DOWNHILL

TESTING PLAN

Version *2.0*
02/24/2021

TABLE OF CONTENTS

1. QA TEAM

i . QA Lead : Shahid Khan

a. 514-754-6900
b. shahid98@live.com

ii . Internal Tester: Ashwin Philip

a. Contact number: 438-934-6547
b. ashphil30@gmail.com

iii . Internal Tester: Juan Hoyos

a. Contact number: 438-492-3306
b. juanse00713@hotmail.com

2. Testing Procedure

2.1 General Approach

a. Basic Responsibilities of Test Team

i. Bugs

1. Detect them as soon as possible after they enter the build

2. Research them

3. Communicate them to the dev team

4. Help get them resolved

5. Track them

ii. Maintain the Daily Build

iii. Levels of Communication. There's no point in testing unless the results of the tests are communicated in some fashion. There are a range of possible outputs from QA. In increasing levels of formality, they are:

1. Conversation

2. Discord channel meetings

3. Discord channel messages

4. Email to individuals

5. Emails to group

2. Daily Activities

a. The Build

i. Generate a daily build.

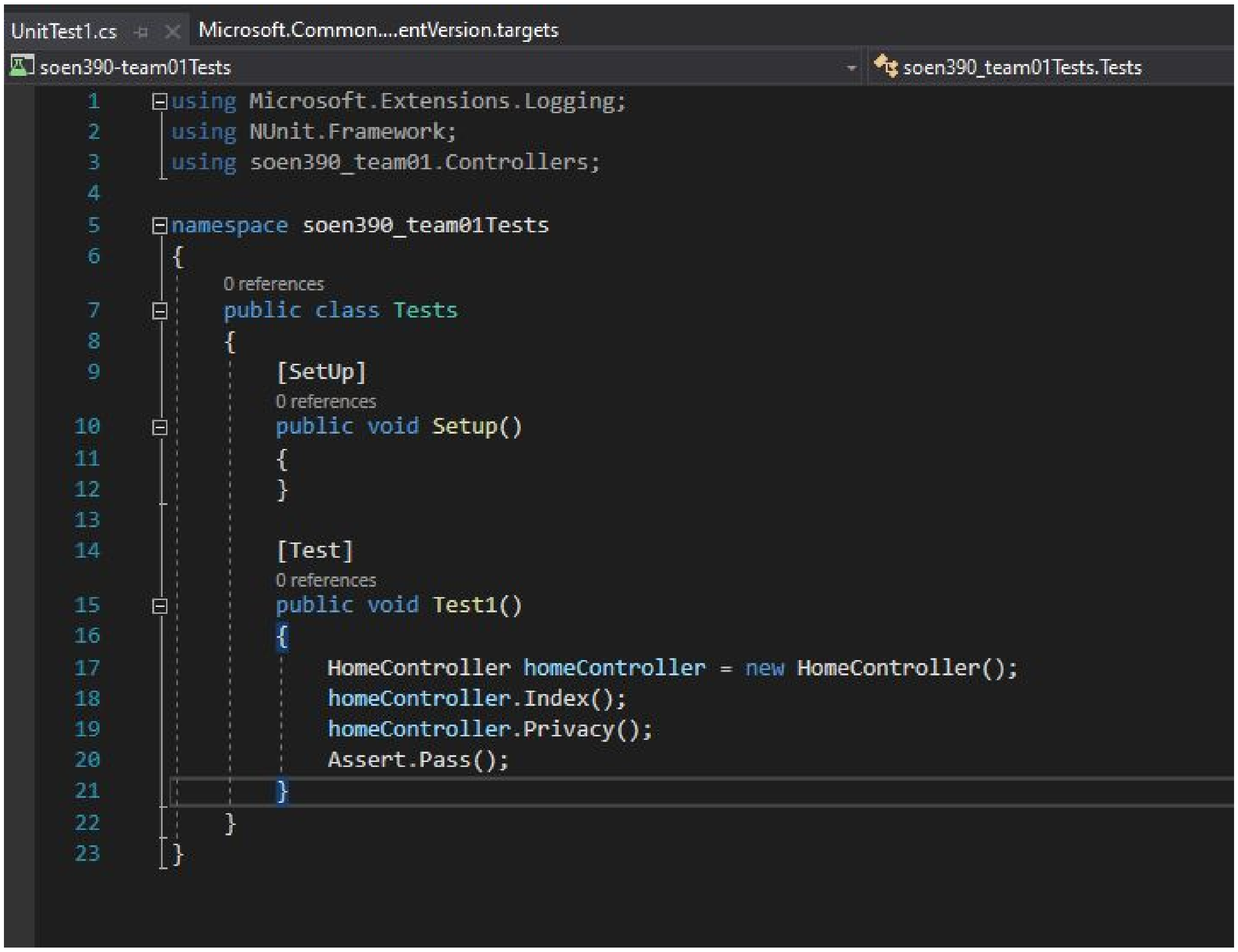
ii. If everything is okay, push to main so everyone can get it.

ii. If there's a problem, send a message on the discord channel and address the dev that is responsible for the problem and find a solution

iv. Decide whether a new build needs to be run that day.

2.2 Unit Testing

We will be using NUnit as our testing framework for our Unit tests. We chose to use NUnit since it is widely considered as one of the best unit-testing frameworks for .NET languages and a few members of our team have experience with it. We will be using version 3.13 of NUnit since it is the latest and most up to date version. It can be installed from the NUnit github page at the following link: https://github.com/nunit/nunit/releases/tag/v3.13
We have already configured this into our code and have a dummy test set up in the file "UnitTest1.cs" as seen below:


UnitTest1.cs
Microsoft.Common....entVersion.targets
soen390-team01Tests
soen390_team01Tests.Tests

```
using Microsoft.Extensions.Logging;
using NUnit.Framework;
using soen390_team01.Controllers;

namespace soen390_team01Tests
{
    0 references
    public class Tests
    {
        [SetUp]
        0 references
        public void Setup()
        {
        }

        [Test]
        0 references
        public void Test1()
        {
            HomeController homeController = new HomeController();
            homeController.Index();
            homeController.Privacy();
            Assert.Pass();
        }
    }
}
```

Another reason we chose to use NUnit is due to the suite attribute which allows for aggregating of tests to execute separately which is useful in projects like ours.

All new components that are added must have their corresponding unit tests before they can be added to the project.

2.3 Test Execution on Pull Requests

We will be using SonarCloud for code analysis and to detect quality issues upon pull requests being made. SonarCloud uses the static code analysis approach to allow for easrly detection of issues in order to increase the overall quality of the code. We already have SonarCloud configured with our repository on GitHub.

There are 3 kinds of issues that SonarCloud detects:

Code Smells: These are characteristic of the code that, while not actually preventing the proper functioning of the program, may indicate deeper problems that negatively affect the maintainability of the code. Early identification of these types of issues can help to alleviate technical debt in the application.

Bugs: These are errors in the code that can prevent the program from operating as intended. They affect code reliability.

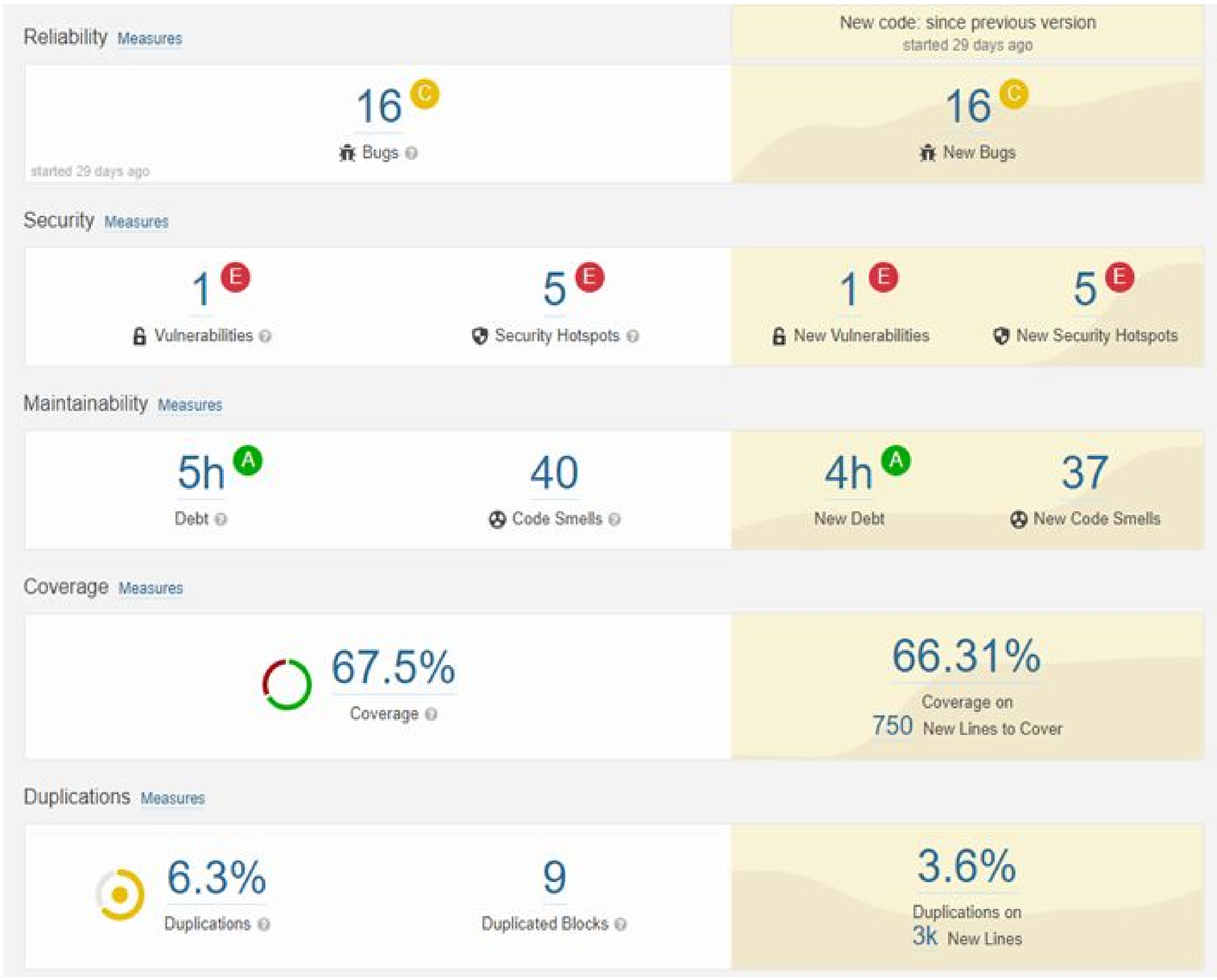
Vulnerabilities: These are problems in the code that could be exploited by a bad actor to compromise the security of the application.

As mentioned previously, this is already configured on our GitHub repository and can be verified.

2.4 Code Coverage

After further discussion, we decided not to use the Coverlet library since we are already using SonarCloud and it provides good data for code coverage as well.

Here is a report from SonarCloud, showing our code coverage for sprint 2:



Reliability Measures

New code: since previous version
started 29 days ago

16 C
Bugs

started 29 days ago

16 C
New Bugs

Security Measures

1 E
Vulnerabilities

5 E
Security Hotspots

1 E
New Vulnerabilities

5 E
New Security Hotspots

Maintainability Measures

5h A
Debt

40
Code Smells

4h A
New Debt

37
New Code Smells

Coverage Measures

67.5%
Coverage

66.31%
Coverage on
750 New Lines to Cover

Duplications Measures

6.3%
Duplications

9
Duplicated Blocks

3.6%
Duplications on
3k New Lines

2.5 UI Testing

In order to test our User Interface, we have implemented a set of steps that should be followed in order to ensure that everything is working fine on the front end. These steps must be followed at any moment when the project is being tested or when any new changes are added that could affect the UI functionality.

For this sprint, we focused on the UI tests for the login page and User Management page:

2.5.1 Login Page Procedure:

TC 1 - Verify that the Welcome and Login boxes are centered.

TC 2 - Verify that the Login button is blue and the Forgot Password button has no color.

TC 3 - Verify that the Login button is and the Forgot Password button are aligned with each other and adjusted to the left of the Log in box, under the text fields.

TC 4 - Verify that the email and password text fields are clickable and text may be written.

TC 5 - Verify that the forgot password button is clickable and redirect the user to the ForgotPassword page.

TC 6 - Verify that when the "Back to Login" button is clicked, the user is returned to the login screen.

TC 7 - Verify that when an invalid email is entered (no @ sign) and Login is clicked, an error will indicate that the email is invalid.

TC 8 - Verify that the "Login" button is clickable and redirects the user to the Home page of the application when a correct email and password are entered.

TC 9 - Verify that when "Login" is clicked and the "Email" field is empty, the appropriate error message is displayed.

TC 10 - Verify that the error messages are displayed in the proper color: red

TC 11 - Verify that upon attempting to login with an incorrect email or password, the login attempt fails and the user remains on the login page.

TC 12 - Verify that when "Login" is clicked and the "Password" field is empty, the appropriate error message is displayed.

2.5.2 User Management page Procedure:

TC 1 - Verify that the First Name, Last Name, Phone Number, Role, Email, Password and Confirm password fields are present and clickable.

TC 2 - Verify the Add User button is clickable and and adds a user to the table upon having valid inputs.

TC 3 - Verify that the user table exists and displays all the required fields.

TC 4 - Verify that the Edit button is clickable and opens an edit menu.

TC 5 - Verify that the Update button is clickable and updates the user table.

TC 6 - Verify that the Add user, Edit and Update buttons have the right color: blue

TC 7 - Verify that once a user is edited, the change appears on the table.

TC 8 - Verify that when a user is edited, they are still able to be modified/edited again.

TC 9 - Verify that when an email is entered, it is validated as done in the login page (See login page Test Case 7).

TC 10 - Verify that when an invalid user is added, they are not added to the table and an error is displayed to the user.

TC 11 - Verify that when a user is added, it is still possible to add more users.

TC 12 - Verify that a user cannot be added if there are fields missing.

TC 12 - Verify that a user cannot be added if the Confirm Password field does not match the Password field.

TC 12 - Verify that the user table displays the information in the correct order.

3. Weekly activities

3.1 Weekly Tests

i. Build and run the most up to date version of the project. A specific set of activities should be performed in order to maintain consistency. For the same reason, the same machine should be used to perform the weekly tests.

ii. Weekly review of bugs:

- Verify that "fixed" bugs are really fixed and do not persist.
- Rank bugs relative to their urgency as well as the progress of the project as a whole.

- Generate a weekly report of fixed bugs.

4. HOW TESTING REQUIREMENTS ARE GENERATED

1. Some requirements are generated by this plan.

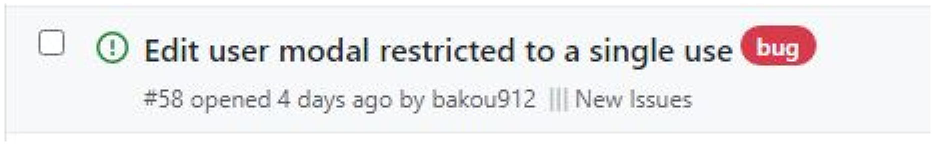
2. Requirements can also be generated during project meetings, or other formal meetings held to review current priorities.

3. Requirements can also result from changes in a bug's status within the bug tracking system. For example, when a bug is marked "fixed" by a developer, a requirement is generated for someone to verify that it has been truly killed and can be closed out.

5. BUG TRACKING

1. We will keep track of bugs using github issues. We have created a "bug" label to differentiate the issues related to bugs to the rest of the tasks in the project.

Here is an example of a way we keep track of bugs on github:



Edit user modal restricted to a single use bug

#58 opened 4 days ago by bakou912 New Issues

In addition to the label, we add a bug description and details about the bug.

The following 3 sections are in each bug report:

a. Describe the bug

The bug in question is explained in detail, specifying which part of the code/project is broken and the components involved.

b. How to reproduce the bug

Steps must be included in order for a dev or anyone else to be able to reproduce the bug in question. These steps must be clear and reproducible.

c. Expected behavior

The way the program SHOULD be responding to the actions performed in the steps. This is what should be produced when the bug is resolved.

d. Screenshots

Screenshots can be added to simplify the process for the dev. They make things much more clear by showing exactly what the bug looks like.

Here is an example of a bug report from this sprint:

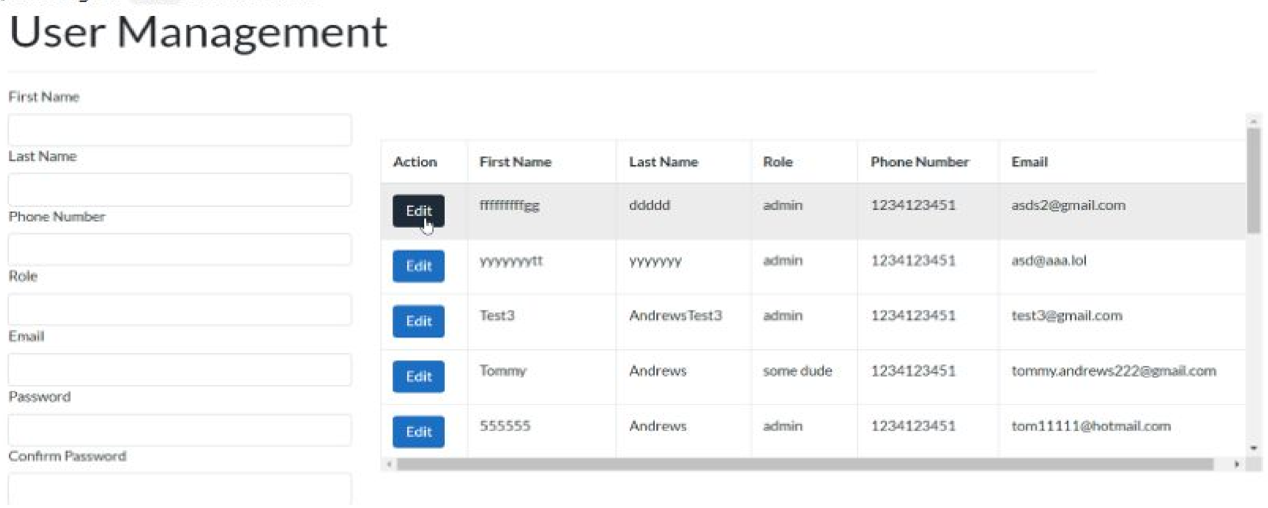
Describe the bug
In the `User Management` module, updating a user once breaks the `Edit` modal. After the modal is closed on the first edit, clicking on the `Edit` button again does not make the modal visible again, preventing the module user from modifying the entry multiple times.

To Reproduce
Steps to reproduce the behavior:

1. Start the application
2. Sign in
3. Go to the `User Management` tab
4. If not there is no user in the table, use the `Add User form`
5. Click on the `Edit` button for a user in the table
6. Modify a field's value in the modal and click on `Update`
7. You can now no longer bring back the `Edit` modal

Expected behavior
After the first usage of the `Edit` modal, the module's user should be able to re-use it how many times they want on the same table entry.

Screenshots
5) Clicking on `Edit` the first time


User Management

First Name
Last Name
Phone Number
Role
Email
Password
Confirm Password

Action	First Name	Last Name	Role	Phone Number	Email
Edit	ffffffffffgg	ddddd	admin	1234123451	asds2@gmail.com
Edit	yyyyyyytt	yyyyyyy	admin	1234123451	asd@aaa.lol
Edit	Test3	AndrewsTest3	admin	1234123451	test3@gmail.com
Edit	Tommy	Andrews	some dude	1234123451	tommy.andrews222@gmail.com
Edit	555555	Andrews	admin	1234123451	tom11111@hotmail.com

2.

a. Who assigns the bug?

The bug can be assigned by any member of the team, normally the issue will be created by whoever finds the bug.

b. What happens when the bug is fixed?

Since the bugs are reported as issues on github, once they are fixed, a pull request can be made and a review of the code will be done.

c. What happens when the fix is verified?

Once the fix has been verified by a dev and the code seems good, the pull request will be accepted and a merge will be done.

6. SCHEDULING

6.1. Rotation Plan

Throughout our project's life cycle, different people will perform tests in order to ensure that we always have "fresh eyes" on the project. Usually, the dev that worked on a section will test their code themself and then assign another team member to verify their code. This ensures that it is not always the same person looking at their own code.

7. EQUIPMENT

1. QA Team Personnel with Hardware and Software Toolset

- **a.** Ashwin Philip
  - **i.** Hardware
    - **1.** Testing PC
      - **a.** 16GB RAM, 3.6GHz CPU
- **b.** Shahid Khan
  - **i.** Hardware
    - **1.** Testing Laptop
      - **a.** 16GB RAM, 3.5GHz CPU
- **b.** Juan Hoyos
  - **i.** Hardware
    - **1.** Testing Laptop
      - **a.** 16GB RAM, 3.5GHz CPU
- **b.** Noah Freger
  - **i.** Hardware
    - **1.** Testing PC
      - **a.** 16GB RAM, 4.0 GHz CPU