

CS 30700

Design Document

**Team 34:**

* Charlie Schmidt
* Sriram Kiron
* Leon Yee
* Caleb Chua

# Index

[**Purpose 3**](#_jigi5shtx5rx)

[Functional Requirements 3](#_8qc46xcq2kt0)

[Non Functional Requirements 5](#_3n7lupdnl676)

[**Design Outline 6**](#_s9ipb5diikh)

[High Level Overview 6](#_fvtkhde5umw8)

[Sequence of Events Overview 7](#_1743zw2a8h6e)

[**Design Issues 8**](#_ryeyra6sz941)

[Functional Issues: 8](#_sn8dv57fqztp)

[Non Functional Issues 9](#_kyazsaf9mya9)

[**Design Details 11**](#_c8yb7d7k5379)

[Class Design 11](#_fs9dmqnfje9a)

[Descriptions of Classes and Interaction between Classes 11](#_l97qiqlsm72k)

[Sequence Diagram 13](#_pcvznmjzar7a)

[Navigation Flow Map 19](#_qninnw8ztvkf)

[UI Mockup 20](#_cc453rci0792)

# Purpose

There are many different places to park on the Purdue campus, and there are multiple types of permits you can have. Permissions for different permits change based on the day and time, and even if all of these details are figured out, parking can still be hard to find. And for those without permits, parking spots are especially scarce around campus.

Our app aims to make the parking process easy, allowing users to input what kind of permit they have (if any) and displaying all the closest available parking options on a map. Users are also provided with other parking-related information such as where most parking tickets are given out, and can opt to receive notifications for weather advisories, timed parking, etc. Anyone on campus can use Lighthouse to park quickly, confidently, and safely.

## Functional Requirements

1. User Account

As a user,

* 1. I want to register for an account using email and password
  2. I want to log into my account
  3. I want to reset my password if I forget it
  4. I want to be able to manage my account (email, password, etc.)
  5. I want to update my profile with specific vehicle information (permit type, compact car, electric car, etc)

1. Location Preferences

As a user,

* 1. I want the app to ask if it can track my location and track my location if I allow it

1. Parking Viewing - List View

As a user,

* 1. I want to be able to view all the Purdue parking options as a list view
  2. I want to be able to click into a list item and see the specific parking information

1. Parking Viewing - Map View

As a user,

* 1. I want to see a map of my surroundings
  2. I want the map to display available parking spots near my location
  3. I want the map to display parking spots based on my permit type and time of day
  4. I want the map to display changes in parking availability given football and basketball game days
  5. I want to see all the parking options on campus as icons on the map (having them bunch together through a heatmap as I zoom out)
  6. I want to be able to filter parking options and vehicle information, and only see parking options that fit my selected filter

1. Parking Viewing - Map Popup View

As a user,

* 1. Clicking on a specific parking option on the map should reveal a modal popup
  2. I want the popup to display parking garage information
  3. I want the popup to display changes in parking garage information given football and basketball game days

1. Parking Status

As a user,

* 1. I want to set my parking status
  2. I should be able to see whether a street parking spot is taken or not, and how long it has been taken for

1. Realtime Notifications

As a user,

* 1. I want a separate settings page to edit my preferences for alerts
  2. I want to receive alerts about parking time expiry, weather-related advisories, and game-day parking changes

1. Automatic Scheduling

As a user,

* 1. I want to be able to input my school schedule into the application
  2. I want the app to generate places and times I should park given my schedule

1. Navigation System

As a user,

* 1. I want to select a destination on the interactive map and get the directions to my selected destination (on the app or through an external app)
  2. I want to see all the closest parking spots to my selected destination

1. Parking Details

As a user,

* 1. I want to see where parking tickets are frequently given out
  2. If I get a parking ticket, I want the ability to input where and when it occurred
  3. I want to see where there is more/less parking availability
  4. I want to be able to input into the app if there are more/less parking availability

## Non Functional Requirements

1. Performance
   1. I would like the application to run smoothly without crashing.
   2. I would like the applications to be launched in under 5 seconds.
   3. I would like the application to support at least 15,000 users, to account for all vehicle owners in Purdue.
   4. I would like the application to support parking outside of Purdue
   5. As a developer, I would like to obtain parking information outside of Purdue automatically through web scraping
2. Server
   1. I would like the server to support real-time server-client communication
   2. I would like the server to be able to store users’ data in a database
3. Appearance

As a developer,

1. I would like to create a well-rounded and pleasant user interface without being too cluttered
2. (If time allows) I would like to allow users to customize their user interface (ex. dark mode)
3. Security

As a developer,

* 1. I would like to let the users choose whether to enable access to their precise location or not
  2. I would like all data within the database to be encrypted
  3. I would like to limit users to one account per individual to avoid botting or spamming the application

1. Usability

As a developer,

1. I would like the application to be friendly and self-explainable
2. I would like the application to be able to run on iOS and Android devices

# Design Outline

## High Level Overview

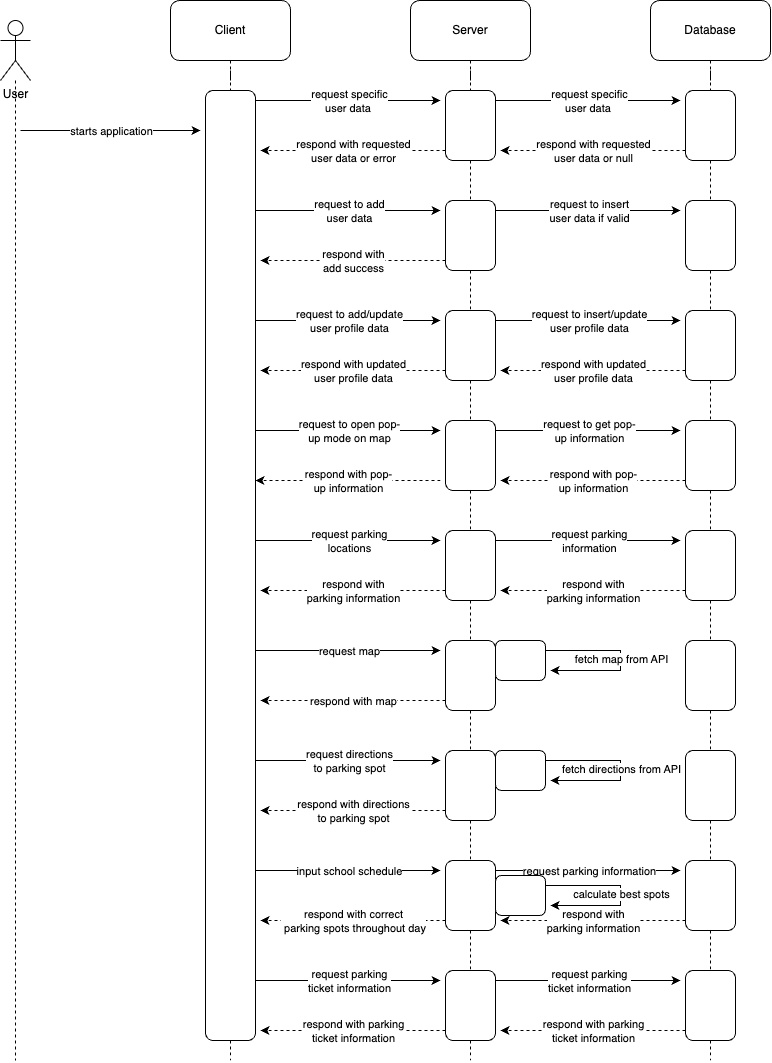
Our project will be a map application that allows users to see available parking in pins. Clicking on the pins, they will then be able to get directions to the parking. Users will also be able to input their class schedules and get a parking plan for the day that tells them what parking areas they should use and at what times. This application will use a client-server model, where the Node.js server will respond to requests sent by the client, and send queries to our Firebase database.



1. Client
   1. Provides an interface to the user
   2. Allows the user to make inputs
   3. Sends requests to the server to retrieve information that needs to be displayed, calculated, or updated
   4. Handles responses from the server and displays them to the user
2. Server
   1. Receives and handles requests from clients
   2. Keeps application running 24/7
   3. Communicates with database to provide specific information to client
   4. Performs all code that isn’t UI-centric
3. Database
   1. Non-relational database stores all data about users, parking, and events
   2. Responds to queries from the server, and fetches and transmits the appropriate data back to the server

## Sequence of Events Overview

The sequence of events diagram shows the interactions between the client, server, and database. The user only sees the front-end of the application, but not all of the code can be written in the front-end. The logic for functions such as calculating what parking locations are best is done in the server, and operations using data access the database.



# Design Issues

## Functional Issues:

1. What information do we need to sign up for an account?

* Option 1: Username and password only
* Option 2: Username, password, email address
* Option 3: Google account

Choice: Option 2

Justification: Usernames and passwords are essential when creating an account to identify users and for the protection of each account. Emails will be used to verify users when signing up for accounts, and will allow users to reset their password should they forget it. Signing up for an account using a Google account is a potential alternative to a username and password, however, some users may not have a Google account or simply prefer to not link their Google accounts when signing up for apps.

1. How should we display all the parking information to users?

* Option 1: Map view
* Option 2: List view
* Option 3: Map and list view

Choice: Option 3

Justification: Having a map view of all the parking options is helpful for users to see where spots are located and allow for easier navigation around campus. Having a list view of all parking options is more compact and allows users to easily browse the different parking availabilities around campus. By giving users both options, it enables them to choose what view is best suited for their circumstances.

1. When do we keep track of a user’s location?

* Option 1: Continuously, even when the user is not using the app
* Option 2: Only when the user is actively using the app
* Option 3: User manually enters their location information
* Option 4: Allow the user to choose between continuously or when active

Choice: Option 4

Justification: Our app uses a user’s location data to locate them on the map, search for nearby parking options, and to direct them to parking destinations. Having the user manually enter their location will be tedious for the user, as the app uses their location quite often. Allowing the user to decide between Option 1 and Option 2 will give them control over their location data, but will also allow for a better user experience.

1. How should we display details of parking spots in map view?

* Option 1: Center screen modal popup
* Option 2: Bottom screen modal popup
* Option 3: New page
* Option 4: Tooltip popup

Choice: Option 2

Justification: Having a bottom screen modal popup when a user clicks to view detailed information on a parking spot is the least invasive of all three. The bottom screen modal popup will keep navigation of the app easy while containing enough space for all related information. Having a new page means users will have to navigate back and forth, making the app less user-friendly. A tooltip popup will be easier for users to navigate, however, may not be big enough to fit all the related parking details in a user-friendly format. A center screen modal popup is similar to a bottom screen modal popup, however, covers a lot of the screen when in use.

1. How should we obtain information regarding frequent parking tickets or parking availability?

* Option 1: Crowdsourcing
* Option 2: Web scraping
* Option 3: Manually enter information

Choice: Option 1

Justification: Crowdsourcing information regarding where parking tickets are frequently handed out and general parking availability in certain areas is the most reliable option. Crowdsourcing also allows the app to update quickly given information, notifying other users of parking tickets and/or parking availability in the most timely manner.

## Non Functional Issues

1. What mobile app development language/framework should we use?

* Option 1: React Native
* Option 2: Swift
* Option 3: Flutter
* Option 4: Dart

Choice: Option 1

Justification: React Native is easier to pick up for those with prior experience with React, and has a lot of documentation and resources available for developers. React Native also allows our app to run on both iOS and Android (unlike Swift, for example).

1. What backend language/framework should we use?

* Option 1: Node.js (JavaScript)
* Option 2: PHP
* Option 3: Spring Boot
* Option 4: Django

Choice: Option 1

Justification: Node.js is one of the most popular combinations to choose with React Native. This provides a lot of documentation online for us to utilize, and many problems we may encounter could be found online. Additionally, Node.js is very speed-efficient and scalable, which works well in our favor.

1. What database should we use?

* Option 1: MySQL
* Option 2: Firebase
* Option 3: MongoDB
* Option 4: Redis

Choice: Option 2

Justification: Firebase is a cloud database, and is also very easy to scale to large numbers of users. Firebase also pairs well with React Native, as it has a simple UI for database management. The security of Firebase is very well-rounded, providing many features to secure all of our user’s data.

1. Which API should we use to access map and location data?

* Option 1: Google Maps (redirect)
* Option 2: React Native Maps
* Option 3: Leaflet

Choice: Option 2

Justification: The best option is [React Native Maps](https://www.npmjs.com/package/react-native-maps). There seems to be a large amount of documentation provided by the maintainer, which will be very useful for us. The other options aren’t appealing enough, especially because React Native Maps has many features that match our exact needs.

1. What API should we use to request user location?

* Option 1: Google Maps API
* Option 2: React Native Geolocation

Choice: Option 2

Justification: We considered Google Maps API, but it doesn’t seem to mesh well with React Native applications. After a few online searches, there doesn’t seem to be much information about this. [React Native Geolocation](https://www.npmjs.com/package/@react-native-community/geolocation) API seems to offer a simple and efficient way to request and use the user’s location.

# Design Details

## Class Design

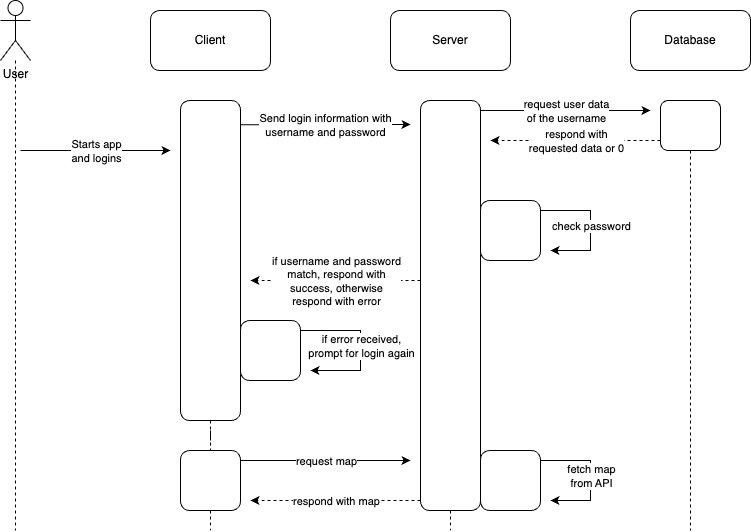
### Descriptions of Classes and Interaction between Classes

* User
  + User object is created when someone signs up in the app
  + Contains all information about the user (including the user’s car type)
* Schedule
  + Schedule object is created when someone inputs their schedule to the app
  + Contains a user’s schedule, stored as a 2-dimensional List of classes (one List for each day of the week)
  + Functionality to import the schedule as a screenshot of their week-at-a-glance
* Class
  + Class object is created when someone adds a class to their schedule in the app
  + Contains information about a specific class in the user’s schedule
* Permit (enum)
  + All the possible types of permit a user can have
* ParkingLot
  + ParkingLot object is created by maintainers when adding information for individual parking lots
  + Contains information about each parking lot, including how many current users are parked and how many tickets have been reported there (as well as the permit permissions for the lot)
* Permissions
  + Permissions object is created when a ParkingLot object is created
  + Contains all the permissions for a ParkingLot, stored as a separate array of booleans for each permit. The array is of size 24, and each element corresponds to whether a User with that permit is allowed to park at that hour of the day
* Address
  + Address object is created when any location information is added to the app
  + Stores all the information for an address. This is used by any class that stores a location.
* Event
  + Event object is created by maintainers to add events happening around campus that affect parking
  + Stores information about an event that is present on campus and will be displayed on the map
* Campus
  + Campus object is created by maintainers when adding parking information to the app for different college campuses
  + Each Campus object contains an array of ParkingLot objects
  + Stores information about a specific college campus, including all the ParkingLots and Events on that campus
* Alert
  + Alert object is created when a user’s parking time is close to expiring, weather advisories have been sent out, or if a user must move their car from the area they parked in
  + This object will be used for sending parking related notifications to users (according to the alert preferences of each user)

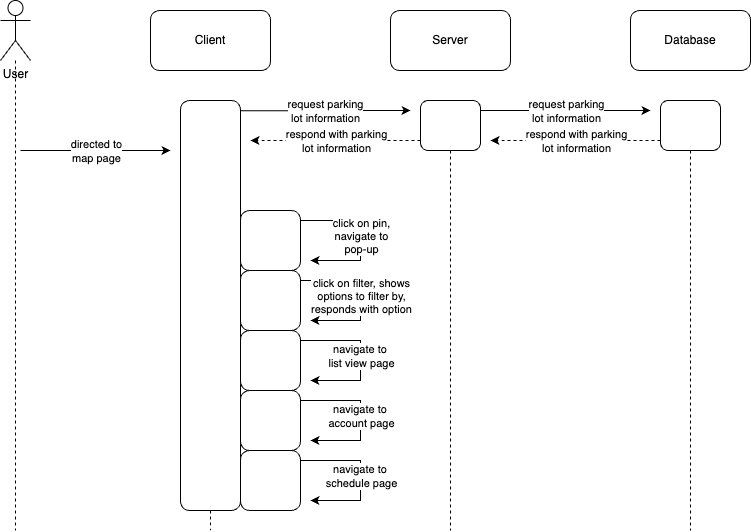
## Sequence Diagram

The following diagrams show the steps for the major functions of the application, including logging in, the map page, the pop-ups, the list view page, the account page, and the schedule page. The client interacts with the server, and the server interacts with the database. In some steps, data from the database is carried over to the client and saved there, preventing excessive trips from client to server to database back to server back to client.

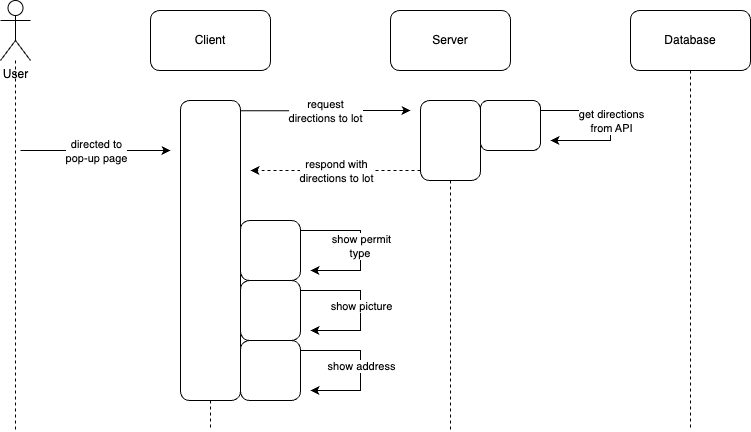
1. Sequence of events when [user logs in]



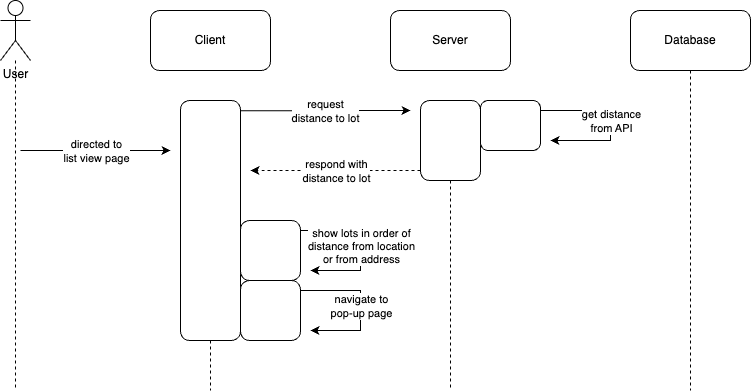
1. Sequence of events when [user is on map page]



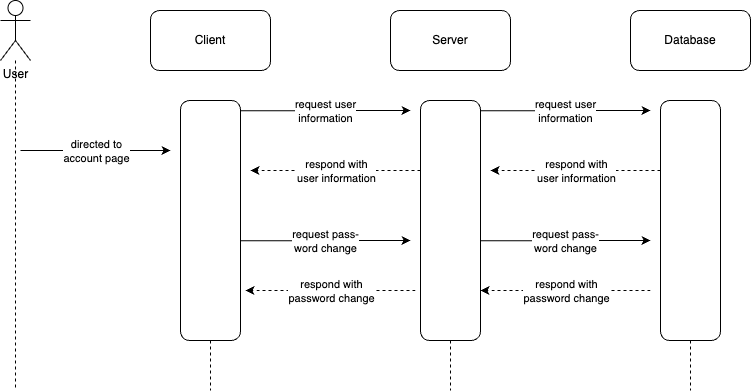
1. Sequence of events when [user is on pop-up]



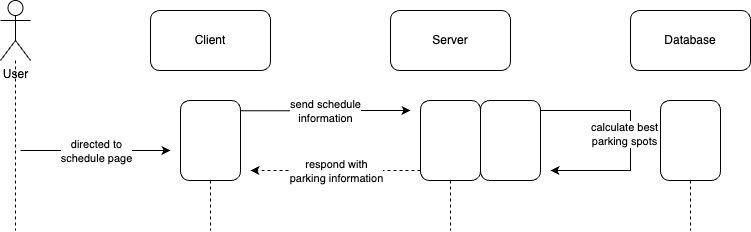
1. Sequence of events when [user is on list view page]



1. Sequence of events when [user is on account page]

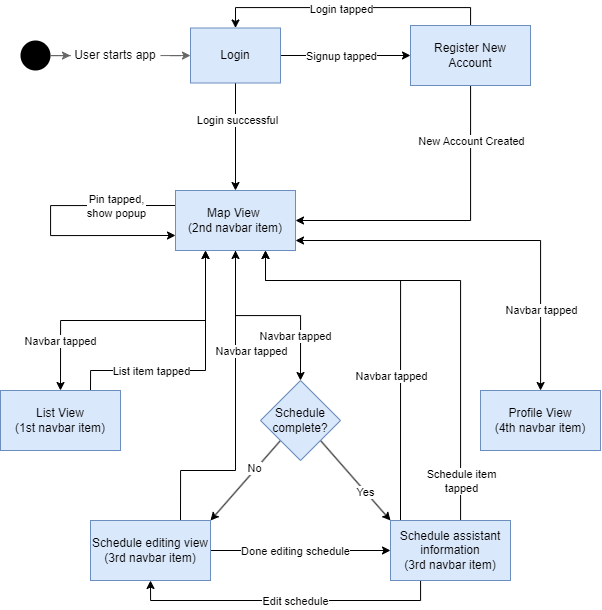


1. Sequence of events when [user is on schedule page]



## Navigation Flow Map

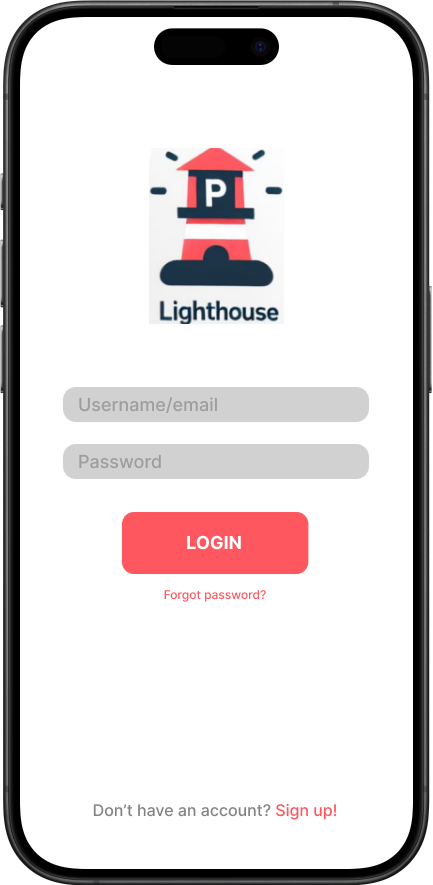
Our navigation design mostly surrounds the navigation bar at the bottom of the app, providing ease of access for all our users. As the user starts the application, the user will have to either log in or sign up with their user information. They will then be presented with the main view of the map, and are able to navigate to any other page using the navbar. Depending on whether the user has created a schedule or not, the 3rd navbar button will direct them to a different page.



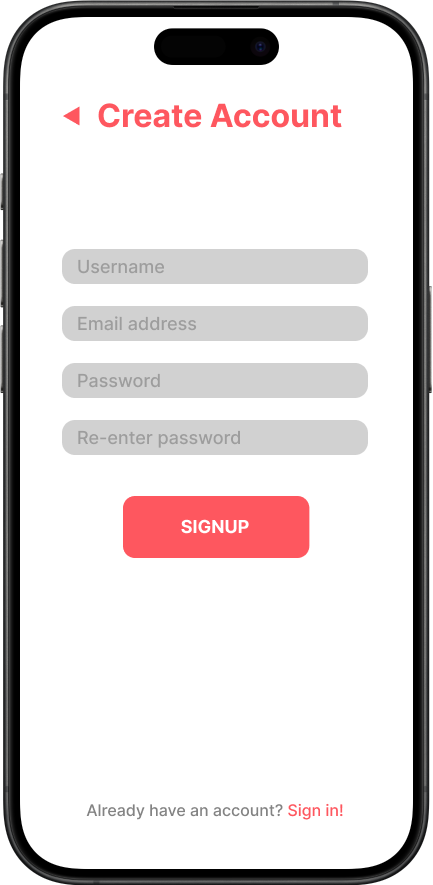
Our navigation flow map clearly indicates how all pages and views revolve around the navigation bar. User experience is prioritized.

## UI Mockup

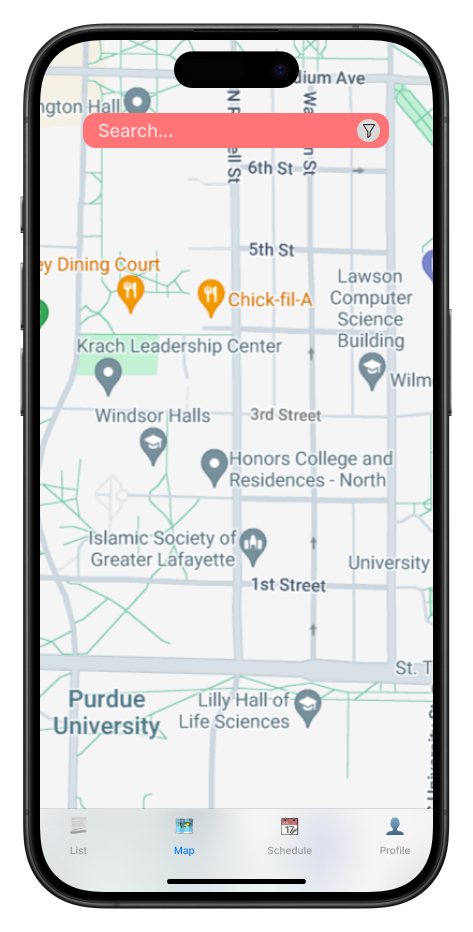
Our UI Mockup of Lighthouse is designed to be simple, yet efficient for any user to navigate. The main idea of our app revolves around the home page and the navigation bar, where the user can travel to the List view, Map view, Scheduling assistant, and Profile page. All the related functions are on the same page. The graphics on the mockup are temporary placeholders and are subject to change.



This is the login page. It allows the user to input their username or email address and a password to log in. It also includes a “forgot password” button, and a “sign up” button for new users.

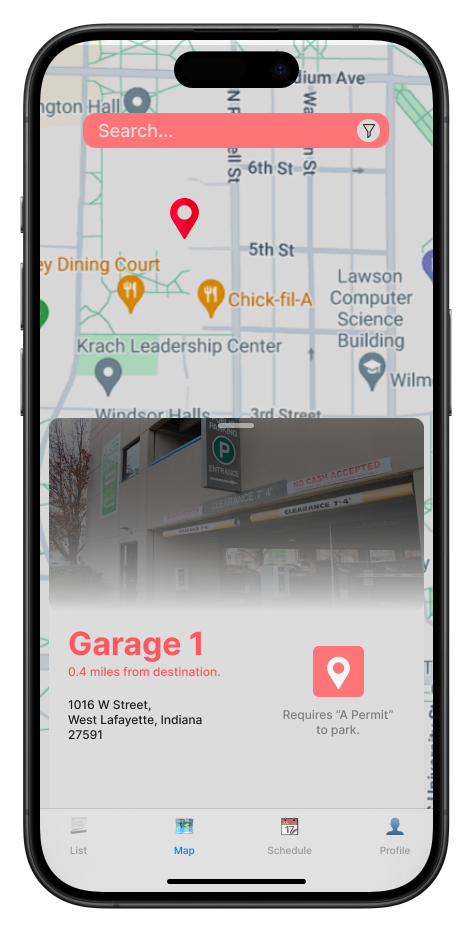


This is the signup page. It prompts the user to input a username, email address, and password for account creation. It also has another field for the user to re-enter their password for confirmation. There is also a button to go back to the login page.



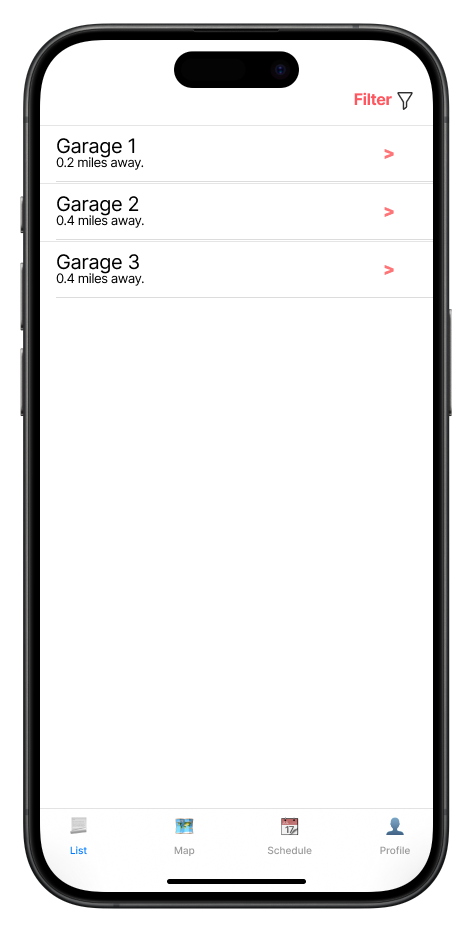
This is the main homepage of the application. The main view is a map of the user’s surrounding area. The map is completely interactive, allowing the user to move, rotate, scale, etc. The map will also be populated with pins for each parking spot, including street parking. The navbar at the bottom allows the user to switch between the 4 main pages.

The search function allows the user to search for their destination, and the map will automatically focus on that destination. Pins would then pop up in the surrounding area, signifying that there is parking nearby.



Clicking on a pin on the map will bring up the popup, which contains specific information about the parking location. It will have an image of the area, the name of the street or parking garage, the distance and address of the parking spot, and a button to start navigation to the parking spot.

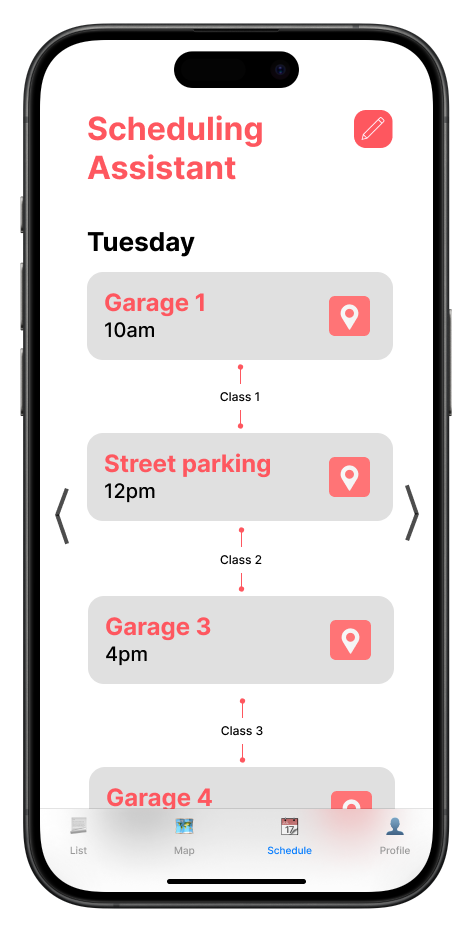
There is also additional information on which permit is required to park there.

This is the list view of the application, navigated to by the navbar. The list will show the nearest parking areas to the user until the user has inputted a location through the search bar. Then, the list will show the closest parking spots to that address. Clicking on a parking option will pull the map back up, focusing on that parking option and showing the popup.

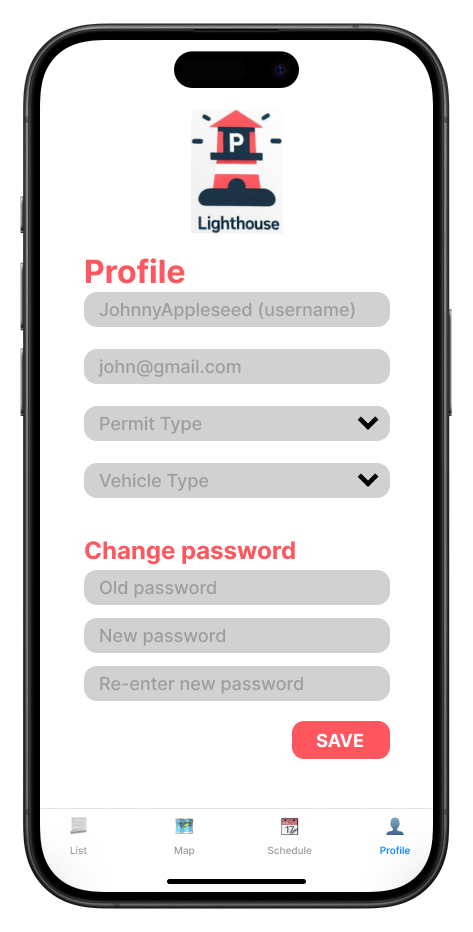
There are also ways to filter, including the cost, distance, permit, and type of parking.



The 3rd tab of the application is the Scheduling Assistant. This page allows the user to input their classes, with the following information: address, start time, end time, and dates. The arrows next to the text boxes signify a wheel picker or multi-select. The plus button allows the user to add additional classes, and the “Done” button moves the user to the next page. There is also an edit button next to the Class name, which allows them to change it.



After the user presses “Done”, they are transported to this page. Our application will automatically find the best spots to park, and display them in a list. Each card has the name of the parking lot, the time of departure, and a button to bring up the parking location on the map. Each card is divided by the class name they inputted for a better user experience. To modify their schedule, the top-right button brings them back to the previous page.



Finally, the 4th tab in the navigation bar brings them to the profile page. Each field is modifiable: username, email, permit type, and vehicle type. Again, the arrows in the text box signify a dropdown.

To change a password, the user has to enter their old password and re-type their new password twice for confirmation. If they change any fields, the “Save” button updates their data within the fields.