The CS461 Guide to the Galaxy

A theme park navigation app

Team 1
Team Members:
Kwek Sin Yee (01388276)
Lindy Lim Li Wen (01387183)
Poo Jing Fei Nikki (01390645)
Tan Jia Min Jasmine (01391479)

Introduction

Guide to Galaxy is a mobile application that is catered towards theme park goers, to provide them with a better experience at the park by allowing them to navigate through the crowded park easily and smoothly. The application uses Bluetooth Low Energy (BLE) to detect beacons placed in prominent locations around the park such as rides, restaurants, carts, souvenir shops or restrooms to provide users with real time crowd levels in each vicinity. These locations are marked out in a map to allow users to navigate around the park. Users would also be able to queue virtually for rides, saving hours waiting in a line and better utilising their time at the park.

Background and Motivation

Theme and amusement parks have always been a popular attraction for locals and foreigners. This constantly creates a big problem, large crowds! From queuing for a popular ride, to enjoying instagrammable food in restaurants, or taking photos at an iconic statue, a sea of people is seen everywhere and is hard to avoid. This wastes a lot of time which is a pain to those who wish to explore all rides and attractions in a day, especially foreigners in a tour group with limited time to spare. Hence, navigating around the park and reducing waiting time for rides and attractions becomes a top priority to theme park visitors, fuelling the need for our application. By sparing waiting time to queue for rides, visitors could use the time to take memorable photos or shop at the souvenir shops to have a piece of the park to takeaway at the end of the day.

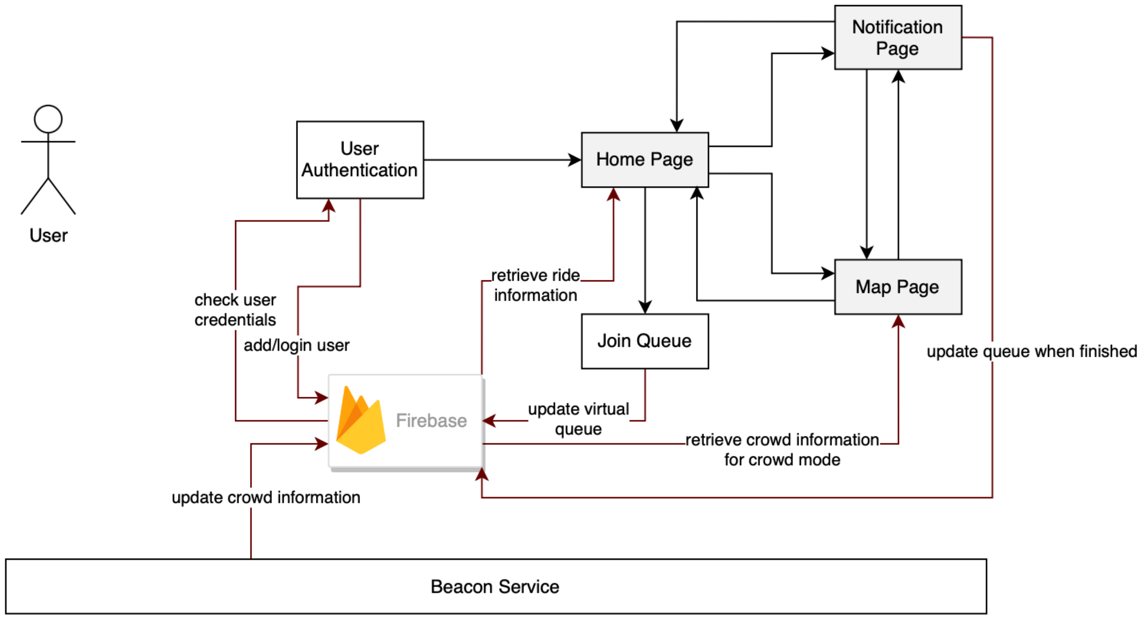
Objective of Our App

Our app aims to effectively mitigate crowd levels by showing users a heatmap of the parks, allowing them to decide to navigate away from massive crowds or join events that are attracting the crowd such as hourly performances. Not only that, users would be able to view and queue virtually for rides and attractions, avoiding the need to stand in line for an hour or two for popular rides. This gives our users a lot more time to venture around the park and be able to enjoy more of what is offered by the theme park.

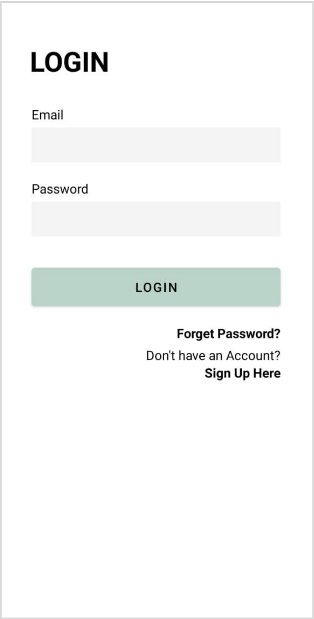
Not only would this be applicable to rides, but if we place beacons around areas such as restrooms, users would be able to search for a restroom with a shorter queue instead of having to wait in line for that as well.

Implementation Details

System Design Overview



Frontend & Design



LOGIN

Email

Password

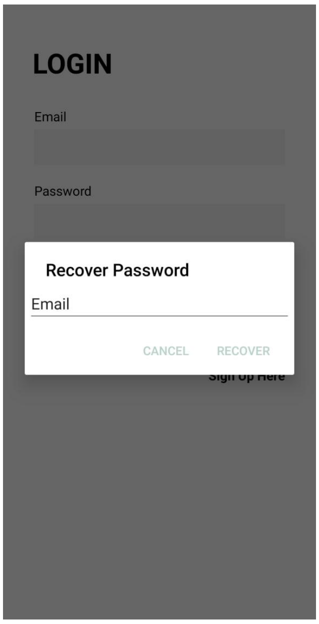
LOGIN

Forget Password?

Don't have an Account?

Sign Up Here

Sign In Page



LOGIN

Email

Password

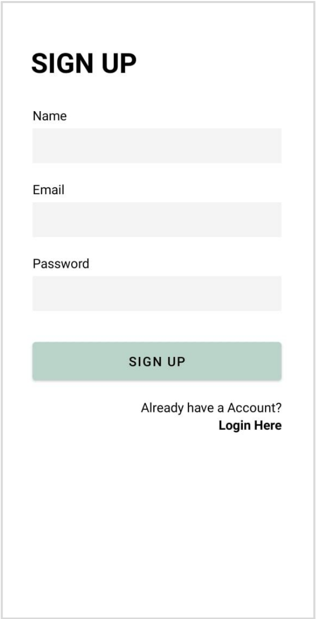
Recover Password

Email

CANCEL RECOVER

Sign Up Here

Forget Password



SIGN UP

Name

Email

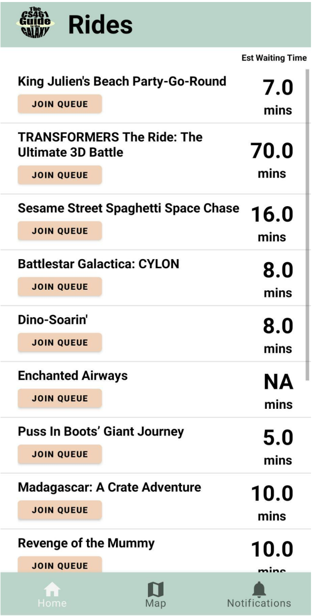
Password

SIGN UP

Already have a Account?

Login Here

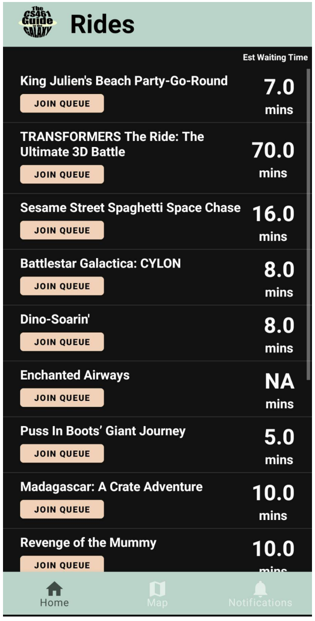
Sign Up Page



Rides

Est Waiting Time

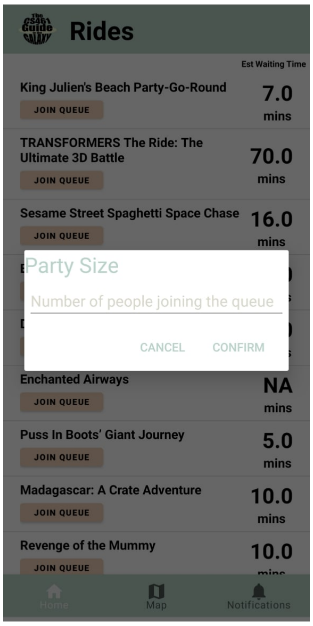
King Julien's Beach Party-Go-Round JOIN QUEUE	7.0 mins
TRANSFORMERS The Ride: The Ultimate 3D Battle JOIN QUEUE	70.0 mins
Sesame Street Spaghetti Space Chase JOIN QUEUE	16.0 mins
Battlestar Galactica: CYLON JOIN QUEUE	8.0 mins
Dino-Soarin' JOIN QUEUE	8.0 mins
Enchanted Airways JOIN QUEUE	NA mins
Puss In Boots' Giant Journey JOIN QUEUE	5.0 mins
Madagascar: A Crate Adventure JOIN QUEUE	10.0 mins
Revenge of the Mummy JOIN QUEUE	10.0 mins



Rides

Est Waiting Time

King Julien's Beach Party-Go-Round JOIN QUEUE	7.0 mins
TRANSFORMERS The Ride: The Ultimate 3D Battle JOIN QUEUE	70.0 mins
Sesame Street Spaghetti Space Chase JOIN QUEUE	16.0 mins
Battlestar Galactica: CYLON JOIN QUEUE	8.0 mins
Dino-Soarin' JOIN QUEUE	8.0 mins
Enchanted Airways JOIN QUEUE	NA mins
Puss In Boots' Giant Journey JOIN QUEUE	5.0 mins
Madagascar: A Crate Adventure JOIN QUEUE	10.0 mins
Revenge of the Mummy JOIN QUEUE	10.0 mins



Rides

Est Waiting Time

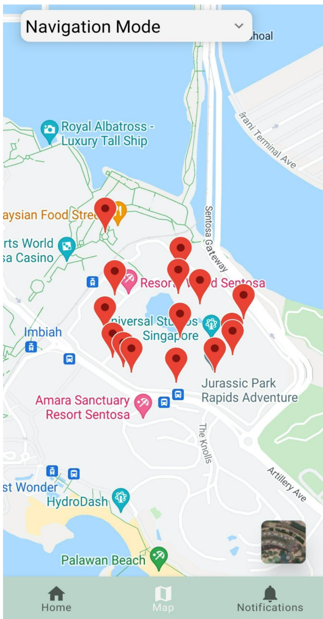
King Julien's Beach Party-Go-Round JOIN QUEUE	7.0 mins
TRANSFORMERS The Ride: The Ultimate 3D Battle JOIN QUEUE	70.0 mins
Sesame Street Spaghetti Space Chase JOIN QUEUE	16.0 mins
Enchanted Airways JOIN QUEUE	NA mins
Puss In Boots' Giant Journey JOIN QUEUE	5.0 mins
Madagascar: A Crate Adventure JOIN QUEUE	10.0 mins
Revenge of the Mummy JOIN QUEUE	10.0 mins

Party Size

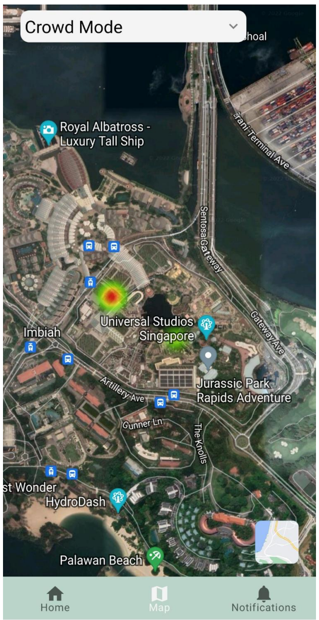
Number of people joining the queue

CANCEL CONFIRM

Home Page Home Page (Dark) Party Size

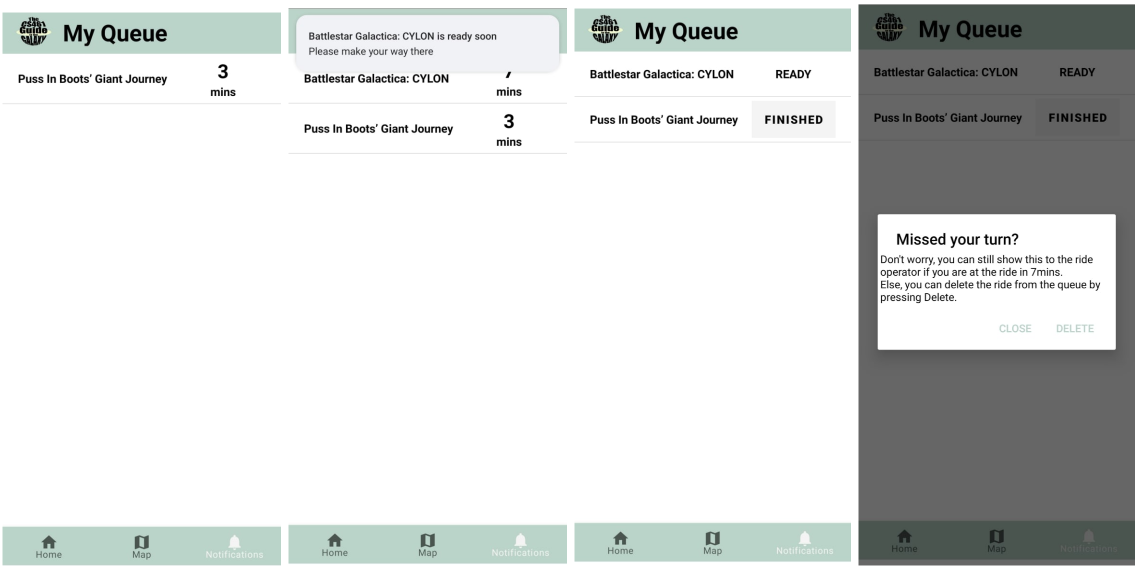


Navigation Mode



Crowd Mode

Map Page (1) Map Page (2)


My Queue

Puss In Boots' Giant Journey 3 mins

Battlestar Galactica: CYLON is ready soon
Please make your way there

Battlestar Galactica: CYLON 7 mins

Puss In Boots' Giant Journey 3 mins

My Queue

Battlestar Galactica: CYLON READY

Puss In Boots' Giant Journey FINISHED

My Queue

Battlestar Galactica: CYLON READY

Puss In Boots' Giant Journey FINISHED

Missed your turn?

Don't worry, you can still show this to the ride operator if you are at the ride in 7mins. Else, you can delete the ride from the queue by pressing Delete.

Notification Pages

Sensors and Libraries

Sensors	**Usage**
Bluetooth Beacons	To sense crowd levels at different locations of the park
GPS	Retrieve user location

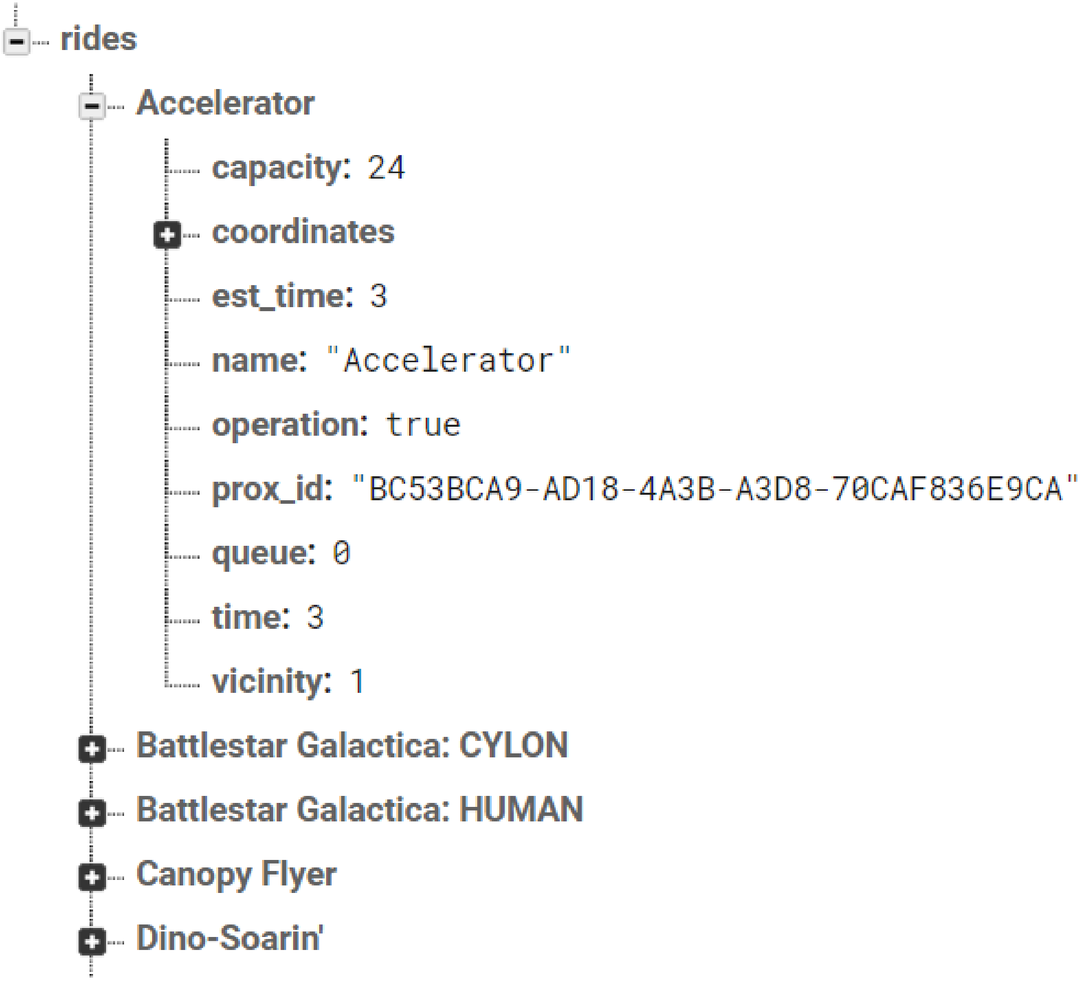
Libraries	**Usage**
Firebase: Realtime Database	To read and write data to Firebase Realtime Database
Firebase: Authentication	To authenticate and manager users
Google Play Services: Location, Maps	To display user location on the map and showing heat map of crowd levels
Android Beacon Library	To detect surrounding Bluetooth Beacons

Firebase

We used Firebase Realtime Database as the cloud storage that stores information about the park's rides and users. Data in this database is stored as a large JSON tree and information of each ride or user are stored as key-value pairs.

For each ride, the following attributes are stored:

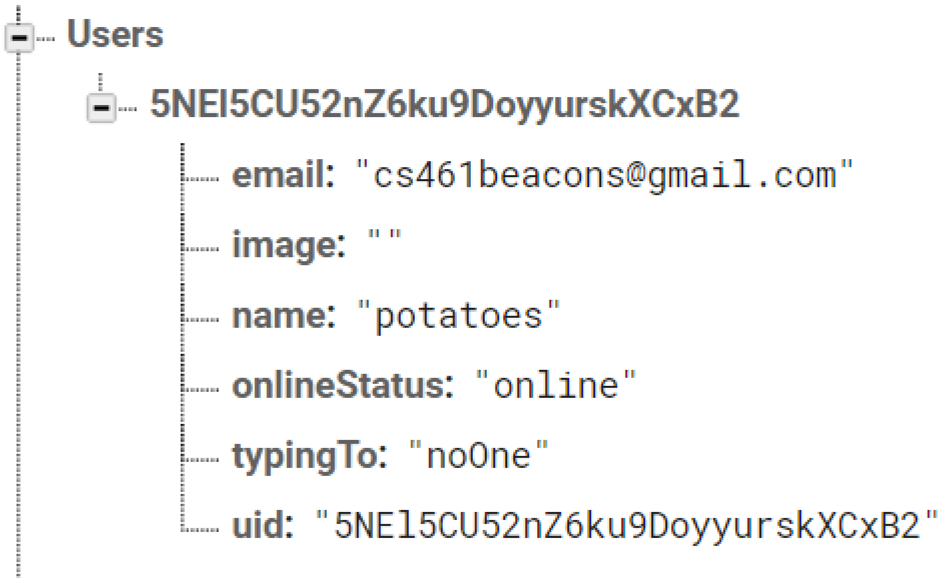
- “name”: name of ride
- “capacity”: how many people ride the ride at one time
- “coordinates”: the longitude and latitude information of where the ride is located
- “est_time”: Estimated waiting time for the ride
- “operation”: whether the ride is currently operating
- “prox_id”: the id of the Bluetooth Beacon that is attached at the ride’s location
- “queue”: total number of people currently in the queue
- “time”: the duration of the ride
- “vicinity”: the number of people sensed in the area by the beacon service



```
rides
    Accelerator
        capacity: 24
        coordinates
        est_time: 3
        name: "Accelerator"
        operation: true
        prox_id: "BC53BCA9-AD18-4A3B-A3D8-70CAF836E9CA"
        queue: 0
        time: 3
        vicinity: 1
    Battlestar Galactica: CYLON
    Battlestar Galactica: HUMAN
    Canopy Flyer
    Dino-Soarin'
```

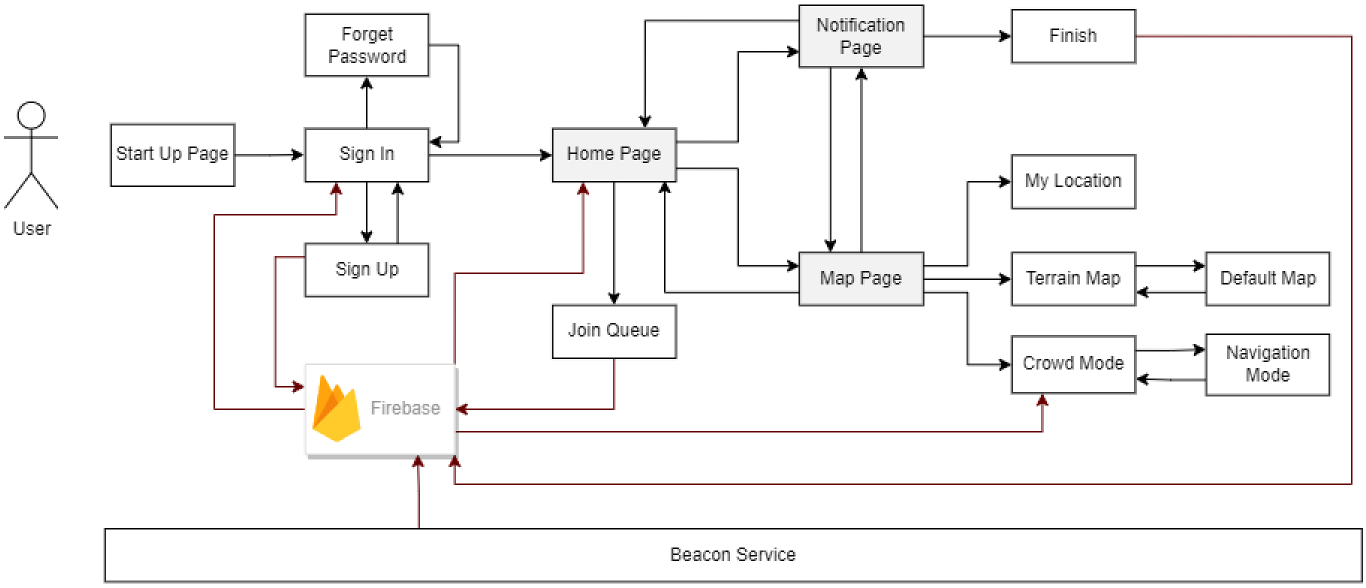
For each user, the following attributes are stored with their unique userID generated by Firebase Authentication as the key:

- "email": email address used to register the user
- "image": profile picture of the user (if any)
- "name": name of the user
- "onlineStatus": if user is online
- "typingTo": nil
- "uid": unique id of user.



```
Users
    5NEI5CU52nZ6ku9DoyyurskXCxB2
        email: "cs461beacons@gmail.com"
        image: ""
        name: "potatoes"
        onlineStatus: "online"
        typingTo: "noOne"
        uid: "5NEI5CU52nZ6ku9DoyyurskXCxB2"
```

Application Flow



<u>Sign In/ Sign Up</u>

When the user first opens our app, a splash screen is displayed to give buffer time for Firebase Authentication to find if there is an authenticated user in the app. If there is, the user is directed to the "Home Page". Else, the user is directed to the "Sign In" page where they can navigate to the "Sign Up" page should they not have an existing account through the "Sign Up" Button. The default authentication method is to enter the user email and password used to register an account. If users forget their password, they can recover their account using the “Forget Password” button where a reset password email will be sent to the email given.

<u>Home Page</u>

After the user is authenticated, they will first see the “Home Page”, where they can see the list of rides in the theme park and their respective estimated waiting times. If a ride is non-functional, the estimated time shown would be NA (Not Applicable) and users would not be able to join the queue for the ride.

Each ride has a “Join Queue” button which the user can click to join the virtual queue. When the join queue button is pressed, the user would be prompted to enter the number of people that will be joining the queue. This facilitates the process of queuing for visitors in groups as only one device is required per group to queue for rides. It also provides flexibility to enqueue groups of visitors into the queue as some members of the group might want to sit out of a ride.

Once the user confirms to join the queue, a method is called to calculate the new estimated time and number of people queued for the ride and the values are updated to our Firebase database. The app also stores the details of the ride in a file in the app's native memory which is used for updating the estimated waiting time displayed in the app as the queue moves forward. The app will also schedule 2 notifications to remind the user of the queued ride. The first notification will be prompted at 15 minutes from the time to onboard the ride or when the ride is queued if the estimated time is less than 15 minutes. This gives users of the app buffer time to navigate to the ride when it's nearing their turn to board the ride. The second notification is sent exactly at the time of the ride to notify users that their ride is ready. When users click on the notification, the app will be resumed or on the foreground and the "Notification Page" will be displayed.

Notification Page

The waiting time for all rides that the user is queued for will be displayed on the “Notification Page” and the user can switch to the “Notification Page” by clicking on the bell icon in the Navigation Bar at the bottom of the screen.

The estimated time would change to “READY” when the ride is ready to be boarded. The app would also decrease the queue number for the ride stored in our Firebase storage to reflect that the queue has moved.

Five minutes after the ride has started, the status of the ride would change to “FINISHED” to indicate that the ride is completed. If the user has not boarded the ride, the user can show the ride operator the "FINISHED" status within 15 minutes from the update of the "FINISHED" status. We choose a 15 minute buffer as we believe it would be sufficient for visitors to navigate from one end of the park to another within the buffered time. After 15 minutes, the ride will be deleted from the file in the app's native storage which will be reflected when the notification page is refreshed at the next minute. After this, users can queue for the ride again. This ensures that users are not less able to abuse our queuing function to hog a ride by queuing for the ride multiple times at one time.

Map Page

Users can navigate to the “Map Page” by clicking on the map icon on the Navigation Bar at the bottom of the screen. By default, users will be shown a Google Maps view zoomed in to Universal Studios Singapore (USS). The map will be populated with markers representing all rides in USS. When users click on any marker, it will display the estimated waiting time of the selected ride.

If users enable our app to access their location information through the settings of their devices or when prompted at the start of our app, a button will appear at the top right corner of the Map view. Users can click on the button to be directed to their own current location on the Map.

When users click on the drop-down menu at the top of the Map view, they can switch to the Crowd Mode, where the markers will clear and a heatmap is shown instead. The map will visualise groups of crowds around the park and indicate its density based on information collected by the Beacon Service using the colours red and green. Red areas will indicate a higher density of a crowd while green areas indicate a lower density of a crowd.

If users want to change the default map view displayed on the screen to the Terrain view, they can click on the mini-map image on the bottom right corner of the map.

Beacon Sensing Service

Our application uses the iBeacon protocol to broadcast and receive bluetooth signals, used to estimate the crowd density at a particular location. While the original plan was to make use of Estimote beacons, the beacons we borrowed were found to be malfunctioning, hence we simulated the iBeacon signals from iPhones instead. To simulate a beacon at a particular location, we input the following: UUID (alphanumeric, randomly generated and associated with a location), major value (fixed to 16808), and minor value (fixed to 19400). The Beacon Service stores a hashmap of these key-value pairs which are obtained from a text file.

Permissions to use Location, Bluetooth and Bluetooth Low Energy are requested in the MainActivity and AndroidManifest files in order to use this service. Users would need to both grant these permissions and own a device with Bluetooth functionality.

We used the Android Beacon Library to allow the app to detect beacons. The BeaconParser provided by the library was configured to detect only iBeacons, and the BeaconManager was set to range in all regions. Scanning all regions means every iBeacon in range will be detected. Technically, each region could be specified with the UUID associated with each ride, but to set up a single region for each ride would be tedious and have low scalability (ie. to add more attractions or locations). This is also why we have opted to use ranging instead of monitoring. Ranging picks up beacons in range, which we can filter and process, whereas monitoring is restricted to certain regions and detects events like entering and exiting the region. Monitoring would require the aforementioned non-scalable implementation of regions.

Each scan is 1.1 seconds long and the time between scan is set to 0. These values can be changed to be more battery-optimal, and to better suit the use case, but are left as such for demonstration purposes (to avoid waiting). In practical use, scans could be around 2 - 10 seconds long, to quickly determine where the user is. The intervals between scans could be around 5 - 10 minutes, since users are unlikely to switch between two locations too frequently.

The following are sample setups, their corresponding simulated situations, and the expected database behaviours.

1) Two iBeacon simulators are set apart, and the phone running the app moves between them. Simulates a park guest's movement between two attractions.
   Expected database behaviour: Increment vicinity count for the closer attraction, and decrement for the further one.
   Complexity: O(n), n = no. of ranged beacons detected

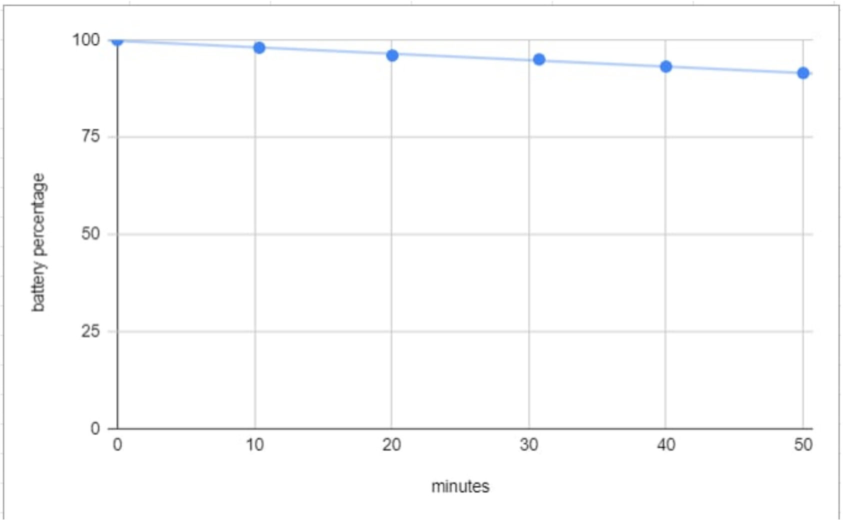
2) All iBeacon simulators are turned off. Simulates a park guest who has already left the park.
   Expected database behaviour: Decrement vicinity count for the last attraction the guest was at, or do nothing if there was no last attraction.
   Complexity: O(1)

3) iBeacon simulators (one or more) are set up but with different UUIDs than the attractions'. Simulates a park guest who has already left the park but is in range of iBeacons from other devices.
   Expected database behaviour: Decrement vicinity count for the last attraction the guest was at, or do nothing if there was no last attraction.
   Complexity: O(n), n = no. of ranged beacons detected

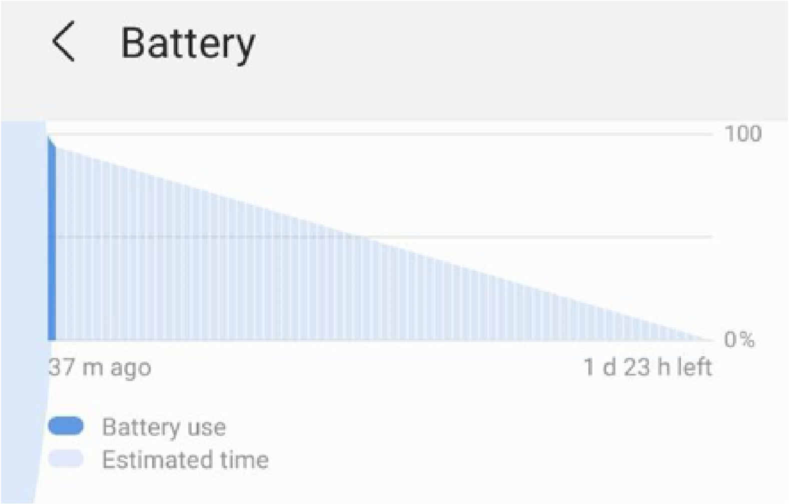
4) iBeacon simulators are on. The app is killed (swipe up). Simulates a park guest killing the app on their phone (eg. to save battery).
   Expected database behaviour: Decrement vicinity count for the last attraction the guest was at, or do nothing if there was no last attraction.
   Complexity: O(1)

Battery Performance

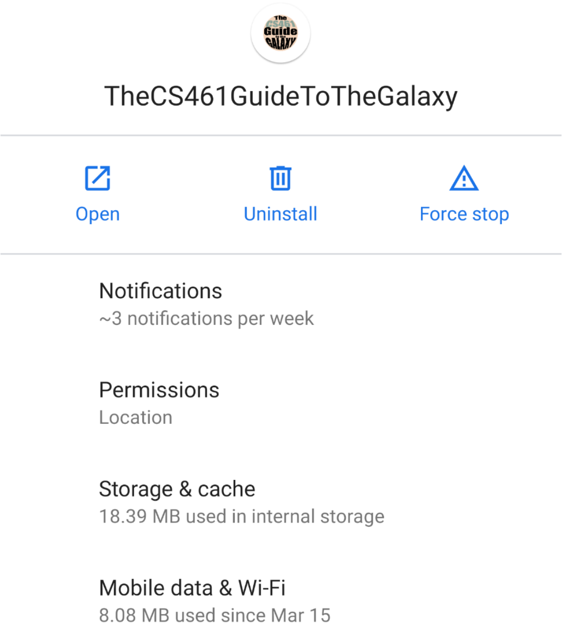
Manual measurements over 30 minutes from full charge, extrapolated to 50 minutes:



Battery usage graph on device:



Memory Usage


TheCS461GuideToTheGalaxy

Notifications
~3 notifications per week

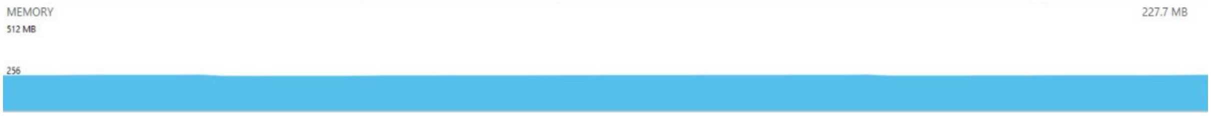
Permissions
Location

Storage & cache
18.39 MB used in internal storage

Mobile data & Wi-Fi
8.08 MB used since Mar 15

Internal storage: 18.39 MB
RAM used during normal processes: Relatively constant at 230 - 280MB
Mobile data / Wifi: 8.08 MB

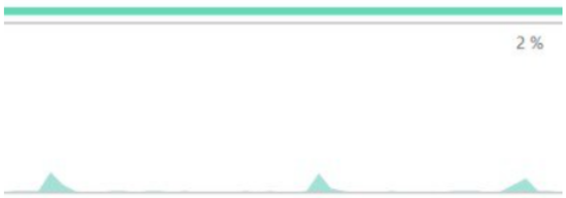
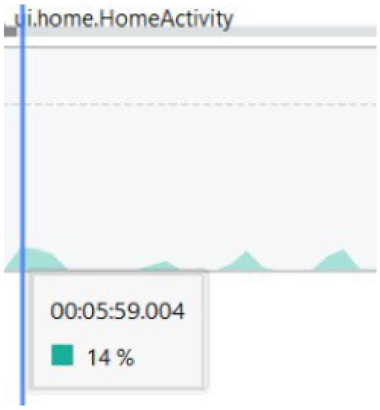
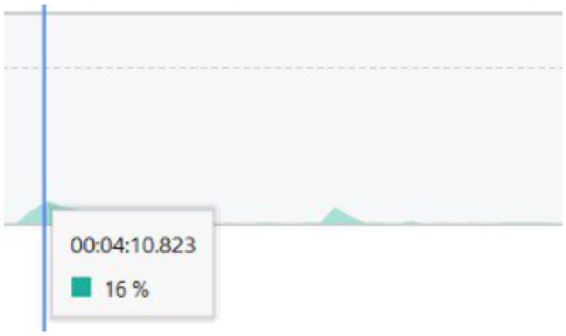
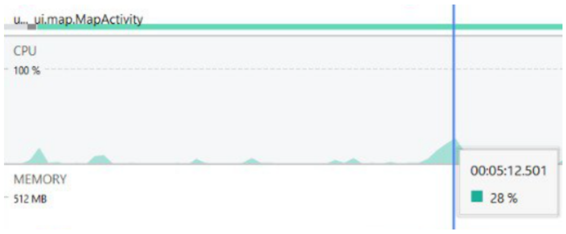
RAM History:



CPU Utilisation

Idle period (inclusive of beacon scanning): Relatively constant at 1-2%
Database-related processes: 12-18%
Map-related processes: 20-30%

CPU History:
Sample spikes during processes:

Process	Average	Snapshot
Idle (inclusive of beacon scanning)	1-2%	
Joining a queue	~14%	
Database calls	~16%	
Loading map	~28%	 u..._ui.map.MapActivity CPU 100 % MEMORY 512 MB 00:05:12.501 28 %

Limitations of Our App Development

We found that drawing a custom map specifically to represent our theme park using Google Map was not possible with the Google Workspace account we had as we did not have access to Google My Maps with our SMU email. Hence, we could only show a view of the theme park using Google Maps.

Also we did not have access to the theme park's estimated waiting time of the physical queue. While the beacons can estimate the number of people in the vicinity of the ride, we still need the ride operators or theme park staff to give us an estimate on the waiting time based on how many people are physically waiting in the queue for an accurate representation of the full queue and estimated waiting time. Most theme parks currently have an existing way of estimating waiting time for rides, either via cameras or physically counting the number of people in line. Hence, it would be better for the park to provide us with that information for us to better estimate the total waiting time and queue size.

Users have to download the app and have their Bluetooth enabled in order for our app to reach its full potential. If users choose to opt out of our participatory sensing by not enabling their Bluetooth, our app would not include them in the calculations of crowd and queue levels, which would lead to a less accurate representation of the actual crowd density and queue levels in the park.

The beacon functionality could've been extended to indicate things like facilities nearby, eg. washrooms or food stands, since the beacons can have a long range, especially outdoors. Furthermore, the beacons could be used to detect which part of a ride's physical queue users are at, and allow them to try out interactive activities with displays along the queue such as scanning a QR code on a sculpture or taking a trivia challenge on the ride they are queueing for. There could also be a more intelligent and adaptive implementation of the beacon sensing, eg. measuring the time between new beacon detections, as users may be in the same place for a long time (eg. waiting in a queue), hence beacon scanning could be decreased during this period.

Device Constraints

Physical Device

A physical Android device is required to run the application with the Beacon service running in the background in order to detect nearby beacons. An emulator will not be able to pick up Bluetooth signals from nearby beacons.

Google Play Store Services Installed

To use Google Maps services, it requires the phone to have Google Play Store services available so that the app can initialise the GoogleMap object.

Minimum SDK: 21

For SDK before level 21, Android uses the Dalvik runtime for executing app code which limits apps to a single classes.dex bytecode file per APK. However, our requested classes cannot fit in a single dex file so it requires minimum API level 21 where ART is used to natively support loading multiple DEX files from APK files instead.

Permissions Required
The user must have a working internet connection so that the latest data on the waiting time can be retrieved from Firebase. The user must also have Locations allowed for the application to access the user’s current location.

Future Expansions

Our application can be extended to include a personalised itinerary based on users' favourite rides and past experiences. This is more plausible for companies owning multiple theme parks such as Disneyland or Universal Studios with a larger set of data to analyse for higher accuracy in personalising the itinerary. The itinerary could be curated based on certain criteria such as the most frequented rides, the most recent rides, preferred shows and timings for meals.

With the integration of Bluetooth Beacons in our solution, our application can be extended to other use cases such as event space management where crowd management would be a very useful feature to have. In an event space management scenario, navigation is also necessary however slight modifications can be made when navigating through an indoor environment.

We are also interested in exploring In-queue tidbits to share information about rides and attractions. For instance, a beacon can be placed on an artwork or interactive placeholders that are left around the queue. When a rider is near the beacon, a notification could be sent to the rider's device to tell them the backstory or fun facts about the artwork or interactive placeholders based on the movie or television series where it originated. This might be harder to implement as it requires more BLE activity which may cause a drastic battery drain.