

# QM Lab Course Project

## Requirements Document

### Group 3

Angela Stankowski (ans723), Bengin Lee (bel529), Connor Nettleton-Gooding (cwn973), Corey Hickson (crh208), Darwin 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)

February 7<sup>th</sup>, 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.

## 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.

## III. Scope

### 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

## 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
- Different 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

# 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.

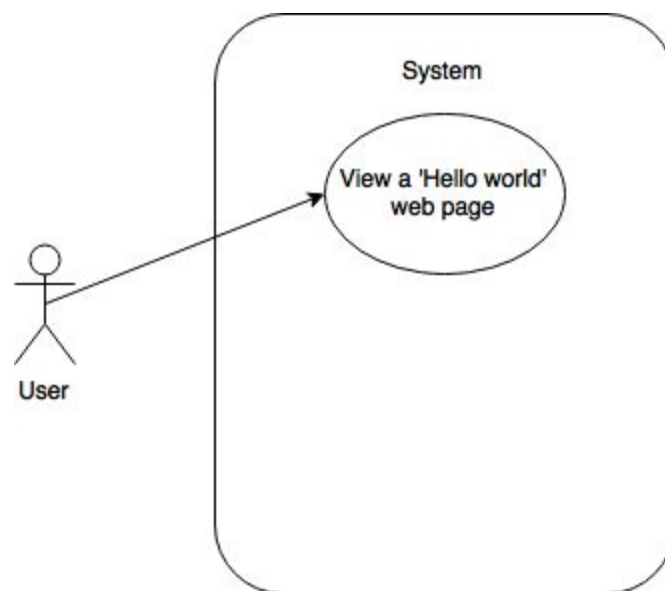
**Server:** The primary function of the server is to store, process and deliver web pages to clients who request these needs. If a process cannot be made, it must output an error. The server must also support multiple users interacting collaboratively on a given project within the system.

**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.

## 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.

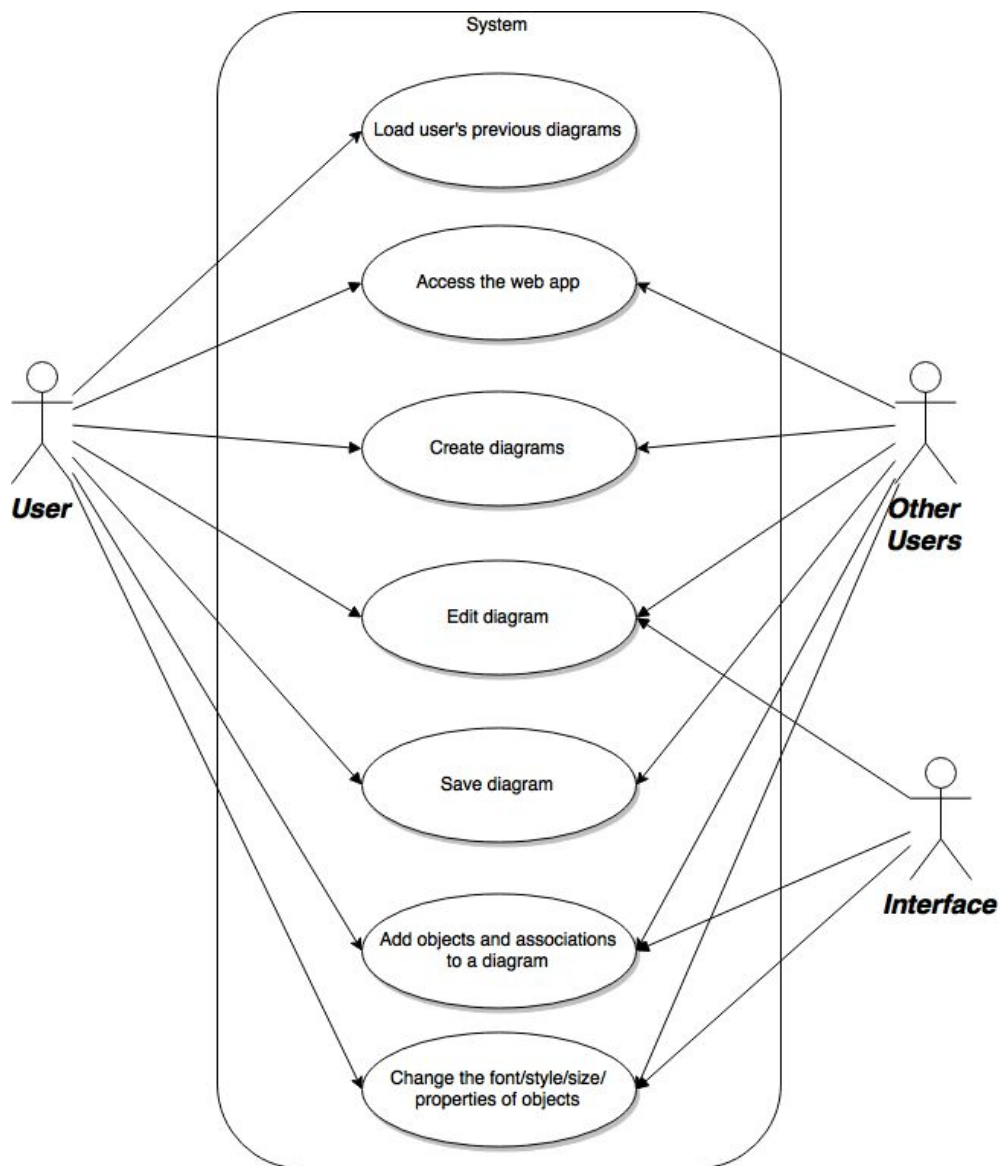


**Figure 5.1:** Use case diagram for the first incremental deliverable

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.

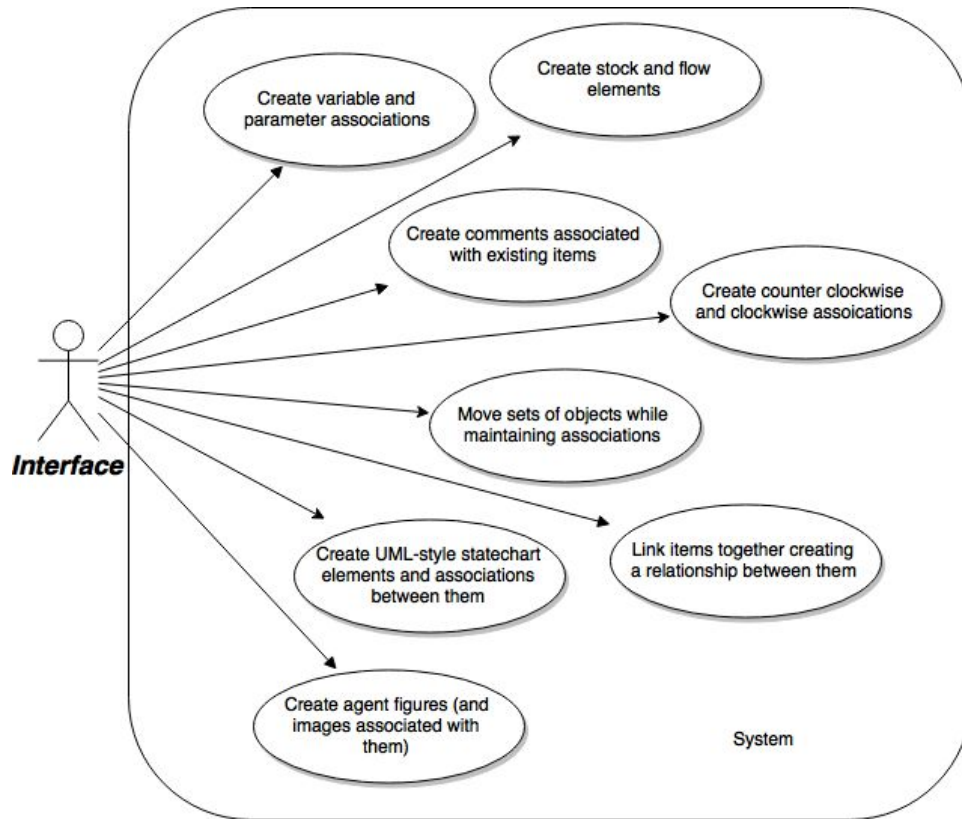
Other than that, we have generalized the interactions which a user would take. Due to the complexity of the actual interface and how it works, this has been saved for another diagram showing off the interface as an actor which interacts with objects.



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

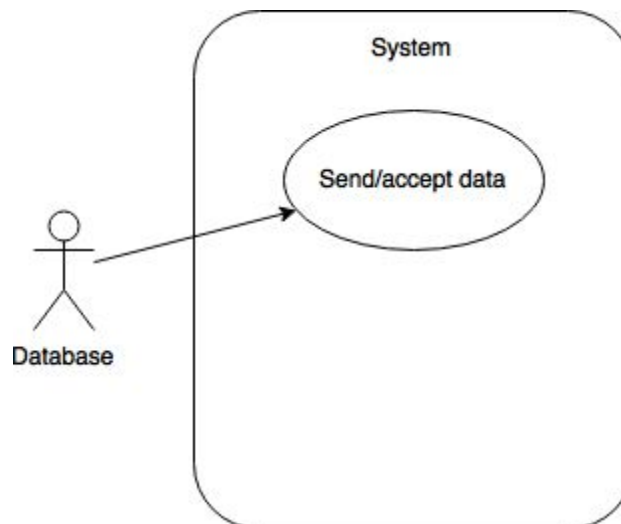
Lastly, we have created a use case for the abstract idea of an interface, seen below. The point of this was to better specify exactly what the interface with the system, so we could broadly say the user was simply adding, editing and creating diagrams.

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.



**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

## 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.

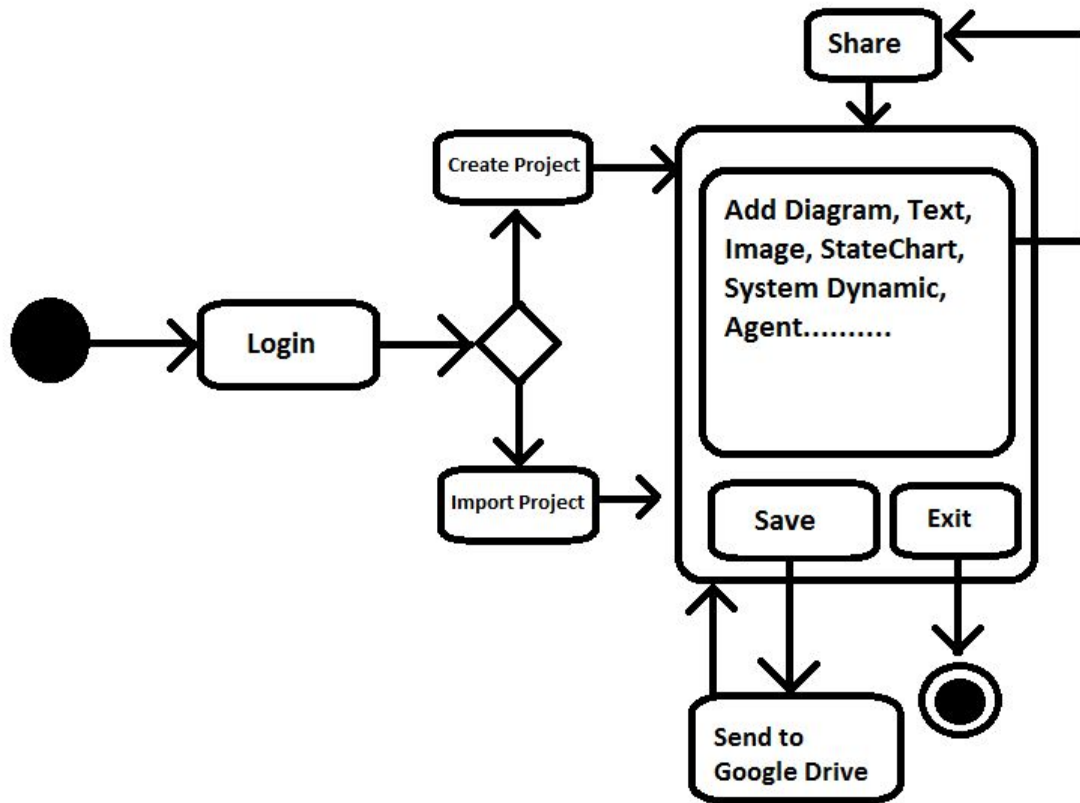


Figure 6.1: Activity diagram of a user using the web application

## 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.

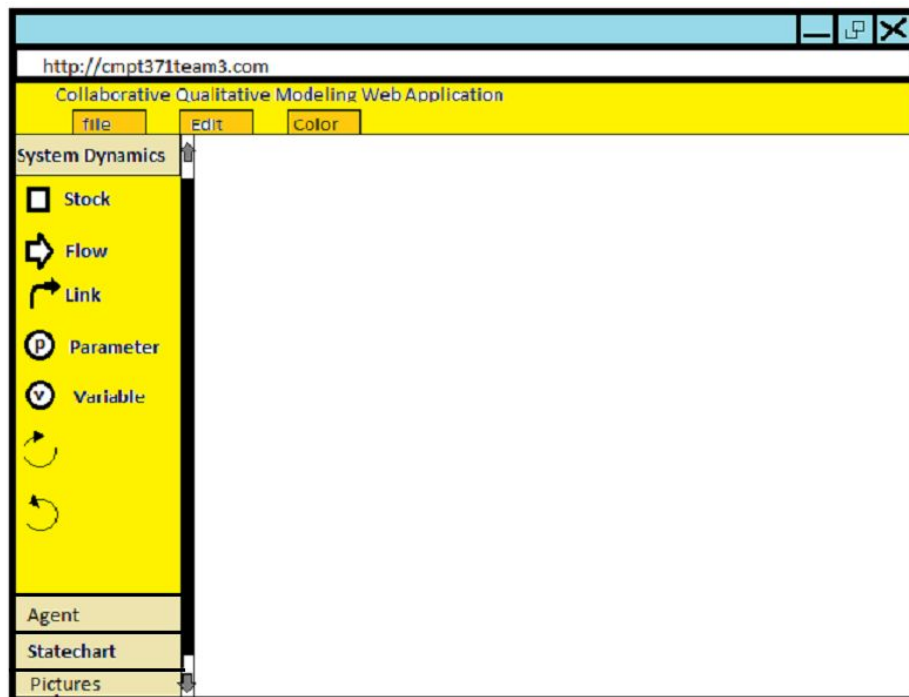
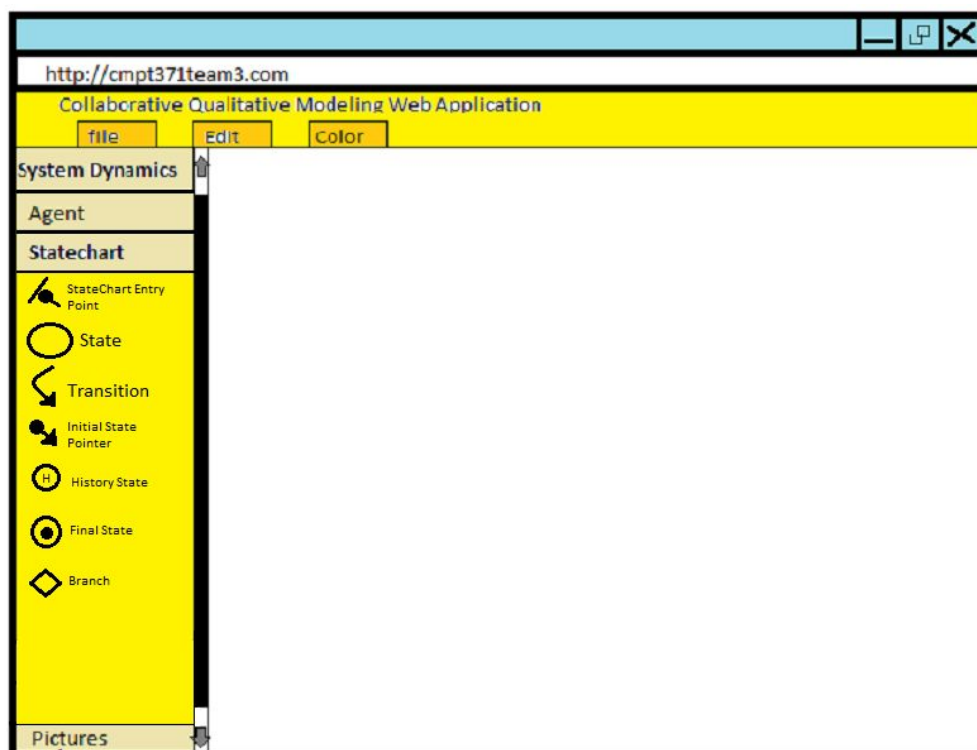


Figure 7.1: Storyboard of the system dynamics

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.



7.2: Storyboard of the statechart

Figure



## 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.

## 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.