ЛАБОРАТОРНА РОБОТА № 5

Аутентифікація та авторизація.

Робота з Web API. JSON Web Token.

Мета: набути навичок роботи з механізмами аутентифікації та авторизації користувачів в ASP.NET, набути навичок роботи з Web API.

Хід роботи

Завдання 1. Реалізувати реєстрацію, автентифікацію та авторизацію користувача з використанням пакету Microsoft Identity.

У межах завдання було інтегровано Microsoft Identity для керування користувачами. Це дозволило реалізувати реєстрацію, логін/логаут, а також контроль доступу до функціоналу застосунку. Під час виконання використовувалась версія пакетів, сумісна з .NET 10.

Спочатку до проєкту було додано залежність Microsoft.AspNetCore.Identity. EntityFrameworkCore та створено окремий контекст бази даних для Identity. Такий підхід (окремий контекст і окремий connection string) дозволяє ізолювати таблиці користувачів/ролей від доменної БД (Events/Bookings) та спрощує супровід міграцій.

Лістинг класу AppIdentityDbContex:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace TicketBooking.Data.Contexts {
	public class AppIdentityDbContext : IdentityDbContext<IdentityUser> {
		public AppIdentityDbContext(DbContextOptions<AppIdentityDbContext> options)
			: base(options) { }
	}
}
```

Далі у appsettings.json додано додатковий рядок підключення IdentityConnection до окремої бази даних для зберігання облікових записів.

Лістинг фрагменту коду appsettings.json:

```
"ConnectionStrings": {
  "TicketBookingConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=TicketBookingDb;MultipleActiveResultSets=true",
  "IdentityConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=TicketBookingIdentity;MultipleActiveResultSets=true"
}
```

У файлі Program.cs виконано конфігурацію Identity. А саме: підключено AppIdentityDbContext через AddDbContext і UseSqlServer, додано AddIdentity з налаштуванням політик безпеки, підключено session і middleware аутентифікації/авторизації.

Окремо встановлено вимоги відповідно до завдання:

- унікальність email (RequireUniqueEmail = true);
- вимоги до пароля: мінімум 8 символів, наявність цифр, літер, хоча б однієї великої літери;
- реалізовано поле ConfirmPassword у формі реєстрації.

Лістинг фрагменту коду Program:

```
builder.Services.AddIdentity<IdentityUser, IdentityRole>(options => {
       options.User.RequireUniqueEmail = true;
       options.Password.RequiredLength = 8;
       options.Password.RequireDigit = true;
       options.Password.RequireLowercase = true;
       options.Password.RequireUppercase = true;
       options.Password.RequireNonAlphanumeric = false;
})
       .AddEntityFrameworkStores<AppIdentityDbContext>()
       .AddDefaultTokenProviders();

var app = builder.Build();

app.UseAuthentication();
app.UseAuthorization();
```

Після налаштувань створено міграцію для Identity-контексту та застосовано її до БД. У результаті з’явилися таблиці користувачів/ролей (AspNetUsers, AspNetRoles, AspNetUserRoles тощо).

Одразу після цього абзацу вставити скріншот з консолі з командами:

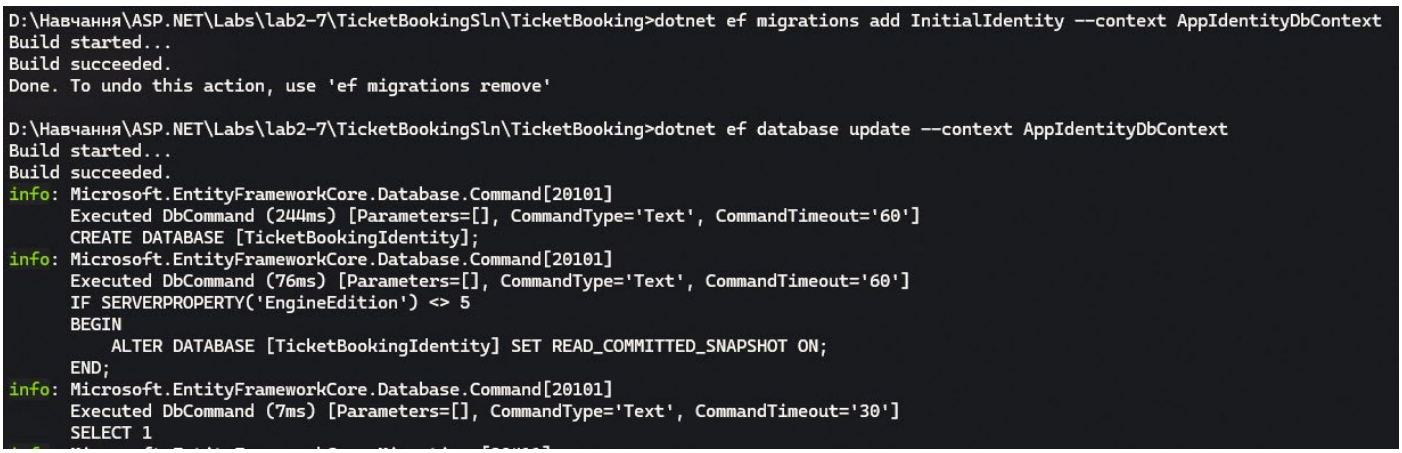

Рис. 1.1. Створення та застосування міграції для Identity-бази даних.

Далі розроблено контролер AccountController, який забезпечує реєстрацію (Register з методами GET/POST), логін (Login з методами GET/POST), вилогінення (Logout), особистий кабінет (Profile).

У формах використано server-side валідацію через DataAnnotations і перевірку ModelState.IsValid. Після успішної реєстрації користувач автоматично входить у систему.

Лістинг контроллеру AccountController:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using TicketBooking.Models.ViewModels;

namespace TicketBooking.Controllers {
	public class AccountController : Controller {

		private readonly UserManager<IdentityUser> _userManager;
		private readonly SignInManager<IdentityUser> _signInManager;

		public AccountController(UserManager<IdentityUser> userMgr,
SignInManager<IdentityUser> signInMgr) {
			_userManager = userMgr;
			_signInManager = signInMgr;
		}

		[HttpGet]
		public IActionResult Register() {
			return View(new RegisterModel());
		}
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Register(RegisterModel model) {
            if (!ModelState.IsValid)
                return View(model);

            var user = new IdentityUser {
                UserName = model.Email,
                Email = model.Email
            };

            var result = await _userManager.CreateAsync(user, model.Password);

            if (result.Succeeded) {
                await _signInManager.SignInAsync(user, isPersistent: false);
                return RedirectToAction("Index", "Home");
            }

            foreach (var err in result.Errors)
                ModelState.AddModelError("", err.Description);

            return View(model);
        }

        [HttpGet]
        public IActionResult Login(string? returnUrl = null) {
            return View(new LoginModel { ReturnUrl = returnUrl });
        }

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

            var user = await _userManager.FindByEmailAsync(model.Email);
            if (user != null) {
                await _signInManager.SignOutAsync();
                var result = await _signInManager.PasswordSignInAsync(user,
model.Password, isPersistent: false, lockoutOnFailure: false);
                if (result.Succeeded)
                    return Redirect(model.ReturnUrl ?? "/");
            }

            ModelState.AddModelError("", "Invalid email or password");
            return View(model);
        }

        public async Task<IActionResult> Logout(string returnUrl = "/") {
            await _signInManager.SignOutAsync();
            return Redirect(returnUrl);
        }

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

            return View(new ProfileModel { Email = user.Email ?? "" });
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Profile(ProfileModel model) {
```

```
                if (!ModelState.IsValid)
                        return View(model);

                var user = await _userManager.GetUserAsync(User);
                if (user == null)
                        return RedirectToAction(nameof(Login));

                user.Email = model.Email;
                user.UserName = model.Email;

                var result = await _userManager.UpdateAsync(user);

                if (result.Succeeded) {
                        ViewBag.Success = "Profile updated";
                        return View(model);
                }

                foreach (var err in result.Errors)
                        ModelState.AddModelError("", err.Description);

                return View(model);
        }

        public IActionResult AccessDenied() => View();
    }
}
```

Для UI створено сторінки Views/Account/Register.cshtml, Login.cshtml, Profile.cshtml. У формах реалізовано показ помилок валідації (asp-validation-summary і asp-validation-for).

Лістинг коду Register.cshtml:

```
@model TicketBooking.Models.ViewModels.RegisterModel

<h2>Register</h2>

<div asp-validation-summary="All" class="text-danger"></div>

<form asp-action="Register" method="post">
    @Html.AntiForgeryToken()

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

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

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

```
    <button class="btn btn-primary" type="submit">Create account</button>
    <a class="btn btn-link" asp-action="Login">Already have an account?</a>
</form>

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

Лістинг коду Login.cshtml:

```
@model TicketBooking.Models.ViewModels.LoginModel

<h2>Login</h2>

<div asp-validation-summary="All" class="text-danger"></div>

<form asp-action="Login" method="post">
    @Html.AntiForgeryToken()
    <input type="hidden" asp-for="ReturnUrl" />

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

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

    <button class="btn btn-success" type="submit">Login</button>
    <a class="btn btn-link" asp-action="Register">Create account</a>
</form>

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

Лістинг коду Profile.cshtml:

```
@model TicketBooking.Models.ViewModels.ProfileModel

<h2>My Profile</h2>

@if (ViewBag.Success != null) {
    <div class="alert alert-success">@ViewBag.Success</div>
}

<div asp-validation-summary="All" class="text-danger"></div>

<form asp-action="Profile" method="post">
    @Html.AntiForgeryToken()

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

    <button class="btn btn-primary" type="submit">Save</button>
```

```
</form>

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

Для підтримки клієнтської валідації створено partial _ValidationScriptsPartial.cshtml у Views/Shared, а в _Layout.cshtml додано рендер секції Scripts через @await RenderSectionAsync("Scripts", required: false).

Лістинг коду _ValidationScriptsPartial.cshtml:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-
validate/1.20.0/jquery.validate.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-
unobtrusive/4.0.0/jquery.validate.unobtrusive.min.js"></script>
```

Контроль доступу реалізовано через [Authorize]: незалогінений користувач має доступ лише до Login та Register, інші дії застосунку доступні лише після автентифікації.

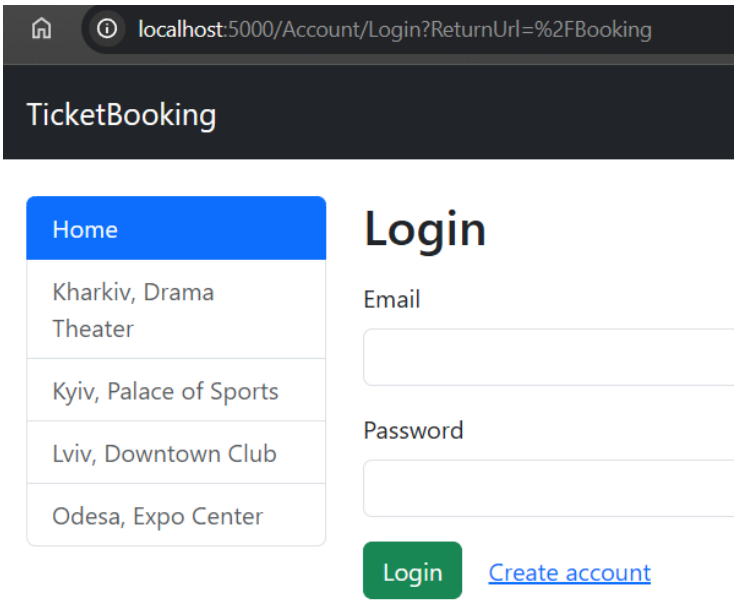

Рис. 1.2. Доступ до функціоналу застосунку обмежений для неавтентифікованих користувачів.

Додатково реалізовано ролі користувачів Admin і User та обмеження адмін-панелі. Кнопки адміністрування в _Layout.cshtml відображаються лише для користувача з

роллю Admin (User.IsInRole("Admin")). Також контролери адміністрування додатково захищені [Authorize(Roles = "Admin")].

Лістинг фрагменту коду _Layout.cshtml з умовним відображенням адмін-кнопок:

```
@if (User.Identity?.IsAuthenticated ?? false) {
    <a class="btn btn-outline-light m-1" asp-controller="Cart" asp-action="Index">
        Booking
    </a>
}
<div class="w-auto d-flex flex-row justify-content-end">
    @if (User.Identity?.IsAuthenticated ?? false) {
        <a class="btn btn-outline-light m-1" asp-controller="Account" asp-
action="Profile">Profile</a>
        <a class="btn btn-danger m-1" asp-controller="Account" asp-action="Logout">Logout</a>

        if (User.IsInRole("Admin")) {
            <div class="dropdown m-1">
                <button class="btn btn-warning dropdown-toggle" type="button" data-bs-
toggle="dropdown" aria-expanded="false">
                    Admin
                </button>

                <ul class="dropdown-menu dropdown-menu-end">
                    <li>
                        <a class="dropdown-item" asp-controller="Event" asp-action="Index">
                            Events
                        </a>
                    </li>
                    <li>
                        <a class="dropdown-item" asp-controller="Booking" asp-action="Index">
                            Bookings
                        </a>
                    </li>
                </ul>
            </div>
        }
    } else {
        <a class="btn btn-success m-1" asp-controller="Account" asp-action="Login">Login</a>
        <a class="btn btn-primary m-1" asp-controller="Account" asp-
action="Register">Register</a>
    }
</div>
```

Лістинг коду-прикладу [Authorize(Roles = "Admin")] у BookingController:

```
namespace TicketBooking.Controllers {
      public class BookingController : Controller {
            private readonly IBookingRepository _bookingRepo;
            private readonly IEventRepository _eventRepo;

            public BookingController(IBookingRepository bookingRepo, IEventRepository
eventRepo) {
                  _bookingRepo = bookingRepo;
                  _eventRepo = eventRepo;
            }

            public IActionResult Index() {
                  var data = _bookingRepo.Bookings.OrderByDescending(b =>
b.BookingId).ToList();
```

```
                return View(data);
        }

        public IActionResult Details(long id) {
                var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
                if (booking == null)
                        return NotFound();
                return View(booking);
        }

        [HttpGet]
        public IActionResult Create() {
                ViewBag.Events = new SelectList(_eventRepo.Events.ToList(), "EventId",
"Title");
                return View(new Booking());
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Create(Booking model) {
                if (!ModelState.IsValid) {
                        ViewBag.Events = new SelectList(_eventRepo.Events.ToList(),
"EventId", "Title", model.EventId);
                        return View(model);
                }

                _bookingRepo.CreateBooking(model);
                return RedirectToAction(nameof(Index));
        }

        [HttpGet]
        public IActionResult Edit(long id) {
                var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
                if (booking == null)
                        return NotFound();

                ViewBag.Events = new SelectList(_eventRepo.Events.ToList(), "EventId",
"Title", booking.EventId);
                return View(booking);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Edit(Booking model) {
                if (!ModelState.IsValid) {
                        ViewBag.Events = new SelectList(_eventRepo.Events.ToList(),
"EventId", "Title", model.EventId);
                        return View(model);
                }

                _bookingRepo.UpdateBooking(model);
                return RedirectToAction(nameof(Index));
        }

        [HttpGet]
        public IActionResult Delete(long id) {
                var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
                if (booking == null)
                        return NotFound();
                return View(booking);
        }
```

```
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(long id) {
            var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
            if (booking == null)
                return NotFound();

            _bookingRepo.DeleteBooking(booking);
            return RedirectToAction(nameof(Index));
        }
    }
}
```

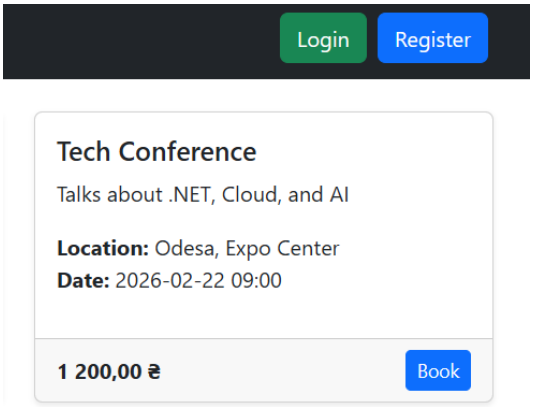

Рис. 1.3. Рольова модель доступу: Навігаційна панель користувача без логіну

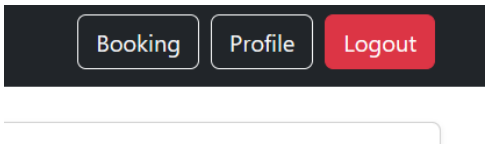

Рис. 1.4. Рольова модель доступу: Навігаційна панель звичайного залогованого користувача

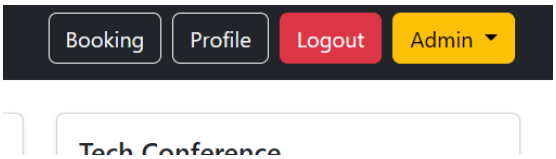

Рис. 1.5. Рольова модель доступу: Навігаційна панель адміністратора

Завдання 2. В рішенні створити новий WebAPI проект. Цей проект повинен реалізовувати всі функціональності, які наявні в MVC проекті у вигляді REST-ендпоїнтів. Для кращої організації і перевикористання коду слід перенести файли для роботи з БД

(репозиторії, контексти, моделі) в окрему бібліотеку і додати посилання на неї в MVC та WebAPI проектах. Встановіть необхідні залежності для проекту бібліотеки.

Після цього налаштуйте ConnectionString'и для створених БД для WebAPI проекту в файлі appsettings.json. Таким чином, після перенесення файлів для роботи з БД в окрему бібліотеку ми можемо використовувати всі необхідні сервіси в обох проектах.

На цьому етапі було виконано реорганізацію рішення таким чином, щоб один і той самий доменний функціонал можна було використовувати у двох проєктах: MVC-застосунку (web-інтерфейс) і WebAPI (REST-доступ). Для цього у рішенні створено новий проєкт TicketBooking.Api на базі шаблону ASP.NET Web API. Паралельно створено бібліотеку класів TicketBooking.Data, яка стала спільним шаром доступу до даних. Основна ідея такого підходу полягає в тому, що моделі, контексти EF Core та репозиторії є «ядром» роботи з БД і не повинні дублюватися у різних проєктах. Натомість MVC і API підключають одну бібліотеку й отримують однакову поведінку при читанні/записі даних.

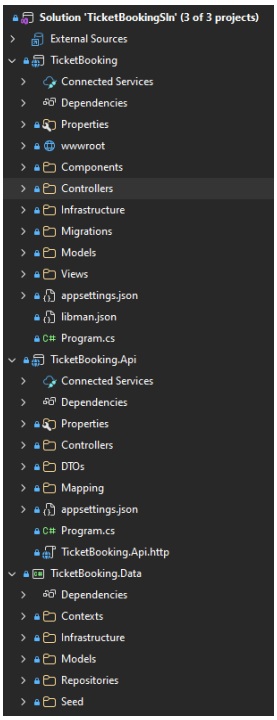

Рис. 2.1. Структура рішення TicketBooking після додавання WebAPI та виділення бібліотеки TicketBooking.Data.

У бібліотеку TicketBooking.Data було перенесено доменні моделі Event і Booking з уже реалізованими правилами валідації (DataAnnotations), а також контексти TicketBookingDbContext і AppIdentityDbContext. Додатково перенесено репозиторії для обох сутностей, які містять CRUD-операції (Create/Update/Delete) і використовуються як єдиний спосіб доступу до даних у двох проєктах. Після перенесення також було налаштовано посилання (project reference) з MVC та WebAPI на бібліотеку даних, що дозволило повторно використовувати весь код доступу до БД без копіювання.

Після цього абзацу вставити лістинг коду: фрагменти Event.cs та Booking.cs (з бібліотеки TicketBooking.Data, щоб показати наявність навігаційних властивостей і валідації).

Оскільки WebAPI є окремим застосунком, для нього окремо налаштовано конфігураційний файл appsettings.json з рядками підключення до доменної БД та до Identity-бази. Це важливо, тому що WebAPI повинно мати можливість працювати з тією ж інформацією, що й MVC, а також створювати/валідувати користувачів у межах Identity. У Program.cs WebAPI підключено контексти EF Core через AddDbContext(...), зареєстровано репозиторії через AddScoped(...), і налаштовано Swagger для тестування ендпоїнтів.

Лістинг фрагменту коду TicketBooking.Api/appsettings.json:

```
"ConnectionStrings": {
  "TicketBookingConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=TicketBookingDb;MultipleActiveResultSets=true",
  "IdentityConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=TicketBookingIdentity;MultipleActiveResultSets=true"
}
```

Лістинг коду TicketBooking.Api/Program.cs:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi;
using System.Text;
using TicketBooking.Data.Contexts;
using TicketBooking.Data.Repositories;
```

```
var builder = WebApplication.CreateBuilder(args);


builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<TicketBookingDbContext>(opts => {
	opts.UseSqlServer(
		builder.Configuration["ConnectionStrings:TicketBookingConnection"]
	);
});

builder.Services.AddDbContext<AppIdentityDbContext>(opts => {
	opts.UseSqlServer(builder.Configuration["ConnectionStrings:IdentityConnection"]);
});

builder.Services.AddIdentity<IdentityUser, IdentityRole>(options => {
	options.User.RequireUniqueEmail = true;
	options.Password.RequiredLength = 8;
	options.Password.RequireDigit = true;
	options.Password.RequireLowercase = true;
	options.Password.RequireUppercase = true;
	options.Password.RequireNonAlphanumeric = false;
})
.AddEntityFrameworkStores<AppIdentityDbContext>()
.AddDefaultTokenProviders();

builder.Services.AddScoped<IEventRepository, EFEventRepository>();
builder.Services.AddScoped<IBookingRepository, EFBookingRepository>();

var app = builder.Build();

app.UseSwagger();
app.UseSwaggerUI();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

Після базової конфігурації було реалізовано REST-контролери для сутностей подій та бронювань. Контролер EventsController надає ендпоїнти для отримання списку подій і одиничної події за ідентифікатором, а також ендпоїнти для створення, редагування та видалення записів. Аналогічний підхід застосовано для BookingsController. Таким чином WebAPI віддзеркалює основні можливості, що є в MVC (CRUD) у вигляді HTTP-ендпоїнтів, які можуть викликатися будь-яким клієнтом (Postman, Swagger, фронтенд тощо).

Лістинг контроллера TicketBooking.Api/Controllers/EventsController.cs:

```
using Microsoft.AspNetCore.Authorization;
```

```
using Microsoft.AspNetCore.Mvc;
using TicketBooking.Data.Models;
using TicketBooking.Data.Repositories;

namespace TicketBooking.Api.Controllers {
	[ApiController]
	[Route("api/[controller]")]
	public class EventsController : ControllerBase {
		private readonly IEventRepository _repo;

		public EventsController(IEventRepository repo) => _repo = repo;

		[HttpGet]
		public IActionResult GetAll()
			=> Ok(_repo.Events.OrderBy(e => e.EventId).ToList());

		[HttpGet("{id:long}")]
		public IActionResult Get(long id) {
			var ev = _repo.Events.FirstOrDefault(x => x.EventId == id);
			return ev == null ? NotFound() : Ok(ev);
		}

		[HttpPost]
		public IActionResult Create([FromBody] Event model) {
			if (!ModelState.IsValid)
				return BadRequest(ModelState);
			_repo.CreateEvent(model);
			return CreatedAtAction(nameof(Get), new { id = model.EventId }, model);
		}

		[HttpPut("{id:long}")]
		public IActionResult Update(long id, [FromBody] Event model) {
			if (id != model.EventId)
				return BadRequest("Id mismatch");
			if (!ModelState.IsValid)
				return BadRequest(ModelState);

			_repo.UpdateEvent(model);
			return NoContent();
		}

		[HttpDelete("{id:long}")]
		public IActionResult Delete(long id) {
			var ev = _repo.Events.FirstOrDefault(x => x.EventId == id);
			if (ev == null)
				return NotFound();

			_repo.DeleteEvent(ev);
			return NoContent();
		}
	}
}
```

Лістинг контроллера TicketBooking.Api/Controllers/BookingsController:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TicketBooking.Data.Models;
using TicketBooking.Data.Repositories;

namespace TicketBooking.Api.Controllers {
	[ApiController]
	[Route("api/[controller]")]
	public class BookingsController : ControllerBase {
```

```
        private readonly IBookingRepository _repo;

        public BookingsController(IBookingRepository repo) => _repo = repo;

        [HttpGet]
        public IActionResult GetAll()
            => Ok(_repo.Bookings.OrderBy(b => b.BookingId).ToList());

        [HttpGet("{id:long}")]
        public IActionResult Get(long id) {
            var b = _repo.Bookings.FirstOrDefault(x => x.BookingId == id);
            return b == null ? NotFound() : Ok(b);
        }

        [HttpPost]
        public IActionResult Create([FromBody] Booking model) {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            _repo.CreateBooking(model);
            return CreatedAtAction(nameof(Get), new { id = model.BookingId }, model);
        }

        [HttpPut("{id:long}")]
        public IActionResult Update(long id, [FromBody] Booking model) {
            if (id != model.BookingId)
                return BadRequest("Id mismatch");
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            _repo.UpdateBooking(model);
            return NoContent();
        }

        [HttpDelete("{id:long}")]
        public IActionResult Delete(long id) {
            var b = _repo.Bookings.FirstOrDefault(x => x.BookingId == id);
            if (b == null)
                return NotFound();

            _repo.DeleteBooking(b);
            return NoContent();
        }
    }
}
```

Окремо в межах цього завдання реалізовано реєстрацію користувача через WebAPI. Для цього створено endpoint POST /api/auth/register, який приймає дані користувача у форматі JSON, виконує серверну валідацію та створює запис у Identity-базі через UserManager. Такий endpoint потрібен, щоб клієнтські застосунки могли створювати облікові записи не лише через MVC-сторінки, але й через REST API.

Лістинг контроллера TicketBooking.Api/Controllers/AuthController:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System.ComponentModel.DataAnnotations;
```

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace TicketBooking.Api.Controllers {
	[ApiController]
	[Route("api/[controller]")]
	public class AuthController : ControllerBase {
		private readonly UserManager<IdentityUser> _userManager;
		private readonly IConfiguration _config;

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

		public class LoginRequest {
			[Required, EmailAddress]
			public string Email { get; set; } = string.Empty;

			[Required]
			public string Password { get; set; } = string.Empty;
		}

		[HttpPost("login")]
		public async Task<IActionResult> Login([FromBody] LoginRequest model) {
			if (!ModelState.IsValid)
				return BadRequest(ModelState);

			var user = await _userManager.FindByEmailAsync(model.Email);
			if (user == null)
				return Unauthorized("Invalid credentials");

			var valid = await _userManager.CheckPasswordAsync(user, model.Password);
			if (!valid)
				return Unauthorized("Invalid credentials");

			var claims = new List<Claim>
			{
				new Claim(JwtRegisteredClaimNames.Sub, user.Id),
				new Claim(JwtRegisteredClaimNames.Email, user.Email ?? ""),
				new Claim(ClaimTypes.Name, user.UserName ?? user.Email ?? ""),
				new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
			};

			var roles = await _userManager.GetRolesAsync(user);
			foreach (var r in roles)
				claims.Add(new Claim(ClaimTypes.Role, r));

			var jwtSection = _config.GetSection("Jwt");
			var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSection["Key"]!));

			var token = new JwtSecurityToken(
				issuer: jwtSection["Issuer"],
				audience: jwtSection["Audience"],
				claims: claims,
				expires: DateTime.UtcNow.AddMinutes(60),
				signingCredentials: new SigningCredentials(key,
SecurityAlgorithms.HmacSha256)
			);

			return Ok(new {
				token = new JwtSecurityTokenHandler().WriteToken(token),
				expires = token.ValidTo
```

```
                });
        }

        public class RegisterRequest {
                [Required, EmailAddress]
                public string Email { get; set; } = string.Empty;

                [Required]
                public string Password { get; set; } = string.Empty;

                [Required]
                [Compare(nameof(Password))]
                public string ConfirmPassword { get; set; } = string.Empty;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] RegisterRequest model) {
                if (!ModelState.IsValid)
                        return BadRequest(ModelState);

                var user = new IdentityUser {
                        UserName = model.Email,
                        Email = model.Email
                };

                var result = await _userManager.CreateAsync(user, model.Password);

                if (!result.Succeeded) {
                        return BadRequest(new {
                                errors = result.Errors.Select(e => e.Description).ToArray()
                        });
                }

                return Ok(new { message = "User created", userId = user.Id });
        }
    }
}
```

Під час тестування ендпоїнта GET /api/bookings було виявлено помилку 500 із повідомленням A possible object cycle was detected. Причина полягала у тому, що моделі Event і Booking мають двосторонні навігаційні властивості: бронювання містить посилання на подію (Booking.Event), а подія містить колекцію бронювань (Event.Bookings). Коли WebAPI намагається серіалізувати Booking у JSON разом із включеною подією (Include(b => b.Event)), серіалізатор «бачить» повний цикл Booking -> Event -> Bookings -> Event -> ... і зупиняє роботу, щоб уникнути нескінченного обходу.

Щоб WebAPI повертало стабільні й прогнозовані відповіді, реалізацію було вдосконалено за рахунок DTO (Data Transfer Objects). Замість повернення EF-entities напряму з контролерів, сервер формує «пласкі» об’єкти відповіді, які містять тільки потрібні поля. Наприклад, для бронювання у відповіді залишено дані клієнта, кількість,

EventId та кілька полів події (назва/дата/локація), але без вкладеної колекції Event.Bookings. У результаті цикл зникає, а контракт API стає зрозумілим і керованим.

Лістинг коду TicketBooking.Api/DTOs/ BookingDto:

```
namespace TicketBooking.Api.DTOs {
	public class BookingDto {
		public long BookingId { get; set; }
		public string CustomerName { get; set; } = "";
		public string CustomerEmail { get; set; } = "";
		public int Quantity { get; set; }
		public long EventId { get; set; }
		public DateTime CreatedAtUtc { get; set; }

		public string EventTitle { get; set; } = "";
		public DateTime EventStartDate { get; set; }
		public string EventLocation { get; set; } = "";
	}
}
```

Лістинг коду TicketBooking.Api/DTOs/BookingCreateUpdateDto:

```
using System.ComponentModel.DataAnnotations;

namespace TicketBooking.Api.DTOs {
	public class BookingCreateUpdateDto {
		[Required, StringLength(100)]
		public string CustomerName { get; set; } = "";

		[Required, EmailAddress, StringLength(120)]
		public string CustomerEmail { get; set; } = "";

		[Range(1, 10)]
		public int Quantity { get; set; } = 1;

		[Range(1, long.MaxValue, ErrorMessage = "Please select an event")]
		public long EventId { get; set; }
	}
}
```

Лістинг коду TicketBooking.Api/DTOs/EventDto:

```
namespace TicketBooking.Api.DTOs {
	public class EventDto {
		public long EventId { get; set; }
		public string Title { get; set; } = "";
		public string Description { get; set; } = "";
		public string Location { get; set; } = "";
		public DateTime StartDate { get; set; }
		public decimal BaseTicketPrice { get; set; }

		public int BookingsCount { get; set; }
	}
}
```

Лістинг коду TicketBooking.Api/DTOs/EventCreateUpdateDto.cs:

```
using System.ComponentModel.DataAnnotations;
```

```
namespace TicketBooking.Api.DTOs {
	public class EventCreateUpdateDto {
		[Required, StringLength(120)]
		public string Title { get; set; } = "";

		[Required, StringLength(1000)]
		public string Description { get; set; } = "";

		[Required, StringLength(80)]
		public string Location { get; set; } = "";

		[Required]
		public DateTime StartDate { get; set; }

		[Range(0.01, 10000000)]
		public decimal BaseTicketPrice { get; set; }
	}
}
```

Щоб мапінг між моделями й DTO не дублювався у різних контролерах, створено окремий файл з методами перетворення (extension methods). Це підвищує читабельність коду і спрощує подальші зміни формату відповіді, оскільки всі правила мапінгу зібрані в одному місці.

Лістинг коду TicketBooking.Api/Mapping/DtoMapping.cs:

```
using TicketBooking.Api.DTOs;
using TicketBooking.Data.Models;

namespace TicketBooking.Api.Mapping {
	public static class DtoMapping {
		public static EventDto ToDto(this Event ev) => new() {
			EventId = ev.EventId ?? 0,
			Title = ev.Title,
			Description = ev.Description,
			Location = ev.Location,
			StartDate = ev.StartDate,
			BaseTicketPrice = ev.BaseTicketPrice,
			BookingsCount = ev.Bookings?.Count ?? 0
		};

		public static BookingDto ToDto(this Booking booking) => new() {
			BookingId = booking.BookingId,
			CustomerName = booking.CustomerName,
			CustomerEmail = booking.CustomerEmail,
			Quantity = booking.Quantity,
			EventId = booking.EventId,
			CreatedAtUtc = booking.CreatedAtUtc,
			EventTitle = booking.Event?.Title ?? "",
			EventStartDate = booking.Event?.StartDate ?? default,
			EventLocation = booking.Event?.Location ?? ""
		};
	}
}
```

Після переходу на DTO запит GET /api/bookings почав коректно повертати дані, а також стало можливим безпечно тестувати інші CRUD-операції без ризику отримати помилки серіалізації.

Також працездатність WebAPI перевірено через Swagger або Postman: виконано запити на читання списків і одиничних записів, створення нових подій/бронювань та перевірено роботу реєстрації користувача через API.

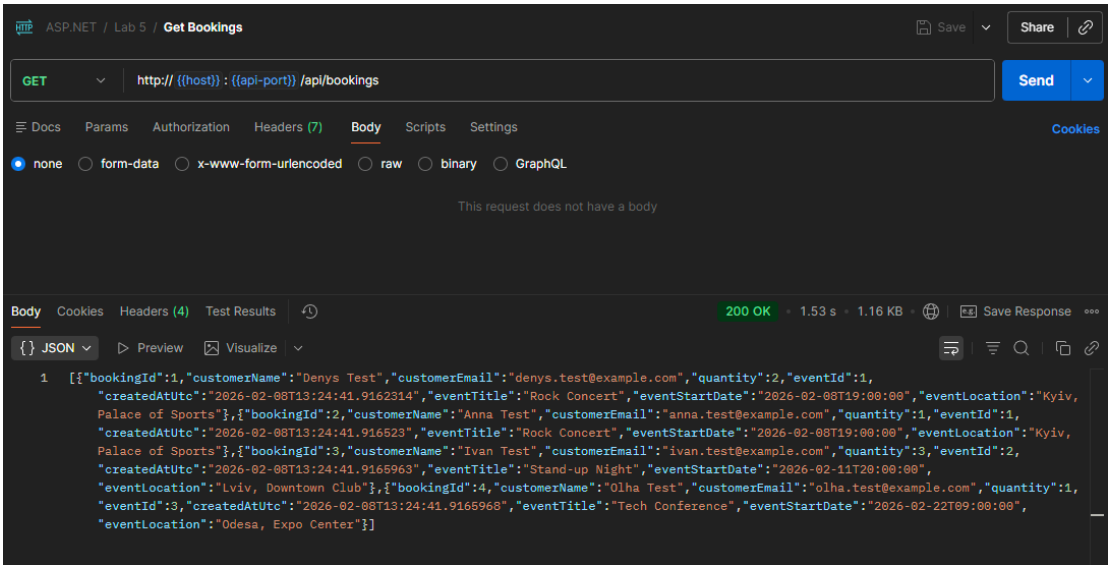

Рис. 2.2.Успішний запит GET /api/bookings.

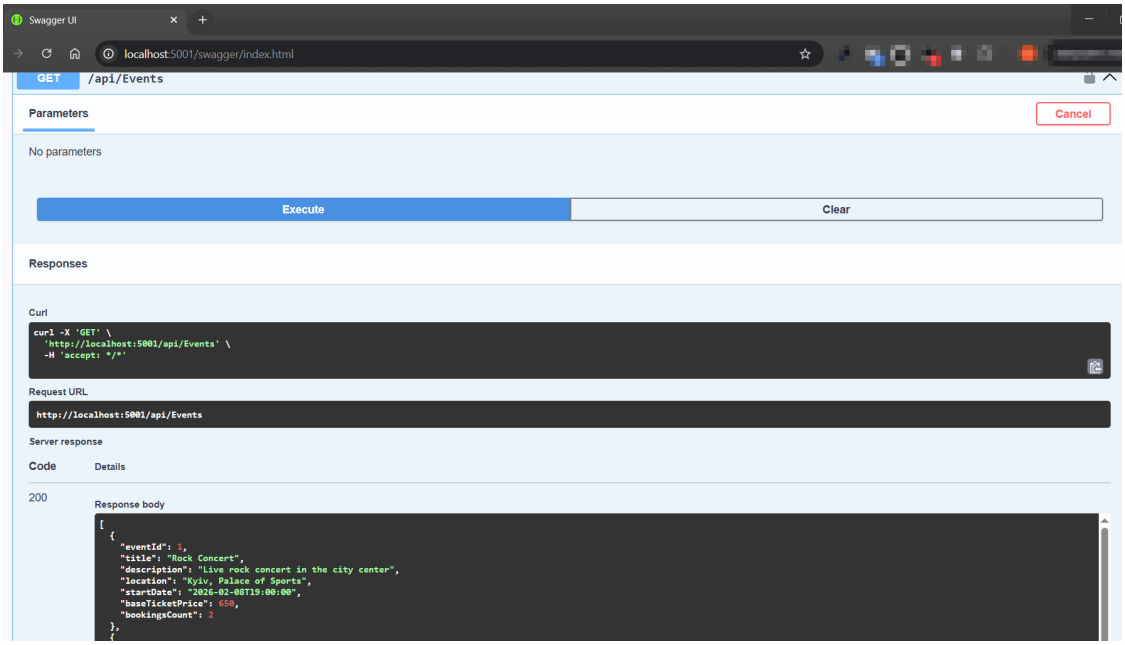

Рис. 2.3. Перевірка REST ендпоїнта GET /api/events у Swagger.

Завдання 3. Реалізуйте автентифікацію та авторизацію користувача з використанням механізму JSON Web Token (JWT). Вона повинна працювати наступним чином:

1. Клієнт відправляє запит на login-ендпоїнт зі своїм логіном та паролем.
2. Якщо користувач з таким логіном та паролем існує, тоді сервер повинен повернути згенерований JSON Web Token.
3. Для отримання доступу до дій, які вимагають авторизації, клієнт відправляє в запиті заголовок Authorization зі значенням Bearer TokenValue, де TokenValue – токен, який було згенеровано під час автентифікації.
4. Якщо токен валідний – сервер виконає запит, інакше поверне помилку 401 Unauthorized.

У межах третього завдання до WebAPI додано автентифікацію та авторизацію на основі JWT (JSON Web Token). Такий підхід застосовується тоді, коли клієнт не використовує cookie-сесію браузера (як у MVC), а взаємодіє з сервером через REST-запити. Логіка роботи JWT у проєкті реалізована за типовою схемою: клієнт надсилає логін і пароль на спеціальний endpoint, сервер перевіряє облікові дані через Identity, генерує токен та повертає його у відповіді. Далі клієнт додає токен у заголовок Authorization зі значенням Bearer <token>, і сервер дозволяє виконання лише тих запитів, які мають валідний токен. Якщо токен відсутній або не проходить перевірку, сервер повертає 401 Unauthorized.

Для підключення JWT у WebAPI встановлено пакет Microsoft.AspNetCore.Authentication.JwtBearer. У файлі appsettings.json додано секцію Jwt з параметрами Issuer, Audience та Key. Ці параметри використовуються як під час генерації токена, так і під час його валідації.

Лістинг фрагменту коду TicketBooking.Api/appsettings.json:

```
"Jwt": {
  "Issuer": "TicketBooking.Api",
  "Audience": "TicketBooking.Client",
  "Key": "df4eb1fee05101e439e2406b2dd286d4"
}
```

Далі у Program.cs налаштовано AddAuthentication().AddJwtBearer(...) із правилами валідації токена: перевірка видавця (issuer), аудиторії (audience), підпису (signing key) та часу життя (lifetime). У пайплайні додано UseAuthentication() перед UseAuthorization(), що гарантує коректну перевірку токена для кожного запиту.

Лістинг фрагменту коду TicketBooking.Api/Program:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi;
using System.Text;
using TicketBooking.Data.Contexts;
using TicketBooking.Data.Repositories;

var builder = WebApplication.CreateBuilder(args);


builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(sgOption => {
	sgOption.SwaggerDoc("v1", new OpenApiInfo { Title = "TicketBooking API", Version = "v1"
});

	sgOption.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme {
		Name = "Authorization",
		Type = SecuritySchemeType.Http,
		Scheme = "bearer",
		BearerFormat = "JWT",
		In = ParameterLocation.Header,
		Description = "Enter: Bearer {your JWT token}"
	});

	sgOption.AddSecurityRequirement(document => new OpenApiSecurityRequirement
	{
		[new OpenApiSecuritySchemeReference("Bearer", document)] = []
	});
});

builder.Services.AddDbContext<TicketBookingDbContext>(opts => {
	opts.UseSqlServer(
		builder.Configuration["ConnectionStrings:TicketBookingConnection"]
	);
});

builder.Services.AddDbContext<AppIdentityDbContext>(opts => {
	opts.UseSqlServer(builder.Configuration["ConnectionStrings:IdentityConnection"]);
});

builder.Services.AddIdentity<IdentityUser, IdentityRole>(options => {
	options.User.RequireUniqueEmail = true;
	options.Password.RequiredLength = 8;
	options.Password.RequireDigit = true;
	options.Password.RequireLowercase = true;
	options.Password.RequireUppercase = true;
	options.Password.RequireNonAlphanumeric = false;
})
.AddEntityFrameworkStores<AppIdentityDbContext>()
```

```
.AddDefaultTokenProviders();

builder.Services
	.AddAuthentication(options => {
		options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
		options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
	})
	.AddJwtBearer(options => {
		var jwt = builder.Configuration.GetSection("Jwt");
		options.TokenValidationParameters = new TokenValidationParameters {
			ValidateIssuer = true,
			ValidIssuer = jwt["Issuer"],

			ValidateAudience = true,
			ValidAudience = jwt["Audience"],

			ValidateIssuerSigningKey = true,
			IssuerSigningKey = new SymmetricSecurityKey(
				Encoding.UTF8.GetBytes(jwt["Key"]!)),

			ValidateLifetime = true,
			ClockSkew = TimeSpan.FromSeconds(30)
		};
	});

builder.Services.AddAuthorization();

builder.Services.AddScoped<IEventRepository, EFEventRepository>();
builder.Services.AddScoped<IBookingRepository, EFBookingRepository>();

var app = builder.Build();

app.UseSwagger();
app.UseSwaggerUI();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

У контролері AuthController реалізовано endpoint POST /api/auth/login, який перевіряє користувача в Identity через UserManager і у випадку успішного логіну формує токен. До токена додано базові claims (ідентифікатор користувача, email, ім'я), а також ролі (якщо в проєкті використовується рольова модель). Завдяки цьому WebAPI може відрізняти звичайних користувачів від адміністраторів і надавати різні права доступу.

Лістинг контроллеру TicketBooking.Api/Controllers/AuthController:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System.ComponentModel.DataAnnotations;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
```

```
namespace TicketBooking.Api.Controllers {
	[ApiController]
	[Route("api/[controller]")]
	public class AuthController : ControllerBase {
		private readonly UserManager<IdentityUser> _userManager;
		private readonly IConfiguration _config;

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

		public class LoginRequest {
			[Required, EmailAddress]
			public string Email { get; set; } = string.Empty;

			[Required]
			public string Password { get; set; } = string.Empty;
		}

		[HttpPost("login")]
		public async Task<IActionResult> Login([FromBody] LoginRequest model) {
			if (!ModelState.IsValid)
				return BadRequest(ModelState);

			var user = await _userManager.FindByEmailAsync(model.Email);
			if (user == null)
				return Unauthorized("Invalid credentials");

			var valid = await _userManager.CheckPasswordAsync(user, model.Password);
			if (!valid)
				return Unauthorized("Invalid credentials");

			var claims = new List<Claim>
			{
				new Claim(JwtRegisteredClaimNames.Sub, user.Id),
				new Claim(JwtRegisteredClaimNames.Email, user.Email ?? ""),
				new Claim(ClaimTypes.Name, user.UserName ?? user.Email ?? ""),
				new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
			};

			var roles = await _userManager.GetRolesAsync(user);
			foreach (var r in roles)
				claims.Add(new Claim(ClaimTypes.Role, r));

			var jwtSection = _config.GetSection("Jwt");
			var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSection["Key"]!));

			var token = new JwtSecurityToken(
				issuer: jwtSection["Issuer"],
				audience: jwtSection["Audience"],
				claims: claims,
				expires: DateTime.UtcNow.AddMinutes(60),
				signingCredentials: new SigningCredentials(key,
SecurityAlgorithms.HmacSha256)
			);

			return Ok(new {
				token = new JwtSecurityTokenHandler().WriteToken(token),
				expires = token.ValidTo
			});
		}
```

```
        public class RegisterRequest {
            [Required, EmailAddress]
            public string Email { get; set; } = string.Empty;

            [Required]
            public string Password { get; set; } = string.Empty;

            [Required]
            [Compare(nameof(Password))]
            public string ConfirmPassword { get; set; } = string.Empty;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] RegisterRequest model) {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            var user = new IdentityUser {
                UserName = model.Email,
                Email = model.Email
            };

            var result = await _userManager.CreateAsync(user, model.Password);

            if (!result.Succeeded) {
                return BadRequest(new {
                    errors = result.Errors.Select(e => e.Description).ToArray()
                });
            }

            return Ok(new { message = "User created", userId = user.Id });
        }
    }
}
```

Для обмеження доступу до ендпоїнтів використано атрибут [Authorize]. У результаті без токена сервер повертає 401, а з коректним токеном запит виконується успішно. Для адмін-операцій (наприклад, керування подіями) може використовуватись [Authorize(Roles = "Admin")], що дозволяє поєднати JWT з рольовою моделлю.

Лістинг-приклад коду:

```
namespace TicketBooking.Api.Controllers {
    [ApiController]
    [Route("api/[controller]")]
    public class BookingsController : ControllerBase {
        private readonly IBookingRepository _repo;
        private readonly IEventRepository _eventRepo;

        public BookingsController(IBookingRepository repo, IEventRepository eventRepo) {
            _repo = repo;
            _eventRepo = eventRepo;
        }

        [HttpGet]
        public IActionResult GetAll() {
            var bookings = _repo.Bookings
                .OrderBy(booking => booking.BookingId)
```

```
                        .Select(booking => new BookingDto {
                                BookingId = booking.BookingId,
                                CustomerName = booking.CustomerName,
                                CustomerEmail = booking.CustomerEmail,
                                Quantity = booking.Quantity,
                                EventId = booking.EventId,
                                CreatedAtUtc = booking.CreatedAtUtc,
                                EventTitle = booking.Event!.Title,
                                EventStartDate = booking.Event!.StartDate,
                                EventLocation = booking.Event!.Location
                        })
                        .ToList();

                return Ok(bookings);
        }

        [HttpGet("{id:long}")]
        public IActionResult Get(long id) {
                var booking = _repo.Bookings.FirstOrDefault(item => item.BookingId == id);
                return booking == null ? NotFound() : Ok(booking.ToDto());
        }

        [Authorize(Roles = "Admin")]
        [HttpPost]
        public IActionResult Create([FromBody] BookingCreateUpdateDto dto) {
                if (!ModelState.IsValid)
                        return BadRequest(ModelState);

                var ev = _eventRepo.Events.FirstOrDefault(item => (item.EventId ?? 0) ==
dto.EventId);
                if (ev == null)
                        return BadRequest("Event does not exist");

                var booking = new Booking {
                        CustomerName = dto.CustomerName,
                        CustomerEmail = dto.CustomerEmail,
                        Quantity = dto.Quantity,
                        EventId = dto.EventId,
                        CreatedAtUtc = DateTime.UtcNow
                };

                _repo.CreateBooking(booking);

                var created = _repo.Bookings.First(b => b.BookingId == booking.BookingId);
                return CreatedAtAction(nameof(Get), new { id = booking.BookingId },
created.ToDto());
        }

        [Authorize(Roles = "Admin")]
        [HttpPut("{id:long}")]
        public IActionResult Update(long id, [FromBody] BookingCreateUpdateDto dto) {
                if (!ModelState.IsValid)
                        return BadRequest(ModelState);

                var booking = _repo.Bookings.FirstOrDefault(b => b.BookingId == id);
                if (booking == null)
                        return NotFound();

                var ev = _eventRepo.Events.FirstOrDefault(e => (e.EventId ?? 0) ==
dto.EventId);
                if (ev == null)
                        return BadRequest("Event does not exist");

                booking.CustomerName = dto.CustomerName;
                booking.CustomerEmail = dto.CustomerEmail;
                booking.Quantity = dto.Quantity;
```

```
            booking.EventId = dto.EventId;

            _repo.UpdateBooking(booking);
            return NoContent();
        }

        [Authorize(Roles = "Admin")]
        [HttpDelete("{id:long}")]
        public IActionResult Delete(long id) {
            var booking = _repo.Bookings.FirstOrDefault(item => item.BookingId == id);
            if (booking == null)
                return NotFound();

            _repo.DeleteBooking(booking);
            return NoContent();
        }
    }
}
```

Перевірка реалізації виконувалась у Swagger: спочатку було виконано логін для отримання токена, далі токен додано через кнопку Authorize, після чого протестовано доступ до захищених endpoint'ів. Додатково перевірено негативний сценарій — виклик без токена, який повинен завершитися помилкою 401 Unauthorized.

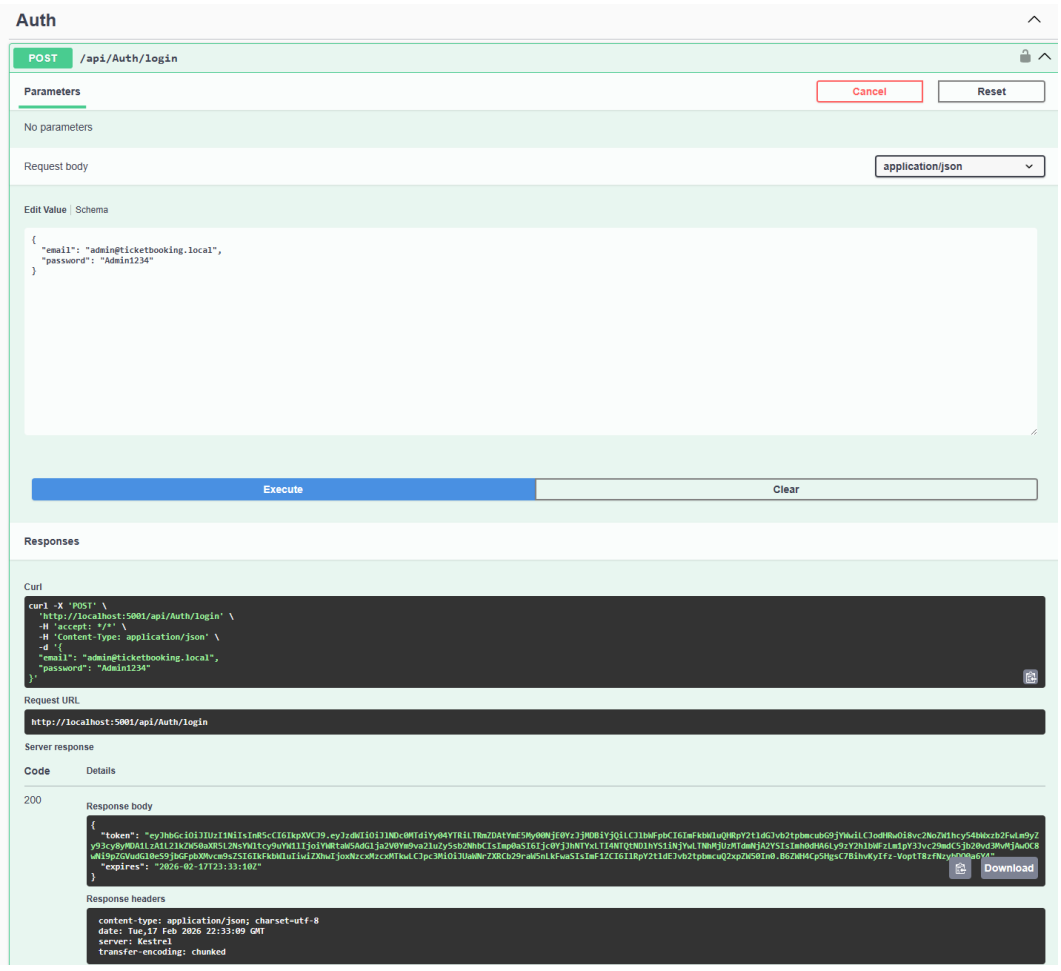

Рис. 3.1. Отримання JWT через POST /api/auth/login.

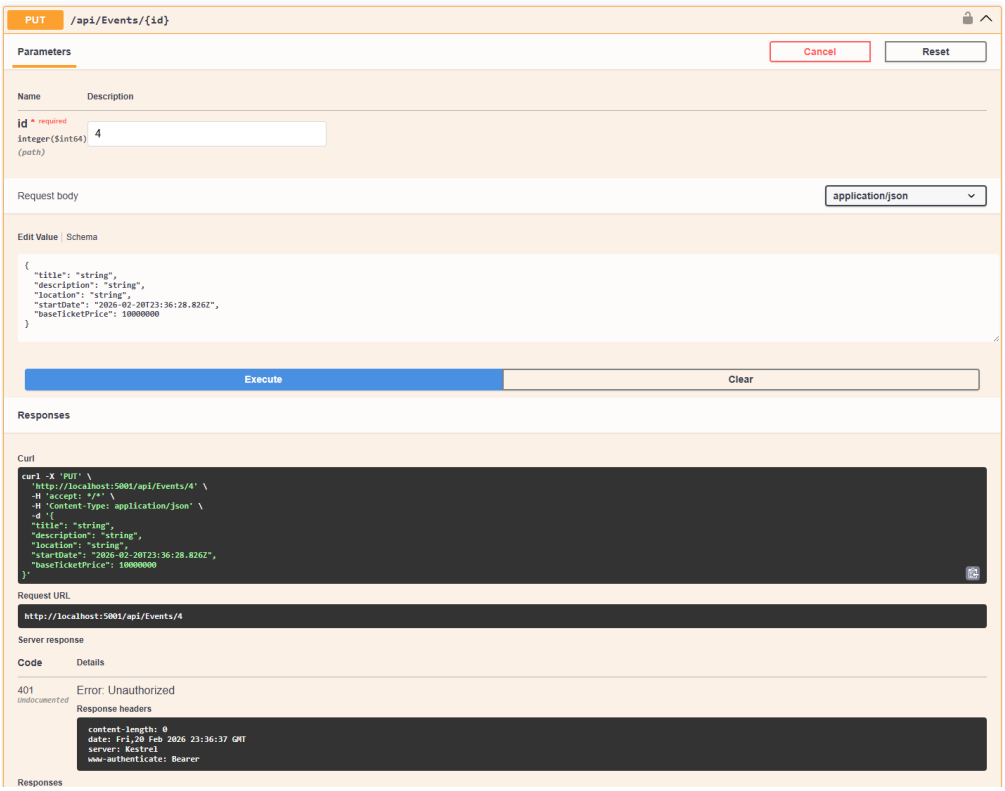

Рис. 3.2. 401 Unauthorized при відсутності Bearer-токена.

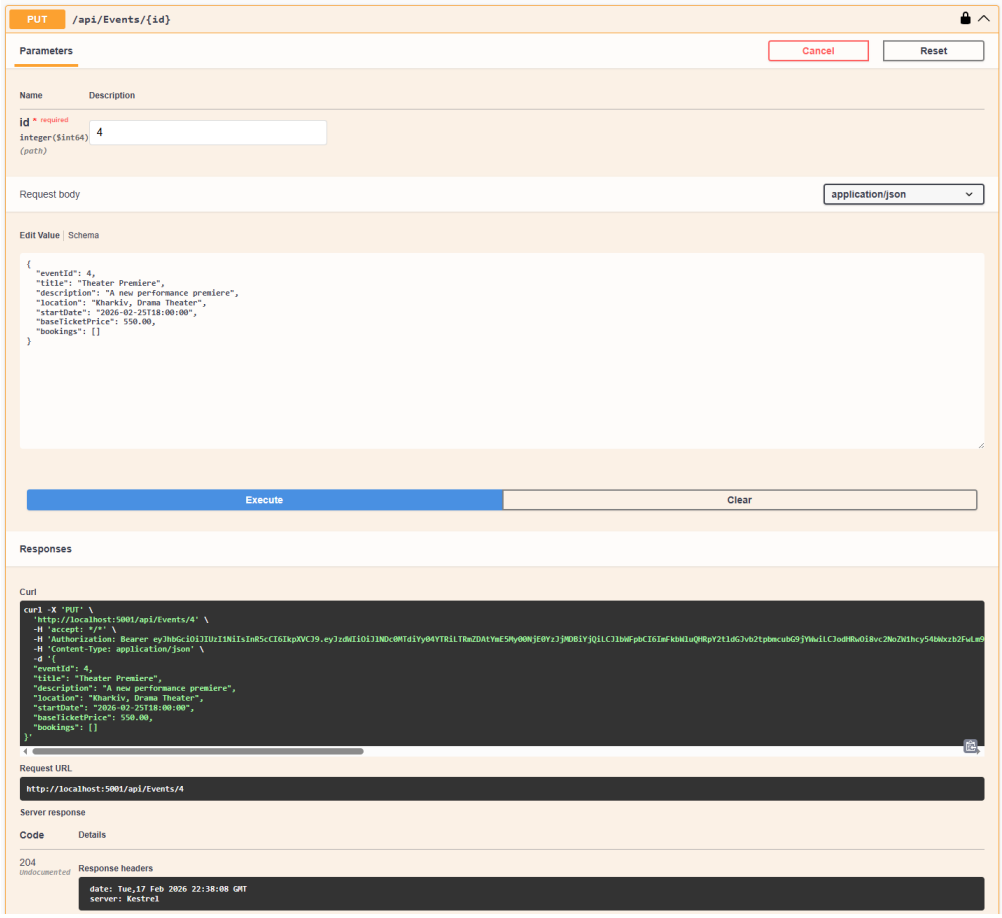

Рис. 3.2. Успішний доступ до захищеного endpoint'а з JWT Bearer Token.

Висновок: У лабораторній роботі №5 було розширено проєкт TicketBooking механізмами безпеки та API-взаємодії. У MVC-застосунку інтегровано Microsoft Identity і реалізовано повний набір базових сценаріїв роботи з користувачами: реєстрація, логін, логаут та особистий кабінет, а також контроль доступу через авторизацію і рольову модель. Для забезпечення інтеграції з зовнішніми клієнтами створено WebAPI-проєкт і винесено роботу з БД у спільну бібліотеку TicketBooking.Data, що дозволило уникнути дублювання моделей, контекстів та репозиторіїв.

Окремо у WebAPI реалізовано JWT-автентифікацію та авторизацію, що забезпечує захист ендпоїнтів через Bearer-токени. Під час тестування було виявлено й усунено проблему циклічної серіалізації даних, яка виникає через двосторонні зв’язки між Event і Booking. Перехід на DTO та централізований мапінг забезпечили стабільні JSON-відповіді та чіткий контракт API. Перевірка через Swagger/Postman підтвердила коректну роботу CRUD-ендпоїнтів, реєстрації користувача, логіну та механізму доступу до захищених ресурсів.