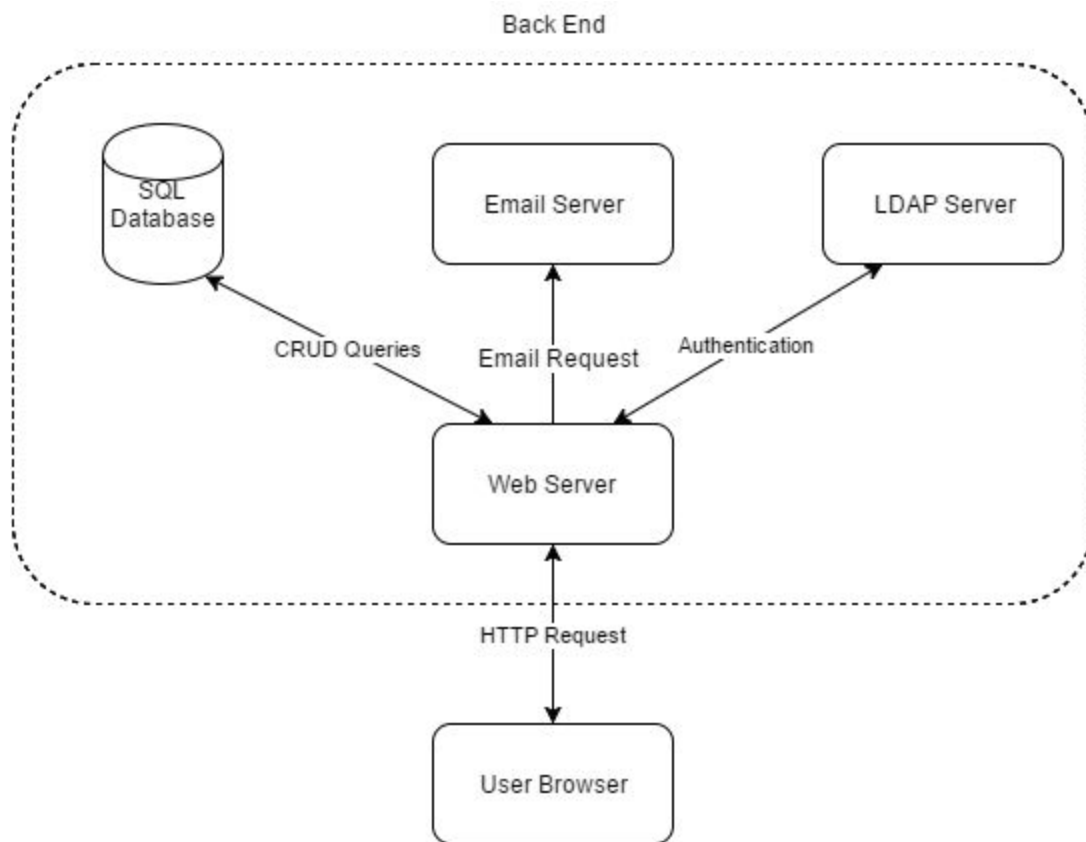


Architecture Design

This website will involve a number of different systems working in tandem to provide the user with a usable end product. The primary parts that will be integrated together are a web server to handle the HTTP requests from the user's web browser, a SQL database to handle the storage of emails used for the email service, an LDAP server to handle authentication via active directory, and an email server to handle the sending of the emails for the subscription service. All of these parts are outlined in the figure below.



The primary control center for the website is the web server. The web server will delegate all interactions from the user's web browser to the appropriate system. New users will be registered with the LDAP server, new email subscriptions will be saved in the database, and when emails need to be sent those requests will be delegated to the email server.

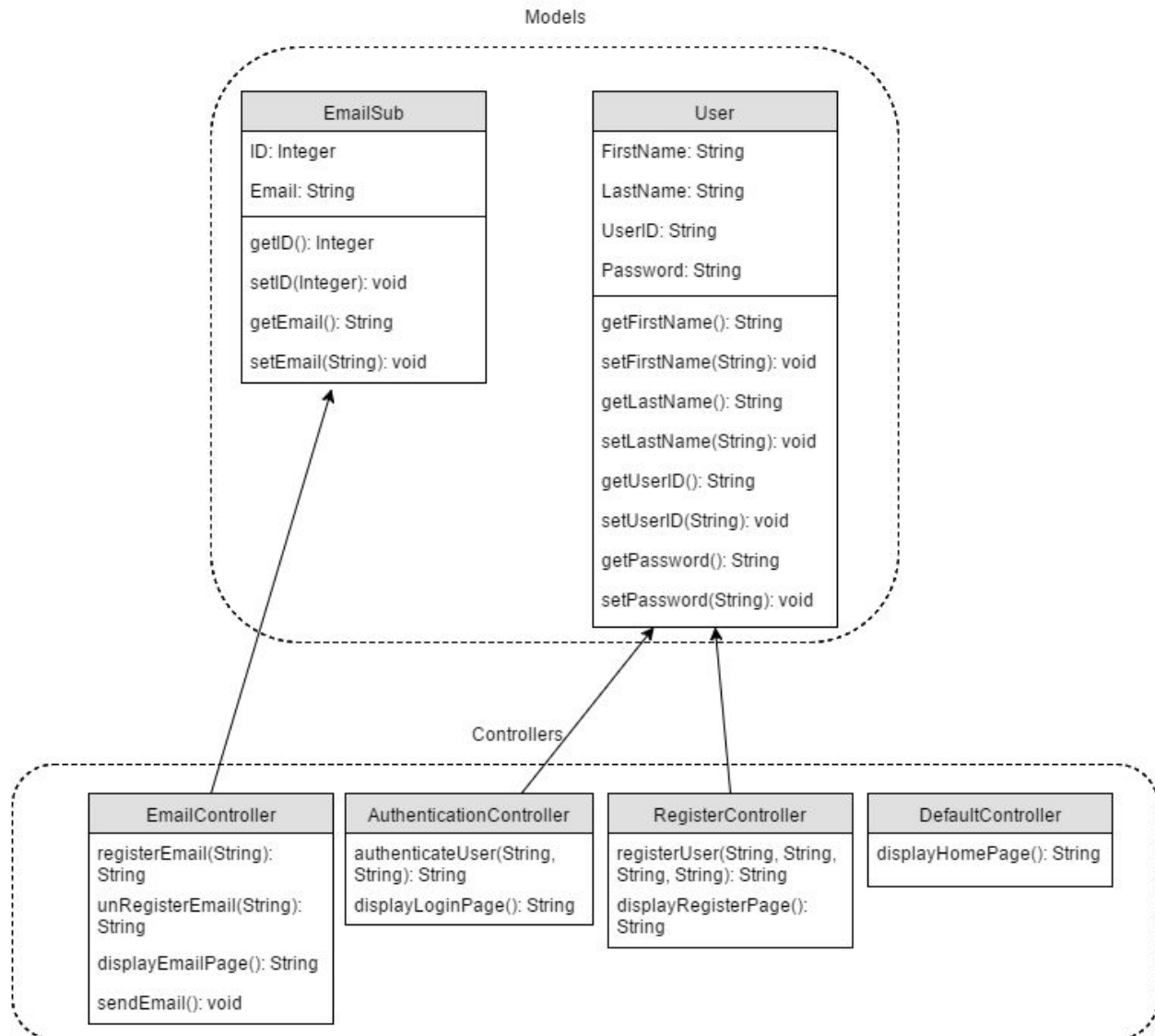
The SQL database will simply be in charge of managing the data for the email subscription service. It will contain a table that will store all of the emails currently subscribed to the email service. It will handle basic create, read, update, and delete queries to add new emails, return all of the existing emails, update emails, and delete emails that are no longer needed. All of these requests will be sent to the database via the web server.

Emails will be sent out from the email server at regular intervals as delegated by the web server. A cron expression will be used in the web server to send these requests to the email server, which will contain all of the email addresses that need to receive the email, the subject of the email, and the body of the email. The email server will then be responsible for sending the emails using SMTP.

Authentication for the website will be handled through an LDAP server. When a user is asked to login, the credentials will be sent to the LDAP server and checked to ensure that the user is registered and has entered his or her credentials correctly. Requests to register for the website will also be sent to the LDAP server to allow new users to register to gain access to the website.

Detailed Design

This website will be using a model, view, controller design. Doing so creates a simple way to design the website, and to add new features as they arise. Each section of this design will be covered more in depth with class diagrams for each in this section.



This figure shows the class diagrams for the models and controllers that will be used in the website. The views will be discussed later, along with prototypes of what the user interface will look like.

The EmailSub class will be the model for saving email subscriptions to the database. It is a very basic class that will only contain the email address and a unique ID that will serve as the primary key. This unique ID has no use to the class itself, it will be serving as a surrogate key in case further expansion is ever needed to the class and database table. The EmailSub class will also contain getters and setters for modifying these variables.

The User class will be used to authenticate users, as well as register new users. It contains fields for the user's first name, last name, userID, and password. User objects will be created and used to store information about users that will be sent to the LDAP server to either

authenticate or register new users. The class contains getters and setters for all member variables to allow for safe modification.

The controllers are the classes that will be doing the work for the website. The first controller in the diagram is the email controller. This controller contains methods for registering new emails for the subscription, unregistering existing emails, navigating to the email subscription page, and a method to send the email requests to the email server. The first three methods will be called by the user via the user interface. The sendEmail method however will be called using a cron expression and will fetch all the registered emails from the database and send them out via the email server. Any errors when registering or unregistering an email will be handled by this controller as well, such as trying to register an email that has already been registered.

The AuthenticationController is used to handle all requests dealing with authentication. This controller will be used to interface with the LDAP server and authenticate users from the login page using the authenticateUser method. It also contains a message to display the login page, which will be called when the user presses the login button. This controller will also handle any errors when the user is logging in, for example if they are not a registered user or input their credentials incorrectly. Both methods will be called from the user interface.

Registering a user in the LDAP server will be done via the RegisterController. This controller will take in a user's information, first name, last name, userID, and password, and will connect to the LDAP server to register the user to access the website. Errors will also be handled by this controller, such as trying to register a userID that has already been registered. Both methods will be called by the user clicking links in the user interface.

Finally, there will be a defaultController that will handle requests to take users back to the home page of the website. This controller will only contain one method that will be called when the user clicks on the home link to return back to the home page.

With the design pattern established, and an idea about how the classes will interact with each other, we can begin to take a look how all of this will come together to form an experience for the user. In designing an (almost) brand new user interface, there were three big goals that we wanted to accomplish before handing it over to SQS. We want it to be very user friendly. It must also blend in reasonably well with the client's existing sites. Lastly, it also has to be efficient on all fronts.

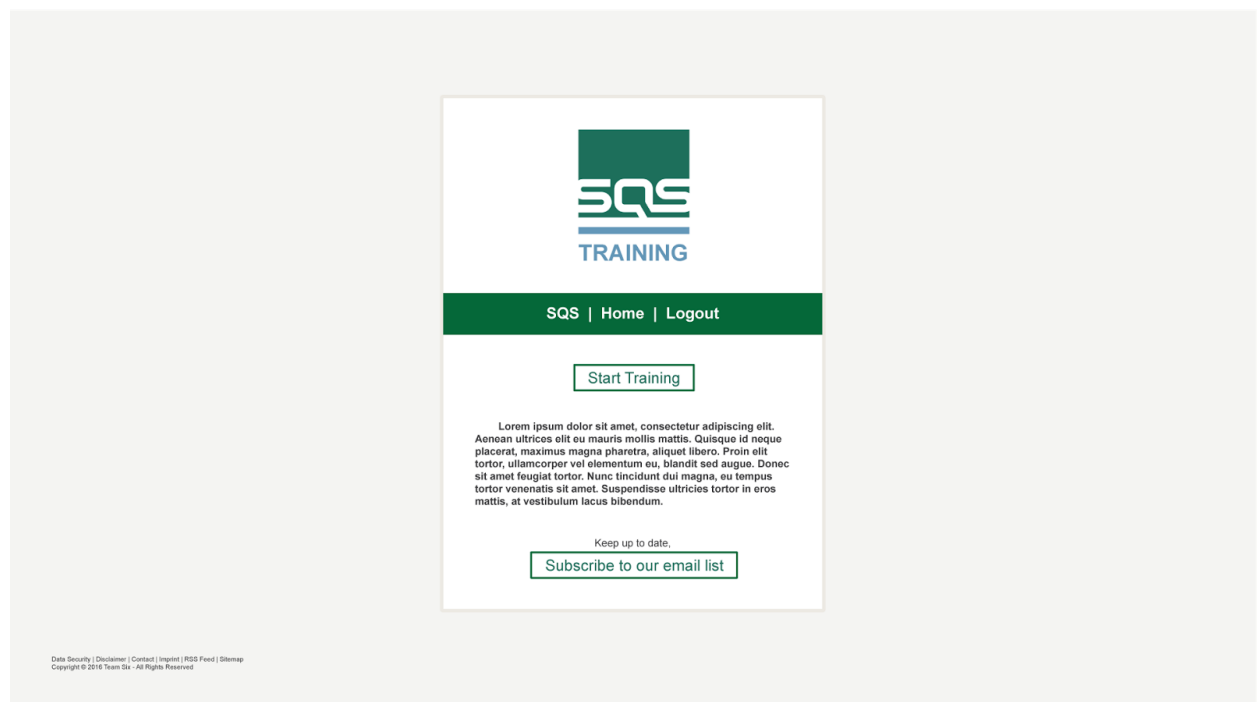
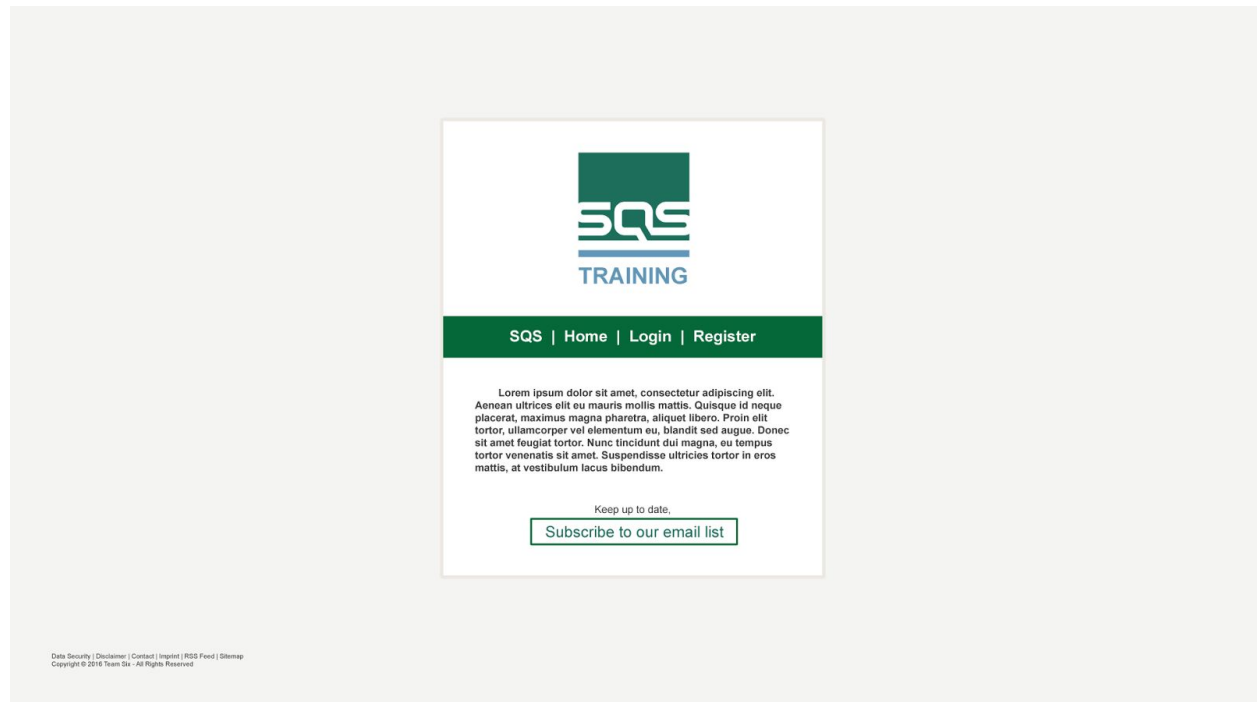
One of the biggest goals for any web-oriented system such as the one we're creating, is for it to be very user friendly. This is crucial because we don't want the site to work against the user, especially when you consider the importance of the training provided by the site. In fact, we've placed so much importance on this concept, that the requirements

specification that we created earlier highlights the importance of this several times in the requirements section.

Another goal for the interface was that we wanted it to both feel familiar to SQS's existing web assets, but to also feel clean and modern. This coincides with what our contacts at SQS requested in their initial pitch, wherein they wanted an updated version of their existing training site built with modern technologies. The real challenge is that former part, keeping the new site thematically consistent with their existing web presence, as this will play an important role in the onboarding of new users. With that said, we think that the below prototypes should be acceptable solutions.

The last of the big goals for the user interface was for it to be efficient. This doesn't just mean that it shouldn't take up a bunch of system resources on the user's computer. It also means that space on the page should be wisely used, and new elements should only be added if they make sense, or help to fulfil the project requirements, or improve the browsing experience. This idea of efficiency ties back heavily into the first major interface goal, user friendliness, as it plays role in how friendly the interface will be to new users. After all, we don't want to have a bunch of extra elements fighting for the user's attention in the background.

With those goals laid out, we can begin to look at the prototype user interfaces built for the site. Below are two example of the homepage, one for logged out users, and one for logged in users. Notice how the navigation bar's contents change based on whether or not the user has logged in, and how the logged in user now has a link to move forward with their training.



At first it might seem like a normal web page, just scaled down and centered. However, the reality is that these pages don't have to convey all that much information, so there's no need to have a giant page that requires tons of scrolling to navigate. Everything is easily seen and reachable, without much thought. It also manages to keep the logo and iconic colors of SQS, so new and old employees should feel right at home. Note that the lorem ipsum

paragraph is just a placeholder for the moment, and it will no doubt be filled with relevant information that the client requests.




SQS | Home | Register

Email Address

Password

Login

[Data Security](#) | [Disclaimer](#) | [Contact](#) | [Imprint](#) | [RSS Feed](#) | [Sitemap](#)
Copyright © 2016 Team Six - All Rights Reserved



SQS | Home | Login

First Name

Last Name

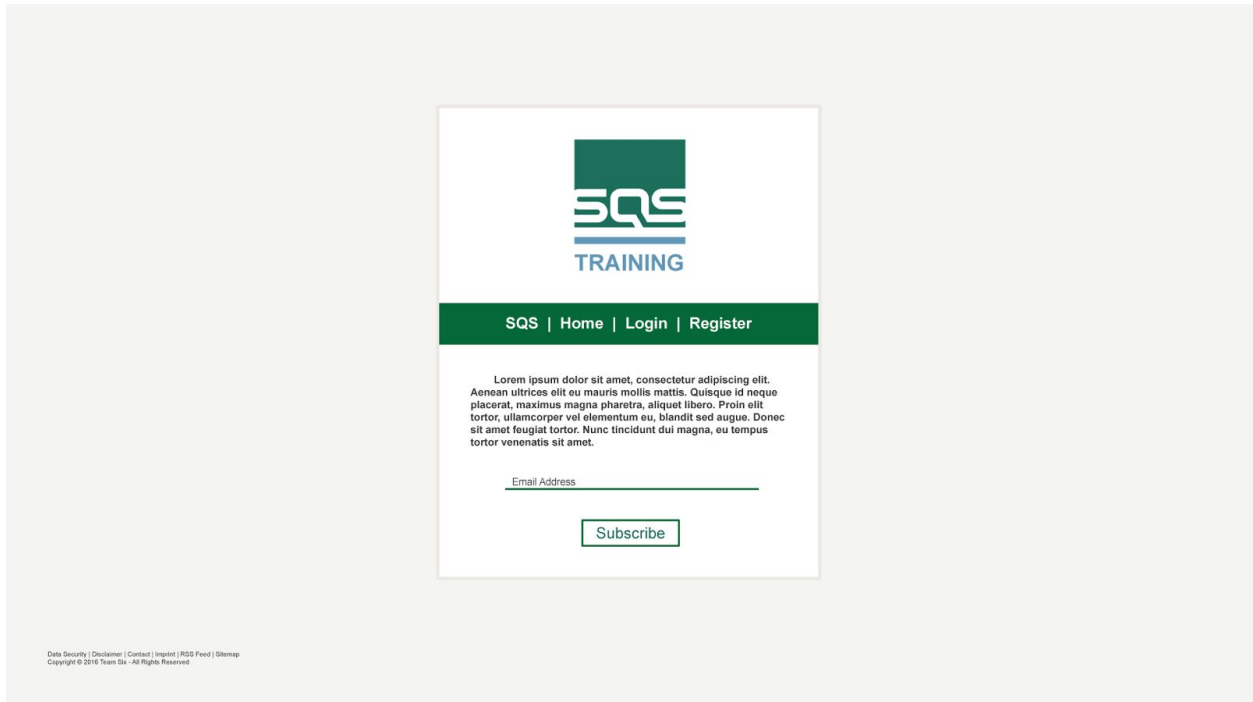
Username

Password

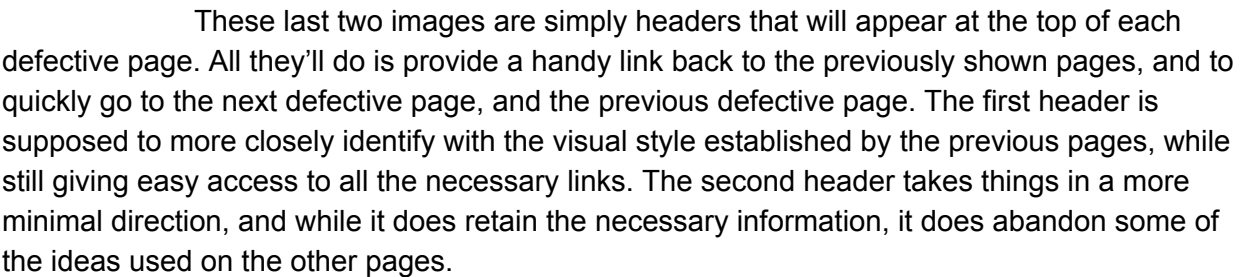
Confirm Password

Register

[Data Security](#) | [Disclaimer](#) | [Contact](#) | [Imprint](#) | [RSS Feed](#) | [Sitemap](#)
Copyright © 2016 Team Six - All Rights Reserved



This next triplet of pages make up the login, registration, and subscription systems. As expected, they're very visually similar to the homepage we looked at earlier. This not only keeps everything visually consistent, but it also reduces the amount of unique code that has to be written to get these pages running, which means less code to maintain in the future. The login page works as expected, you simply type in the relevant information into the fields, and then you'll either be logged in, or rejected. The registration page is similar to the login page, except there's a bit more information to enter before you can have an account. Finally, the subscription page also draws inspiration from the previous two pages, except this one only asks for an email to send subscription letters to. Again, the lorem ipsum placeholder will be replaced with useful information about the subscription, and what the user will receive.



Metrics

The complexity of our system will be naturally low because of the small amount of data transferred to and from objects. Below is a breakdown of the calculated complexity using the Cohesion CK metrics.

Data Field	% Used in Methods
ID	33.3%
Email	16.7%
FirstName	16.7%
LastName	16.7%
UserID	66.7%
Password	16.7%
(100 - Total Avg Percentage)	72.2%

The size of the project can be estimated with eight user stories (User Registration, User Login, User Logout, User Subscribe to Letter, User Unsubscribe to Letter, User Views Homepage, User Views Training Section, Users Views Training Page), x number of test cases (), and 12 methods (getID(), setID(Integer), getEmail(), setEmail(String), getFirstName(), setFirstName(String), getLastName(), setLastName(String), getUserID(), setUserID(String), getPassword(), setPassword(String)). No method should take significant time to code since there are no complex algorithms and consist mainly of transferring data to and from systems.

Effort can be estimated by breakdown of responsibility within the project. Given the four teammates on the team, and adhering to our time estimate given in the Design document the overall time spent on the project should be at maximum forty-six hours with a best guess at around 25 hours. The initial setup of the interfaces can be done by team member a with an approximate allocation of 5-10 hours, each of these method creations can be broken down to approximately 2 hours of effort each. This task can be delegated to team member b and c. Team member d will be testing which should take between 5-10 hours. So far all teammates have contributed roughly 4-6 hours each toward the project design.

No defects were discovered during quality assurance review.