ReCal Course Selection:

A Course Planning Tool for Princeton Students

Dyland Xue

Advisor: Dr. Robert Dondero

1. Introduction

Coursework management and course selection have always been critical to college education, yet, both are demanding tasks. Princeton students are required to take 4 or more courses per semester, for each of which they have to monitor announcements of assignments, office hours, exam times and locations, etc. Due to a lack of standardization, these pieces of information come in different formats dependent on the professor's preference. Some syllabi are Microsoft Word documents; some are posted online; some are only available as handouts. Other information such as office hour changes and assignment deadlines are announced through different media as well.

The difficulty of gathering such information was the motivation for developing *ReCal Dashboard*. *ReCal*, short for rethinking calendar, is an academic calendar, but with its information crowd-sourced from students. By delegating the responsibility of gathering course-related information, *ReCal Dashboard* enables users to manage their coursework for all courses in one-stop.

To complete *ReCal*, however, we realized that we must also tackle the other critical problem—course selection. Each semester, students face the task of choosing a few courses from over 1000 available listings, while taking into consideration of graduation requirements, course reviews, and avoiding schedule conflicts. Similar to coursework management, the difficulty of this problem stems from the de-centralization of course information.

To help students with course selection, and to provide *ReCal Dashboard* users an easy way to initialize their dashboard, we have developed *ReCal Course Selection*. In the next section of this

report, we describe the functionalities of *ReCal CS* by doing a typical scenario walkthrough, and establish that the course selection tool

- helps students visualize their schedules;
- enables students to explore the possibilities of different combinations of courses;
- helps students make informed choices;
- and provides high availability.

Section 3 briefly describes our design on the back-end, while section 4 describes the front-end design in depth as *ReCal CS* is a front-end heavy project. Section 5 illustrate how testing and evaluation shaped the development of *ReCal CS*. Section 6 compares *ReCal CS* with established course selection solutions, notably *ICE* and *TigerHub*, and finally section 7 talks about the future of *ReCal CS* and potential launch plans.

2. Functionality

ReCal Course Selection is a single page web application. In terms of functionality, we focused on making information available and the interface pleasant and intuitive. To demonstrate how the application works, we will do a quick walkthrough of a typical user experience.

2.1. A Scenario Walkthrough

Suppose Bob is a Princeton junior majoring in Computer Science, and in December, he is looking for new courses for the upcoming semester. As a COS major, he wants to take 2-3 departmentals, and 1-2 distribution courses. He has a few courses in mind but would like to explore if there are better options. With this hypothetical scenario in mind, let's see how Bob would use *ReCal CS* to achieve his goals.

Login with Princeton Credentials

Although *ReCal CS* only provides data that is publicly available, we still require the user to login via CAS (Central Authentication System) as this is a software product targeting Princeton students. Bob navigates to the standard CAS login page, and after putting in his Princeton netID credentials,

he is brought to the course selection tool.

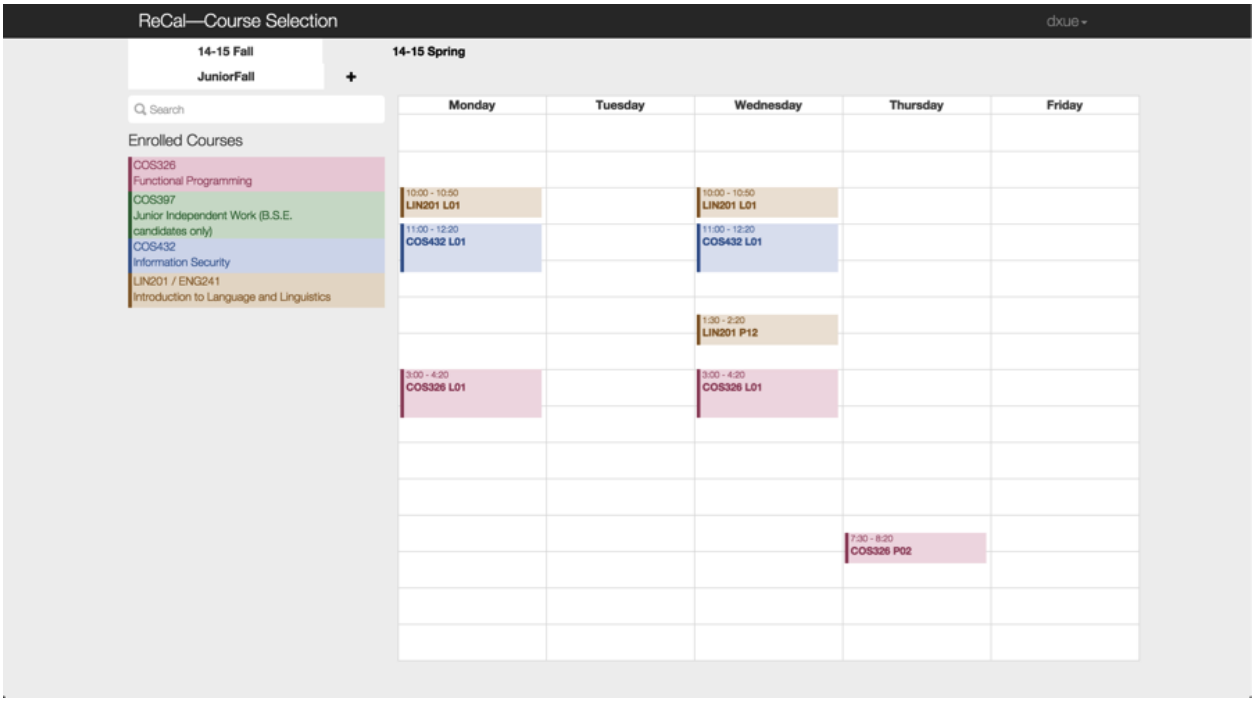

Figure 1: Main interface of *ReCal Course Selection*

Select Semester and Add Schedule

At the top of the page, we see two tabs for semesters—"14-15 Fall" and "14-15 Spring"—as these are the only ones available for Bob. In the current beta version, we automatically populate users' semester tabs to contain only the current semester and the next semester. In the future, if Bob has data from past semesters, he may have more semester tabs available.

Below the semesters, we see a tab labeled "JuniorFall" highlighted in a light background. Each tab at this level corresponds to a schedule. Schedules consist of enrolled courses, enrolled sections, and what colors correspond to those courses. Note that each user may create multiple independent schedules for the same semester; that is, the schedules may have different color schemes with different course enrollments. Switching between schedules is as simple as a click on a schedule tab.

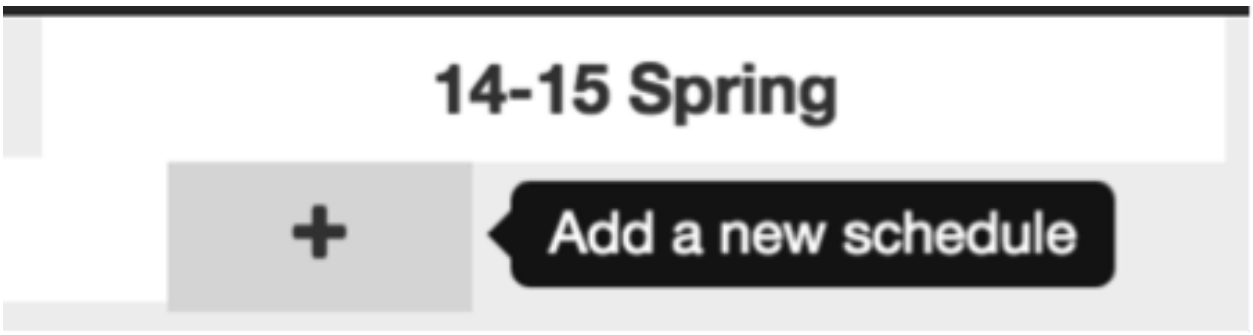

Figure 2: Add schedule button located under semester tabs, shows a tooltip on hover.

We see an add button next to the schedule tab. The button is only visible when the selected semester is active——meaning it is either the current or the next semester. Bob decides to add a new schedule for next spring by clicking the semester tab "14-15 Spring" and then the add button.

Figure 3: Modal for creating a new schedule.

A modal shows up, prompting Bob for a schedule name. Bob decides to name the new schedule "JuniorSpring" and clicks create. A new tab is generated, as well as an empty calendar and a list of recommended courses.

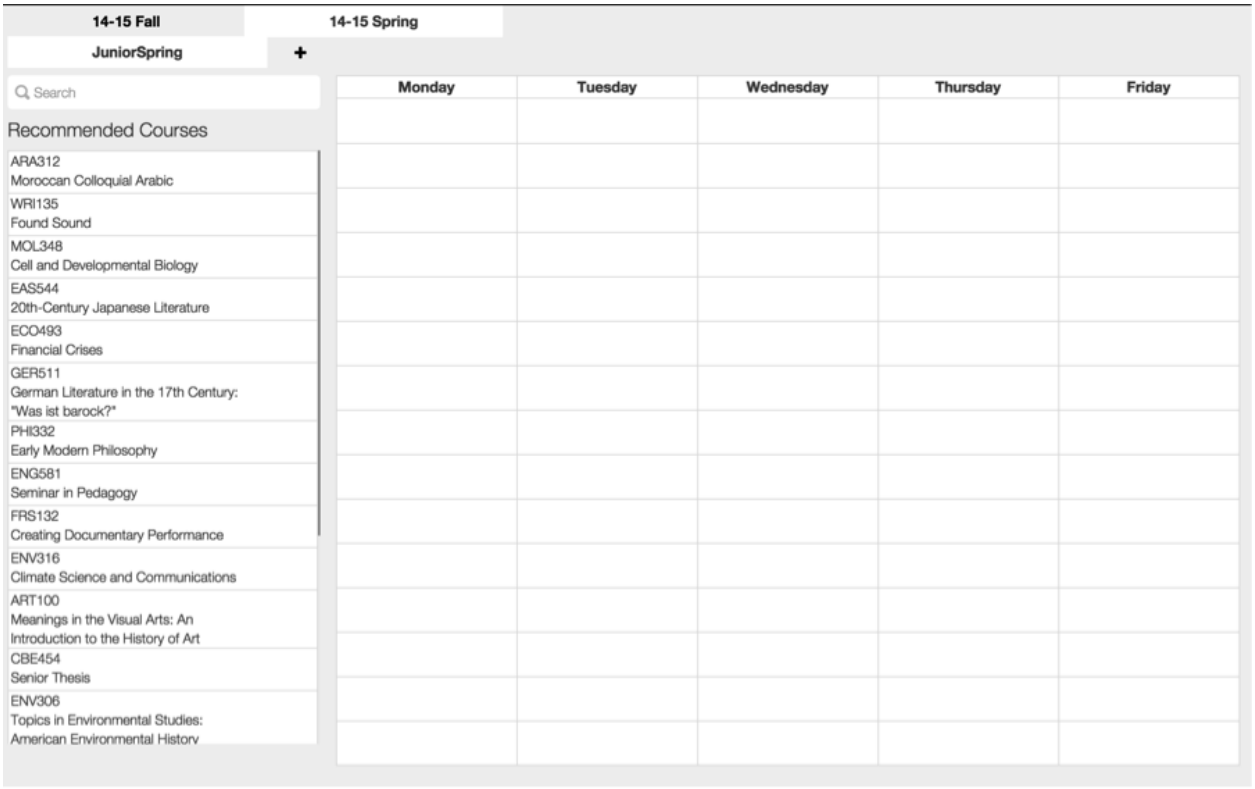

Figure 4: The fresh view after adding a new schedule.

Course Search and Course Enrollment

A list of recommended courses is auto-generated in descending order of the course ratings pulled from easyPCE[11], a student-built website that aggregates course reviews and ratings from Princeton Course Evaluation. This list only appears when there are no enrolled courses in this schedule. Bob first wants to add a few departmentals, so he goes to the search bar above recommended courses and types in "cos" for computer science courses. A list of search results show up in the panel, sorted by their course listings. The calendar is updated as the cursor hovers a course panel—previewing the sections for a particular course as shown in *figure 5*.

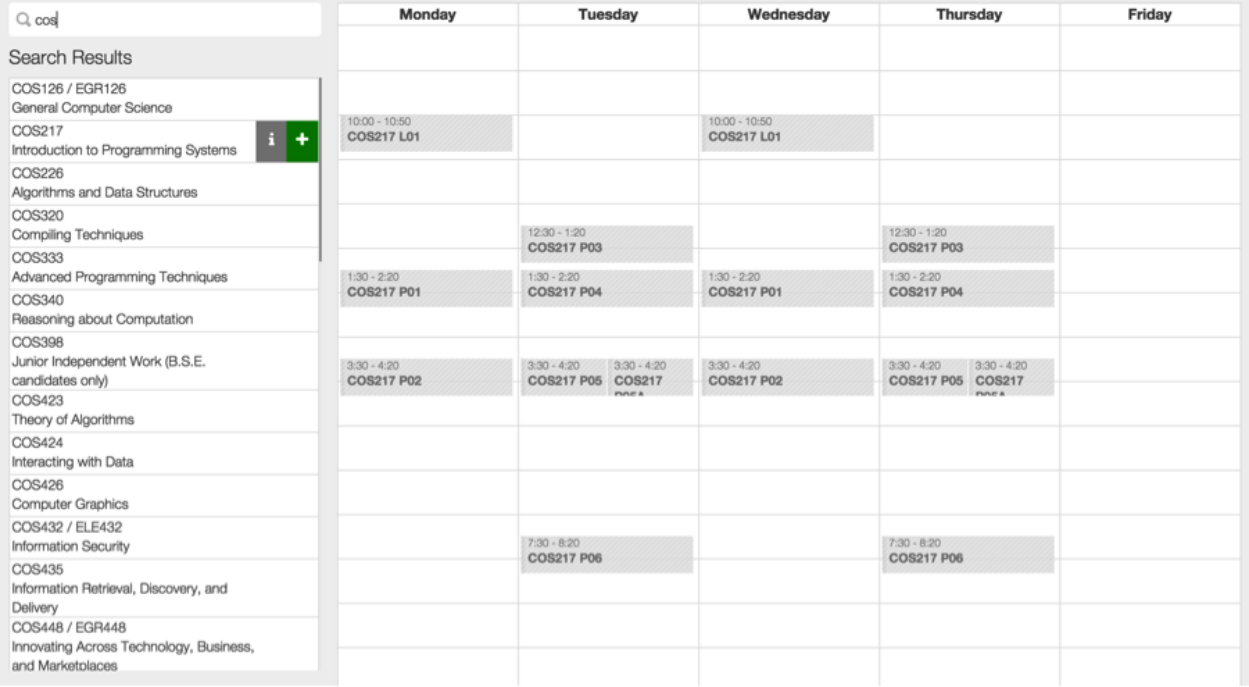

Figure 5: Previewing COS217.

Bob scrolls down the search results, and selects COS423 *Theory of Algorithms* by clicking the green add button to the right of the course panel. The course item is then added to the group "Enrolled Courses". Similarly, Bob added 2 more courses: COS461 and EGR392.

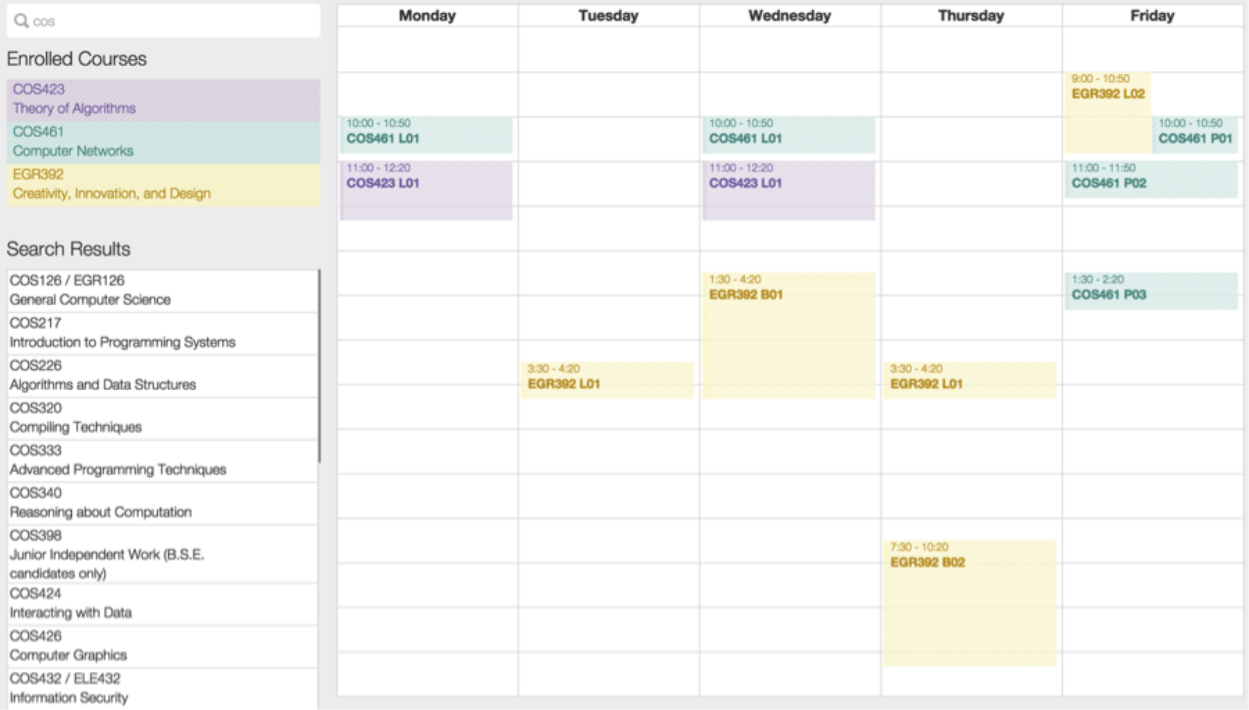

Figure 6: After enrolling in COS461, EGR392, and COS423.

Section Enrollment

Bob notices that some sections are conflicting—EGR392 L02 and COS461 P01. Fortunately, there are other sections of the same type for each course available at different times. To enroll in sections, Bob clicks on the calendar items of the sections he wants to enroll in: "COS461 L01", "COS423 L01", "EGR392 L01", "EGR392 B01", "COS461 P02". A solid vertical bar appears on the left of the corresponding calendar item, and once all section types of a course are confirmed——lectures and labs in the case of EGR392——the course item also gains a solid border on the left. These style changes, as shown in *Figure 7*, confirm the enrollments.

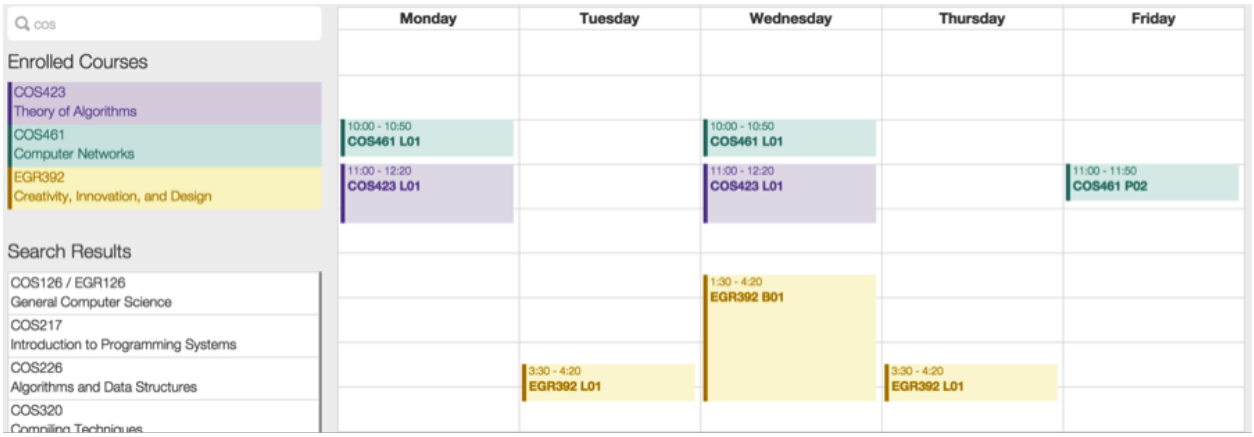

Figure 7: After enrolling in sections COS461 L01, COS423 L01, EGR392 L01, EGR392 B01, COS461 P02.

Course Information

Bob is now looking for a 4th course in the Philosophy department. By searching for courses with the query "phi 3", he notices that PHI301 and PHI304 both fit his schedule. To compare them, he hovers the course item and clicks on the "i" tag, which means more information. This opens up a new tab with the course page in easyPCE.

Figure 8: The tags to the right of each course item on mouse over. Click "i" for more information, and click "-" to remove the course from the current schedule.

Session Restoration

It is important to note that the schedules Bob just created are persistent across sessions; every time Bob returns to *ReCal CS*, the previous schedules and enrollments are still there.

In the walkthrough above, we demonstrated a typical scenario——user choosing courses for

the next semester. Critical functionalities include interactions between course search results and calendar view, tabs for multiple schedules, and persistent data across sessions.

3. Back-end Design

Now that we have a basic understanding of what *ReCal CS* does, let us delve into the technical design.

3.1. Database Schema

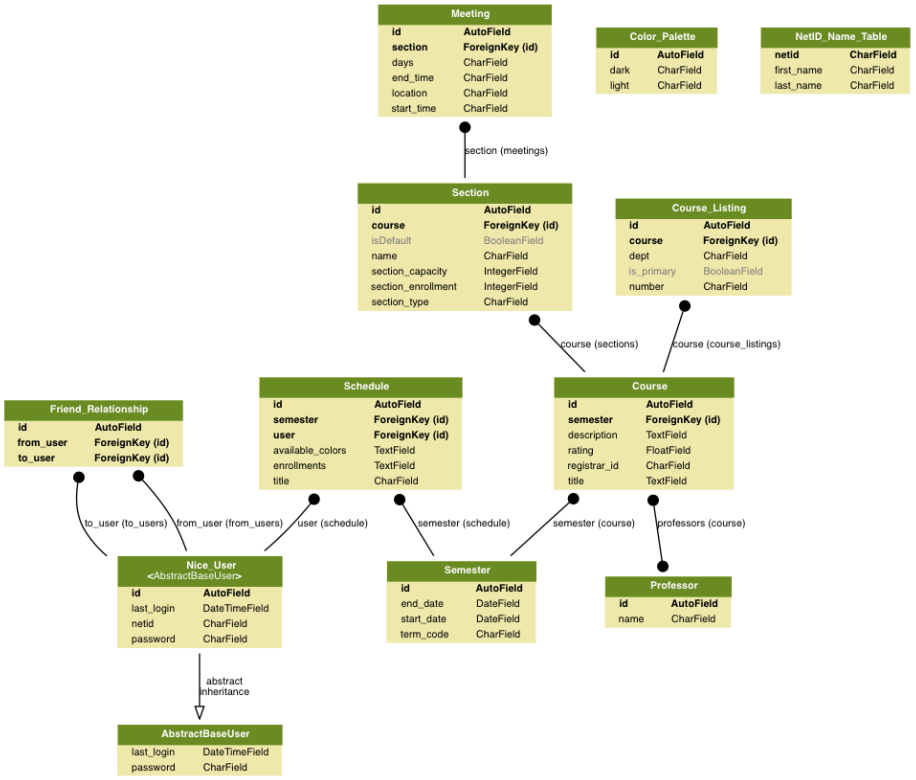

Figure 9: The database schema for *ReCal CS*.

As a continuation of *ReCal Dashboard*, *ReCal CS* reuses Django[15] as the backend service framework. We modified the database schema from *Dashboard* to match the requirements of *ReCal CS*. Overall, the database schema did not suffer much modification since its initial design. Below are a few notable changes that occurred over the semester:

We added a *Schedule* model to represent a set of enrollments. Note that the fields `available_colors`

and `enrollments` are stringified[1] and stored as TextFields for flexibility on the front end. `enrollments` is an array of JSON objects:

```
IEnrollment: {
  course_id: number,
  color: {
    dark: string,
    id: number,
    light: string,
    resource_uri: string
  },
  sections: Array<number>
},
enrollments: Array<IEnrollment>
```

The `Course` model only consists of the metadata of a course, whereas `Sections` and `Meetings` contain section-specific information such as the meeting times and the section capacity. Each `Course` has one or more `Course_Listings` (e.g. COS126/EGR126).

The table `Color_Palette` consists of two fields: `dark` and `light`, each storing a hex string of a color. Currently, this table is initialized with 10 default pairs of colors. The usage of these colors will be discussed in section 5.

The `Friend_Relationship` table was planned for storing multi-user relationships. Although we were unable to complete that feature over this semester, we believe that it is indispensible feature and have staged it for a future release.

3.2. Tastypie[4]

We included a Django module, *Tastypie* to provide a RESTful API service for the front-end to consume.

Using *Tastypie*, we wrapped python objects into JSON objects and only exposed a few fields through the API. `color_palette`, `course`, `semester`, and `user` are queried by the front-

end to initialze data. They are aggressively cached by *Tasytpie* on the back-end as the data rarely changes. On the other hand, `schedule` is contacted most frequently for updating user schedules. Thus, we allowed all `HTTP` methods: `GET`, `POST`, `PUT`, `DELETE` through the `schedule` API, but only exposed `GET` for the others.

```
color_palette: {
  list_endpoint: "/course_selection/api/v1/color_palette/",
  schema: "/course_selection/api/v1/color_palette/schema/"
},
course: {
  list_endpoint: "/course_selection/api/v1/course/",
  schema: "/course_selection/api/v1/course/schema/"
},
schedule: {
  list_endpoint: "/course_selection/api/v1/schedule/",
  schema: "/course_selection/api/v1/schedule/schema/"
},
semester: {
  list_endpoint: "/course_selection/api/v1/semester/",
  schema: "/course_selection/api/v1/semester/schema/"
},
user: {
  list_endpoint: "/course_selection/api/v1/user/",
  schema: "/course_selection/api/v1/user/schema/"
}
```

4. Front-end Design

While developing ReCal dashboard, we wrote a lot of library code for server connections and communication between JavaScript modules. In particular, we devised a module called `EventsManager`,

which essentially took care of all the computation on the front-end, and contacted the server every 5 seconds for syncing. This model, while clever, was undesirable for two main reasons. First, polling the server every few seconds caused major performance issues after a while. Second, it put the web application in a global state—either in sync or out of sync. If the user closed the browser while the local model is out of sync, then the server-side model would not get updated. This system, therefore, required meticulous manipulation of server connections. We had trouble re-syncing after disconnecting and re-connecting to the server, for there were simply too many Ajax connections to manage.

4.1. AngularJS[5]

For *ReCal Course Selection*, we decided to go with AngularJS on the front-end. AngularJS is a JavaScript framework that provides neat functionality such as declarative templates with data-binding, Model-View-ViewModel patterns, but most importantly, it simplifies server connections by utilizing RESTful APIs[13].

To understand an Angular application, we must first understand data-binding:

> "data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes. This is awesome because it eliminates DOM manipulation from the list of things you have to worry about."[5]

In other words, AngularJS binds DOM elements to plain JavaScript objects, and dynamically updates bi-directionally. Data-bindings significantly simplify the code as it eliminates the need for DOM event listeners. This allows us to abtract the view logic from the JavaScript code and embed it in HTML templates.

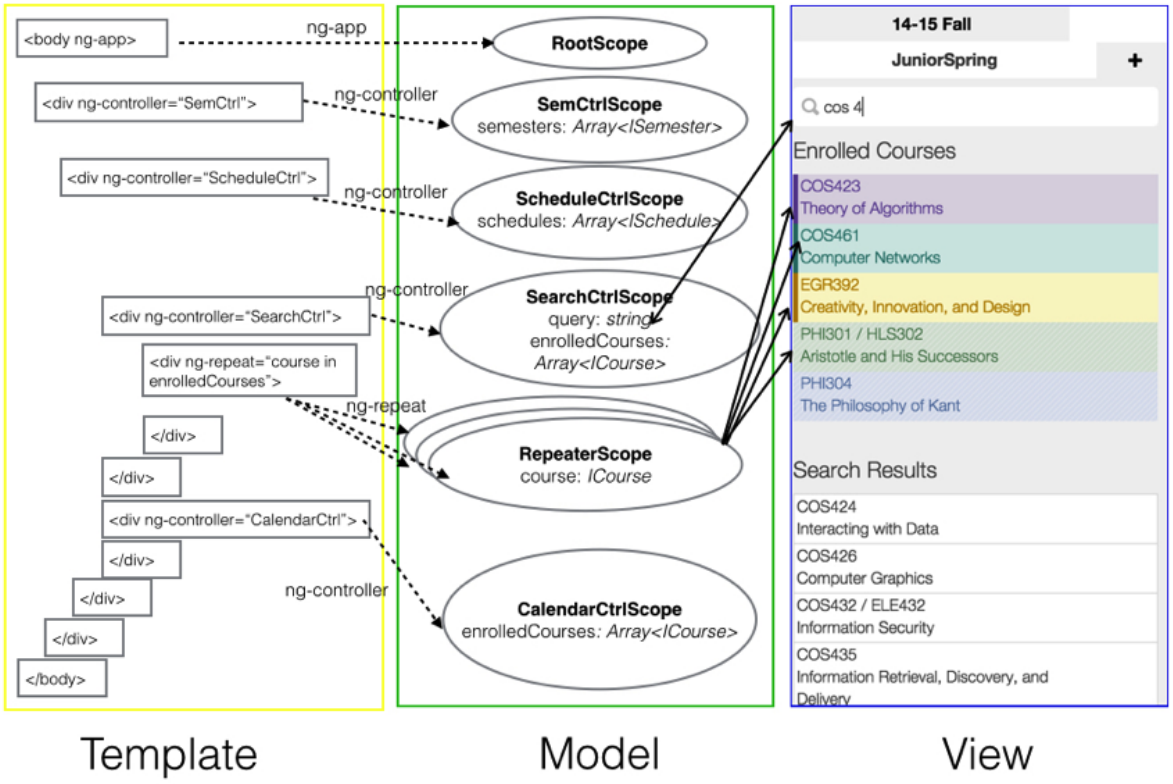

Figure 10: An overview of the template-model-view structure.

ReCal Course Selection mainly consists of four classes of JavaScript modules: *Controller*s, *Service*s, *Directive*s, and *Model*s. *Controller*s are responsible for handling the business logic between the template and the models. *Service*s are substitutable objects that help organize and share code across the application. *Directive*s are reuseable components that expand the HTML vocabulary—in official AngularJS terms, they are "markers on a DOM element that tell AngularJS's HTML compiler to attach a specified behavior to that DOM element or even transform the DOM element and its children." A model, or a "scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events."[6]

In *Figure 10*, we see that the HTML template contains *directive*s `ng-app`, `ng-controller`, and `ng-repeat`. These *directive*s correspond to *scope*s, which are bound to DOM elements such

as the enrolled courses panel.

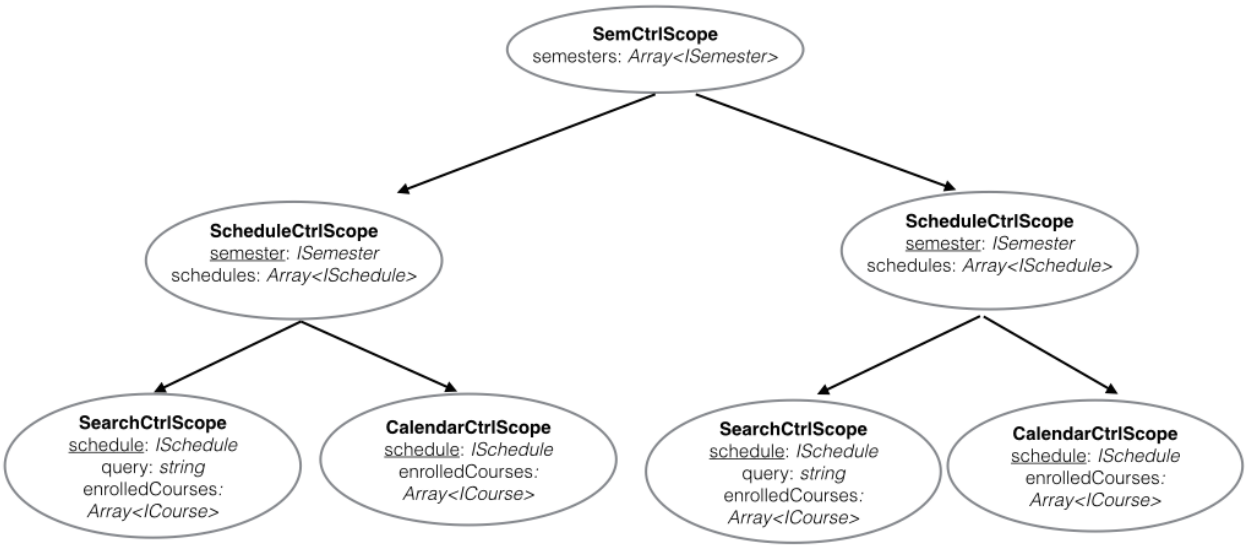

Figure 11: Controller Scope hierarchy

4.2. Design Pattern Tradeoffs

In ReCal Course Selection, SemCtrl consists of an array of semester objects. By inheritance, for each semester in the array, there is a schedule controller that inherits its parent's semester and consists of an array of schedules for that semester. The advantage of this design is that this data structure precisely parallels the view logic: for each semester tab, there is collection of schedule tabs; for each schedule tab, there is a search controller and calendar controller. On the other hand, however, this kind of parent-children scope inheritance sacrifices modular independency and violates the principle of loose coupling. In other words, each controller must be positioned relative to its parent controller; any future modification to the view structure—removing semester tabs, for example—demands a major reorganization of the controller code.

This kind of tradeoff between flexibility and code simplicity is a recurrent theme in the development of this project. As the controllers are mainly application-specific, and not intended to be used as library code for other projects, we more often chose code simplicity over flexibility. While designing services and directives, however, we preferred flexibility over simplicity as services and directives are meant to be reusable in different context.

4.3. UI Design

We spent a lot of time perfecting the user interface of *ReCal CS*. We wanted it to be clean and simple, yet at the same time intuitive and powerful. We drew attention to details, listened to user feedback, and redesigned almost every single component for the better.

Color Schemes and Flat Design

First of all, we wanted *ReCal CS* to have a minimalistic feel. Drawing inspiration from popular calendar applications such as *Sunrise*[14] and Mac OS X's built-in *Calendar*, we decided that a solid, gray-scale background with vibrant colors for calendar events was the most elegant solution. As a result, *ReCal CS* provides enough contrast but does not distract the user from the main elements.

Flat design has become increasingly popular among designers. Apple's *iOS 7*, Microsoft's *Metro*[10] theme, and Google's *Material*[7] design all follow similar design patterns. In designing *ReCal CS*, we removed drop shadows and border radii, used primarily solid colors, and highlighted information by increasing their font-weight. Although there is no right or wrong, we believe that these design decisions made the website more approachable to the users.

Mouseover Behaviors

Another crucial element to user interface is providing feedback—whenever a user interacts with a DOM element, he/she should see that something is happening. Defining mouseover behavoirs is the easiest way to provide feedback on a website. We heavily used `mouseover` and `mouseleave` events.

- If an element is clickable, we changed the cursor style to `pointer`.
- Mouseover enrolled course panel shows two clickable tags: one for more information, the other for unenrolling the course. *Figure 8* is an example.
- Mouseover an enrolled course not only toggles two clickable tags for more information and enrolling in the course, but also shows a preview of this course. As shown in *Figure 5*, all sections of COS217 are shown in gray as the cursor is hovering over the course item COS217 in the search

results.

- Tooltips are shown if the user does not take action after hovering over a clickable item. *Figure 2* is an example of the tooltip for adding schedules.

Animations

Without animations the website is dull and lackluster. Too many animations, however, would undo the minimalistic effort and distract the user from the content. As a compromise, we added subtle animations only as a way to provide feedback:

- The delete buttons for schedules appear slowly as the user hovers over a schedule tab.
- Enrolling or unenrolling a course triggers a sliding effect: the course item slides to its left as it is removed from its panel group, and then slides back into the opposite panel group as it is added.

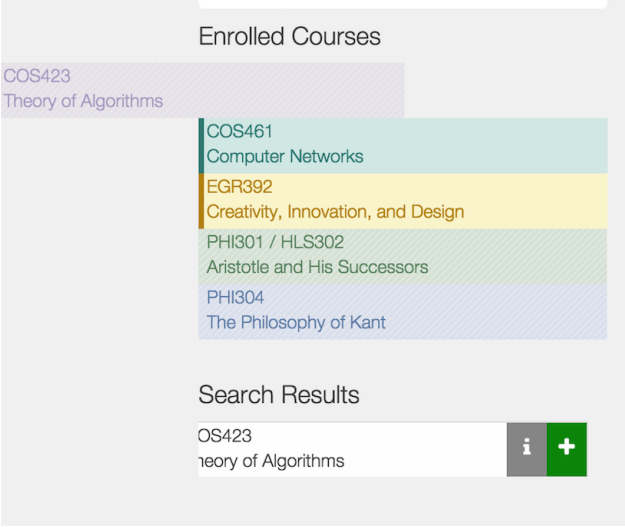

Figure 12: The course item COS423 is entering the panel group *Enrolled Courses*

- A loading bar display the progress of course initialization. This is a critical improvement over none, since it usually takes over 10 seconds for the courses to be loaded for a first-time user without cache.

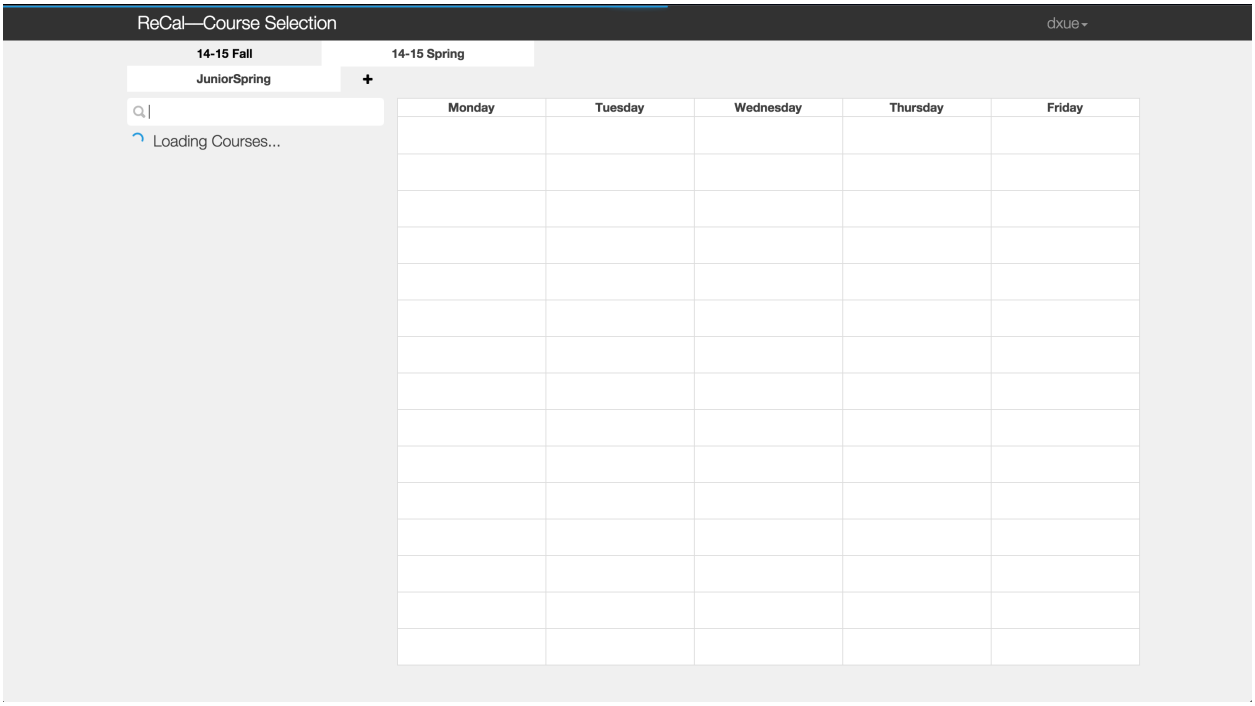

Figure 13: A loading bar at the top and a spinner under the search bar.

5. Testing and Evaluation

AngularJS comes with a strong set of testing tools. We used *Karma*[9] and *Jasmine*[8] to write unit tests for controllers, services, and models, while using *protractor*s to write end-to-end tests for corner-case scenarios. Currently, the unit-test statement coverage is 95%. The missing 5% mostly comes from generated code by TypeScript[3] from simple class inheritance[16]. Notable bugs caught during testing include:

- Searching for a course before courses are loaded causes a scripting error.
- Succession of reenrollments caused a scripting error.
- If there are enrollments from previous sessions, but the courses were not intialized, the calendar events would not load until refresh.

Although we did not conduct any type of formal evaluation, we asked 12 Princeton students to try out *ReCal CS* and provide verbal feedback. We noticed that

- Without a loading bar, the user is clueless while the courses were being loaded for the first time.
- The user expects the cursor to become a pointer if it is positioned on a clickable element.

- The user does not know where to start at the beginning without any guidance.
- Typical users only search for 3-letter department codes; they rarely stumble upon advanced queries without guidance.
- The user does not understand that "i" stands for more information.
- The user wants to change course colors and schedules names.

We learned a lot from our testers, and made changes accordingly when possible. Some requests, such as a friends feature, and the ability to change course colors, will be added in the future.

6. Related Work

6.1. ICE—Integrated Course Engine

Many have attempted building course selection tools. The most widely used one by the Princeton student community is ICE, short for *Integrated Course Engine*. ICE also started as a COS333 project, and involved into a senior independent work project of Gyeong-Sik, a member of the original ICE team[2]. ICE is notable for its functionality. It provides course search, reviews from the student course guide, course information from registrar's website, and allows one to view others' schedules.

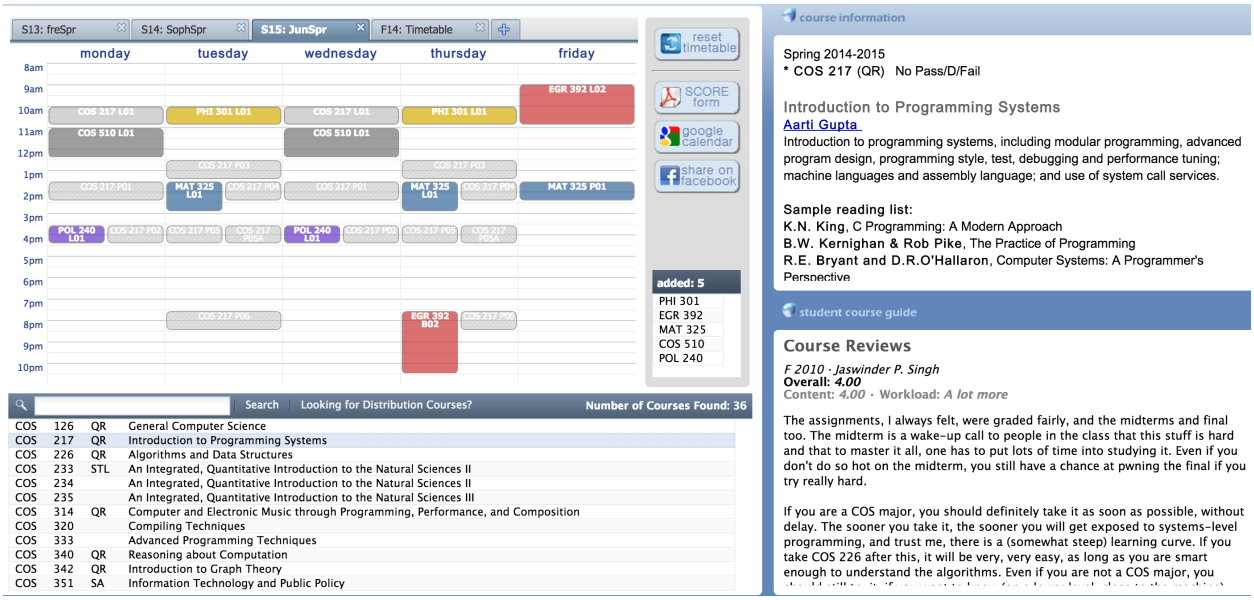

Figure 14: The most popular course selection tool among Princeton students.

ReCal CS learned a lot from ICE. Color coding courses, using tabs to represent schedules are all great ideas that we appreciated and took into our design. Yet, we also made an effort to distinguish ourselves from ICE.

ICE's performance suffered due to the fact that it had to make a server connection for every query. Unlike ICE, *ReCal CS* was designed to perform all computations on the front-end—this is a paradigm shift in recent years as computers have become increasingly more powerful. By loading the course data into JavaScript through 1 call, and then storing them in the browser's local storage, everything query on *ReCal CS* appears to be instantaneous.

Responsive design—the approach to web design that aims at serving devices with different screen properties—is also a very young idea. Only until recent years have smart phones and tablets become popular internet-browsing devices. Hence, it is understandable that ICE does not optimize for a smaller screen. *ReCal CS*, on the other hand, is repsonsive by design, changing its layout for the best viewer experience on different devices using the CSS `@media` property.

As ICE is retiring after the spring semester, we hope that *ReCal CS* can serve the community as successfully as ICE did.

6.2. TigerHub

TigerHub replaced SCORE (the Student Course Online Registration Engine) in November, 2014[12]. It features a calendar-based course planner similar to ICE's and *ReCal CS*'s. It has the advantage of being directly connected to the school's offical database and is always up-to-date. We consider *TigerHub* a competitor; however, we believe that, at the moment, *ReCal CS* still provides value for the users.

First of all, *ReCal CS* is much faster than *TigerHub*. *TigerHub*, just like *ICE*, makes a database query for every single operation. Searching for all computer science courses on *ReCal CS* seems instantaneous, while it takes 0.4 seconds on average on *TigerHub* user the same conditions.

Second, *TigerHub* allows users to add custom events to the calendar. As a result, their calendar must be able to cover 24 hours of a day, and is unable to fit all class hours, 8:30am-11:00pm, in one

screen.

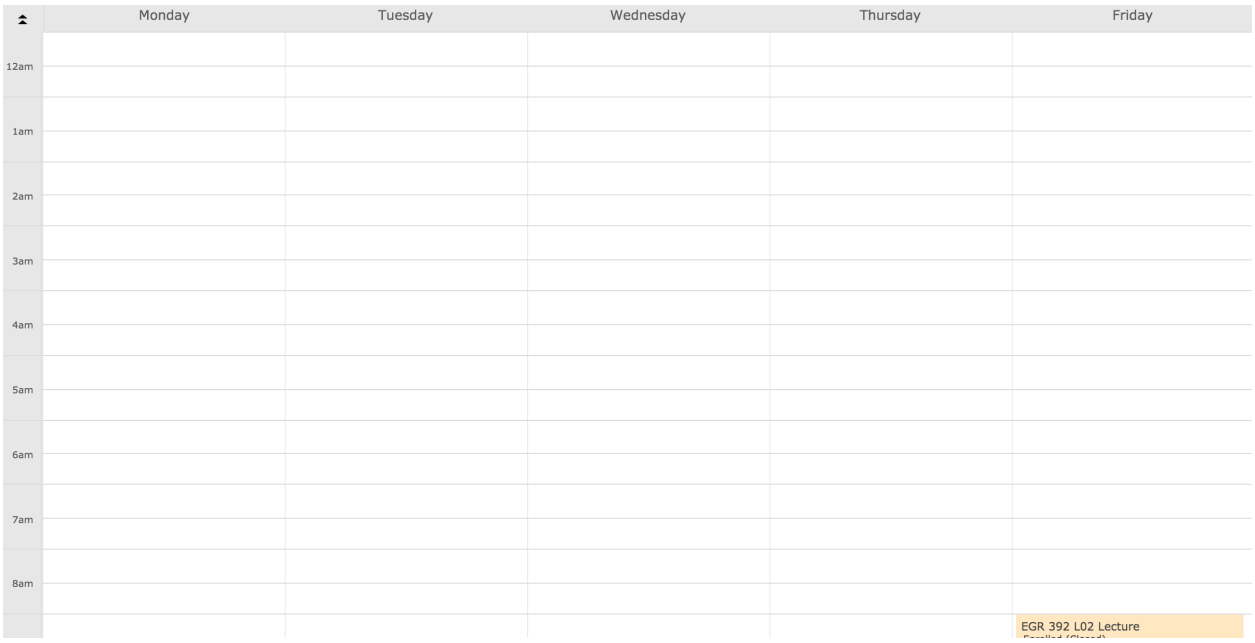

Figure 15: TigerHub's calendar begins at midnight. The user must scroll down to view more.

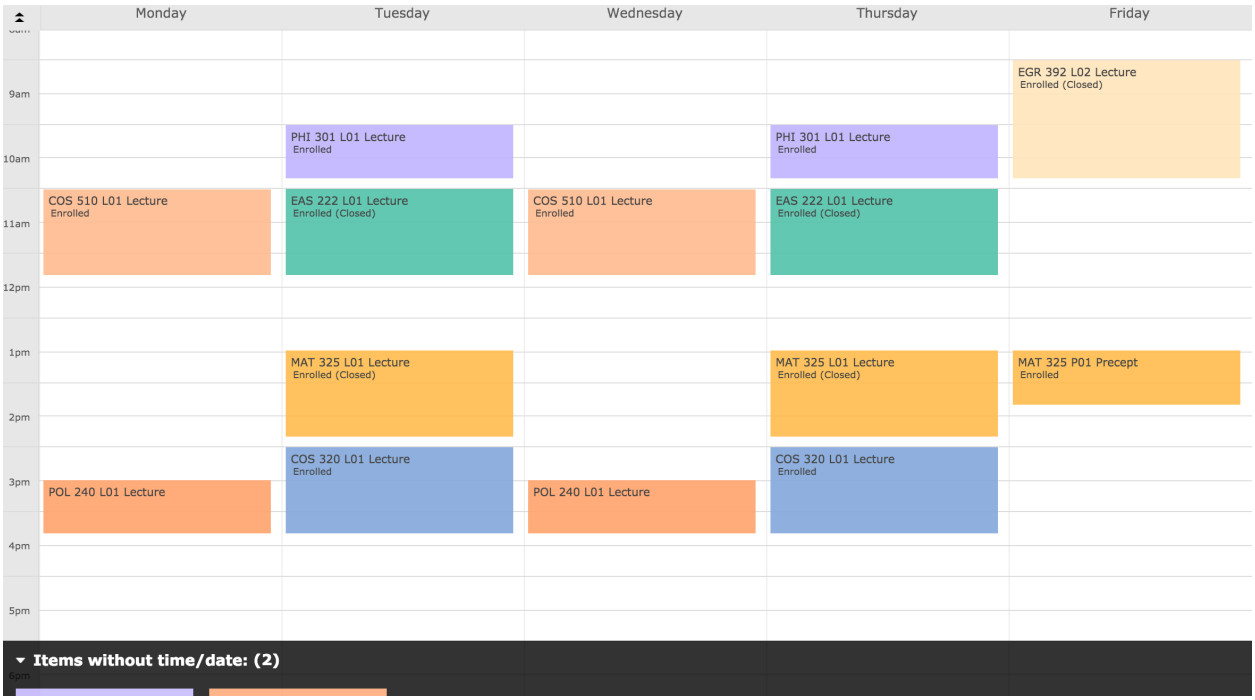

Figure 16: TigerHub's calendar cannot fit all class hours. In this figure, 2 courses at 7:30pm-11:00pm are hidden.

7. Conclusion and Future Work

At the heart of course selection is information-based decision making, and as long as course information remains de-centralized, course selection would always be a non-trivial task. *ReCal Course Selection* aims to lower its difficulty by gathering and pleasantly visualizing course information. Built on top of the ideas of ICE, *ReCal Course Selection* adopts modern web design ideas such as flat design and responsive design to provide a better experience.

We currently plan to launch *ReCal* during spring 2015. We strive to complete the *ReCal* experience by finally linking the databases for dashboard and course selection on both web and mobile platforms so that our users are always on top of their coursework.

References

[1] "JSON.stringify() - JavaScript | MDN," https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify, 2005, [Online; accessed 06-January-2015].

[2] "ICE: Integrated Course Engine," ice.tigerapps.org/ICE3/war/about.html, 2011, [Online; accessed 06-January-2015].

[3] "Welcome to TypeScript," http://www.typescriptlang.org/, 2012, [Online; accessed 06-January-2015].

[4] "Tastypie - RESTful APIs for Django," http://tastypieapi.org/, 2013, [Online; accessed 06-January-2015].

[5] "AngularJS Developer Guide," https://docs.angularjs.org/guide/, 2014, [Online; accessed 06-January-2015].

[6] "AngularJS Developer Guide," https://docs.angularjs.org/guide/scope, 2014, [Online; accessed 06-January-2015].

[7] "Introduction - Material design," http://www.google.com/design/spec/material-design/introduction.html, 2014, [Online; accessed 06-January-2015].

[8] "Jasmine: Behavior-Driven JavaScript," http://jasmine.github.io/, 2014, [Online; accessed 06-January-2015].

[9] "Karma - Spectacular Test Runner for Javascript," http://karma-runner.github.io/0.12/index.html, 2014, [Online; accessed 06-January-2015].

[10] "Metro (design language)," http://en.wikipedia.org/wiki/Metro_%28design_language%29, 2014, [Online; accessed 06-January-2015].

[11] "PCE Home," http://easypce.com/, 2014, [Online; accessed 06-January-2015].

[12] "Princeton University - TigerHub is new course planning, academic information site," http://www.princeton.edu/main/news/archive/S41/39/01K96/, 2014, [Online; accessed 06-January-2015].

[13] "Representational state transfer," http://en.wikipedia.org/wiki/Representational_state_transfer/, 2014, [Online; accessed 06-January-2015].

[14] "Sunrise Calendar," https://calendar.sunrise.am/, 2014, [Online; accessed 06-January-2015].

[15] "The web framework for perfectionists with deadlines," https://www.djangoproject.com/, 2014, [Online; accessed 06-January-2015].

[16] "Typescript generates javascript code for simple class inheritance," http://stackoverflow.com/questions/22901249/typescript-generates-javascript-code-for-simple-class-inheritance, 2014, [Online; accessed 06-January-2015].