LAB 9 WEB API

1. Objectives

Upon completion of this practical exercise, students will be able to:

- Analyse a car sales problem involving multiple interrelated data tables
- Analyse a product management problem involving **multiple interrelated tables**
- Design a database using **Entity Framework Core – Code First**
- Work with **SQL Server** through EF Core
- Organise a project using **N-Tier Architecture**
- Develop **RESTful Web APIs** using ASP.NET Core
- Implement **CRUD APIs** with proper **HTTP Status Codes**
- Upload files using **multipart/form-data**
- Handle errors using **ErrorCode-based API responses**
- Consume Web APIs from:
  - Client side using **AJAX**
  - Server side using **HttpClient (Backend-to-Backend call)**

2. Problem Description

A company needs to develop a **Product Management System** that exposes its data via Web APIs to support multiple client types (Web UI, POS, Mobile App):

- Manage **product categories**
- Manage **products belonging to categories**
- Upload and store **product images**
- Expose RESTful APIs for client and backend consumption
- Store and retrieve data from **SQL Server**

The system is built using ASP.NET Core Web API, applying EF Core Code First, N-Tier Architecture, and Dependency Injection to ensure scalability and maintainability.

3. Required

3.1. Technical Requirements

- ASP.NET Core MVC (.NET 8)
- SQL Server
- Entity Framework Core (Code First + Migration)
- Application of:
  - RESTful API principles
  - N-Tier Architecture

  - Repository Pattern
  - Service Layer
  - Dependency Injection
- Do not use DbContext directly in controllers
- File upload via **multipart/form-data**
- API error handling using:
  - HTTP Status Code
  - ErrorCode (business-level)

3.2. Functional requirements

The system must fulfill the following functions:

- **Category Management**
  - View category list
  - View category details
  - Create, update, delete category
- **Product Management**
  - View product list
  - View product details
  - Create product (with image upload)
  - Update product
  - Delete product
- **API Consumption**
  - Call APIs from client using AJAX
  - Call APIs from backend using HttpClient

4. Project architecture

You can organize the project structure as follows:

- Presentation Layer (Controllers)
- Business Logic Layer (Services)
- Data Access Layer (Repositories)
- Data Layer (Models, DTOs)

5. Implementing basic functions

Database design using Code First & environment configuration. Implementation tasks:

- Build the following models:
  - **Category Model**
  - **Product Model**
- Establish relationships between tables using navigation properties
- Configure SQL Server connection
- Create the database using Migration

5.1. Create models & database (Code First)

In the **Models** folder of the project, proceed to create the necessary models based on the previous analysis.

Category.cs

```
public class Category
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public ICollection<Product> Products { get; set; }
        = new List<Product>();
}
```

Product.cs

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public decimal Price { get; set; }
    public string ImageUrl { get; set; } = null!;

    public int CategoryId { get; set; }
    public Category Category { get; set; } = null!;
}
```

Create DbContext & Configure Database

In DbContext, we need to declare the corresponding models to establish the foundation for generating database tables.

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
        : base(options) { }

    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=YourServer;Database=CarSalesDb;user id = sa; password
= mk@123;TrustServerCertificate=True"
```

```
    }
}
```

Program.cs

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("DefaultConnection")));
```

Create database using Code First

```
Add-Migration InitDB
Update-Database
```

After completion, two tables, Cars and Brands, will be created in the database.

Add sample data

Create a class for seeding sample data.

5.2. Configure DTOs

- CategoryDto
- CreateProductDto (with IFormFile Image)
- ProductDto

5.3. Configure DI & N-Tier Architecture

Implementation tasks:

- Create the following layers:
  - Models (*implemented in section 4.2*)
  - Repositories (Interface + Implementation)
  - Services (Interface + Implementation)
  - Controllers
- Register repositories and services in **Program.cs**

5.4. Repository Layer

- Create repository interfaces and implementations for:
  - Category
  - Product
- Perform CRUD operations
- Encapsulate all database access logic
- Repositories interact with DbContext, not Controllers.

5.5. Service Layer

- Create service interfaces and implementations
- Contain all business logic

- Call repository methods
- Be injected into Controllers

5.6. Register Dependency Injection

```
builder.Services.AddScoped<ICategoryRepository, CategoryRepository>();
builder.Services.AddScoped<IProductRepository, ProductRepository>();
builder.Services.AddScoped<ICategoryService, CategoryService>();
builder.Services.AddScoped<IProductService, ProductService>();
```

5.7. Web API Controllers (Category & Product)

a. CategoryController

Endpoints

- GET /api/categories
- GET /api/categories/{id}
- POST /api/categories
- PUT /api/categories/{id}
- DELETE /api/categories/{id}

b. ProductController (CRUD + Upload Image)

- GET /api/products
- GET /api/products/{id}
- POST /api/products (multipart/form-data + image)
- PUT /api/products/{id}
- DELETE /api/products/{id}

5.8. Client-Side API Consumption (AJAX)

Implementation tasks, build a simple HTML page:

- Category dropdown (load from API)
- Product form (name, price, category, image)

Call APIs using AJAX:

- GET categories → populate <select>
- POST products → submit FormData
- Handle responses using:
  - HTTP Status Code (xhr.status)

- ErrorCode (response.errorCode)

Students use the sample code as a reference and perform Lab 9. Perform all the above requirements and take screenshots (of the execution results), insert them into a Word file, and then submit it to the LMS.

SUBMIT YOUR CODE TO GIT!

- - - END LAB 08 - - -

REFERENCES

[1]. Hejlsberg A., Torgersen M.(2010), "*The C# Programming Language"*. 4th edition, Addison-Wesley.

[2]. Adam Freeman(2018), "*Pro Entity Framework Core 2 for ASP.NET Core MVC*" 1st ed. Edition.

[3]. HTML Tutorial: https://www.w3schools.com/html/default.asp

[4]. CSS Tutorial: https://www.w3schools.com/css/default.asp

[5]. JavaScript Tutorial: https://www.w3schools.com/js/default.asp

[6]. Bootstrap 5 Tutorial: https://www.w3schools.com/bootstrap5/index.php

[7]. Introduction to Minimal APIs in .NET 6, https://www.claudiobernasconi.ch/2022/02/23/introduction-to-minimal-apis-in-dotnet6

[8]. Minimal APIs quick reference, https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-7.0

[9]. Google Search for key word "*C#"; "WebApp*"; ".NET 6"