**.NET Application Programming**

**Project Status and Design Report**

|  |  |  |
| --- | --- | --- |
| **Topic:** | *Initial Gameboard* | |
| **Date:** | *2/7/18* | |
| **Revision:** | *1.0.1* | |
| **Team:** | 1. *Ali Cooper* | |
| 1. *Caleb Ljunggren* | |
|  | |
|  | |
| **Weekly Team Status Summary:** | |  |  |  |  | | --- | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | **Hours Remaining** | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | |
| **GIT URL:** | https://drowsyWarble@bitbucket.org/drowsyWarble/cst247.git | |
| **Loom** |  | |
| **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/ProductLog2.xlsx

**Agile Scrum Sprint Backlog:**

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

**Agile Scrum Burn Down Chart:**

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

**Agile Retrospective Results:**

|  |
| --- |
| **What Went Well** |
|  |
|  |
|  |
|  |

*The following table should be completed after each Retrospective on Things That Didn’t Go Well (Stop Doing) and What Would Be Done Differently Next Time with an Action Plan to Improve (Try Doing and Continuous Improvement). An alternative to the following table is to use a Mind Mapping tool such as Coggle. If you use a Mind Mapping tool you must include a URL or Image File.*

|  |  |  |
| --- | --- | --- |
| **What Did Not Go Well** | **Action Plan** | **Due Date** |
|  |  | 2/11/18 |
|  |  | **2/11/18** |
|  |  |  |

**Design Documentation**

**Install Instructions:**

*Step by step instructions for setting up your database, configuring, and deploying/installing your application. This section should also include detailed instructions for what configuration files are required by your application, what configuration settings need to be adjusted for various runtime (development or production) environments, and where the files need to be deployed to. This section should also contain detailed instructions for how to clone your application source code from BitBucket and deploy the application to an externally hosted site.*

- Clone Repo

- Open Minesweeper.sln

- Create T-SQL DB named Minesweeper

- Create Users

- Add necessary rows in designer

- Start the application

- Register

- Login

**General Technical Approach:**

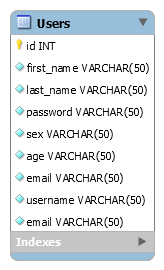
We separated parts of the project into separate layers that each have different classes of responsibilities. The Controllers handle which views get rendered and also instantiate and utilize services to work with user authentication. The Views are meant to display information to the user, depending on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

**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

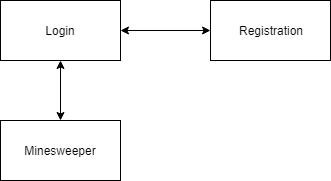
**ER Diagram:**

****

**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

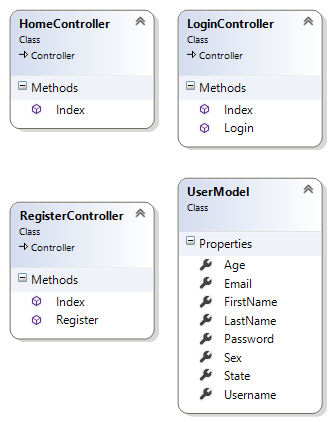
**Flow Charts:**

https://bitbucket.org/drowsyWarble/cst247/raw/53cd0e234da335b043aeb0381f2e6f717e2b26e5/Documentation/Flow%20Charts/MinesweeperDFS.pdf

**User Interface Diagrams:**

<https://bitbucket.org/drowsyWarble/cst247/raw/dc3c7d23605eb84fd2360341b16aed81f09fa4b5/Documentation/StoryBoard/StoryBoard.pdf>

**Class Diagrams:**

**

**Other Documentation:**

