HOTEL API:

CONTEXT:

using HotelApp.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApp.Context

{

public class HotelContext :DbContext

{

public HotelContext(DbContextOptions options) : base(options)

{

}

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

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

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

}

}

HOTELCONTROLLER:

using HotelApp.Exceptions;

using HotelApp.Interfaces;

using HotelApp.Models.DTOs;

using HotelApp.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace HotelApp.Controllers

{

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

[ApiController]

public class HotelController : ControllerBase

{

private readonly IHotelService \_hotelService;

public HotelController(IHotelService hotelService)

{

\_hotelService = hotelService;

}

[HttpGet]

public ActionResult Get()

{

string errorMessage = string.Empty;

try

{

var result = \_hotelService.GetHotels();

return Ok(result);

}

catch (NoRoomsAvailableException e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[Authorize(Roles = "Admin")]

[HttpPost]

public ActionResult Create(HotelDTO hotelDTO)

{

string errorMessage = string.Empty;

try

{

var result = \_hotelService.Add(hotelDTO);

return Ok(result);

}

catch (Exception e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[Authorize(Roles = "Admin")]

[HttpPost("Remove/{remove}")]

public ActionResult Remove( int id)

{

string errorMessage = string.Empty;

try

{

var result = \_hotelService.DeleteHotel(id);

return Ok(result);

}

catch (Exception e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[HttpGet("Location/{location}")]

public ActionResult GetHotelByLocation(string location)

{

string message = string.Empty;

try

{

var result = \_hotelService.GetHotelsByLocation(location);

return Ok(result);

}

catch (NoHotelsAvailableException ex)

{

message = ex.Message;

}

return BadRequest(message);

}

[Authorize]

[HttpPut("{HotelId}")]

public IActionResult UpdateHotel(int HotelId, [FromBody] Hotel updatedHotel)

{

try

{

if (HotelId != updatedHotel.HotelId)

{

return BadRequest("Mismatched quizId in the request.");

}

// Get the existing hotel from the service

var existingHotel = \_hotelService.GetHotel(HotelId);

// Check if the hotel exists

if (existingHotel == null)

{

return NotFound($"Hotel with ID {HotelId} not found.");

}

// Update the properties of the existing hotel with the values from the updatedHotel

existingHotel.Location = updatedHotel.Location;

existingHotel.Phone = updatedHotel.Phone;

// Update the hotel in the service

\_hotelService.UpdateHotel(existingHotel);

return Ok("Hotel updated successfully.");

}

catch (Exception ex)

{

return StatusCode(500, $"An error occurred: {ex.Message}");

}

}

}

}

Room Controller:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using HotelApp.Interfaces;

using HotelApp.Services;

using HotelApp.Exceptions;

using HotelApp.Models;

using HotelApp.Models.DTOs;

namespace HotelApp.Controllers

{

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

[ApiController]

public class RoomController : ControllerBase

{

private readonly IRoomService \_roomService;

public RoomController(IRoomService roomService)

{

\_roomService = roomService;

}

[HttpGet]

public ActionResult GetCountofAvailableRooms()

{

string errorMessage = string.Empty;

try

{

var result = \_roomService.GetRoomCount();

return Ok($"Available Rooms in the Hotel are {result}");

}

catch (NoRoomsAvailableException e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[HttpPost]

[Authorize(Roles ="Admin")]

public ActionResult Create(Room room)

{

string errorMessage = string.Empty;

try

{

var result = \_roomService.Add(room);

return Ok(result);

}

catch (Exception e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[HttpPost]

[Route("BookRoom")]

public ActionResult BookRoom(int RoomId,string GuestName)

{

string errorMessage = string.Empty;

try

{

var result = \_roomService.Book(RoomId,GuestName);

return Ok(result);

}

catch (Exception e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[HttpPost]

[Route("VacateRoom")]

public ActionResult VacateRoom(int RoomId, string GuestName)

{

string errorMessage = string.Empty;

try

{

var result = \_roomService.Vacate(RoomId, GuestName);

return Ok(result);

}

catch (Exception e)

{

errorMessage = e.Message;

}

return BadRequest(errorMessage);

}

[HttpGet("Fare/{fare}")]

public ActionResult<IList<Room>> RoomsByFare(float fare)

{

try

{

var rooms = \_roomService.RoomsByFare( fare);

return Ok(rooms);

}

catch (Exception e)

{

return BadRequest($"Failed to retrieve rooms. {e.Message}");

}

}

}

}

UserController:

using HotelApp.Interfaces;

using HotelApp.Models.DTOs;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

namespace HotelApp.Controllers

{

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

[ApiController]

public class UserController : ControllerBase

{

private readonly IUserService \_userService;

public UserController(IUserService userService)

{

\_userService = userService;

}

[HttpPost]

public ActionResult Register(UserDTO userDTO)

{

string message = "";

try

{

var user = \_userService.Register(userDTO);

if (user != null)

{

return Ok(user);

}

}

catch (DbUpdateException exp)

{

message = "Duplicate username";

}

catch (Exception)

{

}

return BadRequest(message);

}

[HttpPost]

[Route("Login")]

public ActionResult Login(UserDTO viewModel)

{

string message = "";

try

{

var user = \_userService.Login(viewModel);

if (user != null)

{

return Ok(user);

}

else

{

message = "invalid credentials";

}

}

catch (Exception ex)

{

message = "error occured during login";

}

return BadRequest(message);

}

}

}

EXCEPTIONS:

namespace HotelApp.Exceptions

{

public class NoRoomsAvailableException : Exception

{

string message;

public NoRoomsAvailableException()

{

message = "No Questions are available In This Quiz";

}

public override string Message => message;

}

}

namespace HotelApp.Exceptions

{

public class InValidUpdateActionException : Exception

{

string message;

public InValidUpdateActionException()

{

message = "The action you have specified is not valid";

}

public override string Message => message;

}

}

namespace HotelApp.Exceptions

{

public class NoHotelsAvailableException : Exception

{

string message;

public NoHotelsAvailableException()

{

message = "No hotels are available for display";

}

public override string Message => message;

}

}

namespace HotelApp.Exceptions

{

public class NoSuchRoomsException : Exception

{

string message;

public NoSuchRoomsException()

{

message = "The Doctor with the given id is not present";

}

public override string Message => message;

}

}

namespace HotelApp.Interfaces

{

public interface IRepository<K, T>

where T : class

{

public T Add(T item);

public T Delete(K key);

public T GetById(K key);

public IList<T> GetAll();

public T Update(T item);

}

}

using HotelApp.Models.DTOs;

using HotelApp.Models;

namespace HotelApp.Interfaces

{

public interface IRoomService

{

public int GetRoomCount();

Room Add(Room room);

Room Book(int RoomId,string Guestname);

Room Vacate(int RoomId, string Guestname);

Room Delete(int RoomId);

public List<Room> GetAllRooms();

}

}

using HotelApp.Models.DTOs;

namespace HotelApp.Interfaces

{

public interface ITokenService

{

string GetToken(UserDTO user);

}

}

using HotelApp.Models.DTOs;

namespace HotelApp.Interfaces

{

public interface IUserService

{

UserDTO Login(UserDTO userDTO);

UserDTO Register(UserDTO userDTO);

}

}

DTOs:

using System.ComponentModel.DataAnnotations;

namespace HotelApp.Models.DTOs

{

public class RoomDTO

{

[Required(ErrorMessage = "Username is empty")]

public string Username { get; set; }

[Required(ErrorMessage = "RoomId can to be empty")]

public int RoomId { get; set; }

public float Fare { get; set; }

// public string Type { get; set; }

}

}

using System.ComponentModel.DataAnnotations;

namespace HotelApp.Models.DTOs

{

public class UserDTO

{

[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; }

public string Phone { get; set; }

}

}

namespace HotelApp.Models.DTOs

{

public class HotelDTO

{

public string Location { get; set; }

// public int HotelId { get; set; }

public string Phone { get; set; }

}

}

MODELS:

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace HotelApp.Models

{

public class Hotel

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int HotelId { get; set; }

public string Name { get; set; } = "XYZ";

public string Location { get; set; }

//public int RoomCount { get; set; }

public string Phone { get; set; }

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

}

}

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace HotelApp.Models

{

public class Room

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

//public string Type { get; set; }

public float Fare { get; set; }

public string? Name { get; set; } = string.Empty;

public string Status { get; set; }

public Room() {

int Id = 0;

float Fare = 0.0f;

}

}

}

using System.ComponentModel.DataAnnotations;

namespace HotelApp.Models

{

public class User

{

[Key]

//public string Picture { get; set; }

public string Username { get; set; }

public string Phone { get; set; }

public byte[] Password { get; set; }

public byte[] Key { get; set; }

public string Role { get; set; }

}

}

REPOSITORIES:

using HotelApp.Context;

using HotelApp.Interfaces;

using HotelApp.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApp.Repositories

{

public class HotelRepository : IRepository<int, Hotel>

{

public HotelContext \_hotelContext;

public HotelRepository(HotelContext context)

{

\_hotelContext = context;

}

public Hotel Add(Hotel item)

{

\_hotelContext.Hotels.Add(item);

\_hotelContext.SaveChanges();

return item;

}

public Hotel Delete(int key)

{

var result = GetById(key);

if (result != null)

{

\_hotelContext.Hotels.Remove(result);

\_hotelContext.SaveChanges();

return result;

}

return null;

}

public IList<Hotel> GetAll()

{

if (\_hotelContext.Rooms.Count() != 0)

{

return \_hotelContext.Hotels.ToList();

}

return null;

}

public Hotel GetById(int key)

{

var res = \_hotelContext.Hotels.SingleOrDefault(r => r.HotelId == key);

return res;

}

public Hotel Update(Hotel entity)

{

var hotel = GetById(entity.HotelId);

if (hotel != null)

{

\_hotelContext.Entry<Hotel>(hotel).State = EntityState.Modified;

\_hotelContext.SaveChanges();

return hotel;

}

return null;

}

}

}

using HotelApp.Context;

using HotelApp.Interfaces;

using HotelApp.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApp.Repositories

{

public class RoomRepository :IRepository<int,Room>

{

public HotelContext \_hotelContext;

public RoomRepository(HotelContext context) {

\_hotelContext = context;

}

public Room Add(Room item)

{

\_hotelContext.Rooms.Add(item);

\_hotelContext.SaveChanges();

return item;

}

public Room Delete(int key)

{

var result = GetById(key);

if (result != null)

{

\_hotelContext.Rooms.Remove(result);

\_hotelContext.SaveChanges();

return result;

}

return null;

}

public IList<Room> GetAll()

{

if (\_hotelContext.Rooms.Count()!=0)

{

return \_hotelContext.Rooms.ToList();

}

return null;

}

public Room GetById(int key)

{

var res=\_hotelContext.Rooms.SingleOrDefault(r => r.Id == key);

return res;

}

public Room Update(Room entity)

{

var room = GetById(entity.Id);

if (room != null)

{

\_hotelContext.Entry<Room>(room).State = EntityState.Modified;

\_hotelContext.SaveChanges();

return room;

}

return null;

}

}

}

using HotelApp.Context;

using HotelApp.Interfaces;

using HotelApp.Models;

using Microsoft.EntityFrameworkCore;

namespace HotelApp.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;

}

}

}

SERVICES:

using HotelApp.Interfaces;

using HotelApp.Models;

using HotelApp.Exceptions;

using HotelApp.Repositories;

using HotelApp.Models.DTOs;

namespace HotelApp.Services

{

public class HotelService : IHotelService

{

private readonly IRepository<int,Hotel> hotelRepository;

public HotelService(IRepository<int,Hotel> \_hotelRepository)

{

hotelRepository = \_hotelRepository;

}

public HotelDTO Add(HotelDTO hotelDTO)

{

Hotel hotel = new Hotel()

{

Location= hotelDTO.Location,

Phone = hotelDTO.Phone

};

var res= hotelRepository.Add(hotel);

if (res != null)

{

return hotelDTO;

}

return null;

throw new NotImplementedException();

}

public Hotel DeleteHotel(int id)

{

var result = hotelRepository.GetById(id);

if (result != null)

{

hotelRepository.Delete(id);

return result;

}

return null;

throw new NotImplementedException();

}

public Hotel GetHotel(int id)

{

var result = hotelRepository.GetById(id);

return result == null ? throw new NoRoomsAvailableException() : result;

}

public List<Hotel> GetHotels()

{

var hotels = hotelRepository.GetAll();

if (hotels.Count != 0)

return hotels.ToList();

throw new Exception();

}

public void UpdateHotel(Hotel updatedHotel)

{

if (updatedHotel == null)

{

throw new ArgumentNullException(nameof(updatedHotel), "Updated Hotel data is null.");

}

var existingHotel = hotelRepository.GetById(updatedHotel.HotelId);

if (existingHotel == null)

{

throw new InvalidOperationException($"Quiz with ID {updatedHotel.HotelId} not found.");

}

existingHotel.Phone = updatedHotel.Phone;

existingHotel.Location = updatedHotel.Location;

hotelRepository.Update(existingHotel);

}

public List<Hotel> GetHotelsByLocation(string location)

{

var hotels = hotelRepository.GetAll().Where(c => c.Location.Contains(location, StringComparison.OrdinalIgnoreCase)).ToList();

if (hotels != null)

{

return hotels;

}

throw new NoHotelsAvailableException();

}

}

}

using HotelApp.Context;

using HotelApp.Interfaces;

using HotelApp.Models;

using HotelApp.Models.DTOs;

using HotelApp.Exceptions;

using HotelApp.Repositories;

using Microsoft.EntityFrameworkCore;

namespace HotelApp.Services

{

public class RoomService : IRoomService

{

private readonly IRepository<int, Room> roomRepository;

public HotelContext \_hotelContext;

public RoomService(IRepository<int, Room> \_roomRepository,HotelContext hotelContext)

{

roomRepository = \_roomRepository;

\_hotelContext = hotelContext;

}

public Room Add(Room room)

{

var res = roomRepository.Add(room);

if (res != null)

{

return res;

}

return null;

throw new NotImplementedException();

}

public Room Book(int RoomId, string Guestname)

{

var res = roomRepository.GetById(RoomId);

if (res != null)

{

if (res.Status.Equals("Available", StringComparison.OrdinalIgnoreCase))

{

// Update room properties

res.Status = "Booked";

res.Name = Guestname;

// You can perform additional actions or validations if needed

// Return true indicating successful booking

return res;

}

else

throw new NoRoomsAvailableException();

}

throw new NotImplementedException();

}

public Room Delete(int RoomId)

{

var res = roomRepository.GetById(RoomId);

if (res != null)

{

roomRepository.Delete(RoomId);

return res;

}

throw new NotImplementedException();

}

public int GetRoomCount()

{

if (\_hotelContext.Rooms.Count() != 0)

{

return \_hotelContext.Rooms.AsEnumerable().Count(room => room.Status.Equals("Available", StringComparison.OrdinalIgnoreCase));

}

return 0;

}

public Room Vacate(int RoomId, string Guestname)

{

var res = roomRepository.GetById(RoomId);

try

{

if (res != null)

{

if (res.Status.Equals("Booked", StringComparison.OrdinalIgnoreCase))

{

// Update room properties

res.Status = "Available";

res.Name =null;

// Return true indicating successful booking

return res;

}

}

}

catch (Exception ex)

{

throw new NoRoomsAvailableException();

}

throw new NotImplementedException();

}

public List<Room> GetAllRooms()

{

var res = roomRepository.GetAll();

if (res != null)

{

return res.ToList();

}

return null;

}

public List<Room> RoomsByFare(float fare)

{

var rooms = roomRepository.GetAll();

if (rooms != null)

{

return rooms

.Where(room => room.Status == "Available" && room.Fare <= fare)

.ToList();

}

throw new NoRoomsAvailableException();

}

}

}

using HotelApp.Interfaces;

using HotelApp.Models.DTOs;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace HotelApp.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);

}

}

}

using System.Security.Cryptography;

using System.Text;

using HotelApp.Interfaces;

using HotelApp.Models;

using HotelApp.Models.DTOs;

namespace HotelApp.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)),

Phone=userDTO.Phone,

Key = hmac.Key,

Role = userDTO.Role

};

var result = \_repository.Add(user);

if (result != null)

{

userDTO.Password = "";

return userDTO;

}

return null;

}

}

}

https://json.schemastore.org/appsettings.json:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"ConnectionStrings": {

"conn": "Data source=LAPTOP-HPFDE7IT\\SQLEXPRESS;Integrated Security=true;Initial catalog=dbHotelApp01"

},

"SecretKey": "this is my Secret key",

"AllowedHosts": "\*"

}

Program.cs:

using HotelApp.Interfaces;

using HotelApp.Models;

using HotelApp.Repositories;

using HotelApp.Interfaces;

using HotelApp.Models;

using HotelApp.Repositories;

using HotelApp.Services;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Cors.Infrastructure;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

using HotelApp.Context;

namespace HotelApp

{

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();

builder.Services.AddRazorPages();

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<IRepository<string, User>, UserRepository>();

builder.Services.AddScoped<IRepository<int, Room>, RoomRepository>();

builder.Services.AddScoped<IRepository<int, Hotel>, HotelRepository>();

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

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

builder.Services.AddScoped<IRoomService, RoomService>();

builder.Services.AddScoped<IHotelService, HotelService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseStaticFiles();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

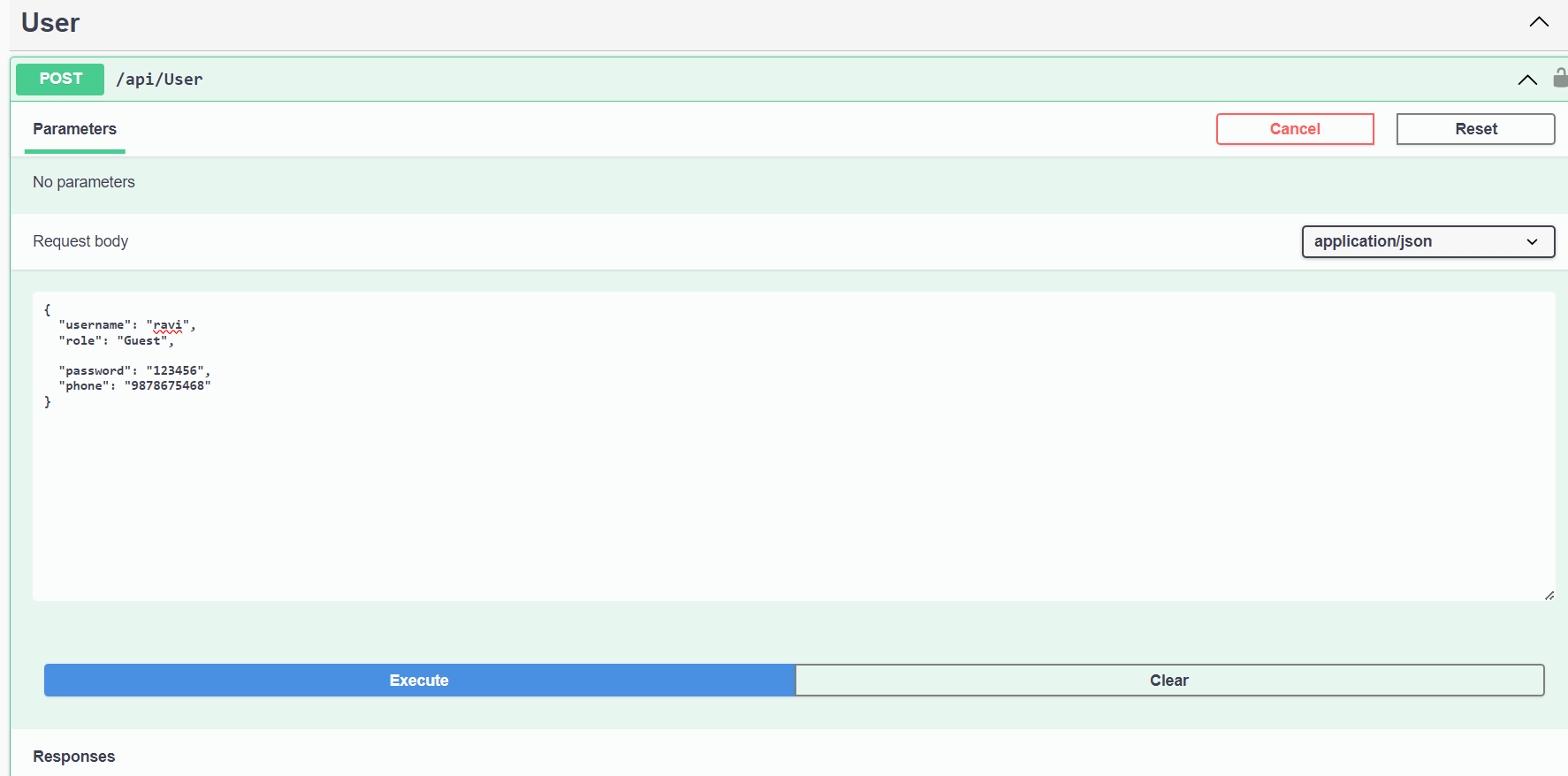
}

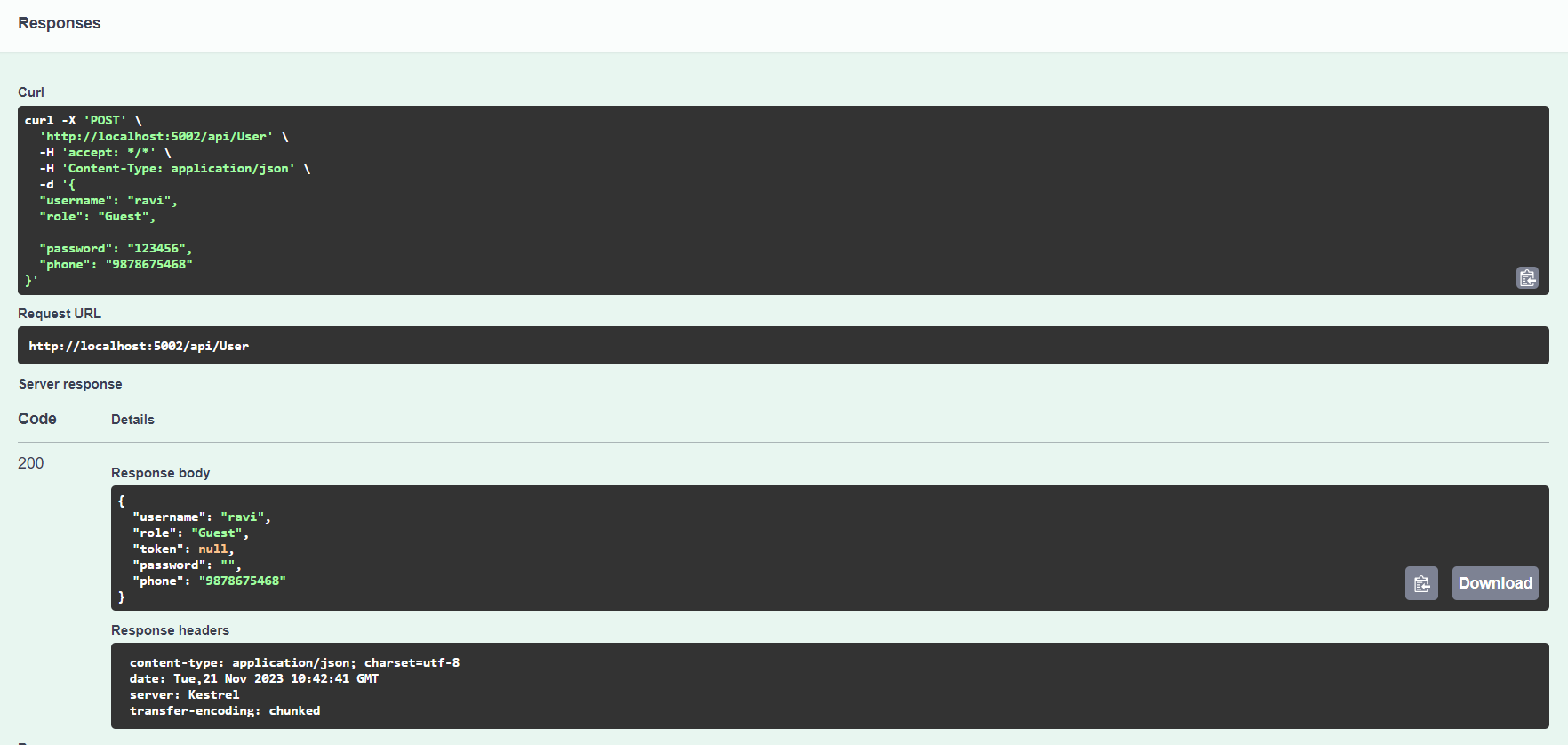
}

}

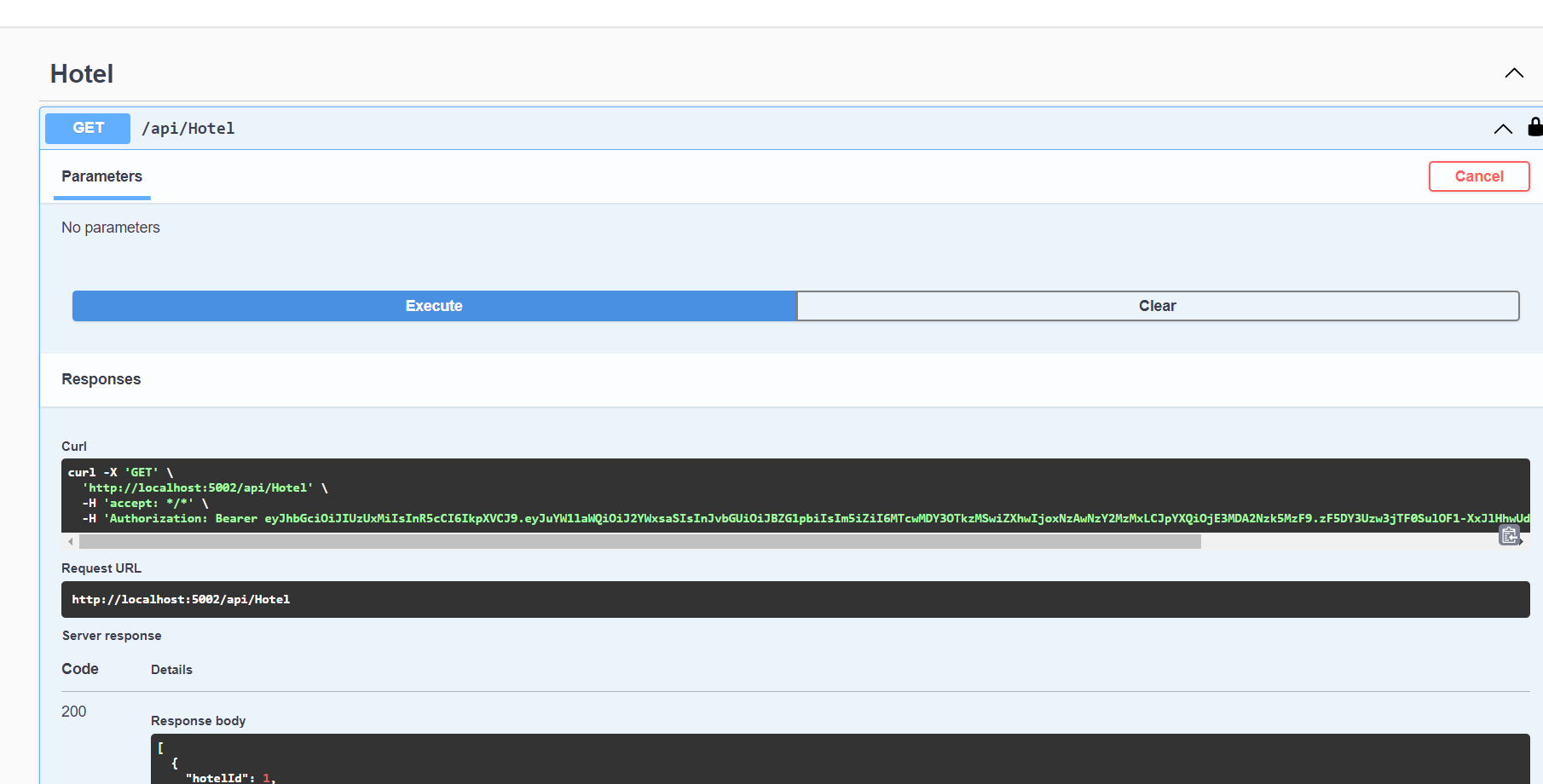
OUTPUT:

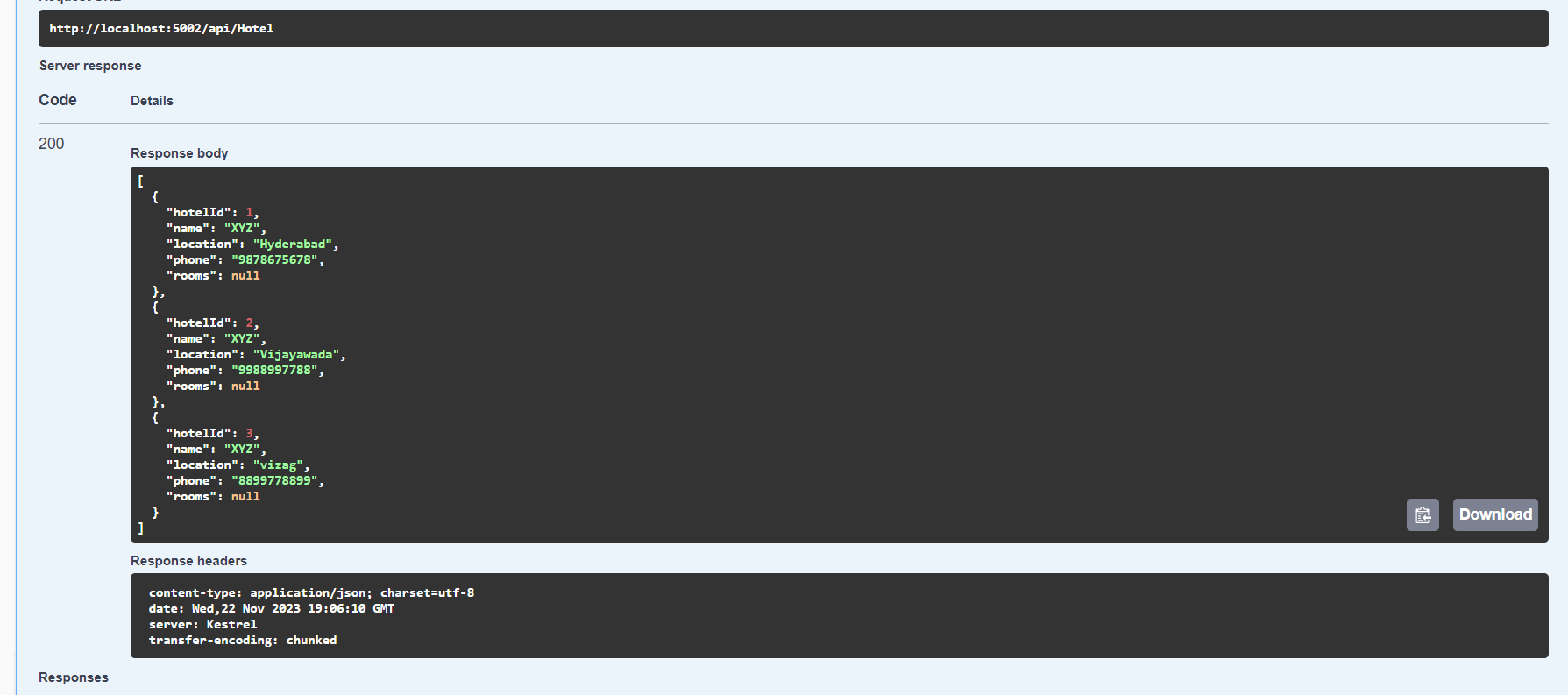
Register and login:



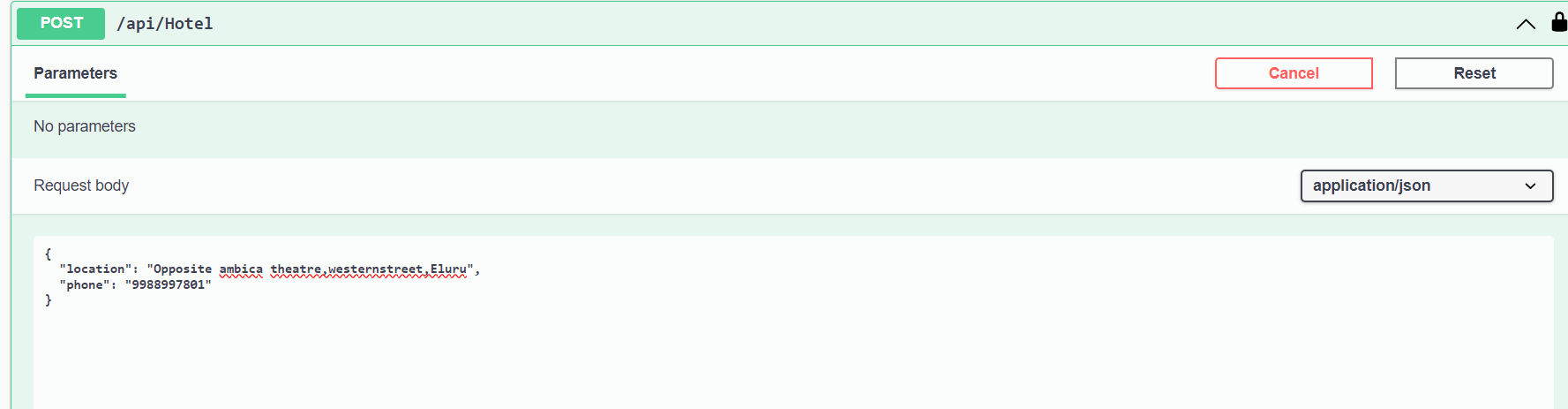


Get hotels:



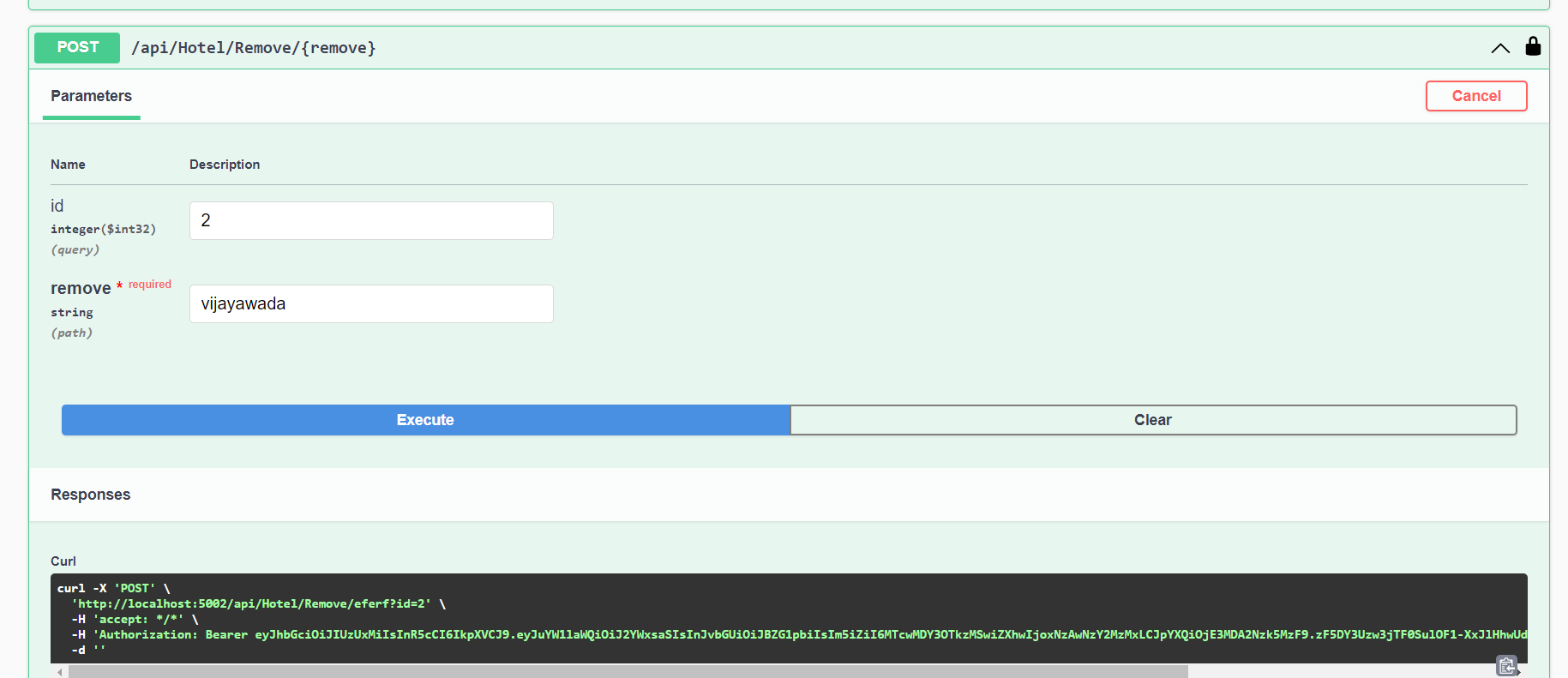


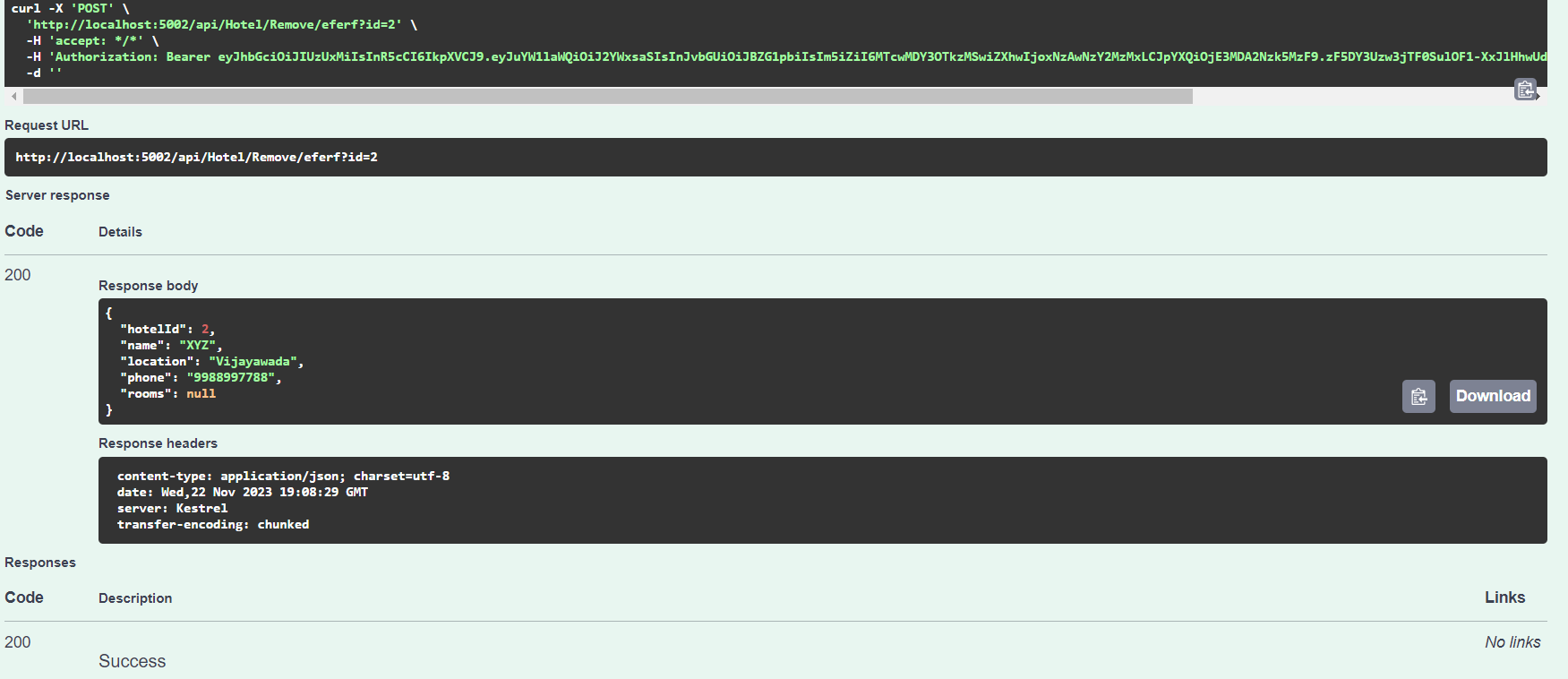
Add hotel:



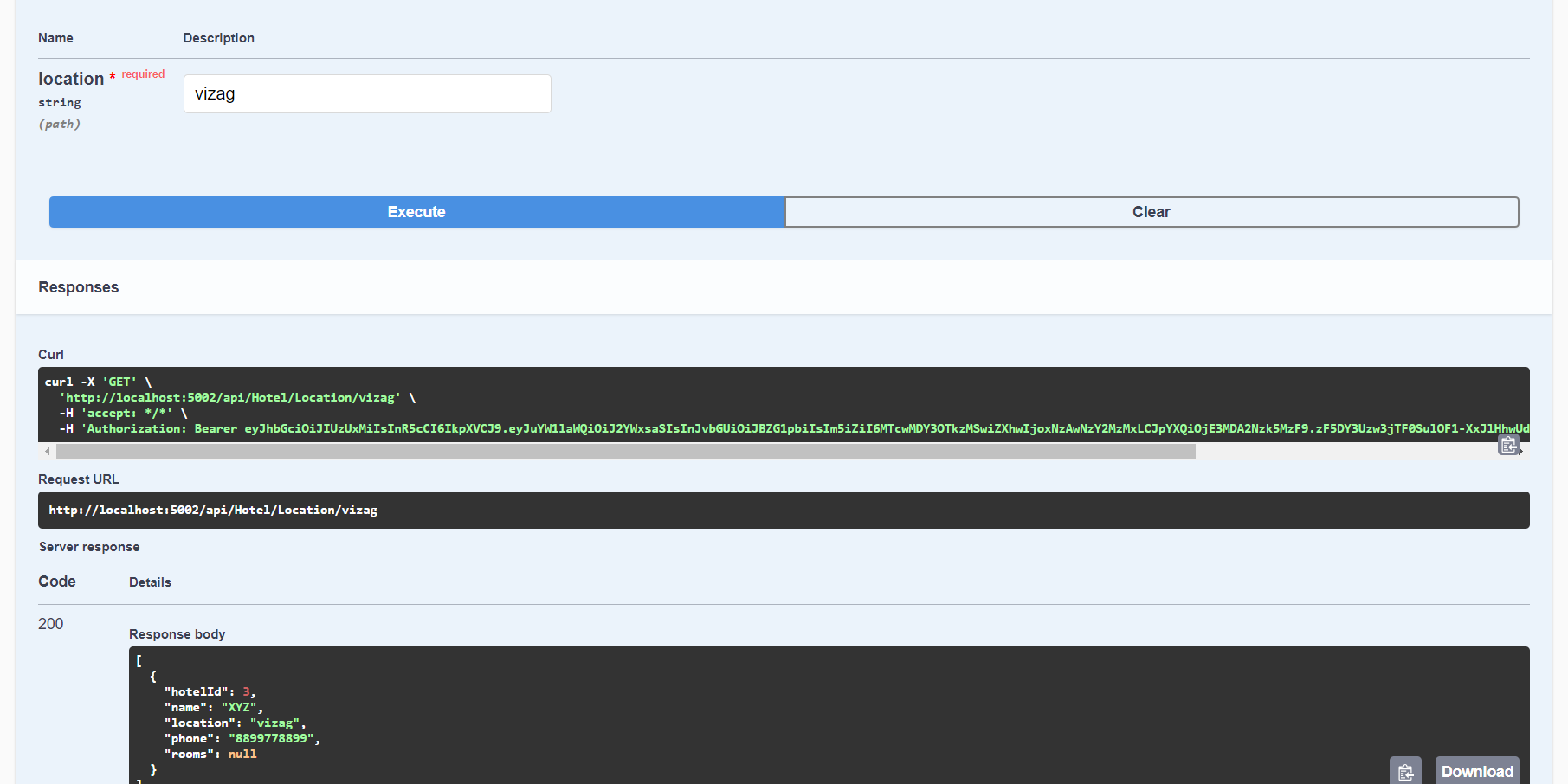


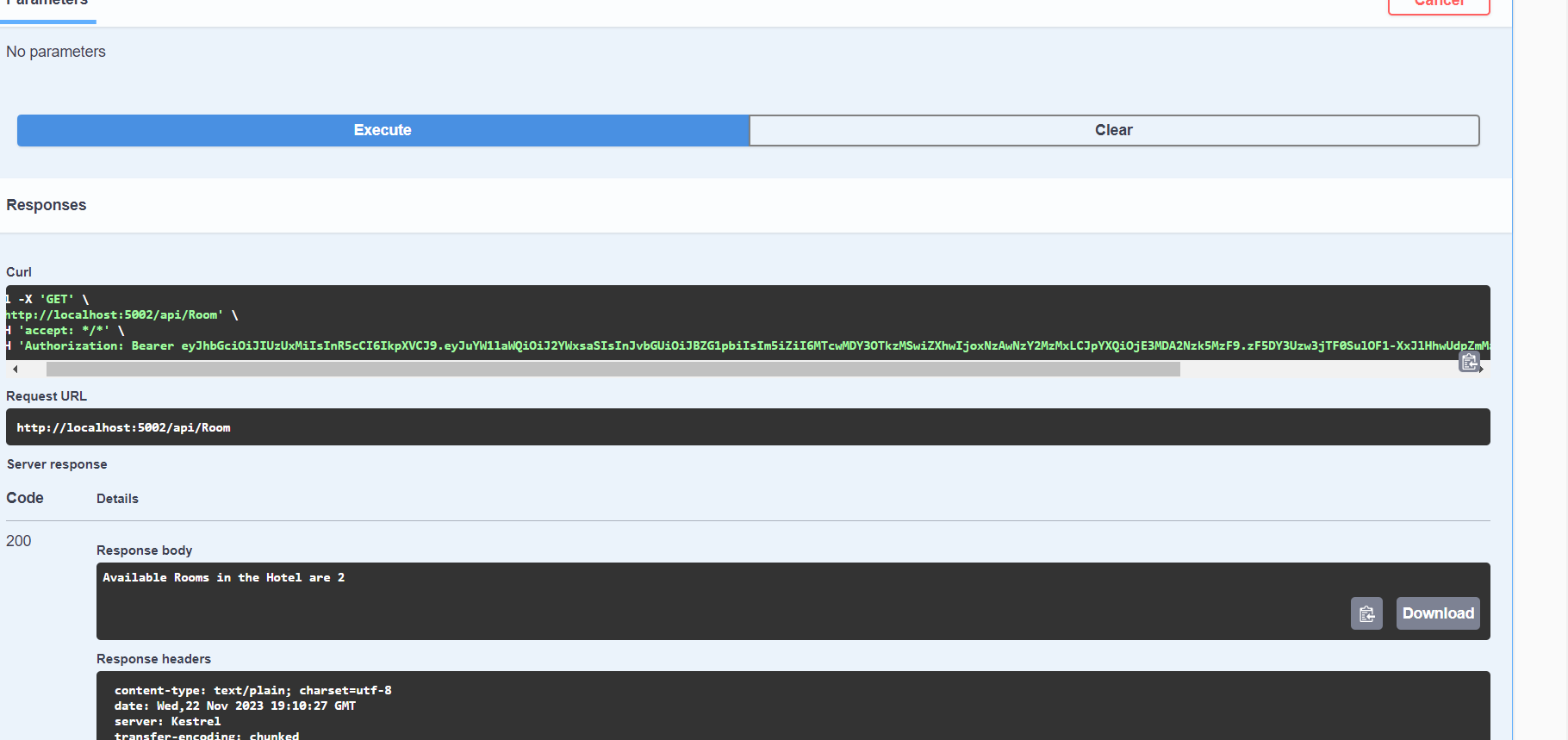
Remove hotel:



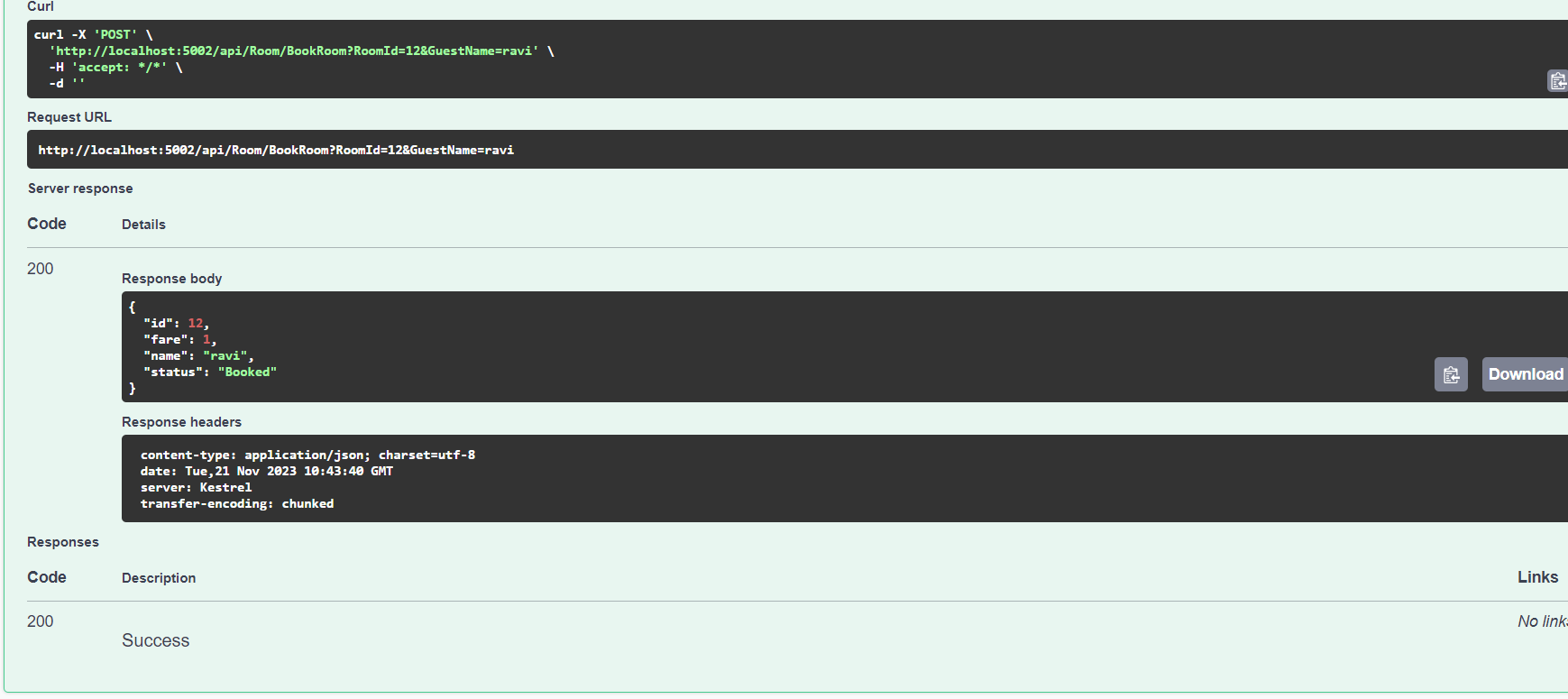
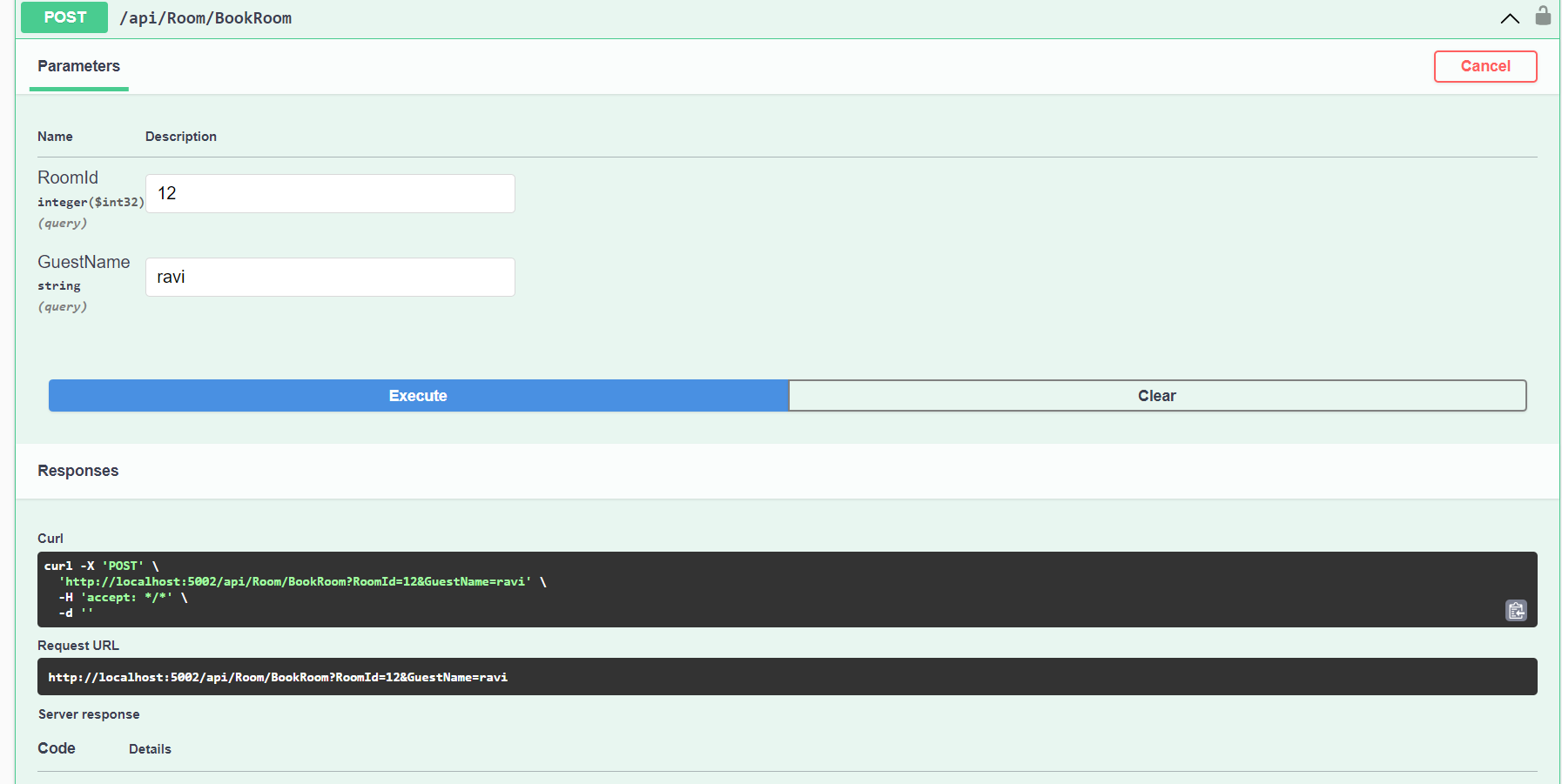


Get hotel by location:

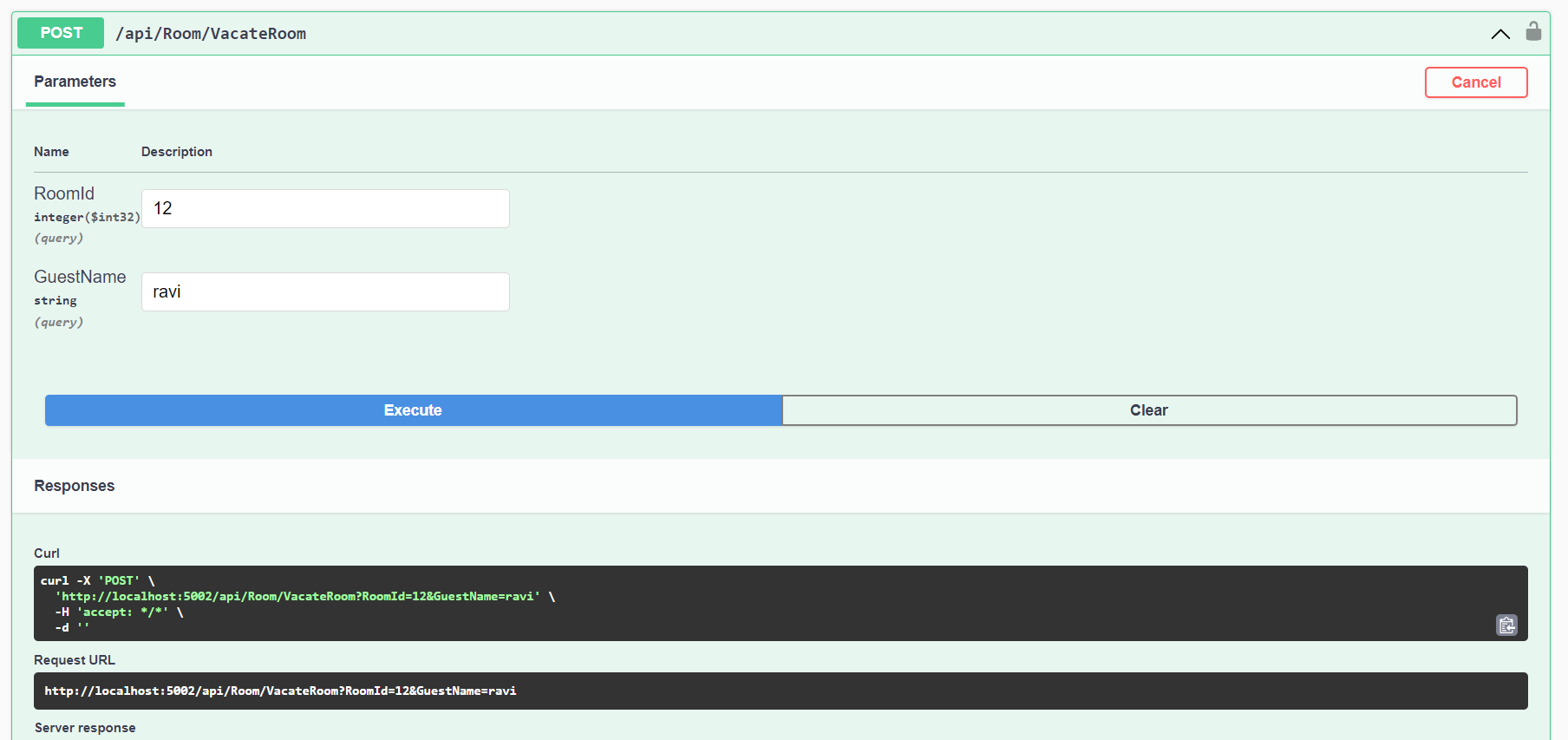


Total Avaialble rooms:

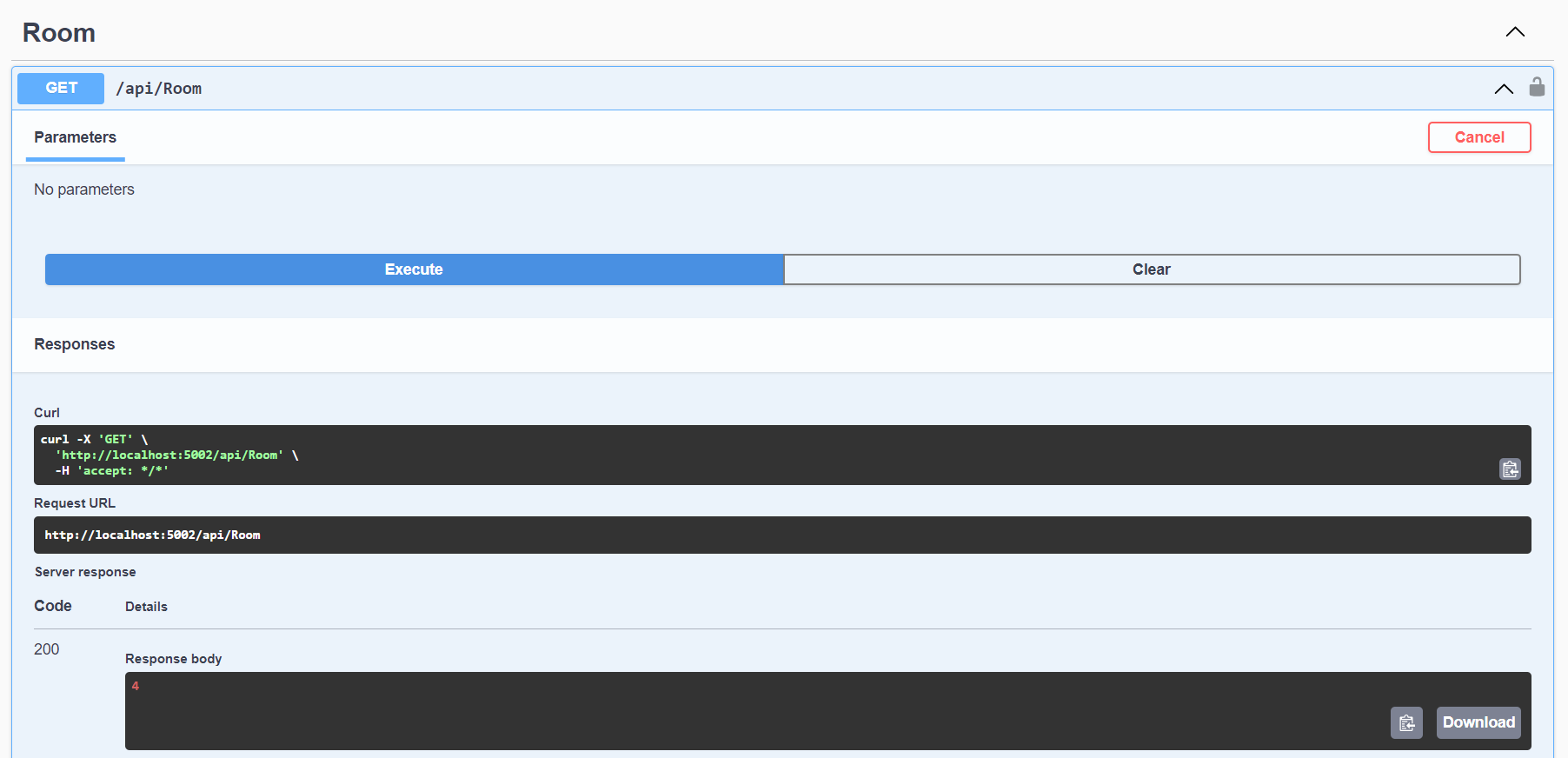
Booking a room:



Vacating a room:



Count Of Available Rooms:



Getting rooms by price:(rooms with fare equal to and below given price):

