Milestone 3

WPF & Design

New information in yellow.

General

Kanban is a project management methodology that allows for organizing and prioritizing projects and their tasks, using boards, lists, and cards. With the right Kanban board software, one can focus on priorities, keep the team in the loop about what's current and what's coming up, and control a steady flow of work, especially in output-heavy environments like production, agencies, support, and maintenance. This is a tool that supports agile development and helps to visualize and manage the workflow.

In this milestone, you are requested to improve the implementation of your Kanban backend and to add a graphical user interface (GUI) to it.

The current version of the kanban board will have a GUI and it will have a service layer that you (and we) will use for testing your software. Thus, your kanban software will have four layers constructed by you: a presentation layer, a service layer, a business layer, and a data-access layer.

Note: milestones 1-3 are not related to assignment 0 in any way.

Goals

The goals of this milestone are:

- Practice WPF.
- Practice MVVM.
- Practice the N-tier architecture.
- Practice teamwork & version control.
- Practice C#.
- Experience right use of OOP with C#.
- Understand and write a Low-Level Design (LLD).
- Working with databases.
- Practice design patterns
- Practice unit testing and mocking

Business and Integrity Rules

1. A user is identified by an email and is authenticated by a password.
2. 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).
    e. The assignee (the person the task is assigned to)
3. A board has a unique id, a name, and three columns: 'backlog', 'in progress', and 'done'.

Functional Requirements

Users

4. A user password must be in length of 4 to 20 characters and must include at least one uppercase letter, one small character and a number. The password must not be any of the top 20 passwords published by the National Cyber Security Centre.
5. Each email address must be unique in the system.
6. The program will allow registration of new users.
7. As a user, I want to be able to login using an email and a password, and logout.
8. As a user, I want to be able to create a new board or join an existing board created by someone else (and become a board member).

Boards

9. Board is identified uniquely by its name and the email of its creator (i.e., the same board name can be used by two different creators).
10. Each column should support limiting the maximum number of its tasks.
11. By default, there will be no limit on the number of the tasks.
12. The board will support adding new tasks to its backlog column by any of the board members.
13. By default, a board has three columns: 'backlog', 'in progress' and 'done', in this order.
14. Columns can be added, removed, renamed, and reordered.
15. A board will always have a minimum of two columns.
16. A task is considered as 'done' if it is in the rightmost column, regardless of the column name.
17. A new task is added to the leftmost column.
18. Any tasks that is not in the rightmost of leftmost columns is considered as 'in progress'

19. 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). The operation should fail if the tasks cannot be moved to the new column (since it will exceed the limit).
20. Only empty columns can be moved.
21. Tasks can be moved from one column to the following column only (left to right). No other movements are allowed.
22. Only the board's creator can delete a board.
23. When deleting a board, all of the tasks are deleted.

Tasks

24. A task that is not done can be changed by its assignee.
25. All the task data can be changed, except for the creation time.
26. As a user, I want to be able to list my 'in progress' tasks (that I am assigned to) from all of my boards, so that I can plan my schedule.
27. A task can be assigned to any board's members. By default, a task is assigned to its creator.

User Interface

28. Tasks can be sorted by the due date.
29. An overdue task will be colored as red.
30. A task that 75% of time passed since the creation will be colored as orange.
31. A task that is assigned to the logged-in user will have a blue border.
32. Tasks can be filtered by its text fields (i.e. the title and/or the description). For example: filtering tasks by the search term "login" will return tasks that their title/description includes the word login, for example:
    a. Implement a login screen.

Non-functional Requirements

1. Persistence:
    a. In this assignment, persistent data is stored in a SQLite database.
    b. The following data should be persisted:
        i. The users
        ii. The boards
    c. Persistent data should be restored once the program starts.
2. Logging:
   You must maintain a log that 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 an error occurs, i.e.:
   a. Handle any malformed input.
   b. Handle logic errors (e.g., log in for non-existing users, etc.).
4. Testing. You should test three methods using Nunit and moq.
   a. The three methods must **not** be constructors or properties.
   b. The three methods **must** be member functions of the same class.
   c. The class **must** belong to the business layer.
   d. Each method **must** have some logic, for example: contain an "if" condition, throw an exception in certain cases, etc.
   e. You should test your methods using a Nunit project as described here.
   f. You should reach 100% coverage on the chosen methods. You can get coverage report by "Test"->"Analyze Code Coverage" -> "All tests"
      i. More info
      ii. Testing the coverage is available only in Visual studio professional. You can download it for a 30 days trial
   g. Your tests must not depend on other classes, you may use mocking when needed.
5. Usability. The GUI must comply with the following requirements:
   a. Buttons in the interface should contain only short text descriptions (no more than two words) or a descriptive icon (preferably both).
   b. Text should be large enough to comfortably accommodate user input, but not so large that they are unwieldy.
   c. Data input by users should be validated, and input of invalid characters should be blocked (e.g. empty messages, nicknames, etc.).
   d. For additional guidelines, read this page on Good User Interface Design Tips.

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/PDpmbCoQ~~
    a. ~~Make sure you select your BGU username in the first window~~
- You will continue using the repository created to you in Milestone 1 in this milestone.
- ~~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.~~
    a. A new service layer is given to you through moodle. Overwrite the old service layer, commit & push before you start this milestone.
    b. Do not change the signatures of the service layer!
        - You are allowed to change the code in the methods
        - Do not change method names, object names, method signatures, file names etc.
- You should create a `README.md` file and place ID# with the IDs of the team members: ID1_ID2_ID3 at the beginning of the first line of the file.

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).~~
- Make sure you understand the access modifiers and use them correctly.
- You may add folders, classes and interfaces **as needed**.
- You **must not** change the methods signatures or the fields/getters/setters in the initial files given to you in the template.
- You should have a DB file named kanban.db. This file should be contained in the ZIP file when submitting. Make sure the database has the tables (schema) and that the tables are empty (from rows) when submitting.
- You **can** add additional methods/fields/getter/setters if needed.
- You **must** implement the methods in Service.cs.
- `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);
}
```

- The code for the GUI tier will be placed in a **different project**, named "Presentation".
- The code for the Tests will be placed in a Nunit project named "Tests"

General

- Document your code thoroughly, specifically – all classes and public methods/getters. See the example of the class "Service" given to you by us. To add such documentation, read Microsoft's tutorial for documenting your C# code.
- 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.
- Use bindings when writing the GUI, and pay attention to the binding mode. **You are not allowed to read properties directly** (for example a function should not use the text of a textbox directly).
- You are required to use design patterns in your code - **where needed**. For your old and for your new code.

Design instructions

- Before starting your design (and of course writing your code), consider the following:
    - What are the operations that each entity in the system can/should do?
    - Where does each method belong? (login & logout/create & move 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 and to MVVM.

- 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 backup data in files) 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 (BL+DAL), added to the root of your `master` branch, by the name "`diagram.pdf`". You can create the diagram using https://draw.io.
    a. An initial version of your class diagram will be submitted by Moodle until **30/5/2021**.
    b. During your work, your design might (and probably will) need modifications. Whenever it happens – change the design and then change the code. Make sure that your code structure is aligned with your diagram.
2. A zip of your final backend code (excluding UI and tests projects), including the latest `diagram.pdf` but not including `bin` and `obj` directories. Upload the zip file to the Moodle assignment.
    a. Your final `diagram.pdf` (submitted inside the zip file) should include a list of all the changes you made from the initial diagram, with explanations why you performed these changes.
3. A diagram of your database added to the root of your `master` branch, by the name "`database_diagram.pdf`"
4. An explanation of where you used design patterns and how, in a single page pdf called "`design_patterns.pdf`"
5. A test coverage report with of your selected class `classname`, named "`<classname>.coverage`"

General Submission Notes

1. The functionality of your system will be tested automatically in Moodle by submitting a ZIP file with your project. You may submit as many times as you wish, as long as the submission deadline has not passed.
    a. ZIP only the backend folder + kanban.db (same as last assignment). Do not submit the unittest/gui.
2. There will be no extension **at all!** Do not wait until 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