Cloud Computing Final Project: Classy

Benson Tran

April 27, 2015

1 Introduction

In this report I will provide a retrospective on my design choices and their impact on completing features for Classy, further providing insight on where such design choices made implementation feasible or a complete hassle. In terms of front end development, I will dive into how the Angular framework provides a scalable architecture for a responsive and dynamic, one-page web application. In terms of back end development, I will discuss the pros and cons of how the Django REST framework may or may not have provided the most feasible outlet for developing viewsets and API endpoints. Moreover, I will dive into the functionality of each of the features at a high level as well as the design and implementation that went into each.

2 Web Stack

2.1 Front End Design Choices - Angular.js

I chose Angular because of its support from Google and its lauded functionality in producing dynamic and modular .html files. Moreover, Angular has provided me with flexibility in designing a one-page application. Angular lends itself to an inherently extendable framework of controllers, directives, and services, in which the latter two can be injected into the former. This provides quick and agile front-end deployment of features.

2.1.1 Angular Architecture: Directives, Controllers and Services

I was able to create front end features such as creating assignments and the classroom bar by simply creating more Angular modules. Each Angular module can contain services, controllers, and directives. The service serves as the factory instantiating the internal API link. The controller serves as the logic that makes the application interactive to the user. Lastly, the directive is not only an unique aspect of Angular, but also has proven to be an invaluable feature used in this developing Classy.

A directive is essentially an HTML template embedded with a bit of Angular magic. In more concrete terms, directives serve as 'front end modules', each with a specific functionality that can be embedded in the DOM. I can access directives via HTML tags. This drastically reduces the unsavvy Javascript overhead that follows with elaborate front end design and logic-heavy, one page applications.

Directives have spurred a wide variety of modules in the open-source community. This makes it possible to deploy what you want when you want. Want to have your client-sided code offload your URL routing? There's a directive for that. Want to effortlessly pass data to and from popups? There's a directive for that.

2.1.2 Modularity and Dependency Injection

What makes a lot of this Angular 'magic' a possibility? Modularity via dependency injection.

Dependency injection is a pivotal and pervasive design pattern used in Angular applications - it provides functionality to components by providing a way for them to get a hold of their dependencies.

The current architecture of this front end application is only possible because of dependency injection. The highest level module, *classy*, introduces accounts, assignments, signups, etc. via dependency injection. Each of these modules introduces services, directives, and controllers via dependency injection as well.

In addition to providing a modular framework, dependency injection provides functionality to components. Directives and Angular objects are introduced through dependency injection. This is particularly pivotal if I want to maintain a one-page application. For example, I have a form for creating assignments and a form for setting up meetings. Each of these pop ups is associated with a controller. To propagate data to the main HTML page I need to broadcast the data from the controller of these forms to the controller associated with the main HTML page. Such communication is possible via injecting *$scope* and *$rootScope* Angular objects.

Ironically, despite Angular's reliance on modules, I've found it hard to maintain a sense of encapsulation within the project. Classy's main HTML page, *account.html*, contains data from a variety of service factories. The main page controller *account.controller.js* aggregates data from these services. The ability to inject a plethora of modules into a controller makes it hard for not to use account controller for the majority of all functionality.

2.1.3 Elegant and Responsive UI with Angular.js & Django Template Inheritance

Although Django is known as a back-end web framework, it provides powerful features for front-end web developers. Django's template inheritance was extremely useful in bookkeeping. Javascript and stylesheet files. *javascript.html* and *stylesheets.css* were included in *index.html*, from which all other HTML files from the project inherit. This reduces clutter of providing <*script*> tags independently to each HTML file and provides access to stylesheets and Javascript files to all HTML files inheriting from the parent template, propagating styling changes to the entire application immediately after a file addition.

Material Design is used for Classy's UI. Material Design can be introduced into a project by dependency injecting *ng-material* into the root module of any project. Moreover, by injecting *ng-material*, I could also inject *ng-animate*, which is dependent on *ng-material.*

ng-animate provides developers with animation-hooks to common Angular directives such as *ng-show*, *ng-hide*, and *ng-repeat.* Take for example *ng-show* and *ng-hide.* Both of these directives hide and show HTML tags. In the event of hiding or showing, *ng-animate* appends classes *ng-hide-add* or *ng-hide-remove*, respectively. CSS animations can then be associated to these classes. In Classy, I use an open source CSS file, *Animate.css*, to provide fading animations for common ng directives.

Django's template inheritance expedites the development and styling of HTML files. Angular's dependency injection also makes it possible to inject animations. The combination of both made it possible to perform a UI overhaul and implement a landing page in a relatively timely manner.

2.2 Back End Design Choices - Django and Django REST Framework

I chose Django because not only does it use Python, but it also abides to a model-view-controller (MVC) framework.

2.2.1 High-level System Overview

The back end of this project does utilize Django's MVC architecture. The view part of this project consists of API endpoints, which is the term used by the Django REST Framework. Those endpoints are essentially functions that handle and respond to corresponding HTTP requests issued by the front-end. The role of the controller in this system is assumed by the serializers. They are modules that are in charge of translating database tables/objects into JSON. The model part of the system defines the database schema for each Django class. This is the standard use of the Django model class. The data and control flow of the system can be summarized by Figure 1.

From the diagram above, one can notice that a router is used in the back end. A router allows the front-end to make HTTP requests that are not hard-coded to call a specific back-end API endpoint

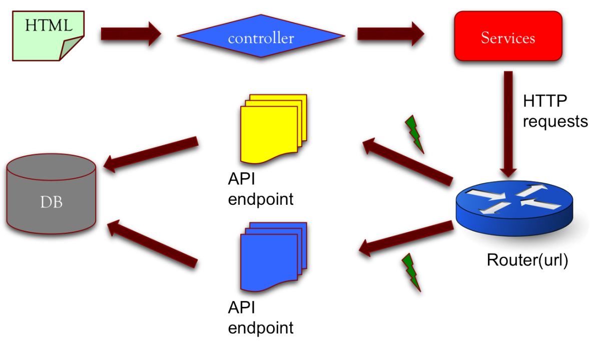


Figure 1: Data and Control Flow of Classy System

3 Classy's Features

Classy aims to be a social education management platform used to engage students, teachers and parents. After logging in as either a teacher, student, or parent, one is introduced to a calendar interface. The purpose of the calendar is to display assignments due dates and to display meetings.

A teacher can add students to his class and assign assignments for each class. These assignments are propagated to both the teacher and students' calendars. Moreover, the students are notified of their open assignments. The teacher can keep track of the status of the assignment of each student. Moreover, each assignment is associated with a hashtag and a video resource. The students can interact with each assignment by posting questions and asking comments via tweeting with the associated hashtag.

Teachers can also create meetings with parents and other teachers. A teacher can provide a minimum and maximum time along with the number of available slots available for signup. The appointees then receive a notification to signup.

Unfortunately, I was unable to implement a grading system for the teacher and a feedback system from the students. There simply was too much back end implementation needed for this 'back-and-forth' communication between the students and teachers. In other words, a teacher needs to create an assignment. I student then needs to notify the teacher when the assignment is completed. After the assignment is completed the teacher can associate a grade to this assignment. Then both the parents and students need to be notified of this grade. If this convoluted back end were implemented, I had plans of displaying the relationship of grades with the feedback of each student on a chart.

3.1 Twitter API Calling

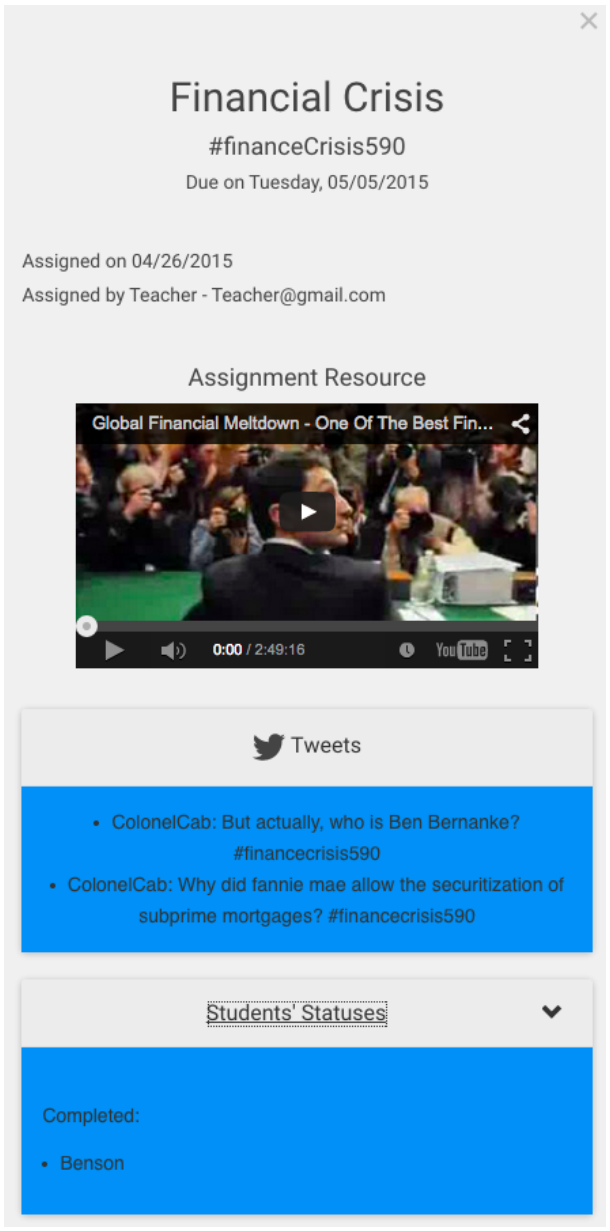
3.1.1 High-Level Description

An example assignment is shown in Figure 2. The hashtag associated with the assignment is also displayed near the top of the pop-up. Any tweet with that hashtag will show up in the tweets accordion. My hope is that tweets will serve as an outlet for discussion and questions. Moreover, rather than using a comment thread, tweeting doesn't require the student to log in into to Classy to ask a question, providing for more accessibility (esp. for younger students).

3.1.2 Implementation

I used the Tweepy, a Python library for making RESTful API calls to Twitter. Obviously, this means the tweets are queried by a Django REST viewset, which is initiated by a call from the front end to the relevant Django REST API endpoint. Unfortunately, using Tweepy provides a lot of superfluous overhead.

I initially tried making client-sided Twitter API calls, unfortunately the attempts were all in vain. I initially tried using a open-source directives; however, these directives were contingent on Twitter's 1.0 API as opposed to the current 1.1 API. Moreover, I using Node.js Twitter libraries. Node.js is a server-sided Javascript framework; therefore, it contains classes and functions that aren't functional in the browser (e.g., .requires(), export). There are many ways to bypass this, however. One can use a tool called Browserify. Browserify packages all of the Node modules into a client-sided Javascript file. It's


Financial Crisis

#financeCrisis590

Due on Tuesday, 05/05/2015

Assigned on 04/26/2015

Assigned by Teacher - Teacher@gmail.com

Assignment Resource

Tweets

- ColonelCab: But actually, who is Ben Bernanke? #financecrisis590
- ColonelCab: Why did fannie mae allow the securitization of subprime mortgages? #financecrisis590

Students' Statuses

Completed:

- Benson

Figure 2: Example Assignment Pop-up

able to do this because it recursively searches for all required Node dependencies and packages them into one file. Unfortunately, this method didn't work for me.

Preferably, I would like to use a streaming API to receive real-time Twitter updates rather than performing multiple GET requests (one to the back end of Classy and one to Twitter's RESTful API).

3.2 Adding Students and Creating Classes

3.2.1 High-Level Description

A pull out sidebar (Figure 3) is used as an interface to view and add both students and classes.

3.2.2 Implementation

The back end implementation of classes is via the model Groups. Groups has a one-to-one relationship with the owner via using a Foreign Key. Moreover, a many-to-many relationship is established with the members of the groups.

3.3 Creating Assignments and Meetings

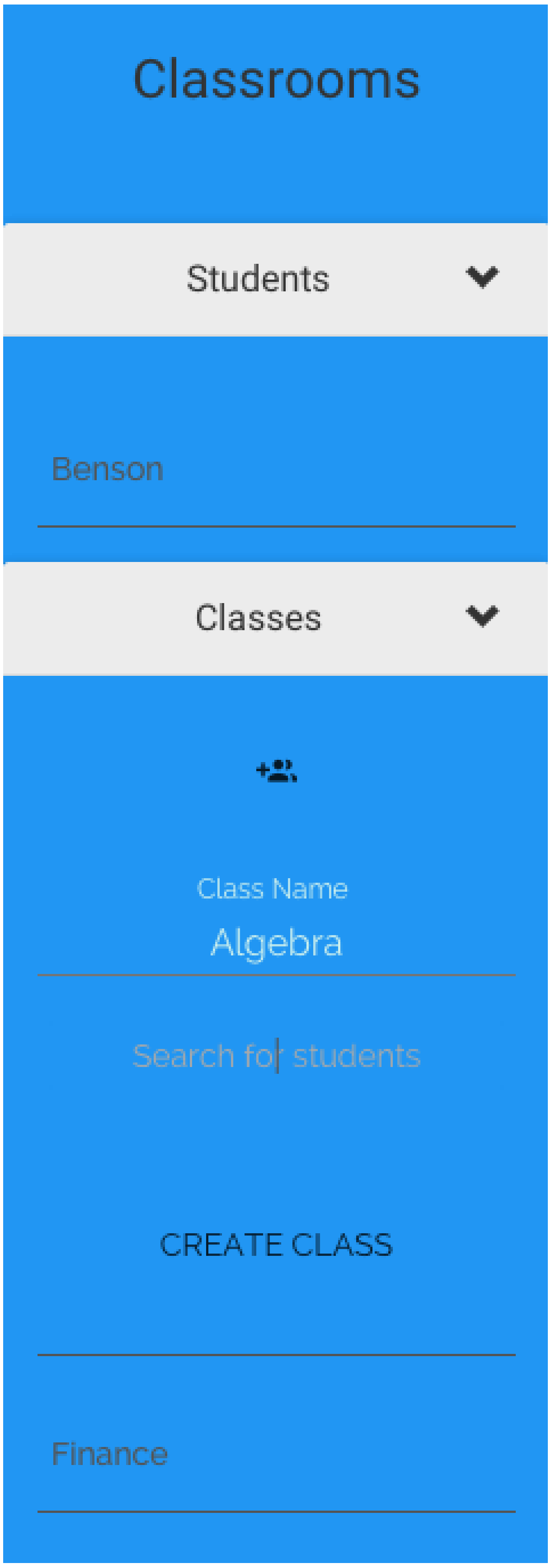
3.3.1 High-Level Description

The nexus of Classy is assignment and meeting creation. Pop-up forms are used for both assignment and meetings as shown in Figure 4. In addition to the assignment name, a due date, a video URL, and a hashtag can be associated with the assignment. Moreover, The assignment can be set to repeat weekly, monthly, or daily (e.g., useful for problem sets). Lastly notifications can be sent to the students.

The meeting form is pretty similar and is shown in Figure 5. In addition to a text field for the name and location, sliders are used to provide the minimum and maximum time slots. Teachers can set the number of available slots for signup per person and create multiple blocks for each signup.

3.3.2 Implementation

The implementation for meetings and assignments is very front end heavy. Data is binded from the form to the corresponding assignment or meeting controller and is then passed back to be serialized to a corresponding model. Similar to classes, each assignment and each meeting has a many-to-many field mapping to groups. This allows sharing of the assignments and meetings.


Classrooms

Students

Benson

Classes

Class Name

Algebra

Search for students

CREATE CLASS

Finance

Figure 3: Classroom

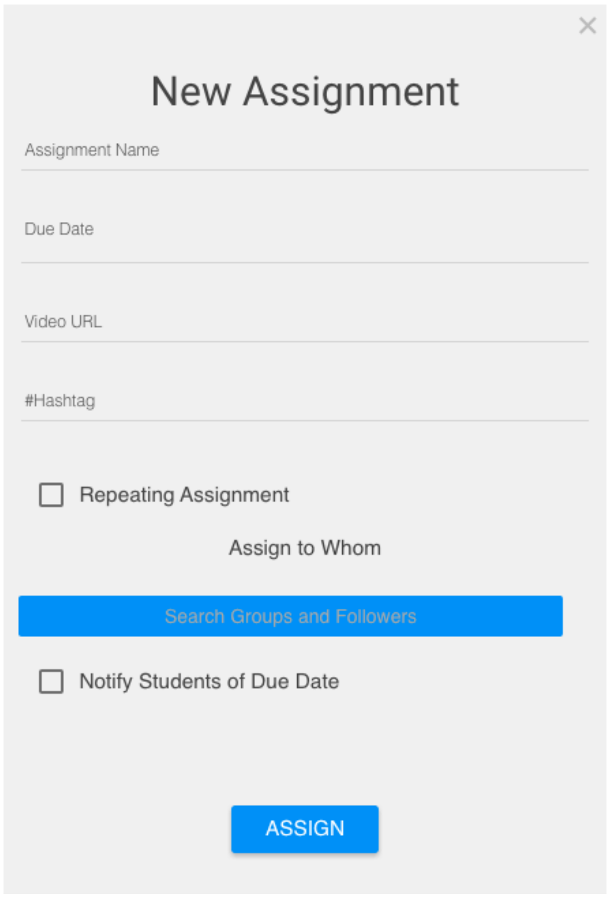
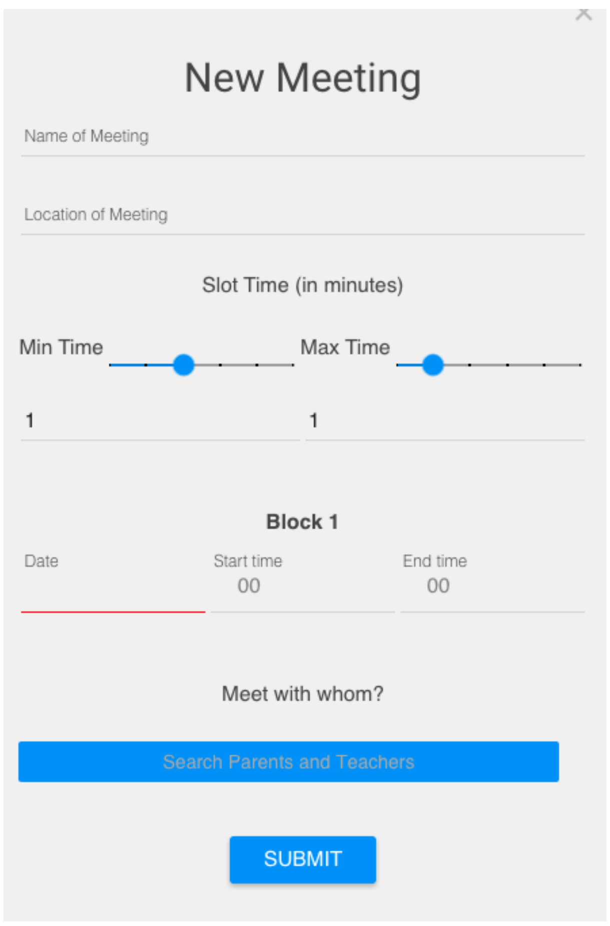


Figure 4: Assignment Form



New Meeting

Name of Meeting

Location of Meeting

Slot Time (in minutes)

Min Time 1

Max Time 1

Block 1

Date

Start time 00

End time 00

Meet with whom?

Figure 5: Meeting Form