Exercise

ASP.NET - WebApi

March 2018

The OdeToFood WebApi

The OdeToFood WebApi offers a REST interface to

- Create, read, update and delete restaurants in an OdeToFood databse
- To create, read, update and delete restaurant reviews

The API should be RESTful. The table below shows how the API should respond to certain http requests:

Http Verb	**CRUD**	**Expected response**
POST	Create	201 (Created), 'Location' header with link to created resource Created resource in the body
GET	Read	200 (OK) Requested resource (list or single item) in the body
PUT	Update / Replace	200 (OK) and empty body 404 (Not Found) if resource could not be found
DELETE	Delete	200 (OK) and empty body 404 (Not Found) if resource could not be found

Step 1 – Create the Restaurants controller

Create a new ASP.NET WebApi project called “OdeToFood.Api” in a solution named “OdeToFood”.

Include a test project “OdeToFood.Api.Tests”.

1. Add a “Restaurants” WebApi controller with Read/Write actions.
2. Add a Class Library project “OdeToFood.Data” to the solution. This project will be our data layer.
    - Add a folder “DomainClasses” to the data layer and add the following domain class in this folder:

C#
```
using System;
using System.ComponentModel.DataAnnotations;

namespace OdeToFood.Data.DomainClasses
{
    public class Restaurant
    {
        public int Id { get; set; }

        [Required]
        public string Name { get; set; }

        public string City { get; set; }

        public string Country { get; set; }
    }
}
```

3. Add a “RestaurantsControllerTests” class in the “Controllers” folder of the test project.
4. Add tests (one by one) and implement the controller as you go (Red-Green-Refactor):
   - Get_ReturnsAllRestaurantsFromRepository()
   - Get_ReturnsRestaurantIfItExists()
   - Get_ReturnsNotFoundIfItDoesNotExists()
   - Post_ValidRestaurantIsSavedInRepository()
   - Post_InValidRestaurantModelStateCausesBadRequest()
   - Put_ExistingRestaurantIsSavedInRepository()
   - Put_NonExistingRestaurantReturnsNotFound()
   - Put_InValidRestaurantModelStateCausesBadRequest()
   - Put_MismatchBetweenUrlIdAndRestaurantIdCausesBadRequest()
   - Delete_ExistingRestaurantIsDeletedFromRepository()
   - Delete_NonExistingRestaurantReturnsNotFound()
5. Tips
   - Use a mock for retrieving the restaurants from a repository (IRestaurantRepository).
   - Use “IHttpActionResult” as return type of the controller actions.
   - Use a “[Setup]” method to create a new instance of the controller before each test.

6. Add a “RestaurantDbRepository” that implements “IRestaurantRepository” and uses Entity Framework to retrieve / manipulate restaurants

    - Add an “OdeToFoodContext” class (derives from DBContext

C#

```
using OdeToFood.Data.DomainClasses;
using System.Data.Entity;

namespace OdeToFood.Data
{
    public class OdeToFoodContext : DbContext
    {
        public OdeToFoodContext() : base("OdeToFoodContext"){}

        public DbSet<Restaurant> Restaurants { get; set; }
    }
}
```

    - Use the following connection string (in the app.config of the data layer and the web.config of the API):

C# (incomplete)

```
...
  <connectionStrings>
    <add name="OdeToFoodContext" connectionString="Data
Source=(localdb)\MSSQLLocalDB; Initial Catalog=OdeToFood; Integrated
Security=True;
MultipleActiveResultSets=True;providerName="System.Data.SqlClient" />
  </connectionStrings>
...
```

    - In the package manager console, Enable Migrations (on the data layer project).

```
PM>enable-migrations
```

    - Generate the database
    - Add some test data by “Seeding” the database in “Configuration.cs”

```
PM>update-database
```

    - Add a “RestaurantDbRepository” that implements “IRestaurantRepository”
        - Inject an instance of “OdeToFoodContext” into the constructor
        - Use the context to retrieve / manipulate data

Note: You have to be careful when you implement the update method of the repository. The passed restaurant might not be tracked (attached) by the entity framework.

One solution is to find the original restaurant in de DB (by ID) and then copy the values from the passed restaurant to the original restaurant:

var original = _context.Restaurants.Find(restaurant.Id);

var entry = _context.Entry(original);

entry.CurrentValues.SetValues(restaurant);

7. Use Fiddler to compose http requests to the WebApi that create, read, update and delete restaurants.

Step 2 – Create the review controller

Use the same methods as in step 1.

But now all action methods must be "async" in the review controller.

Domain class:

C#

```
using System.ComponentModel.DataAnnotations;

namespace OdeToFood.Data.DomainClasses
{
    public class Review
    {
        public int Id { get; set; }

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

        public string Body { get; set; }

        public int RestaurantId { get; set; }
        public virtual Restaurant Restaurant { get; set; }

        [Required]
        public string ReviewerName { get; set; }
    }
}
```

This will be useful if you have a Review repository that is also asynchronous so that you can await database operations (that relatively take a long time to execute).

Make sure the Review repository implements the following interface:

C#

```
using OdeToFood.Data.DomainClasses;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace OdeToFood.Data
{
    public interface IReviewRepository
    {
        Task<IEnumerable<Review>> GetAllAsync();
        Task<Review> GetByIdAsync(int id);
        Task<Review> AddAsync(Review review);
        Task UpdateAsync(Review review);
        Task DeleteAsync(int id);
    }
}
```

Tips:

- Entity Framework offers asynchronous alternatives like
  - *SaveChangesAsync()*
  - ToListAsync()
  - FirstOrDefaultAsync()