**Project Description**

The implemented solution is a simple online TODO list with a web interface that can be used in all popular web browsers. The application support smultiple users, store necessary data in an persistant database and it is built using the following technologies/frameworks:

* C# .NET Framework 4.6.1
* .NET MVC 5 with Entity framework 6.2
* SQL Server 2016
* HTML, CSS, JavaScript/Jquery

**Solution Explanation**

**Architecture**

The technique used for designing the solution is the 3-Layer architecture. A short description about the three layers follows:

1. Data Access Layer

It is a class library project that implements this layer, which is responsible for storing and retrieving data to a persistent database.

1. Business Layer

It is a class library project that implements this layer, which coordinates the application and it is the “middle man” between the data access layer and the presentation layer.

1. Presentation Layer

It is a web application project that serves the user interface of the application.

In more details, the data access layer implements the repository pattern for handling all the transaction with a SQL Server databaase. Due to the complexity of the project, the generic repository pattern is not implemented but for a larger scale application, it should be highly considered. Moreover, Entity Framework 6.2, using code first workflow, is used as the ORM.

The business layer contains the business models and also the classes which coordinate the flow between the two other layers. Due to the complexity of the project, this layer is responsible for mapping (using Automapper library) business models to Entities and vice versa and furthermore, it implements the validation of the model. Therefore, the business models (instead of the Entities) are exposed to the Presentation Layer.

The presentation layer contains the UI of the application and it contains two Views, a login page and a manage tasks page.

These three (3) layers are loosely coupled, meaning the data layer communicates with business layer and the latter communicates with the presentation layer. Therefore, there is no direct connection between the data and the presentation layer. This is achieved by using the technique of dependency of injection, which is implemented with the use of Unity library. This technique was chosen because it offers more scalability and maintainability to the application.

**Functionality**

The application starts with a login page where the user enters his/her credentials. The authentication mechanism is a custom solution where the password entered by the user, is encrypted on the server side (business layer) and checked against the encrypted in the database (user table) for the given username. Therefore the passwords are stored encrypted in the database (using two way encryption). A better and more secure practice is to use one-way encryption.

After successful login, the user is redirected to the main page of the application, where the user can manage his/her tasks. In more detail, the user can add, removes and check/unchecks tasks.

It should be mentioned that, in order to improve performance, the application uses Sessions to store the user and his/her tasks in memory. While the user manages his/her tasks (adds, removes, checks/unchecks tasks), each update is stored in the Session. The application offers a save button, which when clicked, it stores all the data from the Session to the database.

This approach was chosen in order to reduce the transactions with the database. The implementation of the Session mechanism is kept at a basic level and it is open for a more sophisticated implementation.

**Potential improvements on the solution**

1. Use of generic repository pattern
2. A more secure authentication system (e.g. one way encryption)
3. A better and more intuitive UI design (e.g. use of partial views,displaying warning messages)
4. The testing of the application was done manually, using the UI. Ideally I would like to have introduced UI and unit testing.

**Remarks/Issues**

In this chapter some issues and remarks regarding the solution are presented.

Because of not understanding entirely the following requirement:

***<<store necessary data in an in-memory database>>***

I decided to create the database on my local SQL Server 2016. For setting the database with Entity Framework, I followed the Code-First workflow.

Therefore, I followed the steps below:

1. Create the entities in the Data Access Layer project
2. Create the ToDoListContext class
3. Add the connection string in the web.config file, as follows:

<connectionStrings>

<add name="ToDoListContext" connectionString="Data Source=DESKTOP-P42IFH2;Initial Catalog=ToDoListDB;Integrated Security=SSPI;"providerName="System.Data.SqlClient" />

</connectionStrings>

1. From the package manager console, I changed the default project to DataAccessLayer and typed the following commands.

Enable-Migrations

Add-Migration <<name>>

Update-Database

In order to set up the database on an SQL Server, you could follow stepσ 3 and 4.

Moreover, in the DataAccessLayer project, there is a class named Configuration (under Migrations folder), where the method Seed is responsible for populating the database with some data.

For testing reasons, the following users have been created:

Test → psw123 → ax0NS/fHwojKxO42+XM5dQ==

Test2 → psw456 → Fjf/Nkn25cC/3b03/stKdQ==