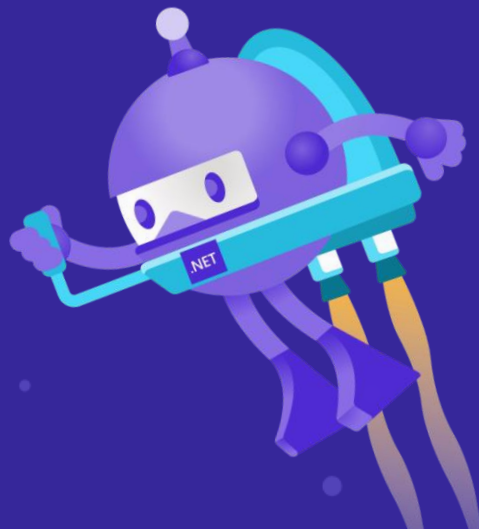
Day 17: Integration Testing, Documenting API, Publishing & Hosting



Integration Tests - Steps

- SUT's (System Under Test) web host is configured
- Test Server is created to accept requests
- Arrange test step is executed
  - Test app prepares the request
- Act test step is executed
  - client submits requests and response is received
- Assert step is executed
  - Actual response is validated as a pass or fail
- Process continues till all the cases are executed

Microsoft.AspNetCore.Mvc.Testing Package

- Copies the dependenices from the SUT to the test project's bin directory
- Sets the content root to the SUT projects' root
- Provides the WebApplicationFactory class to streamline bootstrapping the SUT with TestServer.

Configuring Host

```
0 references | Amal Dev, 5 hours ago | 1 author, 1 change
protected override void ConfigureWebHost(IWebHostBuilder builder)
{
    builder.ConfigureServices(services =>
    {
        // Create a new service provider.
        var serviceProvider = new ServiceCollection()
            .AddEntityFrameworkInMemoryDatabase()
            .BuildServiceProvider();

        // Add a database context (AppDbContext) using an in-memory database for testing.
        services.AddDbContext<EmployeeContext>(options =>
        {
            options.UseInMemoryDatabase("InMemoryAppDb");
            options.UseInternalServiceProvider(serviceProvider);
        });

        // Build the service provider.
        var sp = services.BuildServiceProvider();

        // Create a scope to obtain a reference to the database contexts
        using (var scope = sp.CreateScope())
        {
            var scopedServices = scope.ServiceProvider;
            var appDb = scopedServices.GetRequiredService<EmployeeContext>();

            var logger = scopedServices.GetRequiredService<ILogger<CustomWebApplicationFactory<TSta

            // Ensure the database is created.
            appDb.Database.EnsureCreated();

            try
            {
                // Seed the database with some specific test data.
                SeedData.PopulateTestData(appDb);
            }
            catch (Exception ex)
```

Writing Tests

```
private readonly HttpClient _client;

0 references | Amal Dev, 5 hours ago | 1 author, 1 change
public EmployeeControllerTests(CustomWebApplicationFactory<Startup> factory)
{
    _client = factory.CreateClient();
}

[Fact]
| 0 references | Amal Dev, 5 hours ago | 1 author, 1 change
public async Task CanGetEmployees()
{
    // The endpoint or route of the controller action.
    var httpResponse = await _client.GetAsync("/api/employee");

    // Must be successful.
    httpResponse.EnsureSuccessStatusCode();

    // Deserialize and examine results.
    var stringResponse = await httpResponse.Content.ReadAsStringAsync();
    var employees = JsonConvert.DeserializeObject<IEnumerable<EmployeeEntity>>(stringResponse);
    Assert.Contains(employees, p => p.Name == "Amal");
    Assert.Contains(employees, p => p.Name == "Dev");
}


[Fact]
| 0 references | Amal Dev, 5 hours ago | 1 author, 1 change
public async Task CanGetEmployeeById()
{
    // The endpoint or route of the controller action.
    var httpResponse = await _client.GetAsync("/api/employee/1");

    // Must be successful.
    httpResponse.EnsureSuccessStatusCode();

    // Deserialize and examine results.
    var stringResponse = await httpResponse.Content.ReadAsStringAsync();
    var employee = JsonConvert.DeserializeObject<EmployeeEntity>(stringResponse);
sues found
```

Documenting API with Swagger

- Swagger (OpenAPI) is a language-agnostic specification for describing REST APIs

- Helps to
  - Minimize the amount of work needed to connect decoupled services.
  - Reduce the amount of time needed to accurately document a service.

- Two main OpenAPI implementations for .NET are
  - Swashbuckle
  - NSwag

Using Swagger

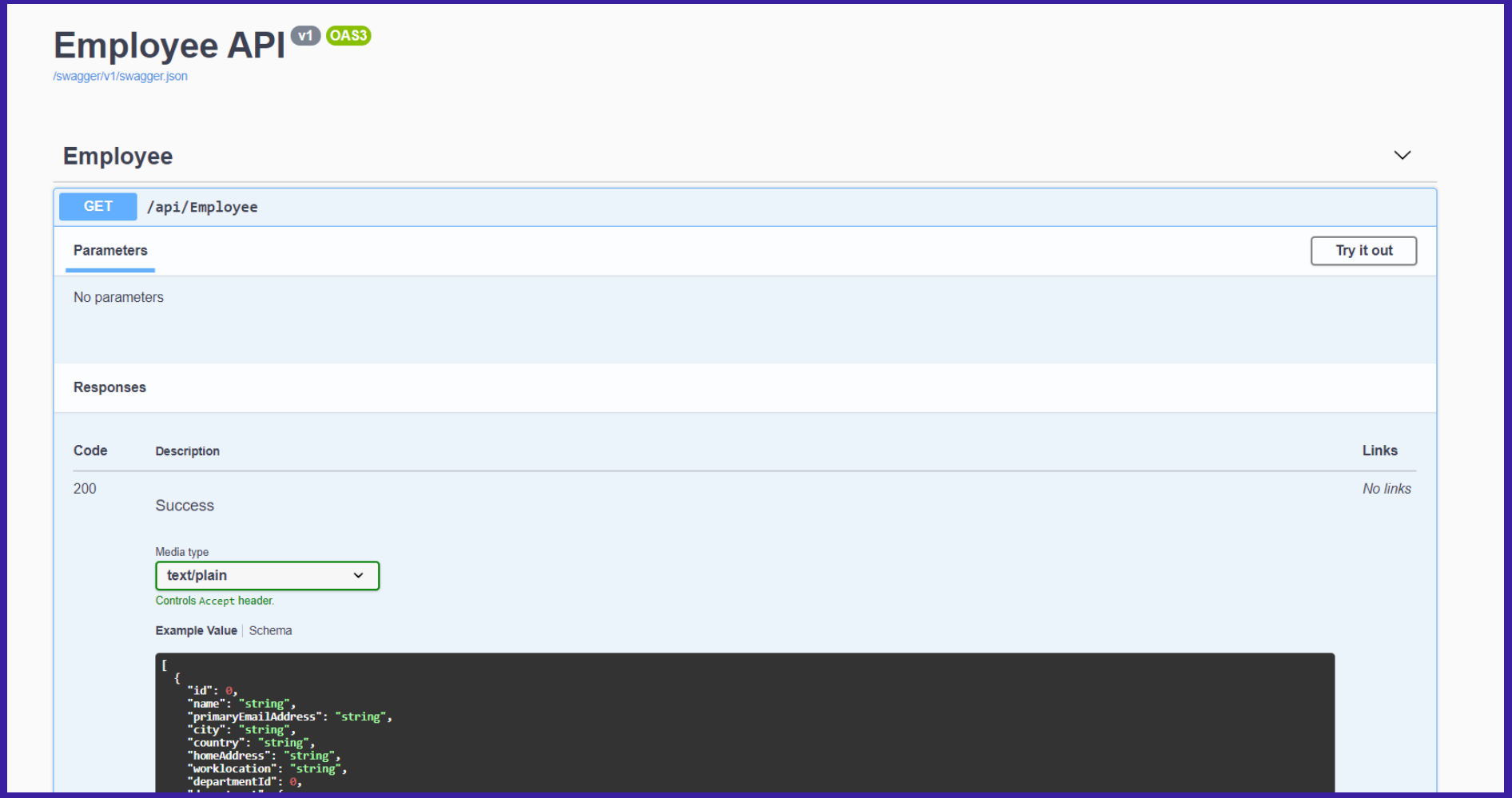
- Install Package Swashbuckle.AspNetCore
- Register Swagger

```
// Register the Swagger generator, defining 1 or more Swagger documents
services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "Employee API", Version = "v1" });
});
```

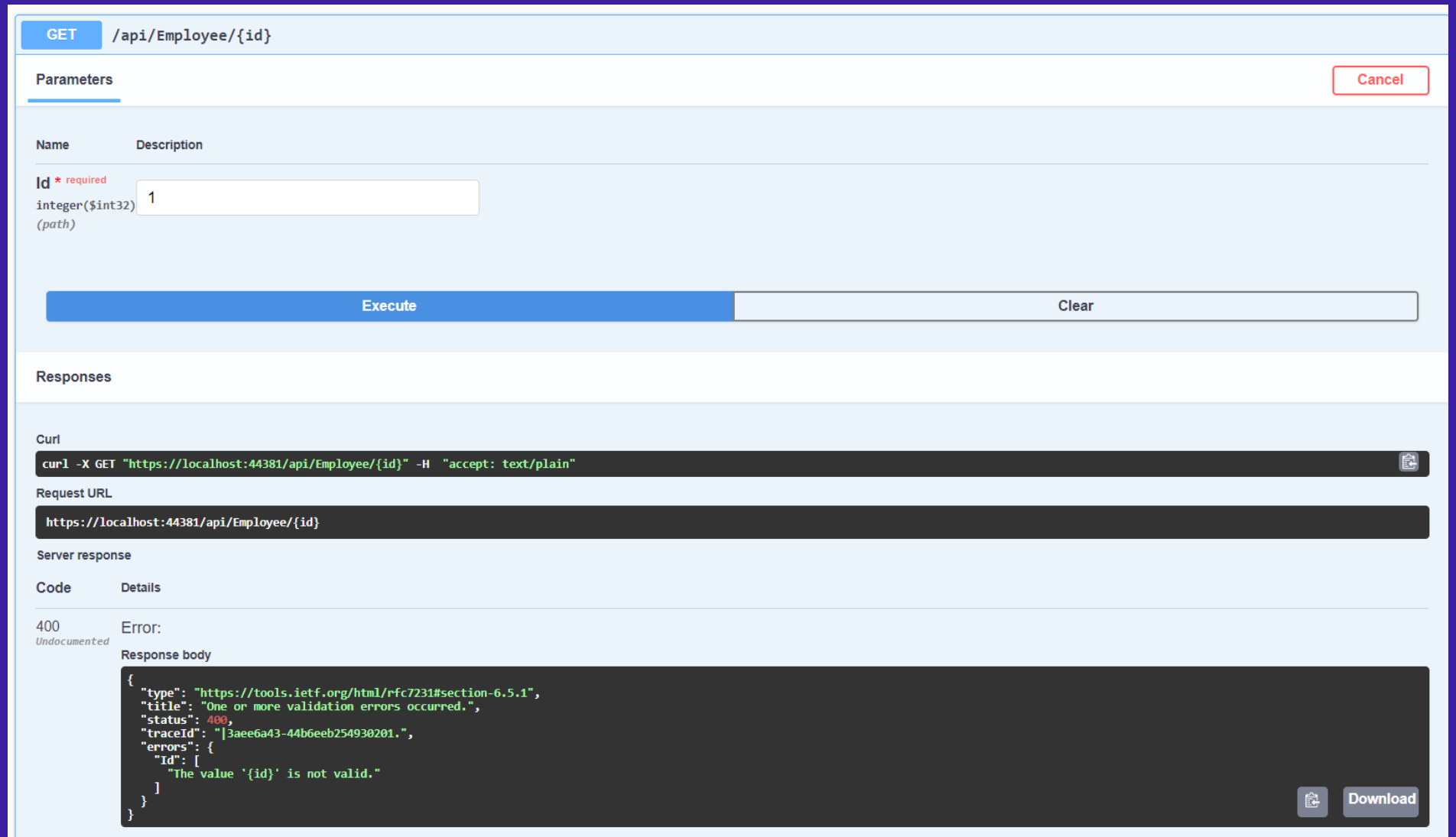
- Configure Endpoints

```
// Enable middleware to serve generated Swagger as a JSON endpoint.
app.UseSwagger();

// Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
// specifying the Swagger JSON endpoint.
app.UseSwaggerUI(c =>
{

    c.SwaggerEndpoint("/swagger/v1/swagger.json", "Employee API V1");

});
```

Swagger Page



Swagger Page – Testing Endpoints



Thanks for joining!