# Exercises: Unit Testing

Problems for exercises and homework for the [“C# MVC Frameworks - ASP.NET Core” course @ SoftUni](https://softuni.bg/courses/asp-net-mvc).

## News REST Service

Design and implement a **REST service** based on **ASP.NET Core Web API**, **Entity Framework Core** and **SQL Server** to hold news.

**News** items should have **title**, **content** and **publish date**.

Your service should implement the following endpoints:

* GET /api/news – returns all news ordered by publish date (from the latest) as JSON array.
* POST /api/news – creates a news item by given **title**, **content** and **publish date** (given in the request body).
* PUT /api/news/{id} – updates an existing news item by given **id** (given in the request URL) and **title**, **content** and **publish date** (given in the request body).
* DELETE /api/news/{id} – deletes an existing news item by given **id**.

Structure your application into layers:

* **Data models layer** – to hold the entity classes.
* **Repository layer** – to hold your EF Core data context and repositories (**if you have such**).
* **Web API services** – ASP.NET Core Web API application that holds your REST services.

## Unit Testing the Web API Controllers

Write **unit tests** to test your **Web API controllers**.Use **in-memory** **Database** to test the **DbContext**. Test all **service endpoints**. Your tests should run without a database. Test at least the following scenarios:

* List all news items 🡪 200 (OK) + returns the news items correctly.
* Create a news item with correct data 🡪 201 (Created) + creates a news item + returns the created item.
* Create a news item with incorrect data 🡪 400 (Bad Request).
* Modify existing news item with correct data 🡪 200 (OK) + modifies the news item.
* Modify existing news item with incorrect data 🡪 400 (Bad Request).
* Modify non-existing news item 🡪 400 (Bad Request).
* Delete existing news item 🡪 200 (OK) + deletes the item.
* Delete non-existing news item 🡪 400 (Bad Request).

## \* Add Users and Authorization in the REST Services

Modify your application to support **users and authentication**. Add an **author** in the news items. The author is the user created the news item. Modify the **POST**, **PUT** and **DELETE** service endpoints to work with authorized users only (after login). Creating a news items will require logged in user (author). Users should be able to modify and delete only their own news items (after login). Everyone (without login) should be able to list all news items. Users should be able to register in the system by username and password.

## \* Test the Authentication Functionality

Implement Unit tests for the users and authentication functionalities you’ve implemented in the previous task.

Test at least the following scenarios:

* Register a new user with correct data - 201 (Created) + creates a user + returns the created user.
* Register a new user with incorrect data (user already exists) – 400 (Bad Request)
* Login an existent user with correct data – 201 (Created) + returns a cookie
* Login an existent user with incorrect data / non-existent user – 400 (Bad Request)
* Create a news item with correct data and logged-in user, should set author – the current user 🡪 201 (Created) + creates a news item + returns the created item.
* Create a news item with correct data and no logged-in user 🡪 401(Unauthorized).
* Modify existing news item with correct data and logged-in user 🡪 200 (OK) + modifies the news item.
* Modify existing news item with correct data and no logged-in user 🡪 401 (Unauthorized).
* Delete existing news item with logged-in user 🡪 200 (OK) + deletes the item.
* Delete existing news item with no logged-in user 🡪 401 (Unauthorized).