Assignment 5: Testing Policy

Realized by
Saif Ul Islam (18K 0307)
Muhammad Anas Aziz Siddiqui (18K 1090)
Muhammad Taha (18K 1270)

As part of the course
Software Engineering

Work presented to
Sir Farrukh Hasan Syed

To The Department of Computer Science
Faculty Of the Department Of Computer Science
National University Of Computing And Emerging Sciences
Date of delivery, May 28, 2021

Contents

1 Question

You are required to write the testing policy of your project. Make sure that the testing policy is sufficient for your project and take into account your project type and associated issues (See Sections 1.1.1 and 1.1.2 of Sommerville to figure this out), and including important functional and non-functional requirements.

2 Categorization Of Software

"Reflex", the project being built for the course, "Software Engineering", falls primarily into the context of a *"Interactive Transaction-Based Applications"*, on the basis of the following observations,

1. Some sort of service is provided which is mainly dealt through the interactive modules on the web application
2. The system's emphasis is not prepossessing large amounts of data, rather that to just store and retrieve information
3. The system will be designed as to make very easy sense to the average person, thus no special knowledge or experience required
4. The system is meant to be designed with easiness, and user interaction in mind
5. The goal of the application is to inform and act as a bookkeeping mechanism, not as (**a**) An entertainment system; (**b**) For modelling complex graphics, statistics, or computations; (**c**) Data Collection mechanism, as it does not interact with other data intensive applications, neither does it export any data anywhere, except its own database; (**d**) Not really a system which contains subsystems, for example, like ERPs

3 Engineering Principles

3.1 Acceptability

Acceptability being a large part of this project to model it as closely as per the requirements of the client, and to work in more depth/detail for the intents of fulfilling the individual function and non-functional requirements.

3.2 Dependability & Security

It is important that the data is stored without compromises to security. As the application is at the moment developed locally, there is no threat of distributed system attacks, web based attacks, or insecure networking protocols. For some measure, the "**bcypt**" cypher mechanism will be used to store the passwords securely, locally, in a hashed fashion.

3.3 Efficiency

It is important to make the system as easy as possible to use, with the intent of making it very simple for users to effectively and quickly reach their goals. Thus, efficiency and user experience is another

metric here.

3.4 Further Factors

3.4.1 Business Changes

On the **functional** aspects, the UI is designed so as to make it easier to further add newer components. Keeping in mind these possible suggestions by the addressed clients. A UI model has been considered that it makes it easier to add, remove, update, and maintain newer features and further developments.

On the **non-functional** aspects, the code will be designed keeping in mind the following factors,

1. Scalability of the database
2. Single Responsibility
3. Separation Of Concerns
4. Modularity
5. Minimal Dependencies
6. Documented sections to explain certain code

4 Testing Strategies

4.1 Testing "Direction"

As far as the testing direction is concerned, it is mainly of concern that we focus on the principles as below,

1. Making small functions that deal with only thing
2. Send manually suggested inputs to test if the right outputs are considered. For this, error checking is done within the code to see if the outputs are of the right format. It can considered, but hasn't been decided on how to, if it is decided later on to add non-functional tests to see the correct outputs to the required inputs
3. If the desired function is working as needed with the given requirements, it is integrated
4. If more units are needed, they are built the same way, as if like in a pipeline. They will receive certain expected inputs from some data source, they will interact with it in someway, and then either pass it on to some other unit, or pass it back to the source
5. Generally, code to test all of this has not yet been written, but it can be considered to do so

4.2 Code

The code will be,

1. Formatted and linted with ESLint to make sure no possible bugs arise on the developer's end
2. Testing manually by the developers after the integration to make sure the feature is working as expected
3. Based on the principles as described in 3.4.1

4.3 Design

The design will be,

1. Asked about in an abstract, questioning way from the clients about what they (**a**) Expect the interface to do; (**b**) How can or should they interact with it; (**c**) What the flow of the integration will be; (**d**) Why do they need this specific design

After elaborating on those ideas, it is then more clearer what the user actually wants. After the aforementioned process, the design will be prototype, shown to the clients, get it approved, and then the work will start.

Through an informal process amongst the developers, the following questions will be asked,

1. Does this design actually fulfil the user's requirements?
2. Is this design easy to use?
3. Is this design intuitive?
4. Does it follow modern design principles?
5. What use case does this relate to?

Due to a lack of time, and a bigger team, the above informal process might not be documented, and there is no external testing team to make sure these ideas are acted upon. They will surely be **"tested"** to again, give a direction to the development team if they on the right track.

4.4 Requirements

Before starting any feature, a GitHub issue will be created that describes either a,

1. Bug Report
2. Feature Request
3. General Request

This is done so to keep a short documentation in an agile fashion, while also serving as a way to describe requirements of the project, what gets assigned to whom, what blocks what feature/functionality, and what obstacles arise to the requested issue.

For each of the above types of issue, a template is used.

4.4.1 Bug Report - Template

```
Describe the bug A clear and concise description of what the bug is.

To Reproduce Steps to reproduce the behavior:

Go to '...'
Click on '....'
Scroll down to '....'
See error
Expected behavior A clear and concise description of what you expected to happen.

Screenshots If applicable, add screenshots to help explain your problem.

Desktop (please complete the following information):

OS: [e.g. iOS]
Browser [e.g. chrome, safari]
Version [e.g. 22]
Smartphone (please complete the following information):

Device: [e.g. iPhone6]
OS: [e.g. iOS8.1]
Browser [e.g. stock browser, safari]
Version [e.g. 22]
Additional context Add any other context about the problem here.
```

4.4.2 Feature Request - Template

```
Is your feature request related to a problem? Please describe. A clear and concise
    description of what the problem is. Ex. I'm always frustrated when [...]

Describe the solution you'd like A clear and concise description of what you want to
    happen.

Describe alternatives you've considered A clear and concise description of any
    alternative solutions or features you've considered.

Additional context Add any other context or screenshots about the feature request here.
```

4.4.3 General Request - Template

```
Is your suggestion related to something in the UI design of the website? I wish this page
     had this feature where I could click on this XYZ button and ABC would happen ...

Is your suggestion related to the database design? We should modify the database schema
    or flow to fit in this use case. That use case is ...

Describe the solution you'd like A clear and concise description of what you want to
    happen.

Describe alternatives you've considered A clear and concise description of any
    alternative solutions or features you've considered.

Additional context Add any other context or screenshots about the feature request here.
```

5 Testing Methodology

5.1 Bottom Up Approach

For this aspect, the approach we take or will take is similar to *"Bottom Up Approach"*, where units will be completed on their level, then interconnected by the correct passing of data, then validated by integrating into the system as a whole.

Thus, the idea is to make logical level as self contained as possible, that when together, build a product that indicates the complete interaction of different units merged into one functionality. For example, the dashboard pages consists of individual pages, which a left sidebar, and a main logic section. The sidebar can be worked separately without disturbing other already built components. The process can simply be,

1. Adding a new button
2. Adding a new page to redirect to when the button is clicked
3. Adding an animation on the button on being clicked
4. Making sure it works by manually moving from the new button to other buttons, and back
5. Then introduce whatever logic is needed in the main logic section

5.2 Smoke Testing

To connect the entire application, **Docker** will be used to facilitate building the application entirely on container like model, with each container being it's own logical application that can interact with other application. Something failing can immediately be seen on the following levels of this styled application,

1. ***Top Level, Front-End*** - Logs printed here can be investigated in the Dev Tools in any modern web browser. Messages can be found in the "Console" part of the DevTools as printed messages from the Front-End
2. ***API Level, Front-End*** - This is the layer that communicates with the Back-End. Any messages here will be printed by Docker in the log stream that will most likely be in a console/terminal
3. ***API Level, Back-End*** - Error messages printed here will most likely be sent up to the layers above, but any exception caught and printed will be shown in the Docker stream log, as before
4. ***Business Logic Level, Controllers, Back-End*** - Same as above
5. ***Database Level*** - Any error messages here will be critical, and will make the database containers stop working. This will make the errors be logged in the Docker stream log

6 Misc Testing Ideas

6.1 Independent Testing Group (ITG)

Sadly, due a lack of time and manpower, the testing will be done by the developers themselves rather than a testing or a SQA team.