**.NET Application Programming**

**Project Status and Design Report**

|  |  |  |
| --- | --- | --- |
| **Topic:** | *Milestone 4* | |
| **Date:** | *3/11/18* | |
| **Revision:** | *1.0.3* | |
| **Team:** | 1. *Ali Cooper* | |
|  | |
|  | |
|  | |
| **Weekly Team Status Summary:** | |  |  |  |  | | --- | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | **Hours Remaining** | | *As a user, I would like to be able to play minesweeper without the page reloading.* | *Ali Cooper* | *2* | *0* | | *As a user, I would like to be able to have the timer work without the timer lagging* | *Ali Cooper* | *.5* | *0* | | As a user I would like a modal prompt for resetting the game board | *Ali Cooper* | *0* | *.5* | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | |
| **GIT URL:** | https://drowsyWarble@bitbucket.org/drowsyWarble/cst247.git | |
| **Loom** | https://www.useloom.com/share/888739708d8744df9dbf0add9d78fa55 | |
| **Peer Review:** | *Y* | We acknowledge that our team has reviewed this Report and we agree to the approach we are all taking. |

**Planning Documentation**

**Agile Scrum Product Backlog:**

https://bitbucket.org/drowsyWarble/cst247/raw/dc3c7d23605eb84fd2360341b16aed81f09fa4b5/Documentation/Scrum/ProductLog.xlsx

**Agile Scrum Sprint Backlog:**

https://bitbucket.org/drowsyWarble/cst247/raw/dc3c7d23605eb84fd2360341b16aed81f09fa4b5/Documentation/Scrum/Sprint1Backlog.xlsx

**Agile Scrum Burn Down Chart:**

https://bitbucket.org/drowsyWarble/cst247/raw/dc3c7d23605eb84fd2360341b16aed81f09fa4b5/Documentation/Scrum/Sprint2Backlog.xlsx

**Agile Retrospective Results:**

|  |
| --- |
| **What Went Well** |
| Separating UI into partial views |
| Ajaxing the gameboard |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| **What Did Not Go Well** | **Action Plan** | **Due Date** |
| Implementation of gameboard view could be made more efficient | On full page load, create an associative array that maps tile id to tile using jquery, then return list of changed tiles. Finally, loop through list of changed elements and update gameboard. | 4/8/18 |
| Code can be made cleaner | Use string interpolation over, regular string concatinations. | 4/8/18 |
| One Gameboard instance is still used for every user | Implement MultiSingleton pattern | 4/8/18 |

**Design Documentation**

**Install Instructions:**

- Clone Repo

- Open Minesweeper.sln

- Create T-SQL DB named Minesweeper

- Create Users

- Start the application

- Register

- Login

- Play Minesweeper

**General Technical Approach:**

I separated parts of the project into separate layers that each have different classes of responsibilities. The Controllers handle which views get rendered and also instantiate and utilize services to work with user authentication. The Views are meant to display information to the user, depending on which view is shown, it might also render information stored in the UserModel data instances that are passed to it.

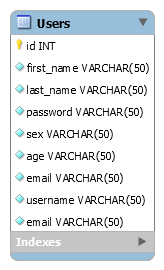
Each UI component of the gameboard is separated into another partial view. When any of these components need to be updated, they will be updated through an ajax call to the game controller. Ajax will

**Key Technical Design Decisions:**

All fields for registration are required, with some fields having specific requirements such as the State variable being restricted to two characters long (minimum and maximum).

Use of serialization to store gameboard state

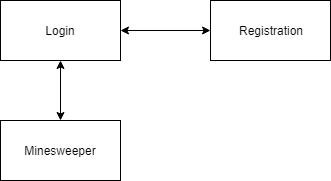
**ER Diagram:**

****

**DDL Scripts:**

N/A

**Sitemap Diagram:**



**Security Design:**

With the login and registration pages, there was a need to make the pages secure enough to prevent users from taking it apart or breaking it. The first part we did to prevent unwanted viewings of data was to use the POST submission type. This prevents users from viewing the posted material (which is why GET was not used for handling the registration and login information). Authentication for the registration process was done in two steps. The first was to verify the credentials given by the user, so that they do not contain invalid fields. Invalid fields for example could be when a user does not enter a .com or similar URL ending to the email, which would in turn render the email they entered useless to the program. Based off the specific field, the program will check the input to see if it passes the specified conditions (length being the most common parameter check). If the input for the given field passes every authentication check, the program moves on to the next field until all fields have been authenticated. Next is the second step, which takes the username given by the user and checks the entire database for any matches. This prevents multiple users from registering under the same username, which would cause many problems further down the road if this was allowed to happen.

**Third Part Interface Design:**

None

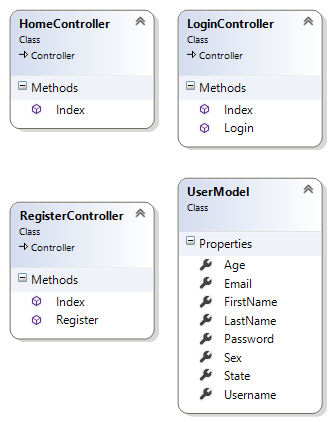
**Flow Charts:**

<https://github.com/n4n0byte/CST247/blob/master/Documentation/Flow%20Charts/MinesweeperDFS.pdf>

**User Interface Diagrams:**

<https://github.com/n4n0byte/CST247/blob/master/Documentation/StoryBoard/StoryBoard.pdf>

**Class Diagrams:**

**

**Other Documentation:**

