WASHINGTON STATE UNIVERSITY
VANCOUVER

CS 320 Course Project Final Report

for

The Pro-Crastinator

Prepared by

Group Name: We Came As Programmers

Spencer Ross	11686231	s.ross@wsu.edu
Sarah Mathes	11065825	sarah.robison-mathes@wsu.edu
Kyle Stennfeld	10722379	k.stennfeld@wsu.edu

Date: October 25, 2019

Contents

1 Introduction

1.1 Project Overview

The Pro-Crastinator is a student planner web application. It allows a users to add classes, coursework, and events to a profile. There is also a feature that will calculate grades and even calculate how a weighted grade will affect the total grade for the class.

Pro-Crastinator uses an account system to keep track of which user's data should be visible. The account also helps segregate each user's data in their own profile and not visible to others.

1.2 Definitions, Acronyms and Abbreviations

- *App = application*
- *API = application program interface*
- *CSS = Cascading Style Sheet*
- *DHTMLX – package and imports used to create scheduler*
- *DB = database*
- *HTML = Hyper Text Markup Language*
- *Iron-Router – package used to implement routing*
- *JS = JavaScript*
- *Meteor – javascript framework for easily creating web apps with universal code for cross-platform compatibiltiy*
- *MongoDB – package used to create backend database*
- *Routing – linking one app to another or having one part of an app redirect to another app*
- *UI = user interface*

1.3 References and Acknowledgments

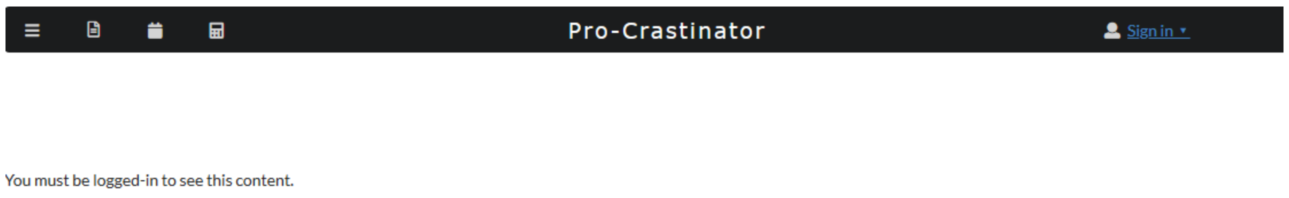
How To Create Meteor App with dhtmlxScheduler [1]
Setting up accounts [2]
Creating task list [3]
Routing and setting up Iron [4]
Importing Collections2 and using "Schema" [5]
Setting up Meteor and tutorials on using Meteor and JavaScript frameworks [6]

2 Design

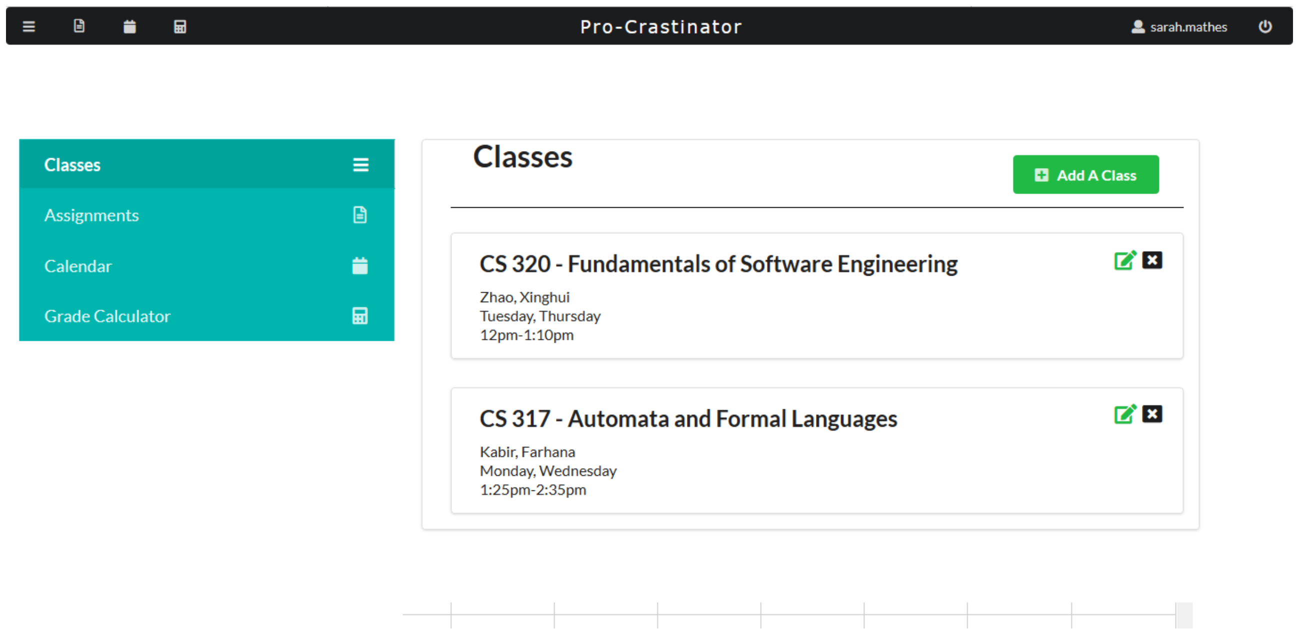
2.1 System Modeling

Our implementation strictly follows the design document from Milestone 2.

2.2 Interface Design



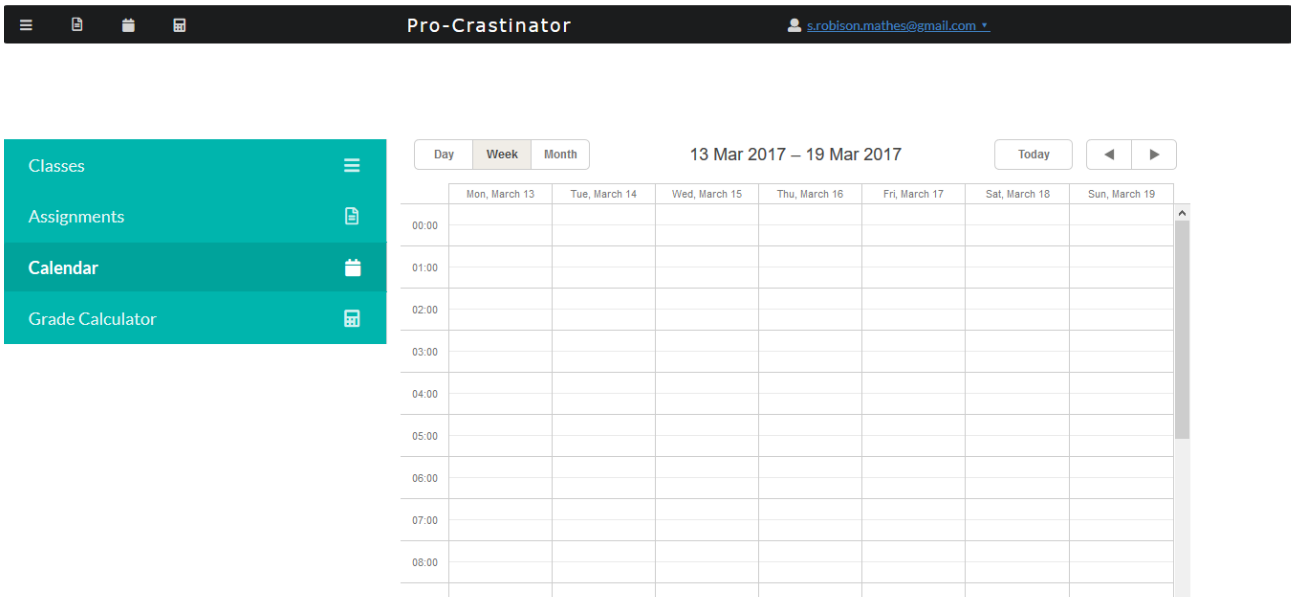
You must be logged-in to see this content.



Classes

CS 320 - Fundamentals of Software Engineering
Zhao, Xinghui
Tuesday, Thursday
12pm-1:10pm

CS 317 - Automata and Formal Languages
Kabir, Farhana
Monday, Wednesday
1:25pm-2:35pm


Pro-Crastinator
s.robison.mathes@gmail.com
Classes
Assignments
Calendar
Grade Calculator
Day
Week
Month
13 Mar 2017 – 19 Mar 2017
Today
Mon, March 13
Tue, March 14
Wed, March 15
Thu, March 16
Fri, March 17
Sat, March 18
Sun, March 19
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00

3 Implementation

3.1 Development Environment

We used IntelliJ IDE as the development environment for this project. We maintained version control by using a GitHub repository, and GitHub Desktop. The language used to set up the website and format the interface was HTML, as well as CSS. Javascript was used to implement IronRouter, MongoDB, and elements of SemanticUI. Other packages used, such as Schedulers and AccountsUI, were also written in Javascript. We wrote backend functions and data structures in Javascript. We used IronRouter to handle routing within the app, and AccountsUI to manage user accounts and log in functions.

3.2 Task Distribution

Tasks were evenly distributed between the three group members. During the preliminary phase, each group member was responsible for brainstorming software ideas, and we met in person to discuss and agree on one. We agreed on our development environment, and who would be responsible for setting up a GitHub repository.

In Milestone 1, the sections of the system specifications requirements document were assigned and deadlines were set so that the group could review the document before it waas turned in. Kyle wrote the sections that referred to the front end, Spencer took on the sections that covered technical and backend aspects, and Sarah focused on the sections that covered clients and development. In Milestone 2, each member was responsible for one UML diagram.

In the development phase, we assigned tasks to each group member. Sarah was responsible for the user interface and routing, Spencer was responsible for account functionality and the calendar interface, and Kyle was responsible for database structuring and form events. All team members were responsible for their own research, tutorial work, and weekly GitHub commits.

3.3 Challenges

The greatest challenge to this project was that none of the group members had experience with web application development. This includes server-client events, web frameworks, packages, events, forms, and databases. All group members had experience and felt confident writing Java and Javascript code, and Sarah had experience with HTML and CSS. Kyle and Sarah both worked through several tutorials and online resources to try and familiarize themselves with Meteor project structures. Efforts were often stymied by tutorials using outdated or unsupported packages. There were often compatibility issues between different packages. We were also limited by the capabilities of the packages that we used. For example, Spencer worked to make the calendar interface from the Scheduler package remember created events, but has no experience with manipulating or customizing other Javascript packages. We also had some challenges with learning how to use GitHub effectively, because none of us had experience with using it collaboratively.

4 Testing

4.1 Testing Plan

If we had reached the testing phase of this project, we would have implemented Mocha and Chai to run behavioral tests using IntelliJ to write the code and Chrome to execute the tests. This would have been an efficient way to test if Meteor and Semantic UI were running to our expectations.

4.2 Tests for Functional Requirements

Since we did not reach the testing phase of our project, we only had a list of functions and what they would test. With that being said, we do not have any test results for the following functions.

· CalendarInput(): The function tests if the user's input data for the calendar was saved under the correct date and time.

· ParseUserInput(): Function parses through the user's input to verify it's correct for each field in its designated form.

· GradeCalculation(): Function Verifies the user's grade data was calculated into their final grade correctly using the correct grade weight.

4.3 Tests for Non-functional Requirements

The following are functions that would have been implemented to test the web applications non-functional requirements:

LoginVerification(): Verify the user's login is in the data base. Return user unknow, if user's credentials do not exist.

MongoVerify(): Verify the user's information is saved in the Mongo database and can still be accessed every time they log.

PerformanceTest(): Function that gives the running time of the grading functions. How long it takes to calculate final and "what if grades"

Finally, test usability by testing semantic UI source code. Items to verify:

· Pages are routed correctly

· There are no broken links

· Page layout is the same in different web bowsers

· Page layout stays constant when loading other pages

· Page layout flows smoothly

4.4 Hardware and Software Requirements

Required software:

· IntelliJ/Sublime Text
· Chrome
· Meteor
· MongoDB
· Blaze
· Semantic UI

Required Hardware:
· Modern Laptop/Desktop running IOS, Windows, or Linux

5 Analysis

Sarah did:

- the majority of getting the project set up
- Front end: HTML, CSS, Mockup, Templates
- Leadership, Organizing the group
- UML, Structural diagram

Kyle did:

- Behavioral diagram
- Assignment list app
- MongoDB database
- Blaze server

Spencer did:

- Set up Git repository for project
- Activity diagram
- Scheduler app
- Account package for website
- Fused separate meteor projects into a unified project/app

We spent about 10 hrs/week per person working on this in between our regular assignments and studying for other classes. Easily most of our time was spent on research and learning from tutorials how to get started and how any of the concepts used for this project work. We started this project with no understanding of web frameworks, or server-client events. Most of our endeavors were in researching and trying to understand how a Meteor JS project should be structured.

We think that if we had a longer period of time and no other assignments and major duties outside of this project, we could have finished the whole application with a fairly robust feature set. At this point, we have a much better understanding of backend operations, databses, and forms. Our effort toward this project was mainly held back by other responsibilties we had to homework, other programs, studying and our jobs.

6 Conclusion

We learned that teams can make a large task much simpler by breaking the workload into smaller, more manageable tasks. One thing we struggled with was judging how to properly divide the project. Because we all have never worked on a web application before, we got stuck doing a lot of research and learning how to get started.

One of the most challenging things we learned was how various pieces all connect together to make a unified application. Some things we now know how to do because of this project are:

1. How to create a basic Meteor app
2. Find and install packages to implement functionality
3. Route a webpage to multiple web apps for a dynamic website
4. How servers and clients are used and which parts of a web app they handle
5. How to set up a server to save data and changes made in an app
6. How apply an account system to give each user an individual experience

Appendix A - Group Log

Our team used a Slack workspace to communicate regularly. We used a Slack channel to plan how to work on this project, delegate the workload, and schedule meet-ups for in person collaboration. We met as a group at least once a week in the computer labs and discussed our progress or struggles with the tasks we were working on. Toward the end of this development phase, we would sit in the same room and work together to get the multiple web apps into a single cohesive website. Each group member was responsible for their own Git commits and updates. Using the Slack chatroom was very effective, and all group members responded in a timely manner to questions or suggestions. None of our group members had used GitHub for group collaboration before. As a result, some of our file organization could have been improved. Despite the learning curve, using GitHub for version control was also effective.

Sources*:*

[1] How To Create Meteor App with dhtmlx Scheduler, DHTMLX*,* March 28, 2019. Accessed on: December 3, 2019. [Online]. https://dhtmlx.com/blog/create-meteor-app-dhtmlxscheduler/
[2] Meteor Tips, Chapter 10: Accounts, Meteor Tips. Accessed on: December 3, 2019. [Online]. http://meteortips.com/first-meteor-tutorial/accounts/
[3] Meteor Tips, Chapter 4: Databases, Meteor Tips. Accessed on: December 3, 2019. [Online]. http://meteortips.com/first-meteor-tutorial/databases-part-1/
[4] Iron Router Guide, Iron Router. Accessed on: December 5, 2019. [Online]. https*://iron-meteor.github.io/iron-router/#using-javascript*
[5] Turnbull, D, *6 Must-Use Meteor Packages for (Almost) Every Project,* Sitepoint, January 29, 2015. Accessed on: December 5, 2019. [Online]. *https://www.sitepoint.com/6-must-use-meteor-packages-almost-every-project/*
[6] "Meteor JS," Accessed on: November 21, 2019. [Online]. Available: https://www.meteor.com