QM Lab Course Project

Requirements Document

Group 3

Angela Stankowski (ans723), Bengin Lee (bel529), Connor Nettleton-Gooding (cwn973), Corey Hickson (crh208), Darvin Zhang (ddz369), James McKay (jlm012), Jordan Wong (jtw289), Mack Mahmoud (mtm464), Matt Hamilton (mch986), Michael Kelly (mlk121), Michael Ruffell (mar492), Mitchell Lau (chl929), Royce Meyer (brm979), Shane Williamsom (saw056)

April 7th, 2016

I. Introduction

In this document, we will look at the different requirements we have set out for the project commissioned to us from Nathaniel Osgood and Geoff MacDonnell. QM Lab will be a qualitative modelling tool with collaborative functionality.

To accomplish this, we have set out the scope that we believe this project encompasses, defined various actors we believe will be interacting with the system, and determined the way in which we believe those actors will interact with the system.

As well, we have determined how we believe the flow of our system will work with an activity diagram, and created storyboards which will allow us to visually model what we want our product to look like.

The following part of this section is an addition for ID3.

Since the first deliverable, we have further understood and realized the requirements of this project. We have documented this by creating new and more accurate storyboards, added in missed requirements, and focused on filling out incomplete requirements.

The following part of this section is an addition for ID5.

For the final deliverable, a small number of clarifications were made which have been reflected in the changes below.

II. Background

In January of 2016, our group was commissioned by Nathaniel Osgood and Geoff MacDonnell to design an application, termed by our group as QM Lab, which would serve as a qualitative and collaborative modelling platform for users to create diagrams of various systems.

The need for this project does not come from the lack of modelling tools, but the lack of modelling tools which allow real time collaboration a la Google Docs. Therefore, a major portion of this project will be combining existing technologies and architectures to meet this need.

The schedule for this project is **five** incremental deliverables dated for February 7th, February 24th, March 8th, March 4th, and April 7th of 2016, respectively. We will complete this project with a group of 14 members broken up into a design team, implementation team, testing team, triage team, as well as a build project lead, integration officer, and risk officer.

The following part of this section is an addition for ID3.

Since the second deliverable, we have decided to work with a more flat group structure to have a better workflow and produce better results (with more effective communication). While we still intend to have team leads, there will be more fluid group members between the teams (such that they have no defined role).

III. Scope

For our project, we have defined the scope based on must-haves, should-haves, and nice-to-haves. These are expanded below.

A. The software must:

- Have the ability to support multiple users accessing the system through web browsers and interactively/simultaneously editing/adding components to diagrams
- To create, save, and load projects
- Tools/elements needed to support creation of UML, flow and stock diagrams, as well as support text
- Able to create agent objects which can hold other objects relationally
- Adding in one of a small number of shapes and images
- Create variables and links between components
- Basic ability for anyone to create a project and generate an editable/viewable link

B. The software should:

- A login system - with user control administration (such as the creator of a project would have the rights to add and kick certain members in/out of a project - or a set of project (creation of rooms))
- A log of who edited what at certain points - potentially save state/backups
- Power to select a set of the diagram, not only the individual parts (component and the links attached to it)
- Components that contain other components
- Ability to resize and color the elements for a diagram (and the diagram itself)
- Change the font/style of text within components
- **(ID3)** Be able to print the model
- **(ID3)** Be able to export as a persistent JSON or text framework

C. It would be nice if the software could:

- Customized components such as a user specified picture is added and named to the elements available for a certain project/team
- Customizable overlay for the software
- Edit profile and view other profiles (friends list)
- A chat system - private chat
- General templates to guide users
- Differents styles of components (different looks available for say a UML class diagram)
- Undo function possibly based on history
- A tutorial
- Support of video maybe a slideshow/add pages to a project

The following part of this section is an addition for ID3.

Certain functionality came up during a stakeholder meeting in January, which included having printing (with correct formatting), presumably in a way current state of the art implements it, as a function in a dropdown menu (or through CTRL+P). This is now included as a **should-have**.

There was also the request of persistent JSON format or text framework, and this has also been included in the **should-haves**. This would be ideal as an exportable file the user can access.

The follow part of this section is an addition for ID5.

Clarification was given by the stakeholder on a variety of visuals, such as the three different types of connections that can exist. This clarification is enumerated below.

There are **links**, which are simple, solid curved lines. They do not have labels, but should be able to have a +, -, or ?.

There are **transitions**, unique to state charts, they are just solid straight line that connect two nodes. They can have text labels, anchors (ie. pivot points), with little arrows.

There are **flows**, which are pipes and have a valve on them. They use larger arrows, and have text labels near the valve. They can go perpendicular or horizontal to the side of the screen. Time permitting, they should exclusively connect stocks.

In addition to the connections, the **chaining of clicks** was described. If you click an empty space as some type of source, start the connection. If it is a flow, that connection starts as a cloud, otherwise (for a link or transition), anchor it to the space. If you click anywhere else, it will make an anchor point for the connection. Specifically, a single click, even on a node, will not connect to the node, but make a new anchor. Finally, to end the connection, you double click. This will either make an end cloud, connect to a node, or terminate at empty space.

For the **agent**, the figure should be a head and torso (no legs) in a lighter colour.

Another node, **interventions**, was described. An intervention is a thunderbolt through a red circle (with text label). This is essentially a different visualization of the parameter or variable.

A few colour clarifications were also described. **Variables** should be orange and **parameters** should be blue (in addition to their circles). The **final states** are red circles with a dot, and the **text boxes** should not have a dialogue tail, with the ability to make the background white.

IV. Actors

For our system, we have identified a number of actors. Actors are abstract entities which interact with different portion of our system and are described below.

User(s): The user (or client) is the primary actor that interacts with the system and its components. The user needs to interact with the system to (along with other users) create, load, save, and edit projects within the system while also allowing other users to interact cooperatively on a project.

Server: The primary function of the server is to store, process and deliver web pages to clients who request these needs. If a process cannot be made, it must output an error. The server must also support multiple users interacting collaboratively on a given project within the system.

Database: The database (although still undecided if it is actually needed in this project) is part of the server, but distinct. When certain data is requested by the client through the system to the server, the database needs to allow access and retrieval of the data or to store certain information.

~~**Interface:** The interface is an abstract concept that reacts with our system. For instance, If the user was editing the diagram the interface would be specifically WHAT the user is doing. The interface might create an object or link two objects together depending on what the user is doing within the program at any given moment. So in a general sense if the user was simply adding, editing and creating diagrams, the interface would be what is happening or how it is being done.~~

The following part of this section is an addition for ID3.

For our actors, we have decided to get rid of the ‘interface’ actor, reevaluate the server as the Google server, reevaluate the database as Google Drive, and consider the Google Realtime API as an actor.

Originally, we intended the ‘interface’ actor as an abstract idea which handled all of our events, but upon reconsideration, this ‘interface’ was really our system controller and calling it an interface was a misnomer.

Although the Google products are not required by the stakeholder, because they are so integral in our program, we have decided to list them within the requirements because they set many constraints for the product we are creating.

These changes are outlined below.

User(s): The user(s) have remained relatively unchanged, except that to the previous requirements, the user(s) must be able to print off the model and be able to get a JSON (or text framework) representation of the model.

Google Realtime API: The Google Realtime API is essentially a tool which interacts with our system through function calls. It is an actor because it will be handling and accepting/pushing collaboration calls that will connect users within the system.

Google Drive: Google Drive accepts information from the system and stores it allowing for persistence, authentication, and future handling of authentication (automatically).

Google Server: The Google server handles calls from the Google Realtime API and manages the Google Drive interaction (in a style similar to black magic). While it interacts with through our API and Drive, it is actually a large part of the application layer of the product to be discussed further in the design document.

V. Use Case Diagrams

For the first incremental deliverable, the use case is rather simple, as diagramed below. The only real requirement for an actor currently is to view a web page. This provides proof of concept that our technologies can connect. The remaining proof of concept come in the form of spike prototypes which will be thrown out later.

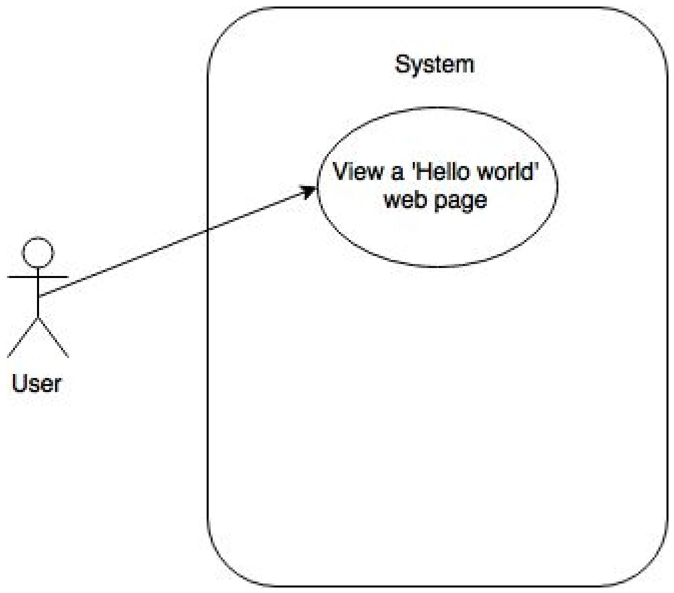

Figure 5.1: Use case diagram for the first incremental deliverable

For our the future, we have also created a potential use case diagram, below, of what we would like the users to ultimately be able to do.

One of the important distinctions on this diagram is the nuance between a user and other users. Other users should not be able to access and load diagrams of the original user. That is, a user can only access their own diagrams, and not access diagrams for which they do not own.

Other than that, we have generalized the interactions which a user would take. Due to the complexity of the actual interface and how it works, this has been saved for another diagram showing off the interface as an actor which interacts with objects.

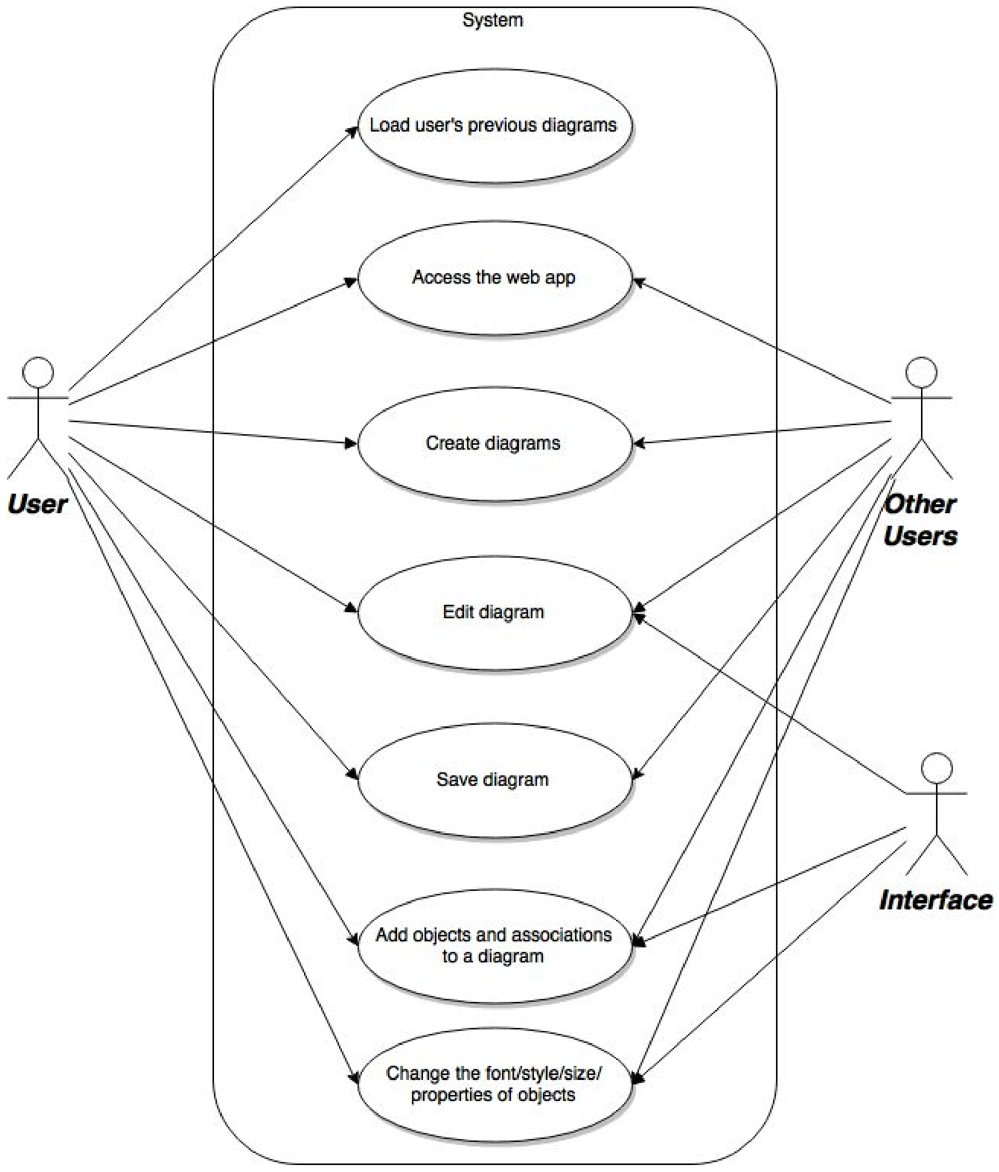
System
Load user's previous diagrams
Access the web app
Create diagrams
Edit diagram
Save diagram
Add objects and associations to a diagram
Change the font/style/size/ properties of objects
User
Other Users
Interface

Figure 5.2: Use case describing the actors and interactions looking beyond ID1

Lastly, we have created a use case for the abstract idea of an interface, seen below. The point of this was to better specify exactly what the interface with the system, so we could broadly say the user was simply adding, editing and creating diagrams.

This use case would similarly extend to the user, as they should be able to create all of the things listed in the use case, except with the interface as an actor, we can specify requirements such as moving while maintaining association between sets of objects. Without the relationships and associations between objects, the project loses a lot of value.

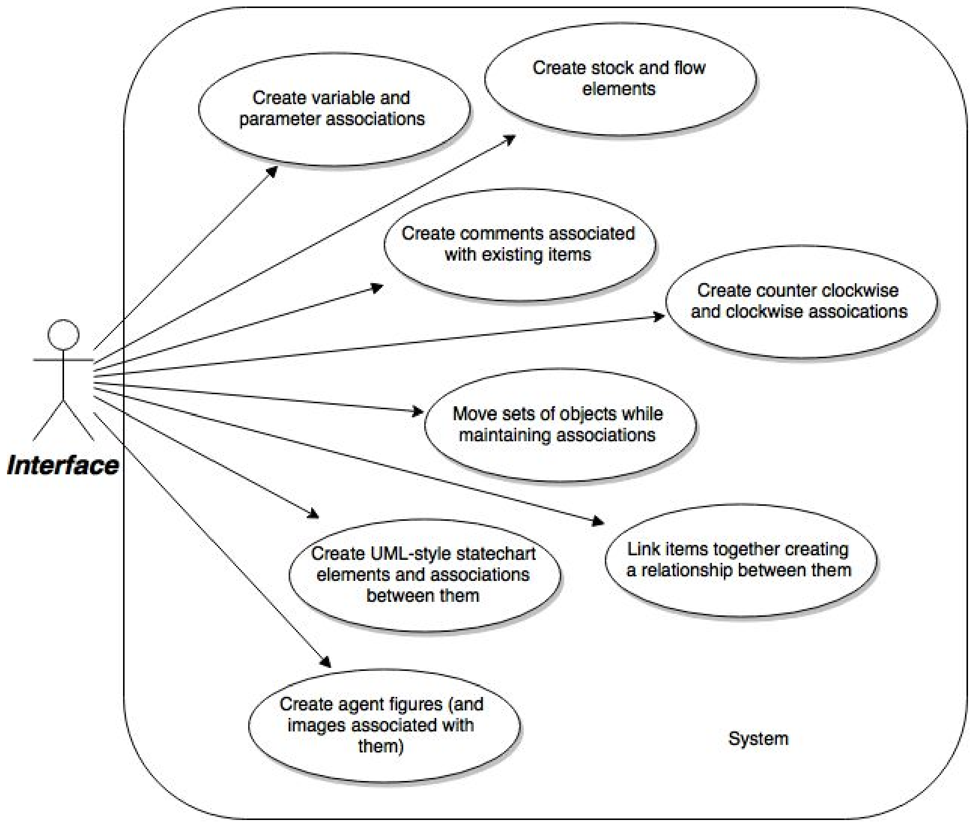
Create variable and parameter associations
Create stock and flow elements
Create comments associated with existing items
Create counter clockwise and clockwise assoications
Move sets of objects while maintaining associations
Link items together creating a relationship between them
Create UML-style statechart elements and associations between them
Create agent figures (and images associated with them)
System
Interface

Figure 5.3: Use case diagram of an abstract interface actor and how it interacts with the system

Lastly, we have included for the sake of completeness, a use-case diagram with our database as an actor. Currently, the database system is relatively unexplored and not even certain if it is needed, but it could be a potential actor interacting with the system depending on how we use it.

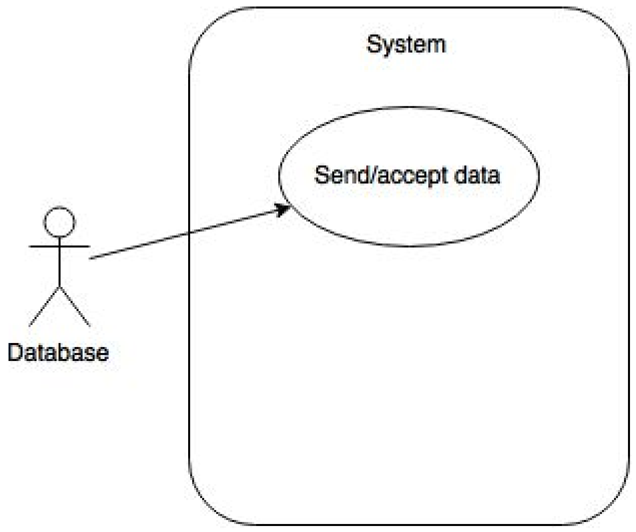

Figure 5.4: Use case of the potential database

The following part of this section is an addition for ID3.

We’ve updated the user use-case diagram to reflect more of the options as seen in the figure below.

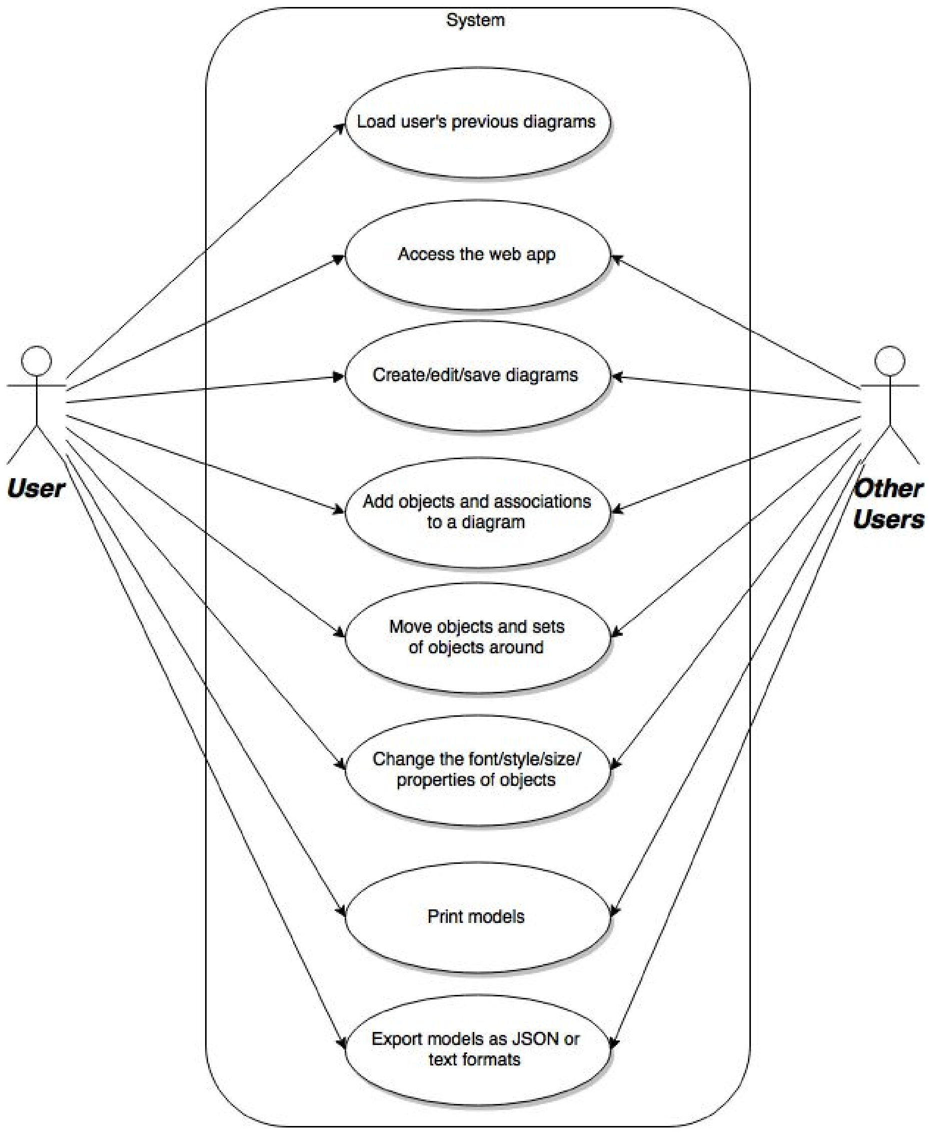
System
Load user's previous diagrams
Access the web app
Create/edit/save diagrams
Add objects and associations to a diagram
Move objects and sets of objects around
Change the font/style/size/ properties of objects
Print models
Export models as JSON or text formats
User
Other Users

Figure 5.5: Use case for ID3 users

In addition to the new use-case for the users, we have created a use-case which aims to interpret how the Google Realtime API and Google Drive will interact with the system (seen below).

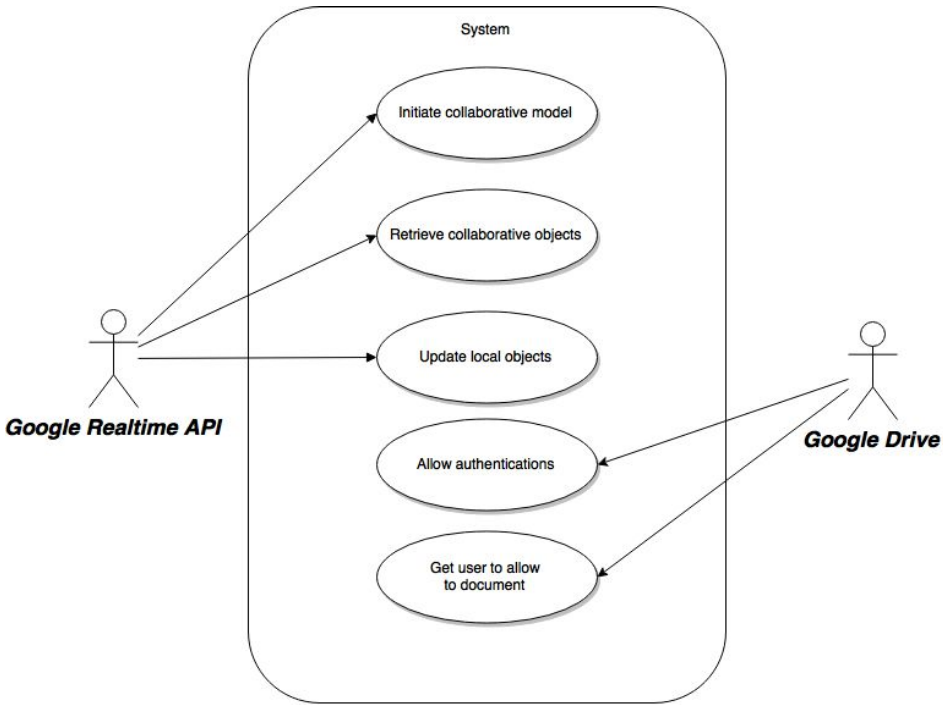
System
Initiate collaborative model
Retrieve collaborative objects
Update local objects
Allow authentications
Get user to allow to document
Google Realtime API
Google Drive

Figure 5.5: Use case for ID3 users

VI. Activity Diagrams

For the project, we have outlined the general flow of the program. This is from the perspective of a user using our web application. This can be seen below.

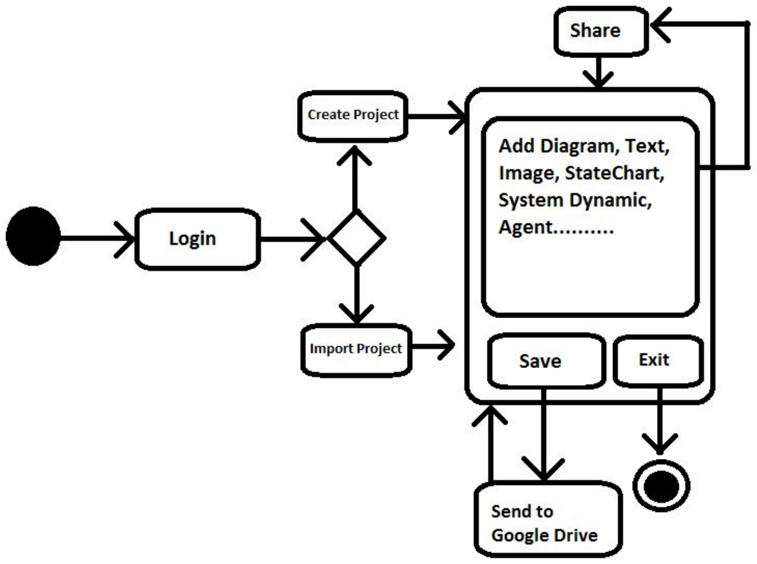
Share
Create Project
Add Diagram, Text, Image, StateChart, System Dynamic, Agent..........
Login
Import Project
Save
Exit
Send to Google Drive

Figure 6.1: Activity diagram of a user using the web application

The following part of this section is an addition for ID3.

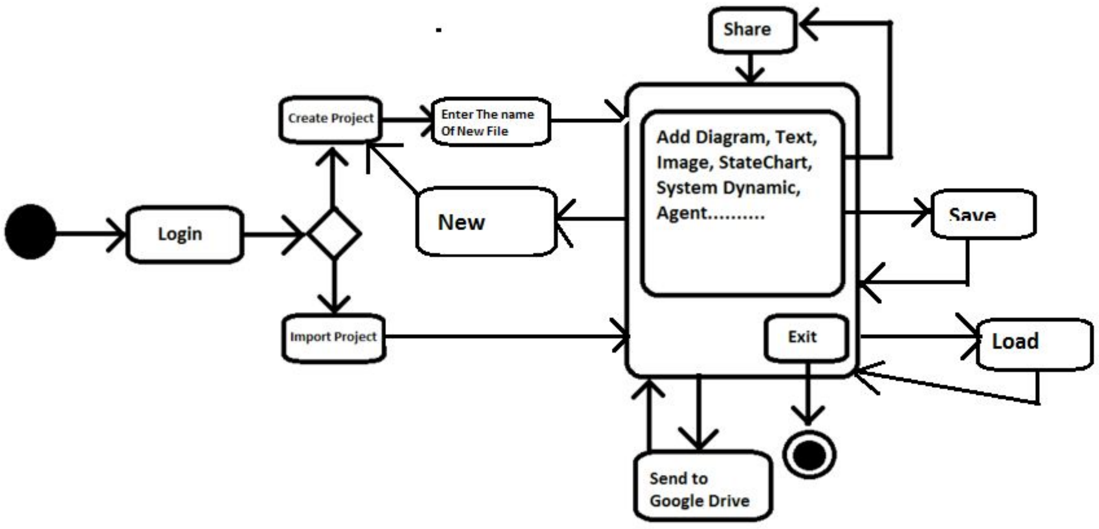
Share
Create Project
Enter The name Of New File
Add Diagram, Text, Image, StateChart, System Dynamic, Agent..........
Login
New
Save
Import Project
Exit
Load
Send to Google Drive

Figure 6.2: New activity diagram of a user using the web application

VII. Storyboards

To better visualize how our web application will look, we have created a number of storyboards to lay out the different components of our system.

Our first storyboard, below, shows a tool list of system dynamic options. This includes stock, flow, link, parameter, variable and positive and negative loop. They can all resize and all include text bars after you draft it from tool list to the workplace.

Furthermore, the 'file' option will be a dropdown menu which will allow functionality such as saving, opening, sharing, editing, etc. The 'edit' option will include specific tool options and other things like undo/redo, or delete object . 'Color' currently would include variations on the font such as colour. It may be renamed to 'tools' later to allow for more broad options.

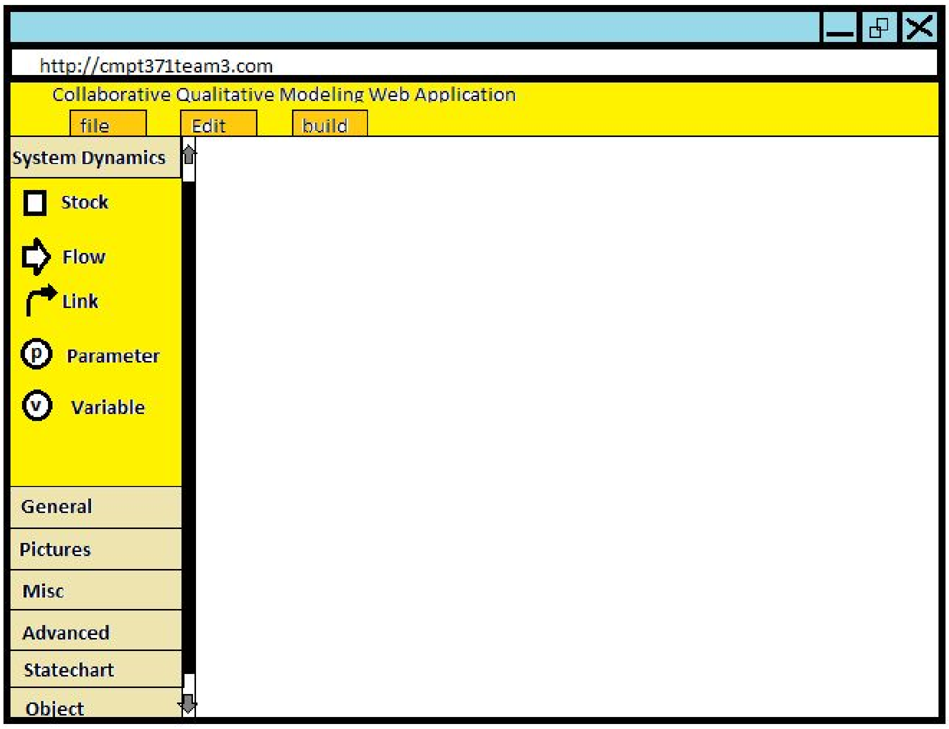

Figure 7.1: Storyboard of the system dynamics **(UPDATE FOR ID2)**

Our second storyboard shows a tool list for the state chart tools. This includes state chart entry, state, transition, initial state pointer, history state, final state and branch. The user can draft tools from tool list and all tool can be resized and renamed.

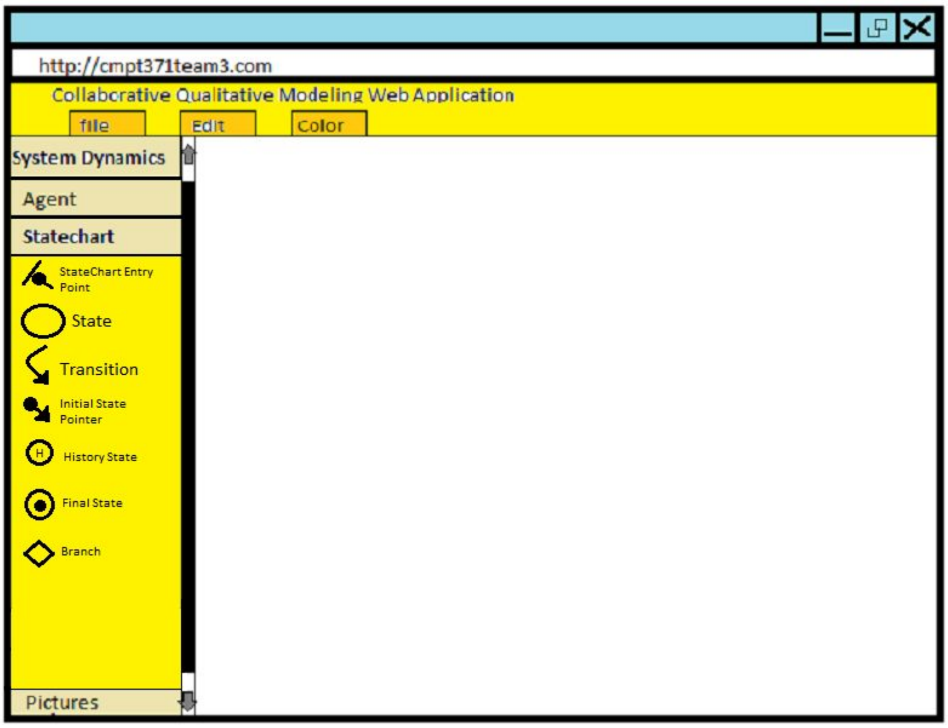

Figure 7.2: Storyboard of the statechart

The following part of this section is an addition for ID3.

As the project has progressed, we have created an updated diagram which aims to more closely reflect the requirements we are focussed on.

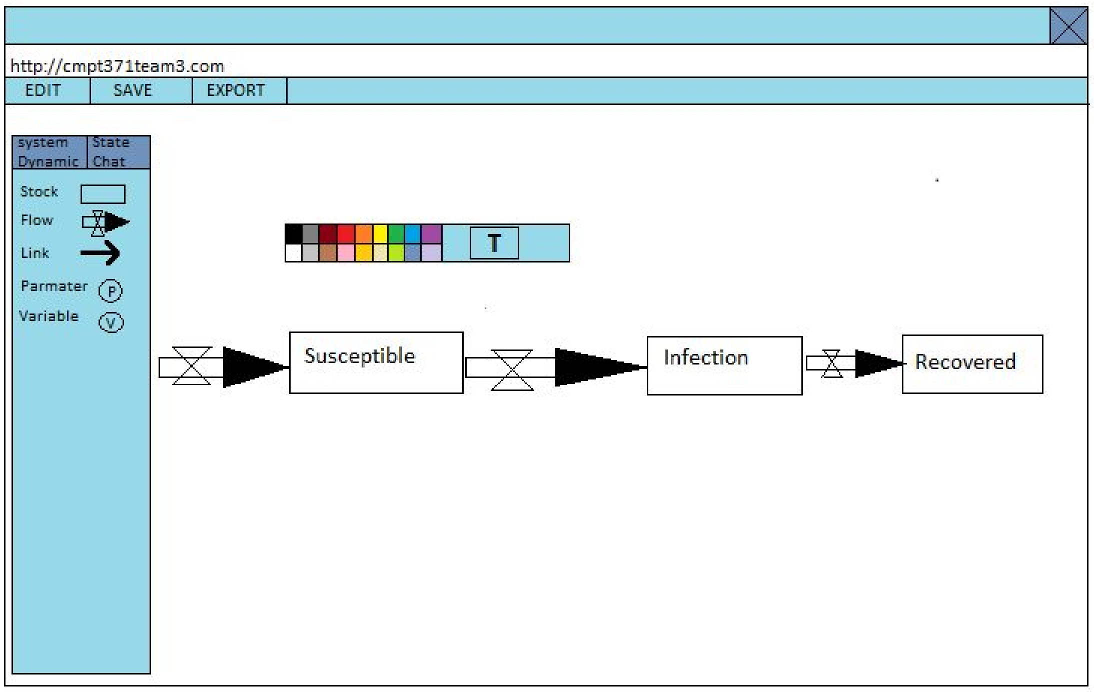

Figure 7.3: Updated storyboard of the interface

VIII. Testability

In order to ensure quality and functionality, the system will be subjected to many test at each deliverable. These tests will include automated and manual tests. QM-Lab contains a large portion that is graphical in nature, therefore tests to verify the quality and useability of the UI will be done manually. However, stress, load and functionality testing will be mostly automated using the Selenium framework as well as Python tests.

With the inclusion of new features at each milestone, the system will be put under an updated test suite that will ensure that the old features are still functioning as well as the newly implemented feature. Moreover, the triage team will be involved in determining whether new features are included in new tests or put aside till further notice.

The bulk of the testability issues will come up with the user interface, as the application will be a very visual project. This will require finding ways which we can mock and model the interactions with the interface and test them at a structural level (ie. through more normal testing means for typical code).

Additionally, when considering systems like the Google Realtime API, we will need to consider that only black box testing will be available.

The following part of this section is an addition for ID3.

From these requirements we have identified certain features and how testable we think they will be from easy, reasonable, or difficult to test. They are listed below and based off the scope outlined in section III.

1. **Multiple users**. This will be **reasonable** to test because we can test this by varying the number of users (0, 1, 2+, many). The general problem this will pose is finding many different users or a way to mock many users to test.
2. **Create, save, or load projects.** This will be **reasonable** to test because the basic functionality is something we can observe work but it will be tough to reach all of the cases where we might find bug (ex. having extremely large files be saved).
3. **Create different elements on the model**. This will be **easy** to test because creating different elements relies on very few other functions. The most difficult part will be actually determining how we will create them since they would typically be created with the interface.
4. **Functionality of certain elements**. This will be **reasonable** to test. There are so many elements that have different features it will require different test for each of them, but overall the features are very well defined (ex. the agent must be able to hold other elements) so they can be enumerated and tested.
5. **Link elements together**. This will be **reasonable** to test because it will require matching different pair combinations of elements but can be represented in a graph of links and nodes. The difficult part will be testing if the visual links are displayed properly and in a way that makes sense (ie. not over top of each other).
6. **System authentication**. This will be **reasonable** to test, because it is a Google product we can assume a lot of the functionality it gives will work, but also that we need to make sure we are connecting to Google's authentication properly and that users are accessing their own documents (which may be more **difficult** to test).
7. **Editing the diagram**. This will be **difficult** to test because it is a very visual testing that requires selecting different objects, moving them around, changing their properties. While we can test things at a functional level (within our code), it will be difficult to make sure it actually works with the visual interface.
8. **Print the model**. Will be **difficult** to test because this is again a very visual system. How can we test that different configurations are actually printing correctly (unless we manually print them all off and compare them).
9. **Persistence**. Persistence will be **reasonable** to test because it will simply require making sure that a project is the same as it was before. This can be done by passing a project through the export and importing process and checking if it is the same and functions correctly still.

IX. Conclusion

In conclusion, we have defined a number of requirements to meet the needs of the QM Labs project given to us by Nathaniel Osgood and Geoff MacDonnell. We have identified what the system must do to be successful, what it should do, and what we would like it to do.

We believe there will be a number of actors who will interact with the system, including the user, other users, the interface, and the database. To help understand the flow of this system, an activity diagram was created and can be seen in section VI. Lastly, we have created storyboards which will help us visualize and get feedback on design from the client for the end product.