# Homework: Testing Web Services

This document defines the homework assignments from the ["Web Services and Cloud" Course @ Software University](https://softuni.bg/courses/web-services-and-cloud/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## News REST Service

Design and implement a **REST service** based on ASP.NET Web API, Entity Framework 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 data context and repositories.
* **Web API services** – ASP.NET MVC Web API application that holds your REST services.

## Testing the Repositories

Write **automated tests** to test your **repositories**. Test the standard CRUD operations. Use transactions to avoid DB changes during the tests. You may work with a temporary database that is cleaned-up before all tests. Test at least the following scenarios:

* List all news items 🡪 returns the news items correctly.
* Create a news item with correct data 🡪 creates and returns an item.
* Create a news item with incorrect data 🡪 exception.
* Modify existing news item with correct data 🡪 modifies the item.
* Modify existing news item with incorrect data 🡪 exception.
* Modify non-existing news item 🡪 exception.
* Delete existing news item 🡪 deletes the item.
* Delete non-existing news item 🡪 exception.

## Unit Testing the Web API Controllers (with Mocking)

Write **unit tests** to test your **Web API controllers**. **Mock** the repositories to decouple them from the database layer using a **mocking framework** of your choice. Test all **service endpoints**. Your tests should run without a database (it should be mocked through the repository interfaces). Test at least the following scenarios:

* List all news items 🡪 200 (OK) + returns the news items from the mocked repository correctly.
* Create a news item with correct data 🡪 201 (Created) + creates a news item in the mocked repository + 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 in the mocked repository.
* 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 from the mocked repository.
* Delete non-existing news item 🡪 400 (Bad Request).

## Integration Testing the Web API Services

Write **integration tests** to test all your **Web API service endpoints**. To simplify your tests, use the OWIN testing server (Microsoft.Owin.Testing). You can use a temporary database, which is automatically cleaned-up at the start of the testing process. Test all **service endpoints** by invoking them and checking their output and the results stored in the database. Test at least the following scenarios:

* List all news items (GET /api/news) 🡪 returns 200 (OK) + all news items as JSON.
* Create a news item with correct data (POST /api/news) 🡪 returns 201 (Created) + the created item.
* Create a news item with incorrect data (POST /api/news) 🡪 returns 400 (Bad Request).
* Modify a news item with correct data (PUT /api/news) 🡪 returns 200 (OK).
* Modify a news item with incorrect data (PUT /api/news) 🡪 returns 400 (Bad Request).
* Delete a news item with correct data (DELETE /api/news) 🡪 returns 200 (OK).
* Delete a news item with incorrect data (DELETE /api/news) 🡪 returns 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.

## \*\* Integration Testing the Users and Authorization

Modify your integration tests to test the service endpoints. Test at least the following scenarios:

* List all news items (GET /api/news) 🡪 returns 200 (OK) + all news items as JSON.
* Register user with correct data (POST /api/users/register) 🡪 creates a new user.
* Register user with incorrect data (POST /api/users/register) 🡪 returns 400 (Bad Request).
* User login with correct data (POST /api/token) 🡪 returns a session token.
* User login with invalid login data (POST /api/token) 🡪 returns 400 (Bad Request).
* Create a news item with correct data and session token (POST /api/news) 🡪 returns the created item.
* Create a news item without session token (POST /api/news) 🡪 returns 401 (Unauthorized).
* Create a news item with incorrect data (POST /api/news) 🡪 returns 400 (Bad Request).
* Modify a news item with correct data and session token (PUT /api/news) 🡪 returns modified item.
* Modify a news item without session token (PUT /api/news) 🡪 returns 401 (Unauthorized).
* Modify a news item with incorrect data (PUT /api/news) 🡪 returns 400 (Bad Request).
* Delete a news item with correct data and session token (DELETE /api/news) 🡪 returns modified item.
* Delete a news item without session token (DELETE /api/news) 🡪 returns 401 (Unauthorized).
* Delete a news item with incorrect data (DELETE /api/news) 🡪 returns 400 (Bad Request).

You may clean the database and put some testing data in it before the execution of all tests to simplify your work.