FRISTRATIONS

PRODUCT GUIDE

by Kevin Bradicich, Chris Hsu, Christina Huang,
Mihika Kapoor and Stephanie Liu

USER PORTION

Purpose

Fristrations is the most efficient, user friendly, well-designed and smart way to alleviate the Princeton student body's fristrations (lack of ability to find classrooms in Frist). Using a combination of data from Princeton's Wi-Fi routers, and its own reservation system, Fristrations allows any member of the Princeton community to easily identify a open study rooms in Frist Campus Center. Users can browse every classroom in Frist and view the schedule of classes and reservations in that classroom for the day, search for the rooms that are currently available, add rooms to their "Favorites," and view their own reservations. Additionally, in the page that lists the schedule of a given room for the day, we show the location of the given room, whether there may be a person in a given room they have selected, and a population density map of Frist's floors that is updated every 5 minutes.

Features

Browse Rooms

After signing in to Fristrations through a secure CAS login page, users are able to browse information about every classroom in Frist. By tapping on a desired floor, users are directed to a list containing each room on the given floor. Another tap, and users can see all of the room information for a given day. Each 30-minute time slot is color-coded to represent the status of a room at any given time. If a time-slot is expired, it appears as gray. If a time-slot is already reserved, it appears as red. If a time-slot is available, it appears as green. For privacy, time-slots that were reserved by a user show the user's own netID, but slots reserved by other users hide the netID by displaying "Reserved." Time-slots that were reserved due to scheduled classes display the class name.

Floor Information

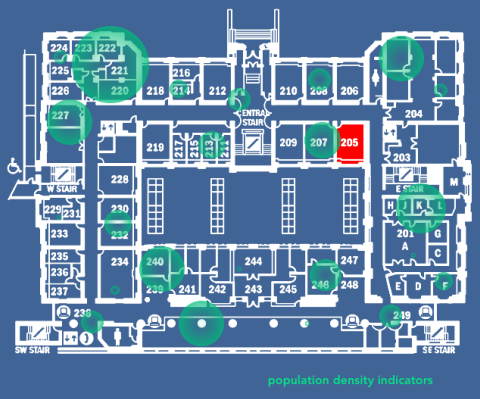
On each room's page, there is a floorplan of the corresponding Frist floor, and the location of the room is highlighted in red. Furthermore, the floor plan shows real-time population density using the number of devices connected to Wi-Fi. Dynamic circles shrink and expand to visualize the number of people currently nearby. Directly

FRIST

100 Level

200 Level

300 Level



This room is currently reserved.
4 people are nearby.

beneath the floor plan, the user can see numerically how many people are nearby.

Currently Available

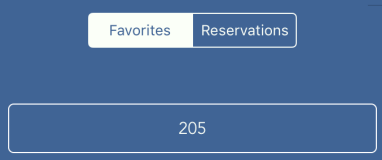
Users can see what rooms are available at any given time by clicking on the "Available" tab, which will display a list of all rooms that are not reserved during the current time slot. By clicking on any of the available rooms, the user will be redirected to the room's page, where the user can then see more specific information about the room.

Reserve and Favorite

Fristrations operates via an intuitive touch interface. Users can reserve up to four time-slots per day by tapping on an available (green) opening. The time-slot will immediately change to reserved (red) and display the user's netID. Similarly, if a user's reservation has not yet expired, the user can unreserve by tapping on a reserved time-slot that displays the user's netID , and the slot will immediately change back to available (green). Users cannot reserve or unreserve time slots that have expired (gray), nor can they modify time slots that were reserved by other users or time slots that contain classes.

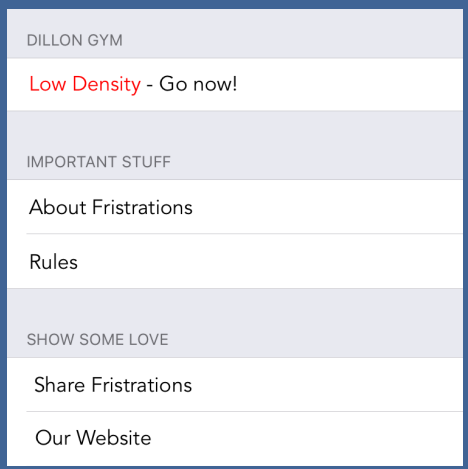
Users can also "favorite" or "unfavorite" rooms by clicking on the heart symbol in the upper-right corner of any room's page. The heart is originally colorless, but will become red when it is "favorited." Users can "un-favorite" a room by tapping on the red heart.

Both "reservation" information and "favorite" information are accessible through the "My Rooms" tab. By clicking on any room in either "Favorites" or "Reservations," the user will be redirected to the specific room's page.



Dillon Gym

Using the same router intelligence applied in Frist, Fristrations also indicates the density of people in Dillon Gym.



Share Our App

Users can share Fristrations across all forms of communication that are downloaded on their phones, such as Facebook, iMessage, Twitter, Groupme, Notes, and more. This feature is available on the "More Page." Sharing includes an enthusiastic piece of text about Fristrations and a link to our website.

DEVELOPER PORTION

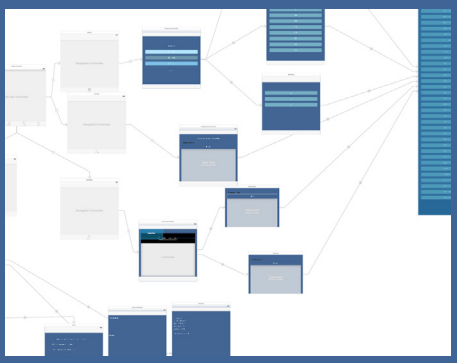
Overview

This portion will outline the structure of the Fristrations iOS application, which consists of two major components, a frontend tabbed iOS application and a backend Firebase database.

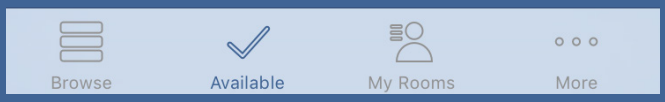
Frontend

Storyboard

As Fristrations was an iOS app, we created it using Swift in XCode. Fristrations' workflow was constructed using storyboard. The app itself can be navigated via a tab bar controller, which links to 4 navigation controllers that direct to the 4 main tabs in the app.



Tabs



BROWSE

The "Browse" tab lists all the floors of Frist implemented as buttons, and once a user clicks on one of the buttons, all the bookable classrooms in Frist on that floor are listed. Users can then select a classroom, implemented as buttons as well, and will be directed to that given room's page (generated from the room query). Users can go back and forth and choose to view different rooms.

AVAILABLE

The "Available" tab iterates through all the different rooms and lists the rooms that are available at the current time. Much like the "Browse" tab, users can click on an available room and will be directed to that given room's page (with the floorplan, population density, and daily schedule). The list of available rooms and their corresponding buttons are generated dynamically in a UITableView, and users can pull to refresh in case another user booked a room in the time that the current user took to look through the rooms. Additionally, if the user reserves a room for the given time, when the user goes back to the "Available" page, that room will no longer be listed, but that reservation will be listed under the user's personal reservations.

MY ROOMS

The "My Rooms" tab shows both the user's reservation information and favorite rooms. The page is implemented using UISegmentedControl. We first created a Container View connected to two UIViewControllers, one for Reservations and one for Favorites. Depending on which tab is clicked, one of the View Controllers becomes hidden.

The two UIViewControllers are implemented similarly. They both show a list of rooms, whether it is a list of favorites or a list of reservations. This is accomplished using a dynamic UITableView consisting of custom cells. Each custom cell contains a UIButton and a title, linking the entry in the table to the room's individual page.

MORE

This tab has four main buckets of info - CAS login/logout, Dillon, information for the user, and social/sharing outlets. It is implemented as a UITableView, with these four buckets of info organized as static groups of cells.

- The CAS part is where users can log out as well as view their login status. If the user logged in, the two UITableViewCells will display the netID and the option to sign out, respectively. If the user is not signed in, the two UITableViewCells will display "no user signed in" and an option to sign in. Logging in will launch a UIWebView of the CAS sign-in page, programmed to disappear when the user enters a correct netID+password. Pressing sign out will trigger and event action of displaying a UIAlertController that asks if the user definitely wants to sign out, whereby the options are "Dismiss" and "Sign Out".

- The Dillon cell relies on a series of if-statements about the number of devices currently connected, based on our prior analysis of trends over a week, and displays if it's a good time to go.

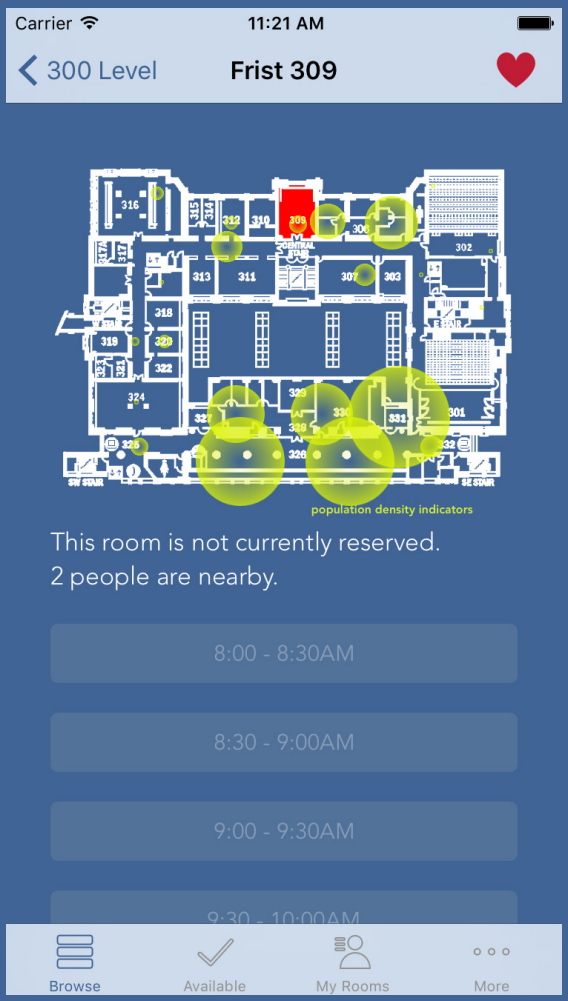
- Info for User: "About Fristrations" cell, "Rules" cell, "About COS 333" call, which each segue to a separate UIViewController and display text about the product, text about our rules, and launch the COS 333 website, respectively (see launching via Safari below).

- The user can share Fristrations by launching a UIActivityViewController; can jump to our Facebook page or to our website by launching Safari using the "openURL"

function; and can view the team members who developed the app with a segue to a new UIViewController. Using the "openURL" function allows a user to easily press back to return to the Fristrations app from Safari.

Room Information Page

The end page for nearly all flows in our app (except for the More tab) is the Room Information Page. The room information page is a dynamically generated page that shows the location of a given room on the floorplan (in an ImageView), the population density of the the given room (via a TextView), population density indicators for the whole floor (via resizable SubViews) and the room's schedule for the day (composed of dynamically updating buttons), which includes both classes and student reservations (the netID's of other students are not shown for privacy reasons).



The page is generated from the query that the user sends by pressing a button corresponding to a room, and then subsequently retrieving the appropriately timed reservation information from Firebase and populating the page accordingly. The colors and text of each time slot are set in RoomInfo.swift, which pulls the data from Firebase. Time slots that are reserved by others, or that have passed are not editable.

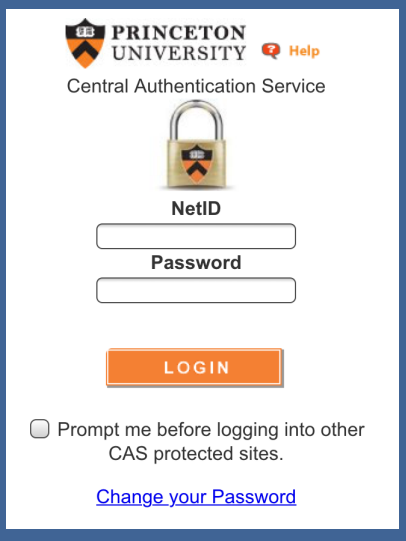
Router Visualization will be discussed further in the UI section.

Dynamically generating this page added modularity to our code and will make it easy for us to add rooms in the future.

CAS

The tabs are designed so that a user cannot view any revealing data unless logged in with a valid Princeton NetID, for security purposes. This is accomplished by opening a UIWebViewController that goes to the following url:

https://www.cs.princeton.edu/~cjhsu/fristrations/CASlogin.php



This php script, hosted in the public folder of a personal CS Department account, is accessible to all. This script will display a typical CAS login page to the user. Our script is derived directly from Professor Moretti's exam-

ple php CAS authentication script. If the user is properly authenticated, an HTML displaying only the user's netID will be returned. We use "Kanna," a Swift HTML parsing framework, to extract the user's netID to use throughout our app. When a user logs in, we associate a key, "netID", with a value, the actual netID, in the persistent NSUserDefaults dictionary which can be accessed throughout the entire app. All frameworks used throughout the project were installed using Cocoapods.

UI

ASSETS

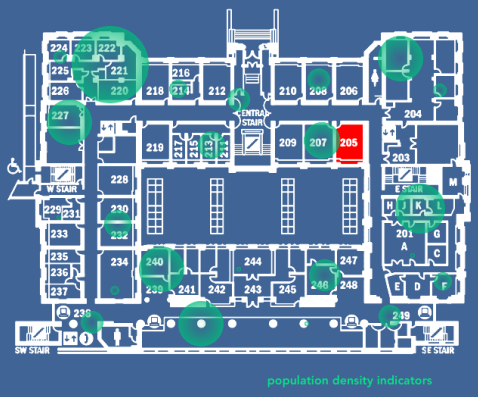
Since Fristrations is an app that is most useful if used by the entire student body, the app was designed to operate intuitively and cleanly. All icons for tabs, the app icon, the favorite icon, the router icons etc were generated using Sketch to ensure that they catered directly to the app and were not generic.

BUTTONS AND CELLS

All buttons and cells for TableViews were visually manipulated in their respective .swift files. Colors, spacing and other qualities are intended to contribute to a clean interface.

ROUTER & FLOORPLAN VISUALIZATION

Each room has its own floorplan in the assets folder, with the respective rooms highlighted in red. Floorplans were pulled from Frist's website and Photoshopped to match the aesthetic of the app. Each router visualization, used to indicate population density in various areas in Frist is generated as a subview with prespecified coordinates in RoomInfo.swift. Each floor has a different colored router. The routers resize every 5 minutes and have a width and height that is 3x the number of devices connected to the respective router.

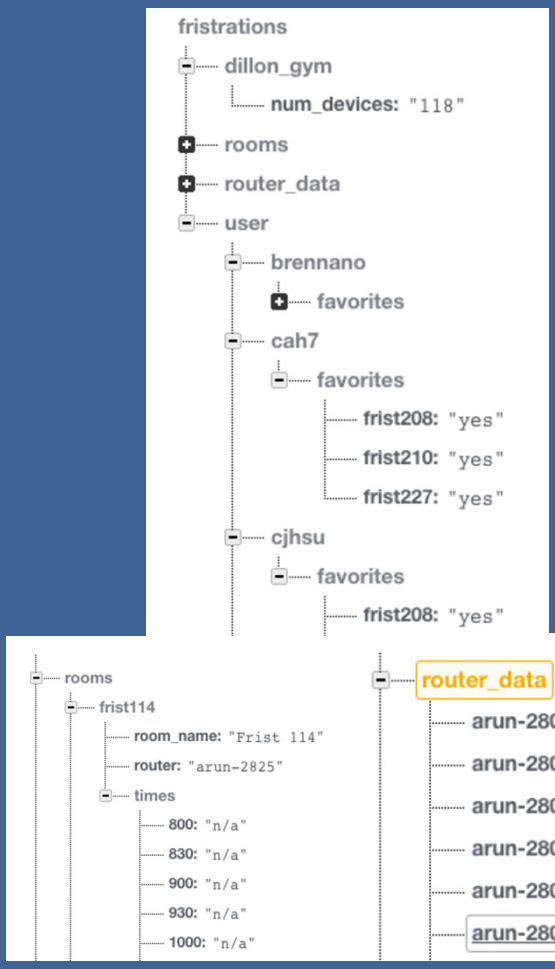


Backend

Firebase

We implemented our application's backend using Firebase, an automatically synchronized BaaS stored as JSON. Firebase has an expansive API with Swift and Python capabilities. Our database was split into three main

categories: rooms, routers, and users. Although not officially documented, we followed the best practices recommended on the official Firebase engineering blog, and implemented all our calls to Firebase in the viewWillAppear() method of all view controllers that used Firebase. Because of its constant synchronization, it was critical for us to properly design our data organization so that data requests would not be too large and expensive. To the left is a snapshot of how we currently organize our data:



We have five distinct requests that occur throughout our app:

A single room's schedule for the day.
A user's favorites
A user's reserved rooms
A specific router count
Dillon gym

Most of these requests yield one result or a dictionary of at most 30 results, keeping each data request relatively small in size. As a result, our application has no noticeable lag time, and reservations can be processed nearly instantly.

Princeton Course Scraping Scripts

In order to scrape classes from the registrar and populate the Room Information Page with the data, we have three python scripts. One of them, scrape_courses2.py (loosely based off a script given to us for Assignment 4), is run once a semester and scrapes the regsitrar for all classes in Frist, plus relevant data (class department and number, start time, end time, room number and days of the week). The term code in this script must be updated once a semester. The second two scripts, clear_courses.py and load_courses.py, are run daily, and based on the day of the week, they clear a rooms previous day's reservations and repopulate based on what classes there are that day (by scraping the file generated by scrape_courses2.py). Both write directly to Firebase, as it is very compatible with Python scripts.

User Profiles: Reserving/Favoriting Rooms

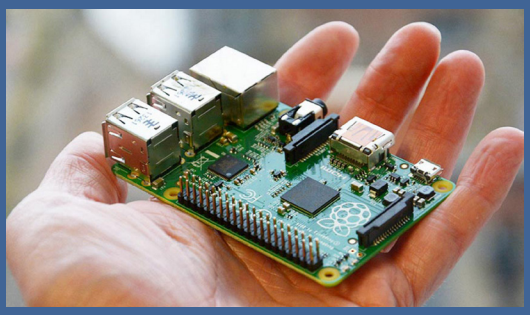
In order to save user data, we created user profiles for each netID.

Whenever a reservation was modified, we would modify

the information in Firebase in two places. First, we would change the Value corresponding to the specific time-slot, either to the user's netID or to "n/a," depending on if the room was already reserved. Next, we would add the room number and time slot under the user's netID in Firebase as a child of "reservations." Thus, we could limit the number of reservations to four by retrieving the number of children of a user's "reservations."

Whenever a room was added or deleted to "favorites" by tapping on the heart, we would change the image being displayed as the heart and add or delete the room number under the user's netID in Firebase, as a child of "favorites."

Raspberry Pi



With the way our database is currently organized, we must populate the room schedules with daily class schedules and have code that clears all reservations at the end of the day. To achieve this, we stored all our Python and shell scripts on a Raspberry Pi with a simple cron job that runs a single shell script encompassing all our scripts that is set to run at 3:30am everyday (a time when Frist is closed, and we hope most students are asleep).

Router Data Scripts

In addition to our scripts on the Raspberry Pi, we needed to continuously run scripts that would push the most recent statistics from routers in Frist and Dillon. Using the python-firebase module, we were able to write a combination of shell and Python scripts that pushed router data to our Firebase every five minutes. These scripts are hosted on OIT's own servers and the cron job is hosted on an OIT machine as well. Raw numbers for number of devices connected to each access point are collected by first calling an Aruba bash command that returns a listing of routers with columns of distinct information, including number of connected devices. We then run an awk script on this data to isolate the number of devices for each access point and push that info to the router_data section of our database.

All relevant router data scripts can be found in the aruba directory of our scripts folder. The script, aruba_script, runs in the following way:

- dillonAwk parses the current_stats returned from the

Aruba routers and prints an aggregate number of devices in Dillon Gym to dillon_num

- push_dillon_data.py is called to push the value from the previous step to Firebase

- push_router_data.py pushes a complete dictionary of all routers in current_stats to Firebase