## Team details

|  |  |
| --- | --- |
| Batch & Team name | MS1 Group 9 |
| Team members | Marut Prasad, M Chethan Kumar, Mohan Prabhu |
| Date | 24-09-2019 |
| Status | In-progress |

## Compiler warnings and Linting

|  |  |  |
| --- | --- | --- |
|  | Assertion | Exceptions |
|  | Metric: The warning + linting count is zero | 2 warnings are still persistent |
|  | Sustain: The ‘treat warnings as errors’ setting is enabled | No. Marut will work on resolving these 2 warnings. |

#### What linting tool is used?

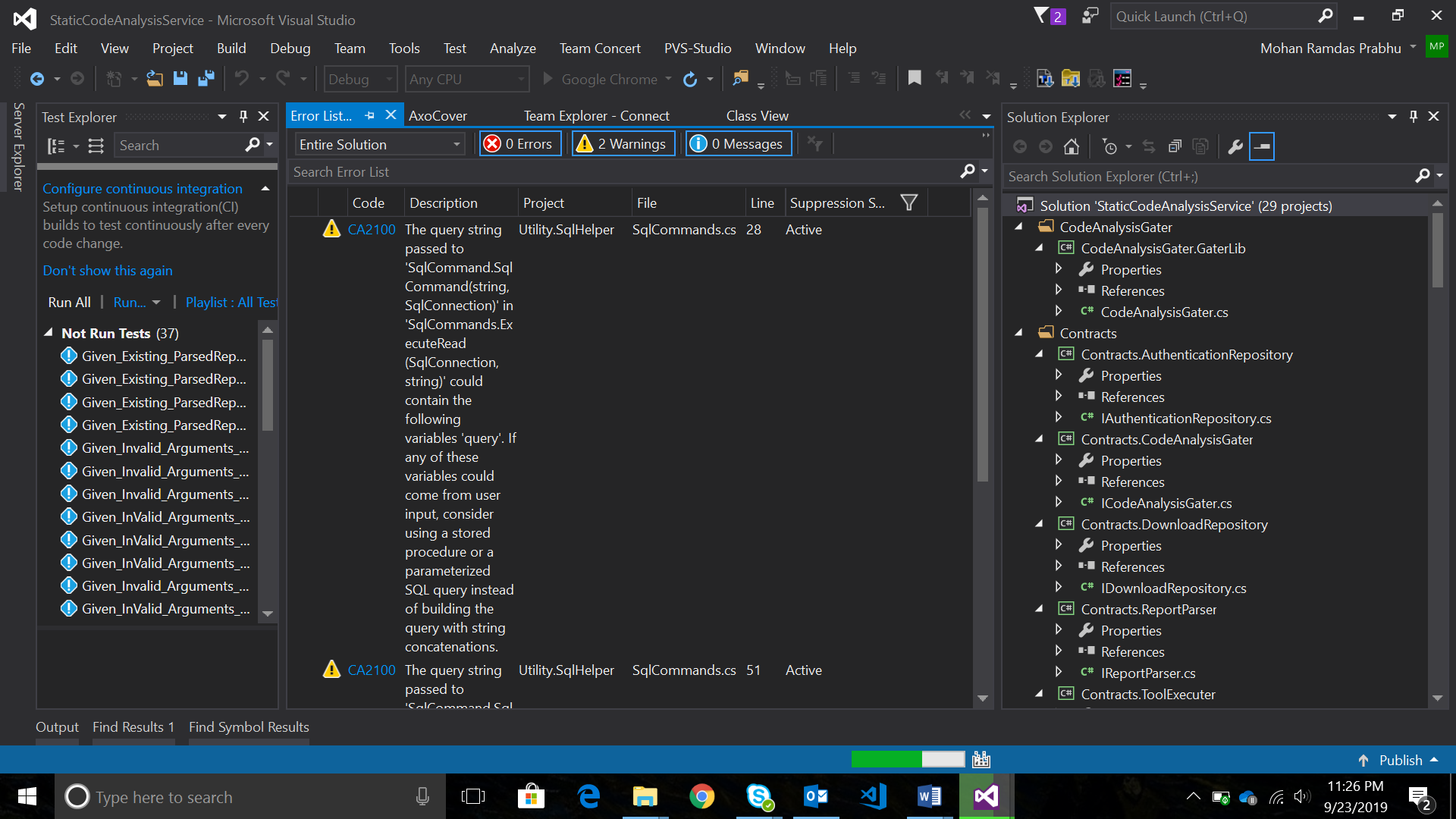
SonarLint

#### How have the warnings fared, since you took over in Phase-2?

While receiving the project, we saw 40 errors from SonarLint. We have managed to bring down the warnings to 4 warnings which state that we have to use parameterized queries. These 4 warnings were introduced by us because we built a utility class for sending queries.   
Using parameterized queries will help in avoiding data injection in SQL and make it more secure.

### Evidence

Attach/link: build & linting logs/screenshots



## Static Analysis / Coding standards

#### What is the max cyclomatic complexity of a function in your code?

3

#### Where did you reduce the complexity of the code in Phase2?

ResharperGating and ResharperService had one method each with complexity 4. We have refactored the project at set the highest allowable complexity to 3.

#### What are the instances where you improved the naming for enhanced clarity / consistency?

The solution has been re-written to make it more extensible and readable. Some of the changes that we introduced:

* The signin/signup feature was not working, and the old code allowed any new user to use the tool without signing in. The old code did not have password authentication. We have made it a session based sign in using a username-password authentication.
* We increased the readability of the code by changing the project names and class names. Previous naming convention was not in sync with the functionality.
* The received code was not covering all functionality with their test cases. It did not have negative test cases as well and also did not have end to end test cases. We have introduced new and robust test cases to cover more code and test each functionality with a positive and a negative test case.

### Evidence

Attach/link: static-analysis / complexity report

Excel Sheet Report in the folder.

## Duplication

|  |  |  |
| --- | --- | --- |
|  | Assertion | Exceptions |
|  | Metric: The duplication count is zero |  |
|  | Sustain: We have a mechanism to eliminate duplication | Mohan will implement an automatic code duplication check upon each successful build. |

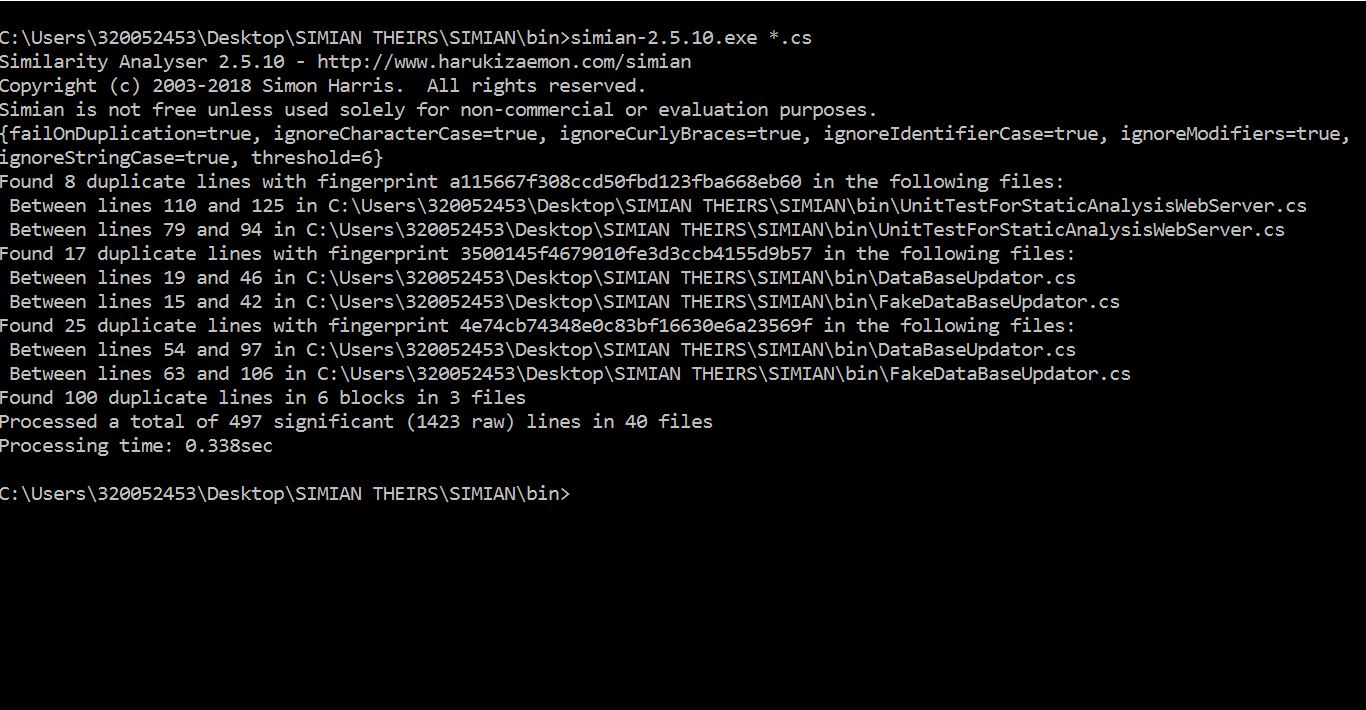
#### How much duplication did you reduce in Phase2? List the instances.

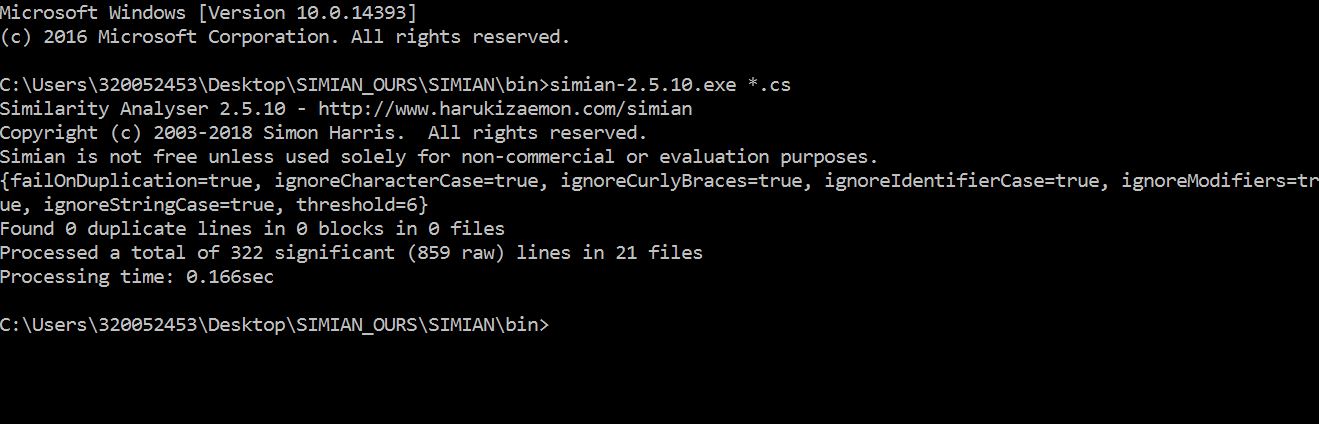
We have reduced the code duplication in all CS files from 100 duplicate lines to zero duplicate lines. The result is attached in the screenshot below. The first screenshot shows the old code duplication and the second one shows the new code duplication report.

The main instance where we reduced code duplication was in the database operations. Because of introducing a utility function, we were able to constantly talk to the database without rewriting the same piece of code over and over again.

### Evidence

Attach/link: Simian output





## Unit testing

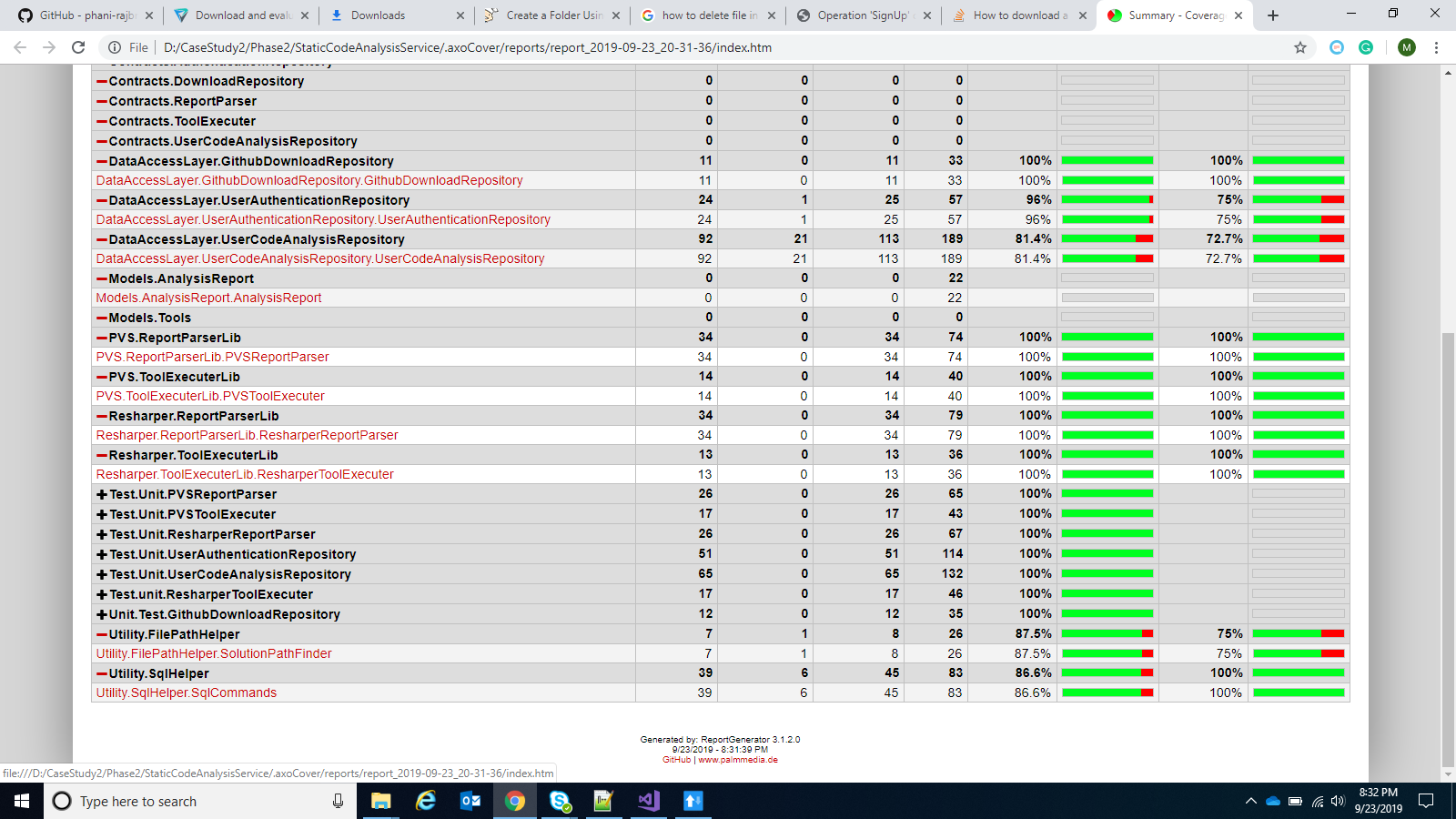
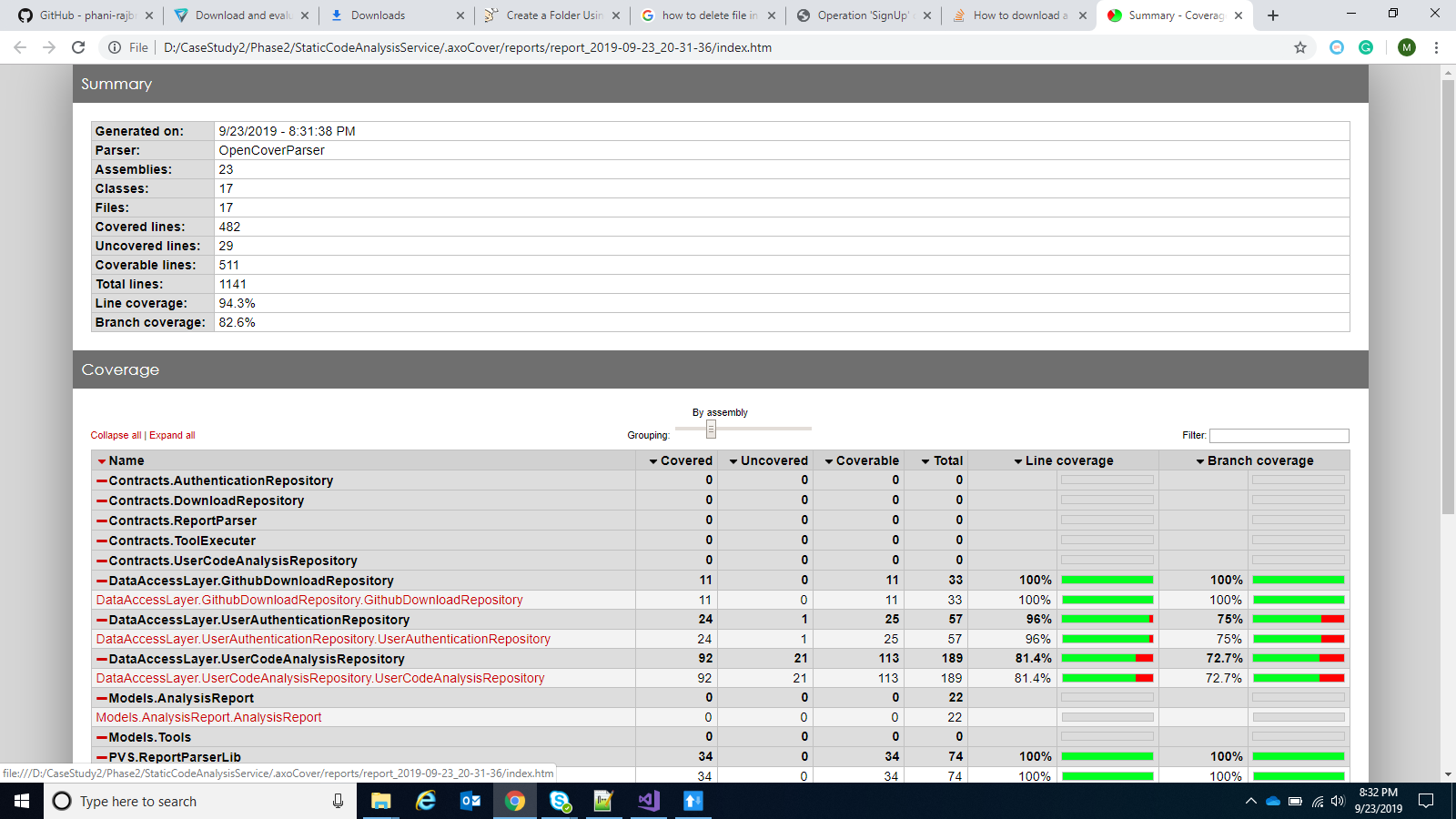
|  |  |  |
| --- | --- | --- |
|  | Assertion | Exceptions |
|  | Metric: We met our code-coverage target  (if you didn’t fix a target, take 75% as reference) |  |
|  | Sustain: We have a mechanism to enhance coverage and not let it slip. | We manually run AXO cover and check the code coverage. |

#### For each not-covered-code: What is the consequence of not-covering?

For every not covered code, it takes more effort and resources to detect faults in them at a later stage, which is UI development. The earlier stage refers to the one where the service is being built.

In case any untested line during the UI development time starts behaving unexpectedly, a lot of time will be put into refactoring the service and this might affect the already developed UI as well.

### Evidence

Attach/link: coverage report / screenshot

## Dynamic Analysis

|  |  |  |
| --- | --- | --- |
|  | Assertion | Exceptions |
|  | Metric: We have timing- and usability-related tests | None |
|  | Sustain: We have a mechanism to avoid slippage | Have not implemented yet |

### Evidence

Attach/link: timing- and usability- tests and checks

We will implement dynamic analysis after we have built the UI completely.

## Functional / end-to-end tests

#### How many end-to-end tests do you execute, to verify all the functionality?

A minimum of 2 tests per functionality. (One negative and one positive)

#### How many are automated?

All of them will be automated. As of now, 6 tests have been written

### Evidence

Attach/link: test-scripts and/or test-spec

## Retrospection

If you had to do Phase-1 again (with the same time & scope), what would you do different?

* We were unable to implement the login functionality
* Most of our functions in the service returned a void, we would have liked to make them return boolean

Since we have given a lot more options than the user stories demand, a lot of time has to be put into UI development. This hinders the development of UI for the core feature, which is patient alerting. A significant amount of time has to be put into making the registration UI and this is not favorable specially in a time critical project development.