Advanced Development Techniques

Project Work

2021/22/1 semester

During the project work each student have to develop a git version controlled layered CRUD application which uses a database.

Here below in this document the requirements can be read. The obligatory parts must be completed fully, otherwise the project work can not be accepted. There are some requirements which are counted as “minus points” if they are not fullfilled, these are listed at the end of the document.

Minimal requirements regarding the project work:

- The project work must be version controlled via git, starting from the very first steps. Please note, that this is not an “upload site”, so adding the project as a last step before deadline is not acceptable.
- To the local git repository a remote must be connected. It must be created at github.com.
- At github.com the repository’s name must be exactly: **ABC123_HFT_2021221**, where the ABC123 must be replaced with your own Neptun code, HFT is the subject’s name (in Hungarian) and 2021221 is the semester identifier. Only the ABC123 part should be changed, all other must remain as it stands here.
- The repository’s visibility must be **private**! In the repository settings the **oenikprog** github user must be added. (If github asks about the role, it should be added as an **admin**.) After the invitation the invited user has to accept it manually, by hand so it will take some time.
- During the development, as one-one unit of code is created it’s advised to commit the changes. In the project work at least 25 commits must be made.
- Before the milestones (see later) the commits must be pushed to the remote repository. Commits can be pushed more often of course, but at least before each milestone.
- Between the student and the instructor the code must be shared only via github. Sending via email or other methods are not accepted.

Minimal requirements regarding the empty solution's structure:

- To start the project work an empty **Console Application** (in **.Net 5.0**!) should be created from Visual Studio. The solution's name must be: **ABC123_HFT_2021221** (where the ABC123 must be replaced with your own Neptun code as well). The project name must be **Client**.
- In the solution the following structure must be created, by creating new projects. In the projects' name the solution's name must be used as well (with the replaced Neptun code of course), like below:
    - **ABC123_HFT_2021221 Models** (Class Library)
    - **ABC123_HFT_2021221 Data** (Class Library)
    - **ABC123_HFT_2021221 Logic** (Class Library)
    - **ABC123_HFT_2021221 Repository** (Class Library)
    - **ABC123_HFT_2021221 Endpoint** (ASP.NET Core Empty and does not need https support)
    - **ABC123_HFT_2021221 Test** (Class Library)

- In the root, 3 folders must be created by clicking on the solution's name and Add new folders from the Visual Studio's solution explorer. Then, after the folders created via drag-n-drop method the previously created projects should be moved accordingly to this:
    - **Backend** (Data, Endpoint, Logic, Repository, Test)
    - **Frontend** (Client)
    - **Shared** (Models)
- It must be set that clicking on the Start button both the backend application (server) and the frontend application (client) starts. Right click to solution's name, properties, startup project, multiple startup projects, and mark client and endpoint as start.
- Dependencies must be set as well. Right click to any given project, add, project reference. Dependencies listed below, these must be set accordingly! Other dependencies must not be used, except if you want to test the Client, but before the final deadline the Client can only have the Models.
    - **Endpoint** 's dependencies: **Data, Logic, Repository, Models**
    - **Test** 's dependencies: **Logic, Repository, Models**
    - **Data** 's dependencies: **Models**
    - **Logic** 's dependencies: **Repository, Models**
    - **Repository** 's dependencies: **Models, Data**
    - **Client** 's dependencies: **Models**
- When all these are set up and there is no C# code written yet, it's time to initialize a git repository.
    - **Please note** that the following steps will include a VS integrated git management tool, which is easy to use. But feel free to use other methods like discussed in the lecture video you can use GitKraken as external dedicated git GUI tool, or simply CLI itself. You can also create the repoistory itself from the Github GUI and then clone it, and initialize the solution into that folder. In any way, at the end you will end up having one git repository with a local (on your machine) and a remote (on Github).
    - In the VS at the bottom right corner there is a button "Add to source control". This is visibly only if during the VS install, on the individual components page **Git for Windows** and **Github Extension for Visual Studio** was previously selected. If it has not been done then the VS install should be modified. On Windows go to Control Panel / Install and delete programs / Visual Studio 2019 Community (or other version) and click on change.

For database management **Data Storage and Processing Workload** should be added as well from here.

- After clicking on the "Add to source control" button the Github account's details should be entered, check if the repository name is correct, check if the repository's visibility is correct. If everything is good click on "**Create and Publish**".
- When we follow the above mentioned steps and start version control from VS itself, then the **.gitignore** file is automatically created to fit our needs for the C# language. But, one modification must be done regarding the mdf and ldf files.
- Create an "Others" folder in the solution, right click on it, add existing item and open the .gitignore file from our project's root.
- The file's 265. and 266. line contains *.mdf and *.ldf, delete these lines. (By deleting these, it means that these file types will not be ignored, as they are not listed in the .gitignore file.)
- Since we already made our first two modifications after the git init, create our first commit for the changes.
- At the bottom right corner of the VS now there are other buttons available. Look for the pencil icon, which will show how many files are changed. Click on it and in the commit box write "mdf and ldf removed from gitignore" and click on **Commit All** button.
- After this a local commit was created, but it's needed to be pushed to the remote repository at Github. On the panel the **up arrow** means the **push** command, click on it and then check it from the Github web UI if everything is good.
- During the development try to commit often and commit small pieces of code changes. After let's say each day push the changes to the remote. Note that 1 push can push whatever number of commits.
- It's a good practice to download the project from Github (using the green clone button and select download zip), and unzip it to a separate folder on your machine. If everything is fine you should be able to build the full solution without any problem. The instructor will see exactly this state!

Minimal requirements regarding the software:

- The software must be build without any problem on the instructor's computer.
- The software must be created in **.NET 5.0** version and must use **MS SQL** database with **LocalDB**, above **Entity Framework Core**. Different approaches will not be accepted.
- The program's classes, methods, variables must be named in English, and also comments must be made in English.
- In the project work at least 3 data tables must be created which has a connection to each other using foreign keys. This means that 3 **Model** classes must be made in the **Model Class Library**. Example: one brand has many cars, every car has many renting event. If many-to-many relation is created then the connection table does not count as a table (out of the 3).
- In the **Model** classes the foreign keys should be added and use **Navigation Properties** with **LazyLoader**. Use join in the Linq queries if there is no other way.
- The **MDF** and **LDF** files should be created in the **Data** layer and their **Build Action** must be set to **Content**, and the **Copy to output directly** must be set to **Copy always**.
- In the **Data** layer at the **DbContext** class's **OnModelCreating** method the database should be seeded with test items.

- During the development for testing, it's possible that the **ConsoleApp** receives the Logic, Repository, Data and Models layers as project reference, but at the end the ConsoleApp only can have **API calls** to the endpoint and can only know the Models library.
- The Logic can only receive the Repository as a dependency through interface reference in the constructor (dependency injection). The repository can only receive the DbContext as a dependency through it's constructor. The Endpoint's controllers can only receive the logic as a dependency through interface reference in the constructor as well. The insertion of the dependencies is made by the Endpoint project, using IoC container. For testing purposes they can be created manually in the Console App.
- Every **Model** class must have one dedicated repository class, which handles the **CRUD** operations (Create, Read, ReadAll, Update, Delete). The ReadAll method should return the DBSet collection to the Logic as an **IQeryable<T>** interface reference.
- The **Logic** layer must have methods for these **CRUD** operations and also must have **at least 5 non-crud** methods, which are used for multi-table queries. The CRUD and and non-crud methods' return values must be passed as an **IEnumberable<T>** reference to the above layers. Example for a non-crud method: for a given car brand who is the customer who has placed the most renting demand (in order to satisfy this query all 3 tables/entities must be used).
- In the **Test** project **Nunit** and **Moq** libraries must be used. The Logic must receive a **fake-database** (mocked database) using moq. Unit tests firstly should be aimed to test the **non-crud functionalities** and secondly to test the **exception handling** of the create methods (eg. for an empty username an exception should be thrown). The logic's create differ from the repository's create in terms of error handling! The create method in the repository simply saves the entity to the database, without any validation or checking!
- In the project work **at least 10 Unit Tests** must be created. For example 5 non-crud tests, 3 tests for the create functions and 2 can be chosen freely.
- Every Model class should have one dedicated Repository class and one dedicated Logic class. Example:
    - Car → CarRepository
    - Car → CarLogic

  One logic class can use multiple repositories in order to have more complex queries in the non-crud methods.
- The project's **Endpoint** layer can know the Logic classes and their functions will be published to the outside world in form of **API Endpoints**. To every Logic class there could be one or more **ApiController**. The **Actions** of the **ApiController**s can be understood as the Logic's methods. Usually:
    - HTTP GET Read, ReadAll
    - HTTP POST Create
    - HTTP PUT Update
    - HTTP DELETE Delete
- The **Console** application **sends API requests** to the **Endpoint** as **JSON** messages. From the Console application all the CRUD and non-crud methods should be available. For this, from the nuGet repository the ConsoleMenu-Simple can be used.

Milestones of the project work:

The project work will have milestones when an inspection mechanism will be used (gitstat) which will check for every student that the requirements of a given milestone is fulfilled or not. The result of this will be available to the students. Every milestone check will contains the previous milestones as well. After the 5th milestone this mechanism will be run daily.

October 10. 23:59:59

- Github repository is created with the correct naming
- oenikprog user was invited to the repository
- the empty projects are created
- .gitignore file was modified
- .mdf and .ldf files are created in the Data layer
- the last three steps were committed and pushed to the remote repository (at Github)

October 17. 23:59:59

- in the Model layer at least 3 classes are created with content inside
- in the Data layer the XYZDbContext class is created
- in the DbSeed all three tables (or if there is more, all of them) are filled with test data as part of the initialization

November 7. 23:59:59

- in the Repository layer at least 3 classes are created
- in these classes all the CRUD methods are created

November 21. 23:59:59

- in the Logic layer the classes are created
- in these classes all the CRUD and non-crud methods are created
- 10 out of 10 unit tests are created and all can be run without any fails

November 28. 23:59:59

- the Endpoint application should be ready, it can be built/run and it reacts to API calls

Deadline #1: December 2. 23:59:59

Deadline #2: December 9. 23:59:59

The deadline means that at that time the instructor will download the code and starting from that, no change in the codebase can be accepted.

Minus points of the project work:

For the project work students can have 20 points, everybody starts from this amount. 50% must be reached, so in total 10 points can be lost, if there is more, then the project work will not be accepted. If for the 1st deadline there are some minus points (determined by the instructor) but in total they reach 50% or more (so the project work can be accepted), then those can be corrected for the 2nd deadline if wanted by the student to reach a better grade. If something is missing for any given milestone, it means that that given milestone is not fulfilled, which will result in -1 point. The instructor can give minus points at his own discretion for the followings, justifying the reason:

- if the layers not correctly used
- if there are any hidden dependencies
- if there are pointless unit tests
- if there are run time errors
- if there are pointless classes, variables, methods (usage or naming)

Moreover, everything which was described in detail as a minimal requirement in this document must be fulfilled. Not satisfying any of those will result in a not accepted project work.

Rules of the communication with the instructor regarding the project works:

- firstly, the lab instructor should be asked personally at the end of the lab occasions
- secondly, the lab instructor should be asked via email with a detailed problem description and precisely defining what is the question and listing what methods you have already tried to solve the problem
    - if any of these is missing it would be more problematic to help, so please respect the rules and try to be as much detailed as possible but in the meanwhile be straightforward