LABORATORY ASSIGNMENT №5

Authentication and authorization. Web API. JSON Web Token.

Goal: acquire skills with authorization authentication mechanisms in ASP.NET, get skills working with Web API.

Workflow:

Task 1. Implement user registration, authentication and authorization using Microsoft Identity package.

First, I’ve installed identity and configured identity:

```
public class WebsiteIdentityDbContext : IdentityDbContext<IdentityUser>
 {
     public WebsiteIdentityDbContext(DbContextOptions<WebsiteIdentityDbContext> options)
         : base(options) { }
 }
```

```
builder.Services.AddDbContext<WebsiteIdentityDbContext>(options =>
    options.UseSqlServer(builder.Configuration["ConnectionStrings:IdentityConnection"]));
```

```
"ConnectionStrings": {
  "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=TicketBookingWebsite;MultipleActiveResultSets=tr
ue",
  "IdentityConnection":
"Server=(localdb)\\mssqllocaldb;Database=TicketBookingWebsiteIdentity;MultipleActiveResul
tSets=true"
}
```

					ДУ «Житомирська політехніка».25.121.01.000 – Лр5
Змн.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Бейлах С.В.			Звіт з лабораторної роботи
Перевір.		Українець М.О.			
Керівник					
Н. контр.					
Зав. каф.					

Літ.	Арк.	Аркушів
	1	18
ФІКТ Гр. ІПЗ-23-3[1]		

After configuring Identity, implement the following functionalities:

1. User registration
   a. Set a requirement for the uniqueness of the user's email address
   b. Set password requirements: no shorter than 8 characters, contains numbers and letters, must contain at least one uppercase letter
   c. Implement a field for confirming the entered password *(you can add client side equality check using JS and get additional points)*
2. User authentication (Login)
3. User logout (Logout)
4. User's personal account, where he will be able to view or change his own data. *(you can add password recovery with email for additional points)*

1. User registration

a. Set a requirement for the uniqueness of the user's email address

b. Set password requirements: no shorter than 8 characters, contains numbers and letters, must contain at least one uppercase letter

I’ve added configuration into Program.cs to match requirements.

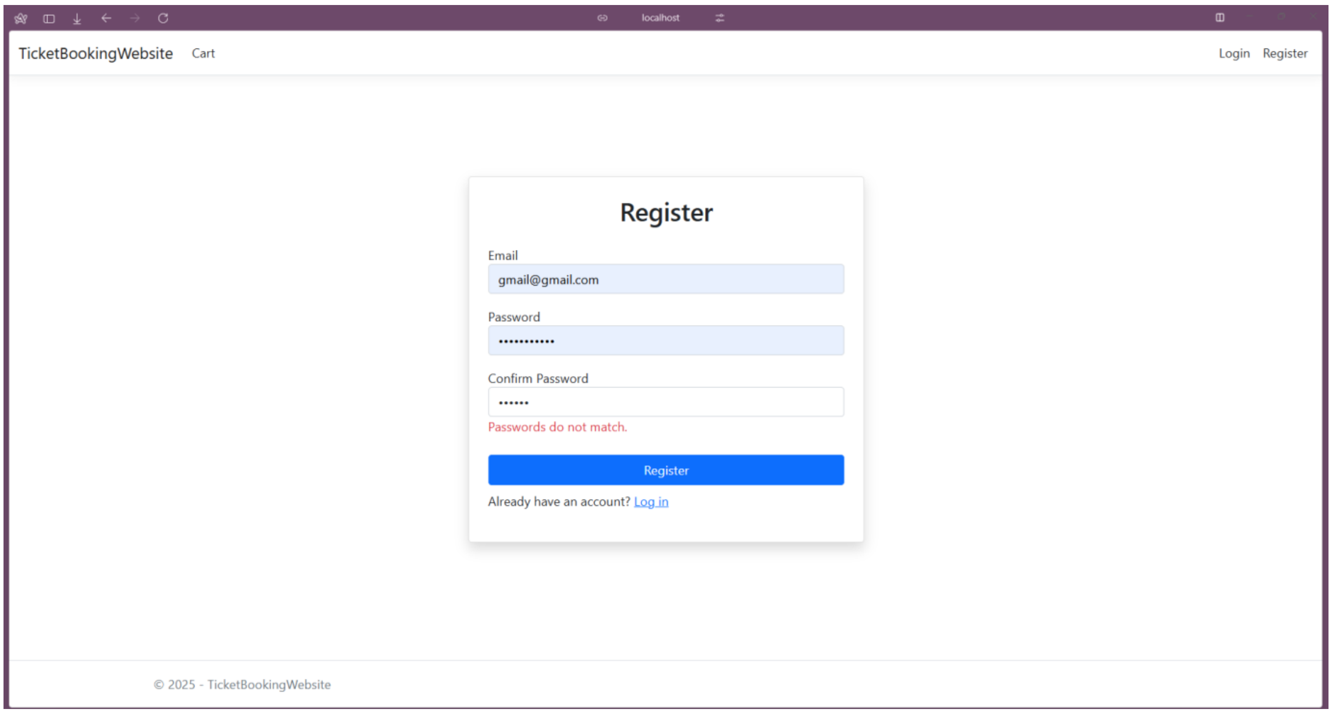
```
builder.Services.AddIdentity<IdentityUser, IdentityRole>(options =>
{
    options.User.RequireUniqueEmail = true;

    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = true;
})
```

c. Implement a field for confirming the entered password *(you can add client side equality check using JS and get additional points)*

```
public class RegisterModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [MinLength(8, ErrorMessage = "Password must be at least 8 characters long")]
    [RegularExpression(@"^(?=.*[A-Z])(?=.*\d)(?=.*[a-z]).{8,}$", ErrorMessage = "Password
must contain at least one uppercase letter, one lowercase letter, and one number.")]
    public string Password { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Compare("Password", ErrorMessage = "Passwords do not match.")]
    public string ConfirmPassword { get; set; }
}
```

In fact it was implemented using JS, cause it was using DataAnnotations, which equality checks data using jquery validation scripts. Script is included at the end of view.

```
@model TicketBookingWebsite.ViewModels.RegisterModel

<div class="d-flex justify-content-center align-items-center" style="min-height: 80vh;">
    <div class="card p-4 shadow" style="min-width: 300px; max-width: 500px; width:
100%;">
        <h2 class="text-center mb-4">Register</h2>

        <form asp-action="Register" method="post">
            <div class="form-group mb-3">
                <label for="Email">Email</label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>

            <div class="form-group mb-3">
                <label for="Password">Password</label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>

            <div class="form-group mb-4">
                <label for="ConfirmPassword">Confirm Password</label>
                <input asp-for="ConfirmPassword" class="form-control" />
                <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
            </div>

            <button type="submit" class="btn btn-primary w-100">Register</button>

            <div class="form-group mb-2 mt-2">
                <p>Already have an account? <a asp-action="Login">Log in</a></p>
            </div>
        </form>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```



Register

Email

gmail@gmail.com

Password

•••••••••••

Confirm Password

••••••

Passwords do not match.

Register

Already have an account? Log in

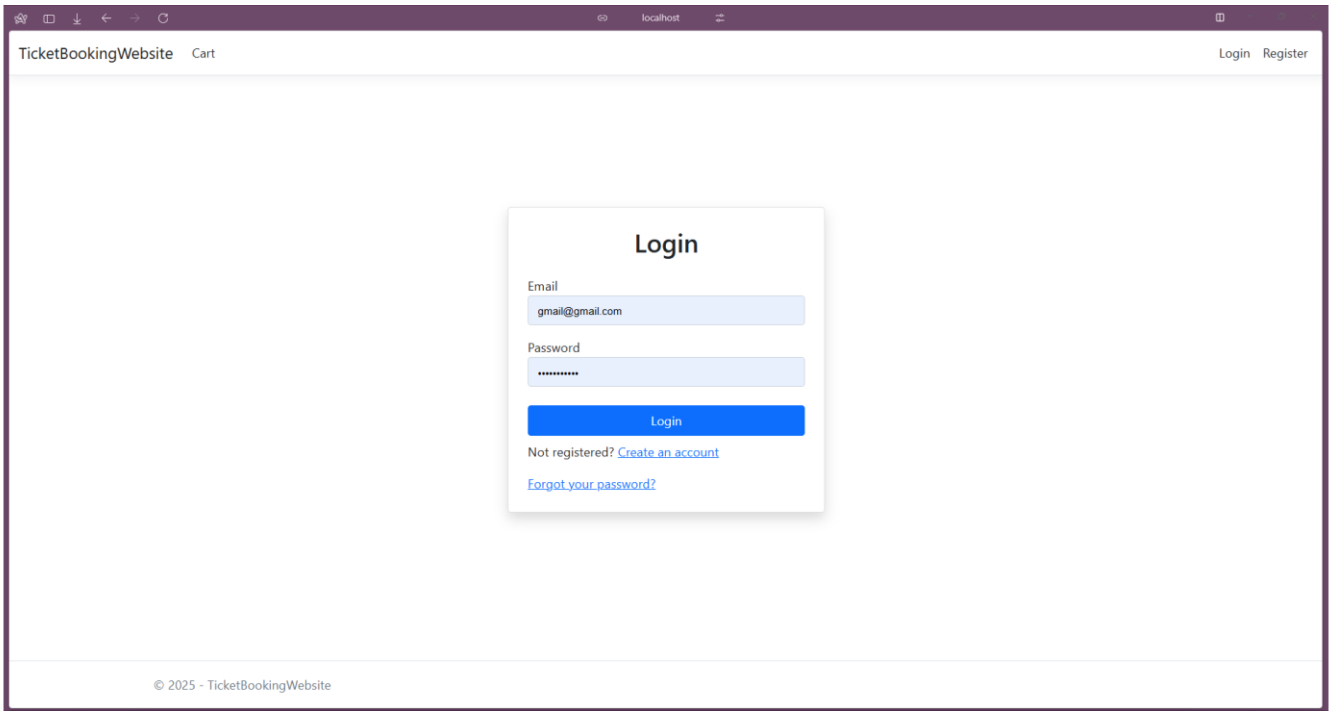
© 2025 - TicketBookingWebsite

2. User authentication (Login):

```
public IActionResult Login(string returnUrl) =>
    View(new LoginModel { ReturnUrl = returnUrl });

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginModel loginModel)
{
    if (ModelState.IsValid)
    {
        var user = await _userManager.FindByEmailAsync(loginModel.Email);
        if (user != null)
        {
            var result = await _signInManager.PasswordSignInAsync(user,
loginModel.Password, false, false);
            if (result.Succeeded)
            {
                return Redirect(loginModel.ReturnUrl ?? "/");
            }
        }
        ModelState.AddModelError("", "Invalid name or password");
    }
    return View(loginModel);
}


@model TicketBookingWebsite.ViewModels.LoginModel

<div class="d-flex justify-content-center align-items-center" style="height: 80vh;">
    <div class="card p-4 shadow" style="min-width: 300px; max-width: 400px; width: 
100%;">
        <h2 class="text-center mb-4">Login</h2>

        <form asp-action="Login" method="post">
            <div class="form-group mb-3">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>

            <div class="form-group mb-4">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" type="password" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>

            <button type="submit" class="btn btn-primary w-100">Login</button>

            <div class="form-group mb-2 mt-2">
                <p>Not registered? <a asp-action="Register">Create an account</a></p>
            </div>

            <div class="form-group">
                <a href="#" onclick="redirectToForgotPassword()" 
id="forgotPasswordLink">Forgot your password?</a>
            </div>
        </form>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
    <script>
        function redirectToForgotPassword() {
            const email = document.querySelector('input[name="Email"]').value;
            const url = `/Account/ForgotPassword${email ? '?email=' + 
encodeURIComponent(email) : ''}`;
            window.location.href = url;
        }
    </script>
}
```

I’ve also added password recovery here, which was mentioned in subtask 4:

User's personal account, where he will be able to view or change his own data. ***(you can add password recovery with email for additional points)***



TicketBookingWebsite Cart Login Register

Login

Email
gmail@gmail.com
Password
••••••••••
Login
Not registered? Create an account
Forgot your password?

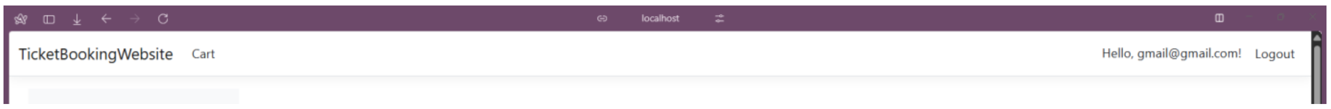
© 2025 - TicketBookingWebsite

3. User logout (Logout)

```
[Authorize]
public async Task<IActionResult> Logout()
{
    await _signInManager.SignOutAsync();
    return RedirectToAction("Index", "Home");
}
```

Logout button is placed on _Layout:

```
        @if (User.Identity?.IsAuthenticated ?? false)
        {
            <li class="nav-item">
                <a class="nav-link text-dark" asp-controller="Account" asp-
action="Manage">Hello, @User.Identity.Name!</a>
            </li>
            <li class="nav-item">
                <form class="form-inline" asp-area="" asp-controller="Account" asp-
action="Logout" method="post">
                    <button type="submit" class="nav-link btn btn-link text-
dark">Logout</button>
                </form>
            </li>
        }
```



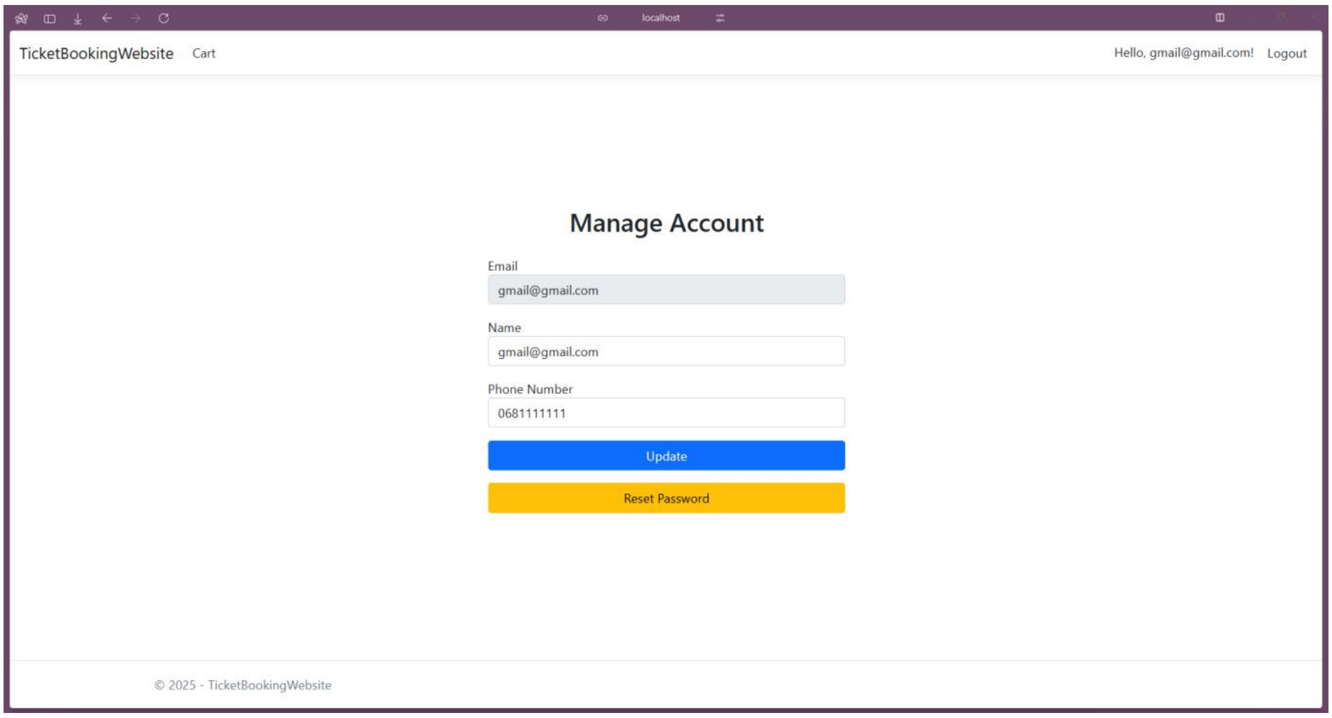
TicketBookingWebsite Cart Hello, gmail@gmail.com! Logout

4. User's personal account, where he will be able to view or change his own data. ***(you can add password recovery with email for additional points)***

Here is all controlled methods related to account management:

```
[Authorize]
[HttpGet]
public async Task<IActionResult> Manage()
{
    var user = await _userManager.GetUserAsync(User);
    if (user == null) return RedirectToAction("Login");

    var model = new UserProfileModel
    {
        Email = user.Email,
        Name = user.UserName,
        PhoneNumber = user.PhoneNumber
    };

    return View(model);
}

[HttpPost]
[Authorize]
[ValidateAntiForgeryToken]
public async Task<IActionResult> UpdateProfile(UserProfileModel model)
{
    var user = await _userManager.GetUserAsync(User);
    if (user == null) return RedirectToAction("Login");

    if (ModelState.IsValid)
    {
        user.UserName = model.Name;
        user.PhoneNumber = model.PhoneNumber;
        var result = await _userManager.UpdateAsync(user);

        if (result.Succeeded)
        {
            ViewBag.StatusMessage = "Profile updated successfully!";
        }
        else
        {
            foreach (var error in result.Errors)
                ModelState.AddModelError("", error.Description);
        }
    }
    return View("Manage", model);
}

[HttpGet]
[AllowAnonymous]
public IActionResult ForgotPassword(string email = "")
{
    var model = new ForgotPasswordModel { Email = email };
    return View(model);
}

[HttpPost]
[AllowAnonymous]
```

```
[ValidateAntiForgeryToken]
public async Task<IActionResult> ForgotPassword(ForgotPasswordModel model, [FromServices]
IEmailSender emailSender)
{
    if (!ModelState.IsValid) return View(model);

    InfoPageModel infoPageModel = new InfoPageModel
    {
        Message = "Password reset mail has been sent. Check your email."
    };

    var user = await _userManager.FindByEmailAsync(model.Email);
    if (user == null)
    {
        return View("InfoPage", infoPageModel);
    }

    var token = await _userManager.GeneratePasswordResetTokenAsync(user);
    var callbackUrl = Url.Action("ResetPassword", "Account",
        new { token, email = model.Email }, Request.Scheme);

    await emailSender.SendEmailAsync(model.Email, "Reset Password",
        $"Please reset your password by clicking <a href='{callbackUrl}'>here</a>.");

    return View("InfoPage", infoPageModel);
}

[HttpGet]
[AllowAnonymous]
public IActionResult InfoPage() =>
    View();

[HttpGet]
[AllowAnonymous]
public IActionResult ResetPassword(string token, string email) =>
    View(new ResetPasswordModel { Token = token, Email = email });

[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> ResetPassword(ResetPasswordModel model)
{
    if (!ModelState.IsValid) return View(model);

    InfoPageModel infoPageModel = new InfoPageModel
    {
        Message = "Password has been reset."
    };

    var user = await _userManager.FindByEmailAsync(model.Email);
    if (user == null)
    {
        return View("InfoPage", infoPageModel);
    }

    var result = await _userManager.ResetPasswordAsync(user, model.Token,
model.Password);
    if (result.Succeeded)
    {
        return View("InfoPage", infoPageModel);
    }

    foreach (var error in result.Errors)
    {
        ModelState.AddModelError("", error.Description);
    }

    return View(model);
}
```



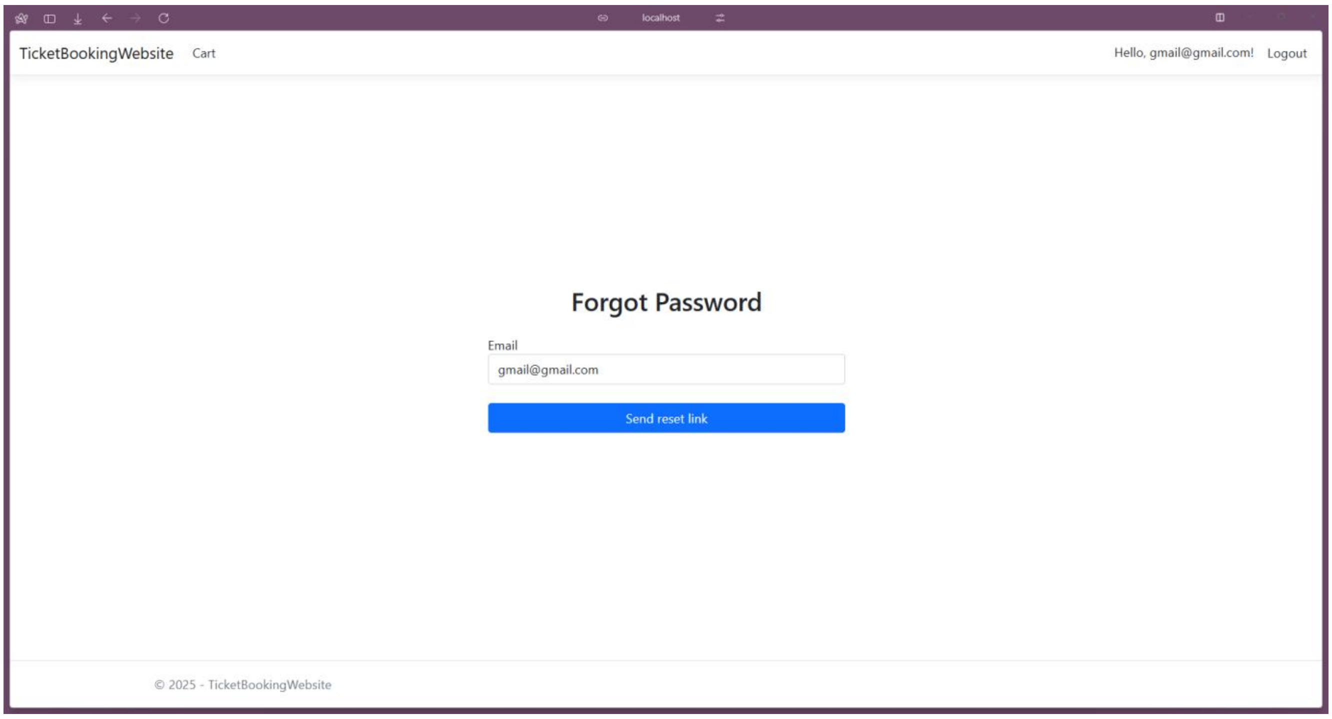
TicketBookingWebsite Cart Hello, gmail@gmail.com! Logout

Manage Account

Email: gmail@gmail.com

Name: gmail@gmail.com

Phone Number: 0681111111



TicketBookingWebsite Cart Hello, gmail@gmail.com! Logout

Forgot Password

Email: gmail@gmail.com

Password recovery is emplemented using Email sender:

```
using System.Diagnostics;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using System.Net.Mail;
using System.Net;
using System.Threading.Tasks;

namespace TicketBookingWebsite.Helpers
{
    public class EmailSettings
    {
        public string SmtpServer { get; set; }
        public int SmtpPort { get; set; }
        public string SmtpUser { get; set; }
        public string SmtpPass { get; set; }
    }

    public class EmailSender : IEmailSender
    {
        private readonly EmailSettings _settings;

        public EmailSender(IOptions<EmailSettings> settings)
        {
            _settings = settings.Value;
        }

        public Task SendEmailAsync(string email, string subject, string htmlMessage)
        {
            try
            {
                var mail = new MailMessage
                {
                    From = new MailAddress(_settings.SmtpUser),
                    Subject = subject,
                    Body = htmlMessage,
                    IsBodyHtml = true
                };
                mail.To.Add(email);

                var client = new SmtpClient(_settings.SmtpServer, _settings.SmtpPort)
                {
                    Credentials = new NetworkCredential(_settings.SmtpUser,
_settings.SmtpPass),
                    EnableSsl = true
                };

                return client.SendMailAsync(mail);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"[ERROR] Email sending failed: {ex.Message}");
                throw;
            }
        }
    }
}


builder.Services.Configure<EmailSettings>(builder.Configuration.GetSection("EmailSettings
"));
builder.Services.AddTransient<IEmailSender, EmailSender>();
```

Additional points task (5 points): implement work with user roles.

Provide the possibility of registering two types of users in the project, for example, a patient and a doctor. If the project topic does not involve working with several types of users, then implement the roles of a regular user and an administrator.

```
builder.Services.AddIdentity<IdentityUser, IdentityRole>(
```

I’ve added *IdentityRole* to *Identity* service then seeded database with roles:

```
private static async Task SeedRoles(RoleManager<IdentityRole> roleManager)
 {
     string[] roleNames = { "Admin", "User" };

     foreach (var roleName in roleNames)
     {
         var roleExist = await roleManager.RoleExistsAsync(roleName);
         if (!roleExist)
         {
             await roleManager.CreateAsync(new IdentityRole(roleName));
         }
     }
 }
```

Also added default admin and user seeding:

```
private static async Task SeedUsers(UserManager<IdentityUser> userManager)
 {
     var adminUser = await userManager.FindByEmailAsync("admin@ticketbooking.com");

     if (adminUser == null)
     {
         adminUser = new IdentityUser
         {
             UserName = "admin@ticketbooking.com",
             Email = "admin@ticketbooking.com"
         };
         var result = await userManager.CreateAsync(adminUser, "AdminPassword123!");

         if (result.Succeeded)
         {
             await userManager.AddToRoleAsync(adminUser, "Admin");
         }
     }

     var regularUser = await userManager.FindByEmailAsync("user@ticketbooking.com");

     if (regularUser == null)
     {
         regularUser = new IdentityUser
         {
             UserName = "user@ticketbooking.com",
             Email = "user@ticketbooking.com"
         };
         var result = await userManager.CreateAsync(regularUser, "UserPassword123!");
```

```
            if (result.Succeeded)
            {
                await userManager.AddToRoleAsync(regularUser, "User");
            }
        }
    }
```

I’ve implemented 2 roles: *User* and *Admin*. *User* is just basic without additional permissions. *Admin* is user, who can control all events and bookings.

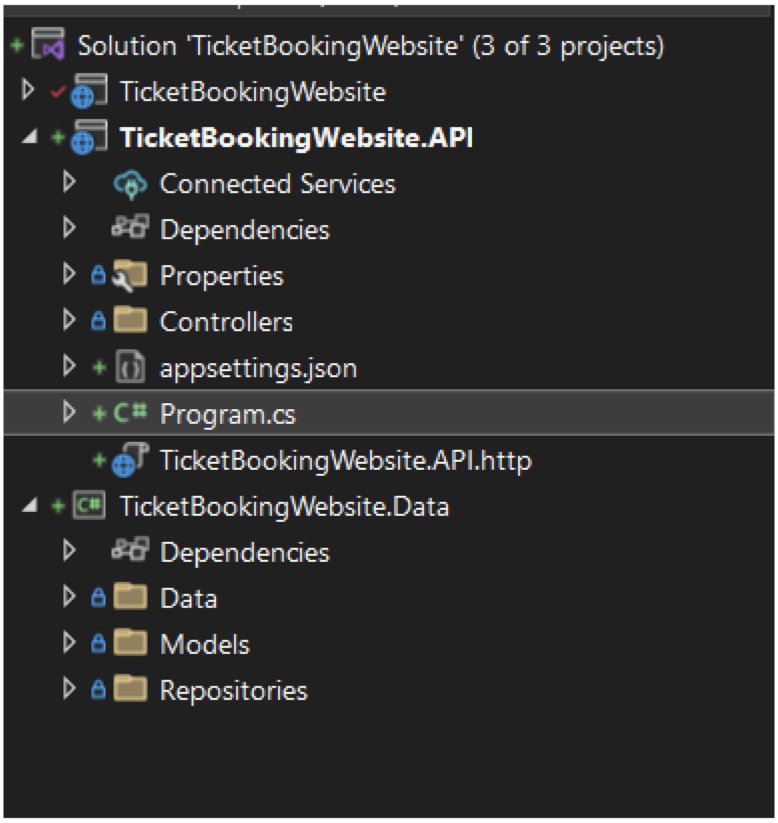
```
[Authorize(Roles= "Admin")]
 public class BookingController : Controller


[Authorize(Roles = "Admin")]
public class EventController : Controller
```

Task 2.

Create a new WebAPI project in your solution. This project should implement the functionality of your MVC project as REST endpoints. For better code organization it is recommended to move classes for interacting with the database (models, contexts, repositories) to the separate Class Library project and add reference to this library in both MVC and WebAPI projects.
After this, configure ConnectionStrings for the created databases inside the WebAPI project appsettings.json file. Thus, after moving the files for the interaction with the database to a separate library, we can reuse all the necessary services in both projects.

Implement all the necessary CRUD-operations in your WebAPI project, including user registration. Test your endpoints using Postman, Swagger or any other suitable tool.

First, I’ve split my project into 3:



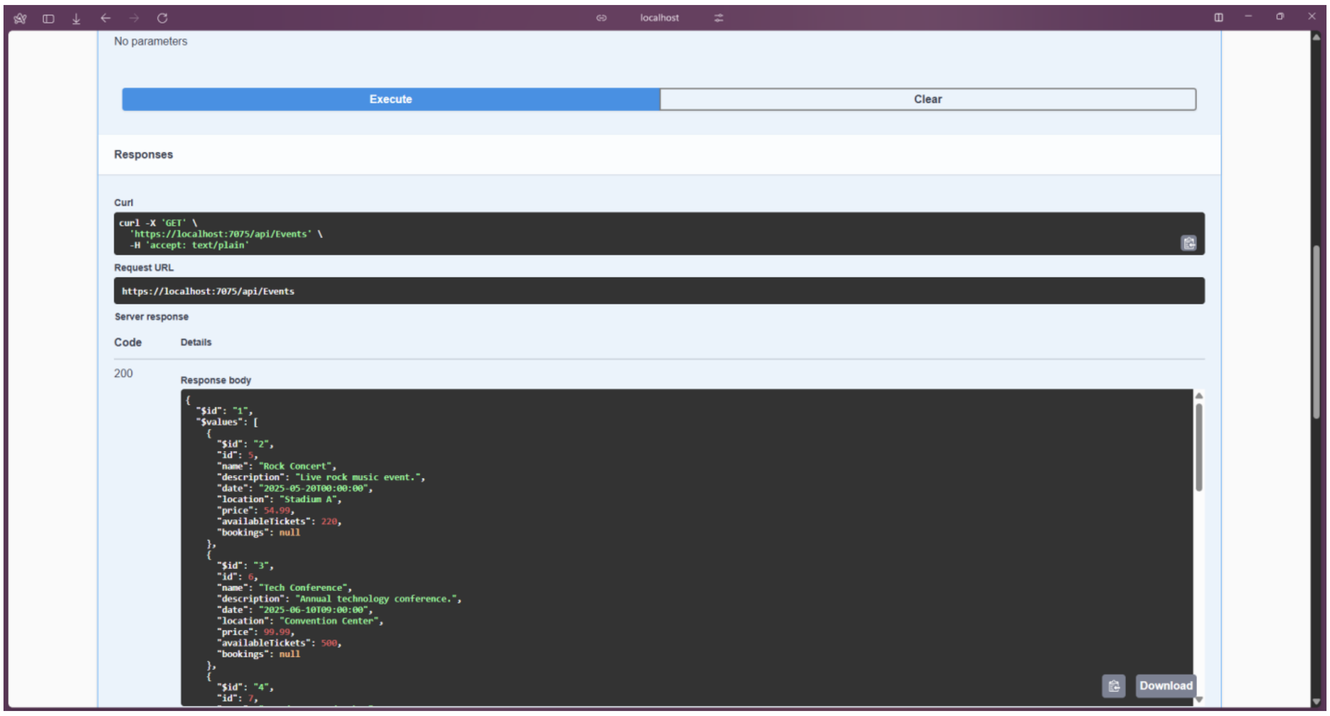
```
Solution 'TicketBookingWebsite' (3 of 3 projects)
  TicketBookingWebsite
  TicketBookingWebsite.API
    Connected Services
    Dependencies
    Properties
    Controllers
    appsettings.json
    Program.cs
    TicketBookingWebsite.API.http
  TicketBookingWebsite.Data
    Dependencies
    Data
    Models
    Repositories
```

Then I’ve modified Program.cs and created controllers for Booking and Events:

```
builder.Services.AddDbContext<WebsiteDbContext>(options =>
    options.UseSqlServer(builder.Configuration["ConnectionStrings:DefaultConnection"]));

builder.Services.AddDbContext<WebsiteIdentityDbContext>(options =>
    options.UseSqlServer(builder.Configuration["ConnectionStrings:IdentityConnection"]));


builder.Services.AddControllers().AddJsonOptions(options =>
{
    options.JsonSerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.Preserve;
    options.JsonSerializerOptions.WriteIndented = true;
});
```

Controllers are in the GitHub repository

Testing API:



Curl

```
curl -X 'GET' \
  'https://localhost:7075/api/Events' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7075/api/Events
```

Server response

Code: 200

Response body

```
{
  "$id": "1",
  "$values": [
    {
      "$id": "2",
      "id": 5,
      "name": "Rock Concert",
      "description": "Live rock music event.",
      "date": "2025-05-20T00:00:00",
      "location": "Stadium A",
      "price": 54.99,
      "availableTickets": 220,
      "bookings": null
    },
    {
      "$id": "3",
      "id": 6,
      "name": "Tech Conference",
      "description": "Annual technology conference.",
      "date": "2025-06-10T09:00:00",
      "location": "Convention Center",
      "price": 99.99,
      "availableTickets": 500,
      "bookings": null
    },
    {
      "$id": "4",
      "id": 7,
```

Task 3.

Implement user authentication and authorization using the JWT mechanism. It should work the following way:

1. The client sends a request to the login endpoint with user login and password.
2. If a user with such a login and password exists, then the server must return the generated JSON Web Token.
3. To gain access to actions that require authorization, the client sends an Authorization header in the request with the value Bearer TokenValue, where TokenValue is the token that was generated during authentication.
4. If the token is valid, the server will execute the request, otherwise it will return a 401 Unauthorized error.

First I’ve configured JWT:

```
var jwtKey = builder.Configuration["Jwt:Key"];
var keyBytes = Encoding.ASCII.GetBytes(jwtKey);

builder.Services.AddControllers().AddJsonOptions(options =>
{
    options.JsonSerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.Preserve;
    options.JsonSerializerOptions.WriteIndented = true;
});

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(keyBytes),
        ClockSkew = TimeSpan.Zero
    };
});


builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new() { Title = "TicketBookingWebsite.Api", Version = "v1" });

    c.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme
    {
        Description = "Enter JWT token like this: Bearer {your token}",
        Name = "Authorization",
        In = Microsoft.OpenApi.Models.ParameterLocation.Header,
        Type = Microsoft.OpenApi.Models.SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });

    c.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement
    {
        {
            new Microsoft.OpenApi.Models.OpenApiSecurityScheme
            {
                Reference = new Microsoft.OpenApi.Models.OpenApiReference
                {
                    Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            Array.Empty<string>()
        }
    });
});
```

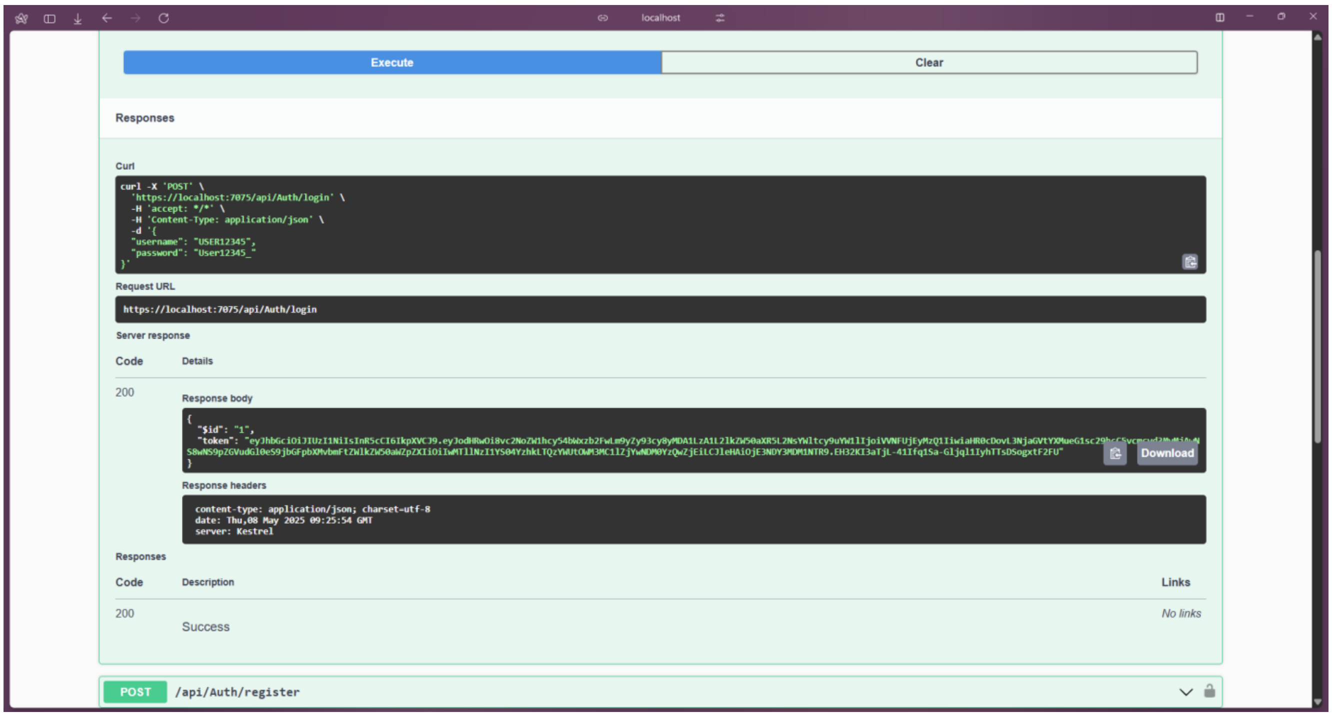
Added JWT Key:

```
"Jwt": {
  "Key": "very-cool-128-bit-long-secret-key"
}
```

And implemented AuthController:

```
[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    private readonly UserManager<IdentityUser> _userManager;
    private readonly SignInManager<IdentityUser> _signInManager;
    private readonly IConfiguration _config;

    public AuthController(UserManager<IdentityUser> userManager,
                          SignInManager<IdentityUser> signInManager,
                          IConfiguration config)
    {
        _userManager = userManager;
        _signInManager = signInManager;
        _config = config;
    }

    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] LoginModel model)
    {
        var user = await _userManager.FindByNameAsync(model.Username);
        if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
        {
            var token = GenerateJwtToken(user);
            return Ok(new { token });
        }
        return Unauthorized();
    }

    [HttpPost("register")]
    public async Task<IActionResult> Register([FromBody] RegisterModel model)
    {
        var user = new IdentityUser { UserName = model.Username, Email = model.Email };
        var result = await _userManager.CreateAsync(user, model.Password);

        if (result.Succeeded)
            return Ok(new { message = "User registered successfully." });

        return BadRequest(result.Errors);
    }

    private string GenerateJwtToken(IdentityUser user)
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.Name, user.UserName),
            new Claim(ClaimTypes.NameIdentifier, user.Id)
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var token = new JwtSecurityToken(
            claims: claims,
            expires: DateTime.UtcNow.AddHours(2),
            signingCredentials: creds);

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}

public class LoginModel
{
    public string Username { get; set; }
    public string Password { get; set; }
}

public class RegisterModel
{
    public string Username { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
}
```

Getting JWT Bearer:



```
curl -X 'POST' \
  'https://localhost:7075/api/Auth/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "USER12345",
  "password": "User12345_"
}'
```

Request URL

```
https://localhost:7075/api/Auth/login
```

Server response

Code 200

Response body

```
{
  "$id": "1",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiVVNFUjEyMzQ1IiwiaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvd2MvMjAwN
S8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZWlkZW50aWZpZXIiOiIwMTllNzI1YS04YzhkLTQzYWUtOWM3MC1lZjYwNDM0YzQwZjEiLCJleHAiOjE3NDY3MDM1NTR9.EH32KI3aTjL-41Ifq1Sa-Gljql1IyhTTsDSogxtF2FU"
}
```

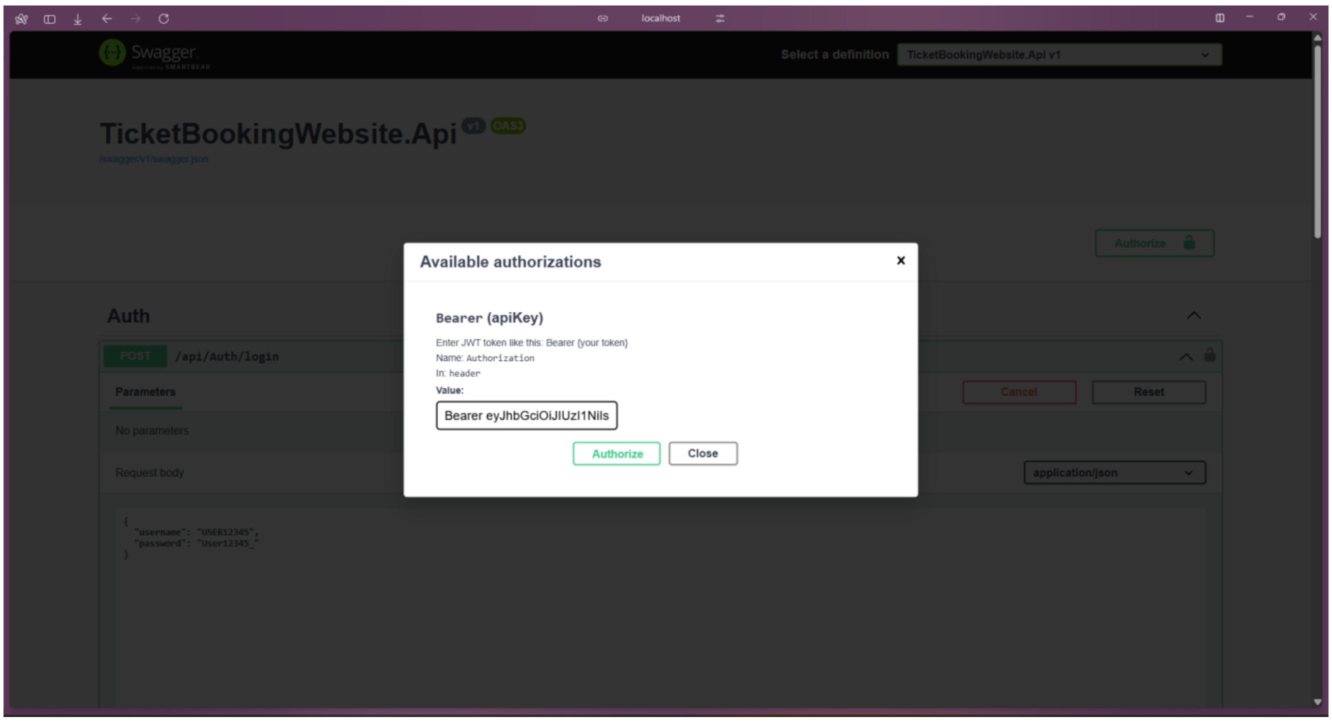
Response headers

```
content-type: application/json; charset-utf-8
date: Thu,08 May 2025 09:25:54 GMT
server: Kestrel
```

Responses

Code	Description	Links
200	Success	*No links*

Authorizing:



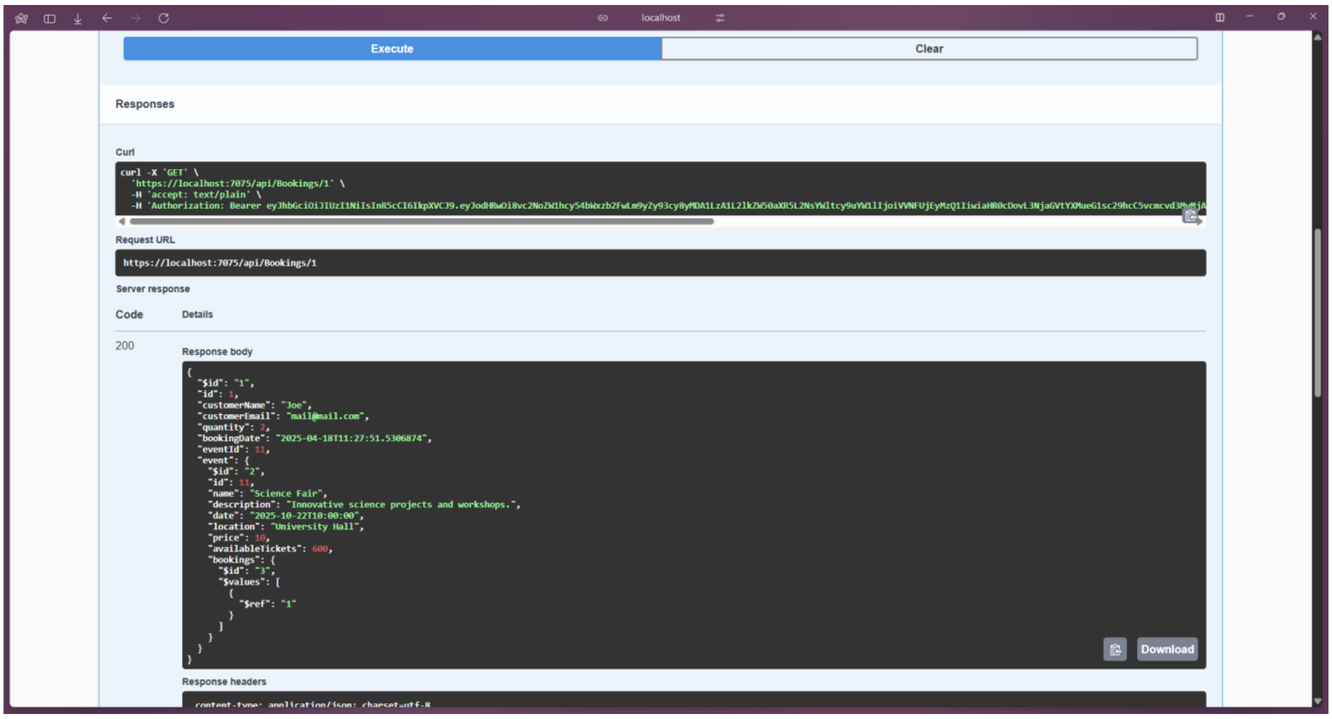
Available authorizations

Bearer (apiKey)

Enter JWT token like this: Bearer {your token}
Name: `Authorization`
In: `header`
Value: Bearer eyJhbGciOiJIUzI1NiIs

```
{
  "username": "USER12345",
  "password": "User12345_"
}
```

Getting booking with id 1, which requires authorization:



Curl

```
curl -X 'GET' \
  'https://localhost:7075/api/Bookings/1' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiVVNFUjEyMzQ1IiwiaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvd3MvMjA
```

Request URL

```
https://localhost:7075/api/Bookings/1
```

Server response

Code 200

Response body

```
{
  "$id": "1",
  "id": 1,
  "customerName": "Joe",
  "customerEmail": "mail@mail.com",
  "quantity": 2,
  "bookingDate": "2025-04-18T11:27:51.5306874",
  "eventId": 11,
  "event": {
    "$id": "2",
    "id": 11,
    "name": "Science Fair",
    "description": "Innovative science projects and workshops.",
    "date": "2025-10-22T10:00:00",
    "location": "University Hall",
    "price": 10,
    "availableTickets": 600,
    "bookings": {
      "$id": "3",
      "$values": [
        {
          "$ref": "1"
        }
      ]
    }
  }
}
```

Response headers

```
content-type: application/json; charset=utf-8
```

Goal: I acquired skills with authorization authentication mechanisms in ASP.NET, gained skills working with Web API.