Quarter Black

Tempeters Project – CSI 3372 Software Engineering II

Matt Dulany, Dylan Hines, Marcus Hollingsworth, Collin Wiginton

Dec 4th, 2017

Contents

[Requirements Documents 2](#_Toc500163836)

[Use Case Diagram 2](#_Toc500163837)

[Use Cases 2](#_Toc500163838)

[Brief Cases 2](#_Toc500163839)

[Casual Cases 3](#_Toc500163840)

[Detailed Case 3](#_Toc500163841)

[Domain Model 4](#_Toc500163842)

[SSD 5](#_Toc500163843)

[Operation Contracts 5](#_Toc500163844)

[Architectural Model 7](#_Toc500163845)

[Design Documents 8](#_Toc500163846)

[Design Class Diagram 8](#_Toc500163847)

[SDs 8](#_Toc500163848)

[Design Overview 10](#_Toc500163849)

[Testing Documents 11](#_Toc500163850)

[Test Plan 11](#_Toc500163851)

[Unit Tests 14](#_Toc500163852)

[Functional Tests 14](#_Toc500163853)

[Installation 15](#_Toc500163854)

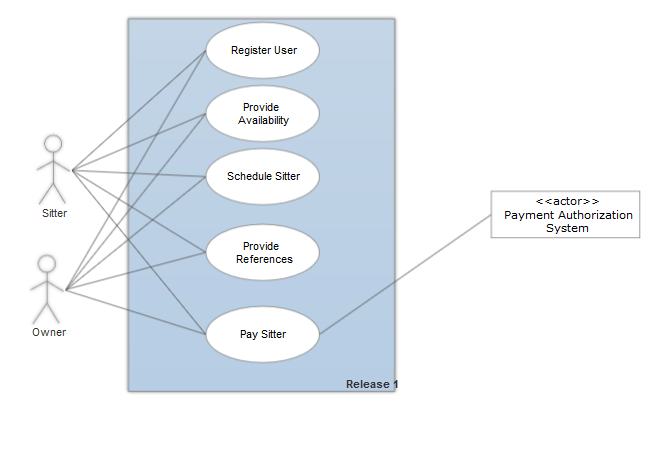
[User Manual 15](#_Toc500163855)

[Work Summary 17](#_Toc500163856)

[Meeting Summary 18](#_Toc500163857)

# Requirements Documents

## Use Case Diagram



## Use Cases

### Brief Cases

Register User: The user selects the register option. The system then prompts the user for information about themselves such as how old they are and what kind of pets they prefer to sit for. Once the user enters all the information, the system will verify and save it.

Provide Availability: The user opens their personal calendar and logs their unavailable times for sitting. The system then verifies these logs and the changes are saved to the user’s profile.

Schedule Sitter: The owner selects the desired time to have their pet watched and the urgency of the situation. The system then displays matched users. The owner browses the matched sitters and selects the one that they feel best fits. The chosen sitter then gets a notification from the system. The sitter accepts the task and the appointment is logged on both accounts.

Provide Preferences: The user opens their profile and logs their pet preferences. The system verifies this information and then saves the changes.

Pay sitter: The owner reviews the bill and chooses to pay the bill. The system verifies the payment with a third-party authentication service and it is sent to the sitter’s account, along with a notification.

### Casual Cases

Provide Availability:

Main Scenario: The user opens their personal calendar and logs the time they are not available during the week for sitting. The system then verifies these logs and the changes are saved to the user’s profile.

Alternate Scenario 1: The user would like to add a onetime unavailable time to his calendar. User selects time on calendar. System displays calendar time. User selects onetime occurrence. User chooses the save calendar. System notifies user that calendar has been update.

Alternate Scenario 2: User is tries to add an unavailable time to their schedule when a sitting is already scheduled. System asks user if they want to cancel sitting appointment (not available inside of 24 hours). User selects they would like to cancel adding the appointment. (Not completely implemented)

Schedule Sitter: The owner selects the desired time to have their pet watched and the urgency of the situation. The system then displays matched users. The owner browses the matched sitters and selects the one that they feel best fits. The chosen sitter then gets a notification from the system. The sitter accepts the task and the appointment is logged on both accounts.

Alternate Scenario 1: Sitter declines appointment. The owner is notified that the sitter has declined the appointment. The system the provide available alternates to the owner and returns to the main scenario of selecting a sitter.

Alternate Scenario 2: No sitters are available during that time. The system will first check if any sitters are available for an emergency sitting. If there exist such a sitter the owner will be notified of the sitter’s availability but will be charged extra for the situation. The owner chooses to cancel the pet sitting. If no sitter is available at all the system notifies the user. (Not completely implemented)

### Detailed Case

**Use Case UC1: Schedule Sitter**

**Scope**: *Tempetūrs* system

**Level**: user goal

**Primary Actor**: Pet owner

**Frequency of Occurrence**: high

**Preconditions**:

* Pet owner has successfully created an account and provided owner profile information.
* Pet owner is logged in.

**Postconditions**:

* An appointment is logged on both Owner and Sitter accounts.
* Both Owner and Sitter are notified of a confirmed appointment.

**Main Success Scenario**:

1. Pet owner provides the desired time to have their pet watched, urgency of the situation, and other preferences.
2. System displays sitters available during that period which best match Owner’s preferences.
3. Owner browses matched sitters and selects his or her preference.
4. System notifies matched Sitter of the potential appointment.
5. Sitter accepts the appointment.
6. System logs the appointment on the accounts of both Owner and Sitter.

**Alternate Scenarios**:

2a. No sitters are available during the selected time.

1. System checks if any sitters are available for an emergency sitting.
2. If there is an emergency sitter available, System notifies Owner and assesses an additional emergency appointment fee.
3. Owner chooses to confirm an emergency appointment (if available) or cancel the request.

5a. Sitter declines the appointment.

1. Owner is notified that Sitter has declined the appointment.
2. System provides available alternates to the owner and proceeds back at step 3.

## Domain Model



## SSD



## Operation Contracts

1. getSitters(startTime, endTime, urgency, preferences)

Use Cases: Schedule Sitter

Pre-Conditions:

* OwnerEndpoint exists as an appointmentScheduler.
* OwnerEndpoint has been associated to the sitterCatalog
* Current Owner o is logged in and authenticated

Post Conditions:

* Available Sitters who fit above parameter restrictions have been found.
* appointmentSchedule associated with available sitters.

1. scheduleSitter(sitter, appointment)

Use Cases: Schedule Sitter

Pre-Conditions:

* Current Owner o is logged in and selecting a sitter.
* Sitter sitter has been selected by user
* appointment contains all details about the requested appointment

Post Conditions:

* Appointment A has been created and its members are set to data in appointment.
* A is associated to sitter.
* A is associated with o.
* A notification is created and associated with sitter.

1. acceptAppointment(a)

Use Cases: Schedule Sitter

Pre-Conditions:

* A has been associated with sitter.
* A notification has been sent to sitter.

Post Conditions:

* A’s appointmentStatus has been updated to reflect accepted.
* An accepted notification has been created and is associated with o.
* An upcoming appointment notification has been created and is associated with sitter and o.

# Architectural Model

User Interface

React

\* Unimplemented

\*calculatePayment()

\*changePayment()

payAppt()

\*verifyPayment()

Payment

getOwner()

getUser()

getSitter()

getPet()

regPet()

regUser()

regOwner()

regSitter()

UserInfo

registerAppt()

sortSitters()

createBlock()

acceptAppt()

denyAppt()

cancelAppt()

getAppointments()

getNotifciations()

Schedule

Alexa

Alexa Interface

High Rest Client Interface

ElasticSearch

Payment

Schedule

UserInfo

Alexa

Elastic Search

Database

Voice Recognition

(Unimplemented)

Rating & Preference Manager

Payment Manager

Appointment Scheduler

# Design Documents

## Design Class Diagram



## SDs

GetSitter



ScheduleSitter



acceptAppointment



DenyAppointment (Alternate Senario)



# Design Overview

In our project there are three main areas of patterns that we are using. In our Java backend you can see a few of these types of patterns. First, we are using a Strategy pattern to determine the type of sort we want to use on our sitter suggestions. There are six possible sorting algorithms that the user can chose from and based on their choice the program responds with that type of algorithm. This is all encapsulated inside of one sorting class. Another method that we are using is a façade pattern. The multiple service classes that we have act as a façade between the data repository and the user API. This is designed like this and we have several functions that act like this. Due to convenience and time not, all functions have been put behind this façade at this point. If we have time I would like to refactor our code to abstract many of the function like this, but the façade pattern has been used for much of the functionality. The main code that I am talking about is I would like the class DAO objects to contain the code to connect to the elastic search database in order to abstract this away from the endpoint controllers. Due to time and a need to ensure that the connection was working properly I we haven’t implemented this yet, but would be a good improvement to make if we have time to refactor. Another pattern we are using in many areas in the singleton pattern. We have singleton versions of each one of our service, DAO and endpoint classes. This is important to ensure that only one class of each is used so that we do not have multiple copies of data as well as confusion behind what instance of a class we are connected with. The main place this is important is we have a singleton for our RestClient connection class to the elastic search database. We need to ensure this is true so that we only open one connection to the database.

Another type of pattern we are using is the architecture pattern MVC for our react and spring connections. The front-end web pages controlled by react are the view of the system they reflect any changes that we make to the model. Spring controls the backend and is our controller in the MVC making changes to both the model and reflecting those changes in the view. The elastic search database represents our model of the data and the overall connections of the system. All information regarding users and their appointments are stored in this database. The backend retrieves this data when called and returns this data to the frontend.

Lastly, we are using several GRASP patterns in our system. We have many instances of a controller class. All our endpoints act as a controller to delegate and ensure that each command is fulfilled correctly. We are also using the information expert pattern. Our services are experts on their specific type of data. They know how to retrieve and return the data required about a specific type of object. Lastly, spring is acting as our creator. It is responsible for creating these objects (many of which are created as singletons) and then passing those objects to the classes that need them.

# Testing Documents

## Test Plan

Introduction

Purpose

The goal of this document is to:

1. Identify the components of the Tempeturs system that need to be tested.
2. Define the requirements needed to test the system.
3. Layout testing strategies
4. Identify the resources needed for testing purposes
5. List deliverables to be created by testing.

Background

The Tempeturs system is a pet sitter matching site, designed to match possible pet sitters with pet owners. This system contains three main systems. The online front end needed to display information and provide content to user. The java based backend needed to connect users to each other and the elastic search database where user information is stored.

Scope

This front end and backend portion of this system will be tested separately. The front end will be tested manually, while the backend will be unit tested. The overall system will also be tested manually tested.

Backend unit tests will cover the following:

1. The connection to the elastic search database.
2. Interaction between subsystems
3. System functionality

Requirements

The following list lays out the requirements that testing must accomplish.

Database Testing

Must ensure that users information can be entered and stored correctly

Must ensure that user data can be retrieved correctly

Must ensure that appointment data can be entered and retrieved

Must ensure the payment information can be entered and retrieved.

Functional Testing

Verify that a user can be registered successfully

Verify that a user can be logged in correctly

Verify that a pet owner can add a pet as needed

Verify that sitters can enter calendar appointments correctly

Verify that Owners can create pet sitting appointments

Verify that Sitters can accept possible sitting appointments.

Verify that Owners can rate past sitters

Verify that payment transactions are processed correctly

Verify that sitter suggestions can be made for appointments

Verify that alexa can communicate with the system to create an appointment

User Interface Testing

Navigate through each use case verifying that each page can be easily understood

Verify that all documentation is presented clearly

Performance Testing

Verify that request minimize delay

Verify that responses are received without backend delay

Security Testing

Verify Authentication and Online security measures protected data

Strategy

##### Data Integrity and Database Integration Testing

|  |  |
| --- | --- |
| Test Objective: | Ensure database access methods process data properly and without data corruption. |
| Technique: | * Invoke each database access method and process, seeding each with valid and invalid data (or requests for data). * Inspect the database to ensure the data has been populated as intended, all database events occurred properly, or review the returned data to ensure that the correct data was retrieved (for the correct reasons) |
| Completion Criteria: | All database access methods and processes function as designed and without any data corruption. |
| Special Considerations: | * Processes should be invoked manually. * Small or minimally sized databases (limited number of records) should be used to increase the visibility of any non-acceptable events. |

##### Functional Testing

|  |  |
| --- | --- |
| Test Objective: | Ensure proper target-of-test functionality, including navigation, data entry, processing, and retrieval. |
| Technique: | Execute each use case, use case flow, or function, using valid and invalid data, to verify the following:   * The expected results occur when valid data is used. * The appropriate error / warning messages are displayed when invalid data is used |
| Completion Criteria: | * All planned tests have been executed. * All identified defects have been addressed. |
| Special Considerations: | None. |

##### User Interface Testing

|  |  |
| --- | --- |
| Test Objective: | Verify the following:   * Navigation through the target-of-test properly reflects business functions and requirements, including page to page methods * Web objects and characteristics, such as menus, size, position, state, and focus conform to standards. |
| Technique: | Manually run through each page looking for specific criteria. |
| Completion Criteria: | Each window successfully verified to remain consistent with benchmark version or within acceptable standard |
| Special Considerations: | None |

##### Performance Profiling

|  |  |
| --- | --- |
| Test Objective: | Verify performance behaviors for designated transactions or business functions under the following conditions:   * normal anticipated workload * anticipated worse case workload |
| Technique: | Manually verify that data stats for process meet acceptable standards. |
| Completion Criteria: | All statics conform to an acceptable level. |
| Special Considerations: | None |

Resources

We will be using JUnit to perform backend testing. Matt Dulany will be the test engineering in charge of performing and meeting each test requirement as well as ensuring that each test meets completion criteria. These tests will be developed throughout the development process. The will be developed and run alongside functional development to ensure that each step is being designed correctly.

Deliverables

Test Model

This document will be a guide as to what and how each test will need to be created in order to ensure that each test is completed correctly. It is due as of 10/13/2017.

Test Documentation

Logs and output of each test run will be collected created in word documents

### Unit Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Scenario | Inputs | Expected Result | Actual Result | Tester |
| 1 | Test User Registered Correctly | User is Username jwild7777 | Inputed User equals registered User | Inputed User equals registered User | Matt Dulany |
| 2 | Test Owner Registered Correctly | Owner is Username jwild7777 | Inputed Owner equals registered User | Inputed Owner equals registered User | Matt Dulany |
| 3 | Test Sitter Registered Correctly | Sitter is Username jwild7777 | Inputed Sitter equals registered User | Inputed Sitter equals registered User | Matt Dulany |
| 4 | Test Pet Registered Correctly | Pet with next id in database | Inputed Pet equals registered User | Inputed Pet equals registered User | Matt Dulany |
| 5 | Test Calendar Appointment Created Successfully | Calendar appointment with next id in DB | Input appt equals stored appt | Input appt equals stored appt | Matt Dulany |
| 6 | Test Sitters Sort function | Three sitters with different values for zip, preferences and rating | Sitters sorted in descending order of score | Sitters sorted in descending order of score | Matt Dulany |
| 7 | Test rating function with one previous rating | New rating of 5 for sitter with old rating of 3 | New rating should be 4 |  | Matt Dulany |
| 8 | Test rating function with no previous ratings | New rating of 5 for sitter | New rating should be 5 |  | Matt Dulany |

### Functional Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Scenario | Inputs | Expected Result | Actual Result | Tester |
| 1 | Schedule appt | Schedule appt from front end with actual values | New appt on schedule with appt displaying | New appt on schedule with appt displaying | Matt Dulany |
| 2 | Schedule unavailable block of time | New unavailable time | New block on calendar | New block on calendar | Matt Dulany |
| 3 | Cancel appointment | Cancel chosen from sitter of owner side | Appointment removed from schedules | Appointment removed from schedules | Matt Dulany |
| 4 | Accept Appointment | Sitter Accepts appointment | Appointment is accepted, notifications are created, and sitter can now cancel the appt | Appointment is accepted, notifications are created and sitter can now cancel the appt | Matt Dulany |
| 5 | Deny Appointment | Sitter Denys appointment | Appointment is removed from schedules and notifications are created |  | Matt Dulany |
| 6 | Owner and Sitter Reverse switch | Users can switch between being a sitter and an owner | Sitter or Owner data is collected, or the user is redirected | Sitter or Owner data is collected, or the user is redirected | Matt Dulany |
| 7 | Users can edit data | User selects to edit their personal information | Users data is changed and displayed correctly | Users data is changed and displayed correctly | Matt Dulany |

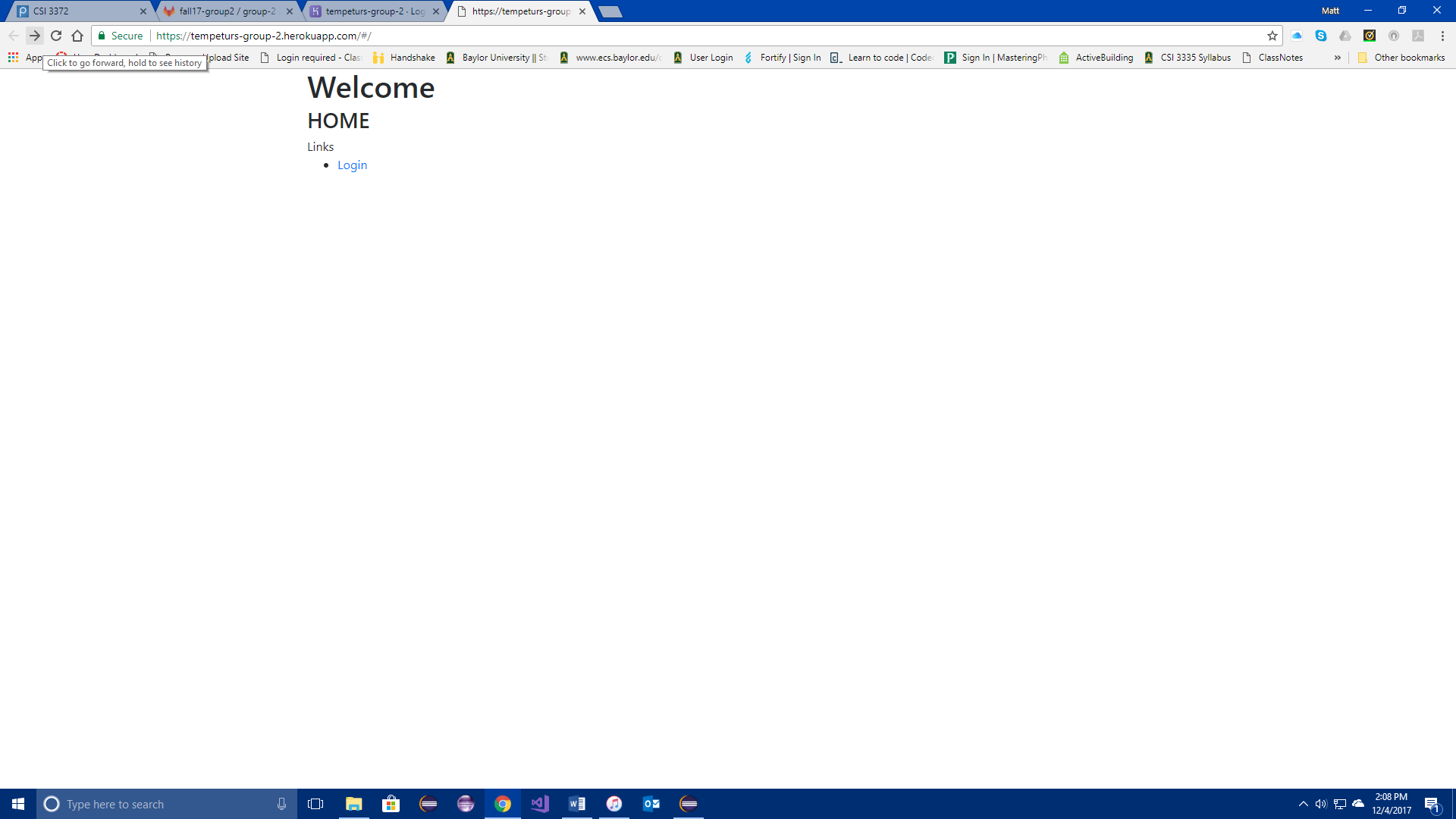
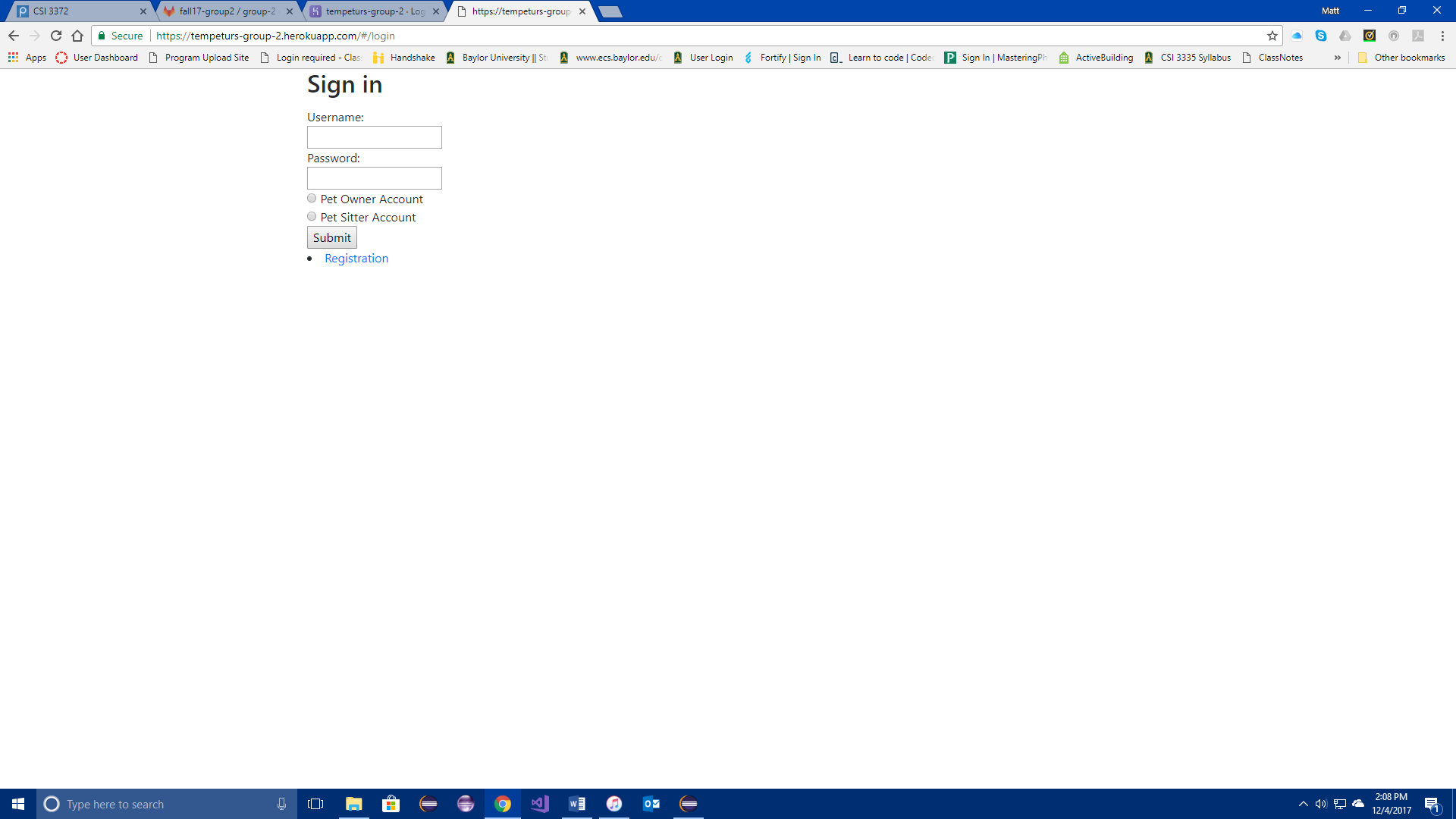
# Installation

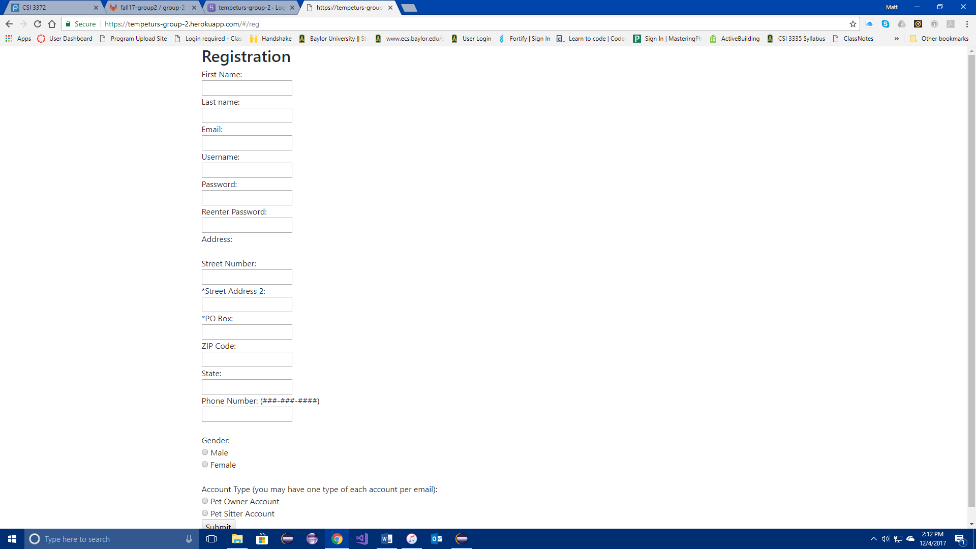
Since this project was a web application there is little to any installation required as a user. You must have access to an internet connection and have a working browser on your computer. If you have both of these, you will be able to access the website at the following url: <https://tempeturs-group-2.herokuapp.com/#/>.

If you are a future developer you will need access to the working gitlab repository and the Heroku app that are used for this code. Please contact [mattldulany@gmail.com](mailto:mattldulany@gmail.com) to request access to both.

# User Manual

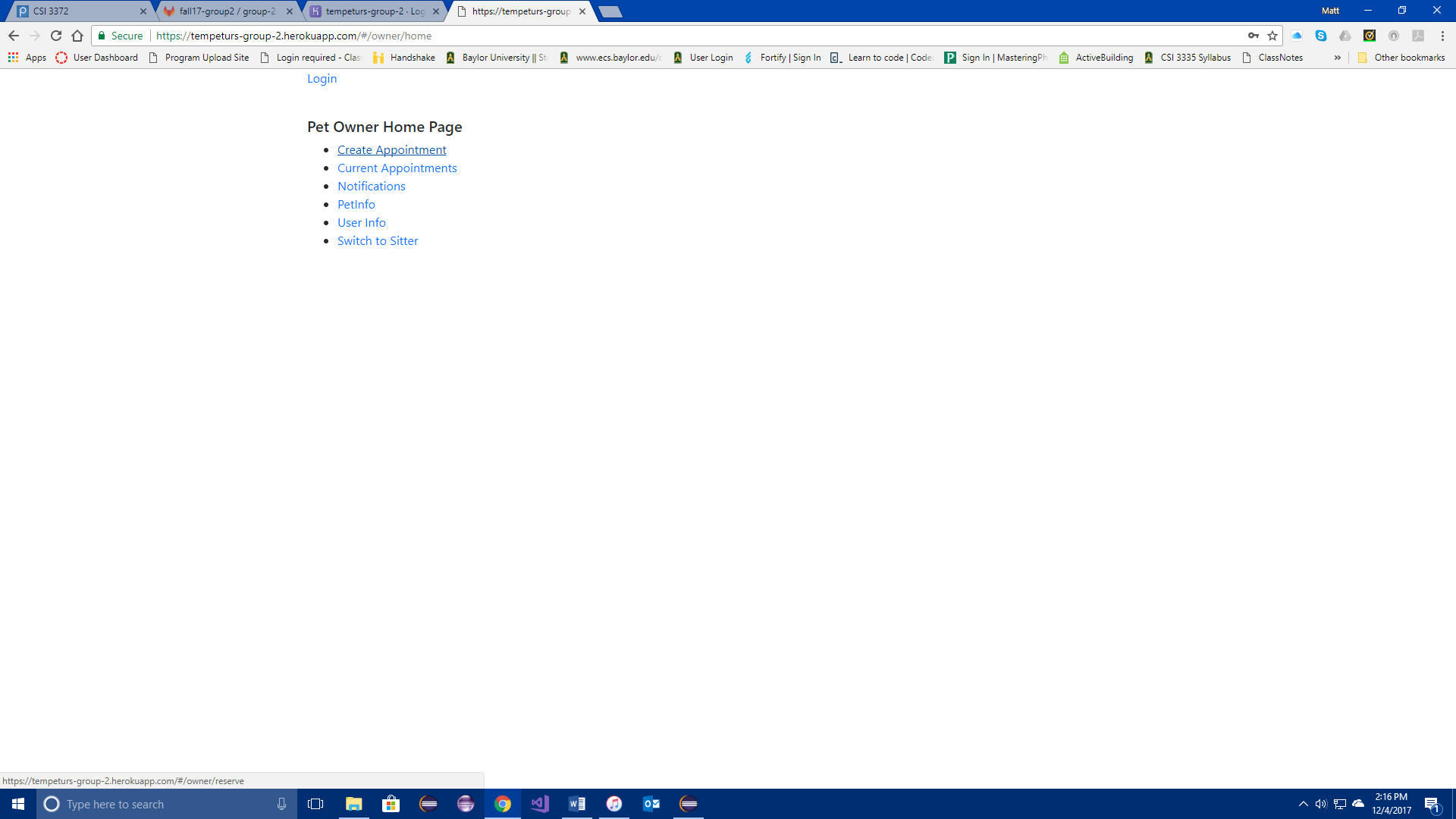
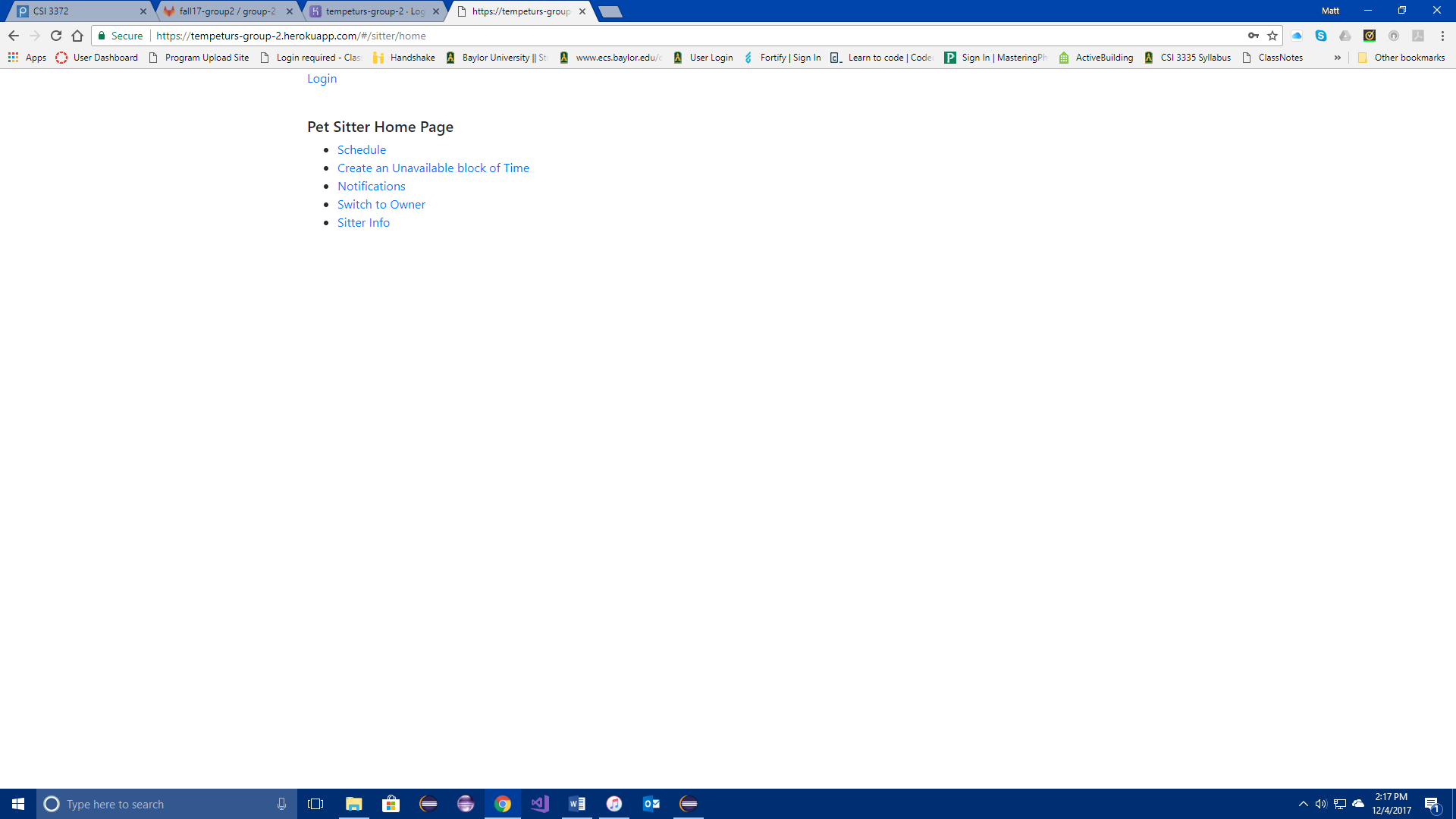
Once as you open this application you should click on the Login button which will take you to the login screen.

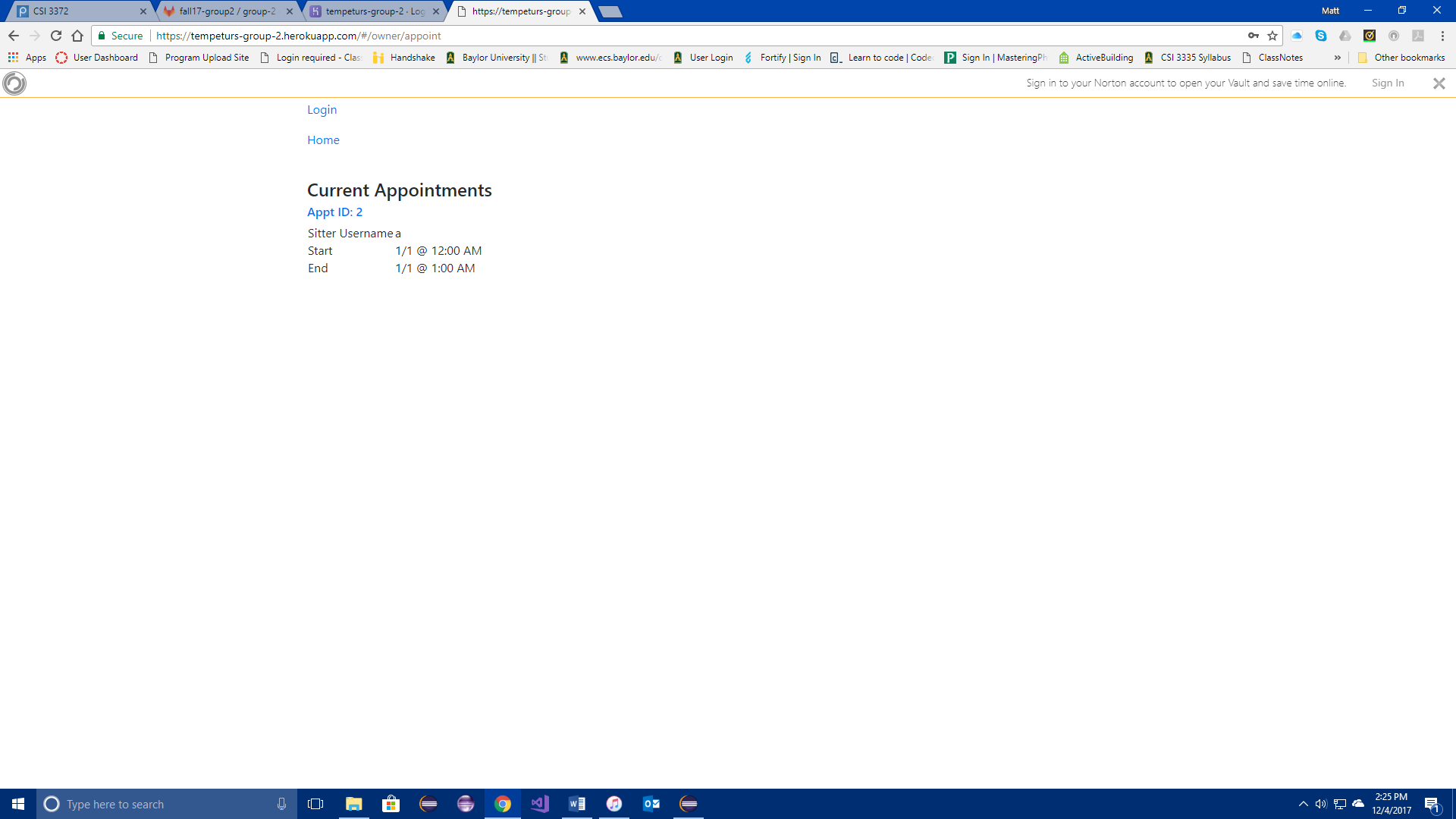
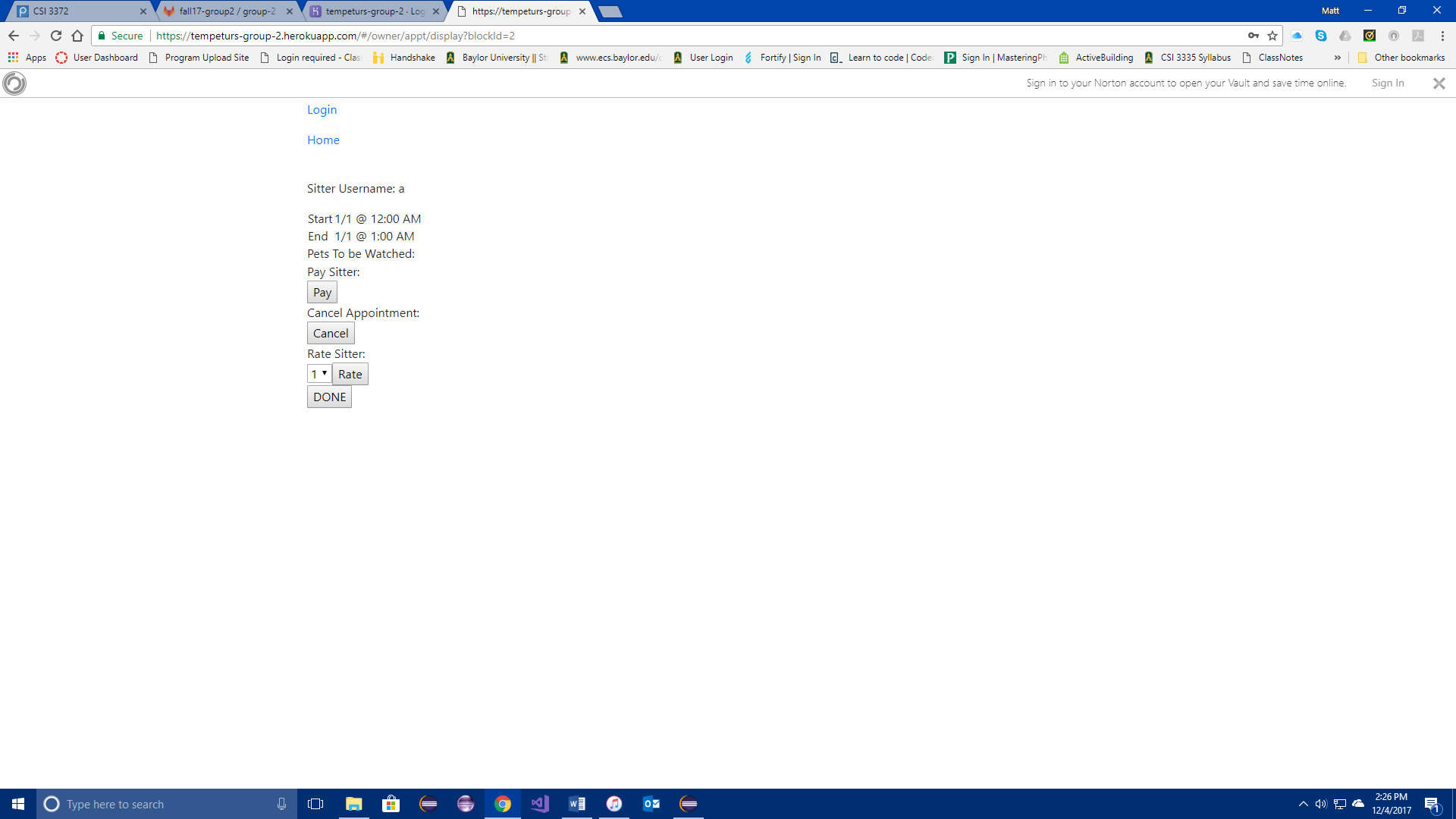




Once on the login screen if you already have an account enter your credentials. Make sure you choose the correct type of account to login with as, since you will be denied access if you do not. If you do not have an account, click Registration. Fill out all the information that is required and make sure you choose the correct type of account that you want to register as. If you would like to register as both this will be an option later.

After you click submit continue through the registration pages and fill out all required information. Be aware a \* denotes and optional field. When you are through you will be redirected back to the login page. Please provide you credentials to log in.

Once in you should see one of the two following home pages. If you would like to create an appointment or block of time, click on either link on the home pages. If you would like to see your notifications click on the notifications tab. Be aware that once as a notification has displayed once it will disappear. This feature is here to prevent a large mass of notifications from building up. If you would like to switch to becoming a sitter or an owner click the relevant links. It will ask you for your information if you are not already both types of users. If you are it will switch you over to the other home page. If you would like to look at your personal information, follow the link to the info page. On the page there is a link to edit that information if you desire. Pet information can be found on the Pet Info link as well as the ability to add pets.

On the schedule page and notifications page you can use the link of the appointment id to follow the appointment to an information page. From that information page you will find many useful tools on handling appointment operations. The tools differ from sitter to owner. Owners will have the ability to pay for the appointment as well as rate the sitter and cancel the appointment. Sitters will have the ability to edit a block of time, accept or deny and scheduled appointment or cancel an accepted appointment. Links back to your home page and the original login screen are at the top of most pages where exiting the page has no risks. Good luck and enjoy the application.

# Work Summary

Matt: Matt Dulany was the project manager for this project. He handled organizing all meetings and took the lead on these meetings and the scrum meetings. He as in charge of assigning tasks for each group member on weekly basis and oversaw each member’s work. He was in charge of compiling, creating and delivering all weekly deliverables. He created and formatted the final group project. He was in charge of approving all code submittals before they were merged into the master branch. He handled cleaning up the final master branch for final submission. He co-wrote the vision document. He was in charge of drawing out the SSDs, operation contracts, design overview for design patterns, SDs, updated Domain model, design class model and the Test plan. He was also in charge of testing all aspects of the project and writing up the test documents.

In code he was responsible for all backend functionality that was created and used in all frontend functionality. He designed all the endpoints and java classes, including the entire calendar package, and adding any classes that were not in the template project. He handled most issue with the authentication including designing the custom user details class and ensuring that the login endpoint worked correctly. He created the Elasticsearch database connection service that used a Rest Client to connect to the service. He designed all the indexes in the database. He wrote all unit tests and ensured that all functionality would connect with the react front end. On the frontend side he designed the home page, co-wrote the login page, wrote the registration pages for all users, for sitters and for owners. He wrote the sitter homepage. He wrote the schedule appointment page and owner notifications page. He Co-wrote the owner current appointments page. He wrote the pet info page the owner info page and all extending pages off those pages. He wrote the switch to sitter pages as well. He co-wrote the sitter notification page and made a few minor tweaks to the sitter schedule and single appointment view page. He wrote the sitter info page and the switch to owner page and all extending pages off those pages. He wrote the schedule and unavailable block of time page as well. He also designed the flow of the final presentation.

Dylan: Dylan Hines was responsible for working with Matt on authentication issues. He worked on redesigning the domain model. He was responsible for setting up the gitlab repository and maintaining all setting surrounding it. He was responsible for setting up CI for Heroku. He was in charge of any work that had to do with Alexa. While we ended up not using it, he worked on creating an Alexa account, an original program and began working on connecting Alexa to our project when it was removed as a requirement. He was then responsible for writing the majority of the owner current appointments page and wrote the single appointment view page. He was in charge of writing each week’s summary reports and writing the final overall meeting summary report. He was also worked with Matt to design out the back-end functionality and wrote a few of the method shells for the owner and sitter endpoints. He also worked on encrypting and decrypting passwords. While this was not a requirement and was removed for convenience purposes this did work correctly.

Collin: Collin Wiginton was responsible for writing several of the original brief and casual use cases as well as the writing the detailed used case for schedule sitter. He also drew the use case diagram. He was responsible for writing most of the login page. Collin also did a lot of work that we ended up having to scrap in the final implementation for time and complexity purposes. He originally spent many weeks working on and designing a calendar that we planned to use for the sitter schedule, as this grew more complicated we scrapped that idea and went to a week’s view idea that would use a table structure. Collin designed the majority of this and in large part it worked but due to time and a few small bugs we scrapped this at the very end to ensure that we had a working product for the final submission. However, Collin spent several weeks working on these pages.

Marcus: Marcus Hollingsworth was responsible for co-writing the vision document. He wrote co-wrote with Collin the brief and casual use cases. He designed the original domain model draft. He drew up a draft of the architectural model that was slightly tweaked for the final submission. Marcus worked on creating a large part of the structure for the sitter notifications page that he co-wrote with Matt.

# Meeting Summary

**September 4, 2017**

**Activities:**

* Project roles were designated
  + Dylan Hines: Project Librarian
  + Matt Dulany: Project Manager and Quality Assurance Engineer
  + Marcus Hollingsworth: Requirements Engineer
  + Collin Wiginton: Design Engineer

**Discussions:**

* We discussed our approach to the project, set up a recurring meeting time, and delegated general roles for each member for the next weeks.
* Decided on an overall look for the document for the vision document
* Created a plan for handling scrum meetings

**September 11, 2017**

**Activities:**

* A small Alexa skill was set up to greet the mentors
* Gitlab CI was set up and running
* The vision document was completed

**Discussions:**

* Began planning out how to tackle milestone 2, as well as the smaller project deliverables for Professor Song, as well as a plan for creating the elastic search demo.

**September 18, 2017**

**Activities:**

* Casual use cases were created for each of our project’s functions
* Basic elastic search indexes were set up for use with out project

**Discussions:**

* UI Design
  + What it will look like
  + Possible Designs
  + How it will communicate with the backend

**September 25, 2017**

**Activities:**

* Matt set up frontend registration pages
* Dylan worked on sitter and owner backend functions for the registration page
* Collin designed the UI for the sign in page
* Marcus wrote up advanced use cases for the project

**Discussions:**

* We discussed how we wanted the website navigation to function, and discussed the process for connecting to registration and home pages for the owner and sitter sides of the site.

**October 2, 2017**

**Activities:**

* Attempted to push to the frontend information to the backend using axios, but we had a few complications.

**Discussions:**

* How can we deal with not having data sent to the backend and still be making progress?
* How will we be storing the data from the calendar?
* How will we design the backed?

**October 9, 2017**

**Activities:**

* Matt added functionality to the owner frontend page
* Dylan thought up a good method for dealing with authentication
* We temporarily disabled CSRF so that we can continue forward sans authentication

**Discussions:**

* We were still having problems with authentication, which caused our group to fall behind on milestones. Therefore, we discussed strategies to get the parts of the project done that did not rely on authentication.

**October 16, 2017**

**Activities:**

* Collin finished the user interface for the owner and sitter calendars to see appointments
* Matt finished setting up the axios calls to connect the frontend and backend
* Dylan began research on how to set up Alexa connections with spring

**Discussions:**

* We discussed how we would calculate how close the owner and a given sitter were, and we decided on a system using their zipcodes.
* Discussed how we would make sure the system knows which user is currently using the website.

**October 23, 2017**

* We were not able to meet this week. However, each member had their own section of the project to work on.
  + Dylan was working on the owner and sitter frontend functionality
  + Matt was working on completing all frontend and backend endpoints and connections
  + Marcus and Collin were working on the sitter calendar

**October 30, 2017**

**Activities:**

* Owner and sitter backend functionality has been finished
* Matt added integration tests using JUnit

**Discussions:**

* We had been running into problems using the calendar that we created, so we discussed whether it was worth our time to continue pursuing the idea.
* Made more plans to get caught up on the milestones
  + Dylan and Matt would work on the backend
  + Collin and Marcus would work on the frontend

**November 6, 2017**

**Activities:**

* We got a version of out frontend calendar working

**Discussions:**

* We did not discuss anything specific. We mostly talked about what needs to be done before the project needed to be turned in.

**November 14, 2017**

* We were not able to meet this week. However, each member had their own section of the project to work on. Additionally, Matt was able to finally fix the authentication issues we were having.
  + Dylan was working on the appointments display for the owner and sitter
  + Matt was cleaning up the backend functions and design
  + Collin and Marcus were working on another iteration of the sitter calendar.

**November 27, 2017**

**Activities:**

* The owner and sitter pages now have the info pertaining to any upcoming appointments
* Login authentication now denies users who enter bad login info

**Discussions:**

* We did not discuss anything specific. We discussed what loose ends needed to be tied up before turning the project in on Thursday.

**November 30, 2017**

**Activities:**

* All the required functionality for the project has now been implemented

**Discussions:**

* We discussed what we wanted our presentation to look like, and what part of the project each member was going to talk about.