inPerson

Product Guide

Alice Gao

Michael Peng

Anna Qin

Ioana Teodorescu

PRODUCT GUIDE

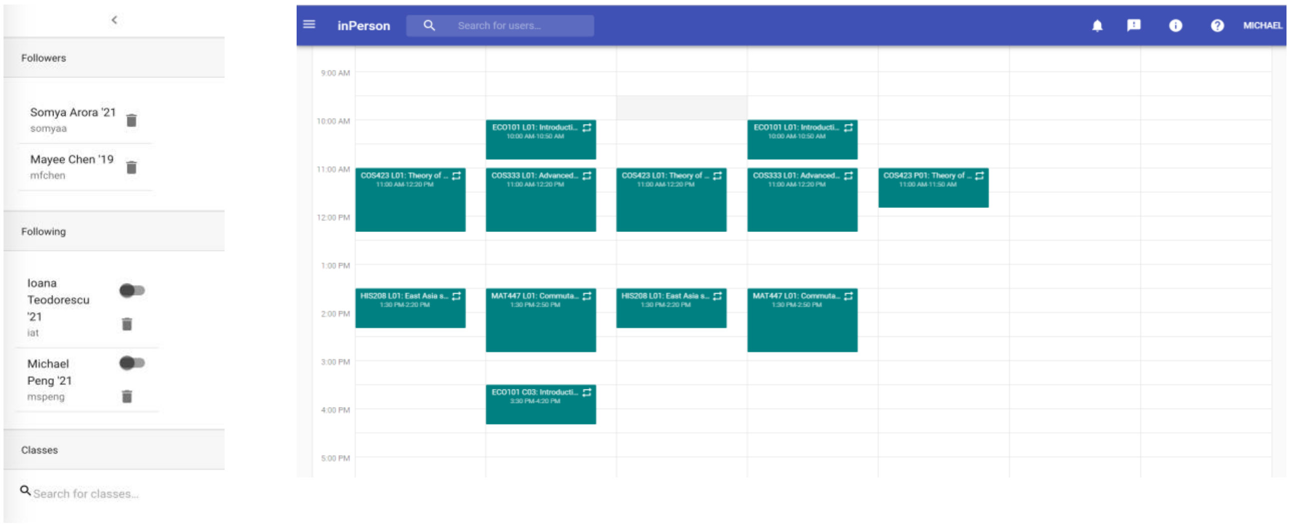
I. OVERVIEW

inPerson is a schedule-sharing web application designed to facilitate meetings between members of the Princeton community. Users may easily view the schedules of other users, which enables a simple and convenient way of checking when others are free to meet without needing to repeatedly fill out forms or send screenshots.

II. USER GUIDE

BASIC ACCESS

The web app can be accessed at https://nperson.herokuapp.com/. Upon navigating to that URL, the user will reach the landing page. From here, they may login through CAS (Princeton University's Central Authentication Service), and once authenticated, they will be sent to their home page, which primarily consists of the user's own calendar, with all of their events, as shown below.

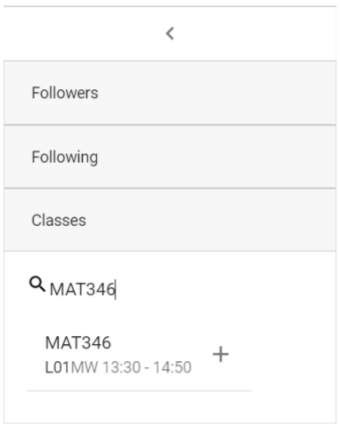


At the top of the page is a header. At the left end of the header, the user may click the menu button to display a sidebar containing three dropdown menus, as shown: "Followers" lists who is following the user, "Following" lists whom the user is following, and "Classes" displays a search bar for classes that the user may add to their schedule. To the immediate right of the menu button, there is another search bar that allows the user to search for other users to follow.

At the far right of the header, there are four buttons and the user's name. In order from left to right, they link to: a notification display that indicates pending follow requests sent to the current user, a Google form for feedback, an about page, and a tutorial explaining how to use the app. Clicking on the name allows the user to logout.

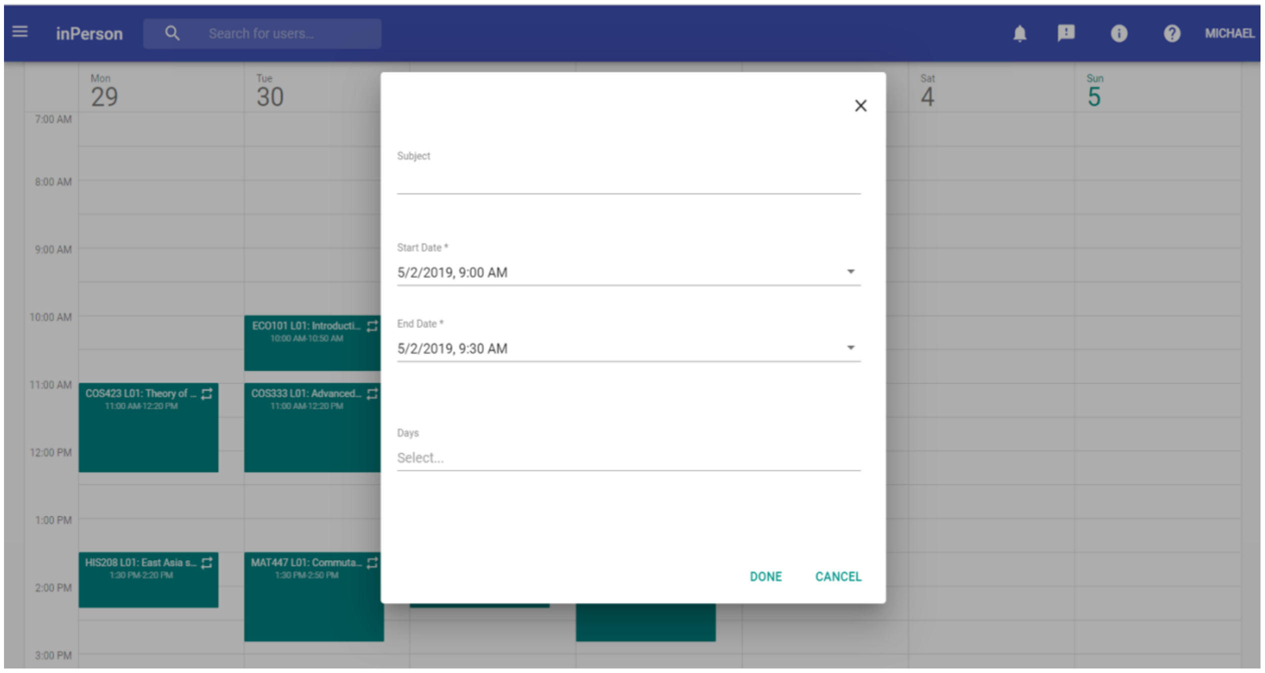
CALENDAR AND EVENTS

By default, the calendar is displayed on a week-by-week basis, with the times extending from 7:00 AM to midnight. The user can navigate between weeks by clicking the arrows at the top left. The date highlighted in blue is the current date, and the red bar on the calendar indicates the current time. The user may also display their schedule for one individual day, or an entire month. A user's events are displayed in teal color.



Classes are a special type of recurrent event supported by this web app. To add a class, the user may use the course search bar in the menu sidebar on the left to search for their classes. Classes may be searched by course code or class title. When the search results are displayed, the user can click on the "+" to add the class section to their calendar as a recurring event for the duration of the semester.

To add an event manually, double click on the calendar. The user will be prompted to enter an event title ("Subject"), the start time and date, the end time and date, and which days of the week the event will recur. Clicking "DONE" will save the event to the calendar, while clicking "CANCEL" or the "X" at the top of the pop-up will cancel the creation of the event.



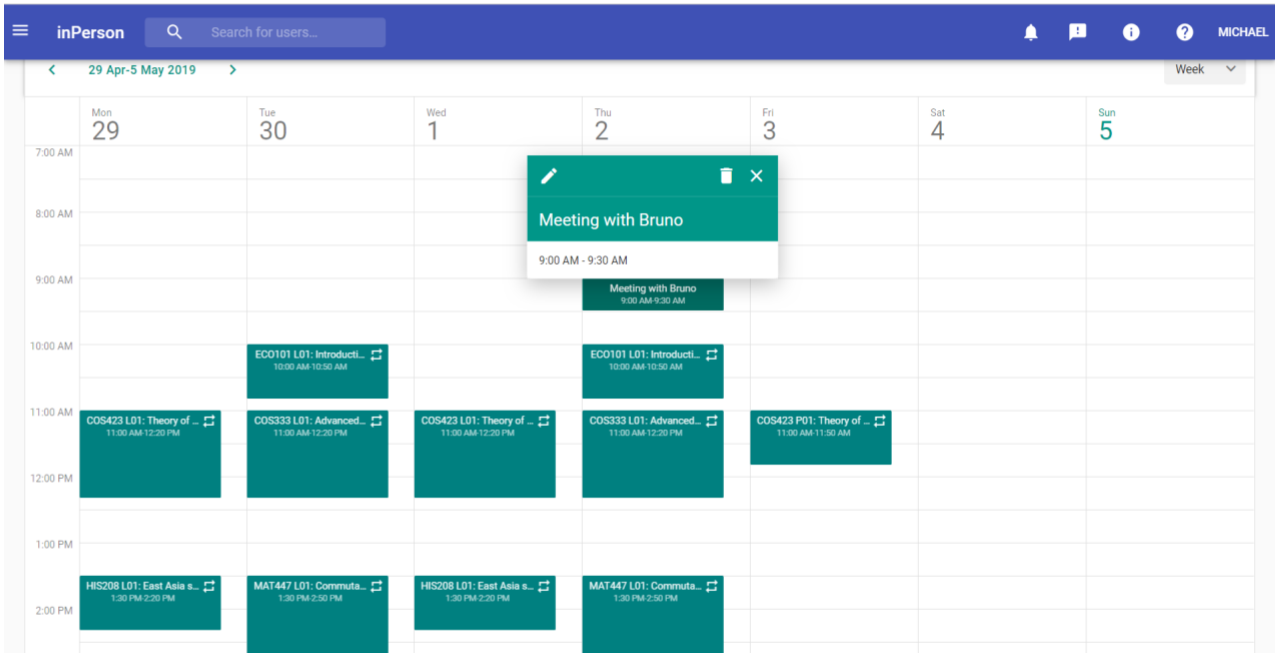
Subject

Start Date *: 5/2/2019, 9:00 AM

End Date *: 5/2/2019, 9:30 AM

Days: Select...

After creating an event, click on it to display full details. Events belonging to the user in question are displayed overlaid on the calendar according to the time they cover. If there are too many events overlapping at the same time to display at once, a circle with the number of undisplayed events will appear. Click on this circle to display a scrollable list of the additional events at that time. Currently, updating and editing events is not supported.


inPerson
Search for users...
MICHAEL
29 Apr-5 May 2019
Week
Mon 29
Tue 30
Wed 1
Thu 2
Fri 3
Sat 4
Sun 5
7:00 AM
8:00 AM
9:00 AM
10:00 AM
11:00 AM
12:00 PM
1:00 PM
2:00 PM
Meeting with Bruno
9:00 AM - 9:30 AM
Meeting with Bruno 9:00 AM-9:30 AM
ECO101 L01: Introducti... 10:00 AM-10:50 AM
ECO101 L01: Introducti... 10:00 AM-10:50 AM
COS423 L01: Theory of ... 11:00 AM-12:20 PM
COS333 L01: Advanced... 11:00 AM-12:20 PM
COS423 L01: Theory of ... 11:00 AM-12:20 PM
COS333 L01: Advanced... 11:00 AM-12:20 PM
COS423 P01: Theory of ... 11:00 AM-11:50 AM
HIS208 L01: East Asia s... 1:30 PM-2:20 PM
MAT447 L01: Commuta... 1:30 PM-2:50 PM
HIS208 L01: East Asia s... 1:30 PM-2:20 PM
MAT447 L01: Commuta... 1:30 PM-2:50 PM

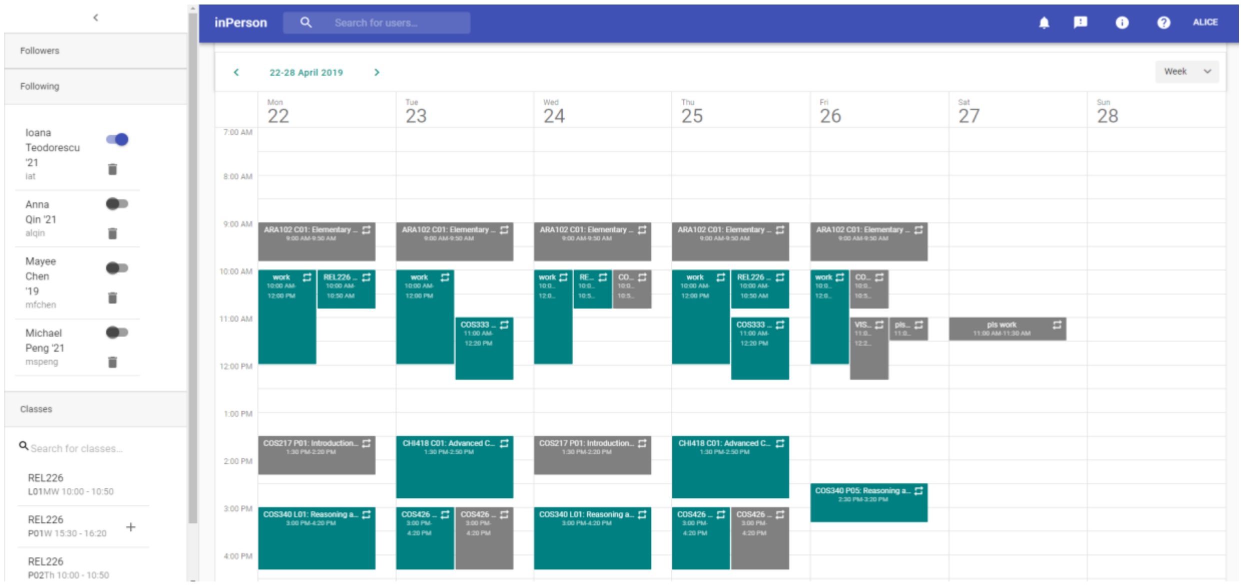
FOLLOWERS AND FOLLOWING

The key feature of this web app is its schedule-sharing functionality. User A may send a "follow request" to User B, and, if B accepts the request, A can now see B's schedule. Note that this is not bidirectional: B cannot see A's schedule unless B sends a follow request to A and A accepts it.

To follow another user, search for their name or netID using the user search bar in the header. Any user in the search results will have a button next to their name; click on that button to send them a follow request. Before the other user responds, the friend request is pending.

A user will receive a notification upon receiving a follow request. Clicking on the notifications button, they can see who has requested to follow them, and from there, they can either accept the follow request, at which point their schedule will be visible to this new follower, or reject the request.

Once a user has successfully followed another user, they will be listed in the sidebar under the "Following" section. To view one of these people's schedules, the user simply needs to click on switch by their name, and that person's events will be displayed on the calendar in gray.



Multiple schedules can be displayed on the calendar at one time. Click on the toggle switch again to remove those events from view. The user can click the delete button next to a name to stop following that person.

If a user no longer wishes to share their schedule with one of their followers, they can delete a follower by clicking the delete button next to the follower's name. If the follower wishes to see the user's schedule again later on, they would have to send another follow request.

III. DEVELOPER'S GUIDE

This web app uses React.js for the front end and Django REST Framework for the back end, with an Amazon RDS instance (hosted on AWS) running the PostgreSQL Database Engine as the database.

BACK END

On the back end, there are several different Django apps that form the core functionality of the web app as a whole.

FOLLOWREQUESTS

The followrequests app defines how users are able to interact with each other in the web app. The bulk of the followrequests app is derived from the django-friendship library, with modifications to ensure that following in the app is not bidirectional (i.e. user A can follow user B without user B necessarily following user A). It defines the following endpoints:

Endpoint	**Request Type**	**Input Data**	**Description**
/user/following	GET	None	Gets a list of users that the current user is following
/user/followers	GET	None	Gets a list of the current user's followers
/user/requests	GET	None	Gets a list of follow requests sent to the current user

/user/requests/sent	GET	None	Gets a list of follow requests that the current user sent
/follow/:userid	PUT	User ID	Sends a follow request to user with the given user ID
/user/requests/:userid	POST	User ID	Accepts a follow request (if it exists) from the user with the given user ID
/cancel/:userid	POST	User ID	Cancels a follow request (if it exists) from the current user to the user with the given user ID
/unfollow/:userid	DELETE	User ID	Unfollow the user with the given user ID
/remove/:userid	DELETE	User ID	Delete the follower with the given user ID

SCHEDULE

The schedule app defines three key models: Schedule, Section, and RecurrentEvent.

Schedule:

Field	**Purpose**
term	The term the schedule applies to (F2018, S2019, etc.)
owner	The user that the schedule belongs to

Section:

Field	**Purpose**
term	The term in which the class section takes place (ex. F2018, S2019)
class_number	Class number (ex. "333" for COS 333)
code	Department code (ex. COS, MAT, etc.)
catalog_number	Course catalog number
title	The title of the course
section	Lecture or precept section, (ex. P01, L01, etc.)
start_time	The start time of the class section each day
end_time	The ending time of the class section each day
days	The days of the week in which the class section takes place
location	The location of the class

RecurrentEvent:

Field	**Purpose**
title	Name or title of the event
start_time	Start time of the event
end_time	End time of the event
days	Days in which the event takes place
location	Location of the event
schedule	Schedule that the recurrent event belongs to

The Schedule model defines a schedule's relevant term, owner, and events on the schedule. The Section model contains all information relevant to a specific class section (i.e. a particular lecture or precept), such as the days of the week in which the section takes place and its starting and ending times during those days. The RecurrentEvent model characterizes events that take place on a regular basis at the same time every week and is the model into which courses are placed. This app also defines how to handle GET, PUT, and POST requests relating to schedules. In particular, it defines the following endpoints:

Endpoint	**Request Type**	**Input Data**	**Description**
/semester	GET	None	Get the start and end date of the semester
/classes/?search=	GET	Name of class	Searches for a class based on course code, course number, and title
/schedule/:userid	GET	User ID	Gets all recurrent events from the current schedule of the user with the given user ID
/events/user/	GET	None	Gets all recurrent events of a user
/events/:id	GET	Event ID	Get the recurrent event with the given ID
/events/:id/	PUT	Event ID	Updates the current user's recurrent event
/user/schedule/classes/	POST	None	Adds a class section to the user's calendar
/schedule/user	POST	None	Creates a new schedule
/events/user/	POST	Event title, start and end time, days of week	Creates a custom recurrent event for the current user
/schedule/user	DELETE	None	Deletes the current user's schedule
/events/:id/	DELETE	Event ID	Gets the event with the given ID

SCRAPING

The scraping script scrapes course data from OIT webfeeds, formatting it as a JSON file; cleans this data (into another JSON file) so that it may be loaded into the database; and adds the cleaned data to the database. The scraping code separates the courses based on section so each lecture, precept, drill, etc. of a class would be stored as a different model in the database. Furthermore, the routine also contains the script used to scrape data regarding undergraduate students from Tigerbook. Following the Tigerbook API, a 32-character key was taken which could then be used to access information about all the Princeton undergraduates. The relevant data was taken and loaded into a JSON file, then cleaned so it could be loaded into the database the same way as the course data. The purpose of loaded students into the database is so that when a user searches for friends, they can search for all existing users, not only those who have used the app. The plan is to scrape course data and student data information at the beginning of every semester and load it into the database.

USERS

The users app defines our custom user model, which is a subclass of AbstractUser. In addition to the existing fields in the default Django user model, our custom user model contains fields to represent a student's netID, class year, and university. This app defines how to handle GET requests pertaining to users, namely searching for a user with a given id (note that this is distinct from the user's netID) as well as getting a list of users who have a certain first name, last name, netID, or class year.

FRONT END

The front end uses React.js to render a one-page responsive web app that does not need to refresh in order to asynchronously read and write from the server. DevExtreme, a third-party app, was used to create to facilitate the creation of custom events and display events on the calendar.

PAGES

The app has three pages so far: the index page (/) and the landing page (/landing), which are each rendered by a separate template which is linked to a main React.js component, as well as an about page (/about). In the near future we plan to add a 404 page as well.

LANDING PAGE

The landing page has one button that redirects to the CAS website for authentication, and is only rendered if the user is not logged in.

INDEX PAGE

The index page has a main component (App.js) that fetches the user's information and stores it in its state. It also contains most of the functionality for sending requests to the API, in particular functionality related to follow requests, a user's followers and a user's following, and displaying events on the schedule. This component passes all the relevant information to its children components, i.e. the Navbar, the Menu (the drawer that contains the followers and following menus, as well as the searching/adding classes), and the Calendar (where the events are displayed).

NAVBAR

The Navbar is a component adapted from MaterialUI's Navbar, and it contains the Follow Requests display (which are being stored in the state), the user search bar (to look for users and send follow requests), the

Feedback Form link, the About page link, the Tutorial, and the User menu. For now, the user menu only displays the user's First Name and a button for logging out. In the future this menu will also contain the option to block users, since it was already implemented on the back end.

MENU

The Menu is displayed as a Drawer component and also adapted from MaterialUI. It contains three displays arranged in MaterialUI Expansion Panels: the Follower display, which lists all the followers and allows users to remove them (there is a Dialog popup that asks for confirmation first), the Following display, which allows for removal of following has a switch that toggles whether the followee's schedule should be displayed on top of the user's schedule, and a Classes display that enables easy adding of classes to the user's schedule. There is a Group Display as well, which has been implemented in the UI but is not yet implemented on the back end.

CALENDAR

The Calendar is rendered as a customized version of DevExtreme's React Scheduler component. The component keeps in its state the events (appointments) that need to be rendered at any point in time, and has functionality from converting appointments from the API's object structure to the structure required by the DevExtreme Scheduler and vice versa.

ABOUT

The About page is linked from the navbar. It uses Bootstrap, and contains an overview of our product, the image and names of the team, acknowledgements, and a link back to the index page.

FRONT END - BACK END INTEGRATION

To integrate the front end and back end, a Django view rendered the main html file (home.html) which was used by React.js. The buildpacks and node modules were then included in the main directory. AJAX requests were sent to get the data from the back end. Upon a GET request to the back end, the server pulls all the relevant data from the PostgreSQL database and returns it to the client as JSON. Making changes to a user's schedule and following other users involved POST, PUT, and DELETE requests to modify or create objects in the database. AJAX and React.js enables us to create a one-page responsive web app that does not need to refresh in order to asynchronously read and write from the server.

KNOWN ISSUES

The biggest issue that currently exists with the calendar is that when a user wants to see someone else's schedule, they must refresh the state by going to a different week in order for the other user's schedule to show up properly. A similar problem exists when adding events; the page needs to be fully refreshed in order for the event to display correctly. Although we spent a lot of time trying to fix these issues, we were unfortunately unable to.

Another issue is with updating events. The default option for custom events has more features than our web app supports, which we have not figured out how to remove yet. Therefore, we have disabled updating and editing custom events for the time being.

Because user data is scraped from Tigerbook, if a user has previously requested to be removed from Tigerbook, then they will be unable to use our app properly. We are currently considering a workaround for this that would involve a user being able to create their own profile so that they are searchable in this system.