

CS 320 Course Project Final Report

for

SubScruple

Prepared by Kevin Black

Group Name: BBS teemMaets

|  |  |  |
| --- | --- | --- |
| Shane Bellika | 11639152 | shane.bellika@wsu.edu |
| Kevin Black | 11480710 | kevin.black@wsu.edu |
| Alex Sloan | 11651363 | alex.sloan@wsu.edu |

December 6, 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 2

3 Implementation 3

3.1 Development Environment 3

3.2 Task Distribution 3

3.3 Challenges 3

4 Testing 4

4.1 Testing Plan 4

4.2 Tests for Functional Requirements 4

4.3 Tests for Non-functional Requirements 4

4.4 Hardware and Software Requirements 4

5 Analysis 5

6 Conclusion 6

Appendix A - Group Log 7

# 

# Introduction

**SubScruple** 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 teemMaets. 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.

## 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 ofour 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 indepth 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.

## 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. IDE | Integrated Development Environment, where the programming is performed. |
| 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. |
| MEAN | A development technique composed of MongoDB, Express.js, AngularJS, and Node.js. |
| HTTP |  |

## 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/>

# Design

## 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 privleges, and a state diagram for profile management. The few exceptions for the system modeling diagrams that changed or are still relavent to the current state of the application can be found below.

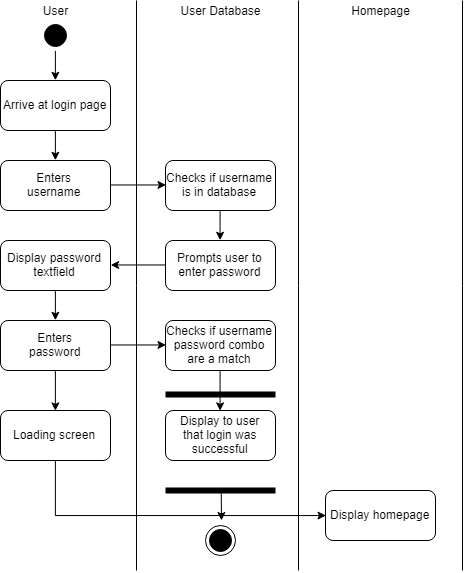


Figure 1. Activity diagram depicting a user logging in to their account.

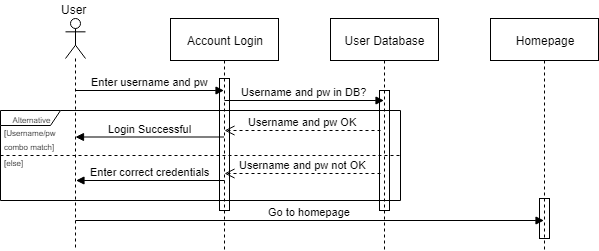


Figure 2. Sequence diagram depicting a user logging in to their account.

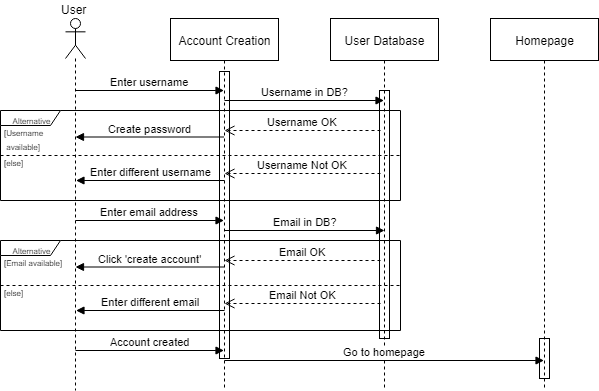


Figure 3. Sequence diagram showing how a user would create their account.

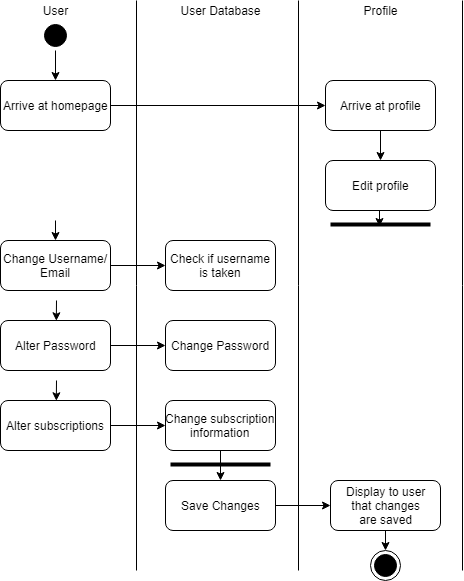


Figure 4. Activity diagram showing how a user can manage their profile.



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

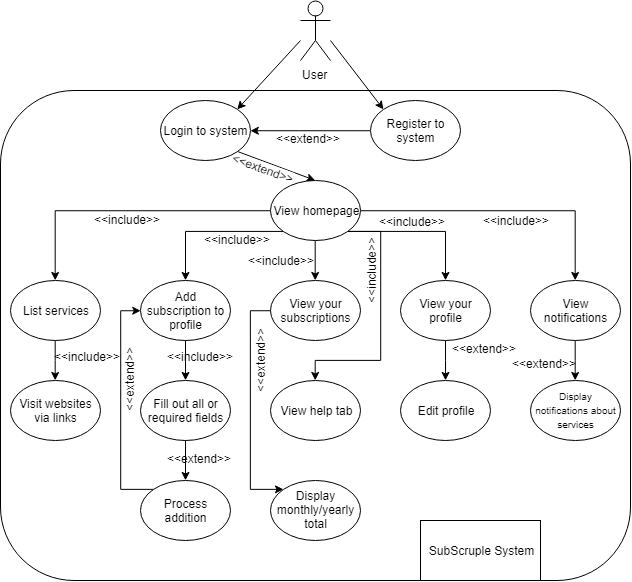


Figure 6. The overall system user case of SubScruple with slight variations from the initial concept.

CREATE NEW DIAGRAMS FOR:

* Figure 7. Class diagram overview for **SubScruple**.
* Figure 8. State diagram representation of listing services.
* Figure 9. Activity diagram for adding/viewing subscriptions.
* Figure 10. Activity diagram showing for viewing profile/notifications.

## Interface Design

Figure 11. Homepage (Pre-login)

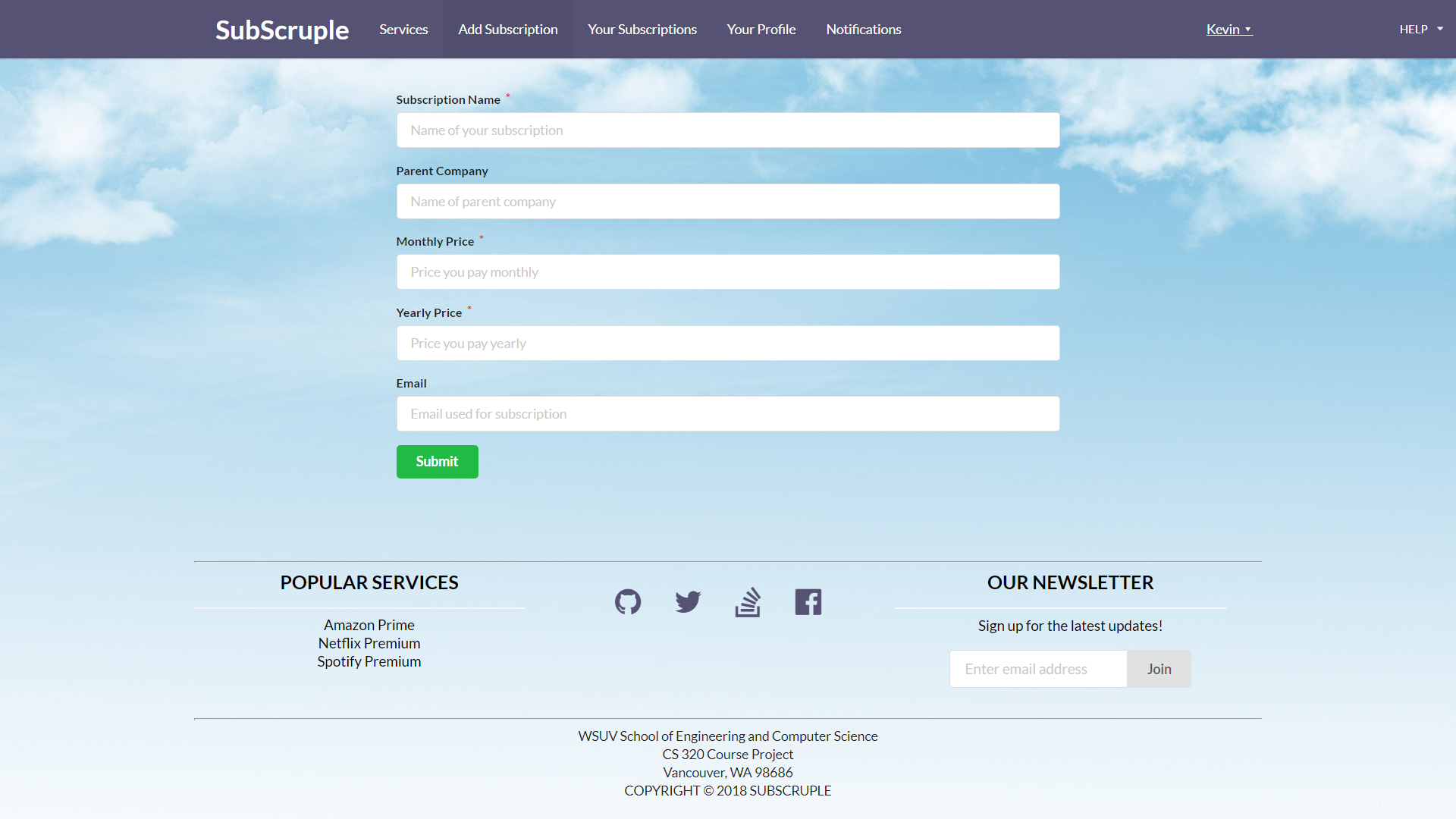


Figure 12. Adding a subscription to your profile.

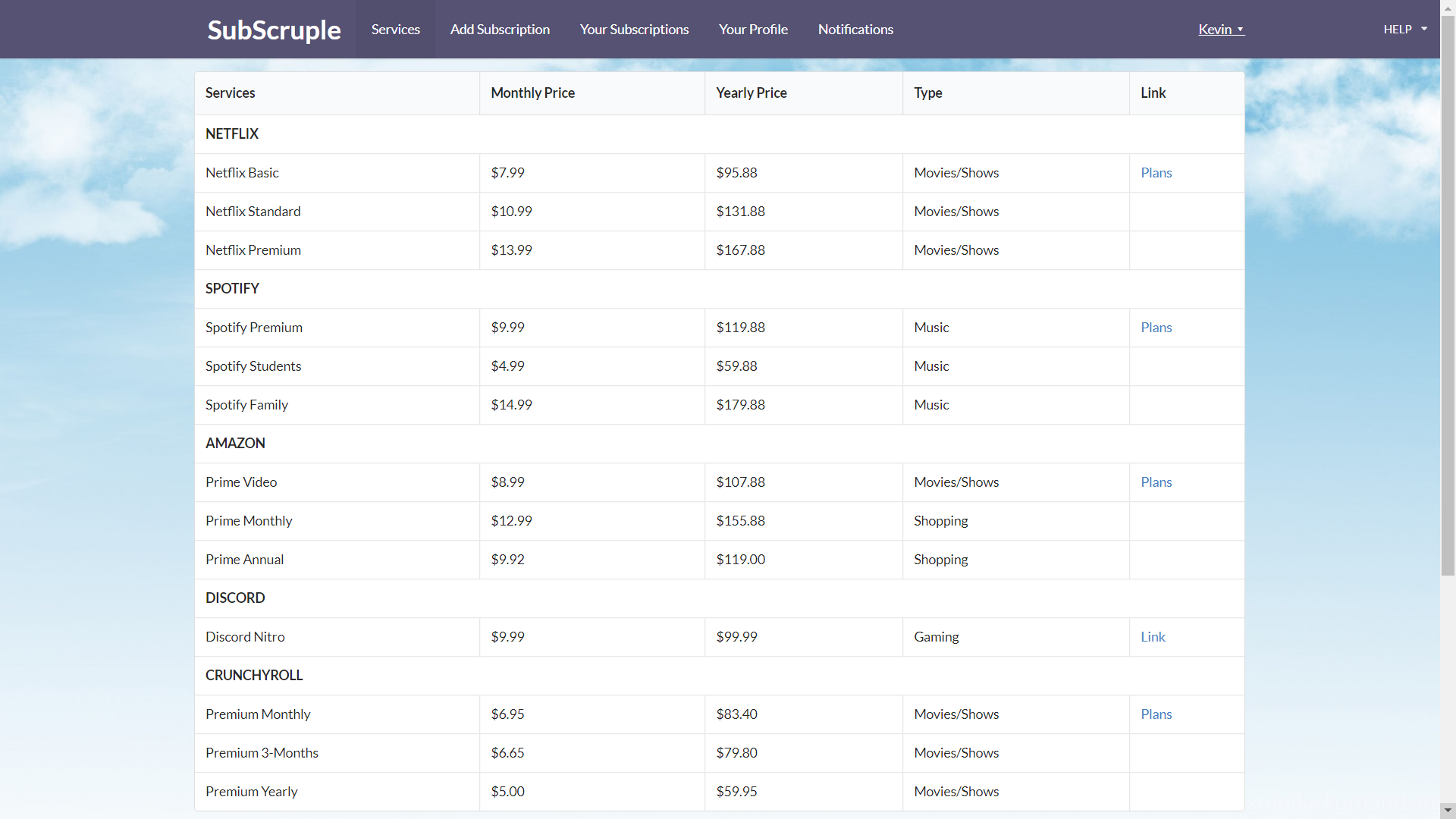


Figure 13. Listing the various subscriptions offered by different companies.

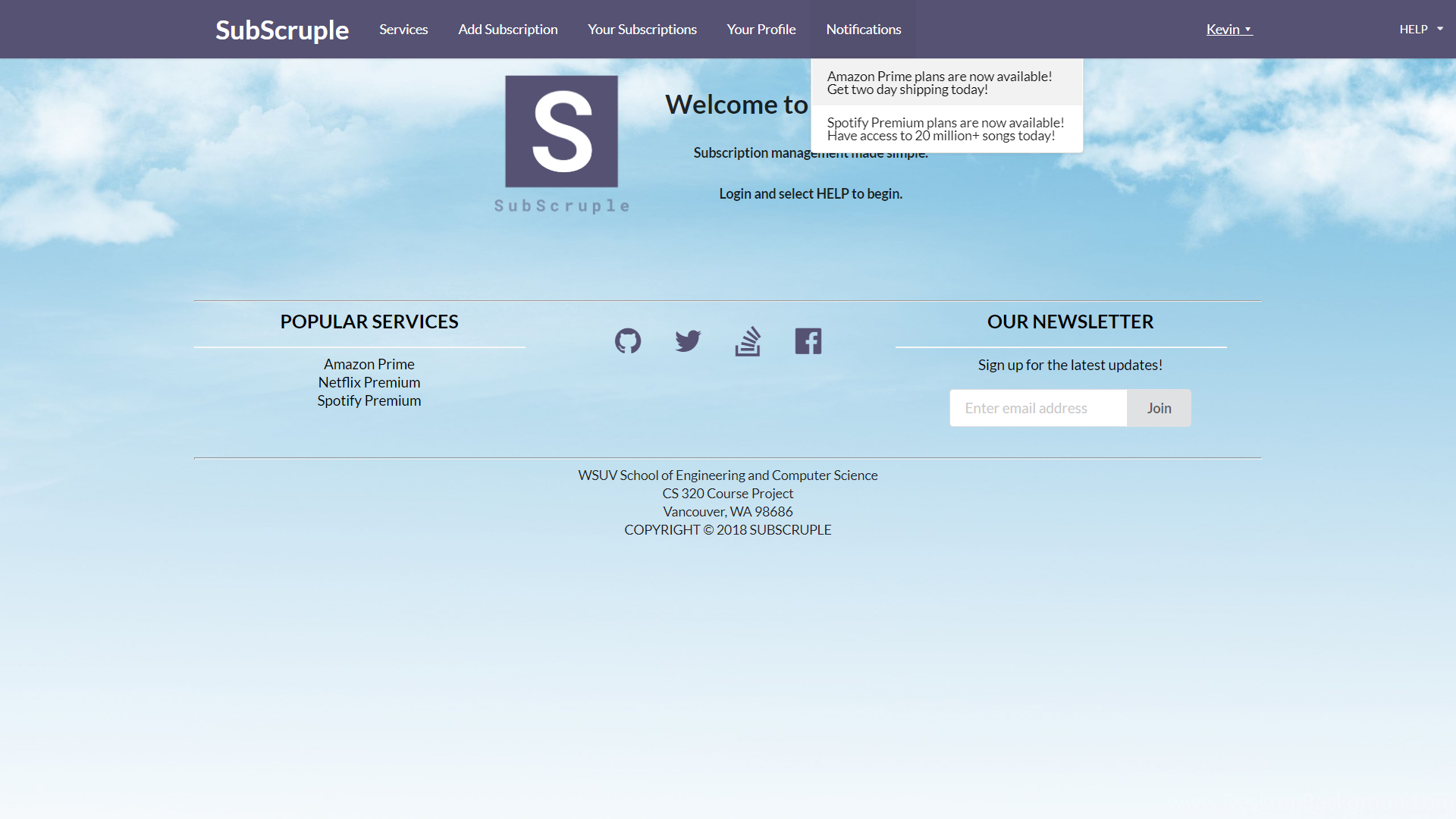


Figure 14. A user bringing down the help menu and listing their notifications, respectively.

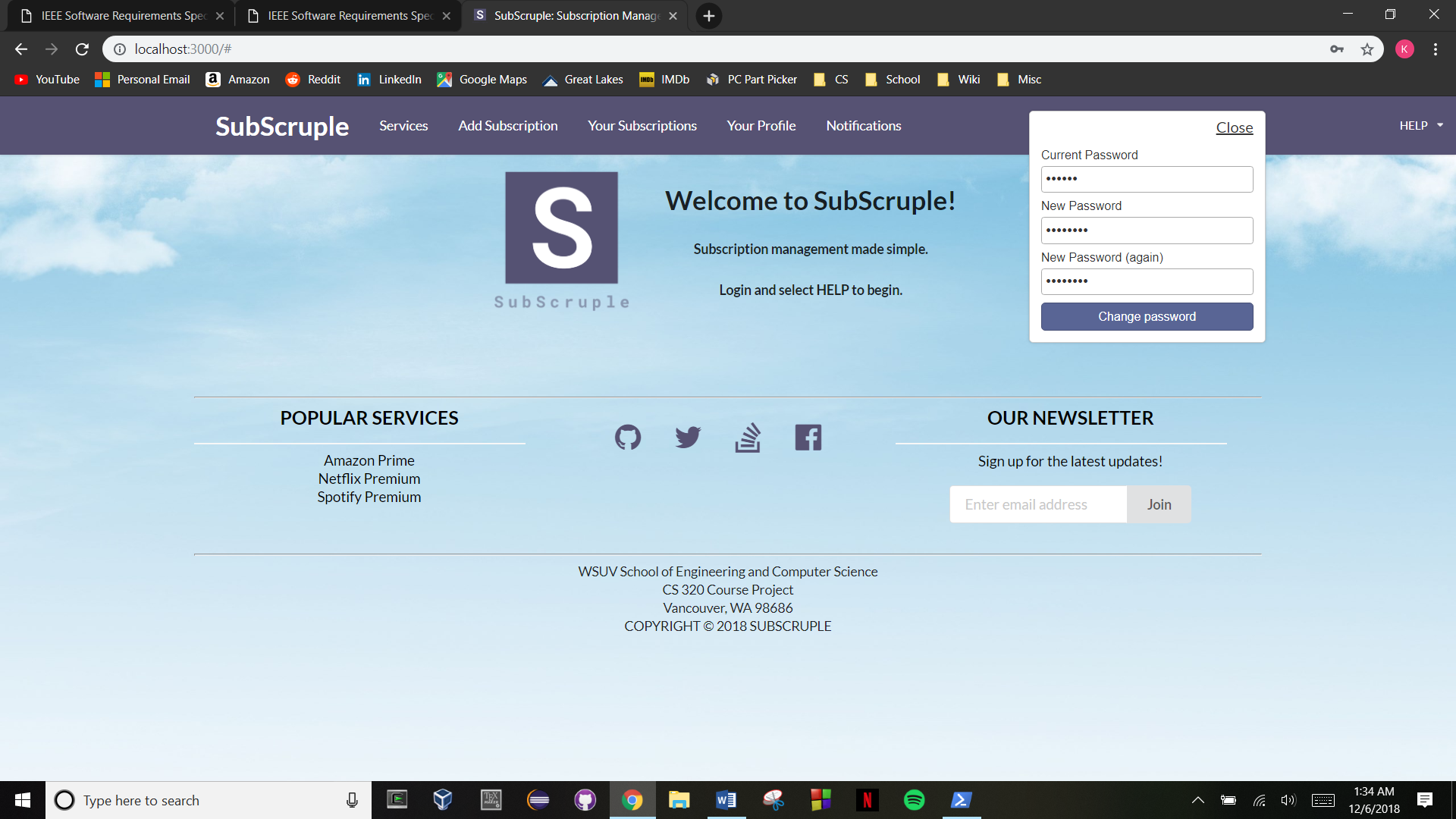
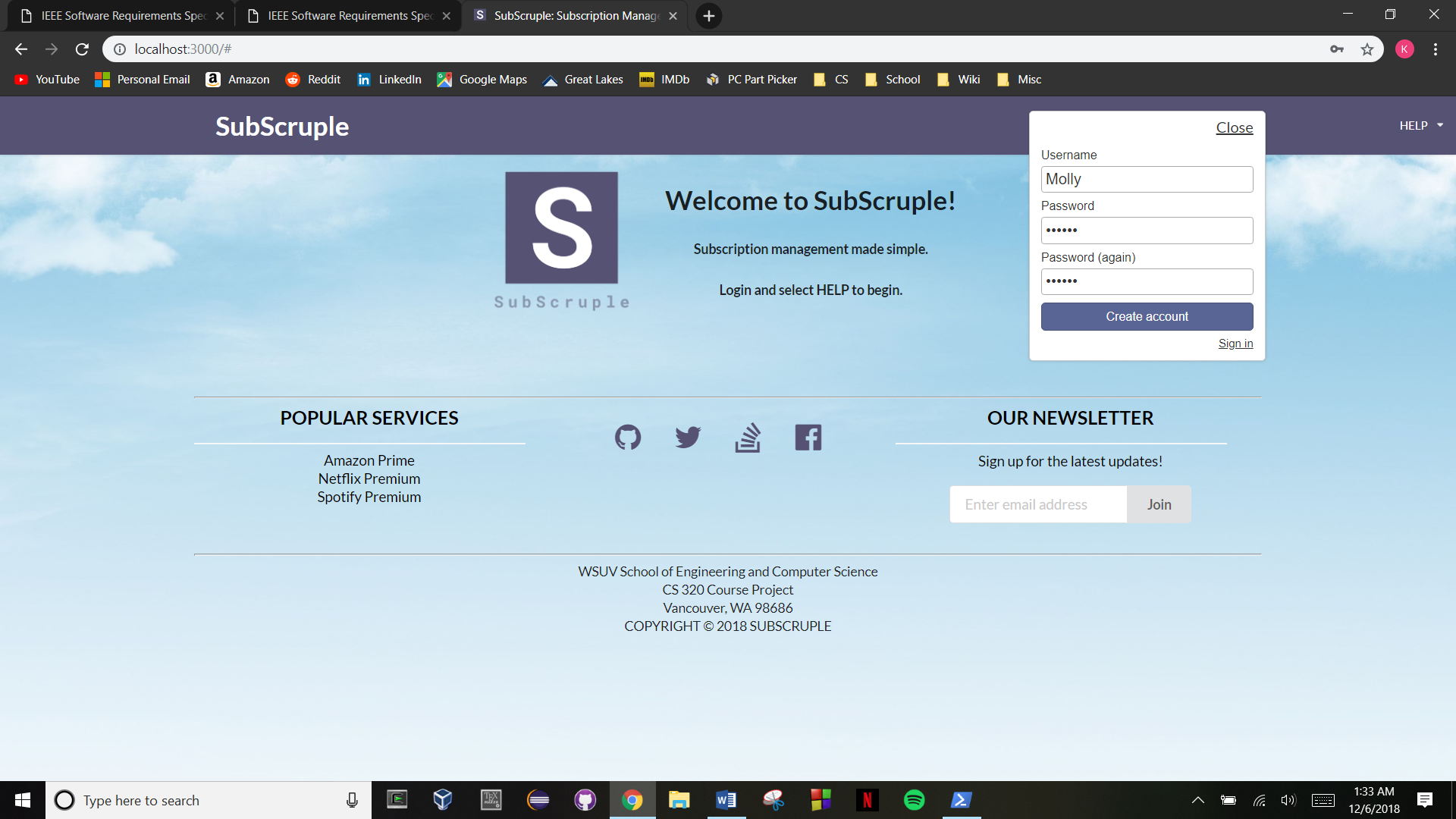
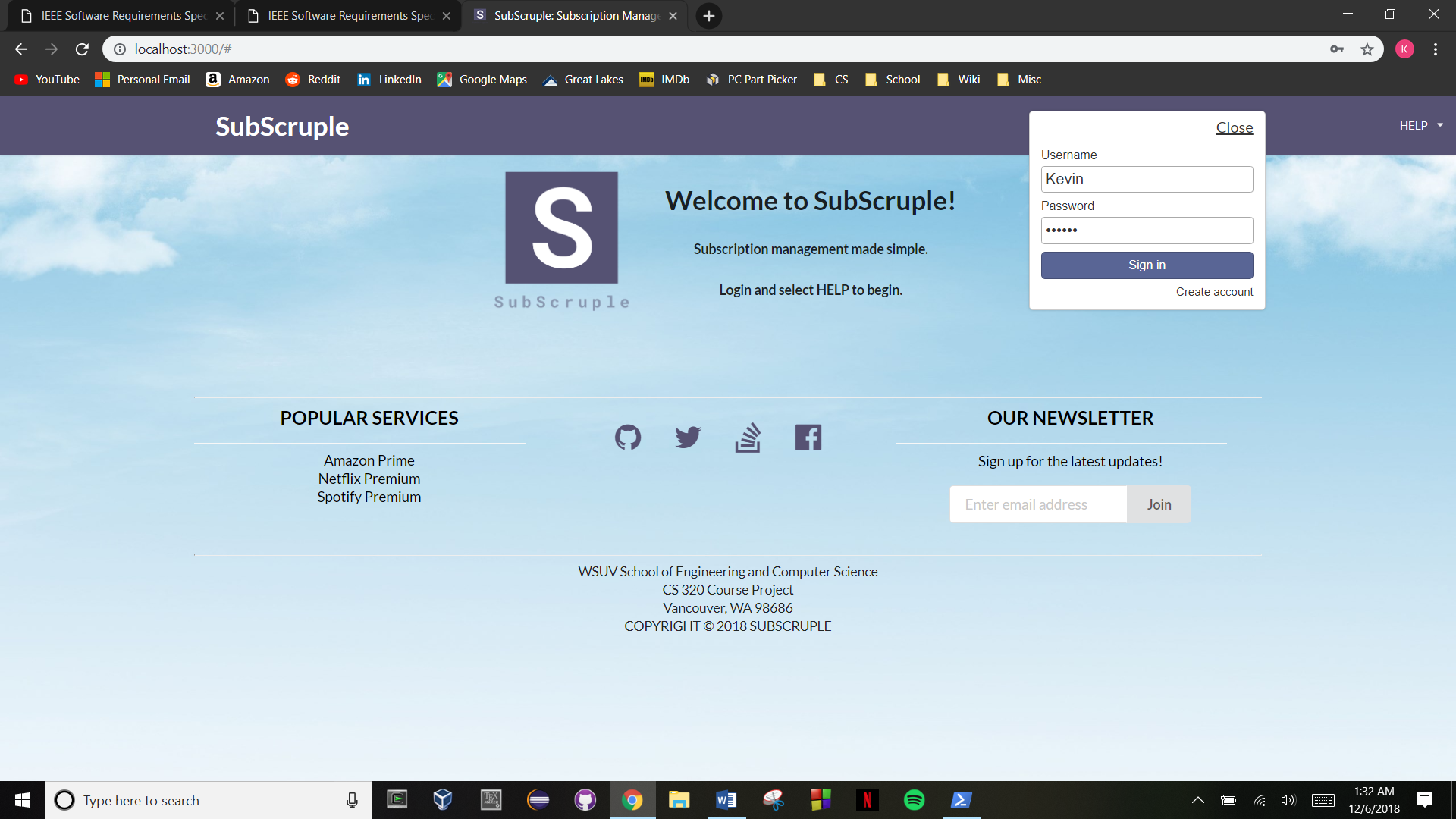


Figure 15. A user creating an account, logging in, and changing their password, respectively.

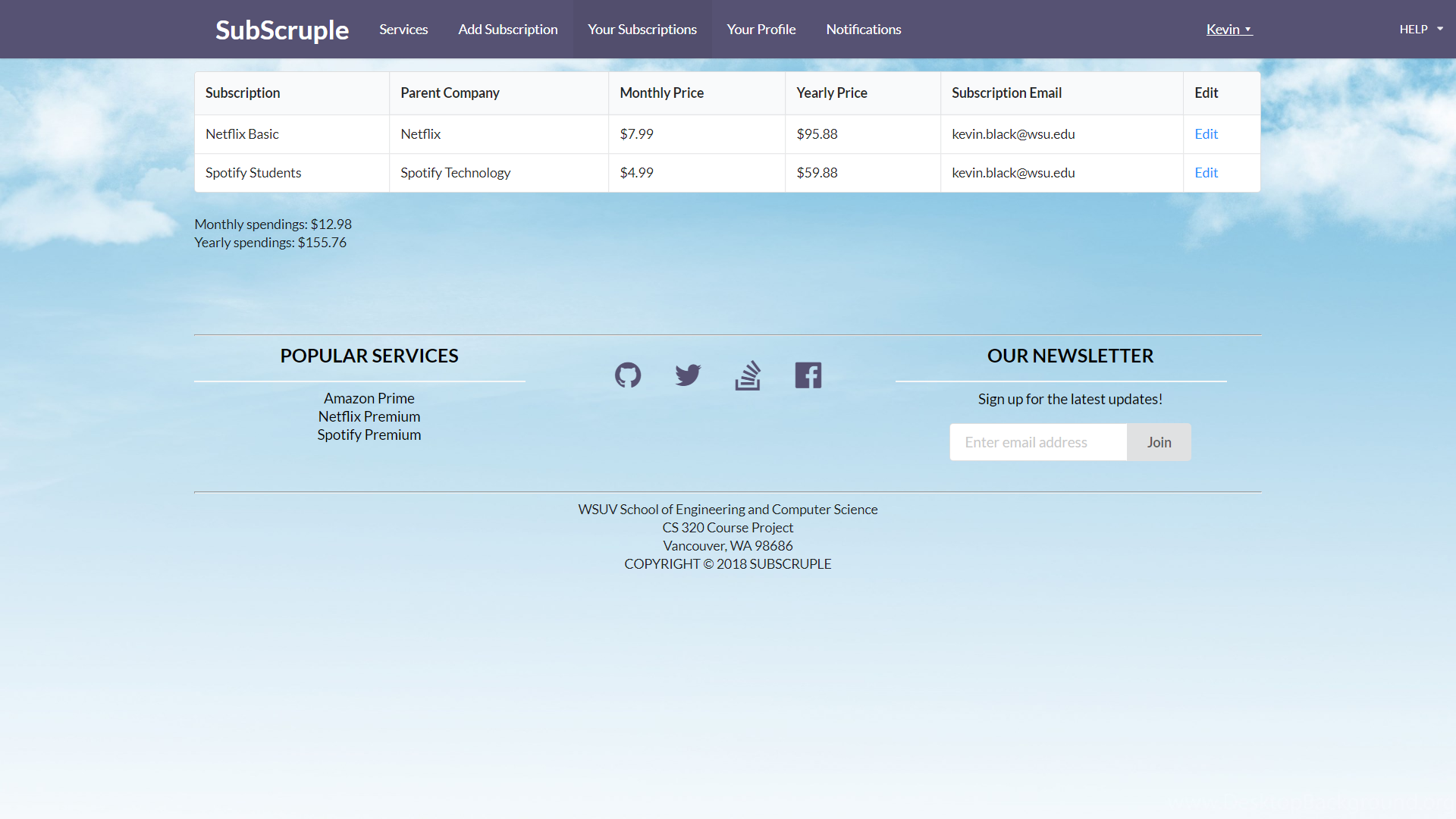


Figure 16. Example subscriptions owned by the user.

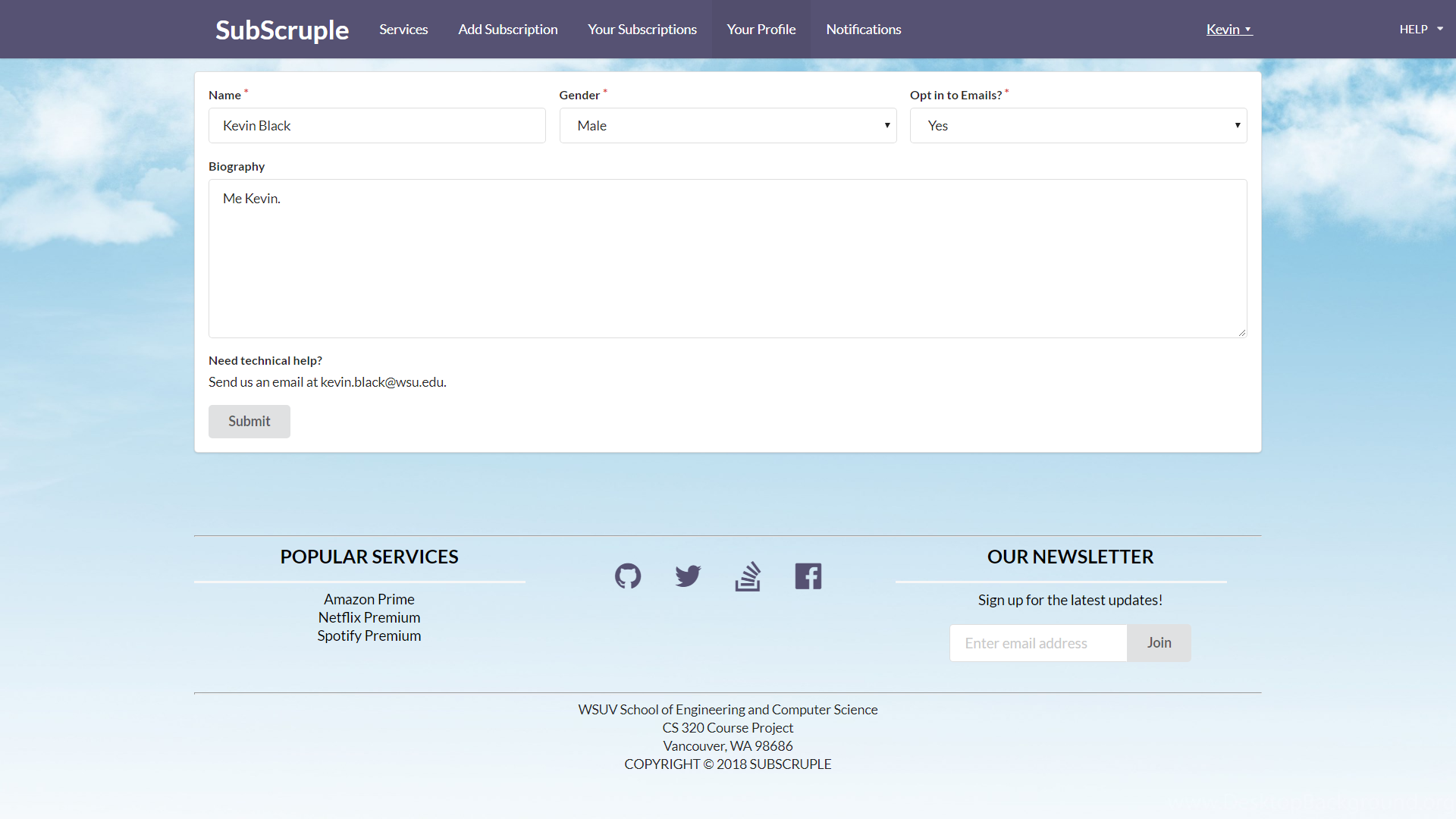


Figure 17. Editing a user profile with some easy-to-understand functionalities.

# Implementation

## 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.

## 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.

## 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.

# 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.

## 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 worked on had no specific scheduling, more of just simply working on it and trying out different methods until they worked.

## 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.

## 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 emial if there is a clarity issue that needs to be reaffirmed.

## 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 though VirtualBox on Windows.

# Analysis

<In this Section you need to analyze the effort that has been put on this project.

TODO: Describe how many hours (approximately) each team member spent on the project, for each milestone, which milestone takes the most effort and why. >

# Conclusion

<Conclude the document with what you have learned through working on the project.>

Appendix A - Group Log

< Describe how frequently the group meembers meet during the semester, and how effective the communication is. This is optional for one-person projects.>