**Models**

**User.cs**

using System.ComponentModel.DataAnnotations;

namespace HotelApplication.Models

{

public class User

{

[Key]

public int UserId { get; set; }

[Required]

public string Username { get; set; }

public byte[] Password { get; set; }

public string Role { get; set; }

public byte[] Key { get; set; }

}

}

**Hotel.cs**

using System.ComponentModel.DataAnnotations;

namespace HotelApplication.Models

{

public class Hotel

{

[Key]

public int HotelId { get; set; }

public string HotelName { get; set; }

public string Location { get; set; }

public int AvailableRooms { get; set; }

public List<Room>? Rooms { get; set; }

}

}

**Room.cs**

using System.ComponentModel.DataAnnotations;

namespace HotelApplication.Models

{

public class Room

{

[Key]

public int RoomId { get; set; }

public int HotelId { get; set; }

public string RoomType { get; set; }

public float Price { get; set; }

public int AvalaibleRooms { get; set; }

public Hotel? Hotel { get; set; }

}

}

**Booking.cs**

using System.ComponentModel.DataAnnotations;

namespace HotelApplication.Models

{

public class Booking

{

[Key]

public int BookingId { get; set; }

public int UserId { get; set; }

public int RoomId { get; set; }

public int HotelId { get; set; }

public DateTime CheckInDate { get; set; }

public DateTime CheckOutDate { get; set; }

public int NoOfPeople { get; set; }

public Room? Room { get; set; }

public User? User { get; set; }

}

}

**UserDTO.cs**

using System.ComponentModel.DataAnnotations;

namespace HotelApplication.Models.DTO

{

public class UserDTO

{

[Required]

public int UserId { get; set; }

[Required(ErrorMessage = "Username cannot be empty")]

public string Username { get; set; }

public string? Role { get; set; }

public string? Token { get; set; }

[Required(ErrorMessage = "Password cannot be empty")]

public string Password { get; set; }

}

}

**Interfaces**

**IRepository**

namespace HotelApplication.Interfaces

{

public interface IRepository<K, T>

{

T GetById(K key);

IList<T> GetAll();

T Add(T entity);

T Update(T entity);

T Delete(K key);

}

}

**IUserService.cs**

using HotelApplication.Models.DTO;

namespace HotelApplication.Interfaces

{

public interface IUserService

{

UserDTO Login(UserDTO userDTO);

UserDTO Register(UserDTO userDTO);

}

}

**ITokenService.cs**

using HotelApplication.Models.DTO;

namespace HotelApplication.Interfaces

{

public interface ITokenService

{

string GetToken(UserDTO user);

}

}

**IRoomRepository.cs**

using HotelApplication.Models;

namespace HotelApplication.Interfaces

{

public interface IRoomRepository

{

List<Room> GetAllRooms();

List<Room> GetAllRoomsByHotelId(int hid);

string AddNewRoom(Room room);

string UpdateRoom(Room room);

string DeleteRoom(int id);

Room GetRoomById(int id);

}

}

**IHotelRepository.cs**

using HotelApplication.Models;

namespace HotelApplication.Interfaces

{

public interface IHotelRepository

{

Hotel GetHotelById(int key);

List<Hotel> GetAllHotels();

string AddHotel(Hotel entity);

string UpdateHotel(Hotel entity);

string DeleteHotel(int key);

}

}

**IBookingRepostiroy.cs**

using HotelApplication.Models;

namespace HotelApplication.Interfaces

{

public interface IBookingRepository

{

List<Booking> GetAllBookings();

string AddNewBooking(Booking booking);

string UpdateBooking(Booking booking);

string DeletBooking(int id);

List<Booking> GetBookingById(int id);

}

}

**Repositories**

**UserRepository.cs**

using HotelApplication.Contexts;

using HotelApplication.Contexts.HotelApplication.Contexts;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApplication.Repositories

{

public class UserRepository : IRepository<string, User>

{

private readonly HotelContext \_context;

public UserRepository(HotelContext context)

{

\_context = context;

}

public User Add(User entity)

{

\_context.Users.Add(entity);

\_context.SaveChanges();

return entity;

}

public User Delete(string key)

{

var user = GetById(key);

if (user != null)

{

\_context.Users.Remove(user);

\_context.SaveChanges();

return user;

}

return null;

}

public IList<User> GetAll()

{

if (\_context.Users.Count() == 0)

return null;

return \_context.Users.ToList();

}

public User GetById(string key)

{

var user = \_context.Users.SingleOrDefault(u => u.Username == key);

return user;

}

public User Update(User entity)

{

var user = GetById(entity.Username);

if (user != null)

{

\_context.Entry<User>(user).State = EntityState.Modified;

\_context.SaveChanges();

return user;

}

return null;

}

}

}

**HotelRepository.cs**

using System;

using System.Collections.Generic;

using HotelApplication.Contexts.HotelApplication.Contexts;

using HotelApplication.Exceptions;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace HotelApplication.Repositories

{

public class HotelRepository : IHotelRepository

{

private readonly HotelContext context;

public HotelRepository(HotelContext hotelContext)

{

context = hotelContext;

}

public List<Hotel> GetAllHotels()

{

return context.Hotels.ToList();

}

public Hotel GetHotelById(int id)

{

Hotel hotel = context.Hotels.Find(id);

if (hotel == null)

{

throw new NotFoundException($"Hotel with ID {id} not found");

}

return hotel;

}

public string AddHotel(Hotel hotel)

{

try

{

int count = context.Hotels.Count();

context.Hotels.Add(hotel);

context.SaveChanges();

int newcount = context.Hotels.Count();

if (newcount > count)

{

return "Record inserted successfully";

}

else

{

throw new DatabaseException("Oops, something went wrong while inserting");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error adding hotel: {ex.Message}");

}

}

public string UpdateHotel(Hotel newhotel)

{

try

{

Hotel hotel = context.Hotels.FirstOrDefault(d => d.HotelId == newhotel.HotelId);

if (hotel == null)

{

throw new NotFoundException($"Hotel with ID {newhotel.HotelId} not found");

}

hotel.HotelName = newhotel.HotelName;

hotel.Location = newhotel.Location;

hotel.AvailableRooms = newhotel.AvailableRooms;

context.SaveChanges();

return "Hotel details updated successfully";

}

catch (Exception ex)

{

throw new DatabaseException($"Error updating hotel: {ex.Message}");

}

}

public string DeleteHotel(int id)

{

try

{

Hotel hotel = context.Hotels.Find(id);

if (hotel == null)

{

throw new NotFoundException($"Hotel with ID {id} not found");

}

context.Hotels.Remove(hotel);

context.SaveChanges();

return "Hotel removed from database";

}

catch (Exception ex)

{

throw new DatabaseException($"Error deleting hotel: {ex.Message}");

}

}

}

}

**RoomRepository.cs**

using HotelApplication.Contexts.HotelApplication.Contexts;

using HotelApplication.Exceptions;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using System;

namespace HotelApplication.Repositories

{

public class RoomRepository : IRoomRepository

{

HotelContext context;

public RoomRepository(HotelContext roomContext)

{

context = roomContext;

}

public string AddNewRoom(Room room)

{

try

{

int count = context.Rooms.Count();

context.Rooms.Add(room);

context.SaveChanges();

int newcount = context.Rooms.Count();

if (newcount > count)

{

return "Record inserted successfully";

}

else

{

throw new DatabaseException("Oops, something went wrong while inserting");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error adding room: {ex.Message}");

}

}

public string DeleteRoom(int id)

{

try

{

Room room = context.Rooms.Find(id);

if (room != null)

{

context.Rooms.Remove(room);

context.SaveChanges();

return "Room removed successfully";

}

else

{

throw new NotFoundException("Room not available");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error deleting room: {ex.Message}");

}

}

public List<Room> GetAllRooms()

{

try

{

return context.Rooms.ToList();

}

catch (Exception ex)

{

throw new DatabaseException($"Error getting all rooms: {ex.Message}");

}

}

public string UpdateRoom(Room newroom)

{

try

{

Room room = context.Rooms.FirstOrDefault(d => d.RoomId == newroom.RoomId);

if (room != null)

{

room.RoomId = newroom.RoomId;

room.HotelId = newroom.HotelId;

room.AvalaibleRooms = newroom.AvalaibleRooms;

room.Price = newroom.Price;

room.RoomType = newroom.RoomType;

context.SaveChanges();

return "Room details updated successfully";

}

else

{

throw new NotFoundException("Room details not found");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error updating room: {ex.Message}");

}

}

public Room GetRoomById(int id)

{

try

{

Room room = context.Rooms.Find(id);

if (room != null)

{

return room;

}

else

{

throw new NotFoundException($"Room with ID {id} not found");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error getting room by ID: {ex.Message}");

}

}

public List<Room> GetAllRoomsByHotelId(int hid)

{

try

{

List<Room> roomlist = context.Rooms.Where(d => d.HotelId == hid).ToList();

return roomlist;

}

catch (Exception ex)

{

throw new DatabaseException($"Error getting rooms by hotel ID: {ex.Message}");

}

}

}

}

**BookingRepository.cs**

using System;

using System.Collections.Generic;

using HotelApplication.Contexts;

using HotelApplication.Contexts.HotelApplication.Contexts;

using HotelApplication.Exceptions;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApplication.Repositories

{

public class BookingRepository : IBookingRepository

{

HotelContext context;

public BookingRepository(HotelContext bookingContext)

{

context = bookingContext;

}

public string AddNewBooking(Booking booking)

{

try

{

int count = context.Bookings.Count();

context.Bookings.Add(booking);

context.SaveChanges();

int newCount = context.Bookings.Count();

if (newCount > count)

{

List<Booking> bookings = new List<Booking>();

bookings = context.Bookings.ToList();

Booking booking1 = bookings.Last();

int rid = booking1.RoomId;

Room room = context.Rooms.Find(rid);

if (room.AvalaibleRooms < 1)

{

throw new DatabaseException("Oops, something went wrong. No available rooms.");

}

else

{

if (room != null)

{

int a = room.AvalaibleRooms;

a = a - 1;

room.AvalaibleRooms = a;

context.SaveChanges();

}

}

return "Record inserted successfully";

}

else

{

throw new DatabaseException("Oops, something went wrong.");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error adding booking: {ex.Message}");

}

}

public string DeletBooking(int id)

{

try

{

Booking booking = context.Bookings.Find(id);

if (booking != null)

{

int rid = booking.RoomId;

Room room = context.Rooms.Find(rid);

if (room != null)

{

int a = room.AvalaibleRooms;

a = a + 1;

room.AvalaibleRooms = a;

context.SaveChanges();

}

context.Bookings.Remove(booking);

context.SaveChanges();

return "Booking is deleted";

}

else

{

throw new NotFoundException("Booking is not available");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error deleting booking: {ex.Message}");

}

}

public List<Booking> GetAllBookings()

{

try

{

return context.Bookings.ToList();

}

catch (Exception ex)

{

throw new DatabaseException($"Error getting all bookings: {ex.Message}");

}

}

public string UpdateBooking(Booking newbooking)

{

try

{

Booking booking = context.Bookings.FirstOrDefault(d => d.BookingId == newbooking.BookingId);

if (booking != null)

{

booking.BookingId = newbooking.BookingId;

booking.HotelId = newbooking.HotelId;

booking.RoomId = newbooking.RoomId;

booking.CheckInDate = newbooking.CheckInDate;

booking.CheckOutDate = newbooking.CheckOutDate;

booking.NoOfPeople = newbooking.NoOfPeople;

context.SaveChanges();

return "Booking details are updated";

}

else

{

throw new NotFoundException("Booking details are not available");

}

}

catch (Exception ex)

{

throw new DatabaseException($"Error updating booking: {ex.Message}");

}

}

public List<Booking> GetBookingById(int id)

{

try

{

List<Booking> bookinglist = context.Bookings.Where(d => d.UserId == id).ToList();

return bookinglist;

}

catch (Exception ex)

{

throw new DatabaseException($"Error getting bookings by user ID: {ex.Message}");

}

}

}

}

**Services**

**UserService.cs**

using HotelApplication.Interfaces;

using HotelApplication.Models.DTO;

using HotelApplication.Models;

using System.Security.Cryptography;

using System.Text;

namespace HotelApplication.Services

{

public class UserService : IUserService

{

private readonly IRepository<string, User> \_repository;

private readonly ITokenService \_tokenService;

public UserService(IRepository<string, User> repository, ITokenService tokenService)

{

\_repository = repository;

\_tokenService = tokenService;

}

public UserDTO Login(UserDTO userDTO)

{

var user = \_repository.GetById(userDTO.Username);

if (user != null)

{

HMACSHA512 hmac = new HMACSHA512(user.Key);

var userpass = hmac.ComputeHash(Encoding.UTF8.GetBytes(userDTO.Password));

for (int i = 0; i < userpass.Length; i++)

{

if (user.Password[i] != userpass[i])

return null;

}

userDTO.Token = \_tokenService.GetToken(userDTO);

userDTO.Password = "";

return userDTO;

}

return null;

}

public UserDTO Register(UserDTO userDTO)

{

HMACSHA512 hmac = new HMACSHA512();

User user = new User()

{

Username = userDTO.Username,

Password = hmac.ComputeHash(Encoding.UTF8.GetBytes(userDTO.Password)),

Key = hmac.Key,

Role = userDTO.Role

};

var result = \_repository.Add(user);

if (result != null)

{

userDTO.Password = "";

return userDTO;

}

return null;

}

}

}

**TokenService.cs**

using HotelApplication.Interfaces;

using HotelApplication.Models.DTO;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace HotelApplication.Services

{

public class TokenService : ITokenService

{

private readonly SymmetricSecurityKey \_key;

public TokenService(IConfiguration configuration)

{

var secretKey = configuration["SecretKey"].ToString();

\_key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

}

public string GetToken(UserDTO user)

{

var claims = new List<Claim>()

{

new Claim(JwtRegisteredClaimNames.NameId,user.Username),

new Claim("role",user.Role)

};

var cred = new SigningCredentials(\_key, SecurityAlgorithms.HmacSha512Signature);

var tokenDescription = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.Now.AddDays(1),

SigningCredentials = cred

};

var tokenHandler = new JwtSecurityTokenHandler();

var token = tokenHandler.CreateToken(tokenDescription);

return tokenHandler.WriteToken(token);

}

}

}

**HotelContext.cs**

using global::HotelApplication.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApplication.Contexts

{

namespace HotelApplication.Contexts

{

public class HotelContext : DbContext

{

public HotelContext(DbContextOptions<HotelContext> options) : base(options)

{

}

public DbSet<User> Users { get; set; }

public DbSet<Hotel> Hotels { get; set; }

public DbSet<Room> Rooms { get; set; }

public DbSet<Booking> Bookings { get; set; }

}

}

}

**Controllers**

**UserController.cs**

using HotelApplication.Interfaces;

using HotelApplication.Models.DTO;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

namespace HotelApplication.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class UserController : ControllerBase

{

private readonly IUserService \_userService;

public UserController(IUserService userService)

{

\_userService = userService;

}

[HttpPost]

[Route("Register")]//attribute based routing

public ActionResult Register(UserDTO viewModel)

{

string message = "";

try

{

var user = \_userService.Register(viewModel);

if (user != null)

{

return Ok(user);

}

}

catch (DbUpdateException exp)

{

message = "Duplicate username";

}

catch (Exception)

{

}

return BadRequest(message);

}

[HttpPost]

[Route("Login")]//attribute based routing

public ActionResult Login(UserDTO userDTO)

{

var result = \_userService.Login(userDTO);

if (result != null)

{

return Ok(result);

}

return Unauthorized("Invalid username or password");

}

}

}

**HotelController.cs**

using HotelApplication.Exceptions;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using Microsoft.AspNetCore.Mvc;

namespace HotelApplication.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class HotelController : ControllerBase

{

private readonly IHotelRepository Repositories;

public HotelController(IHotelRepository repo)

{

Repositories = repo;

}

[HttpGet]

public ActionResult GetAllHotels()

{

try

{

var hotels = Repositories.GetAllHotels();

return Ok(hotels);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpGet("{id:int}")]

public ActionResult GetHotelById(int id)

{

try

{

var hotel = Repositories.GetHotelById(id);

return Ok(hotel);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpPost]

public ActionResult<string> AddHotel(Hotel hotel)

{

try

{

string response = Repositories.AddHotel(hotel);

return Ok(response);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpPut]

public ActionResult<string> UpdateHotel(Hotel hotel)

{

try

{

string response = Repositories.UpdateHotel(hotel);

return Ok(response);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpDelete("{id:int}")]

public ActionResult<string> DeleteHotel(int id)

{

try

{

string response = Repositories.DeleteHotel(id);

return Ok(response);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

}

}

**RoomsController.cs**

using HotelApplication.Exceptions;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using Microsoft.AspNetCore.Mvc;

namespace HotelApplication.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class RoomsController : ControllerBase

{

private readonly IRoomRepository Repositories;

public RoomsController(IRoomRepository repo)

{

Repositories = repo;

}

[HttpGet]

public ActionResult<IEnumerable<Room>> GetRooms()

{

try

{

var rooms = Repositories.GetAllRooms();

return Ok(rooms);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpGet("{id}")]

public ActionResult<Room> GetRoom(int id)

{

try

{

var room = Repositories.GetRoomById(id);

return Ok(room);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpPut("{id}")]

public ActionResult<string> PutRoom(int id, Room room)

{

try

{

var response = Repositories.UpdateRoom(room);

return Ok(response);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpPost]

public ActionResult<string> PostRoom(Room room)

{

try

{

var response = Repositories.AddNewRoom(room);

return Ok(response);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpDelete("{id}")]

public ActionResult<string> DeleteRoom(int id)

{

try

{

var response = Repositories.DeleteRoom(id);

return Ok(response);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

}

}

**BookingController.cs**

using HotelApplication.Exceptions;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using Microsoft.AspNetCore.Mvc;

namespace HotelApplication.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class BookingController : ControllerBase

{

private readonly IBookingRepository Repositories;

public BookingController(IBookingRepository repo)

{

Repositories = repo;

}

[HttpGet]

public ActionResult<IEnumerable<Booking>> Get()

{

try

{

var bookings = Repositories.GetAllBookings();

return Ok(bookings);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpGet("{id:int}")]

public ActionResult<List<Booking>> Get(int id)

{

try

{

var bookings = Repositories.GetBookingById(id);

return Ok(bookings);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpPost]

public ActionResult<string> Post(Booking booking)

{

try

{

var response = Repositories.AddNewBooking(booking);

return Ok(response);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpPut]

public ActionResult<string> Put(Booking booking)

{

try

{

var response = Repositories.UpdateBooking(booking);

return Ok(response);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

[HttpDelete("{id:int}")]

public ActionResult<string> Delete(int id)

{

try

{

var response = Repositories.DeletBooking(id);

return Ok(response);

}

catch (NotFoundException ex)

{

return NotFound(ex.Message);

}

catch (DatabaseException ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

}

}

**Exceptions**

**DatabaseException.cs**

using System.Runtime.Serialization;

namespace HotelApplication.Exceptions

{

[Serializable]

public class DatabaseException : Exception

{

public DatabaseException() : base("An error occurred while accessing the database.")

{

}

public DatabaseException(string message) : base(message)

{

}

}

}

**NotFoundException.cs**

using System.Runtime.Serialization;

namespace HotelApplication.Exceptions

{

[Serializable]

public class NotFoundException : Exception

{

public NotFoundException() : base("The requested resource was not found.")

{

}

public NotFoundException(string message) : base(message)

{

}

}

}

**Program.cs**

using HotelApplication.Contexts;

using HotelApplication.Contexts.HotelApplication.Contexts;

using HotelApplication.Interfaces;

using HotelApplication.Models;

using HotelApplication.Repositories;

using HotelApplication.Services;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

namespace HotelApplication

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(opt =>

{

opt.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name = "Authorization",

Type = SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT",

In = ParameterLocation.Header,

Description = "JWT Authorization header using the Bearer scheme."

});

opt.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = false,

ValidateAudience = false,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["SecretKey"])),

ValidateIssuerSigningKey = true

};

});

builder.Services.AddDbContext<HotelContext>(opts =>

{

opts.UseSqlServer(builder.Configuration.GetConnectionString("conn"));

});

builder.Services.AddScoped<IBookingRepository,BookingRepository>();

builder.Services.AddScoped<IHotelRepository, HotelRepository>();

builder.Services.AddScoped<IBookingRepository, BookingRepository>();

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

builder.Services.AddScoped<IRepository<string, User>, UserRepository>();

builder.Services.AddScoped<ITokenService, TokenService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

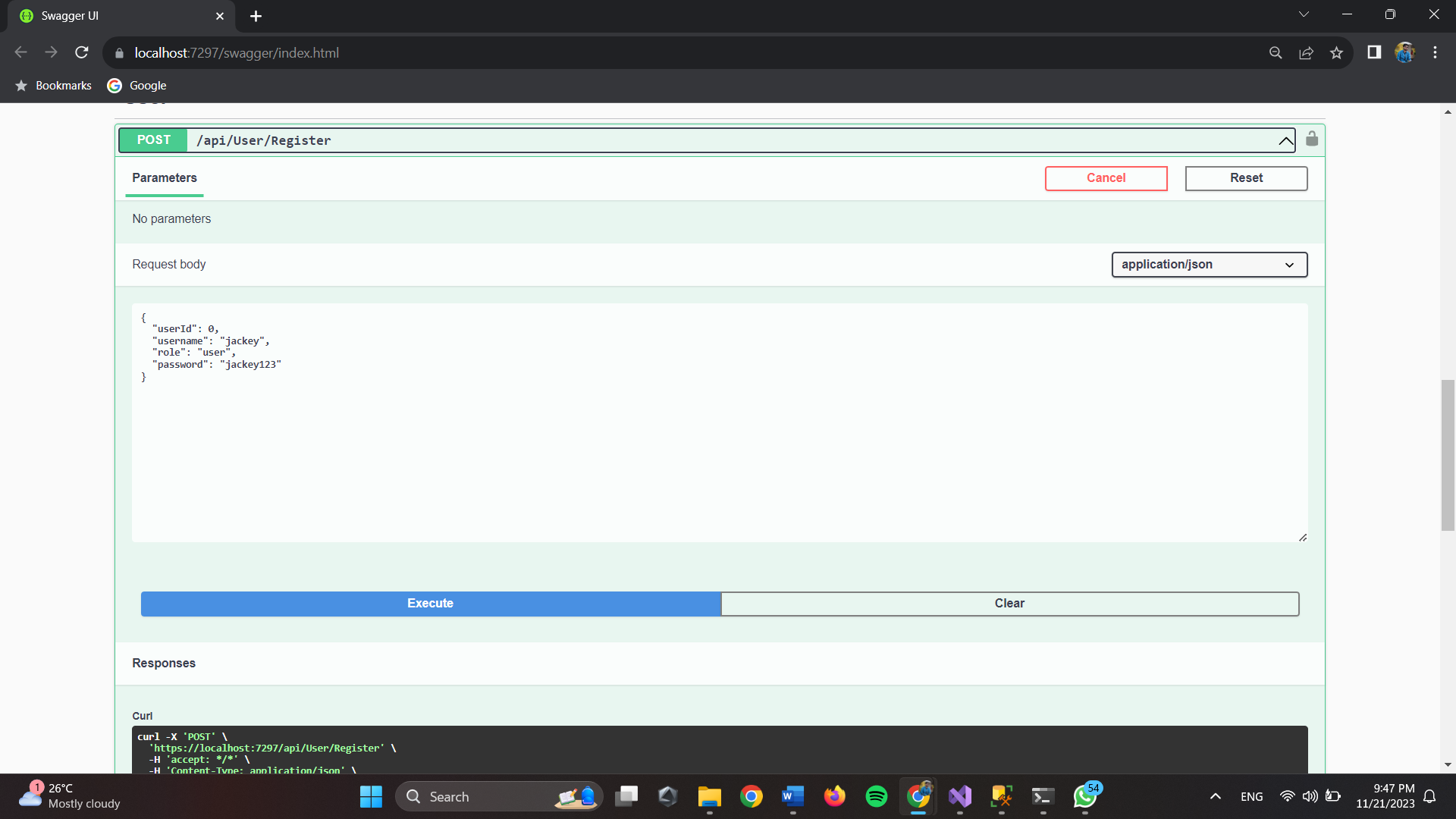
app.Run();

}

}

}

**Outputs:**



A screenshot of a computer

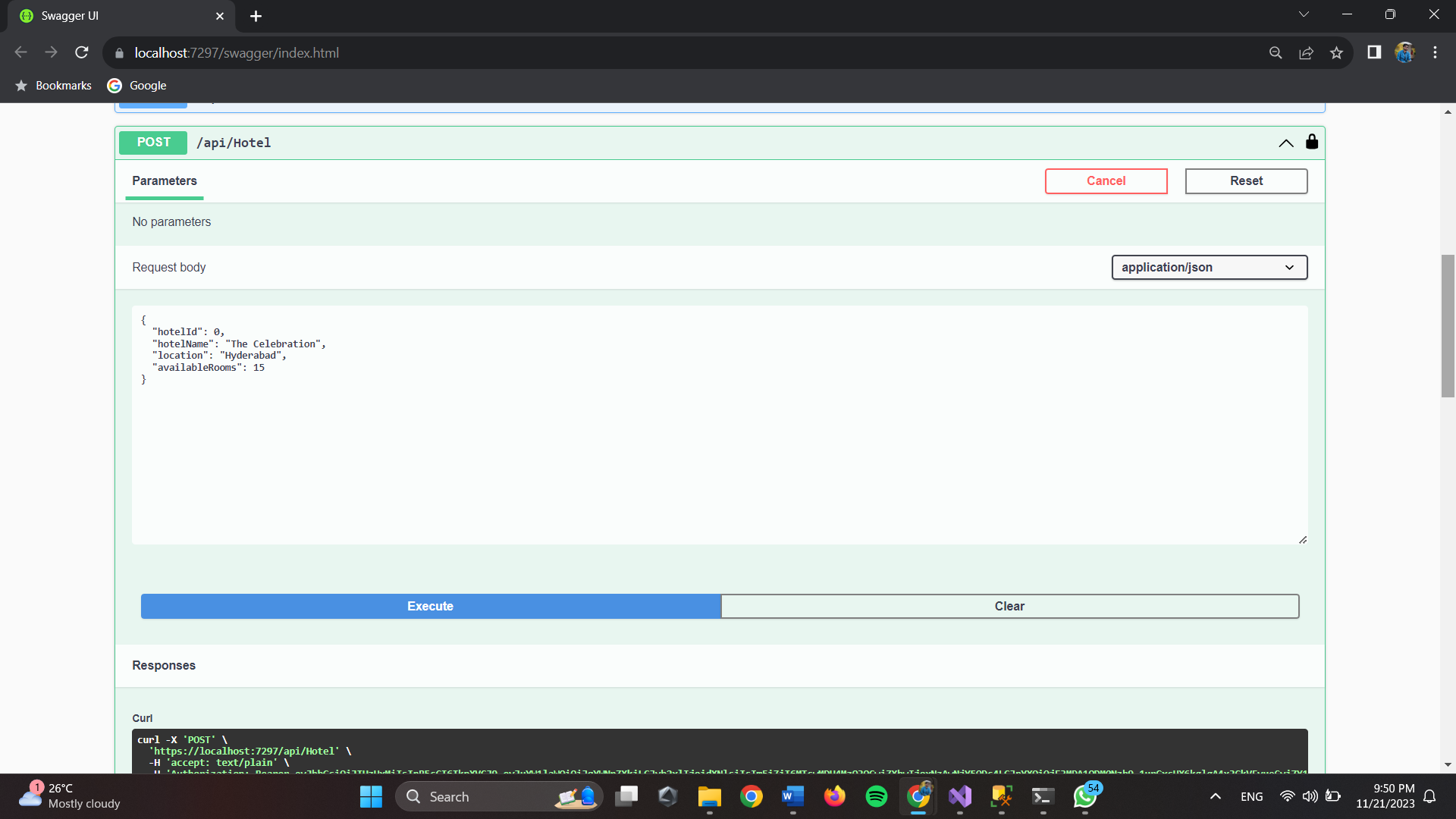
Description automatically generated

A screenshot of a computer

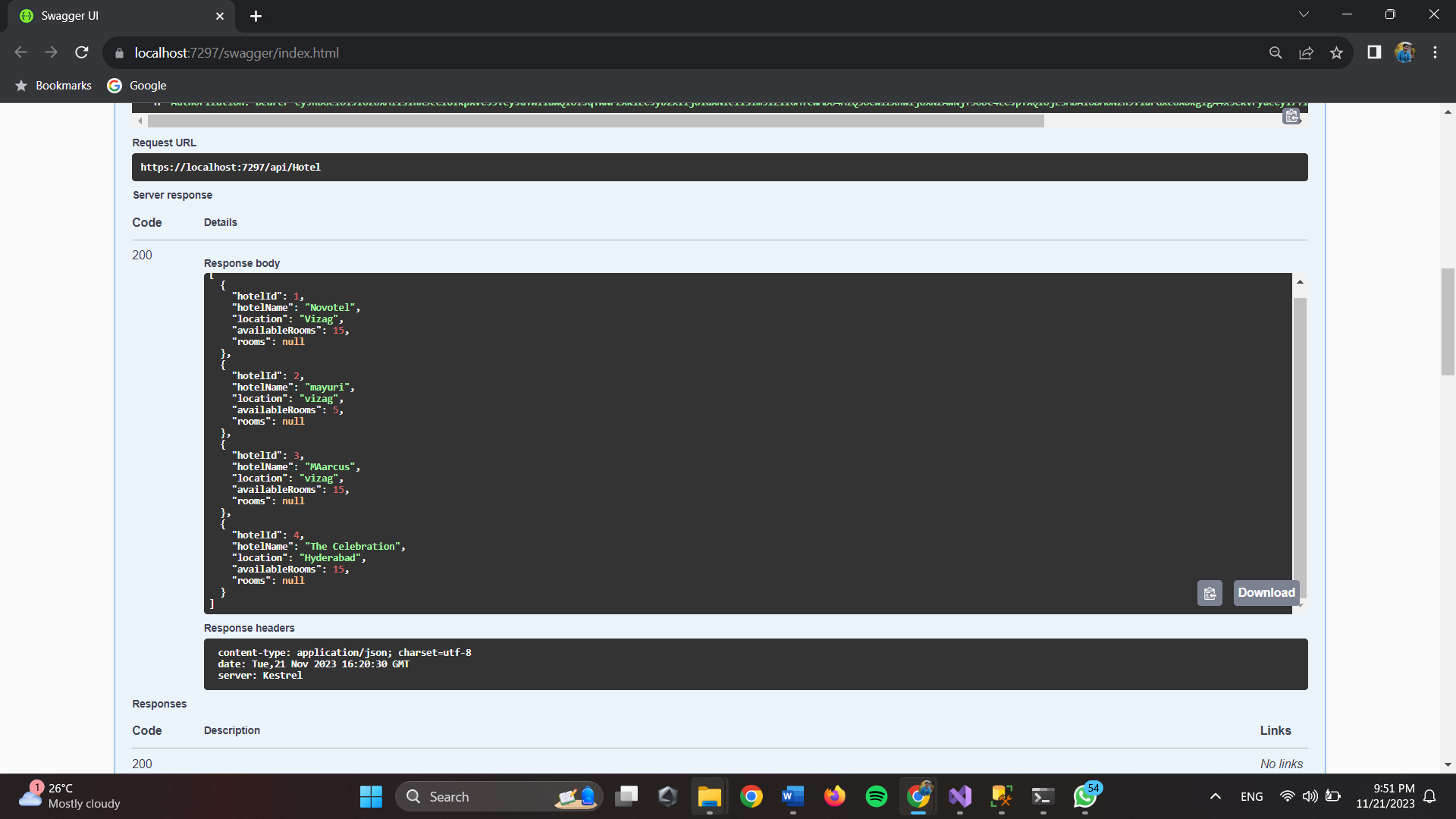
Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

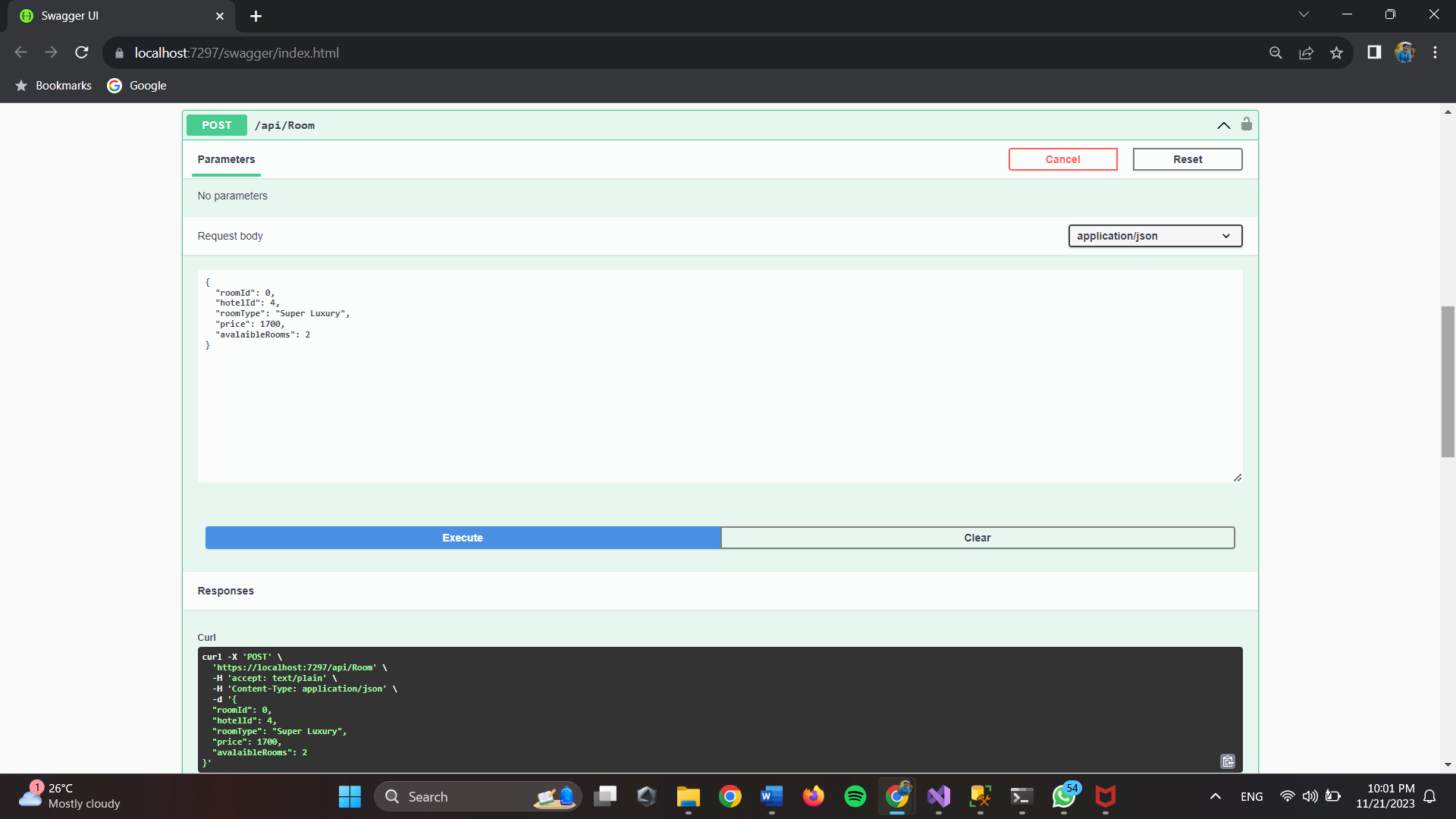
Description automatically generatedA screenshot of a computer

Description automatically generated

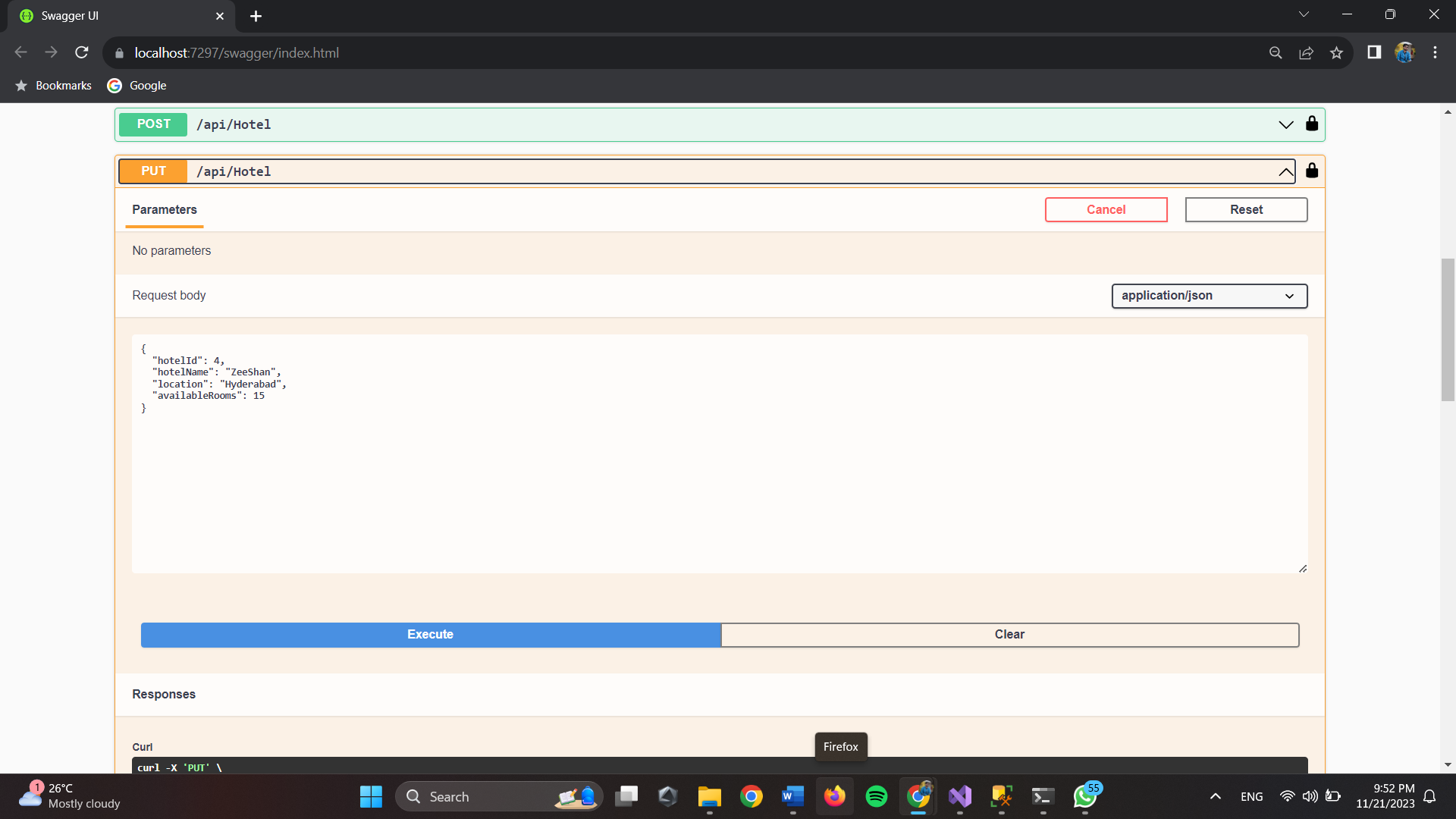
A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

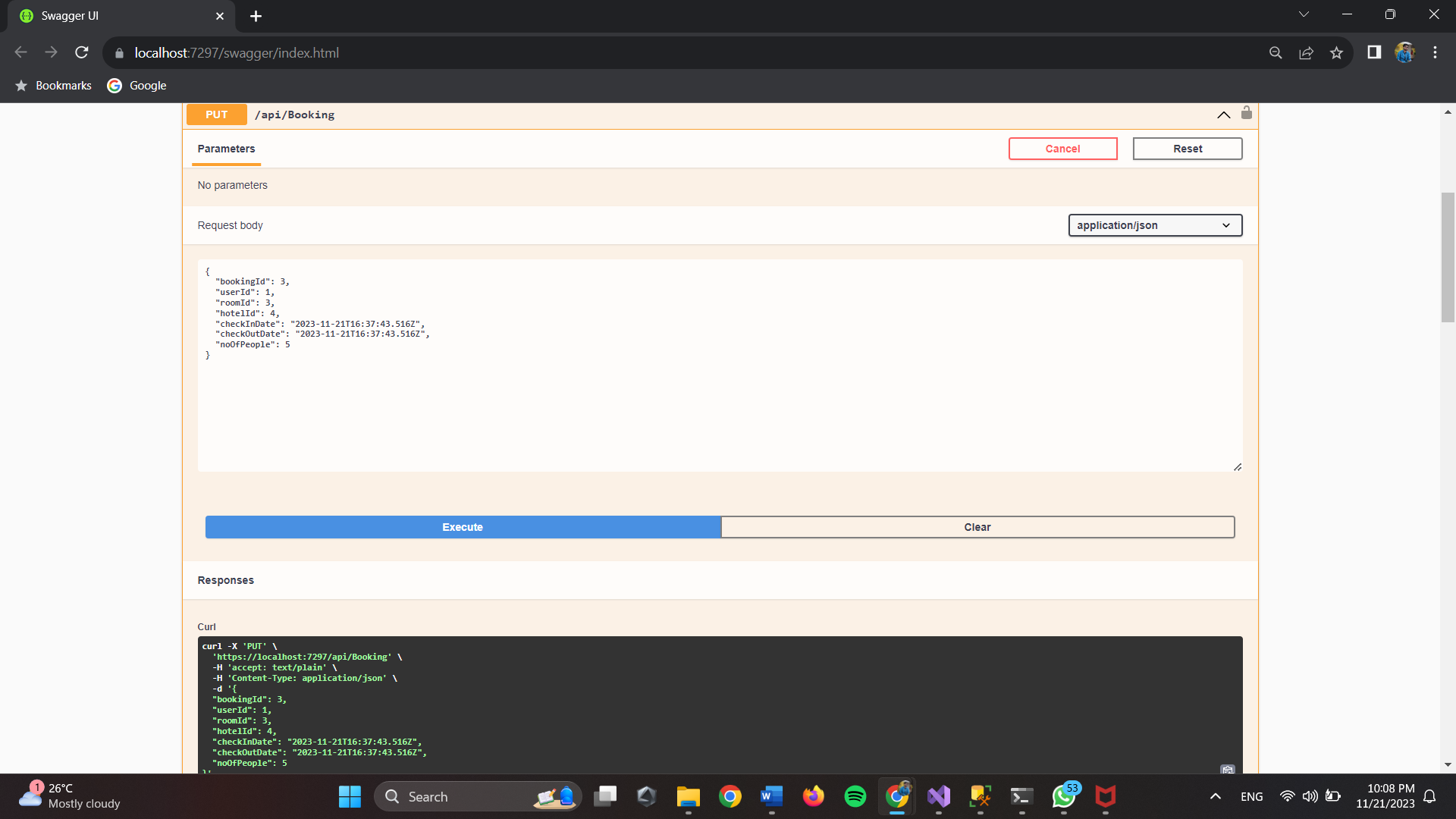
Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

