**Лабораторна робота №4**

**Мета**:Ознайомитися з процесом створення клієнт-серверної архітектури. Навчитися створювати RESTful API для взаємодії між клієнтом і сервером. Розвинути вміння проєктувати та реалізовувати ендпойнти для типових CRUD-операцій.

**Завдання 1**

Ознайомитися з кроками створення **ASP.NET Core Minimal API** | [Check out the](https://docs-mai.vercel.app/docs/developmement/minimal_api).

Створити **ASP.NET Core Minimal API**:

1. Створити **3–4 групи маршрутів** відповідно до Додатку А(наприклад: /todos, /users, /categories).
2. Використати **in-memory “сховище”** (простий список List в коді) для зберігання даних.
3. Реалізувати **CRUD** для кожної групи маршрутів:
   * **Create** (POST) — створення нового об’єкта.
   * **Read** (GET) — отримання списку об’єктів або одного об’єкта за ID.
   * **Update** (PUT/PATCH) — оновлення існуючого об’єкта.
   * **Delete** (DELETE) — видалення об’єкта за ID.
4. Виконати **валідацію даних** під час створення нових об’єктів (наприклад, перевірка порожніх полів, мінімальної довжини рядка, формату email тощо).
5. Повертає стандартні HTTP-відповіді:
   * 200 OK — успішний запит,
   * 201 Created — створення нового ресурсу,
   * 400 Bad Request — помилка валідації,
   * 404 Not Found — об’єкт не знайдено.
6. Для кожної групи створити **окремий набір ендпойнтів** у коді.
7. **Винести ендпойнти в окремі файли** (наприклад: Endpoints/TodoEndpoints.cs, Endpoints/UserEndpoints.cs) і підключити їх у Program.cs.
8. **Організувати структуру проєкту** за папками:

/Models  
/Endpoints  
/Data (за потреби)  
Program.cs

1. **Додати Swagger (OpenAPI)** для зручного тестування API через браузер:
2. Перевірити роботу API через **Swagger** або **Postman**.

***Варіант*:**

**24. Сервіс для планування та проведення квест-кімнат**

Створимо папки Models, Endpoints,Data. В папці Models створюємо 4 класа, а саме User.cs, Review.cs, Reservation.cs, QuestRoom.cs. В папці Endpoints створюємо також 4 класа, а саме UserEndpoints.cs, RoomEndpoints.cs, ReservationEndpoints.cs, ReviewEndpoints.cs. В папці Data створюємо QuestData.cs

Ось як виглядає наша ієрархія.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.

Тепер перейдемо до написання коду.

**Models**

**User.cs**

using System.ComponentModel.DataAnnotations;

public class User

{

public int Id { get; set; }

[Required, MinLength(2)]

public string FullName { get; set; } = "";

[Required, EmailAddress]

public string Email { get; set; } = "";

}

**QuestRoom.cs**

using System.ComponentModel.DataAnnotations;

public class QuestRoom

{

public int Id { get; set; }

[Required, MinLength(3)]

public string Title { get; set; } = "";

[Required, MinLength(10)]

public string Description { get; set; } = "";

[Range(1, 20)]

public int MaxPlayers { get; set; }

[Range(0, double.MaxValue)]

public decimal PricePerHour { get; set; }

public List<string> Tags { get; set; } = new();

}

**Reservation.cs**

using System.ComponentModel.DataAnnotations;

public class Reservation

{

public int Id { get; set; }

[Required]

public int QuestRoomId { get; set; }

[Required]

public int UserId { get; set; }

[Required]

public DateTime StartAt { get; set; }

[Range(1, 8)]

public int Hours { get; set; } = 1;

}

**Review.cs**

using System.ComponentModel.DataAnnotations;

public class Review

{

public int Id { get; set; }

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

[Required, MinLength(2)] public string AuthorName { get; set; } = "";

[Range(1, 5)] public int Rating { get; set; }

[MinLength(5)] public string? Comment { get; set; }

}

**Endpoints**

**UserEndpoints.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using System.ComponentModel.DataAnnotations;

public static class UserEndpoints

{

public static void MapUserEndpoints(this WebApplication app)

{

var users = QuestData.Users;

app.MapGet("/users", () => Results.Ok(users));

app.MapGet("/users/{id:int}", (int id) =>

{

var u = users.FirstOrDefault(x => x.Id == id);

return u is null ? Results.NotFound() : Results.Ok(u);

});

app.MapPost("/users", async (HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<User>();

if (model is null) return Results.BadRequest("Invalid payload");

var errors = Validate(model);

if (errors.Any()) return Results.BadRequest(errors);

model.Id = (users.Any() ? users.Max(r => r.Id) : 0) + 1;

users.Add(model);

return Results.Created($"/users/{model.Id}", model);

});

app.MapPut("/users/{id:int}", async (int id, HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<User>();

if (model is null) return Results.BadRequest("Invalid payload");

var existing = users.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

var errors = Validate(model);

if (errors.Any()) return Results.BadRequest(errors);

existing.FullName = model.FullName;

existing.Email = model.Email;

return Results.Ok(existing);

});

app.MapDelete("/users/{id:int}", (int id) =>

{

var existing = users.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

users.Remove(existing);

return Results.NoContent();

});

static List<string> Validate(User m)

{

var ctx = new ValidationContext(m);

var results = new List<ValidationResult>();

if (!Validator.TryValidateObject(m, ctx, results, true))

return results.Select(r => r.ErrorMessage ?? "Validation error").ToList();

return new();

}

}

}

**RoomEndpoints.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using System.ComponentModel.DataAnnotations;

public static class RoomEndpoints

{

public static void MapRoomEndpoints(this WebApplication app)

{

var rooms = QuestData.Rooms;

app.MapGet("/rooms", () => Results.Ok(rooms));

app.MapGet("/rooms/{id:int}", (int id) =>

{

var r = rooms.FirstOrDefault(x => x.Id == id);

return r is null ? Results.NotFound() : Results.Ok(r);

});

app.MapPost("/rooms", async (HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<QuestRoom>();

if (model is null) return Results.BadRequest("Invalid payload");

var errors = Validate(model);

if (errors.Any()) return Results.BadRequest(errors);

model.Id = (rooms.Any() ? rooms.Max(r => r.Id) : 0) + 1;

rooms.Add(model);

return Results.Created($"/rooms/{model.Id}", model);

});

app.MapPut("/rooms/{id:int}", async (int id, HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<QuestRoom>();

if (model is null) return Results.BadRequest("Invalid payload");

var existing = rooms.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

var errors = Validate(model);

if (errors.Any()) return Results.BadRequest(errors);

existing.Title = model.Title;

existing.Description = model.Description;

existing.MaxPlayers = model.MaxPlayers;

existing.PricePerHour = model.PricePerHour;

existing.Tags = model.Tags;

return Results.Ok(existing);

});

app.MapDelete("/rooms/{id:int}", (int id) =>

{

var existing = rooms.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

rooms.Remove(existing);

return Results.NoContent();

});

static List<string> Validate(QuestRoom m)

{

var ctx = new ValidationContext(m);

var results = new List<ValidationResult>();

if (!Validator.TryValidateObject(m, ctx, results, true))

return results.Select(r => r.ErrorMessage ?? "Validation error").ToList();

return new();

}

}

}

**ReservationEndpoints.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using System.ComponentModel.DataAnnotations;

public static class ReservationEndpoints

{

public static void MapReservationEndpoints(this WebApplication app)

{

var reservations = QuestData.Reservations;

var rooms = QuestData.Rooms;

var users = QuestData.Users;

app.MapGet("/reservations", () => Results.Ok(reservations));

app.MapGet("/reservations/{id:int}", (int id) =>

{

var r = reservations.FirstOrDefault(x => x.Id == id);

return r is null ? Results.NotFound() : Results.Ok(r);

});

app.MapPost("/reservations", async (HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<Reservation>();

if (model is null) return Results.BadRequest("Invalid payload");

var errors = Validate(model);

if (errors.Any()) return Results.BadRequest(errors);

if (!rooms.Any(r => r.Id == model.QuestRoomId))

errors.Add("QuestRoomId not found.");

if (!users.Any(u => u.Id == model.UserId))

errors.Add("UserId not found.");

var nowUtc = DateTime.UtcNow;

var startUtc = DateTime.SpecifyKind(model.StartAt, DateTimeKind.Utc);

if (startUtc < nowUtc)

errors.Add("StartAt must be in the future.");

var newEnd = startUtc.AddHours(model.Hours);

bool overlaps = reservations.Any(r =>

r.QuestRoomId == model.QuestRoomId &&

DateTime.SpecifyKind(r.StartAt, DateTimeKind.Utc) < newEnd &&

startUtc < DateTime.SpecifyKind(r.StartAt, DateTimeKind.Utc).AddHours(r.Hours));

if (overlaps) errors.Add("Time slot overlaps with an existing reservation for this room.");

if (errors.Any()) return Results.BadRequest(errors);

model.Id = (reservations.Any() ? reservations.Max(r => r.Id) : 0) + 1;

reservations.Add(model);

return Results.Created($"/reservations/{model.Id}", model);

});

app.MapDelete("/reservations/{id:int}", (int id) =>

{

var existing = reservations.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

reservations.Remove(existing);

return Results.NoContent();

});

static List<string> Validate(Reservation m)

{

var ctx = new ValidationContext(m);

var results = new List<ValidationResult>();

if (!Validator.TryValidateObject(m, ctx, results, true))

return results.Select(r => r.ErrorMessage ?? "Validation error").ToList();

return new();

}

}

}

**ReviewEndpoints.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using System.ComponentModel.DataAnnotations;

public static class ReviewEndpoints

{

public static void MapReviewEndpoints(this WebApplication app)

{

var reviews = QuestData.Reviews;

var rooms = QuestData.Rooms;

app.MapGet("/reviews", () => Results.Ok(reviews));

app.MapGet("/reviews/{id:int}", (int id) =>

{

var r = reviews.FirstOrDefault(x => x.Id == id);

return r is null ? Results.NotFound() : Results.Ok(r);

});

app.MapGet("/rooms/{roomId:int}/reviews", (int roomId) =>

{

if (!rooms.Any(r => r.Id == roomId)) return Results.NotFound($"Room {roomId} not found");

var list = reviews.Where(x => x.QuestRoomId == roomId).ToList();

return Results.Ok(list);

});

app.MapPost("/reviews", async (HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<Review>();

if (model is null) return Results.BadRequest("Invalid payload");

var errors = Validate(model);

if (!rooms.Any(r => r.Id == model.QuestRoomId))

errors.Add("QuestRoomId not found.");

if (errors.Any()) return Results.BadRequest(errors);

model.Id = (reviews.Any() ? reviews.Max(r => r.Id) : 0) + 1;

reviews.Add(model);

return Results.Created($"/reviews/{model.Id}", model);

});

app.MapPut("/reviews/{id:int}", async (int id, HttpContext http) =>

{

var model = await http.Request.ReadFromJsonAsync<Review>();

if (model is null) return Results.BadRequest("Invalid payload");

var existing = reviews.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

var errors = Validate(model);

if (!rooms.Any(r => r.Id == model.QuestRoomId))

errors.Add("QuestRoomId not found.");

if (errors.Any()) return Results.BadRequest(errors);

existing.QuestRoomId = model.QuestRoomId;

existing.AuthorName = model.AuthorName;

existing.Rating = model.Rating;

existing.Comment = model.Comment;

return Results.Ok(existing);

});

app.MapDelete("/reviews/{id:int}", (int id) =>

{

var existing = reviews.FirstOrDefault(r => r.Id == id);

if (existing is null) return Results.NotFound();

reviews.Remove(existing);

return Results.NoContent();

});

static List<string> Validate(Review m)

{

var ctx = new ValidationContext(m);

var results = new List<ValidationResult>();

if (!Validator.TryValidateObject(m, ctx, results, true))

return results.Select(r => r.ErrorMessage ?? "Validation error").ToList();

return new();

}

}

}

**Data**

**QuestData.cs**

public static class QuestData

{

public static List<QuestRoom> Rooms { get; } = new()

{

new QuestRoom { Id = 1, Title = "Lost Temple", Description = "A mysterious temple full of puzzles.", MaxPlayers = 6, PricePerHour = 25 },

new QuestRoom { Id = 2, Title = "Space Station", Description = "Save the station before it's too late.", MaxPlayers = 5, PricePerHour = 30 }

};

public static List<User> Users { get; } = new()

{

new User { Id = 1, FullName = "Ivan Petrov", Email = "ivan@example.com" }

};

public static List<Reservation> Reservations { get; } = new();

public static List<Review> Reviews { get; } = new();

}

**Результат роботи:**

**Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

Програма показує те, що нам доступна вся інформація по посиланню.

**Запити**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, комп’ютер

Вміст на основі ШІ може бути неправильним.**

**Отримати кімнати**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Додати кімнату**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Отримати по конкретному айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Замінити по конкретному айді кімнату**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить текст, програмне забезпечення, Мультимедійне програмне забезпечення, знімок екрана

Вміст на основі ШІ може бути неправильним.**

**Видалити кімнату по конкретному айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст на основі ШІ може бути неправильним.**

**Отримати юзера**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Додати юзера**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, ряд

Вміст на основі ШІ може бути неправильним.**

**Отримати юзера за конкретним айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Замінити юзера**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить текст, програмне забезпечення, Мультимедійне програмне забезпечення, ряд

Вміст на основі ШІ може бути неправильним.**

**Видалити за конкретним айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст на основі ШІ може бути неправильним.**

**Отримати резерви**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Додати резерв**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить текст, програмне забезпечення, Мультимедійне програмне забезпечення, ряд

Вміст на основі ШІ може бути неправильним.**

**Отримати резерв за конкретним айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Видалити резерв**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Паралель

Вміст на основі ШІ може бути неправильним.**

**Отримати відгуки**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Додати відгук**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

**Отримати конкретний відгук**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Замінити відгук**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Видалити відгук за айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст на основі ШІ може бути неправильним.**

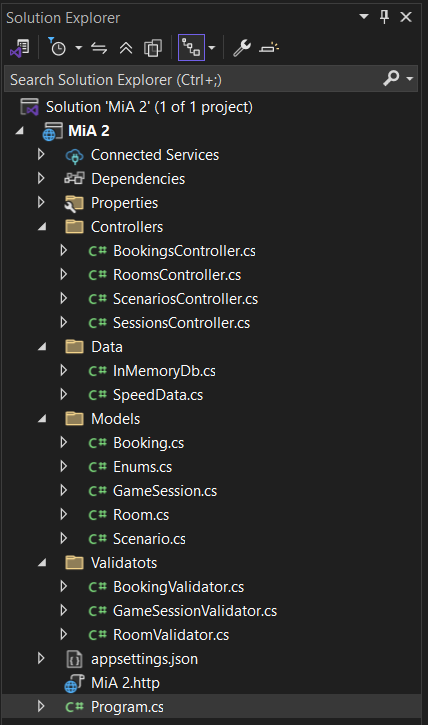
**Відгуки по конкретній кімнаті**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Завдання 2**

**Складова проєкту**

****

**Коди програми**

**Controllers**

**BoookingControllers.cs**

using Microsoft.AspNetCore.Mvc;

using Data;

using Models;

namespace Controllers;

[ApiController]

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

public class BookingsController : ControllerBase

{

private readonly InMemoryDb \_db;

public BookingsController(InMemoryDb db) => \_db = db;

[HttpGet] public ActionResult<IEnumerable<Booking>> Get() => \_db.Bookings;

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

public ActionResult<Booking> GetById(int id)

=> \_db.Bookings.FirstOrDefault(b => b.Id == id) is { } b ? Ok(b) : NotFound();

[HttpPost]

public ActionResult<Booking> Create([FromBody] Booking input)

{

if (!ModelState.IsValid) return ValidationProblem(ModelState);

var room = \_db.Rooms.FirstOrDefault(r => r.Id == input.RoomId && r.Active);

if (room is null) return BadRequest(new { error = "RoomId invalid or inactive." });

if (input.Players > room.Capacity)

return BadRequest(new { error = "Players exceed room capacity." });

bool overlaps = \_db.Bookings.Any(b => b.RoomId == input.RoomId &&

Math.Abs((b.StartUtc - input.StartUtc).TotalMinutes) < 90);

if (overlaps) return Conflict(new { error = "Time slot overlaps with existing booking." });

input.Id = \_db.NewBookingId();

\_db.Bookings.Add(input);

return CreatedAtAction(nameof(GetById), new { id = input.Id }, input);

}

[HttpPatch("{id:int}/approve")]

public ActionResult<Booking> Approve(int id)

{

var idx = \_db.Bookings.FindIndex(b => b.Id == id);

if (idx < 0) return NotFound();

var b = \_db.Bookings[idx]; b.Approved = true; \_db.Bookings[idx] = b;

return Ok(b);

}

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

public IActionResult Delete(int id)

{

var b = \_db.Bookings.FirstOrDefault(x => x.Id == id);

if (b is null) return NotFound();

\_db.Sessions.RemoveAll(s => s.BookingId == id);

\_db.Bookings.Remove(b);

return NoContent();

}

}

**RoomsControllers.cs**using Microsoft.AspNetCore.Mvc;

using Data;

using Models;

namespace Controllers;

[ApiController]

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

public class RoomsController : ControllerBase

{

private readonly InMemoryDb \_db;

public RoomsController(InMemoryDb db) => \_db = db;

[HttpGet] public ActionResult<IEnumerable<Room>> Get() => \_db.Rooms;

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

public ActionResult<Room> GetById(int id)

=> \_db.Rooms.FirstOrDefault(r => r.Id == id) is { } r ? Ok(r) : NotFound();

[HttpPost]

public ActionResult<Room> Create([FromBody] Room input)

{

if (!ModelState.IsValid) return ValidationProblem(ModelState);

input.Id = \_db.NewRoomId();

\_db.Rooms.Add(input);

return CreatedAtAction(nameof(GetById), new { id = input.Id }, input);

}

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

public ActionResult<Room> Update(int id, [FromBody] Room input)

{

if (!ModelState.IsValid) return ValidationProblem(ModelState);

var idx = \_db.Rooms.FindIndex(r => r.Id == id);

if (idx < 0) return NotFound();

input.Id = id;

\_db.Rooms[idx] = input;

return Ok(input);

}

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

public IActionResult Delete(int id)

{

var r = \_db.Rooms.FirstOrDefault(x => x.Id == id);

if (r is null) return NotFound();

\_db.Scenarios.RemoveAll(s => s.RoomId == id);

\_db.Bookings.RemoveAll(b => b.RoomId == id);

\_db.Rooms.Remove(r);

return NoContent();

}

}

**ScenariosControllers.cs**using Microsoft.AspNetCore.Mvc;

using Data;

using Models;

namespace Controllers;

[ApiController]

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

public class ScenariosController : ControllerBase

{

private readonly InMemoryDb \_db;

public ScenariosController(InMemoryDb db) => \_db = db;

[HttpGet] public ActionResult<IEnumerable<Scenario>> Get() => \_db.Scenarios;

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

public ActionResult<Scenario> GetById(int id)

=> \_db.Scenarios.FirstOrDefault(s => s.Id == id) is { } s ? Ok(s) : NotFound();

[HttpPost]

public ActionResult<Scenario> Create([FromBody] Scenario input)

{

if (!ModelState.IsValid) return ValidationProblem(ModelState);

if (!\_db.Rooms.Any(r => r.Id == input.RoomId))

return BadRequest(new { error = "RoomId does not exist." });

input.Id = \_db.NewScenarioId();

\_db.Scenarios.Add(input);

return CreatedAtAction(nameof(GetById), new { id = input.Id }, input);

}

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

public ActionResult<Scenario> Update(int id, [FromBody] Scenario input)

{

if (!ModelState.IsValid) return ValidationProblem(ModelState);

var idx = \_db.Scenarios.FindIndex(s => s.Id == id);

if (idx < 0) return NotFound();

if (!\_db.Rooms.Any(r => r.Id == input.RoomId))

return BadRequest(new { error = "RoomId does not exist." });

input.Id = id;

\_db.Scenarios[idx] = input;

return Ok(input);

}

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

public IActionResult Delete(int id)

{

var s = \_db.Scenarios.FirstOrDefault(x => x.Id == id);

if (s is null) return NotFound();

\_db.Scenarios.Remove(s);

return NoContent();

}

}

**SessionControllers.cs**using Microsoft.AspNetCore.Mvc;

using Data;

using Models;

namespace Controllers;

[ApiController]

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

public class SessionsController : ControllerBase

{

private readonly InMemoryDb \_db;

public SessionsController(InMemoryDb db) => \_db = db;

[HttpGet] public ActionResult<IEnumerable<GameSession>> Get() => \_db.Sessions;

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

public ActionResult<GameSession> GetById(int id)

=> \_db.Sessions.FirstOrDefault(s => s.Id == id) is { } s ? Ok(s) : NotFound();

[HttpPost]

public ActionResult<GameSession> Create([FromBody] GameSession input)

{

if (!ModelState.IsValid) return ValidationProblem(ModelState);

if (!\_db.Bookings.Any(b => b.Id == input.BookingId && b.Approved))

return BadRequest(new { error = "BookingId must exist and be approved." });

input.Id = \_db.NewSessionId();

input.Status = SessionStatus.Scheduled;

\_db.Sessions.Add(input);

return CreatedAtAction(nameof(GetById), new { id = input.Id }, input);

}

[HttpPatch("{id:int}/status")]

public ActionResult<GameSession> SetStatus(int id, [FromBody] SessionStatus status)

{

var idx = \_db.Sessions.FindIndex(s => s.Id == id);

if (idx < 0) return NotFound();

var s = \_db.Sessions[idx]; s.Status = status; \_db.Sessions[idx] = s;

return Ok(s);

}

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

public IActionResult Delete(int id)

{

var s = \_db.Sessions.FirstOrDefault(x => x.Id == id);

if (s is null) return NotFound();

\_db.Sessions.Remove(s);

return NoContent();

}

}

**Data**

**InMemoryDb.cs**using Models;

namespace Data;

public class InMemoryDb

{

public List<Room> Rooms { get; } = new();

public List<Scenario> Scenarios { get; } = new();

public List<Booking> Bookings { get; } = new();

public List<GameSession> Sessions { get; } = new();

public int NextRoomId { get; private set; } = 0;

public int NextScenarioId { get; private set; } = 0;

public int NextBookingId { get; private set; } = 0;

public int NextSessionId { get; private set; } = 0;

public int NewRoomId() => ++NextRoomId;

public int NewScenarioId() => ++NextScenarioId;

public int NewBookingId() => ++NextBookingId;

public int NewSessionId() => ++NextSessionId;

}

**SpeedData.cs**using Models;

namespace Data;

public static class SeedData

{

public static void EnsureSeed(InMemoryDb db)

{

if (db.Rooms.Any()) return;

var r1 = new Room { Id = db.NewRoomId(), Name = "Pharaoh's Tomb", Capacity = 6, Active = true };

var r2 = new Room { Id = db.NewRoomId(), Name = "Cyber Heist", Capacity = 5, Active = true };

db.Rooms.AddRange(new[] { r1, r2 });

var s1 = new Scenario { Id = db.NewScenarioId(), Title = "Beginner Tomb", DurationMinutes = 60, Difficulty = Difficulty.Medium, RoomId = r1.Id };

var s2 = new Scenario { Id = db.NewScenarioId(), Title = "Hardcore Heist", DurationMinutes = 75, Difficulty = Difficulty.Hard, RoomId = r2.Id };

db.Scenarios.AddRange(new[] { s1, s2 });

var b1 = new Booking

{

Id = db.NewBookingId(),

RoomId = r1.Id,

StartUtc = DateTime.UtcNow.AddHours(2),

Players = 4,

CustomerName = "Taras Hrytsenko",

CustomerPhone = "+380671112233",

Approved = true

};

var b2 = new Booking

{

Id = db.NewBookingId(),

RoomId = r2.Id,

StartUtc = DateTime.UtcNow.AddHours(4),

Players = 3,

CustomerName = "Oksana S.",

CustomerPhone = "+380931234567",

Approved = false

};

db.Bookings.AddRange(new[] { b1, b2 });

var g1 = new GameSession { Id = db.NewSessionId(), BookingId = b1.Id, Status = SessionStatus.Scheduled, Notes = "Seeded session" };

db.Sessions.Add(g1);

}

}

**Booking.cs**namespace Models;

public class Booking

{

public int Id { get; set; }

public int RoomId { get; set; }

public DateTime StartUtc { get; set; }

public int Players { get; set; }

public string CustomerName { get; set; } = "";

public string CustomerPhone { get; set; } = "";

public bool Approved { get; set; } = false;

}

**Enums.cs**

namespace Models;

public enum Difficulty { Easy, Medium, Hard, Extreme }

public enum SessionStatus { Scheduled, InProgress, Completed, Cancelled }

**GameSession.cs**namespace Models;

public class GameSession

{

public int Id { get; set; }

public int BookingId { get; set; }

public SessionStatus Status { get; set; } = SessionStatus.Scheduled;

public string? Notes { get; set; }

}

**Room.cs**

namespace Models;

public class Room

{

public int Id { get; set; }

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

public int Capacity { get; set; }

public bool Active { get; set; } = true;

}

**Scenario.cs**using System.ComponentModel.DataAnnotations;

namespace Models;

public class Scenario

{

public int Id { get; set; }

[Required, MinLength(3)]

public string Title { get; set; } = "";

[Range(1, 240)]

public int DurationMinutes { get; set; } = 60;

[Required]

public Difficulty Difficulty { get; set; }

[Required]

public int RoomId { get; set; }

}

**Validatots**

**BookingValidators.cs**using FluentValidation;

using Models;

namespace Validators;

public class BookingValidator : AbstractValidator<Booking>

{

public BookingValidator()

{

RuleFor(x => x.RoomId).GreaterThan(0);

RuleFor(x => x.StartUtc).GreaterThan(DateTime.UtcNow.AddMinutes(-1));

RuleFor(x => x.Players).InclusiveBetween(1, 10);

RuleFor(x => x.CustomerName).NotEmpty().MinimumLength(3);

RuleFor(x => x.CustomerPhone)

.NotEmpty()

.Matches(@"^**\+**380\d{9}$")

.WithMessage("Phone must be like +380XXXXXXXXX");

}

}

**GameSessionValidator.cs**using FluentValidation;

using Models;

namespace Validators;

public class GameSessionValidator : AbstractValidator<GameSession>

{

public GameSessionValidator()

{

RuleFor(x => x.BookingId).GreaterThan(0);

RuleFor(x => x.Status).IsInEnum();

RuleFor(x => x.Notes).MaximumLength(500);

}

}

**RoomValidator.cs**using FluentValidation;

using Models;

namespace Validators;

public class RoomValidator : AbstractValidator<Room>

{

public RoomValidator()

{

RuleFor(x => x.Name).NotEmpty().MinimumLength(3);

RuleFor(x => x.Capacity).InclusiveBetween(1, 10);

}

}

**Program.cs**

using FluentValidation;

using FluentValidation.AspNetCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddSingleton<Data.InMemoryDb>();

builder.Services.AddFluentValidationAutoValidation();

builder.Services.AddValidatorsFromAssemblyContaining<Validators.RoomValidator>();

var app = builder.Build();

using (var scope = app.Services.CreateScope())

{

var db = scope.ServiceProvider.GetRequiredService<Data.InMemoryDb>();

Data.SeedData.EnsureSeed(db);

}

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.MapControllers();

app.Run();

**Запускаєм програму**

Зображення, що містить текст, знімок екрана, число, Паралель

Вміст на основі ШІ може бути неправильним.

**Отримати бронювання**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

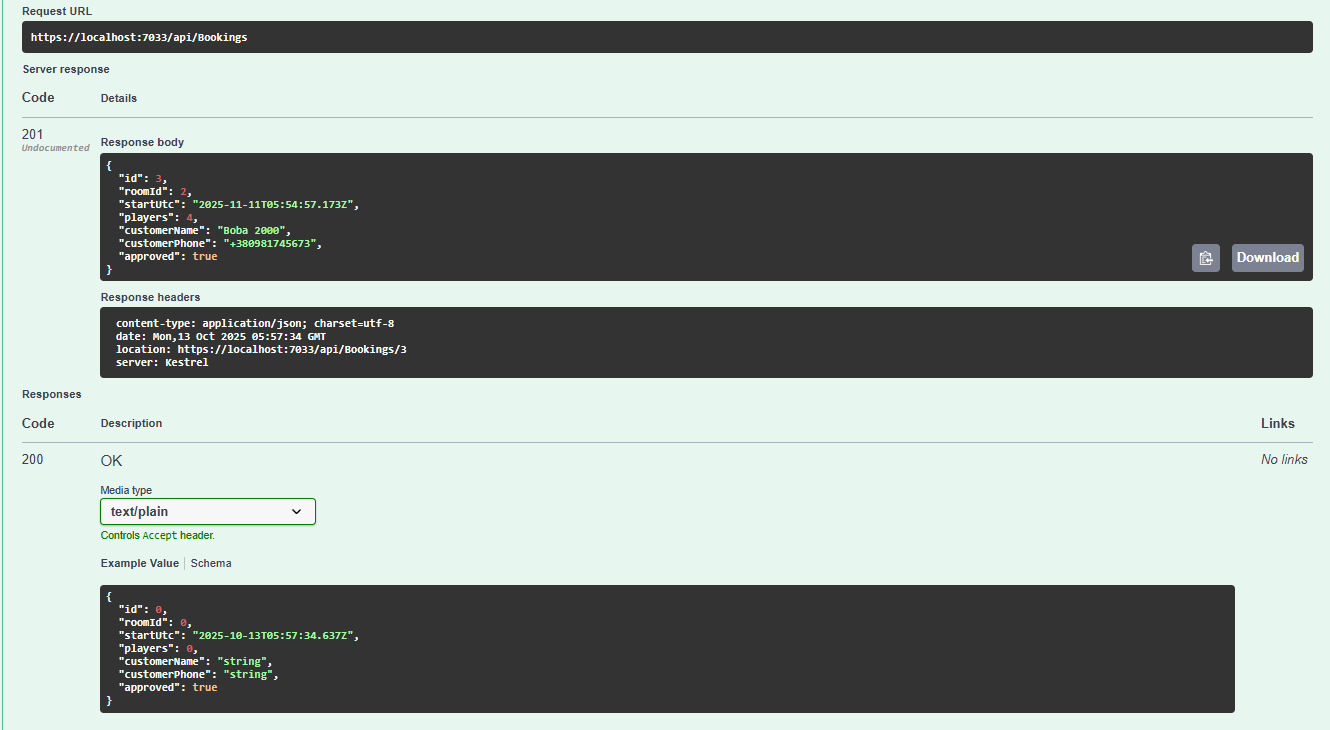
**Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Додаєм елемент**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

****

**Отримати бронювання по конкретному айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, програмне забезпечення, текст, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Видалити за конкретним айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст на основі ШІ може бути неправильним.**

**Підтверджуєм за конкретним айді  
Зображення, що містить текст, знімок екрана, програмне забезпечення, ряд

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, текст, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Отримати список кімнат**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, програмне забезпечення, текст, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Додати нову кімнату**

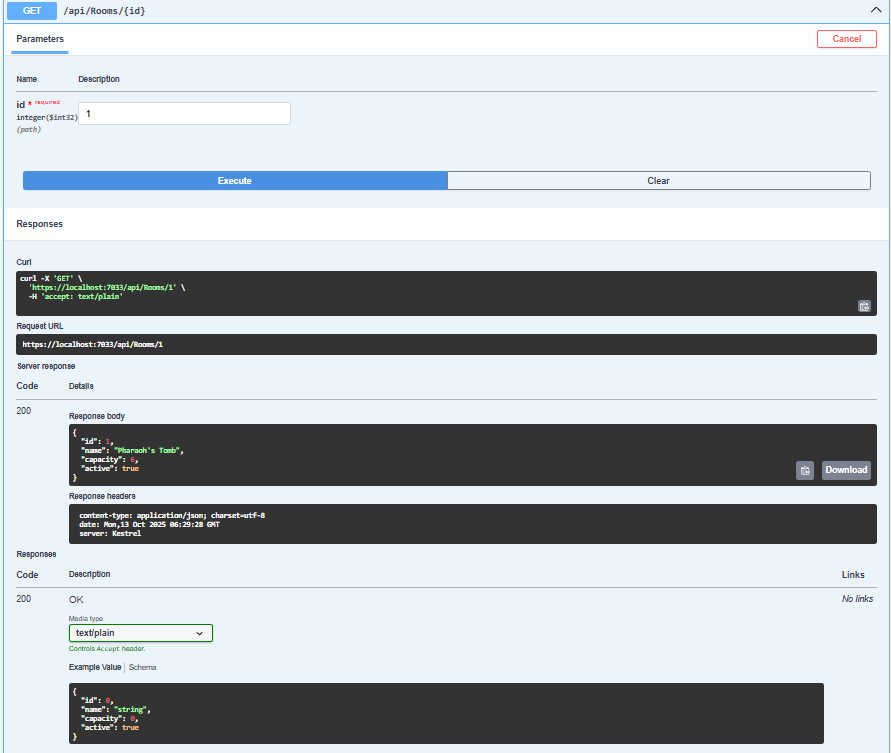
**Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Вміст на основі ШІ може бути неправильним.**

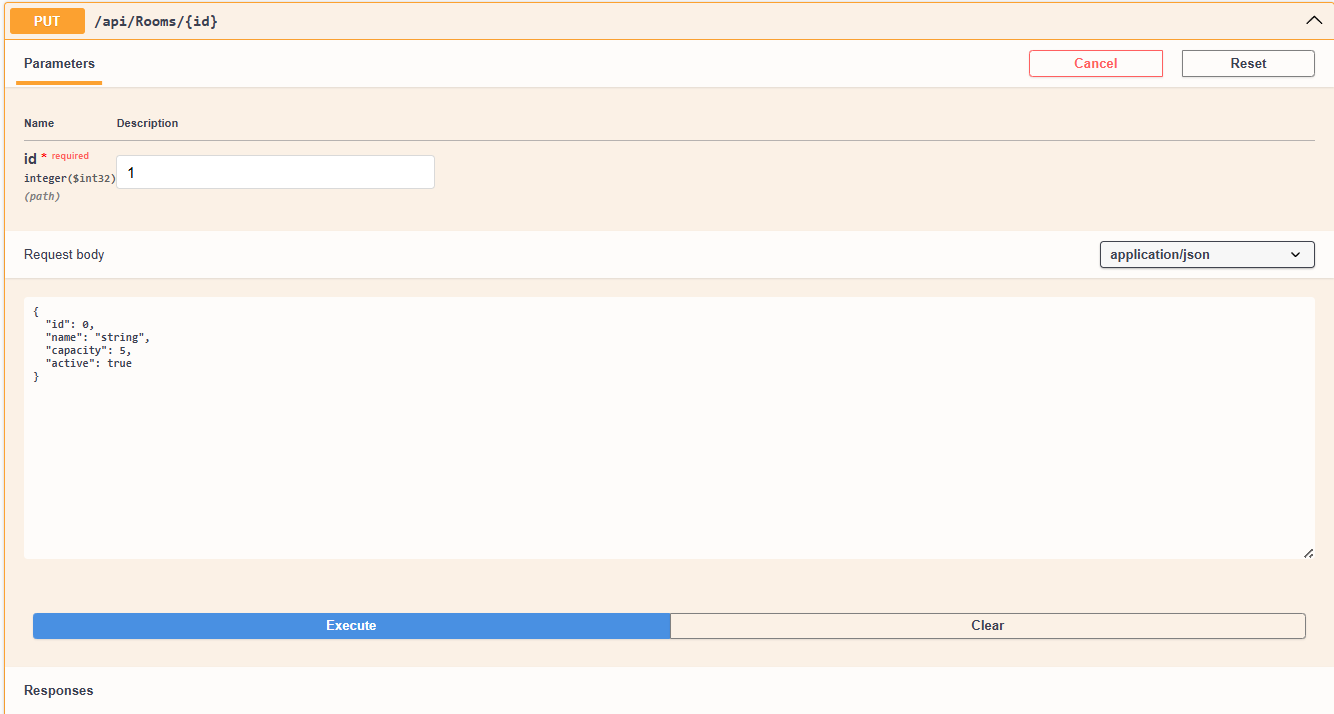
**Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

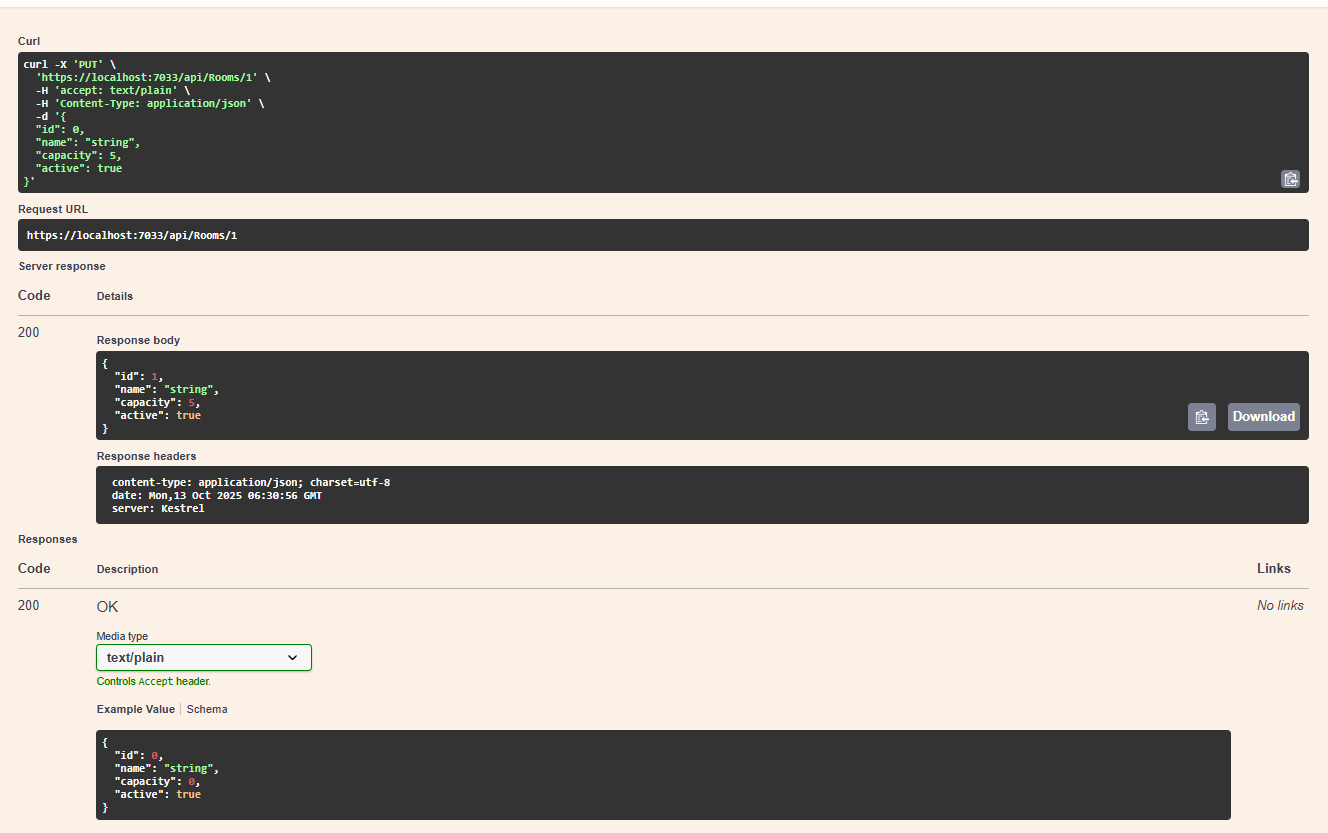
Вміст на основі ШІ може бути неправильним.**

**Отримати конкретну кімнату за айді**

****

**Змінити шось в якійсь конкретній кімнаті**

****

****

**Видаляєм кімнату по конкретному айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст на основі ШІ може бути неправильним.**

**Scenario  
Отримати сценарії**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Додаєм новий сценарій**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Отримати сценарій по конкретному айді**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Вміст на основі ШІ може бути неправильним.**

**Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст на основі ШІ може бути неправильним.**

**Видалити  
Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст на основі ШІ може бути неправильним.**