**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**Факультет комп’ютерних наук та кібернетикиКафедра інтелектуальних програмних систем**

**Курсова робота**

за спеціальністю 121 Інженерія програмного забезпечення:

на тему:

**РОЗРОБКА WEB-СЕРВІСІВ НА ОСНОВІ АРХІТЕКТУРИ REST ВИКОРИСТОВУЮЧИ ФРЕЙМВОРК WEB-РОЗРОБКИ ASP.NET**

Виконав студент 3-го курсу

Ткачук Олексій Васильович \_\_\_\_\_\_\_\_

(підпис)

Науковий керівник:

Доцент

Катеринич Лариса Олександрівна \_\_\_\_\_\_\_\_

(підпис)

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

Студент \_\_\_\_\_\_\_\_

(підпис)

**Київ – 2022**

# **РЕФЕРАТ**

Обсяг роботи # сторінок, # ілюстрацій, # джерел посилань.Для курсової роботи буде реалізовано web-сервіс з бронювання номера готелю. В системі взаємодії цього web-сервісу є власники готелю, користувачі, які хочуть забронювати номер в готелі і власне сам web-сервіс.

Може виникнути питання чому для реалізації сервісу для бронювання номера в готелі необхідно застосувати технологію web-сервісу. Відповідь в тому, що наш сервіс з бронювання номера готелю може бути не єдиним шляхом, через який люди можуть забронювати номер. У нас є три основні випадки бронювання відносно нашого сервісу: людина забронювала номер через наш сервіс, людина забронювала номер на місці в самому готелі, людина забронювала номер на іншому сервісі бронювання. Якщо людина забронювала конкретний номер, на якесь конкретне число на місці в самому готелі, то треба буде оновити інформацію про бронювання конкретного номера на всіх сервісах бронювання для того, щоб користувачі сервісу повторно не забронювали готель який вже зданий. Оскільки для зручності і надійності вигідніше використовувати автоматизовані системи, то розробникам програмного забезпечення для адміністрування готелем вигідно реалізувати систему в якій данні про статус бронювання номерів в готелі синхронізувались автоматично. У випадку якщо розробники сервісів з бронювання номера готелю будуть використовувати не стандартизовані способи для обміну інформації, то це скоріше за все приведе то ситуації, в якій розробник програмного забезпечення для адміністрування готелем має зробити громіздку надбудову для того, щоб зробити реалізацію підтримки не підтримуваного за замовчуванням засобу передачі інформації через інтернет в певному середовищі розробки. Тому для розробника сервісу з бронювання номера готелю набагато вигідніше використати стандартизований спосіб обміну інформації між самим сервісом і власниками готелю.

**ЗМІСТ**

[РЕФЕРАТ 2](#_Toc104674250)

[ВСТУП 5](#_Toc104674251)

[СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 6](#_Toc104674252)

[1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ 7](#_Toc104674253)

[1.1 Принцип роботи web-сервісів та їх типи 7](#_Toc104674254)

[1.2 Архітектурний стиль REST 8](#_Toc104674255)

[1.3 Архітектурні властивості та обмеження REST 9](#_Toc104674256)

[1.4 Структура HTTP запитів і відповідей 10](#_Toc104674257)

[2 ЗАСТОСУВАННЯ ASP.NET ДЛЯ РОЗРОБКИ WEB-CЕРВІСУ І WEB-ЗАСТОСУНКУ 11](#_Toc104674258)

[2.1 Вступ до ASP.NET 11](#_Toc104674259)

[2.2 Розробка RESTful web-сервісу на ASP.NET Web API 12](#_Toc104674260)

[2.3 Розробка web-застосунку на ASP.NET Core MVC 13](#_Toc104674261)

[2.4 Аутентифікація і JWT 14](#_Toc104674262)

[3 ТЕСТУВАННЯ WEB-СЕРВІСУ 15](#_Toc104674263)

[3.1 Swagger 15](#_Toc104674264)

[3.2 Керуюча панель власника готелю 16](#_Toc104674265)

[ВИСНОВКИ 17](#_Toc104674266)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 18](#_Toc104674267)

[ДОДАТОК 1 UML діаграми веб-сервісу бронювання готелю 19](#_Toc104674268)

[ДОДАТОК 2 Реляційна модель бази даних web-сервісу 20](#_Toc104674269)

# **ВСТУП**

Web-сервіс — це набір відкритих протоколів і стандартів, які дозволяють обмінюватися даними між різними програмами або системами. Web-сервіси можуть використовувати програми, написаними різними мовами програмування та запущеними на різних платформах, для обміну даними через комп’ютерні мережі, такі як Інтернет, подібно до міжпроцесної комунікації на одному комп’ютері.

Будь-яке програмне забезпечення, додаток або хмарна технологія, що використовує стандартизовані веб-протоколи (HTTP або HTTPS) для підключення, взаємодії та обміну повідомленнями даних через Інтернет, вважається веб-сервісом.

Перевага веб-сервісів полягає в тому, що вони дозволяють програмам, розробленим на різних мовах, підключатися один до одного, шляхом обміну даних через веб-сервіс між клієнтами і серверами. Клієнт викликає веб-сервіс, надсилаючи HTTP запит, на який служба відповідає HTTP відповідь.

**Актуальність роботи** обумовлена високим рівнем потреби в застосуванні web-сервісів для розробки сучасних розрахунків, оскільки в сьогодення існують безліч засобів для розробки web-сервісів і розробникам зручніше використати уніфікований стандарт передачі інформації ніж вчити засіб передачі даних в кожному окремому засобі розробки. Це дає змогу розробнику зекономити величезну кількість часу без жодного компромісу в плані швидкодії застосунку та їх функціональності.

**Мета й завдання.** Метою курсової роботи є реалізація web-сервісу і демонстрації принципів розробки web-сервісів та їх застосування, використовуючи фреймворк веб-розробки ASP.NET.

# **СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ**

**IDE** – Integrated Design Environment, інтегроване середовище розробки;

**Web** (з англійської “павутина”) – одне з означень Всесвітньої Павутини

**XML** (*eXtensible Markup Language*) – мова розмітки та формат файлу для зберігання, передачі та відновлення довільних даних.

**REST** (*Representational State Transfer)* – це архітектурний стиль програмного забезпечення, який був створений для опису проектування і розробки архітектури для всесвітньої павутини.

**Web Service –** web-сервіс

**RESTful Web Service** – web-сервісстворений на основі архітектури REST

**API (***Application Programming Interface* – з англійської “програмний інтерфейс застосунку” – опис способу взаємодії однієї програми з іншою

**Кросплатформність** – властивість програмного забезпечення працювати більш ніж на одній операційній системі

**JSON**(*JavaScript Object Notation*) – це текстовий формат обміну даними між комп'ютерами.

**JWT**(JSON Web Token) – це стандарт токена доступу на основі JSON

# **1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ**

## **1.1 Принцип роботи web-сервісів та їх типи**

Веб-сервіси використовують метод запит-відповідь для зв’язку між додатками. Для будь-якої комунікації нам потрібен носій і загальний формат, який може бути зрозумілий кожному, у випадку веб-сервісів середовищем є Інтернет, а загальним форматом є формат XML, оскільки кожна мова програмування може зрозуміти мову розмітки XML.

Клієнт - це той, хто запитує певну послугу від сервера, який відомий як постачальник послуг. Запит надсилається через повідомлення в загальному форматі XML, і у відповідь на цей запит постачальник послуг відповідає повідомленням у загальному форматі (тобто XML).

Існують два основні типи типи веб-сервісів:

1. Веб-сервіси SOAP: Цей протокол побудований на основі XML. Цей протокол не залежать від мови програмування і може працювати на будь-якій платформі. SOAP підтримує операції з визначенням стану та без збереження стану. Визначення стану означає, що сервер відстежує інформацію, отриману від клієнта за кожним запитом. У той час як відсутність збереження стану означає, що кожен запит містить достатньо інформації про стан клієнта, і, таким чином, серверу не потрібно турбуватися про збереження стану клієнта, що збільшує швидкість зв'язку.
2. Веб-сервіси RESTful: це розшифровка від Representational State Transfer. Вони також незалежні від мови програмування та платформи і є швидшими в порівнянні з SOAP. Сьогодні веб-сервіси RESTful використовуються частіше, ніж SOAP. Вони розглядають дані як ресурси. Веб-сервіси RESTful повертають дані у форматі JSON або рідше в XML. Ці веб-сервіси створюють об’єкт і надсилають стан об’єкта у відповідь на запит клієнта.

## **1.2 Архітектурний стиль REST**

REST – це архітектурний стиль, який визначає набір обмежень, які будуть використовуватися для створення веб-сервісів. REST API – це спосіб доступу до веб-сервісів простим і гнучким способом без необхідності обробки даних. Комунікація через REST API застосовує тільки HTTP запит.

У HTTP є чотири основні методи, які зазвичай використовуються в архітектурі на основі REST. Вони відповідають операціям створення, читання, оновлення та видалення даних.

1. GET: метод HTTP GET використовується для отримання ресурсу. У випадку успішного виконання GET повертає данні в форматі JSON і код відповіді HTTP 200. У випадку не знаходження ресурсів відповідних до запиту повертає код помилки 404, а у випадку невірного запиту повертає код помилки 400
2. POST: використовується для створення нових ресурсів. Після успішного створення повертає код HTTP 201, разом з новоствореними даними.
3. PUT: Використовується для оновлення існуючих даних. Однак PUT також можна використовувати для створення ресурсу в тому випадку, коли ідентифікатор ресурсу вибирається клієнтом, а не сервером. Після успішного оновлення повертає код HTTP 200. Якщо PUT використовується для створення, то повертає код HTTP 201 після успішного створення.
4. DELETE: Використовується для видалення ресурсу. Після успішного видалення повертає код HTTP 200 разом із тілом відповіді.

Варто зазначити, що REST не є реалізацією для передачі запитів даних і запитів, але лише описує правила поведінки. Для web-служб, побудованих на архітектурі REST, застосовують термін RESTful.

## **1.3 Архітектурні властивості та обмеження REST**

*Архітектурні властивості:*

* висока продуктивність, за рахунок використовування кешування
* масштабованість, що дозволяє підтримувати велику кількість компонентів і взаємодій між компонентами
* простота єдиного інтерфейсу
* портативність компонентів
* легкість внесення модифікацій

*Архітектурні обмеження:*

Архітектурний стиль REST визначає шість обов’язкових обмежень. Коли ці обмеження застосовуються до архітектури системи, то вона набуває бажаних властивостей.

Обов’язковими обмеженнями є:

1. Клієнт-серверна модель

Відокремлення потреб інтерфейсу клієнту від потреб серверу, який зберігає дані, підвищує портативність модулей клієнтського коду на інші платформи, а спрощення серверної частини підвищує масштабованість.

1. Відсутність стану

Обмеження "відсутність стану" не означає, що архітектура REST дозволяє будувати лише системи, в яких немає стану. Відсутність стану означає, що сервер не знає про стан клієнта і не повинен запам'ятовувати послідовність здійснених до нього запитів, тому що клієнт і сервер є незалежним одне до одного.

1. Кешування

Добре налаштоване кешування частково або повністю мінімізує взаємодії клієнт-сервер, ще більше покращуючи масштабованість і продуктивність. Клієнту не буде мати потреби звертатися до сервера за тими самими даними, які він вже має.

1. Однорідність інтерфейсу

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.

1. Багатошаровість системи

Кожен компонент потрапляє в якийсь шар і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмеження знання системи одним шаром зменшує складність компонентів.

## **1.4 Структура HTTP запитів і відповідей**

HTTP запити та відповіді мають близьку структуру. Вони складаються з:

1. Стартового рядку, який описує запит, або статус (успіх або помилка). Це завжди один рядок.
2. Довільного набору заголовків HTTP, що визначають запит або описують тіло повідомлення.
3. Пустий рядок, що вказує, що вся мета інформація надіслана/отримана.
4. Довільного тіла, що містить дані, що пересилаються із запитом або документ, що відправляється у відповідь. Наявність тіла та його розмір визначається стартовим рядком та заголовками HTTP.

Стартовий рядок разом із заголовками повідомлення HTTP називають головою запиту, я його дані - тілом.

# **2 ЗАСТОСУВАННЯ ASP.NET ДЛЯ РОЗРОБКИ WEB-CЕРВІСУ І WEB-ЗАСТОСУНКУ**

## **2.1 Вступ до ASP.NET**

ASP.NET – це кросплатформний фреймворк з відкритим вихідним кодом для створення web-застосунків і web-сервісів. Для реалізації web-сервісу для бронювання номера готелю використаємо найновішу версію ASP.NET на день написання курсової роботи – ASP.NET Core 6.0

Перевагами для вибору ASP.NET Core є:

* Можливість розробки веб-застосунків та веб-служб
* Розробка для платформ Windows, Linux та MacOS
* Використовує мову програмування C#
* Має набагато більшу швидкість оброблення великої кількості запитів в порівнянні з PHP і NodeJS
* Наявність зручного пакетного менеджера NuGet

Для реалізації нашого веб-сервісу будуть застосовані наступні компоненти фреймворку ASP.NET: Web API і ASP.NET Core MVC.

Web API – засіб побудови веб-сервісів, який спеціально заточений для роботи в стилі REST. Даний компонент застосований для створення веб інтерфейсів, які будуть використовувати власники готелів для оновлення і додавання інформації їх готелів і номерів в них.

ASP.NET Core MVC – це частина платформи ASP.NET Core, його ключовою особливістю є застосування патерну MVC. Перевагою використання фрейморку ASP.NET Core MVC у порівнянні з "чистим" ASP.NET Core є те, що він спрощує в ряді ситуацій та сценаріїв організацію та створення програм, особливо це стосується розробки великих програм. Він буде застосований для розробки веб-застосунку, через який простий користувач зможе робити бронювання номера готелю.

## **2.2 Розробка RESTful web-сервісу на ASP.NET Web API**

Основною частиною розробки web-сервісу є додавання можливості виконання HTTP запитів до нашого сервісу. Для цього в платформі ASP.NET існує гнучкий і простий в застосуванні спосіб створення web-інтерфейсів. Перед початком створення сервісу нам необхідно реалізувати класи моделей, які будуть відправлятись або отримуватись власником готелю під час виконання запитів. Застосовуючи модифікатор [JsonIgnore] до об’єктів класу моделей, ми вказуємо про те, що даний об’єкт треба ігнорувати в HTTP запитах. Далі для роботи з нашими моделями необхідно реалізувати службу для кожної моделі. Для створення служб в ASP.NET необхідно реалізувати інтерфейс з набором методів для роботи з конкретною моделлю і реалізувати клас який буде наслідувати цей інтерфейс і реалізовувати його набір методів. Для роботи служби нам треба додати її в набір служб застосунку. Робиться це шляхом передачі інтерфейсу та класу, що його реалізує, в будівник серверу. Далі буде наведений приклад в якому в будівник(builder) додається служба для користувачів, де IUserService – це інтерфейс служби, а UserService його реалізація.

builder.Services.AddScoped<IUserService, UserService>();

Створення таких служб дає можливість загального доступу контролерам до реалізації методів для роботи з певною моделлю, шляхом додавання в список параметрів конструктора інтерфейс служби. Зазвичай в таких службах описується взаємодія до бази об’єктів конкретної моделі.

Далі ми безпосередньо переходим до реалізації RESTful сервісу. Для створення web-інтерфейсів з доступом за певним шляхом ми створюємо контролери. Вони створюються шляхом створення класів наслідуючих клас ControllerBase. Для надання доступу контролеру доступу до служби певної моделі додаємо в перелік параметрів конструктора інтерфейс служби. Нижче наведений приклад приєднання служби номерів готелей до контролера HotelNumberController

private readonly IHotelNumberService \_hotelNumberService;

public HotelNumberController(IHotelNumberService hotelNumberService)

{

\_hotelNumberService = hotelNumberService;

}

В середині контролера ми реалізовуємо функцію з обробки HTTP запиту. Для роботи функції в якості обробника REST запиту треба додати відповідний модифікатор до цієї функції

1. [HttpPost("назва методу")] – створення обробника методу POST
2. [HttpGet("назва методу")] – створення обробника методу GET
3. [HttpPut("назва методу")] – створення обробника методу PUT
4. [HttpDelete("назва методу")] – створення обробника методу DELETE

Данні які має надсилати зовнішній користувач задаються в списку аргументів функції, а данні які він отримає у відповідь через типом повернення значення функції. Нижче наведений приклад методу функції обробки додавання готелю в наш веб-сервіс

[HttpPost("add\_hotel"), Authorize]

public async Task<ActionResult<Hotel>> AddHotel(HotelDto request)

{

var hotel = new Hotel();

hotel.Name = request.Name;

hotel.City = request.City;

hotel.Address = request.Address;

hotel.User = \_userService.GetUser();

\_hotelService.AddHotel(hotel);

return Ok(hotel);

}

## **2.3 Розробка web-застосунку на ASP.NET Core MVC**

## **2.4 Аутентифікація і JWT**

Однією з найважливіших частин розробки web-сервісу є реалізація системи аутентифікації, так як web-сервіси є публічними службами і розробник має ввести обмеження доступу до певних HTTP запитів. Найбільш доцільним засобом авторизації у web-сервісах є застосування JWT. Це токен, що створюється сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи. В реалізації сервісу з бронювання номера готелю, отримання токену JWT відбувається шляхом виконання HTTP запиту /api/Auth/login. В подальшому власник готелю має прикріплювати отриманий токен в заголовок кожного запиту, який потребує аутентифікації.

В нашому випадку ми маємо використовувати авторизацію, оскільки ми маємо запобігти редагування даних, які не створював даний користувач. В такій ситуації ми маємо індивідуально порівнювати доступ до даних кожного користувача, однак є і варіант авторизації за ролями. В такому типі авторизації певній групі користувачів призначається роль, що дає доступ до HTTP запитів з обмеженням по заданій ролі.

Для реалізації аутентифікації в сервісі бронювання номера готелю була застосовна бібліотека *Microsoft.AspNetCore.Authentication.JwtBearer* додавши в сервіси авторизацію JwtBearer.

# **3 ТЕСТУВАННЯ WEB-СЕРВІСУ**

## **3.1 Swagger**

Swagger – це інструмент з відкритим вихідним кодом, створений для проектування, розробки та тестування API. Основним компонентом цього інструменту, який був застосований для розробки web-сервісу є Swagger UI. Він автоматично створює візуальне представлення для всіх реалізованих HTTP запитів і автоматично створює графічний інтерфейс для взаємодії з ними. Для застосування Swagger UI в ASP.NET Core необхідно встановити пакет Swashbuckle.AspNetCore.SwaggerUI і задати в параметри ініціалізації сервера о його застосуванні наступним чином:

app.UseSwagger();

app.UseSwaggerUI();

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.1 Приклад інтерфейсу SwaggerUI

Варто зазначити, що Swagger не додає ніяких функціональних можливостей для web-сервіса, а значно поліпшує процес його розробки, оскільки дає швидко вручну протестувати HTTP запити, без застосування іншого стороннього програмного забезпечення. Під час розробки web-сервісу

SwaggerUI автоматично генерував графічне представлення розроблених HTTP запитів і дозволяв більш зрічніше тестувати їх в ручному режимі.

## **3.2 Керуюча панель власника готелю**

В системі web-сервісу з бронювання номера готелю у нас є три об’єкти, які взаємодіють між собою: користувач, який хоче забронювати готель, власник готелю, який хоче виставити свої номера готелю для здачі і власне сам web-сервіс. В попередніх розділах було реалізовано web-сервіс і сторінку, через яку користувачі можуть бронювати готель. На даний момент ми можемо продемонструвати роботи нашого сервісу лише вручну додаючи дані готелей, що є не дуже зручним і не повністю відображає можливості web-сервісу, а тому додатково треба реалізувати програмне забезпечення керуючої панелі для адміністрування готелем.

Для реалізації цієї керуючої панелі використаємо мову програмування Python. Вибір цієї мови обумовлюється тим, що ми можемо продемонструвати відсутність прив’язки RESTful сервісів до програмної платформи.

# **ВИСНОВКИ**

Таким чином, під час написання курсової роботи було досліджено сферу web-сервісів. Було з’ясовано при яких випадках нам варто їх використовувати і був описаний стиль архітектурної розробки REST.

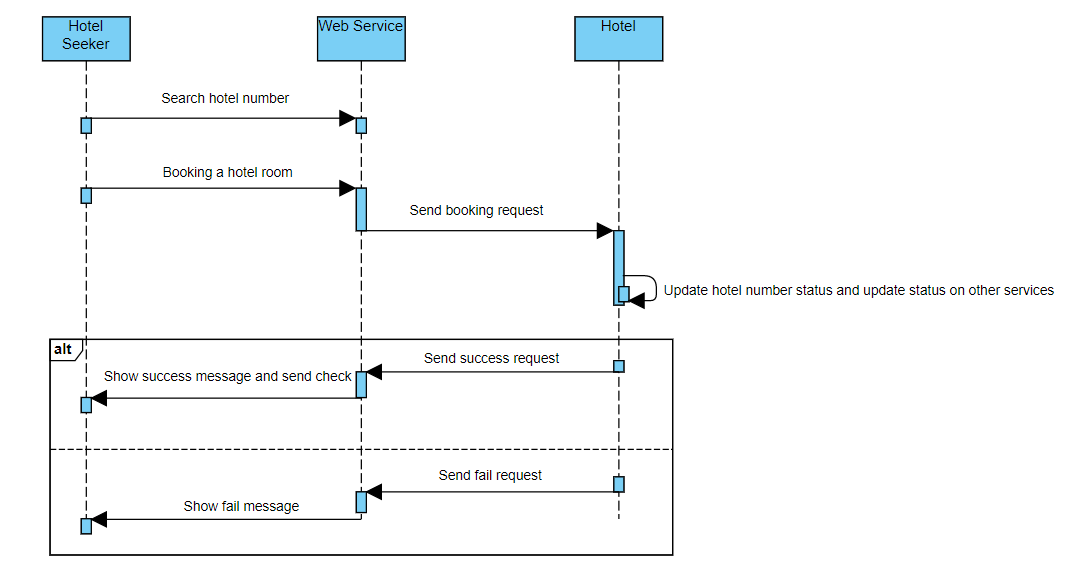
В якості демонстрації прикладу web-сервісу було реалізовано сервіс з бронювання номера готелю. На даному прикладі ми побачили, що для реалізації даного рішення нам необхідно створити можливість для обміну даними між нашою системою і зовнішніми користувачами, без прив’язки до конкретної програмної платформи, оскільки програмне рішення власника готелю може відрізнятись від програмного рішення нашого сервісу, або розробник може не володіти певним засобом мережевої взаємодії.

Для демонстрації працездатності web-сервісу було реалізовано web-застосунок для користувачів, які бажають забронювати номер в готелі, і тестову панель для адміністрування готелем.

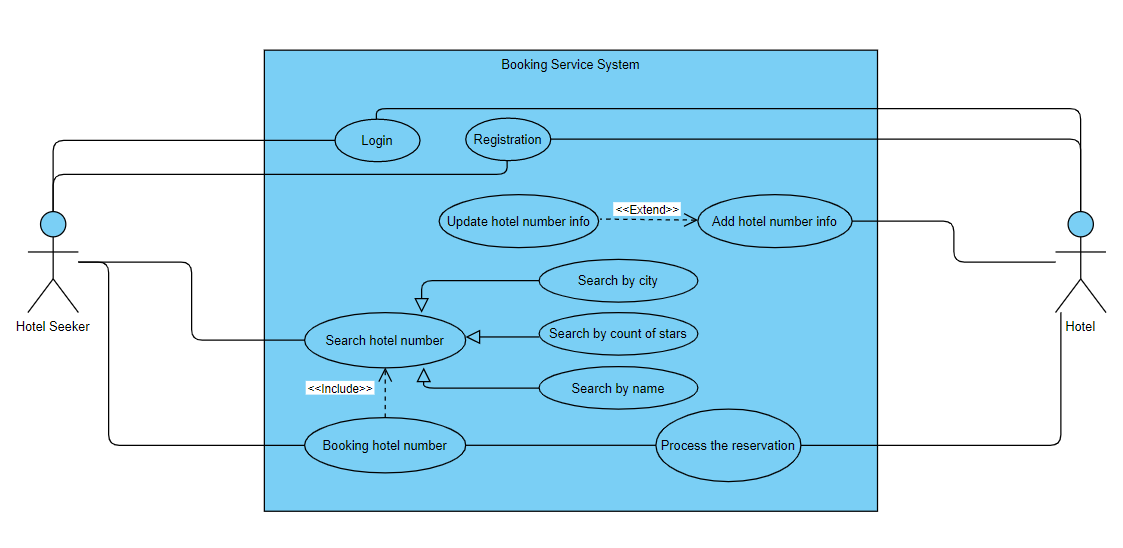
# **ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

|  |
| --- |
| **Електронні ресурси** |
| 1. Create a web API with ASP.NET Core URL: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio> 2. Overview of ASP.NET Core authentication URL: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0> 3. Implement CRUD Functionality - ASP.NET MVC with EF Core URL: <https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/crud?view=aspnetcore-6.0> 4. ASP.NET Core web API documentation with Swagger / OpenAPI URL: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-6.0> 5. HttpContext.User, ClaimPrincipal и ClaimsIdentity URL: <https://metanit.com/sharp/aspnet6/13.5.php> 6. GET and POST requests using Python URL: <https://www.geeksforgeeks.org/get-post-requests-using-python/> 7. Web Services – Definition, Working, Types, Applications URL: <https://www.geeksforgeeks.org/web-services-definition-working-types-applications/> |

# **ДОДАТОК 1 UML діаграми веб-сервісу бронювання готелю**

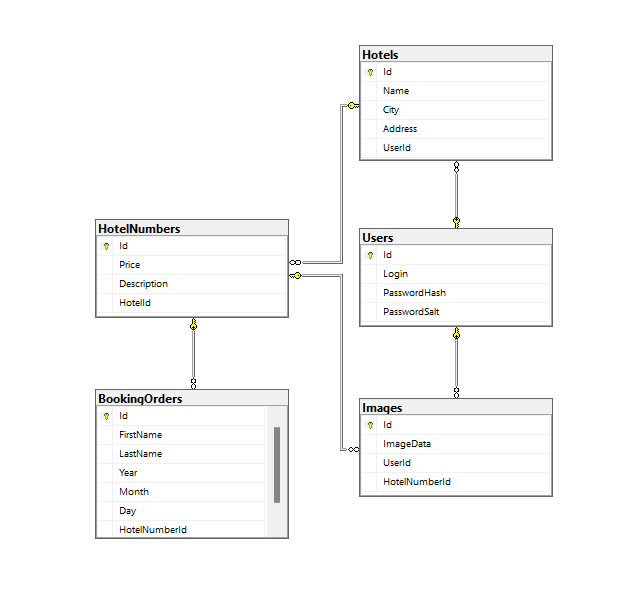


Додаток 1. Рисунок 1. Sequence діаграма процесу резервації готелю користувачем



Додаток 1. Рисунок 2. Use case діаграма процесу взаємодії логіки між власником готелю і веб-сервісом, і між звичайним користувачем і веб-сервісом

# **ДОДАТОК 2 Реляційна модель бази даних web-сервісу**

****