

CS 320 Course Project Final Report

for
SubScruple

Prepared by Kevin Black

Group Name: BBS teemMaets

**Shane Bellika
Kevin Black
Alex Sloan**

**11639152
11480710
11651363**

**shane.bellika@wsu.edu
kevin.black@wsu.edu
alex.sloan@wsu.edu**

December 7, 2018

Contents

CONTENTS	II
1 INTRODUCTION.....	1
1.1 PROJECT OVERVIEW	1
1.2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS.....	1
1.3 REFERENCES AND ACKNOWLEDGMENTS.....	1
2 DESIGN.....	2
2.1 SYSTEM MODELING	2
2.2 INTERFACE DESIGN	8
3 IMPLEMENTATION	11
3.1 DEVELOPMENT ENVIRONMENT	11
3.2 TASK DISTRIBUTION	11
3.3 CHALLENGES.....	11
4 TESTING.....	12
4.1 TESTING PLAN.....	12
4.2 TESTS FOR FUNCTIONAL REQUIREMENTS.....	12
4.3 TESTS FOR NON-FUNCTIONAL REQUIREMENTS	12
4.4 HARDWARE AND SOFTWARE REQUIREMENTS.....	13
5 ANALYSIS.....	14
6 CONCLUSION	15
APPENDIX A - GROUP LOG	16

1 Introduction

SubScruptle is a free, web-based application focused on letting users manage their personal subscriptions for a variety of services and was developed and managed by BBS teamMaets. The user is able to enact different actions while using this application, such as: view a multitude of services listed by the application, adding subscriptions to their own profile, viewing all their subscriptions and a running total for how much they spend monthly/yearly, editing their profile, and checking notifications which, depending on the user's respective location, they would receive notifications about newly available services or different deals their services are currently offering.

1.1 Project Overview

While the project has functionally changed since the initial ideas stated in the respective software requirements specification document, the sole purpose of this project stayed pretty much the same. The latter being that we wanted to create an application that could give users the possibility to manage their subscriptions in one place and being able to keep track of how much they spend as a result. In this brief section, you were introduced to a summary of our finalized version of our application thus far, as well as any terms that need defining and references/acknowledgements utilized for this document. The remaining sections of this document will go in depth into the various steps of the development process, such as: the seemingly constant changing design of the application throughout its early stages, interface designs, certain development environments/techniques, how we went about testing our application, and discussing the challenges and future associated with this project.

1.2 Definitions, Acronyms and Abbreviations

1. CS 320	Fundamentals of Software Engineering, of which this document and application were developed at Washington State University under the tutelage of Professor Xinghui Zhao.
2. HTTP	Hypertext Transfer Protocol, an application protocol for online systems.
3. IDE	Integrated Development Environment, where the programming is performed.
4. MEAN	A development technique composed of MongoDB, Express.js, AngularJS, and Node.js.
5. Meteor	Web application framework written using Node.js, with one of its key features being that web applications don't need manual refreshes to display changes to in application.

1.3 References and Acknowledgments

- Standard IEEE Citation Guide: https://learn.wsu.edu/bbcswebdav/pid-2795651-dt-content-rid-91333015_1/courses/2018-FALL-VANCO-CS-320-7661-LEC/IEEE-Citation-StyleGuide.pdf
- Slide reference material provided by Professor Xinghui Zhao of WSUV.
- Application coding standard (for JavaScript): <https://eslint.org/>
- Application development environments: <https://www.jetbrains.com/idea/?fromMenu>
- Web Framework: <https://www.meteor.com/>

2 Design

2.1 System Modeling

Despite quite a bit of changes in some functionalities in our original take of the application, the general system modeling diagrams created for this project in the software design document have stayed pretty much the same. As much as our team wanted to implement the application following the goals we set in place in our second milestone, some features were just not feasible for our time frame and our experience. Certain system modeling diagrams that were present in milestone two were scrapped or are yet to be implemented, such as: database management figures, subscription purchases, the overall class diagram, diagrams depicting administrative privileges, and a state diagram for profile management. The few exceptions for the system modeling diagrams that changed or are still relevant to the current state of the application can be found below.

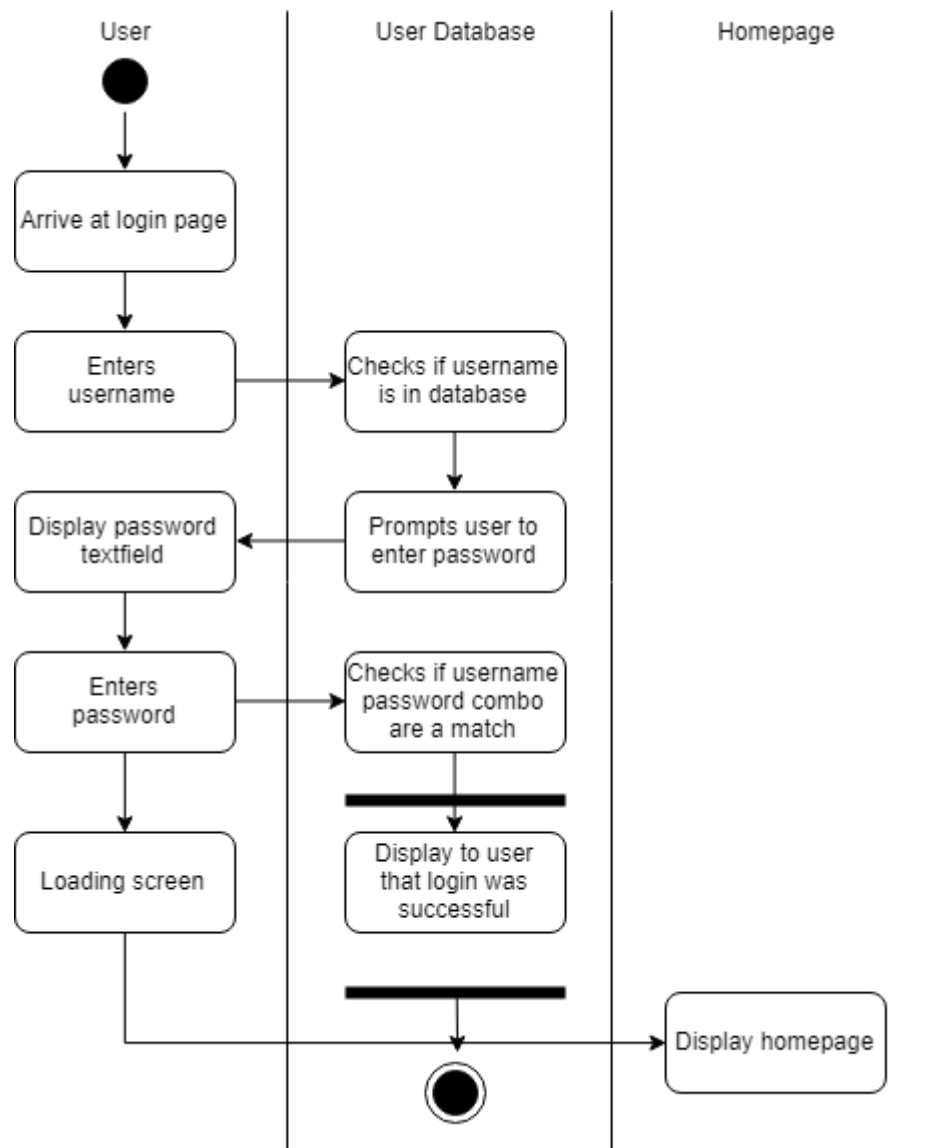


Figure 1. Activity diagram depicting a user logging in to their account.

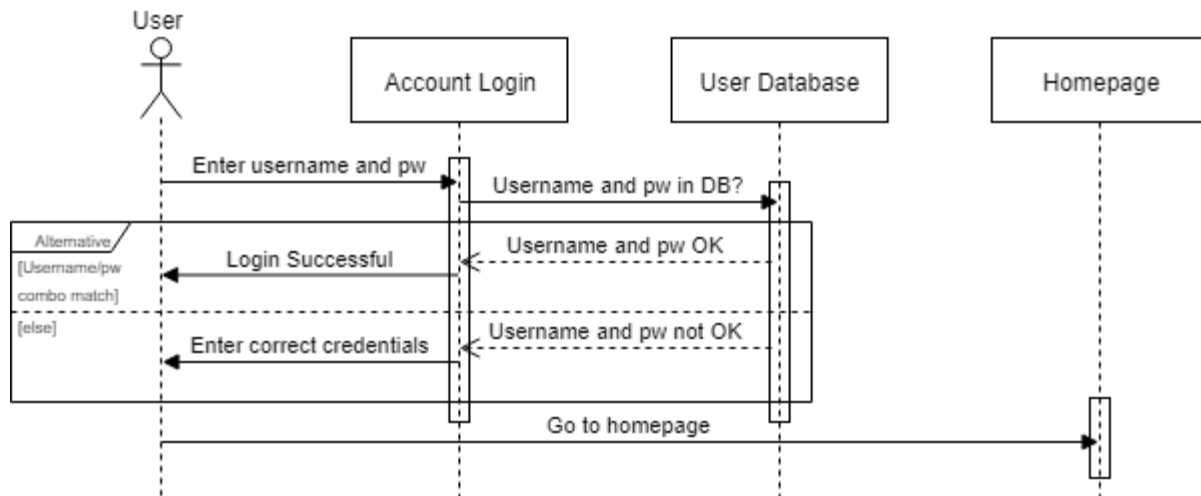


Figure 2. Sequence diagram depicting a user logging in to their account.

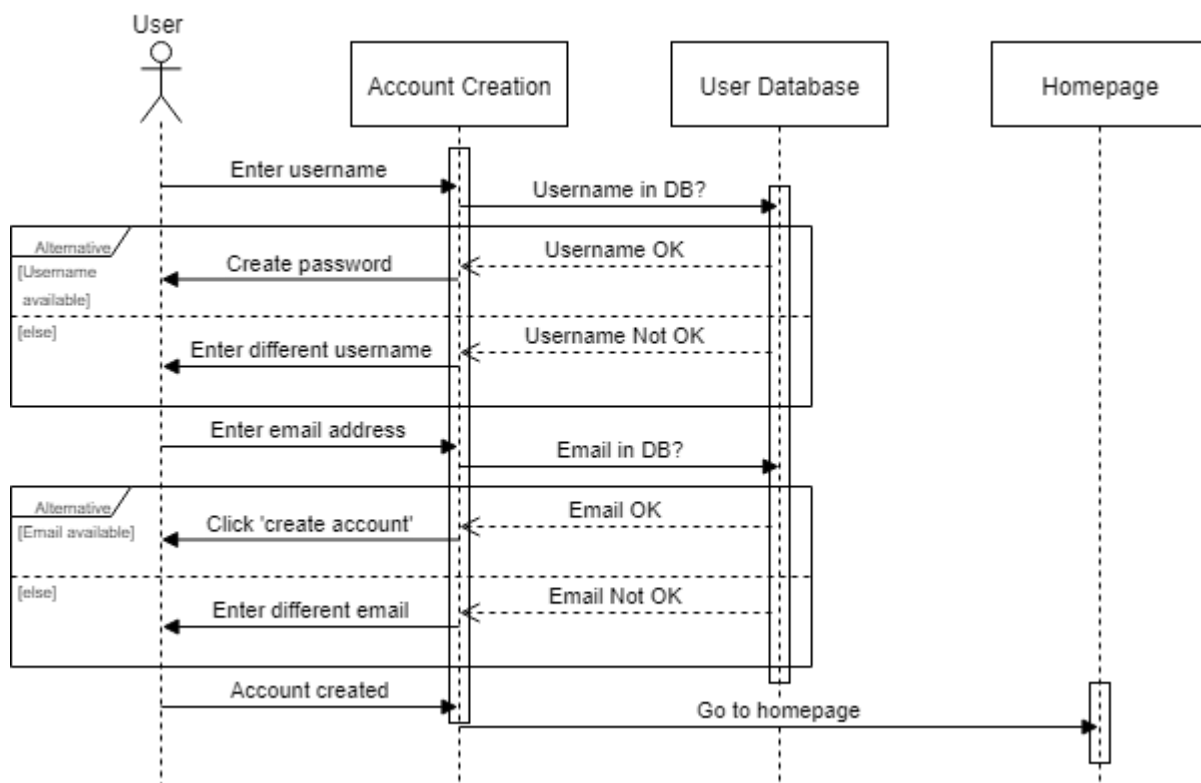


Figure 3. Sequence diagram showing how a user would create their account.

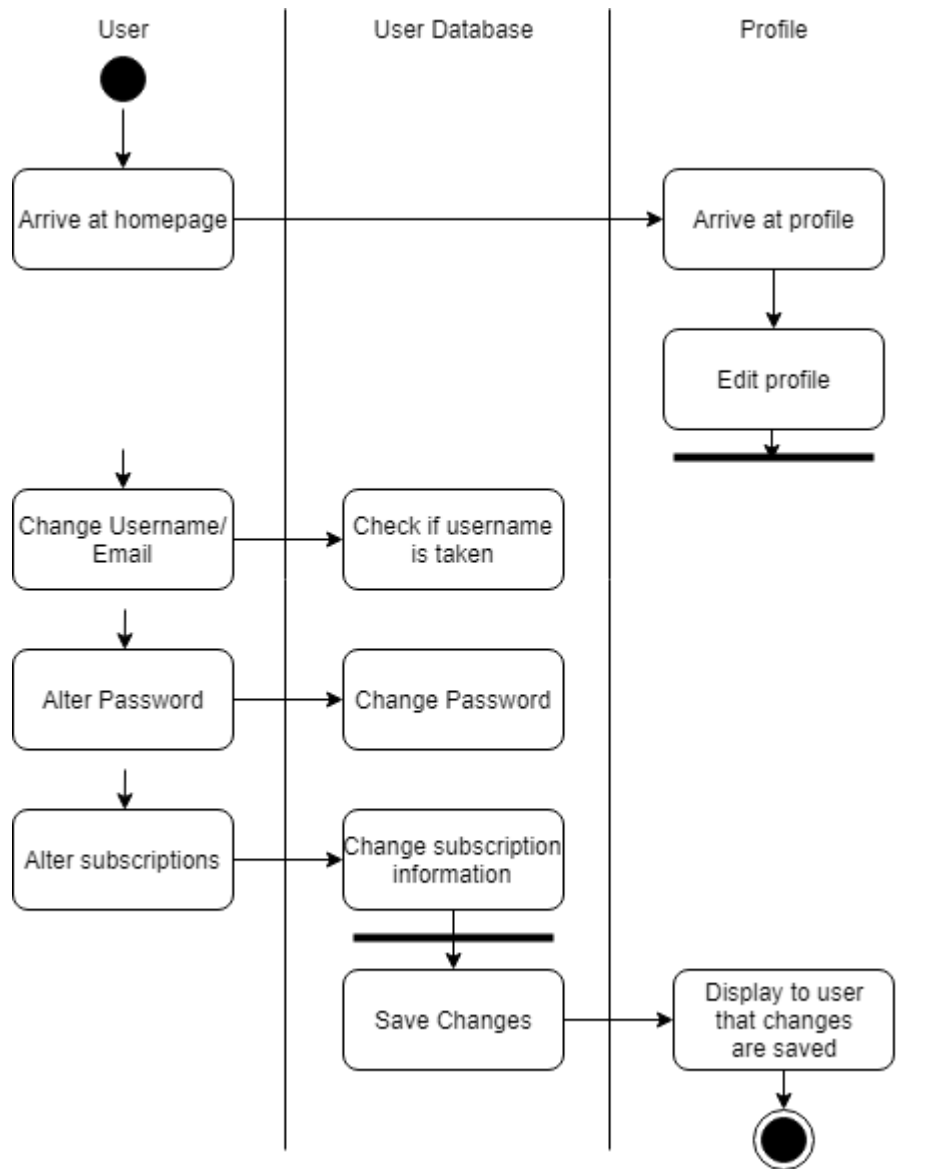


Figure 4. Activity diagram showing how a user can manage their profile.

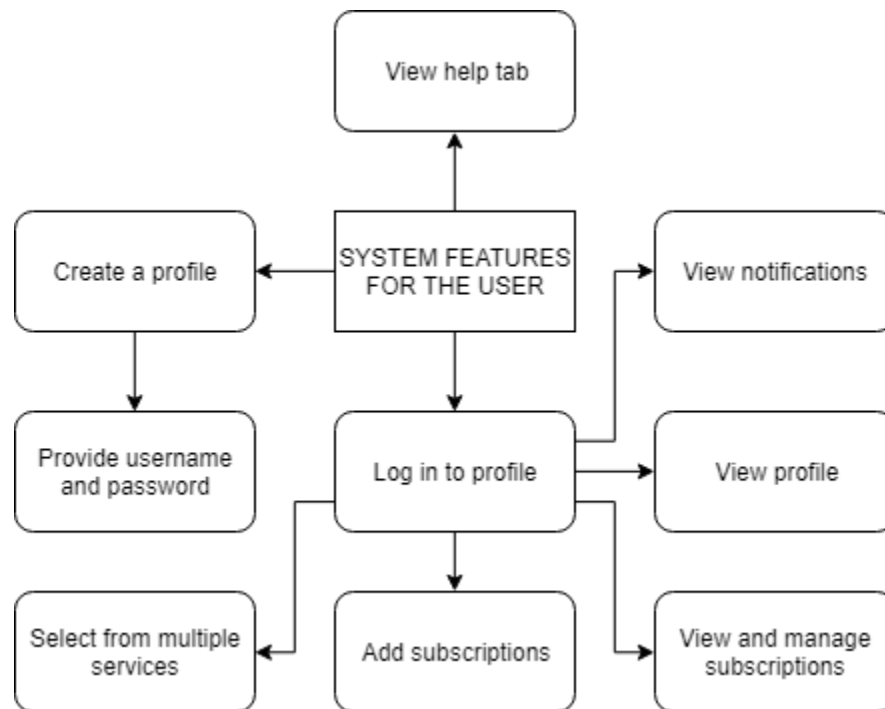


Figure 5. The product perspective altered from the initial concept.

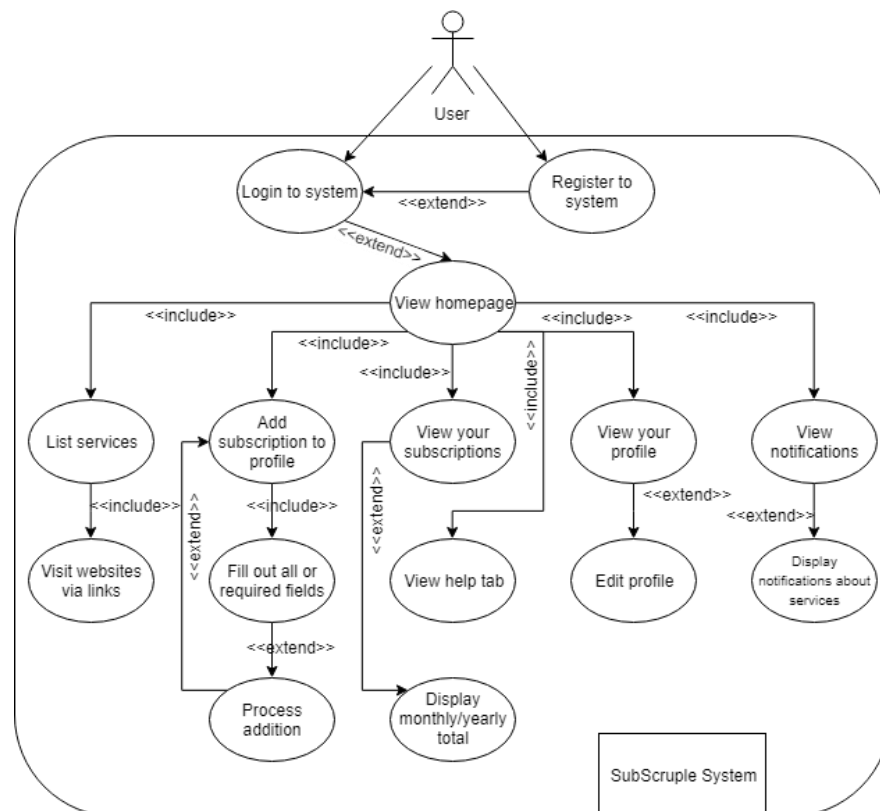


Figure 6. The overall system user case of SubScrip with slight variations from the initial concept.

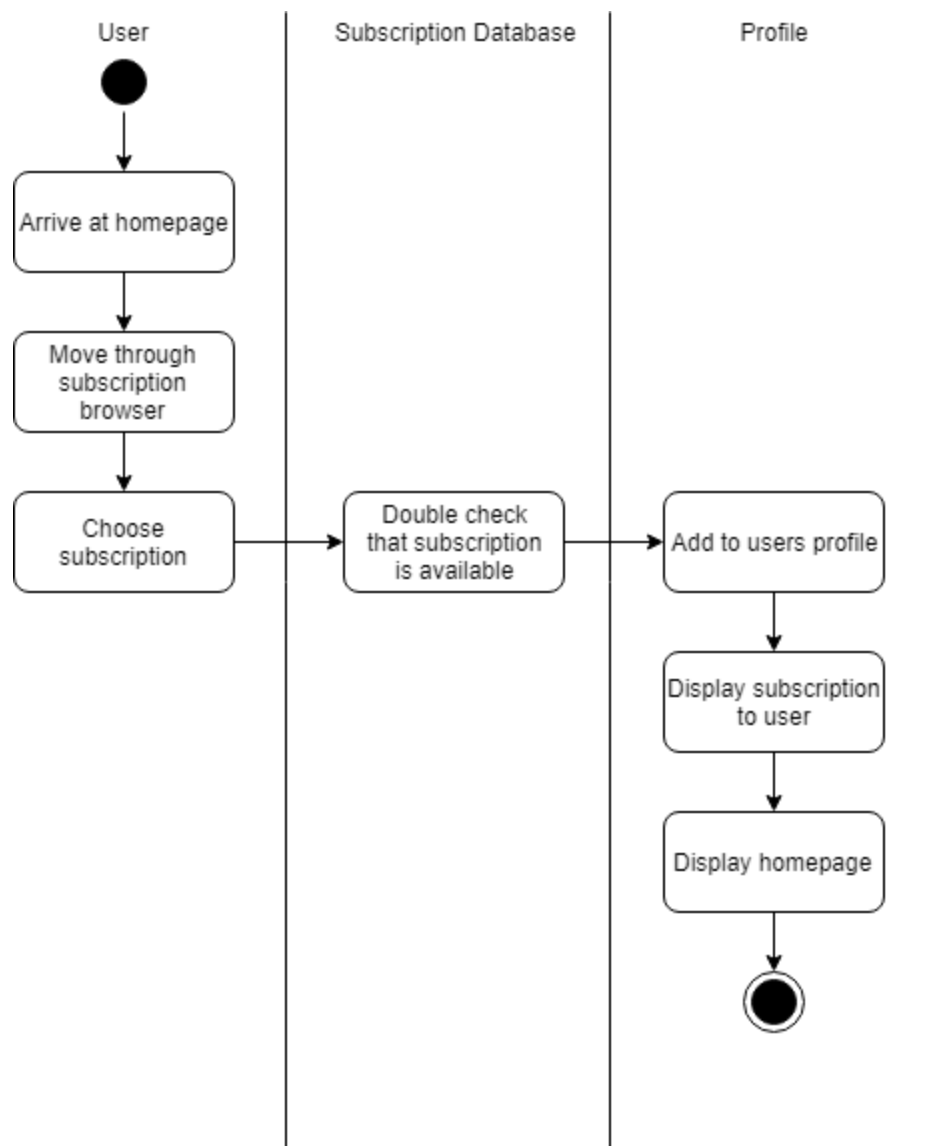


Figure 7. Activity diagram for adding/viewing subscriptions.

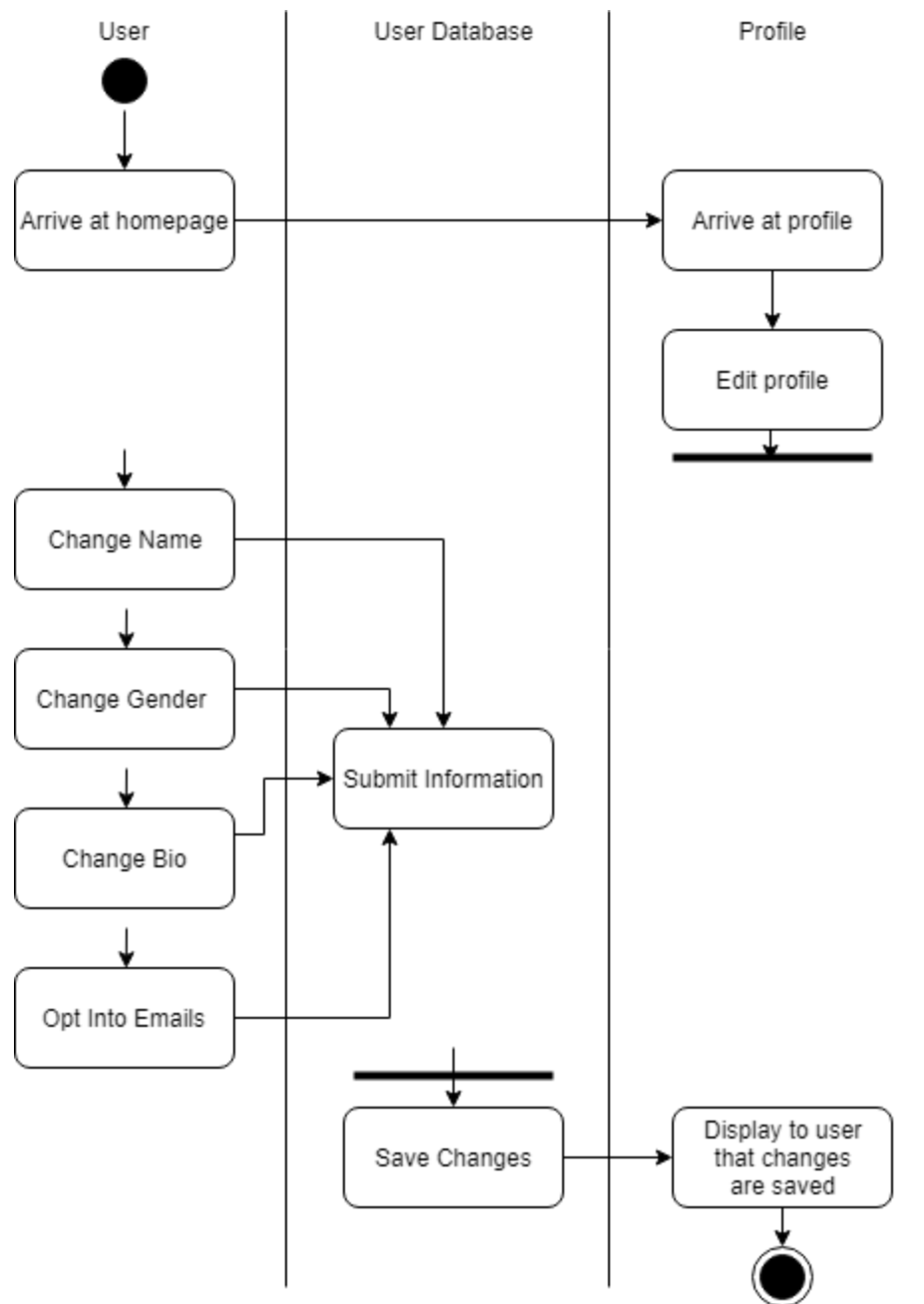


Figure 8. Activity diagram showing for viewing profile/notifications.

2.2 Interface Design

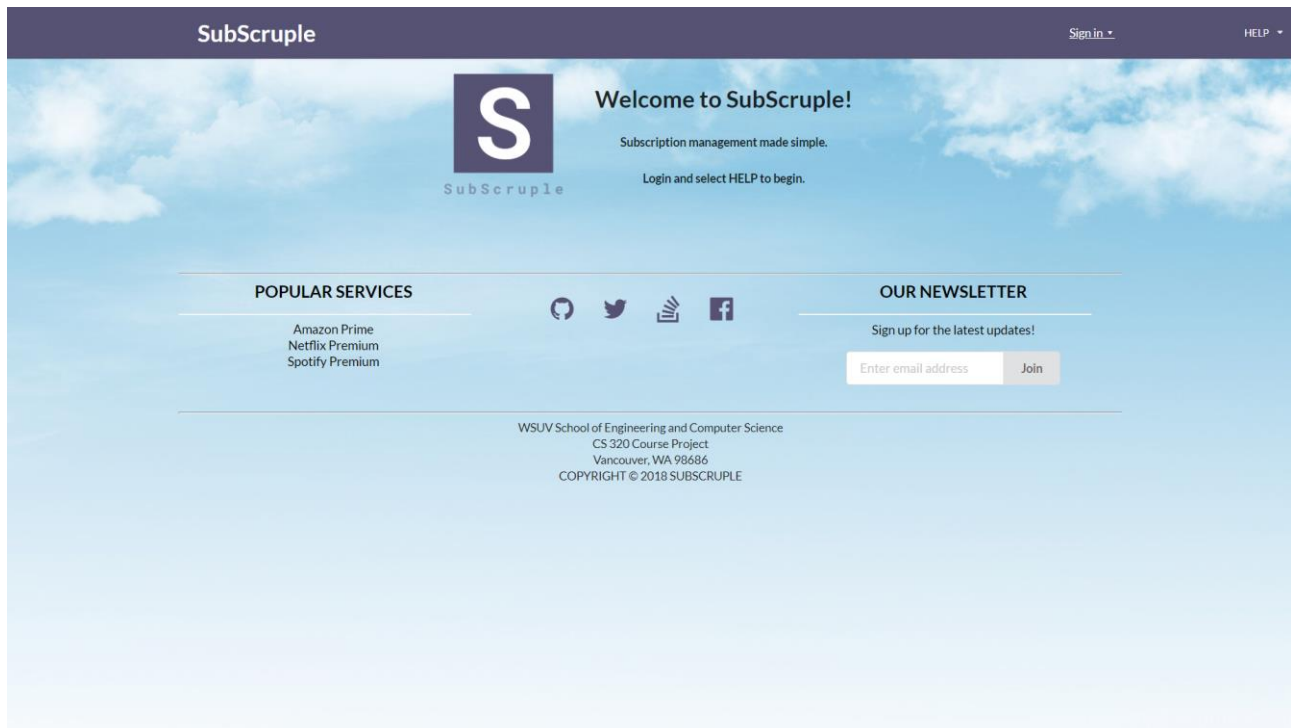


Figure 9. Homepage (Pre-login)

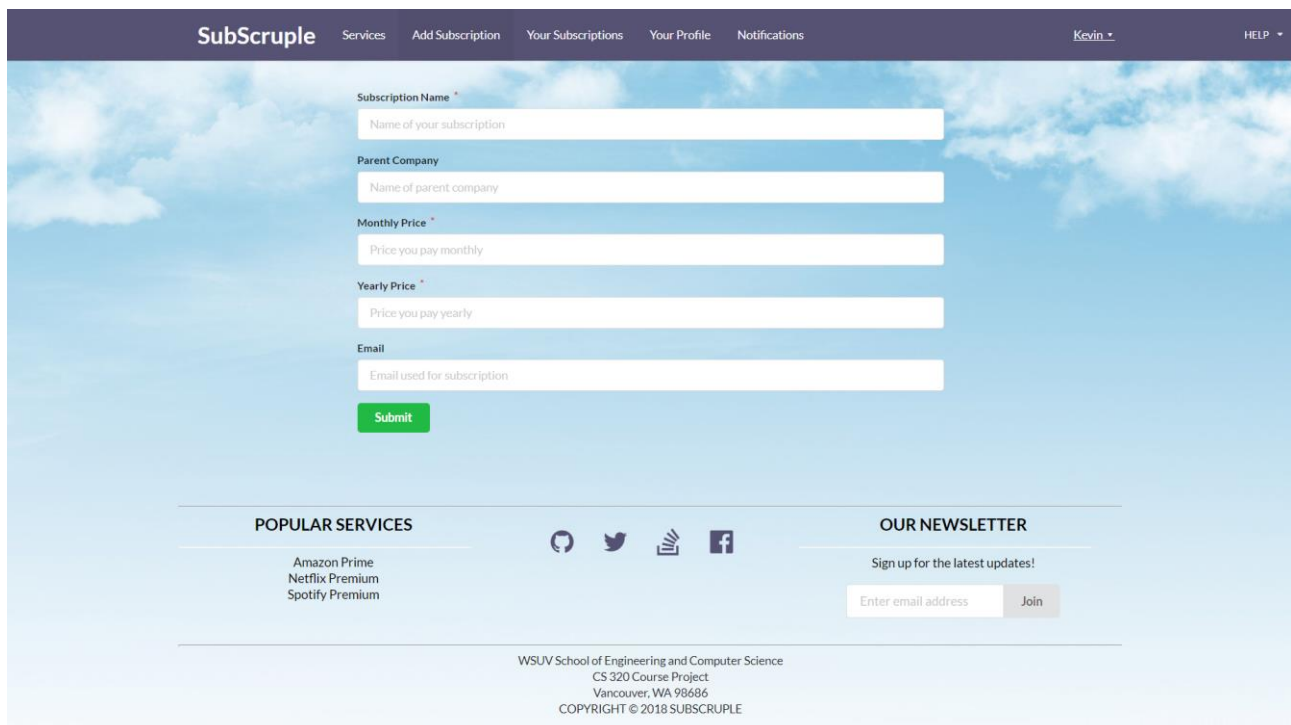
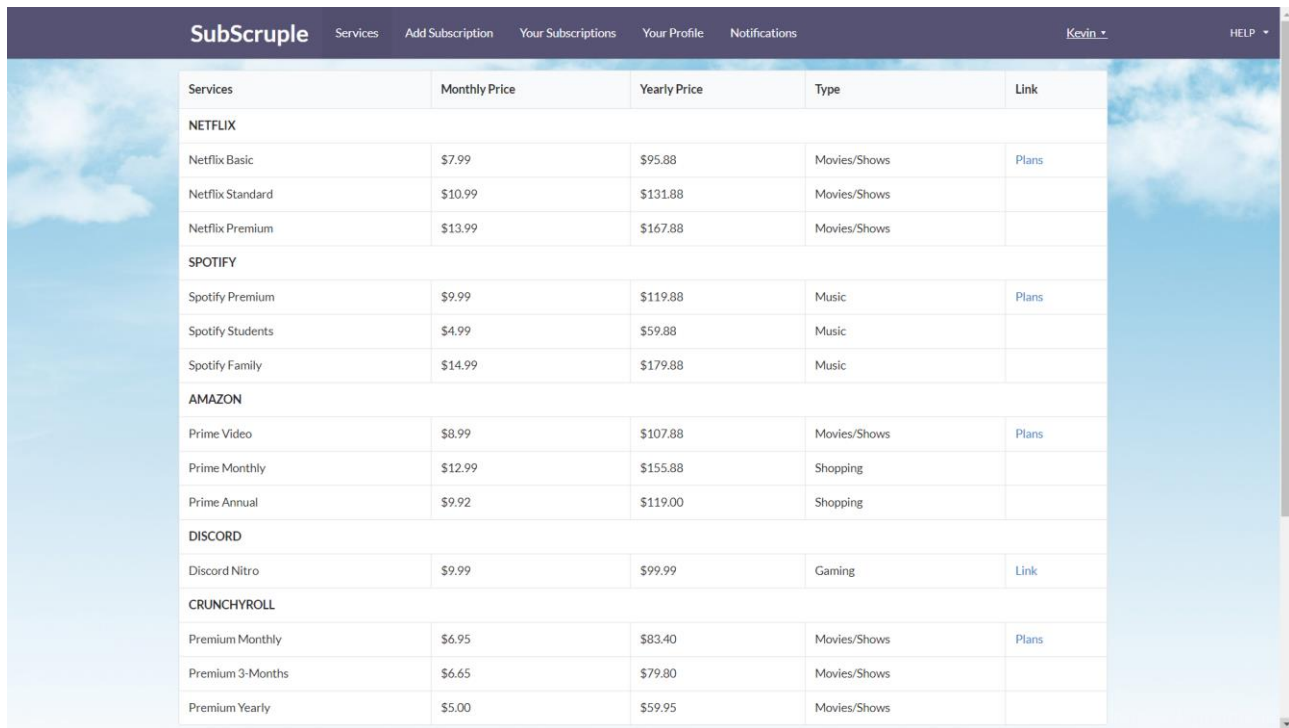


Figure 10. Adding a subscription to your profile.



Services	Monthly Price	Yearly Price	Type	Link
NETFLIX				
Netflix Basic	\$7.99	\$95.88	Movies/Shows	Plans
Netflix Standard	\$10.99	\$131.88	Movies/Shows	
Netflix Premium	\$13.99	\$167.88	Movies/Shows	
SPOTIFY				
Spotify Premium	\$9.99	\$119.88	Music	Plans
Spotify Students	\$4.99	\$59.88	Music	
Spotify Family	\$14.99	\$179.88	Music	
AMAZON				
Prime Video	\$8.99	\$107.88	Movies/Shows	Plans
Prime Monthly	\$12.99	\$155.88	Shopping	
Prime Annual	\$9.92	\$119.00	Shopping	
DISCORD				
Discord Nitro	\$9.99	\$99.99	Gaming	Link
CRUNCHYROLL				
Premium Monthly	\$6.95	\$83.40	Movies/Shows	Plans
Premium 3-Months	\$6.65	\$79.80	Movies/Shows	
Premium Yearly	\$5.00	\$59.95	Movies/Shows	

Figure 11. Listing the various subscriptions offered by different companies.

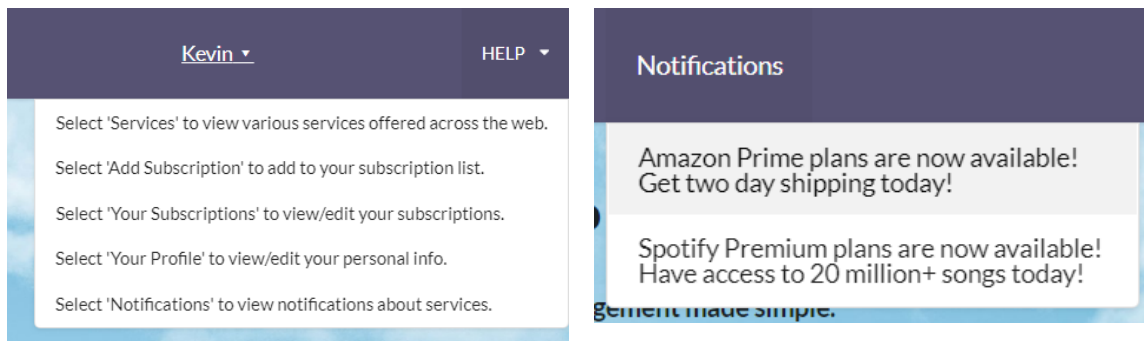


Figure 12. A user bringing down the help menu and listing their notifications, respectively.

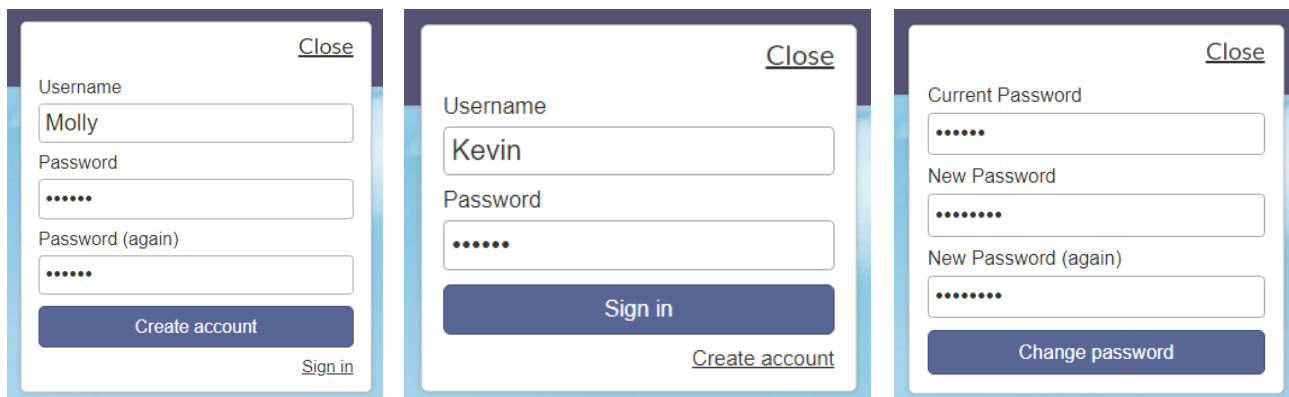


Figure 13. A user creating an account, logging in, and changing their password, respectively.





SubScruptle Services Add Subscription Your Subscriptions Your Profile Notifications Kevin ▾ HELP ▾

Subscription	Parent Company	Monthly Price	Yearly Price	Subscription Email	Edit
Netflix Basic	Netflix	\$7.99	\$95.88	kevin.black@wsu.edu	Edit
Spotify Students	Spotify Technology	\$4.99	\$59.88	kevin.black@wsu.edu	Edit

Monthly spendings: \$12.98
Yearly spendings: \$155.76

POPULAR SERVICES

Amazon Prime
Netflix Premium
Spotify Premium



OUR NEWSLETTER

Sign up for the latest updates!

WSUV School of Engineering and Computer Science
CS 320 Course Project
Vancouver, WA 98686
COPYRIGHT © 2018 SUBSCRUPLE

Figure 14. Example subscriptions owned by the user.

SubScruptle Services Add Subscription Your Subscriptions Your Profile Notifications Kevin ▾ HELP ▾

Name *

Kevin Black

Gender *

Male ▾

Opt in to Emails? *

Yes ▾

Biography





Me Kevin.

Need technical help?

Send us an email at kevin.black@wsu.edu.

POPULAR SERVICES

Amazon Prime
Netflix Premium
Spotify Premium



OUR NEWSLETTER

Sign up for the latest updates!

WSUV School of Engineering and Computer Science
CS 320 Course Project
Vancouver, WA 98686
COPYRIGHT © 2018 SUBSCRUPLE

Figure 15. Editing a user profile with some easy-to-understand functionalities.

3 Implementation

3.1 Development Environment

SubScruple was developed using two certain assignments from CS 320 as a foundation for the application, since a lot of the elements contained in those assignments were very similar to what we desired in our application, saving on time and utilizing reusability. The application was programmed in three development languages: JavaScript, HTML, and CSS. The IDE used for the entirety of this development was IntelliJ, as being able to enforce the coding standard provided by professor Xinghui Zhao was easy to install and follow in real-time. Distributed version control was of the utmost importance for communication of progress between group members and was done by using a public repository containing all the files for the application on GitHub. While not an absolutely necessary web application framework to work with for this project, we used Meteor due to its advantages in using a MEAN style stack, no HTTP request response cycle, and access to a local, client-side database. While testing the application, Meteor was ran on Windows 10 through the administrator version of Windows PowerShell.

3.2 Task Distribution

Each member of this team had specific tasks to perform, of which is listed below:

- Kevin Black: The main duties of Kevin consisted of creating: the initial implementation of the application, the software requirements specification, the software design document, and developing a majority of the final application features that were presented to the CS 320 course, such as: listing services, adding subscriptions, viewing subscriptions, editing profiles, listing notifications, designing the homepage, etc.
- Shane Bellika: His main duties were focused on tweaking the user interfaces in certain areas, altering the associated style sheet, and providing the background for the application.
- Alex Sloan: His particular task contributed was the initial idea/name for the application.

3.3 Challenges

Our group experienced a few daunting challenges in some of the early portions of our application, such as what properties to include for each user and subscription in two different databases, as well as how to implement them. This was the initial goal based off of a database style text file from an earlier assignment in CS 320, but this implementation was scrapped once we discovered Meteor in assignment 7 from the course mentioned earlier. A particular holistic challenge faced by our team was altering the original ideas we had in mind for this application, as certain features we wanted to implement just simply weren't possible within the time frame we had remaining for our project and would've required much more time to garner the necessary knowledge and research. After our team was able to pick out bits and pieces for our original idea of **SubScruple**, being able to implement features with less complexity proved to be much simpler and within our grade of knowledge.

4 Testing

In all honesty, the testing conducted for this application was lacking and definitely shouldn't be tried in a professional setting. The only reason for such a lax approach to testing for this project was that since the application incorporated elements from a few assignments in CS 320, the requirements and modules were well understood and working as expected. Thus, being able to implement new features or modify existing ones weren't difficult at all.

4.1 Testing Plan

Testing for this application was not done in a specific order or fashion. We considered it as prioritizing certain functionalities to receive the most attention, such as getting the subscription database to accept entries from the user, creating a separate database for user data, and so on. The time frame for which modules or functions we wanted to work on had no specific scheduling, more of just simply working on it and trying out different methods until they worked.

4.2 Tests for Functional Requirements

Not too many tests were conducted for functional requirements, as having another assignment as a foundation, a thorough understanding of the latter, and having semi-clear goals set in place since the beginning, made it pretty straightforward to implement certain features without having to go through rigorous testing. The only testing performed in this category would be making sure that the user was able to successfully submit and/or access subscription data from the database, as well as making sure the user could successfully save their respective profile data. The former ended up passing through various tests, but the latter has yet to be resolved.

4.3 Tests for Non-functional Requirements

Just like the testing section for functional requirements, not many tests were performed for these kinds of requirements, we mainly focused on how to prevent certain errors from arising by expecting them and implementing how to counteract them accordingly. As previously stated in the software requirements specification for **SubScruple**, some non-functional requirements that were to be addressed during development were: performance requirements, safety and security requirements, and various software quality attributes. How we went about handling these different requirements can be found below.

- **Performance Requirements:** Given the small scale implementation of this application and the client-side database, the response time for each functionality shouldn't take too long on modern systems. Updates to the database should also require refresh of the webpage thanks to how Meteor handles request-response cycles.
- **Safety/Security Requirements:** In the event of lost data or other security issues, the user can refer to the contact information in the 'Your Profile' section to contact the developers directly through email.
- **Software Quality Attributes:** For usability, one of the primary concerns for the developers in this group, we designed the application to contain simple to understand user interfaces that make navigating and maintaining the application pretty trivial. As for portability, all that the application requires is a stable HTTP protocol for communication over the internet for all of the functionalities, so the application should be suitable on any operating system. Once

again, a user is able to refer to the contact information in the 'Your Profile' section to contact the developers directly through email if there is a clarity issue that needs to be reaffirmed.

4.4 Hardware and Software Requirements

While this application shouldn't be physically demanding hardware-wise, performing tests on this application seemed to be heavily dependent on optimal software. The latter even applied to simply running the application on Windows 10, which is why I believe this application would perform exceptionally well if the program was ran on a Linux operating system. If one is to continue running this application from Windows 10 and experience issues in the installation process with Meteor, try some of the following solutions that worked for others: disable Windows virus scanners, add security exceptions for each executable used for installing Meteor, replace Meteor's built-in zip program, or remove the association of .js files to Windows Script Host. If none of these solutions work, it would be optimal to consider operating this application on a Linux based operating system, whether it is ran on native Linux hardware or ran through VirtualBox on Windows.

5 Analysis

The amount of time each member spent on each of the three milestones varies quite a bit, and is represented in the table below. As for which milestone took the most effort in terms of the amount of time spent, we came to a consensus that milestone one took the most effort as there were a wealth of sections that required researching on our own and wasn't taught in the CS 320 course. Milestone two didn't require as much thought since the software design document was composed mainly of figures, of which we were taught how to design, as well as being able to reuse content from the software requirements specification in our introduction. Although milestone one definitely took the most time, that doesn't mean it was necessarily congruent with difficulty.

Milestone Effort (hr)	Shane Bellika	Kevin Black	Alex Sloan
One	~1-2	16	~1-2
Two	1	14	1
Three	7	22	6-7

6 Conclusion

Over the course of the semester, we learned quite a few key things to keep in mind when developing this application as a group. Some of these things include but are not limited to:

Make clear the objectives and functionalities of the application by keeping the development within the scope of our original brainstorm. If not, we would alter our original goals to fit our new ideas. Generally, not having a set plan over the course of a project could potentially give one developers block.

Depending our development knowledge and the time frame of our project, we'll remain ambitious with our goals, but also be realistic by not setting impractical goals for ourselves. To limit setting impractical goals, we should definitely refer to the software requirements specification and software design document whenever we're unsure of how certain areas of our application function. Also, we must make sure to plan for other classes and outside material schedules that could potentially hinder project development. One cannot predict unforeseen events from life and one should be prepared, as with most things.

Appendix A - Group Log

Date	Duration	Members	Description
09/28/2018	20 min	All members	Very quickly discussed the group name, application name, and basic functionalities of the application. Also worked briefly on section 1 together.
12/07/2018	1 hour	All members	Worked on finishing up sections 5, 6, and group log on milestone 3.
Multiple dates	Multiple instances	All members	Most of the brainstorming, SRS creation, SDD creation, and Meteor development through GitHub was all communicated over the application Discord.