PrincetEvents : A User's Guide

Divya Mehta, Maia Ezratty, Alexa Wojak, Mitchell Hamburger, Harry Heffernan

Spring 2016

PrincetEvents has three main tabs: Calendar, Upcoming, and Matches. Upon opening the app, the user is presented with the CAS login screen and after successfully entering a valid university NetID and password, he or she is delivered to the first of the tabs, Calendar(1). On this page, the app displays all of the days for the next year, with today's date highlighted in orange. Selecting any of these days guides the user to what we call the day page, which is a list of events on campus for any given day. The user can then select any of these events, navigating them to the event page, which contains information for a specific event and will be described in more detail later on in this document(2). To return to the calendar from this page, the user can press the back button in the top left corner of the screen to return first to the day page and then to the calendar, or else press the calendar tab bar icon at the bottom of the screen.

The next tab, Upcoming, directs the user to a list of events sorted by day. If a day does not contain any events, then it is not displayed on the upcoming page(3). Selecting any of these events sends the user to the event page, which includes important information about individual events. This same page can also be reached using the calendar by selecting the date of the event and selecting the event from the resulting day page. The page presents the event title, the opponent or description based on whether or not the event is an athletic one, and finally, the date, time, and location. If the event requires a ticket, then two buttons appear at the bottom, reading "Request Ticket" and "Post Ticket". These buttons are the tools for our ticket exchange

functionality. Both of the buttons navigate to the same page, which prompts the user for his or her first and last name. This information, combined with the NetID which was stored from the CAS login, is the basis of the ticket exchange.

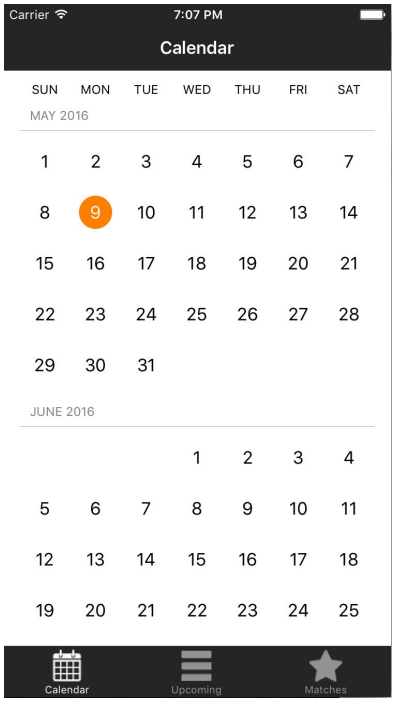
When the user submits his or her information to request or post a ticket, his or her information (first name, last name, netid) is added to the end of the appropriate list. That is, if the user requests a ticket for a dance show, and there are fewer people posting tickets than requesting tickets, then he or she will be put at the end of the list of all other users requesting tickets to this same show. Similarly, if the user posts a ticket for the dance show, and there are fewer users requesting tickets than posting, he or she will be put at the end of the list of all other users posting tickets to this show.

In either of these cases, the user will be notified if he or she is involved in a match. If the user is in the app at the time of the match, then a badge number will appear over the matches tab, showing how many unviewed matches he or she has(4). If the user is not in the app, then the user will be notified via push notification that he or she has a match to view and a badge number will appear over the app icon showing how many unviewed matches the user has(5). A match is considered unviewed (and the badge numbers on the app icon and matches tab remain unchanged) until the user selects the Matches tab.

The third and final tab, the Matches tab, presents the user's information regarding ticket exchanges(6). If the user has any matches, they will be displayed in this tab, showing the first name, last name, and netid of the other person involved, as well as the event title for which the match was made and whether the user was the buyer or seller. The page consists simply of a list of the user's matches with a "clear matches" button at the bottom of the page. When the user

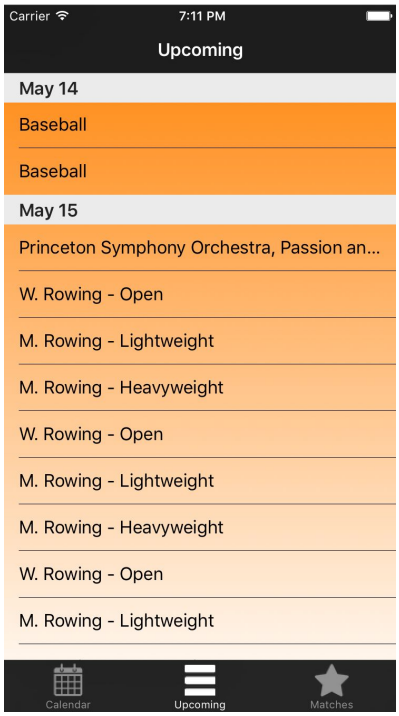
presses this button, all of his or her matches are cleared from the page. The user can exit the Matches tab by selecting either the Upcoming or Calendar tabs at the bottom of the screen.

(1)

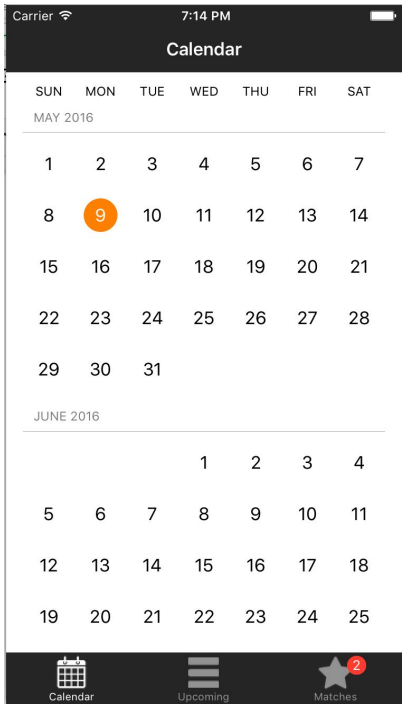


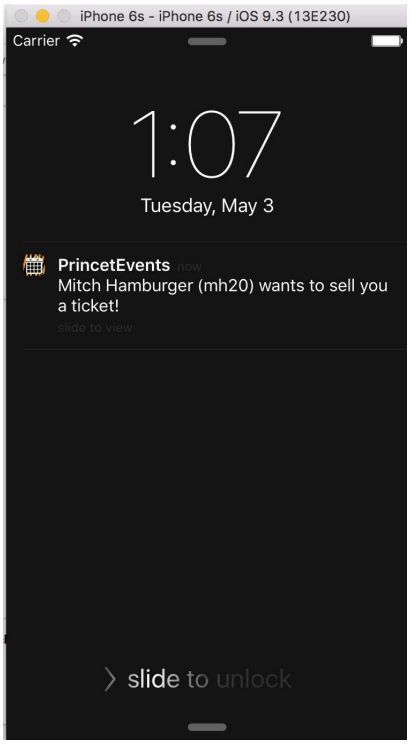
(2)

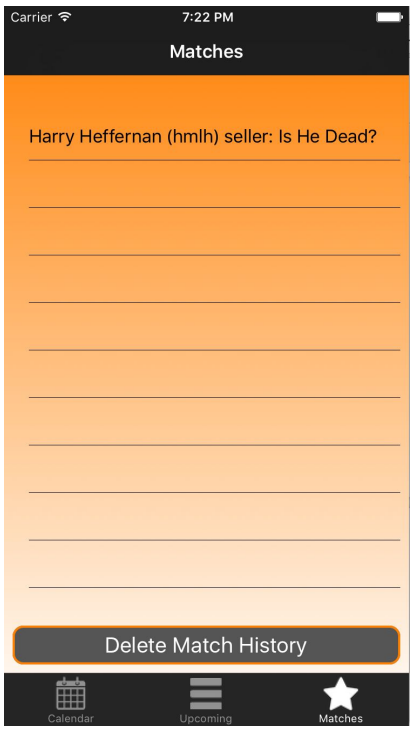
(3)



(4)



(5) 
iPhone 6s - iPhone 6s / iOS 9.3 (13E230)
Carrier
1:07
Tuesday, May 3
PrincetEvents now
Mitch Hamburger (mh20) wants to sell you a ticket!
slide to view
slide to unlock

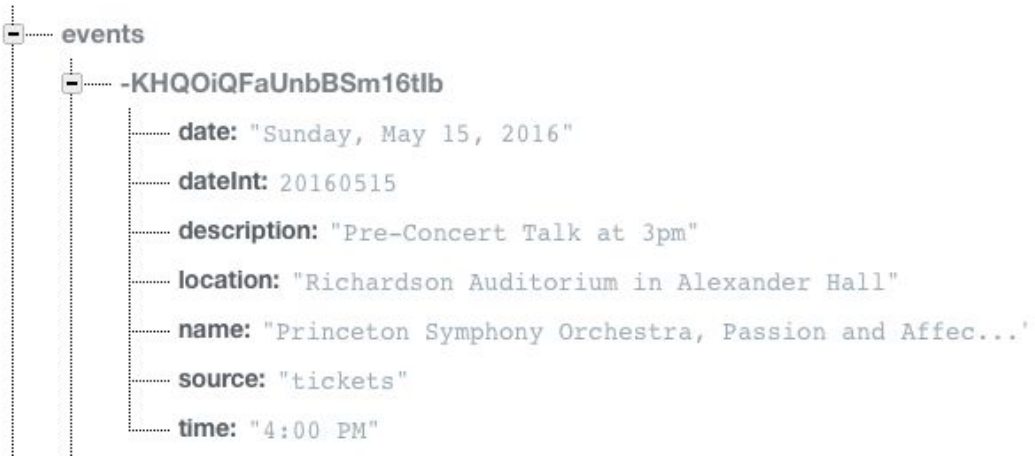
(6) 
Carrier
7:22 PM
Matches
Harry Heffernan (hmlh) seller: Is He Dead?
Delete Match History
Calendar
Upcoming
Matches

PrincetEvents : A Developer's Guide

I. Overview

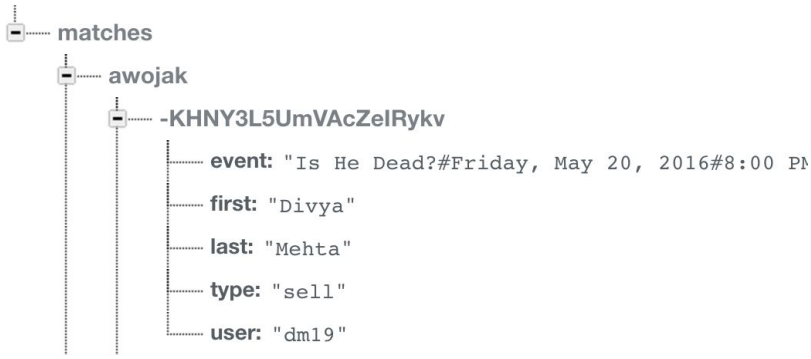
1. Firebase

a. **Events Database**: In our firebase database, we store all of the information about the events that we want to present on each event page in the app. We scrape this data from our two websites, the Princeton ticketing site and the goprincetontigers site, so that we can access it as needed throughout the app.



b. **Matching/Notifications/Buy/Sell Database:** There are four tables within our database that deal with the matching functionality of PrincetEvents. The **buy** and **sell** tables are populated when a user requests or posts a ticket that at the time has not matches. Entry include the event's information as well as the user's information to enable the creation of matches and then transmission of user information to the matching party. The **matches** and **notifs** tables in the database are organized by branches corresponding to a user's netId. This allows for the most efficient querying of the table. Data is added to these tables upon creation of

a match. Data is deleted from the notifs table as soon a push notification is deployed, whereas data is only deleted from a user's matches table if they click the **Delete Match History** button.



2. **Remote Servers:**
    a. **Heroku:** PrincetEvents uses heroku as a remote server to host our scraping script for the athletics page. We deployed the script from heroku, and within the python code we use **apscheduler** to remove the current athletics data from the firebase database and rescrape the data, so that we always have the most up to date information.
    b. **cPanel**: We host our CAS login script from cPanel. We chose to host this from cPanel rather than heroku, because heroku free applications require six hours of sleep every 24 hours.
3. **Dependencies**
    a. **Pods:** Throughout the project we imported a few different pods like Alamofire, Firebase, HTMLReader, and PDTSimpleCalendar.

II. <u>Frontend</u>

1. **CAS:** To enter the application you must first log in with Princeton's Central Authentication System (CAS). To add CAS to our application we used the CASClient.php and a modified version of CAStest.php provided from

http://www.cs.princeton.edu/~cmoretti/cos333/CAS/. All of the PHP scripts used for CAS are hosted through cPanel, specifically the directory https://awojak.mycpanel2.princeton.edu/333. Within the directory, index.php calls CASClient.php, which returns the user's netId upon validation. Index.php then redirects to index1.php with the user's netId in the id field of the URL. This field can then be accessed by PrincetEvents allowing for your personal matching data to be displayed.

2. **Controllers and Queries:**
    a. **WebController.swift**: WebController.swift is the entry point for PrincetEvents. The controller contains a UIWebView which loads CAS on launch (https://awojak.mycpanel2.princeton.edu/333). Once the user has signed in with CAS this controller also parses their netId from the returned URL and sets the global variable **globalNetId**. Furthermore, this controller performs the segue **LoggedIn**, which sends the user from the CAS page to the PrincetEvents' calendar. Finally, this controller is where push notifications are deployed from. Push notifications utilize a **ChildAdded** type of query, so when a new entry is added to the user's branch of the notification table the notification code within the query statement will be run. After sending the push notification the corresponding entry is deleted from the notifications table.
    b. **CalWrapper.swift**: This view controller contains a container in which the calendar resides. It also handles the setup around passing a date as an NSDate to the next page through the segue **DayView** so that when a date is selected in the calendar the segue is setup to have associated date.
    c. **SecondViewController.swift**: The table of events from the upcoming tab is handled by this controller. The controller queries the **events** table of the firebase

database for all entries after the current date and orders them on the field **dateInt** (puts them in date order). The results of the query are then separated by date to create the subsections in the table of type UITableView. This controller also handles clicking on an event in the upcoming table and passes the associated eventId in the segue **UpToEvent**.

d. **DayTable.swift**: The DayTable.swift controller is the controller that handles the table displayed when the user selects a specific date on the calendar. This page queries the database for all entries with the same **dateInt** as the date which the user selected. These are displayed in a UITableView. This controller also controls the segue **EventView**, which brings the user to a single event page when they select a cell in the table.

e. **EventView.swift**: This controller is for a single a single event screen accessed either from the day table or the upcoming table. The event information is displayed by accessing the child event in the events table with the corresponding **eventId** through this URLString: "https://blistering-torch-3510.firebaseio.com/events/" + eventId. Furthermore, if the event is a ticketed event this controller shows the **Request Ticket** and **Post Ticket** buttons. Both of these buttons perform the segue **ToForm** and pass the event information as well as if the button pressed was to buy or to sell.

f. **FormController.swift**: FormController.swift is the controller for the page to submit a request to buy/sell a ticket. On the user side it has two fields for first name and last name and a button to submit. This controller also deals heavily with database management. When the user tries to buy/sell a ticket the controller first queries the opposite database table to see if a corresponding request already exists

using the following query:

refTwo.queryOrderedByChild("event").queryEqualToValue("\(eventTitle)#\(eventDate)#\(eventTime)").queryLimitedToFirst(1).observeSingleEventOfType. If they query does not return a result, than the user's information is entered into the correct buy/sell table in the firebase database. If the query does return a result, however, this indicates a match. The opposite user's information and the event information are entered into both the matches and notifications table of the database under the branch corresponding to the user's netId using the following four urlStrings.

urlString = "https://blistering-torch-3510.firebaseio.com/matches/" + globalNetId

urlStringTwo = "https://blistering-torch-3510.firebaseio.com/notifs/" + globalNetId

urlString = "https://blistering-torch-3510.firebaseio.com/matches/" + matchNetId

urlStringTwo = "https://blistering-torch-3510.firebaseio.com/notifs/" + matchNetId

g. **HistoryController.swift**: This controller displays the matches of the user currently signed into the PrincetEvents. It does this by returning all the results of the branch of the matches table which corresponds to the variable **globalNetId**. There is also a delete history button which clears this branch of the matches table from the database.

3. **Open Source Calendar:** The calendar used within PrincetEvents is an open source calendar called PDTSimpleCalendar (https://github.com/jivesoftware/PDTSimpleCalendar). Within the Xcode project, the view controller and view delegate for the calendar is called CalendarViewController.swift. The customization of the calendar occurs in this controller and also through changing the default colors set in the controllers PDTSimpleCalendar

directory within the Pods directory in Pods.xcodeproj. The CalendarViewController.swift also handles the functionality of selecting a date, and passes the information of the selected date in the form of a NSDate object to the next view.

4. **Notifications:** There are two main pieces to the push notifications. The first is the function in **AppDelegate.swift** called **registerForPushNotifications** enables push notifications generally in PrincetEvents. The actual sending of push notifications occurs from **WebController.swift**.

III. Backend

1. **Site 1: Athletics Events: BeautifulSoup**

To scrape the sports events from goprincetontigers.com, we inspected the HTML of the website to see where the HTML we needed was living, and wrote a python program using BeautifulSoup to extract that data.

2. **Site 2: Ticketed Events: PhantomJS & Beautiful Soup**

The ticketed events page required a more complicated solution. First of all, the data we needed to extract lived inside of a <script> tag that BeautifulSoup had trouble reading. Then, we were only able to access one page at a time, which limited us to about 5-7 events at a time. This was because accessing the "next page" button was private to us. We taught ourselves how to use something called PhantomJS in order to access more events at a time. We wrote a PhantomJS program that submits a search query by submitting a form. This way, we were able to access about 30 events at a time. Using document.scripts in the PhantomJS file found the part of the HTML that we needed (that BeautifulSoup couldn't find), so we had the PhantomJS output that part of the HTML and piped that output into a python file which uses regular expressions to find the information about each event that we needed.