефункціональними вимогами і необхідними властивостями ПЗ в аспектах зручності супроводу та переносимості. При цьому для побудови гарної архітектури треба враховувати можливі протиріччя між вимогами до різних характеристик і вміти вибирати компромісні рішення, що дають прийнятні значення за всіма показниками. Є багато поширених способів розробки програмних модулів та їх зв’язків, розглянемо найвідоміші.

1.1.Клієнт-серверна архітектура

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери і клієнти є незалежними один від одного і функціонують паралельно. Також немає жорсткої прив'язки клієнтів до серверів. Одна за типових ситуацій, коли один сервер одночасно обробляє запити від різних клієнтів; також, клієнт може звертатися то до одного сервера, то до іншого. Клієнти повинні знати про доступні сервери, але можуть не уявляти про існування інших клієнтів.

Дуже важливо мати уявлення, що розглядається як «клієнт». Це може бути клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів, а може бути клієнтське та серверне програмне забезпечення.

Клієнти та сервери у першу чергу це програмні модулі. Скоріше за все їх розміщують на різних комп'ютерах, але буває, коли обидві програми фізично розміщуються на одній машині. У такій ситуації сервер часто іменують локальним.

1.1.1. Обов'язки та взаємодія

Перш за все модель клієнт-серверної взаємодії визначається розподілом обов'язків між клієнтом та сервером. Можна виділити три рівні операцій:

- представлення даних, що по суті відповідає за введення керуючих команд і представлення даних користувачу та являє собою інтерфейс користувача;
- прикладний рівень, реалізує основну логіку застосування,яка здійснює необхідну обробку інформації;
- рівень управління даними, що відповідає за зберігання даних та контроль доступу до них.[6]

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від розподілення наведених вище функцій, розрізняють:

- модель тонкого клієнта, тобто вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська частина забезпечує тільки функції рівня представлення;
- модель товстого клієнта, де сервер тільки управляє даними, а на стороні клієнта вже зосереджена обробка інформації та інтерфейс користувача. Така модель часто іменують як пристрій з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

1.1.2. Трирівнева архітектура

Також є трирівнева клієнт-серверна архітектура, яка почала свій розвиток з середини 90-х років. Вона передбачає відокремлення прикладного

рівня від управління даними. Виділяють окремий програмний рівень,де зосереджується прикладна логіка застосунку. Програми вже проміжного рівня можуть працювати під управлінням спеціальних серверів застосунків, але запуск саме таких програм може бути здійснений і під управлінням звичайного веб-сервера. А управління даними вже здійснюється сервером даних.

Стандартне програмне забезпечення що використовує людина для роботи з системою — це звичайний браузер. Для користувача це зручно, тому що такий підхід позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все ж таки може бути). Але для користувача слід надати в розпорядженні інтерфейс, який допоможе йому взаємодіяти з системою і формувати запити до неї. Цей інтерфейс визначають форми, що розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-клієнт формує запит та надсилає його до сервера, що здійснює обробку. Якщо є така необхідність, сервер викликає програмні модулі серверу, що забезпечують обробку запиту і на разі потреби звертаються до сервера даних. З даними, що зберігаються в системі та складають її інформаційну основу здійснює операції з даними вже сервер даних.. Також, він має можливість здійснювати вибірку з інформаційної бази відповідно до запиту та для подальшої обробки передати її модулю проміжного рівня. Інформація, з якою працює сервер даних, найчастіше організована як реляційна база даних.

Відомо, що серверні модулі проміжного рівня і веб-сервер розміщуються на одному комп'ютері, хоча і відокремлюють їх як логічно незалежні програмні модулі.

1.2.Аспектно-орієнтована

Існує функціональність, яку неможливо виділити за допомогою:

- функції;
- модуля;
- класа.

Її називають наскрізний (від англ. scattered - розкиданий або англ. tangled - переплетений), так як її реалізація розподілена по різним модулям програми. Наскрізна функціональність призводить до розосередженому і заплутаному коду, складного для розуміння і супроводу.

Основним завданням аспектно-орієнтованого програмування є модулярізації наскрізний функціональності, виділення її в аспекти. Для цього мови, підтримують концепцію аспектно-орієнтованого програмування, реалізують наступні засоби для виділення наскрізний функціональності:

- аспект (aspect) - модуль або клас, який реалізує наскрізну функціональність. Аспект змінює поведінку іншого коду, застосовуючи пораду в точках з'єднання, визначених деяким зрізом. Так само аспект може використовуватися для впровадження функціональності;
- порада (advice) - додаткова логіка - код, який повинен бути викликаний з точки з'єднання. Порада може бути виконена до, після або замість точки з'єднання;
- точка з'єднання (join point) – точка, яка віиконується в програмі (виклик методу, створення об'єкта, звернення до змінної), де слід застосувати пораду;
- зріз (pointcut) - набір точок з'єднання. Зріз визначає, чи підходить дана точка з'єднання до заданої поради;
- впровадження (introduction) - зміна структури класу і / або зміна ієрархії успадкування для додавання функціональності аспекту в інший код;
- мета (target) - об'єкт, до якого будуть застосовуватися поради;

- переплетення (weaving) - зв'язування об'єктів з відповідними аспектами (можливо на етапі компіляції, завантаження або виконання програми).

Для програми, написаної в парадигмі ООП, будь-яка функціональність, по якій не була проведена декомпозиція, є наскрізною.

1.3.Сервісно-орієнтована архітектура

Сервіс-орієнтована архітектура (service-oriented architecture, SOA) придумана в кінці 1980-х. Вона бере свій початок в ідеях, викладених в CORBA, DCOM, DCE і інших документах. Про SOA написано багато, є кілька її реалізацій. Але, по суті, SOA можна звести до кількох ідей, причому архітектура не диктує способи їх реалізації:

- Сполучуваність додатків, орієнтованих на користувачів.
- Багаторазове використання бізнес-сервісів.
- Незалежність від набору технологій.
- Автономність (незалежні еволюція, масштабованість і развёртиваемость).

Ця архітекту́ра уявляє собою архітектурний шаблон ПЗ. За допомогою цієї архітктури використовується модульний підхід до розробки ПЗ. Цей підхід заснований на використанні розподілених, слабко пов'язаних замінних компонентів. Дана архітектура наповнена стандартизованими інтерфейсами, завдяки яким можна взаємодіяти за стандартизованими протоколами.

Найчастіше становлення того чи іншого підходу супроводжується появою хибних трактувань. Не оминула це і сервіс-орієнтовану архітектуру:

- SOA не являються чимось новим: IT компанії успішно створювали і розгортали застосунки, що підтримують сервіс-орієнтовану архітектуру, вже багато років — задовго до появи XML і Web-сервісів.

- SOA — не є технологія, це скоріше спосіб проектування і організації інформаційної архітектури та бізнес-функціональності.[6]

Будування застосунків згідно з принципами SOA не значить, що достатньо купити найновіші продукти, які реалізують XML і Web-сервіси.

На основі того, що описано вище можна дати наступне визначення SOA: це парадигма, що призначена для проектування, розробки та управління в обчислювальному середовищі дискретних одиниць логіки. Даний підхід вимагає від розробників проектування застосунків як набору сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні вийти за межі своїх застосунків і подумати, як скористатися вже наявними сервісами, або вивчити, як їх сервіси можуть бути використані їх колегами.[1]

SOA підштовхує до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови застосунків за допомогою зв'язування сервісів, а не за допомогою написання нового програмного коду.[3] У цьому випадку, при належному проектуванні, застосування повідомлень дозволяє компаніям своєчасно реагувати на зміну ринкових умов — настроювати процес обміну повідомленнями, а не розробляти нові програми.

Зовсім нещодавно синонімом «Web-сервіс» був термін «сервіс-орієнтована архітектура». Вона являє собою термін, що з'явився для опису виконуваних компонентів: Web-сервіси, які можуть викликатися іншими програмами і виступають клієнтами або споживачами цих сервісів. Ці сервіси можуть бути повністю сучасними — або навіть застарілими — прикладними програмами, які можна активізувати як чорний ящик. Але від розробника не потрібно знати, як працює програма, достатньо розуміти і знати, які вхідні та вихідні дані повинні бути, та як викликати ці програми для виконання. Загалом SOA припускає наявність трьох основних учасників: постачальника сервісу, споживача сервісу та реєстру сервісів. Взаємодія між цими

учасниками виглядає досить просто: постачальник реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

Сервіс буде працювати коректно, якщо дотримуватись угоди для звернення до сервісу. Інтерфейс від платформи ніяким чином не повинен залежати. Сервіс орієнтована архітектура надає такі дві основні можливості: додавання сервісів та їх модернізацію. Споживач і постачальник сервісу спілкуються за допомогою повідомлень, більше ніяк вони не пов'язані. На основі того, що інтерфейс залежати від платформи не має права, то і технологія, що використовується для визначення повідомлень, так само не повинна залежати від платформи. Таким чином XML-документи виступають у ролі повідомлення, що відповідають схемі XML.[3]

Існуть відкриті стандарти, за допомогою яких можна ознайомитись з описом XML і Web-сервісу перед застосовуванням сервісно-орієнтованої архітектури. Як відомо, Web-сервіси базуються на таких протоколах: HTTP, XML, UDDI, WSDL і SOAP, які реалізують основні вимоги SOA:

1. динамічний виклик(UDDI, WSDL і SOAP);
2. не залежить від платформи (XML).

1.4.REST

REST (Representational state transfer) - це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, Як правило, він застосовується для побудови веб-служб. Термін REST був введений у 2000 році Роєм Філдінгом, одним із творців протоколу HTTP. Системи, що підтримують REST, називаються RESTful-системами. Також Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.[6]

У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним

ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат. Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Такий підхід дозволяє системі еволюціонувати з новими вимогами та забезпечує її масштабовність.

Антиподом REST є підхід, заснований на виклику віддалених процедур. Але з підходом RPC можно використовувати лише невелику кількість мережевих ресурсів, але використовуючи велику кількість методів і зі складним протоколом. Складність протоколу і кількість методів дуже обмежені з підходом REST. Це призводить до того, що кількість окремих ресурсів має бути великою.

REST — являє собою архітектуру для розподілених гіпертекстових систем. REST, як будь-який архітектурний стиль, має ряду архітектурних обмежень. Цей стиль можна іменувати гібридним, тому що він комбінує обмеження з інших архітектурних стилів.

1.4.1. Клієнт-сервер

Клієнт-серверна архітектура звісно перша з якої він успадковує обмеження. Ці обмеження вимагають між компонентами поділити відповідальність. Тобто окремо компонент, що відповідає за зберігання та оновлення даних, і окремо компонент, який відображає дані на інтерфейс користувача та реагує на дії з цим інтерфейсом. Саме це обмеження дозволяє модернізувати компоненти незалежно.

1.4.2. Відсутність стану

Відсутність стану є наступним обмеженням , тому що взаємодія між клієнтом і сервером не має стану. Сервер не знає нічого з попереднього запиту, тому кожен запит повинен містити всю необхідну інформацію для обробки. Але це не означає, що стану немає. Це лише стверджує, що система не має ніякої інформації щодо стану клієнта.. Наприклад, запитуючи головну сторінку сайту ми отримаємо відповідь і сервер забуває про клієнта. Ця сторінка може бути відкрита дуже довго, перш ніж відправиться нове посилання, і тільки тоді сервер наддасть відповідь. А у той же час сервер може нісого н робити або відповідати на запити інших клієнтів. Для нас, як для клієнтів це не має значення.

Таким чином, навіть при аутентифікації, дані про стан сесії зберігаються клієнт у себе і використовує ці дані з кожним запитом. Саме завдя цьому покращує масштабовність, тому що сервер може звільнити всі ресурси без жодного ризику втратити цінну інформацію.[6] Також спрощується моніторинг і зневадження. Розбираючись, що відбувається в певному запиті, достатньо подивитись лише тільки на той запит. Також помилка в одному запиті не зачіпає інші, саме тому покращується надійність.

Але є один недолік цього обмеження. Продуктивність системи знижується, тому що потрібно у кожний запит додавати дані сессії з клієнта. Так само важче підтримувати збереження стану на різних клієнтах. Реалізація клієнтів може бути різною, але середовище сервера повністю під контролем розробника.

1.4.3. Кешування

Додатковим обмеженням стилю REST є те, що системи, написані в цьому стилі, повинні підтримувати кешування, тобто дані, які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність, уникаючи

зайвих запитів, але також зменшує надійність системи, через те, що дані в кеші можуть бути застарілими.

Рання архітектура веб, створена Тімом Бернерсом-Лі відповідала цим трьом обмеженням — клієнт-сервер без стану з підтримкою кешування. Проте стиль REST додає ще додаткові обмеження.

Однорідний інтерфейс

Всі компоненти в архітектурі REST підтримують однорідний інтерфейс. Це зменшує зв'язність між компонентами і сервісами які вони надають і дозволяє нескладно змінювати компоненти при потребі.[5] Це досягається кількома точнішими обмеженнями:

- ідентифікація ресурсів
- маніпуляція ресурсами через представлення
- самоописові повідомлення
- гіпермедіа як рушій стану застосунку

1.4.4. Шари абстракції

Наступним обмеженням для REST є поділ на шари абстракції. Кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмежнення знання системи одним шаром зменшує складність компонентів.

1.4.5. Елементи даних

Компоненти REST системи спілкуються, передаючи один одному представлення ресурсу в форматі, що обирається з оновлюваного набору стандартних форматів даних. Формат обирається динамічно відповідно до бажань компонента-клієнта і можливостей сервера.[6] Чи представлення має той самий формат, що й сам ресурс, чи є результатом якогось перетворення — це деталь реалізації, яка ховається за інтерфейсом.

1.4.6. Ресурс

Ресурс — це ключовий елемент даних в REST. Ресурсом може бути що завгодно що можна назвати: якийсь документ (наприклад зображення), динамічне значення (наприклад погода у Львові), щось з реального світу (наприклад працівник компанії). Але якщо точніше, то ресурс R — це функція приналежності Mr(t) що відображає моменти в часі на множину однотипних сутностей чи значень. Множина може бути порожньою, тобто REST дозволяє посилання на якийсь об'єкт якого ще не існує.

Ресурс може бути динамічним, наприклад ресурс «стаття про REST у вікіпедії» час від часу оновлює свій вміст, а може бути статичним, і після появи ніколи не змінювати свого значення, наприклад. У REST такі два ресурси вважаються різними, хоча в певний момент часу вони можуть вказувати на одну й ту ж сутність. Єдине що важливо — семантика відображення імені ресурсу на його вміст.

1.4.7. Ідентифікатор ресурсу

Для того, щоб посилатись на ресурси, використовуються ідентифікатори ресурсів. Компонент, який надав ресурсу ідентифікатор і дозволяє звертатись до нього за цим ідентифікатором, відповідає за збереження функції приналежності незмінною. Якість ідентифікатора залежить від якості компонента, який цей ідентифікатор надає, тому деякі ідентифікатори стають «мертвими посиланнями», коли інформацію переміщують або знищують.

Приклади ідентифікаторів ресурсу: URL, URN.

1.4.8. Представлення

Представлення (англ. representation) — це послідовність байтів та метадані представлення, для того щоб описати ці байти. Часто, представлення називають документом, файлом, повідомленням HTTP тощо.

Приклади представлення: фотографія JPEG, документ HTML. Приклад метаданих представлення — тип медіа, час останньої зміни.

Метадані також можуть бути не лише в представлення ресурсу, а й в самого ресурсу. Прикладами метаданих ресурсу є посилання на джерело, заголовок HTTP vary.

Контрольні дані в представленні описують ціль повідомлення між компонентами, наприклад прохання про дію (створити, змінити видалити ресурс), або значення відповіді (наприклад поточний стан ресурсу, чи значення якогось іншого ресурсу, наприклад опис помилки).

Також, контрольні дані, включені в запити чи відповіді, можуть керувати поведінкою кеша. Прикладом таких контрольних даних можуть бути заголовки HTTP if-modified-since та cache-control.

Формат даних представлення називають типом медіа (англ. media type). Одні типи медіа краще підходять для автоматичної обробки, інші — для того щоб бути показаними користувачу. Композитні типи медіа можуть використовуватись для того, аби поєднати кілька видів представлення в одному.

Від формату даних дуже залежить латентність застосунку, яку сприймає користувач. Наприклад, браузер може почати показувати сторінку ще до того, як завантажиться ввесь HTML, це збільшує видиму швидкість роботи.

1.4.9. Конектори

Конектори надають інтерфейс для комунікації компонентів, приховуючи реалізації ресурсів та механізм комунікації.

Конектори подібні на віддалений виклик процедур, але з певними нюансами щодо передачі параметрів та результату виклику. Параметри складаються з ідентифікатора ресурсу, контрольних даних та необов'язково, представлення. Результат — з контрольних даних відповіді і представлення. Можна абстрагуватись і вважати такий виклик синхронним, але насправді передача даних відбувається потоково, тому обробку даних можна починати ще до того як отримані всі дані, таким чином зменшуючи латентність.

Двома найважливішими типами конекторів є клієнт і сервер. Відмінністю між ними є те що клієнт ініціює запит, в той час як сервер очікує запитів і відповідає на них даючи доступ до своїх сервісів. Компонент може містити одночасно як серверні так і клієнтські конектори.

Додатковим типом конектора є кеш. Кеш може бути як клієнтським, для уникнення зайвих запитів, так і серверним — для уникнення зайвого обчислення відповіді на запит. Тому що інтерфейс однорідний, кеш легко може дізнатись чи запит можна кешувати. За замовчуванням, відповідь на запит отримання ресурсу можна кешувати, а запити зміни ресурсу — ні. Проте ці замовчування можна перевантажити контрольними даними.

Резолвер (англ. resolver) — це ще один тип конектора, який перетворює ідентифікатори ресурсів в інформацію про мережеві адреси необхідну компонентам щоб отримати цей ресурс. Наприклад в URI міститься доменне ім'я, і для доступу до відповідного домена, потрібно дізнатись в DNS-сервера адресу. Тому в цьому випадку система DNS грає роль резолвера. Використання одного чи кількох резолверів може збільшити життєздатність ідентифікатора ресурсу, через те що він не вказує напряму на фізичне розташування ресурсу яке може змінитись.

Останньою формою конектора є тунель, який просто проводить запити через межу системи, наприклад, через фаєрвол. Причиною, через яку тунелі включено до архітектури REST, а не закрито абстракцією мережі, є те, що певні компоненти можуть перетворюватись на тунелі за запитом. Наприклад тунель HTTP активується при отриманні запиту з методом CONNECT.

1.4.10. Семантика протоколу HTTP

Типове використання HTTP методів:

Хоча ресурс може бути яким завгодно, дії, які можна виконувати над ресурсом, визначаються повідомленнями, які визначено стандартним протоколом. В системі WWW цей протокол — HTTP, але існують REST-архітектури на основі й інших протоколів.

Стандарт HTTP визначає 8 типів повідомлень.

Найчастіше використовують 4 з них:

GET — отримати представлення ресурсу

DELETE — знищити ресурс

POST — створити новий ресурс на місці даного використавши передане представлення

PUT — замінити стан поточного ресурсу станом, що описується переданим представленням

Ці використовуються, щоб дослідити API:

HEAD — отримати заголовки, які б відсилались разом з представленням цього ресурсу, але не саме представлення.

OPTIONS — визначити список методів, на які цей ресурс відповідає.

Інші два методи, CONNECT та TRACE, використовуються лише для HTTP-проксі.

Існує також дев'ятий метод, щоправда, описаний не в HTTP, а в додатку RFC 5789:

PATCH — змінити лише частину цього ресурсу на основі даного представлення. Якщо якась частина ресурсу не згадується в переданому представленні — не чіпати її. Це знижує кількість інформації, яку потрібно передавати.

Ще два методи описуються в пропозиції до стандарту «Internet-Draft „snell-link-method“»:

LINK — прив'язати певний ресурс до цього

UNLINK — відв'язати ресурс від цього

Методи GET, PUT та DELETE — ідемпотентні, що означає, що незалежно від того, скільки разів ви виконаєте операцію, яку вони просять, — ви отримаєте той самий результат. Звісно, спершу DELETE поверне 204 No Content, а потім 404 Not Found, але ресурсу не буде, що після одного видалення, що після десяти. Ідемпотентність є дуже важливою в мережі, де ви не знаєте, чи досяг запит успіху, і, не отримавши відповіді, надсилаєте його ще раз. POST — не ідемпотентний, тобто, відправивши POST на створення повідомлення кілька разів, ви отримаєте кілька повідомлень.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було розглянуто чотири види архітектури програмного забезпечення.

Спираючись на данні вказані вище було обрано архітектуру REST. Вона є кращою завдяки наступним перевагам:

- надійність (за рахунок відсутності необхідності зберігати інформацію про стан клієнта, яка може бути втрачена)
- продуктивність (за рахунок використання кеша)
- масштабованість
- простота інтерфейсів
- портативність компонентів
- легкість внесення змін

Також включає в себе обмеження архітектури клієнт-сервер, що є домінуючою концепцією у створенні розподілених мережних застосунків.

РОЗДІЛ 2

ОБМІН ДАНИМИ З WEB-СЕРВІСОМ

Веб-сервіс – програмна система, що ідентифікується веб-адресою зі стандартизованими інтерфейсами. Веб-служби можуть взаємодіяти одна з одною і зі сторонніми програмами за допомогою повідомлень, заснованих на певних протоколах.[3] У побуті веб-сервісами називають послуги, що надаються в Інтернеті. Ідея веб-сервісу полягала в створенні такого RPC, який буде поміщатись в HTTP пакети.

2.1. Протокол обміну і передачі даних

Web-сервіси використовують XML для обміну даними між додатками, незалежно від використання операційної системи, апаратної платформи і розробника.

Як зображено на рис. 2.1, можна виділити три інстанції, які взаємодіють в рамках web-служби. Їх назви можна перевести як:

- замовник (service requester);
- виконавець (service provider);
- каталог (service broker);

Коли служба розроблена, виконавець реєструє її в каталозі, де її можуть знайти потенційні замовники. Замовник, знайшовши в каталозі відповідну службу, імпортує її звідти WSDL-специфікацію і розробляє відповідно до неї своє програмне забезпечення. WSDL описує формат запитів і відповідей, якими обмінюються замовник і виконавець в процесі роботи.

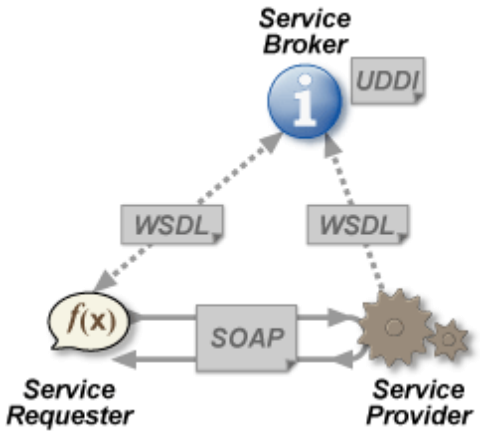

Рис. 2.1 Web-сервіс

Для забезпечення взаємодії використовуються такі нормативні документи:

7. XML;
8. SOAP;
9. WSDL;
10. UDDI.

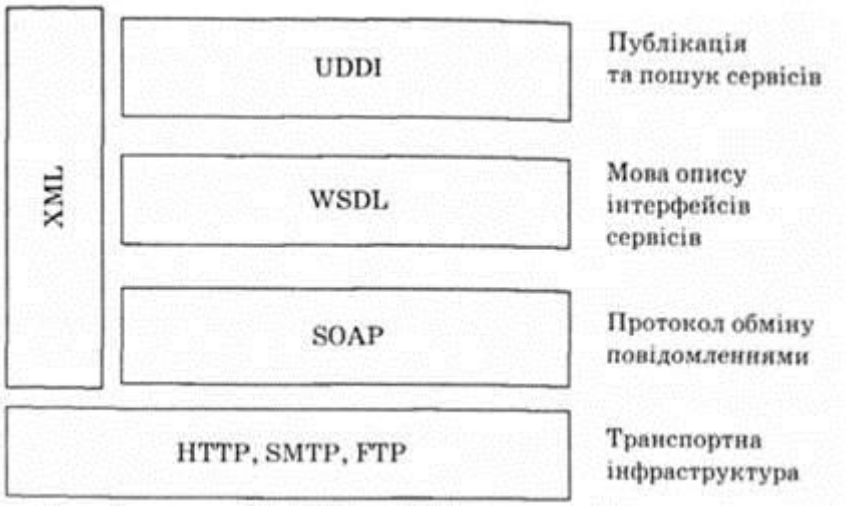

Рис. 2.2 Технології реалізації WEB-сервісів

Всі специфікації, які використовуються в технології, засновані на XML[5] і, відповідно, наслідують його переваги (структурованість, гнучкість тощо) і недоліки (громіздкість, повільність).

2.1.1. SOAP

SOAP – простий протокол доступу до об'єктів (компонентів розподіленої обчислювальної системи), заснований на обміні структурованими повідомленнями.[3] Як будь-який текстовий протокол, SOAP може використовуватися з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTPS та ін., Але частіше за все SOAP використовується поверх HTTP.

Всі повідомлення SOAP оформлюються у вигляді структури, званої конвертом (envelop), що включає наступні елементи[5]:

- ✓ Ідентифікатор повідомлення (локальне ім'я).
- ✓ Опціональний елемент Header (заголовок):
- ✓ Нуль або більше посилань на використовувані у просторі імен;
- ✓ Нуль або більше властивостей, доступних в цьому просторі імен.
- ✓ Обов'язковий елемент Body (тіло повідомлення)
- ✓ Нуль або більше посилань на використовувані у просторі імен;
- ✓ Дочірні елементи тексту повідомлення.

```
<soap: Envelope  xmlns: soap = "http://schemas.xmlsoap.org/soap/envelope/" >
   <soap: Body>
     <getProductDetails  xmlns = "http://warehouse.example.com/ws" >
       <productID > 12345 </ productID>
     </ getProductDetails>
   </ soap: Body>
</ soap: Envelope>
```

Рис. 2.3 Приклад SOAP-запиту на сервер інтернет-магазину

```
<soap: Envelope  xmlns: soap = "http://schemas.xmlsoap.org/soap/envelope/" >
   <soap: Body>
     <getProductDetailsResponse  xmlns = "http://warehouse.example.com/ws" >
       <getProductDetailsResult >
         <productID> 12345 </ productID>
         <productName> Стакан граненый </ productName>
         <опис> Стакан граненый. 250 мл. </ description>
         <price> 9.95 </ price>
         <currency>
             <code> 840 </ code>
             <alpha3> USD </ alpha3>
             <sign>

             <точність> 2 </ accuracy>
         </ currency>
         <inStock> true </ inStock>
       </ getProductDetailsResult>
     </ getProductDetailsResponse>
   </ soap: Body>
</ soap: Envelope>
```

Рис. 2.4 Приклад відповіді на SOAP-запит

2.1.2. XML-RPC

XML-RPC – дуже простий і ефективний протокол взаємодії веб-сервісів. Він не призначений для вирішення глобальних завдань, як SOAP, але широко використовується в багатьох веб-розробках[3].

XML-RPC – це специфікація і набір реалізацій, які дозволяють програмному забезпеченню, що працює на різних операційних системах і в різних умовах, викликати процедури через Інтернет. Це віддалений виклик процедури з використанням HTTP протоколу як транспорту та XML як способу кодування. XML-RPC розроблений настільки простим, наскільки це можливо для складних структур даних, що підлягають передачі, обробці та прийому.

2.1.3. WSDL

Мова опису веб-сервісів WSDL призначена для уніфікованого представлення зовнішніх інтерфейсів веб-служби.[5] Версія протоколу WSDL 2.0 має деякі відміни від попередньої версії.

Таблиця 2.1

Основні елементи протоколу WSDL

Елемент WSDL 1.1	Елемент WSDL 2.0	Короткий опис
PortType	Interface	Являє опис інтерфейсу веб-сервісу (список операцій і їх параметрів).
Service	Service	Список системних функцій.
Binding	Binding	Специфікує інтерфейси і задає параметри зв'язування з протоколом SOAP: стиль зв'язування (RPC/Document) і транспорт (SOAP). Ця секція доступна і для кожної з операцій.
Operation	Operation	Визначає операцію, подану веб-сервером. WSDL-операція – це аналог традиційних функцій і процедур.
Message	Не викор.	Повідомлення, пов'язане з певною операцією. Містить інформацію, необхідну для виконання даної операції. Кожне повідомлення може складатися з декількох логічних частин, що описують типи даних і імена атрибутів. У версії 2.0 було виключено, тому що була впроваджена підтримка XML Schema для всіх елементів.
Types	Types	Опис даних відповідно до XML Schema.

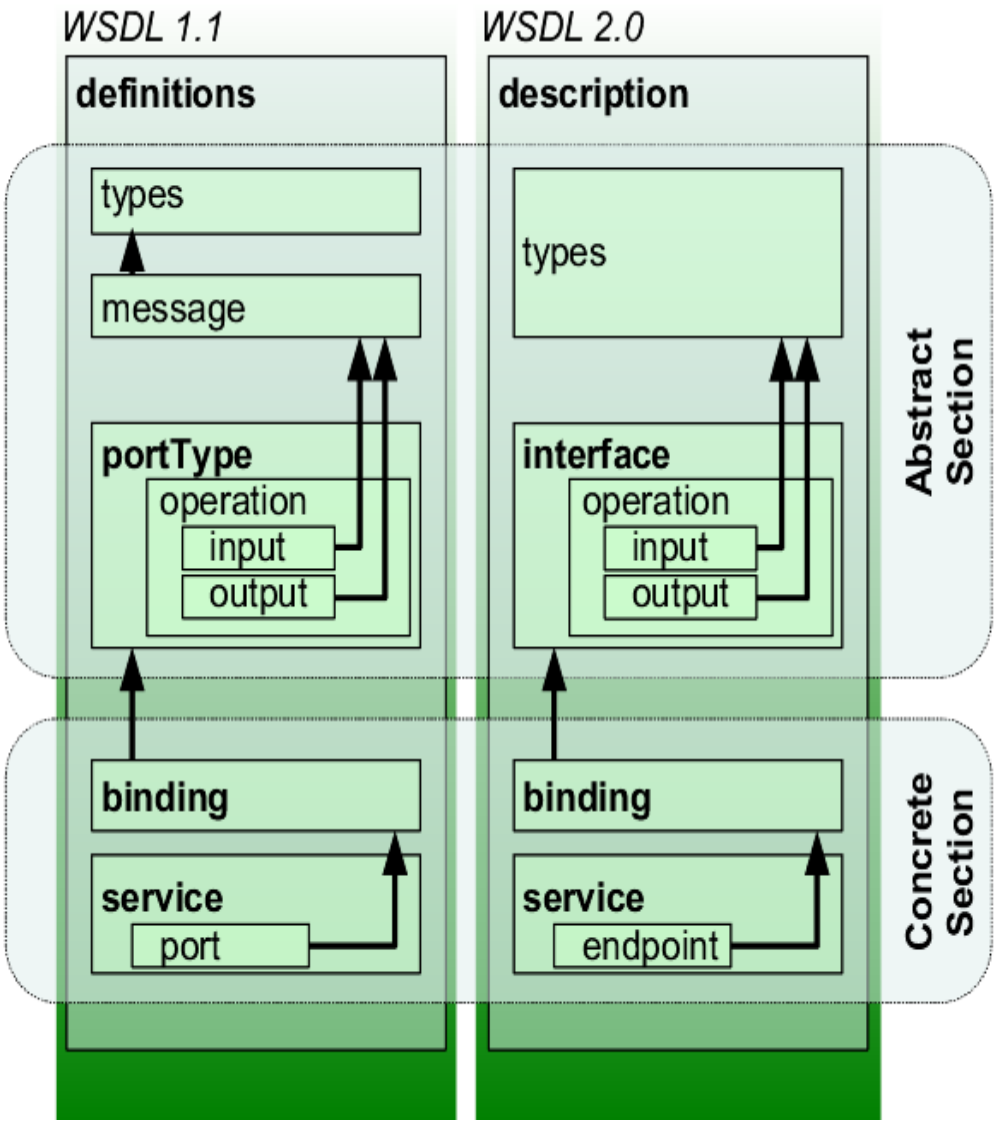

Рис. 2.5 Версії WSDL 1.1 і WSDL 2.0

У специфікації WSDL 1.1 було визначено 4 шаблону обміну повідомленнями (типи операцій):

- Однонаправлені операції (One-way): операція може приймати повідомлення, але не повертатиме відповідь.
- Запит-відповідь (Request-response): операція може прийняти запит і повинна повернути відповідь.
- Питання-відповідь (Solicit-response): операція може надіслати запит і буде чекати відповідь на нього.

- Повідомлення (Notification): операція може послати повідомлення, але не чекатиме відповідь.

У версії WSDL 2.0 ці шаблони змінені і розширені в сторону підтримки повідомлень про помилки (наприклад, шаблон Robust-in-only), але для забезпечення сумісності підтримуються типи WSDL 1.1

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
   <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
```

Рис. 2.6 Приклад WSDL

2.1.4. UDDI

UDDI – інструмент для розташування описів веб-сервісів (WSDL) для подальшого їх пошуку іншими організаціями та інтеграції в свої системи.[4]

UDDI це крос платформне програмне забезпечення, яке засноване на XML. UDDI є відкритим проектом, спонсором якого є OASIS, який дозволяє організаціям публікувати опис веб-сервісів для подальшого їх пошуку іншими організаціями та інтеграції в свої системи, а також визначати, як сервіси або програми взаємодіють через Internet.

UDDI був спочатку запропонований як web-сервіс основного стандарту.[4] Він призначений для опитування SOAP повідомленнями і для забезпечення доступу до WSDL документів, що описує прив'язки протоколів

і форматів повідомлень, необхідних для взаємодії з веб-послугами, переліченими в його каталозі.

Реєстрація UDDI складається з трьох компонентів:

- білі сторінки - адреса, контакти і відомі ідентифікатори;
- жовті сторінки - промислові категоризації на основі стандартної таксономії;
- зелені сторінки - технічна інформація про послуги, доступні в бізнесі.

2.1.4.1. Білі сторінки

Білі сторінки надають інформацію про постачальника послуг, наприклад, назва компанії, опис послуги (можливо, на декількох мовах). Використовуючи цю інформацію, можна знайти службу, про яку деякі відомості вже відомі (наприклад, розміщення сервісу, знайдене на ім'я провайдера) [4] .

Також передбачена можливість передачі контактної інформації (адреси, номери телефону та ін.).

2.1.4.2. Жовті сторінки

Жовті сторінки містять класифікацію служби або бізнесу, на основі стандартних таксономій. До них відносяться Standard Industrial Classification (SIC), North American Industry Classification System (NAICS) [4] або United Nations Standard Products and Services Code (UNSPSC) і географічні таксономії.

Так як один бізнес може надати ряд послуг, може бути кілька жовтих сторінок (кожна з яких описує послугу), пов'язаних з однією білою сторінкою (присутня загальна інформація про бізнес).

2.1.4.3. Зелені сторінки

Зелені сторінки використовуються для опису способу отримання доступу до WEB-служб та інформації про прив'язані послуги. Частину інформації, пов'язану з веб-сервісами – такі як адреса сервера та параметрів, а також посилання на специфікації інтерфейсів.[4] Іншу інформацію, не пов'язану безпосередньо з веб-службою – вона включає в себе електронну пошту, FTP, CORBA і телефоні номери для даного сервісу. Оскільки веб-служби можуть мати кілька прив'язок (як визначено в їх WSDL описі), служба може мати кілька зелених сторінок, тому як кожній прив'язці потрібно буде отримати доступ до різних сторінок.

2.2. WEB-інтерфейс

WEB-інтерфейс – WEB-сторінка або сукупність WEB-сторінок, які надають користувальницький інтерфейс для взаємодії з сервісом або пристроєм за допомогою протоколу HTTP і WEB-браузера.[5] WEB-інтерфейси набули широкого поширення у зв'язку з ростом популярності всесвітньої павутини і відповідно – розповсюдження WEB-браузерів.

Однією з основних вимог до WEB-інтерфейсів є їх однаковий зовнішній вигляд і однакова функціональність при роботі в різних браузерах.

Класичним і найбільш популярним методом створення веб-інтерфейсів є використання HTML із застосуванням CSS і JavaScript'a.

Таким інтерфейсом являється API. Це є набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) або операційною системою для використання у зовнішніх програмних продуктах.

Функції API діляться на 2 напрямки:

- повертають. На запит стороннього додатка будь-якого методу з заданими параметрами сервер повертає запитувану інформацію в певному форматі;

- змінюють. Клієнт викликає деяку функцію сервера, яка вводить нову інформацію або змінює на ньому ряд параметрів.

API може передавати інформацію в відмінному від стандартного HTML форматі, завдяки чому їм зручно користуватися при написанні власних програм. Сторонні загальнодоступні API найчастіше приймають і віддають дані в одному з двох форматів: XML або JSON. JSON набагато більш лаконічний і простий в читанні, ніж XML, а сервіси, що надають доступ до даних в XML-форматі, поступово відмовляються від нього.

2.2.1. XML

XML – розширювана мова розмітки.[5] Він дозволяє перепризначити дані у файлі або автоматизувати процес заміни даних одного файлу даними з іншого файлу. У форматі XML для опису частин файлу (заголовка, матеріалу тощо) застосовуються теги. Теги реалізують розмітку даних, дозволяючи зберігати їх в XML-файлах, а також правильно обробляти при експорті в інші файли. XML слід розглядати як механізм перетворення даних. XML-теги виробляють розмітку тексту будь-якого вмісту файлу, дозволяючи забезпечити правильне розпізнавання і відображення даних в різних програмах.

XML вважається розширюваною мовою, оскільки користувачі можуть створювати свої XML-теги. Для кожного типу даних, перетворення яких необхідно, може бути створений свій тег. XML-теги не містять інформації про те, як повинні відображатися або форматуватися дані. Вони використовуються виключно для ідентифікації вмісту.

XML-дані складаються з елементів, які представляють собою дані, що розмічені тегами. XML-файл складається з безлічі елементів, вкладених один в одного і реалізують ієрархічну структуру даних.

Структуру XML-даних можна побачити в палітрі «Структура», яка відображає ієрархію і порядок проходження елементів. У структурі XML дочірні елементи включені в батьківські, які, в свою чергу, можуть також

бути дочірніми елементами. З іншого боку, батьківські елементи містять дочірні елементи, які, в свою чергу, можуть бути батьківськими елементами для інших дочірніх елементів.

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Рис. 2.7 Приклад XML-файлу

2.2.2. JSON

JSON – текстовий формат обміну даними, заснований на JavaScript.[5] Як і багато інших текстових форматів, JSON легко читається людьми. Незважаючи на походження від JavaScript, формат вважається незалежним від мови і може використовуватися практично з будь-якою мовою програмування. Для багатьох мов існує готовий код для створення і обробки даних в форматі JSON.

JSON-текст являє собою (в закодованому вигляді) одну з двох структур[5]:

- Набір пар *ключ:значення*. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив. Ключем може бути тільки рядок, значенням – будь-яка форма.
- Упорядкований набір значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

Це універсальні структури даних: як правило, будь-яка сучасна мова програмування підтримує їх в тій чи іншій формі. Вони лягли в основу JSON, так як він використовується для обміну даними між різними мовами програмування.

У вигляді значень в JSON можуть бути використані:

Об'єкт - це неврегульована безліч пар *ключ:значення*, укладена в фігурні дужки «{}» . Ключ описується рядком, між ним і значенням стоїть символ «:» . Пари ключ-значення відокремлюються один від одного комами;

Масив (одновимірний) - це впорядкована множина значень . Масив полягає в квадратні дужки «[]» . Значення розділяються комами.

Число;

Літерали true, false і null;

Рядок – це впорядкована множина з нуля або більше символів юнікода, яка укладена в подвійні лапки. Символи можуть бути вказані з використанням escape-послідовностей, що починаються з зворотної косої межі «\» (підтримуються варіанти \", \\, \/, \t, \n, \r, \f і \b), або записані шістнадцятковим кодом в кодуванні Unicode у вигляді \uFFFF.

Рядок дуже схожа на однойменний тип даних в мовах C і Java . Число теж дуже схоже на C- або Java-число, за винятком того, що використовується тільки десятковий формат. Прогалини можуть бути вставлені між будь-якими двома синтаксичними елементами.

Наступний приклад показує JSON-уявлення об'єкта, що описує масив даних про людей. В масиві є строкові поля імені і прізвища. Також значення може являти собою вкладену структуру.[2]

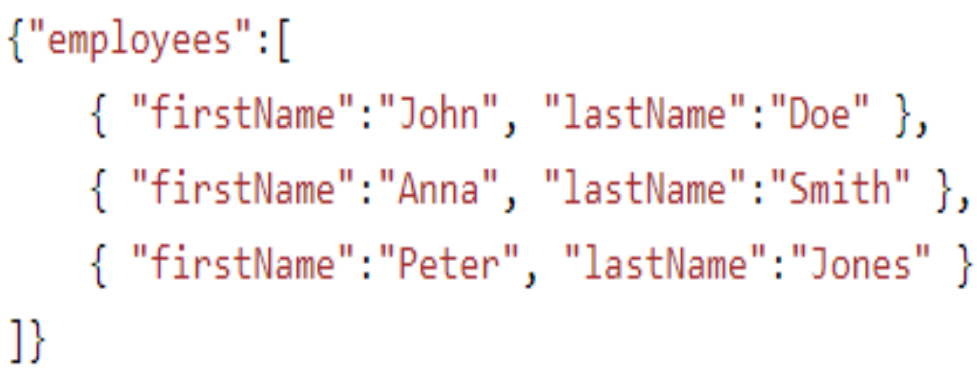

Рис. 2.8 Приклад JSON-файлу

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі були розглянуті протоколи обміну і передачі даних між інтерфейсами та формати, з якими інтерфейс працює.

Стандартним протоколом доступу до об’єктів WEB-сервісу являється SOAP. Спрощений протокол у порівнянні з SOAP це XML-RPC. Він дуже простий та ефективний, але не призначений для вирішення глобальних завдань.

Створення і опис WEB-інтерфейсів реалізується за допомогою окремих інструментів. Для уніфікованого представлення зовнішніх інтерфейсів веб-служби призначена мова опису WEB-сервісів WSDL. А інструментом для розташування описів WEB-сервісів являється інструмент UDDI.

Також існують XML та JSON формати з якими працюють інтерфейси. Обидва формати являються розширюваними та мають можливість використовуватися більшістю мов програмування. За зручністю читання та редагування JSON має переваги, не має надлишковості даних на віміну від XML. Також легше реалізувати взаємодію за допомогою JSON формату. Та вирішальним являється те, що цей формат краще буде парситися на JavaScript, а для WEB-сервісів це важливо. Та якщо сервіс є дуже затребуваний, то зазвичай, реалізується робота з двома форматами.

РОЗДІЛ 3

ПРОЕКТУВАННЯ СИСТЕМИ

Перед початком написання коду для створення програми є важливим спочатку ознайомитись з бізнес-процесами підприємства та спроектувати систему, щоб з часом було простіше її підтримувати та додавати нові можливості. Переді мною була поставлена задача автоматизувати роботу кінотеатру.

Зупинимося детально на процесі проектування. В ході проектування створюється проектна документація, що включає текстові описи, діаграми, моделі майбутньої програми. З цим нам допоможе мова UML.

UML - є графічною мовою для візуалізації, опису параметрів, конструювання та документування різних систем (програм зокрема). Діаграми створюються за допомогою спеціальних CASE засобів, наприклад Rational Rose і Enterprise Architect. На основі технології UML будується єдина інформаційна модель. Наведені вище CASE засоби здатні генерувати код на різних об'єктно-орієнтованих мовах, а так само мають дуже корисною функцією реверсивного інжинірингу, а саме створення графічної моделі з наявного програмного коду і коментарів до нього.

Для проектування нашої системи ми використали такі діаграми:

- Діаграма варіантів використання (use case diagram)
- Діаграма класів (class diagram)

Також, перед тим як почати написання коду програми, потрібно використати

3.1. Діаграми варіантів використання (use case diagrams)

Проектована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою за допомогою, так званих прецедентів. При цьому актором (actor) або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. Іншими словами, кожен варіант використання визначає деякий набір дій, який чинять системою при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів з системою.

3.1.1. Аналіз діаграми використання

Для даного проекту запропоновано наступні варіанти використання:

1. Оформлення замовлення
2. Керування кінотеатром
3. Формування звітів
4. Формування розкладу
5. Запит каталог фільмів у прокаті
6. Запит розкладу сеансів
7. Авторизація
8. Бронювання квитка
9. Проведення оплати
10. Формування квитка

Вони представлені загальною діаграмою та деталізацією певних варіантів використання, котрі вимагають детального аналізу: бронювання квитків, проведення оплати та формування звітів.

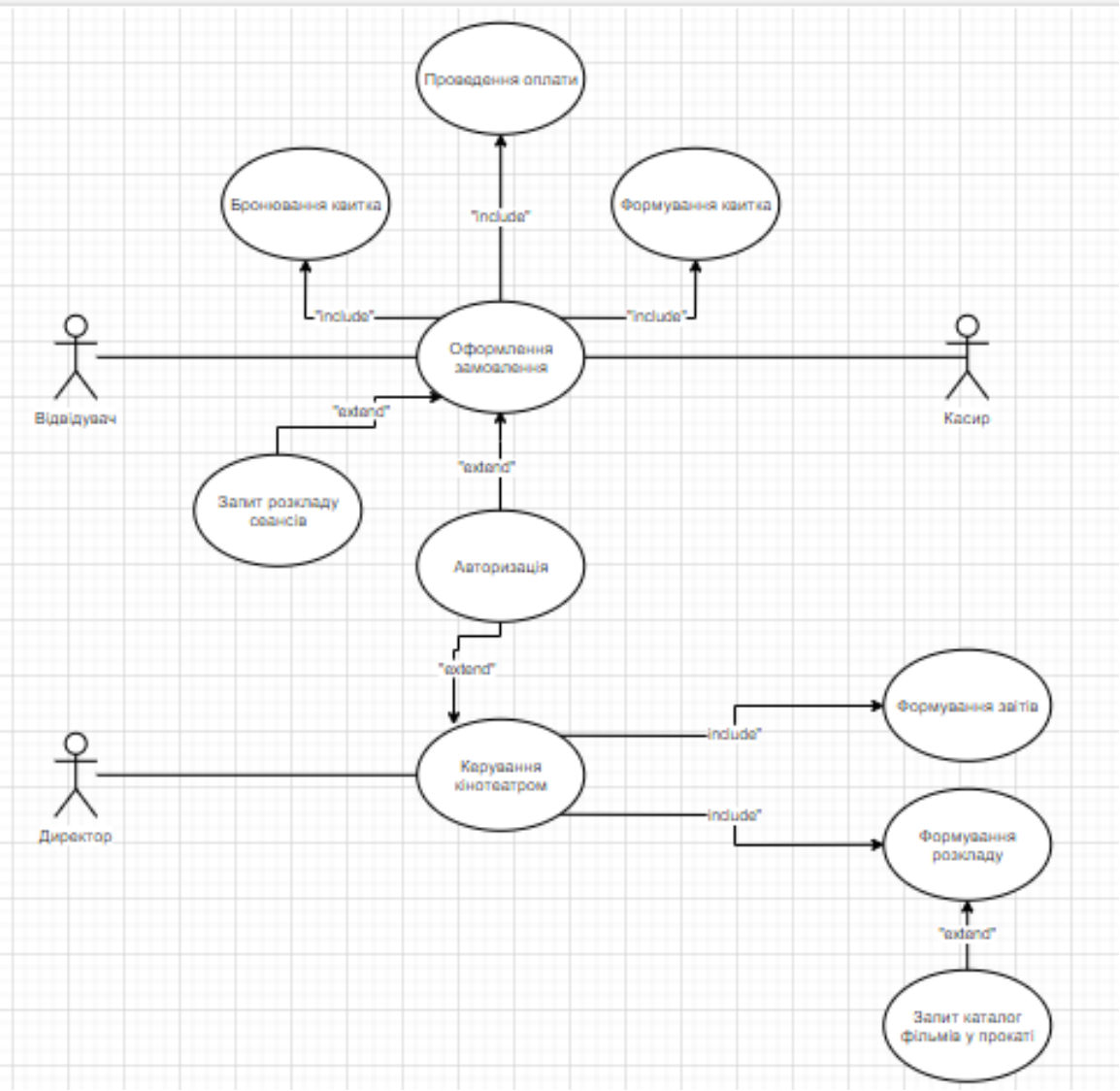

Рис.3.1 Загальна діаграма системи

- Реєстрація – додавання у БД системи нового користувача;
- Оформлення замовлення – відвідувач або касир на сайті купує квитки;
- Бронювання квитка - обирання вільних місць на обраному сеансі, які бронюються на 15 хвилин для проведення оплати;
    - Надання розкладу сеансів – касир відображає користувачу сеанси які можна відвідати;
    - Обирання сеансу – користувач обирає зручний для нього сеанс;
    - Надання список вільних місць – касир відображає користувачу вільні місця обраного сеансу.

- Бронювання місць – обирання користувачем зручних для нього місць, після чого ці місця автоматично бронюються на 15 хвилин очікуючи оплати;
- Робота з БД – отримання, створення, оновлення або видалення даних.

Рис.3.2 Діаграма «Бронювання квитка»

❖ Проведення оплати – етап оплати раніше заброньованих квитків

- Переведення користувача на сторінку платіжної системи – на сайті формується номер замовлення для передачі його платіжній системі
- Повідомлення про успішну чи не успішну оплату – інформує систему про результат оплати та зберігає інформацію в БД;

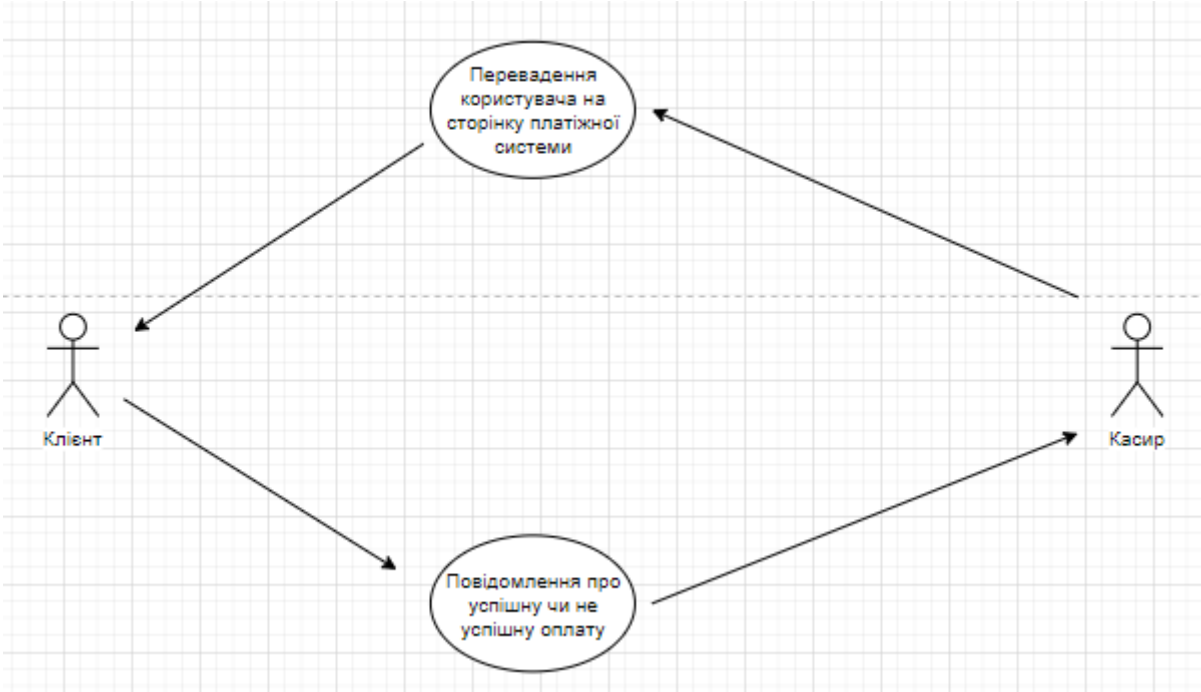

Рис.3.3 Діаграма «Проведення оплати»

- Формування квитка – надання користувачу квитка для печаті або використання у електорному вигляді.
- Керування кінотеатром – використання повноважень керівництва для забезпечення оптимального виконання кінотеатром свого призначення.
- Формування розкладу сеансів – аналіз виходу майбутніх кінофільмів.
- Формування звітів – формування та надання певних звітів у електроному вигляді
    - Формування статистики відвідувань – формування статистики за обраними критеріями.
    - Підрахування рентабельності кінозалу – аналіз ефективності використання кінозалу.
    - Надання звіту для печаті – надає звіти у електроному вигляді

Рис.3.4 Діаграма керування кінотеатром

Дійові особи:

- Директор – фактичне керівництво кінотеатра, котре повинно забезпечувати роботу даної установи.
- Касир – загальна множина робітників кінотеатру, котрі виконують функції продавця.
- Відвідувач – людина, яка звертається за купівлею квитків до кінотеатру, її фактичний клієнт.

3.2. Створення діаграми класів

Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати, зокрема, різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їх внутрішню структуру (поля, методи ...) і типи відносин (спадкування, реалізація інтерфейсів ...). На даній діаграмі не вказується інформація про тимчасові аспектах функціонування системи. З цієї точки зору діаграма класів є подальшим розвитком концептуальної моделі проектованої системи.

Для даної системи були обрані такі класи:

- Сеанс
- Зал
- Фільм
- Місце
- Користувач
- Бронь

В нас є сутність «Фільм». Цей клас зберігає важливу інформацію по фільму за допомогою атрибутів:

- назва;
- жанр;
- тривалість;
- дата початку прокату.

Є сутність «Сеанс» має єдиний атрибути «Дата». Тобто завдяки цієї сутності ми легко дізнаємося розклад фільмів на обраний день. Окрім дати проведення сеансу, ця сутність має зв’язок композиції з іншими сутностями:

- «Фільм»
- «Зал»

Сутність «Зал» у свою чергу зберігає інформацію по кількості приміщень, де можна демонструвати фільми. Ця сутність теж має зв’язок композиція, але з сутністю «Місце», яка має атрибути «Ціна», «Ряд» і «Місце», завдяки чому після купівлі система формує квиток на основі цих даних. Тобто у цій сутності вже йде зв’язок з сутністю «Бронь», але це зв’зок асоціації. Сама сутність «Бронь» має такі атрибути:

- «Номер бронювання»;
- «Оплачено»;
- «CreatedAt».

Ця сутність має зв'зок залежності, тому що інформація, яка надається у цьому класі, буде використана в таких сутностях як «Бронювання» і «Купівля».

Ці дві сутності мають по декілька операцій, які проводяться над даними, що належать сутності від якої залежать. Сутність «Купівля» виконує операції:

- «Отримати ціну»;
- «Проведення оплати»;
- «Фрмування квитків».

А сутність «Бронювання» має такі методи:

- «Отримання розкладу»;
- «Отримання місць»;
- «Бронювання»;
- «Скасування»;
- «Отримання бронювань користувача».

Також, інсує сутність «Користувач», яка має атрибути:

- «Email»;
- «Роль»;
- «ФИО»;
- «Password»;

Вона має з сутністю «Бронь» зв'зок асоціаціїї, та з сутністю «Бронювання» зв’язок залежності, тому що вона має таку операцію, як «Отримання бронювань користувача», яка не буде виконана, не володіючи інформацією по користувачу

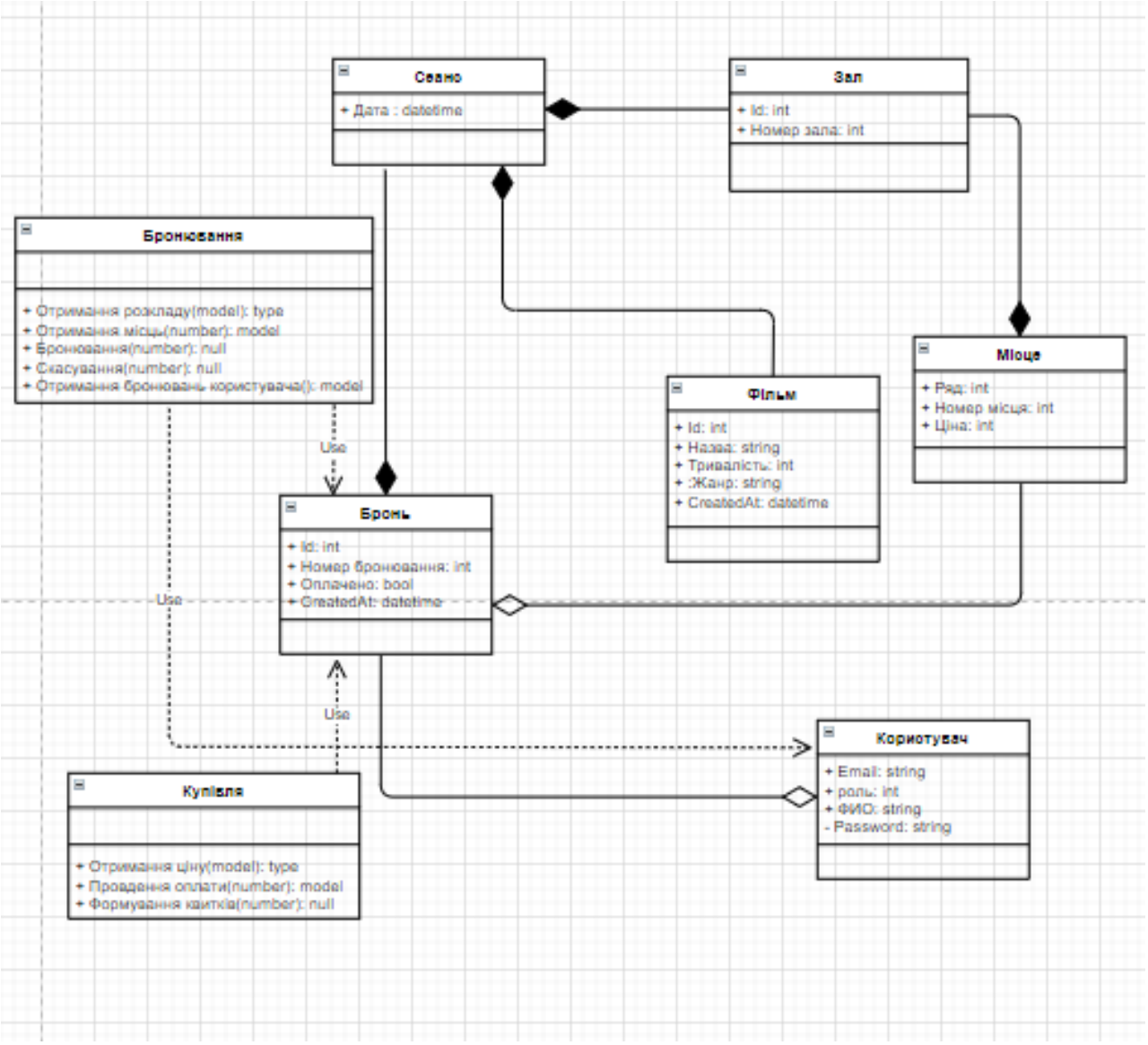

Рис.3.5 Діаграма класів системи

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі з метою кращого розуміння бізнес процесів системи була поставлена ціль побудувати діаграми. Були обрані дві діаграми use case і діаграма класів. Use case відображає взаємодію між сутностями, а діаграма класів структуру моделі у системі. І вже на основі цих діаграм буде проходити розробка системи.

РОЗДІЛ 4

ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ

Перед початком написання коду, звісно потрібно обрати технології та принципи за якими буде відбуватися робота. Засновуючись на тому, що у першому розділі, як більш підходяща, була обрана архітектура REST, то вона і впливала на подальший вибір.

Так само була обрана об’єктно-орієнтовна мова програмування C#. На сьогодні він є одним за найпотужніших мов програмування, він продовжує швидко розвиватись і з кожною новою версією з’являється все більше цікавого функціоналу. Він є стандартизованим у ECMA (ECMA-334)[6] и ISO (ISO/IEC 23270)[7]. Також обираючи цю мову програмування, мається на увазі, що буде використана платформа .NET, яка довгий час розвивалася головним чином як платформа для Windows під назвою .NET Framework. В 2019 вийша остання версія цієї платформи - .NET Framework 4.8. Більше вона на буде розвиватися, але від 2014 року Microsoft став розвивати альтернативну платформу - .NET Core.

4.1. .Net Core

Ця платформа вже призначалася для різних платформ і вбирає в себе всі можливості застарілого .NET Framework з новим функціоналом. Саме .NET Core розрахована в першу чергу на розробку для серверних і хмарних рішень.

Говорячи про. NET в першу чергу мають на увазі великі корпоративні проекти з високими вимогами до надійності, масштабованості і розширюваності. І при цьому з дорогою інфраструктурою.

Тому розробка під .NET завжди була досить недешевим задоволенням. Однак з .NET Core все трохи інакше.

Залишаючи в наявності всі плюси класичної версії, нова версія дозволяє значно знизити інфраструктурні витрати. Хостинг проекту на .NET Core обходиться набагато дешевше хостингу проекту на «великому» .NET.

У поточному ж своєму вигляді нова платформа буде грати в тому ж сегменті, де зараз знаходяться такі технології, як Node.js і Ruby - це невеликі проекти з обмеженим бюджетом і невисокою архітектурної складністю. Великі рішення як і раніше будуть реалізовуватися на «великому». NET. Таким чином, цільовою аудиторією .NET є стартапи.

4.2. ASP.Net Core

Працюючи на обраній платформі застосовувався фреймворк ASP.NET Core. З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, яка може бути розгорнута на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої

компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати. Для обробки запитів тепер використовується новий конвеєр HTTP.

4.3. Архітектура сервісу

4.3.1. Монолітна архітектура

Монолітне додаток являє собою додаток, що доставляється через єдине розгортання. Таким є додаток, зібране у вигляді однієї WAR або додаток Node з однією точкою входу.

Великою перевагою моноліту є те, що його легше реалізувати. У монолітної архітектури можна швидко почати реалізовувати свою бізнес-логіку, замість того щоб витрачати час на роздуми про взаємодії між процесами. Ще одна річ - це наскрізні (E2E) тести. У монолітної архітектурі їх легше виконати.

Говорячи про операції, важливо сказати, що моноліт простий в розгортанні і легко масштабується. Для розгортання ви можете використовувати скрипт, що завантажує ваш модуль і запускає додаток. Масштабування досягається шляхом розміщення Loadbalancer перед кількома екземплярами вашого застосування. Як ви можете бачити, моноліт досить простий в експлуатації.

Незважаючи на ці переваги, монолiти, як правило, перероджуються зі свого чистого стану в так звану «велику кульку бруду». Це описується як стан, що виник, тому що архітектурні правила були порушені і з часом компоненти переплелись.

Це перероджування уповільнює процес розробки: кожну майбутню

функцію буде складніше розвивати. Через те що компоненти зростають разом, їх також необхідно міняти разом. Створення нової функції може означати дотик до 5 різних місць: 5 місць, в яких потрібно написати тести; 5 місць, які можуть мати небажані побічні ефекти для існуючих функцій.

Вище було написано, що в моноліті легке масштабування. Це дійсно так до тих пір, поки він не переросте в «велику кульку бруду». Масштабування може бути проблематичним, коли тільки однієї частини системи потрібні додаткові ресурси, адже в монолітної архітектури ви не можете масштабувати окремі частини вашої системи. У моноліті практично немає ізоляції. Проблема або помилка в модулі може уповільнити або зруйнувати все додаток.

Будівництво моноліту часто протікає за допомогою вибору основи. Відключення або оновлення вашого початкового вибору може бути складним, тому що це має бути зроблено відразу і для всіх частин вашої системи.[5]

4.3.2. Багаторівнева архітектура

Ця архітектура є однією з найуспішніших. Вона передбачає поділ додатка на три рівні. Класична трирівнева система складається з наступних рівнів:

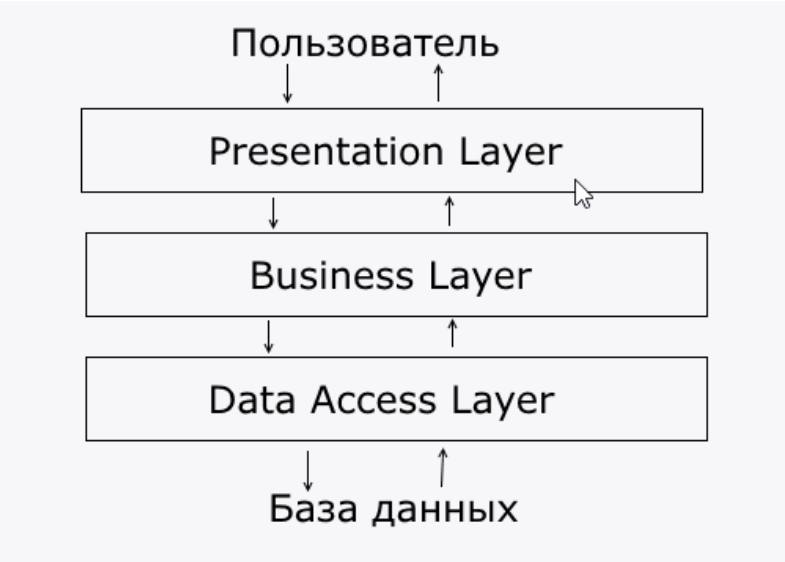

Рис.4.1 Рівні многоріневої архітектури

Presentation layer (рівень представлення): це той рівень, з яким безпосередньо взаємодіє користувач. Цей рівень включає компоненти для користувача інтерфейсу, механізм отримання введення від користувача. Стосовно до asp.net core на даному рівні розташовані уявлення і всі ті компоненти, з яких складається призначений для користувача інтерфейс (стилі, статичні сторінки html, javascript), а також моделі уявлень, контролери, об'єкти контексту запиту.

Business layer (рівень бізнес-логіки): містить набір компонентів, які відповідають за обробку отриманих від рівня уявлень даних, реалізує всю необхідну логіку додатка, все обчислення, взаємодіє з базою даних і передає рівнем подання результат обробки.

Data Access layer (рівень доступу до даних): зберігає моделі, що описують використовувані суті, також тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, наприклад, клас контексту даних Entity Framework. Тут також зберігаються репозиторії, через які рівень бізнес-логіки взаємодіє з базою даних.[5]

При цьому треба зазначити, що крайні рівні не можуть взаємодіяти між собою, тобто рівень представлення не можуть безпосередньо звертатися до бази даних і навіть до рівня доступу до даних, а тільки через рівень бізнес-логіки .

Рівень доступу до даних не залежить від інших рівнів, рівень бізнес-логіки залежить від рівня доступу до даних, а рівень представлення - від рівня бізнес-логіки.

Компоненти, як правило, повинні бути слабо зв'язаної (loose coupling), тому невід'ємною ланкою багаторівневих додатків є впровадження залежностей.

4.3.3. Onion-архітектура

Термін "Onion Architecture" ("цибулевий" архітектура) був запропонований Джеффрі Палермо (Jeffrey Palermo) ще в 2008 році. Через роки ця концепція стала досить популярною і є однією з найбільш вживаних типів архітектури при побудові програми на ASP.NET.

Onion-архітектура являє собою поділ програми на рівні. При чому є один незалежний рівень, який знаходиться в центрі архітектури. Від цього рівня залежить другий рівень, від другого - третій і так далі. Тобто виходить, що навколо першого незалежного рівня нашаровується другий-залежний. Навколо другого нашаровується третій, який також може залежати і від першого. Образно це може бути виражено у вигляді лука, в якому також є серцевина, навколо якого нашаровуються всі інші верстви, аж до лушпиння.[5]

Кількість рівнів може відрізнятися, але в центрі завжди знаходиться модель домену (Domain Model), тобто ті класи моделей, які використовуються в додатку і об'єкти яких зберігаються в базі даних:

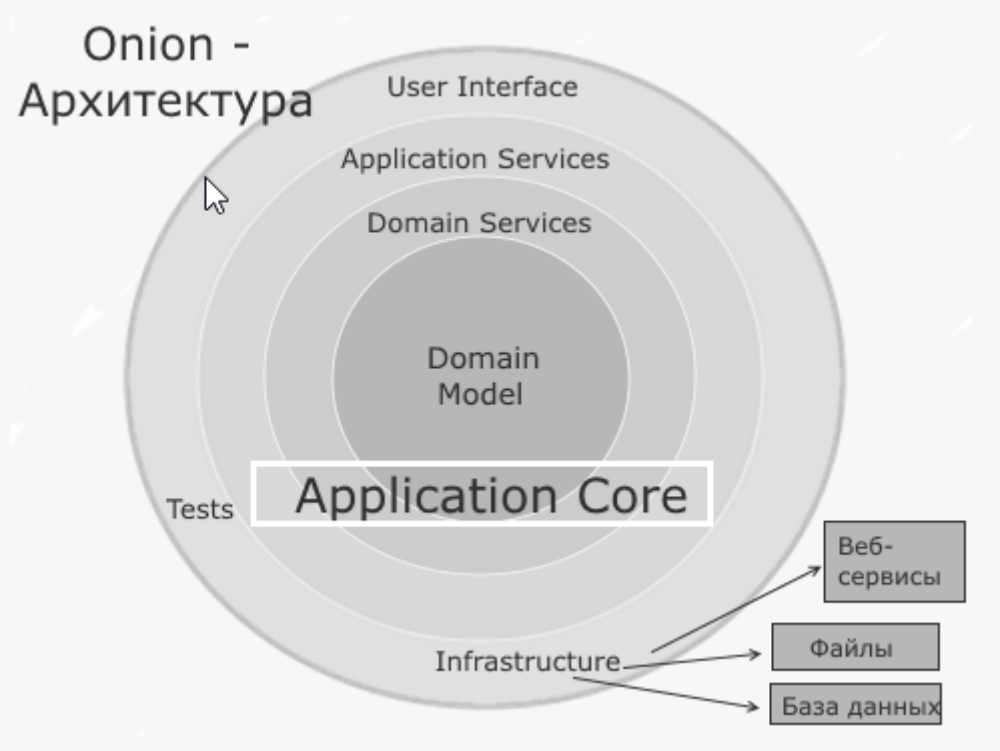

Рис.4.2 Onion-архітектура

4.4. Entity Framework

Для роботи з базою даних на платформі .NET існує технологія Entity Framework. Він представляє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працює з об'єктами.

Відмінною рисою Entity Framework є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо не тільки отримувати певні рядки, що зберігають об'єкти, з бд, а й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям є Entity Data Model. Ця модель зіставляє класи сутностей з реальними таблицями в БД. Entity Data Model складається з трьох рівнів: концептуального, рівень сховища і рівень зіставлення (маппінга). На концептуальному рівні відбувається визначення класів сутностей, які використовуються в додатку. Рівень сховища визначає таблиці, стовпці, відносини між таблицями і типи даних, з якими порівнюється використовувана база даних. Рівень зіставлення (маппінга) служить посередником між попередніми двома, визначаючи зіставлення між властивостями класу суті і стовпцями таблиць.

Таким чином, ми можемо через класи, визначені у додатку, взаємодіяти з таблицями з бази даних.

ВИСНОВОК ДО РОЗДІЛУ 4

Перед створенням програми потрібно ще обрати такі речі, як платформа, фреймворк і архітктуру за принципом якої буде побудовано програму. Було обрано одна за найвідоміших і найпотужніших платформ .NET Core.

Спираючись на те, що описано у розділі, можна відокремити багаторівневу архітектуру. Завдяки їй, притримуючись вимог, буде побудована на перший погляд складна система, але саме це дозволяє легко додавати нові модулі, змінювати готовий функціонал не руйнуючи вже створений. Також для зручної роботи з базою даних була обрана технологія Entity Framework.

ВИСНОВКИ

У дипломному проекті були проаналізовані чотири архітектури програмного забезпечення. Серед них було обрано архітектуру REST. Вона є кращою завдяки надійності, продуктивності, масштабованості, простоті інтерфейсів, портативність компонентів та легкості внесення змін.

На основі вибраної архітектури були більш детально розібрані протоколи обміну і передачі даних та формати, з якими працюють WEB-сервіси.

Для візуалізації бізнес-процесів системи були побудовані дві діаграми: use case і діаграма класів.

Також для розробки було застосовано:

- фреймворк ASP.NET Core;
- контейнер впровадження залежностей Autofac;
- ORM (object-relational mapping) Entity Framework Core.

В основу програми покладено багаторівневу архітектуру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. www.information-management.com [Електронний ресурс]. – Режим доступу: www.information-management.com/author/jeremy-westerman-im32391

2. Озімай Д. О. Прикладна програма для використання Web-сервісу на основі локальної бази даних: дипломний проект. – М.: Національний авіаційний університет, 2018. – 75 с.

3. SOA www.opengroup.org [Електронний ресурс]. – Режим доступу: http://www.opengroup.org/

4. Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford. Documenting Software Architectures: Views and Beyond. — Second Edition. — Addison-Wesley Professional, 2010. — ISBN 978-0-13-248861-7.

5. Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice, Third Edition. Addison Wesley, 2012, ISBN 978-0321815736 (This book, now in third edition, eloquently covers the fundamental concepts of the discipline. The theme is centered around achieving quality attributes of a system.)

6. Standard ECMA-334 C# Language Specification, 4rd edition (англ.). Ecma International (June 2006). Дата обращения 16 мая 2017.

7. ISO/IEC 23270:2003 Information technology -- C# Language Specification (англ.). International Organization for Standardization (April 2003). Дата обращения 16 мая 2017

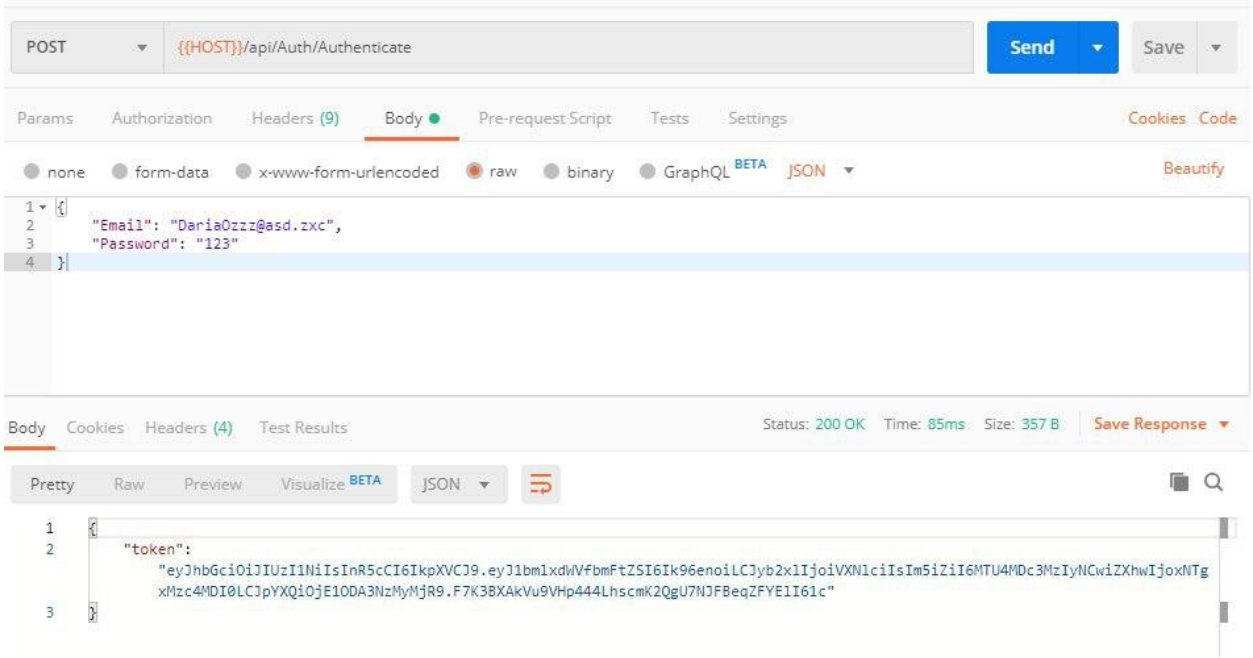

Рис. А.1. Приклад запиту до сервісу авторизації користувача

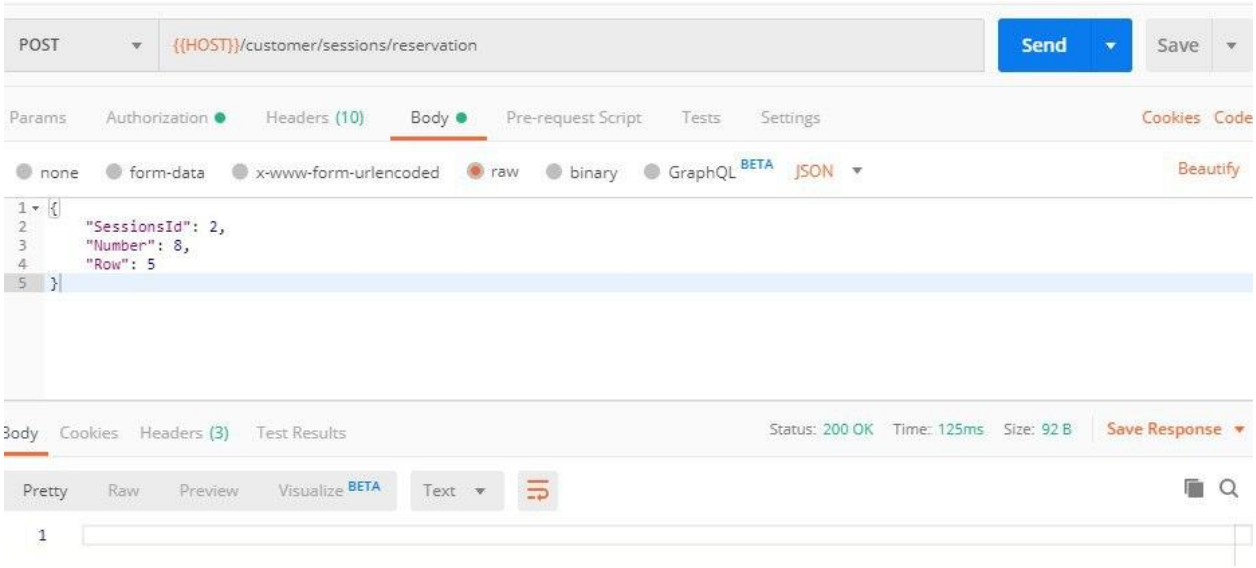

Рис. А.2. Приклад запиту до сервісу бронювання квитка

Додаток Б

Код прикладної програми

```
using System.Net.Mime;
using System;
using System.IO;
using Autofac;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Configuration.Json;
using UniverseKino.Data.EF;
using UniverseKino.Data.Interfaces;
using UniverseKino.Data.Repositories;
using UniverseKino.Data.Entities;

namespace UniverseKino.Data
{

    public class DataModule : Module
    {
        protected void OnModelCreating(UniverseKinoContext ctx)
        {
            InitDb init = new InitDb();

            ctx.CinemaHalls.AddRange(
                new CinemaHall[]
                    {
                        new CinemaHall { Number = 1, Id = 1, Seats = init.GetSeats(10, 10, 100, 1) },
                        new CinemaHall { Number = 2, Id = 2, Seats = init.GetSeats(10, 10, 150, 2) },
```

```
                new CinemaHall { Number = 3, Id = 3, Seats = init.GetSeats(13, 13, 200, 3) },
                new CinemaHall { Number = 4, Id = 4, Seats = init.GetSeats(10, 10, 177, 4) },
            }
        );


        ctx.Movies.AddRange(
            new Movie[]
            {
                new Movie {Id = 1, Name = "Movie1", Genre = "FantastikaBlya", Duration = 100
},
                new Movie {Id = 2, Name = "Second movie ska", Genre = "TravelYopta", Duration
= 187  },
                new Movie {Id = 3, Name = "LastNah", Genre = "Ujastik", Duration = 250 },
            }
        );


        ctx.Sessions.AddRange(
            new Session[]
            {
                new Session {Id = 1, Date = GetDate(1, 9), IdMovie = 1, IdCinemaHall = 1 },
                new Session {Id = 2, Date = GetDate(3, 9), IdMovie = 1, IdCinemaHall = 3 },
                new Session {Id = 3, Date = GetDate(5, 9), IdMovie = 1, IdCinemaHall = 3 },
                new Session {Id = 4, Date = GetDate(1, 12), IdMovie = 2, IdCinemaHall = 1 },
            }
        );

        ctx.SaveChanges();

    }
```

```
        private static DateTime GetDate(int day = 0, int hour = 8)
        {
            var dateString = (new DateTime()).AddDays(day).ToShortDateString();
            var date = DateTime.Parse(dateString);
            date.AddHours(hour);
            return date;

        }
        protected override void Load(ContainerBuilder builder)
        {
            var config = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("appsettings.json")
            .Build();

            var connString = config.GetConnectionString("Develop");

            builder.RegisterType<AuthRepository>();

            builder.Register(u => new UnitOfWork(u.Resolve<UniverseKinoContext>()))
            .As<IUnitOfWorkEntities>();
        }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace UniverseKino.Services
{
```

```
    public class LoginRequestDTO
    {
        public string Email { get; set; }
        public string Password { get; set; }

    }
    public class RegistrationRequestDTO
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string UserName { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
    }
}

using System;
using System.Linq;
using System.Reflection;
using Autofac;
using AutoMapper;
using Microsoft.Extensions.Logging;
using UniverseKino.Core;
using UniverseKino.Data;
using UniverseKino.Data.EF;
using UniverseKino.Data.Entities;
using UniverseKino.Data.Interfaces;
using UniverseKino.Data.Repositories;
using UniverseKino.Services.Services;
using UniverseKino.Services.Interfaces;
```

```
namespace UniverseKino.Services
{
    public class ServicesModule : Autofac.Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterModule<DataModule>();
            builder.RegisterType<AuthService>().As<IAuthService>();

            builder.RegisterType<ViewInfoService>().As<IViewInfoService>();
            builder.RegisterType<CheckService>().As<ICheckService>();
            builder.RegisterType<ManageMoviesService>().As<IManageMoviesService>();

            builder.Register(x => new InfoMoviesService(
                x.Resolve<IUnitOfWorkEntities>(),
                x.Resolve<IMapper>()))
                .As<IInfoMoviesService>();

            builder.Register(x => new SessionsInfoService(
                x.Resolve<IUnitOfWorkEntities>(),
                x.Resolve<IMapper>()))
                .As<ISessionsInfoService>();
        }
    }
}
namespace UniverseKino.WEB.Models
{
    public class TokenResponseDTO
    {
```

```
        public string Token { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using Autofac;
using AutoMapper;
using UniverseKino.Core;
using UniverseKino.Services;
using UniverseKino.WEB.Models;
using UniverseKino.WEB;
using UniverseKino.Services.Services;

namespace UniverseKino.WEB
{
    public class ControllersModule : Autofac.Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.Register(x => new PasswordHasher());
            builder.RegisterType<MappingProfile>();
            builder.RegisterType<ServicesMappingProfile>();

            builder.Register(x =>
                    new MapperConfiguration(mc =>
                    mc.AddProfiles(new List<Profile>
                    {
                        x.Resolve<MappingProfile>(),
```

```
                x.Resolve<ServicesMappingProfile>()
            }
            )
            ).CreateMapper()
        )
        .As<IMapper>();

        builder.RegisterModule<ServicesModule>();
    }
  }

}
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Autofac.Extensions.DependencyInjection;
//using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using Microsoft.AspNetCore.Hosting;

namespace UniverseKino.WEB
{
  public class Program
  {
    public static void Main(string[] args)
    {
```

```
            var host = WebHost.CreateDefaultBuilder(args)
                .ConfigureServices(services => services.AddAutofac())
                .UseStartup<Startup>()
                .Build();

            host.Run();

        }

    }
}

using Autofac;
using Autofac.Extensions.DependencyInjection;
using Microsoft.AspNetCore.Builder;
using System;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using System.Linq;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.Collections.Generic;
using System.IO;
using UniverseKino.Core;
using AutoMapper;
using UniverseKino.Services;

using Microsoft.IdentityModel.Tokens;
```

```
using System.Security.Claims;
using System.Text;
using AutoMapper.Configuration;

namespace UniverseKino.WEB
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddEnvironmentVariables();
            this.Configuration = builder.Build();
        }

        public IConfigurationRoot Configuration { get; private set; }

        public void Configure(IApplicationBuilder app)
        {

            app.UseRouting();
            app.UseCors();

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
```

```
            endpoints.MapControllers();
            // endpoints.MapControllerRoute("default",
"{controller=Home}/{action=Index}/{id?}");
        });

    }

    public void ConfigureContainer(ContainerBuilder builder)
    {
        builder.RegisterModule(new ControllersModule());
    }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();
        services.AddControllersWithViews();
        services.Configure<JWTHelper>(Configuration);

        var key = Encoding.ASCII.GetBytes("THIS IS USED TO SIGN AND VERIFY JWT
TOKENS, REPLACE IT WITH YOUR OWN SECRET, IT CAN BE ANY STRING");
        services.AddAuthentication(x =>
        {
            x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
            x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        })
            .AddJwtBearer(options =>
            {
                options.TokenValidationParameters = new TokenValidationParameters
                {

                    ValidateIssuerSigningKey = true,
```

```
                IssuerSigningKey = new SymmetricSecurityKey(key),
                ValidateIssuer = false,
                ValidateAudience = false
            };
        });
    }
  }
}

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using UniverseKino.Data.Entities;

namespace UniverseKino.Data.EF
{
  public class ApplicationContext : DbContext
  {
  public ApplicationContext(DbContextOptions<ApplicationContext> options)
      : base(options)
    {
      Database.EnsureCreated();
    }

    public DbSet<ApplicationUser> ApplicationUsers { get; set; }
  }
}
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Text;
```

```
using UniverseKino.Data.Entities;

namespace UniverseKino.Data
{
    class InitDb
    {
        public List<Seat> GetSeats(int countPlace, int numberRow, decimal cost, int cinemaId)
        {
            var list = new List<Seat>();

            for (int i = 1; i <= countPlace; i++)
            {
                for (int j = 1; i <= numberRow; i++)
                {
                    list.Add(new Seat { Id = 1, Cost = cost, IdCinemaHall = 1, Number = j, Row = i });
                }
            }

            return list;
        }
    }
}
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;

namespace UniverseKino.Data.EF
{
```

```
public class UniverseKinoContext : DbContext
{
    public DbSet<Movie> Movies { get; set; }

    public DbSet<CinemaHall> CinemaHalls { get; set; }
    public DbSet<Seat> Seats { get; set; }

    public DbSet<Session> Sessions { get; set; }
    public DbSet<ApplicationUser> ApplicationUsers { get; set; }

    public DbSet<Reservation> Reservations { get; set; }

    public UniverseKinoContext(DbContextOptions<UniverseKinoContext> options)
            : base(options)
    {
        Database.EnsureCreated();
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<CinemaHall>().HasData(

                new CinemaHall[]
                {
                    new CinemaHall { Number = 1, Id = 1 },
                    new CinemaHall { Number = 2, Id = 2 },
                    new CinemaHall { Number = 3, Id = 3 },
                    new CinemaHall { Number = 4, Id = 4 },
                }
        );
```

```
            modelBuilder.Entity<Movie>().HasData(
                new Movie[]
                {
                    new Movie {Id = 1, Name = "Movie1", Genre = "FantastikaBlya", Duration = 100 },
                    new Movie {Id = 2, Name = "Second movie ska", Genre = "TravelYopta", Duration = 187  },
                    new Movie {Id = 3, Name = "LastNah", Genre = "Ujastik", Duration = 250 },
                }
            );
            modelBuilder.Entity<Session>().HasData(
                new Session[]
                {
                    new Session {Id = 1, Date = GetDate(1, 9), IdMovie = 1, IdCinemaHall = 1 },
                    new Session {Id = 2, Date = GetDate(3, 9), IdMovie = 1, IdCinemaHall = 3 },
                    new Session {Id = 3, Date = GetDate(5, 9), IdMovie = 1, IdCinemaHall = 3 },
                    new Session {Id = 4, Date = GetDate(1, 12), IdMovie = 2, IdCinemaHall = 1 },
                }
            );
        }
        private static DateTime GetDate(int day = 0, int hour = 8)
        {
            var dateString = (new DateTime()).AddDays(day).ToShortDateString();
            var date = DateTime.Parse(dateString);
            date.AddHours(hour);
            return date;
        }
    }
}

using Microsoft.AspNetCore.Identity;
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace UniverseKino.Data.Entities
{
    public class ApplicationUser  : BaseEntity
    {
        public string Email { get; set; }
        public string Role { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace UniverseKino.Data.Entities
{
    public class BaseEntity
    {
        [Key]
        public int Id { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace UniverseKino.Data.Entities
{
    public class CinemaHall : BaseEntity
    {
        [Required]
        public int Number { get; set; }

        public List<Seat> Seats { get; set; }

        public List<Session> Sessions { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace UniverseKino.Data.Entities
{
    public class Movie : BaseEntity
    {
        [Required]
        public string Name { get; set; }
```

```
        [Required]
        public string Genre { get; set; }

        [Required]
        public int Duration { get; set; }  //stored in minutes

        public List<Session> Sessions { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace UniverseKino.Data.Entities
{
    public class Reservation : BaseEntity
    {
        [Required]
        public string Number { get; set; }

        [Required]
        public int IdSession { get; set; }
        public Session Session { get; set; }

        [Required]
        public string UserName { get; set; }
    }
```

```
}

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace UniverseKino.Data.Entities
{
    public class Seat : BaseEntity
    {
        [Required]
        public int IdCinemaHall { get; set; }
        public CinemaHall CinemaHall { get; set; }

        [Required]
        public decimal Cost { get; set; }

        [Required]
        public int Row { get; set; }

        [Required]
        public int Number { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;
```

```
namespace UniverseKino.Data.Entities
{
    public class Session : BaseEntity
    {
        [Required]
        public DateTime Date { get; set; }

        [Required]
        public int IdMovie { get; set; }
        public Movie Movie { get; set; }

        [Required]
        public int IdCinemaHall { get; set; }
        public CinemaHall CinemaHall { get; set; }

        public List<Reservation> Reservations { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;

namespace UniverseKino.Data.Interfaces
{
    public interface IGenericRepository<TEntity> where TEntity : BaseEntity
    {
        void Add(TEntity entity);
```

```
        void Update(TEntity entity);

        void Remove(int id);

        IEnumerable<TEntity> GetAll();

        IEnumerable<TEntity> Find(Func<TEntity, bool> predicate);

        TEntity GetById(int id);
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;

namespace UniverseKino.Data.Interfaces
{
    public interface IUnitOfWorkEntities : IDisposable
    {
        IGenericRepository<Movie> Movies { get; }
        IGenericRepository<CinemaHall> CinemaHalls { get; }
        IGenericRepository<Seat> Seats { get; }
        IGenericRepository<Session> Sessions { get; }
        IGenericRepository<Reservation> Reservations { get; }

        void Save();
    }
```

```
}

using System.Collections.Generic;
using System.Net.Mime;
using UniverseKino.Data.EF;
using UniverseKino.Data.Entities;
using System.Linq;
using System.Threading.Tasks;

namespace UniverseKino.Data.Repositories
{
    public class AuthRepository
    {
        public List<ApplicationUser> Users
        {
            get
            {
                return _appContext.ApplicationUsers.ToList();
            }
        }

        public async Task<ApplicationUser> AddAsync(ApplicationUser user)
        {

            var savedUser = await _appContext.ApplicationUsers.AddAsync(user);
            await _appContext.SaveChangesAsync();

            return _appContext.ApplicationUsers.Where(u => u.Email ==
user.Email).FirstOrDefault();
        }
```

```
        private UniverseKinoContext _appContext;
        public AuthRepository(UniverseKinoContext dbContext)
        {
            _appContext = dbContext;
        }

        public void SaveChanges()
        {
            _appContext.SaveChanges();
        }
        public async Task SaveChangesAsync()
        {
            await _appContext.SaveChangesAsync();
        }

    }
}
using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;
using UniverseKino.Data.Interfaces;
using UniverseKino.Data.EF;
using System.Linq;
using Microsoft.EntityFrameworkCore;

namespace UniverseKino.Data.Repositories
{
    public class GenericRepository<TEntity> : IGenericRepository<TEntity> where TEntity :
BaseEntity
    {
```

```
private readonly UniverseKinoContext dbContext;

private DbSet<TEntity> Entities
{
    get
    {
        return dbContext.Set<TEntity>();
    }
}

public GenericRepository(UniverseKinoContext dbContext)
{
    this.dbContext = dbContext;
}

public void Add(TEntity entity)
{
    Entities.Add(entity);
}

public IEnumerable<TEntity> Find(Func<TEntity, bool> predicate)
{
    var listResult = Entities.Where(predicate);

    return listResult;
}

public IEnumerable<TEntity> GetAll()
{
    return Entities;
```

```
        }

        public TEntity GetById(int id)
        {
            var entity = Entities.Where(x => x.Id == id).FirstOrDefault();

            return entity;
        }

        public void Remove(int id)
        {
            var removeEntity = GetById(id);

            Entities.Remove(removeEntity);
        }

        public void Update(TEntity entity)
        {
            dbContext.Entry(entity).State = EntityState.Modified;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using UniverseKino.Data.Entities;
using UniverseKino.Data.Interfaces;
using UniverseKino.Data.EF;
```

```
using Microsoft.EntityFrameworkCore;

namespace UniverseKino.Data.Repositories
{
    public class UnitOfWork : IUnitOfWorkEntities, IDisposable
    {
        private readonly UniverseKinoContext dbContext;

        private IGenericRepository<Movie> movies;
        private IGenericRepository<CinemaHall> cinemaHalls;
        private IGenericRepository<Session> sessions;
        private IGenericRepository<Reservation> reservations;
        private IGenericRepository<Seat> seats;

        public UnitOfWork(UniverseKinoContext dbContext)
        {
            this.dbContext = dbContext;
        }

        private IGenericRepository<T> GetRepository<T>() where T : BaseEntity
        {
            return new GenericRepository<T>(dbContext);
        }

        public IGenericRepository<Movie> Movies
        {
            get
            {
                if (movies == null)
                    movies = GetRepository<Movie>();
```

```
        return movies;
    }
}

public IGenericRepository<CinemaHall> CinemaHalls
{
    get
    {
        if (cinemaHalls == null)
            cinemaHalls = GetRepository<CinemaHall>();

        return cinemaHalls;
    }
}

public IGenericRepository<Seat> Seats
{
    get
    {
        if (seats == null)
            seats = GetRepository<Seat>();

        return seats;
    }
}

public IGenericRepository<Session> Sessions
{
    get
```

```
    {
        if (sessions == null)
            sessions = GetRepository<Session>();

        return sessions;
    }
}

public IGenericRepository<Reservation> Reservations
{
    get
    {
        if (reservations == null)
            reservations = GetRepository<Reservation>();

        return reservations;
    }
}

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

private bool disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!disposed)
```

```
        {
            if (disposing)
            {
                dbContext.Dispose();
            }
            disposed = true;
        }
    }

    public void Save()
    {
        dbContext.SaveChanges();
    }
  }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace UniverseKino.Services.Dto
{
  public class MovieDTO
  {
    public string Name { get; set; }

    public string Genre { get; set; }

    public int Duration { get; set; }
  }
```

```
}

using System;
using System.Collections.Generic;
using System.Text;

namespace UniverseKino.Services.Dto
{
    public class SeatDTO
    {
        public decimal Cost { get; set; }

        public int Row { get; set; }

        public int Number { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace UniverseKino.Services.Dto
{
    public class SessionDTO
    {
        public DateTime Date { get; set; }

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

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

        public int DurationMovie { get; set; }

        public int NumberHall { get; set; }

        public List<SeatDTO> Seats { get; set; }
    }
}

namespace UniverseKino.Services
{
    public class UserDTO
    {
        public string Id { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public string UserName { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public string Role { get; set; }
    }
}
using System.Collections.Generic;
using System.Security.Claims;
using System.Threading.Tasks;
using UniverseKino.Data.Entities;
using UniverseKino.WEB.Models;

namespace UniverseKino.Services
```

```
{
    public interface IAuthService
    {
        Task<TokenResponseDTO> Register(RegistrationRequestDTO data);
        Task<TokenResponseDTO> Authenticate(LoginRequestDTO data);

        List<dynamic> AllUsers();
        IEnumerable<ApplicationUser> GetAll();
        // Task<OperationDetails> Create(UserDTO userDto);
        // Task<ClaimsIdentity> Authenticate(UserDTO userDto);
        // Task SetInitialData(UserDTO adminDto, List<string> roles);
    }
}
using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;

namespace UniverseKino.Services.Interfaces
{
    public interface ICheckService
    {
        bool ValidationMovie(Movie movie);
        bool IsExistMovie(Movie movie);

        void ValidationSession(Session session);
    }
}

using System;
```

```
using System.Collections.Generic;
using System.Text;
using UniverseKino.Services.Dto;

namespace UniverseKino.Services.Interfaces
{
    public interface IManageMoviesService
    {
        void Add(MovieDTO session);
        void Remove(int id);
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;
using UniverseKino.Services.Dto;

namespace UniverseKino.Services.Interfaces
{
    public interface IManageSessionsService
    {
        void Add(SessionDTO session);
        void Update(SessionDTO session);
        bool Remove(int id);
    }
}

using System;
```

```
using System.Collections.Generic;
using System.Text;
using UniverseKino.Services.Dto;

namespace UniverseKino.Services.Interfaces
{
    public interface ISessionsInfoService
    {
        SessionDTO GetSession(int id);
        List<SessionDTO> GetAllSessions();
        List<SessionDTO> GetSessionsByMovie(int idMovie);
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Services.Dto;

namespace UniverseKino.Services.Interfaces
{
    public interface IViewInfoService
    {
        SessionDTO GetSession(int id);
        List<SessionDTO> GetSessions();
        MovieDTO GetMovie(int id);
        List<MovieDTO> GetMovies();
    }
}
```

```
using System.Net.Mime;
using AutoMapper;
using System;
using UniverseKino.Core;
using UniverseKino.Services;
using UniverseKino.Data.Entities;
using UniverseKino.Services.Dto;
using System.Collections.Generic;

namespace UniverseKino.Services
{
    public class ServicesMappingProfile : Profile
    {
        public ServicesMappingProfile()
        {
            CreateMap<RegistrationRequestDTO, ApplicationUser>()
                .ForMember(x => x.Role, opt => opt.MapFrom(y => "User"));



            CreateMap<ApplicationUser, RegistrationRequestDTO>();


            CreateMap<SeatDTO, Seat>();
            CreateMap<Seat, SeatDTO>();


            CreateMap<MovieDTO, Movie>();
            CreateMap<Movie, MovieDTO>();


            CreateMap<SessionDTO, Session>();


            CreateMap<Session, SessionDTO>()
```

```
            .ForMember(a => a.NameMovie, opt => opt.MapFrom(b => b.Movie.Name))
            .ForMember(a => a.GenreMovie, opt => opt.MapFrom(b => b.Movie.Genre))
            .ForMember(a => a.DurationMovie, opt => opt.MapFrom(b => b.Movie.Duration))
            .ForMember(a => a.NumberHall, opt => opt.MapFrom(b =>
b.CinemaHall.Number))
            .ForMember(a => a.Seats, opt => opt.MapFrom(b => b.CinemaHall.Seats));
      }
   }
}

using System.Net.Mime;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using UniverseKino.Data.EF;
using UniverseKino.Data.Entities;
using UniverseKino.WEB.Models;
using System;
using AutoMapper;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using Microsoft.AspNetCore.Identity;
using UniverseKino.Services.Services;
using Microsoft.Extensions.Options;
using UniverseKino.Core;
using UniverseKino.Data.Repositories;

namespace UniverseKino.Services
{
```

```
public class AuthService : IAuthService
{
    private AuthRepository _userRepo;
    private JWTHelper _jwt;
    private IMapper _mapper;
    private SymmetricSecurityKey key;
    private JwtSecurityTokenHandler tokenHandler = new JwtSecurityTokenHandler();
    private PasswordHasher _hasher;

    public AuthService(AuthRepository userRepo, IOptionsMonitor<JWTHelper> jwt,
PasswordHasher hasher, IMapper mapper)
    {
        _mapper = mapper;
        _jwt = jwt.CurrentValue;
        _userRepo = userRepo;
        _hasher = hasher;
        key = _jwt.GetSecretKey();
    }
    private dynamic Convert(ApplicationUser user) => new { User = user };
    public List<dynamic> AllUsers()
    {
        return _userRepo.Users.ConvertAll<dynamic>(
            new Converter<ApplicationUser, dynamic>(Convert)
        );
    }

    public Task<TokenResponseDTO> Authenticate(LoginRequestDTO data)
    {

        var user = _userRepo.Users
```

```
            .Where(u => u.Email == data.Email && _hasher.Check(u.Password,
data.Password))
            .FirstOrDefault();

        if (user == null)
        {
            throw new Exception();
        }

        // var tokenHandler = new JwtSecurityTokenHandler();
        // var key = new SymmetricSecurityKey(Encoding.ASCII.GetBytes("THIS IS USED TO
SIGN AND VERIFY JWT TOKENS, REPLACE IT WITH YOUR OWN SECRET, IT CAN BE
ANY STRING")); //AppSettings.GetSymmetricAuthSecretKey();
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(
             new Claim[] {
            new Claim (ClaimsIdentity.DefaultNameClaimType, user.Username),
            new Claim (ClaimsIdentity.DefaultRoleClaimType, user.Role),
             },
             "Token",
             ClaimsIdentity.DefaultNameClaimType,
             ClaimsIdentity.DefaultRoleClaimType
             ),
            Expires = DateTime.UtcNow.AddDays(7),
            SigningCredentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256Signature),
        };
        var token = tokenHandler.CreateToken(tokenDescriptor);

        return Task.Run(() => new TokenResponseDTO { Token =
tokenHandler.WriteToken(token) });
```

```
}



public IEnumerable<ApplicationUser> GetAll()
{
    return _userRepo.Users;
}

public async Task<TokenResponseDTO> Register(RegistrationRequestDTO data)
{

    var user = _userRepo.Users
            .Where(u => u.Email == data.Email)
            .FirstOrDefault();

    if (user != null)
    {
        throw new Exception();
    }

    var newUser = _mapper.Map<ApplicationUser>(data);

    newUser.Role = "User";
    newUser.Password = _hasher.Hash(data.Password);

    var savedUser = await _userRepo.AddAsync(newUser);

    var tokenDescriptor = new SecurityTokenDescriptor
    {
```

```
            Subject = new ClaimsIdentity(
                    new Claim[] {
            new Claim (ClaimsIdentity.DefaultNameClaimType, savedUser.Username),
            new Claim (ClaimsIdentity.DefaultRoleClaimType, savedUser.Role),
                    },
                    "Token",
                    ClaimsIdentity.DefaultNameClaimType,
                    ClaimsIdentity.DefaultRoleClaimType
                    ),
            Expires = DateTime.UtcNow.AddDays(7),
            SigningCredentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256Signature),
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);

        return new TokenResponseDTO { Token = tokenHandler.WriteToken(token) };
    }
  }
}
using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;
using UniverseKino.Data.Interfaces;
using UniverseKino.Services.Interfaces;
using System.Linq;

namespace UniverseKino.Services.Services
{
  public class CheckService : ICheckService
```

```
{
    private readonly IUnitOfWorkEntities _uow;

    public CheckService(IUnitOfWorkEntities uow)
    {
        _uow = uow;
    }

    public bool ValidationMovie(Movie movie)
    {
        return movie != null && movie.Duration <= 0;
    }

    public bool IsExistMovie(Movie movie)
    {
        var checkMovie = _uow.Movies.Find(x => x.Name == movie.Name
                                    && x.Genre == movie.Genre
                                    && x.Duration == movie.Duration).FirstOrDefault();

        return checkMovie != null;
    }



    public void ValidationSession(Session session)
    {
        if (session.Date < DateTime.Now)
        {
            throw new Exception();
        }
```

```
        }
    }
}

using System.Linq;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using AutoMapper;
using UniverseKino.Data.Entities;
using UniverseKino.Data.Interfaces;
using UniverseKino.Services.Dto;

namespace UniverseKino.Services.Services
{
    public interface IInfoMoviesService
    {
        MovieDTO GetMovieByID(int id);
        MovieDTO GetMovieByName(string movieName);
        List<SessionDTO> GetMoviesSessions(int id);
        List<MovieDTO> GetAllMovies();



    }
    public class InfoMoviesService : IInfoMoviesService
    {
        private IUnitOfWorkEntities _unit;
        private IMapper _mapper;
        public InfoMoviesService(IUnitOfWorkEntities unit, IMapper mapper)
        {
```

```
    _unit = unit;
    _mapper = mapper;
}


public List<MovieDTO> GetAllMovies()
{
    var movies = _unit.Movies.GetAll();

    return _mapper.Map<List<MovieDTO>>(movies);
}

public MovieDTO GetMovieByID(int id)
{
    var movie = _unit.Movies.GetById(id);

    return _mapper.Map<MovieDTO>(movie);
}

public MovieDTO GetMovieByName(string movieName)
{
    var movie = _unit.Movies.Find(m =>
                m.Name.ToLower() == movieName.ToLower())
                .FirstOrDefault();

    if (movie == null)
        throw new Exception();

    var movieDTO = _mapper.Map<MovieDTO>(movie);
```

```
            return movieDTO;
        }

        public List<SessionDTO> GetMoviesSessions(int id)
        {
            var movie = _unit.Movies.GetById(id);

            var sessions = movie.Sessions.OrderBy(s => s.Date).ToList();

            var sessionsDTO = _mapper.Map<List<SessionDTO>>(sessions);

            return sessionsDTO;
        }
    }
}
using AutoMapper;
using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Entities;
using UniverseKino.Data.Interfaces;
using UniverseKino.Services.Dto;
using UniverseKino.Services.Interfaces;

namespace UniverseKino.Services.Services
{
    public class ManageMoviesService : IManageMoviesService
    {
        private readonly IUnitOfWorkEntities _uow;
        private readonly IMapper _mapper;
```

```
    private readonly ICheckService _check;

    public ManageMoviesService(IUnitOfWorkEntities uow, IMapper mapper, ICheckService
check)
    {
        _uow = uow;
        _mapper = mapper;
        _check = check;
    }

    public void Add(MovieDTO newMovie)
    {
        var movie = _mapper.Map<Movie>(newMovie);

        if (!_check.ValidationMovie(movie))
        {
            throw new Exception("Invalid data");
        }

        _uow.Movies.Add(movie);

        if (!_check.IsExistMovie(movie))
        {
            throw new Exception("Failed to add movie");
        }
    }

    public void Remove(int id)
    {
        var movie = _uow.Movies.GetById(id);
```

```
            if (movie == null)
            {
                throw new Exception("IsNotExist");
            }

            _uow.Movies.Remove(id);
        }
    }
}

using System;
using System.Linq;
using System.Security.Cryptography;
using Microsoft.Extensions.Options;

namespace UniverseKino.Services.Services
{
    public sealed class HashingOptions
    {
        public int Iterations { get; set; } = 10000;
    }
    public sealed class PasswordHasher
    {
        private const int SaltSize = 16; // 128 bit
        private const int KeySize = 32; // 256 bit

        public PasswordHasher()
        {
            Options = new HashingOptions();
        }
```

```
private HashingOptions Options { get; }

public string Hash(string password)
{
    using (var algorithm = new Rfc2898DeriveBytes(
      password,
      SaltSize,
      Options.Iterations,
      HashAlgorithmName.SHA512))
    {
        var key = Convert.ToBase64String(algorithm.GetBytes(KeySize));
        var salt = Convert.ToBase64String(algorithm.Salt);

        return $"{Options.Iterations}.{salt}.{key}";
    }
}

public bool Check(string hash, string password)
{
    var parts = hash.Split('.', 3);

    if (parts.Length != 3)
    {
        throw new FormatException("Unexpected hash format. " +
          "Should be formatted as `{iterations}.{salt}.{hash}`");
    }

    var iterations = Convert.ToInt32(parts[0]);
    var salt = Convert.FromBase64String(parts[1]);
```

```
            var key = Convert.FromBase64String(parts[2]);

            var needsUpgrade = iterations != Options.Iterations;

            using (var algorithm = new Rfc2898DeriveBytes(
              password,
              salt,
              iterations,
              HashAlgorithmName.SHA512))
            {
                var keyToCheck = algorithm.GetBytes(KeySize);

                var verified = keyToCheck.SequenceEqual(key);

                return verified;
            }
        }
    }
}
using AutoMapper;
using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Data.Interfaces;
using UniverseKino.Services.Dto;
using UniverseKino.Services.Interfaces;
using System.Linq;

namespace UniverseKino.Services.Services
{
```

```
public class SessionsInfoService : ISessionsInfoService
{
    private readonly IUnitOfWorkEntities _uow;
    private readonly IMapper _mapper;

    public SessionsInfoService(IUnitOfWorkEntities uow, IMapper mapper)
    {
        _uow = uow;
        _mapper = mapper;
    }

    public List<SessionDTO> GetAllSessions()
    {
        var sessions = _uow.Sessions.GetAll().OrderBy(x => x.Date).ToList();

        var sessionsDTO = _mapper.Map<List<SessionDTO>>(sessions);

        return sessionsDTO;
    }

    public SessionDTO GetSession(int id)
    {
        var session = _uow.Sessions.GetById(id);

        if (session == null)
            throw new Exception("DataNotExist");

        var sessionDTO = _mapper.Map<SessionDTO>(session);

        return sessionDTO;
```

```
        }

        public List<SessionDTO> GetSessionsByMovie(int idMovie)
        {
            var movie = _uow.Movies.GetById(idMovie);

            if (movie == null)
                throw new Exception("Movie is not exist");

            var sessions = _uow.Sessions.Find(s => s.IdMovie == idMovie).OrderBy(s =>
s.Date).ToList();

            var sessionsDTO = _mapper.Map<List<SessionDTO>>(sessions);

            return sessionsDTO;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using UniverseKino.Services.Dto;
using UniverseKino.Services.Interfaces;
using UniverseKino.Data.Interfaces;
using System.Linq;
using AutoMapper;
using UniverseKino.Data.Entities;

namespace UniverseKino.Services.Services
{
```

```
public class ViewInfoService : IViewInfoService
{
    private readonly IUnitOfWorkEntities _uow;
    private readonly IMapper _mapper;

    public ViewInfoService(IUnitOfWorkEntities uow, IMapper mapper)
    {
        _uow = uow;
        _mapper = mapper;
    }

    public MovieDTO GetMovie(int id)
    {
        var movie = _uow.Sessions.GetById(id);

        var movies = _mapper.Map<MovieDTO>(movie);

        return movies;
    }

    public List<MovieDTO> GetMovies()
    {
        var movies = _uow.Movies.GetAll().ToList();

        var moviesDTO = _mapper.Map<List<MovieDTO>>(movies);

        return moviesDTO;
    }

    public SessionDTO GetSession(int id)
```

```
        {
            var session = _uow.Sessions.GetById(id);

            var sessionDTO = _mapper.Map<SessionDTO>(session);

            return sessionDTO;
        }

        public List<SessionDTO> GetSessions()
        {
            var sessions = _uow.Sessions.GetAll().OrderBy(x => x.Date).ToList();

            var sessionDTO = _mapper.Map<List<SessionDTO>>(sessions);

            return sessionDTO;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace UniverseKino.WEB
{
    [Route("admin")]
    [ApiController]
```

```
    [Authorize(Roles = "Admin")]
    public class AdminController : ControllerBase
    {
        [HttpPost("sessions")]
        public IActionResult CreateSessinons([FromBody] CreateSessionRequestModel
newSession)
        {
            return Ok();
        }

        [HttpDelete("sessions/{id}")]
        public IActionResult DeleteSessinons([FromRoute]int id)
        {
            return Ok();
        }

        [HttpPost("movies")]
        public IActionResult CreateMovies([FromBody] CreateMovieRequestModel newMovie)
        {
            return Ok();
        }

        [HttpDelete("movies/{id}")]
        public IActionResult DeleteMovies([FromRoute] int id)
        {
            return Ok();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using UniverseKino.Services;
using UniverseKino.WEB.Models;

namespace UniverseKino.WEB
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private IAuthService _authServ;
        private IMapper _mapper;
        public AuthController(IAuthService authServ, IMapper mapper)
        {
            _authServ = authServ;

            _mapper = mapper;
        }
        [HttpGet]
        [Route("users")]
        public IActionResult AllUsers() => Ok(_authServ.AllUsers());

        [HttpPost]
        [Route("Registration")]
```

```
        public async Task<IActionResult> Registration([FromBody] RegistrationRequestView
data)
        {
            var RegisterServiceDTO = _mapper.Map<RegistrationRequestDTO>(data);
            var token = await _authServ.Register(RegisterServiceDTO);

            if (token == null)
            {
                return BadRequest();
            }

            return Ok(token);
        }

        [HttpPost]
        [Route("Authenticate")]
        public async Task<IActionResult> Authenticate(LoginRequestView data)
        {
            var serviceModel = _mapper.Map<LoginRequestDTO>(data);
            var token = await _authServ.Authenticate(serviceModel);

            if (token == null)
            {
                return BadRequest();
            }

            return Ok(token);
        }
    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace UniverseKino.WEB
{
    [Route("customer")]
    [ApiController]
    [Authorize]
    public class CustomerController : ControllerBase
    {
        [HttpPost("sessions/tobook")]
        public async  Task<IActionResult> ToBook([FromBody] ReservationRequestModel places)
        {
            return await Task.Run( () => Ok());
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using UniverseKino.Services.Dto;
```

```
using UniverseKino.Services.Services;
using UniverseKino.Services.Interfaces;
using UniverseKino.WEB.Models;
using UniverseKino.Services;

namespace UniverseKino.WEB.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class InfoController : ControllerBase
    {
        private IInfoMoviesService _moviesServ;
        private IMapper _mapper;
        private readonly ISessionsInfoService _sessionsInfoService;

        public InfoController(IInfoMoviesService moviesServ, ISessionsInfoService
sessionsInfoService, IMapper mapper)
        {
            _moviesServ = moviesServ;
            _sessionsInfoService = sessionsInfoService;
            _mapper = mapper;
        }



        [HttpGet("sessions")]
        public async Task<IActionResult> GetAllSessions()
        {
            var sessionDTO = _sessionsInfoService.GetAllSessions();

            var sessionModel = _mapper.Map<List<SessionModel>>(sessionDTO);
```

```
    return await Task.Run(() => Ok(sessionModel));
}

[HttpGet]
[Route("sessions/{id}")]
public async Task<IActionResult> GetSession([FromRoute] int id)
{
    var sessionDTO = _sessionsInfoService.GetSession(id);

    var sessionModel = _mapper.Map<SessionModel>(sessionDTO);

    return await Task.Run(() => Ok(sessionDTO));
}

[HttpGet("movies")]
public IActionResult GetAllMovies()
{
    var movies = _moviesServ.GetAllMovies();

    return Ok(_mapper.Map<List<MovieDTO>>(movies));
}

[HttpGet("movies/{id}")]
public IActionResult GetMovie([FromRoute] int id)
{
    var movie = _moviesServ.GetMovieByID(id);

    return Ok(_mapper.Map<MovieDTO>(movie));
}
```

```
        [HttpGet("movies/{id}/sessions")]
        public IActionResult GetSessionsMovie([FromRoute] int id)
        {
            var sessions = _moviesServ.GetMoviesSessions(id);

            return Ok(_mapper.Map<List<SessionDTO>>(sessions));
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace UniverseKino.WEB.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
```

```
        [HttpGet("{id}")]
        public ActionResult<string> Get(int id)
        {
            return "value";
        }

        // POST api/values
        [HttpPost]
        public void Post([FromBody] string value)
        {
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] string value)
        {
        }

        // DELETE api/values/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
        }
    }
}

using Autofac;
using AutoMapper;
using System.Collections.Generic;
using UniverseKino.Services;
```

```
using UniverseKino.Services.Dto;
using UniverseKino.WEB.Models;

namespace UniverseKino.WEB
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            CreateMap<RegistrationRequestView, RegistrationRequestDTO>();

            CreateMap<LoginRequestView, LoginRequestDTO>();

            CreateMap<SeatModel, SeatDTO>();
            CreateMap<SeatDTO, SeatModel>();

            CreateMap<SessionModel, SessionDTO>();
            CreateMap<SessionDTO, SessionModel>();
        }
    }

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace UniverseKino.WEB
{
    public class CreateMovieRequestModel
    {
```

```
        public string Name { get; set; }
        public string Genre { get; set; }
        public int Duration { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace UniverseKino.WEB
{
    public class CreateSessionRequestModel
    {
        public DateTime Date { get; set; }
        public int NumberHall { get; set; }
        public string NameMovie { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace UniverseKino.WEB
{
    public class Place
    {
```

```
        public int NumberSeat { get; set; }
        public int NumberRow { get; set; }
    }
}

using UniverseKino.Core;

namespace UniverseKino.WEB.Models
{
    public class LoginRequestView
    {
        public string Email { get; set; }
        public string Password { get; set; }

    }
    public class RegistrationRequestView
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string UserName { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace UniverseKino.WEB
```

```
{
    public class ReservationRequestModel
    {
        public int SessionsId { get; set; }
        public List<Place> PlacesToBook { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace UniverseKino.WEB.Models
{
    public class SeatModel
    {
        public decimal Cost { get; set; }

        public int Row { get; set; }

        public int Number { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace UniverseKino.WEB.Models
{
    public class SessionModel
    {
        public DateTime Date { get; set; }

        public string NameMovie { get; set; }

        public string GenreMovie { get; set; }

        public int DurationMovie { get; set; }

        public int NumberHall { get; set; }

        public List<SeatModel> Seats { get; set; }
    }
}

namespace UniverseKino.WEB.Models
{
    public class TokenResponseView
    {
        public string Token { get; set; }
    }
}
```