Milestone 3

DBMS

General

After the success of the second milestone, you are now requested to further expand the functionality of your kanban application, and to replace the JSON files with a database.

Note: this milestone is a direct continuation of milestone 2.

Goals

The goals of this milestone are:

- Practice the use of databases.
- Practice the N-tier architecture.
- Practice team work & version control.
- Practice C#.
- Experience right use of OOP with C#.

Business and Integrity Rules

1. A user is identified by an email, and is authenticated by a password.
2. A board has two or more columns.
3. A column has the following attributes:
    a. A name (max. 15 characters, not empty).
    b. A list of tasks.
    c. The maximum number of tasks that it may contain (unlimited by default).
4. A task has the following attributes:
    a. The creation time.
    b. A due date.
    c. A title (max. 50 characters, not empty).
    d. A description (max. 300 characters, optional).

Functional Requirements

Users

1. A user password must be in length of 5 to 25 characters and must include at least one uppercase letter, one small character and a number.
2. Each email address must be unique in the system.
3. A user has a nickname.
4. The program will allow registering of new users.
5. The program will allow login and logout of existing users.
6. The login will succeed if, and only if, the email and the password matches the email and password of a registered user.

Boards/Columns

1. By default, a board has three columns: 'backlog', 'in progress' and 'done', in this order.
2. The order of the columns can be changed.
3. Each column should support changing ~~limiting~~ the maximum number of its tasks.
4. ~~By default, there will be no limit on the number of the tasks.~~ (move to the definition of column in 3-c.
5. The board will support adding new tasks to its ~~backlog~~ leftmost column only.
6. Columns can be added to the board and removed.
7. When a column is removed, its existing tasks are moved to the column on its left (unless it is the leftmost column – then its tasks are moved to the column on its right).
8. Tasks can be moved from ~~'backlog' to 'in progress' or from 'in progress' to 'done' columns.~~one column to the following column only (left to right). No other movements are allowed.

Tasks

1. A task is done when it is moved to the last column.
2. Tasks that are not done yet, can be changed.
3. All the task data can be changed, except for the creation time.

Non-functional Requirements

1. Persistence:
    a. In this assignment, persistent data should be managed by a DBMS (SQLite).
    b. The following data should be persisted:
        i. The users

ii. The boards (to be clear: column order, name and task limits should be persistent.

c. Persisted data should be restored once the program starts.

2. Logging:
   You must maintain a log which will track important events (e.g., a user created or a task was created) and all errors in the system. Note that "error" does not necessarily mean an exception that is "thrown" or "raised" by your runtime environment, but any situation which counts as invalid in our domain (more details on errors in the following requirement). Some guidelines:
   a. Tag entries with their severity / priority
   b. Provide enough information to understand what went wrong, and where
   c. Avoid storing entire stack traces directly in the log (they are more verbose than useful)
   d. Use log4net for logging your application.
3. Error Handling is the process of responding to the occurrence, during computation, of errors – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution. Your program is expected to operate even when error occurs, i.e.:
   a. Handle any malformed input.
   b. Handle logic errors (e.g., login for non-existing users, etc.).

General Guidelines

Version Control

- You are expected to work as a team and use source control (GitHub specifically). You should use the git workflow taught in class (branch for each feature and use pull requests).
- ~~We will use GitHub classroom. You must register and login to GitHub **With your BGU email**.~~
- ~~Join our GitHub classroom using this link: https://classroom.github.com/g/XwgfxBlY~~
- ~~One team member should create a team.~~
- ~~The team name should consist of 4 characters exactly!~~
- ~~The other team members should join the team.~~
- ~~A repository will be created for the team automatically with the interface supplied by us. You should use **only** this repo during the development of this milestone.~~
- ~~You should edit the `README.md` file and replace ID# with the IDs of the team members.~~
- You will continue your work on the repository from the last milestone. New service files are uploaded to the model. Replace them all in one commit.

Files

- The code for each Tier (service layer/data access layer/business logic) will be placed in a separate directory. In total, you should have three folders with the names of the layers.
- To be clear: you do not develop a presentation layer (e.g., GUI/console).
- You may add folders, classes and interfaces **as needed**.
- You must not changed the initial files given to you in the template, except for:
    a. In `ServiceLayer/Objects` you can only add code to the files and not change the current code.
    b. In `Service.cs you` can only add private fields, private methods, and change all the places with "`throw new NotImplementedException();`"
    c. Changing the signatures in these files is forbidden and might result in exceptions raised during our test, the grade for failed submissions will be zero.
- `Service.cs` will be used by us for testing your code, which is why you must not change the signatures of its methods. However, the cohesion of `Service.cs` is bad. You should create multiple service files, each contains only part of the methods (with the same signatures), and call these methods from `Service.cs`. For example, the `Register` methods of `Service.cs` should look like:

```
public Response Register(string email, string password, string
nickname)
{
    return _otherServiceClass.Register(email, password, nickname);
}
```

General

- Document your code thoroughly. See the example of the class "Service" given to you by us. To add such comments place your cursor above a function/class and write three forward slashes: "///". Additionally, add comments to your code where necessary. See `IService.cs` for examples.
- Pay attention to "Magic numbers".
- Static members/methods/classes are forbidden.
- You may test your application by:
    a. Create a branch called, e.g., "tests", and checkout to this branch.
    b. (In this branch) Create an additional "console" project in the solution and add the dll of the "Backend" project as a project resource.

c. In the `Program.cs` file of the "test" project – call the service layer methods.

Design instructions

- Before starting your design (and of course writing your code), consider the following:
  - Are there any operations that a column can/should do?
  - Are there any operations that a task can/should do?
  - Are there any operations that a user can/should do?
  - Where does each method belong? (login & logout/create & move a task, etc.)
  - Can you think of new requirements that the client (the course staff) will **probably** request? Can you make the design and the code versatile enough to support such requests (of course without too much work on your side because we might not ask for it)?
- N-Tier structure: pay attention to right use of the N-tier model.
- OOP principles: remember OOP principles and use them (Encapsulation, Abstraction, Interface, and Inheritance).
- The aforementioned definitions for a user, a board and a task, contain the required information for the functional requirements. You are allowed to add or change the data fields as long as the requirements are met.
- Having a persistent layer (and backuping data in a database) does not mean that the data objects (i.e., users and boards) are stored in files only. In fact, these data objects will probably have some logic and methods (e.g., the user object). Moreover, your data should be stored in the RAM for fast retrieval. The persistent layer should be called only upon system loading (for restoring the persistent data) and upon data updating (e.g., registration of a new user).

Submission Dates, Deliverables and Grading

Deliverables

1. An updated class diagram of your project, added to the root of your `master` branch, by the name "`diagram.pdf`". You can create the diagram using https://draw.io.
2. A zip of your final code, including `diagram.pdf` but not including `bin` and `obj` directories. Upload the zip file to the Moodle assignment.
3. You should submit your files to the department submission system by this link. You can login with your university username and password. Then, you select the course "IntroSE" and the assignment is "Milestone2". You should upload the **zip file** of your submission and execute it in order to see the result.

General Submission Notes

1. You may submit as many times as you wish, as long as the submission deadline has not passed.
2. There will be no extension **at all!** Do not wait for the last minute to submit.
3. Your class diagram should match and reflect your actual program. Please update it and re-push it to the repository before the final submission.


Good Luck