1. Create the Domain (Tables) using MS Excel spreadsheet
2. Create the GitHub repositories and then clone it to your development machine
3. Open Visual Studio 2022 Community Edition and create an API .NET Core project .NET8.0
4. Create the Domains on your API project by
   1. New folder named Models
   2. New folder inside the Models folder named Domain
   3. Create each domain (tables) inside the Domain folder
5. Adding Entity Framework Core Packages to connect to a SQL Server Database
   1. Right click on Dependencies and go to Manage NuGet Packages
   2. Search and browse for the following:
      1. Microsoft.EntityFrameworkCore.SqlServer
      2. Microsoft.EntityFrameworkCore.Tools
6. Creating the DBContext Class
   * Maintaining Connection to Db
   * Track Changes
   * Perform CRUD operations
   * Bridge between domain models and the database
7. Create a folder for the DBContext
   1. Create a folder named Data on the project
8. Inside the Data folder, create a new Class file for the DBContext
   1. For example, PersonalExpensesDbContext.cs
   2. Inherit DbContext to the class file, DbContext class from Microsoft.EntityFrameworkCore
   3. Create a constructor for this class (PersonalExpensesDbContext.cs)
   4. After the constructor we want to create a dbSets. A DbSet is a property of DbContext class that represents a set of entities of the database.
9. Adding ConnectionString to the Database in AppSetting.Json
10. Understanding Dependency Injection and Injecting DbContext into our application.
    1. Inject a service to Program.cs, inject the DbContext class
11. Create the database to the SQL Server using the Entity Framework Migrations
    1. Tools > NuGet Package Manager > Package Manager Console
    2. Create 2 commands:
       1. Add-Migration “Name of Migration”
       2. Update-Database
12. Create Controller and Actions: GET/POST/PUT/DELETE
13. Let’s create a controller or an endpoint for our Categories resource
14. Right click on the Controllers folder, go to the Add menu and click on Controller. Go to the API from the left hand side and choose API Controller – Empty, then click the Add button to add a controller
15. Now we have to specify the name of the controller and as you know, it has to be suffixed with the keyword controller so that the application can identify this class as a controller class. Example: CategoriesController.cs
16. Using Dependency Injection (DI), now we can use the DbContext inside the controller thru constructor
17. We start using the private file on the Action method

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PersonalExpenses.API.Data;

using PersonalExpenses.API.Models.Domain;

namespace PersonalExpenses.API.Controllers

{

// https://localhost:portaNumber/api/categories

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

[ApiController]

public class CategoriesController : ControllerBase

{

private readonly PersonalExpensesDbContext dbContext;

public CategoriesController(PersonalExpensesDbContext dbContext)

{

this.dbContext = dbContext;

}

// GET ALL CATEGORIES

// GET: https://locahost:portNumber/api/categories

[HttpGet]

public IActionResult GetAll()

{

var categories = dbContext.Categories.ToList();

return Ok(categories);

}

}

}

1. Get Region By Id Action Method

// GET SINGLE CATEGORY (Get Category By ID)

// GET: https://localhost:portnumber/api/categories/{id}

[HttpGet]

[Route("{id:Guid}")]

public IActionResult GetById([FromRoute] Guid id)

{

//var category = dbContext.Categories.Find(id);

var category = dbContext.Categories.FirstOrDefault(x => x.Id == id);

if (category == null)

{

return NotFound();

}

return Ok(category);

}

1. Change methods to us DTO – Data Transfer Object
   1. Client <-> DTO |API| Domain <-> Database
   2. Advantages of DTO:
      1. Separation of concerns
      2. Performance
      3. Security
      4. Versioning

Asynchronous Programming

* Traditional Synchronous Programming – program execution is blocked
* Poor performance (Synchronous programming)
* Async/wait keywords
* More requests