2024

Q-Up - Virtual Attraction Queueing System with Android App

FINAL YEAR PROJECT REPORT

AUTHOR: DYLAN FENNELLY (20093427)
SUPERVISOR: JOHN RELLIS

Table of Contents

List of Figures

1. Plagiarism Declaration

I hereby certify that this assignment is all my own work and contains no plagiarism. By submitting this assignment, I agree to the following terms:

Any text, diagrams or other material copied from other sources have been clearly acknowledged and referenced as such in the text by the use of citations including the author's name and date of publication [e.g. (Byrne, 2008)] in the text proceeding the referenced material. These details are then confirmed by a fuller reference in the bibliography.

I have read the sections on referencing and plagiarism in the handbook or in the SETU Quality Manual and I understand that only assignments which are free of plagiarism will be awarded marks. I further understand that SETU has a plagiarism policy which can lead to the suspension or permanent expulsion of students in serious cases (SETU, 2023)

Signed: Dylan Fennelly
Date: 23/04/2024

2. Acknowledgements

I would like to extend my gratitude to my project supervisor, John Rellis, for his assistance throughout this process. His suggestions and recommendations have been grounded in both experience and expertise, and I would have developed a much worse project were it not for his knowledge.

3. Introduction

Amusement parks are a place of wonder and joy for many, museums a place of learning and enlightenment, and festivals a place of relaxation and connection. However, one common problem plagues of all these leisure centres and dampens the experience for all involved: queueing. Waiting builds anticipation, but no level of anticipation can outweigh the frustration and boredom of standing in a barely moving queue, looking at the attraction and thinking of all the other experiences you had to give up on in favour of this. Thankfully, advancements in smart phones, network connectivity, and cloud computing has brought forward a new solution to queuing: virtual queues, allowing for visitors to queue virtually for an attraction without being bound to a physical queue.

This project, Q-Up, is a full-stack implementation of both a virtual queuing system and an attraction-based facility companion app. The Android app allows for visitors to view a list of attractions in the facility – either in a list or on a map – and enter a virtual queue for an attraction, granting them the freedom to roam the facility and enjoy other, smaller attractions while remaining in queue for the main attraction. Delivered in a Software-as-a-Service fashion, users will only need to download a singular app for many facilities, as attraction and facility data is retrieved from the cloud through Amazon Web Services.

3.1 Background

I love theme parks: the rides, the atmosphere, the history. Since I was young, I've always had a large interest in theme parks, watching video recordings of rides and reading up on the history of famous parks and attractions. Additionally, during my 6-month internship in Errigal in 2023, I got to work on the company's mobile interface of their network management system and Google Maps implementation, with custom markers and functionality tied to interaction with the markers (Errigal Software Solutions, 2023).

3.2 Motivation

As much as I love theme parks, I do not enjoy queueing for extended periods of time. Very few people do, and queues are common source of visitor boredom, discomfort, and overall dissatisfaction with the amusement park experience (Pearce, 1989). While many efforts have been made to make queue experience more enjoyable – from physical aspects such as environmental storytelling and interactive queue elements, to psychological effects that make the visitor feel like they are closer to boarding the ride than they actually are (Ledbetter, Mohamed-Ameen, Oglesby, & Boyce, 2013) – or to offer ways to skip large parts of the queue through "FastPass" like systems, none have managed to reduce the need for and time spent in physical queues by all park visitors.

Pondering upon this issue through the lens of software development and my interest in cloud computing, I began to think about a software solution that could reduce time spent physically in queues by allowing visitors to join virtual queues that would permit them the freedom to roam the park and possibly board other attractions while remaining in the virtual queue for their chosen ride. Considering the ubiquity of smart phones, the ever-expanding standard of internet connectivity (through both Wi-Fi and cellular networks), and the easily scalable, fault tolerant, and global nature of cloud computing infrastructures, I began to consider a mobile app that syncs with the cloud for queue times and positions.

3.3 Problem Statement

The problems posed by physical queuing are multi-faceted and affect both visitors and facility operators.

For visitors, physical queues introduce prolonged periods of standing and waiting, which causes discomfort, boredom, and frustration. Popular attractions and those with longer runtimes are bound to have larger queues, which presents an issue of opportunity cost for the visitor: they can choose to queue for the popular attraction and lose the opportunity to experience other smaller attractions, or skip the popular attractions and miss out on possibly the best experiences in the facility. Both issues combine to reduce visitor satisfaction and enjoyment of the facility – a result which is often contradictory to the purpose of these facilities.

For facility operators, low visitor satisfaction is bad. Visitors who had a poor experience due to long physical queues and the inability to experience multiple key, 'big-ticket' attractions within the facility are less likely to return for another visit in the future and may advise friends and family against visiting. In addition, large numbers of visitors caught up in a queue for a singular key attraction and the unavoidability of these long queues means less traffic is going towards smaller attractions and concessions, as visitors spend upwards of 1-2 hours in a single physical queue. This leaves these smaller attractions under-attended and less worthy of the cost of maintenance and upkeep.

3.4 Industry Examples

The most high-profile implementation of virtual attraction queues is in Disney World's various parks. Through the My Disney Experience app, visitors can enter virtual queues for select attractions and experiences (Disney World, n.d.). This virtual queuing system is a custom solution tailor-made to Disney's parks and is only selectively rolled out to specific attractions.

Attractions.io is an attraction-based facility companion app that encompasses many components and features, including directions, live attraction status updates, and virtual queueing (Attractions.io, 2021). Aimed at being a full companion guide, Attractions.io offer their app in a Software-as-a-Service (SaaS) format, which is then customized to suit the solution of the partnering facility and release on the app stores as an individual app for that specific facility. While their primary focus revolves around theme parks, they also offer their app for zoos, resorts, and cultural facilities. Notable facilities that make use of Attractions.io's service include Alton Towers, Legoland, and San Diego Zoo.

Accesso are another industry example that provide virtual queueing and guest experience management to a variety of industries: theme parks, zoos, cultural facilities, resorts, ski lodges, live entertainment, and more (accesso, n.d.). However, compared to Attractions.io, Accesso do not offer a mobile app, and instead are focused on facility-operator side software and physical hardware solutions to provide their services. Notable partners include Cedar Fair, Six Flags, and Columbus Zoo.

3.5 Aims and Objectives

I aimed to produce an Android application that allows for visitors of a facility to view the attractions in said facility through both a map and a list. Through the list, the visitors can view information further information about each attraction, including the estimated queue time, and join a virtual queue for the attraction, with a list of their active virtual queues viewable for ease of access. Once the visitor has joined a virtual queue for an attraction, their remaining queue time is displayed instead of the total queue time, calculating the user's position in the queue and estimating how long they have left to wait.

The app periodically checks the users position and calculates how far away they are from each attraction they have queued for. When the distance between the user and the attraction is great enough such that it would take longer than their remaining queue time to walk to the attraction itself, the app sends a push notification prompting the visitor to make their way to the attraction.

Once there is only five minutes remaining in the queue, an attraction entrance ticket becomes available to the user. This QR code is scanned at the entrance to the attraction to permit the visitor entry, after which the queue is completed and their entry removed. After this ticket is generated, if the user does not use it within twenty minutes, the ticket becomes invalidated and their queue position lost.

The app also allows users to log-on anonymously without requiring an account to be made through the use of QR codes provided by the facility when the visitor enters the facility or books their visit. These codes are one time use, preventing multiple users from logging on to the app with a single code.

The app is delivered in a Software-as-a-Service format, similar to Attractions.io. However, unlike Attractions.io, Q-Up is a singular application that works the same with all facilities that use it, simplifying the application experience for visitors – particularly those who frequently visit multiple different attraction-based facilities – and enabling smaller facilities to access virtual ride queuing functionality.

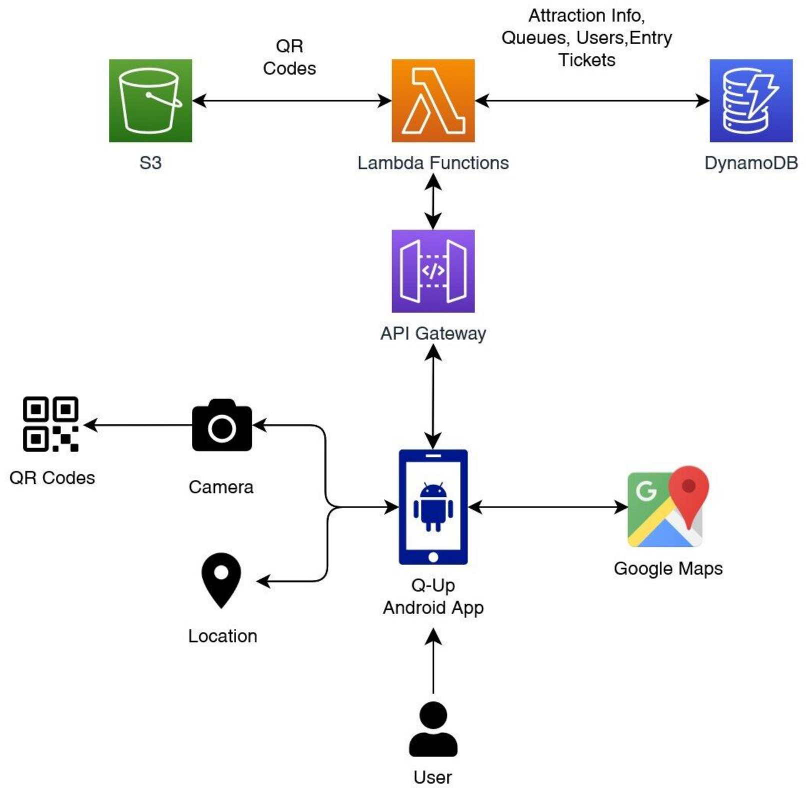


Figure 1 - Overview of Q-Up application technology

Figure 1 shows the basic overview of the technology and implementation used for the Q-Up application. The visitor downloads the Android app, which makes use of Google Maps to display facility maps in the app, device location to check distance from attractions, and the camera to scan QR codes to check-in to facilities. The app communicates with Amazon Web Services resources through a REST API and serverless Lambda functions to verify the validity of the scanned QR codes, lease temporary user IDs to enable anonymous login, obtain attraction and queue information, and to join, leave, and complete user queues.

The minimum viable product deliverable for Q-Up allowed for the user to view attraction information for the specific facility on both a map and in a list and join a queue for them, from which then the user is notified when it is their turn to enter the attraction. The stretch goals – features that are not essential to the operation of the app but still provide important and useful functionality - for the Q-Up application were as follows:

1. Facility staff IDs that allow for the execution of certain functions, including managing attraction statuses and sending out facility-wide notifications through the app.
2. Allow for users to join into a group, which enables visitors to queue for attractions as a singular unit instead of requiring each visitor to queue for the attraction separately.
3. Control how visitors enter attractions through the use of one-time scannable tickets.

4. Design

This section will discuss the final architectural and technological design of the Q-Up application, including further explanation and details on each component and feature of the application. The application's design underwent a number of revisions throughout the development process, which are detailed in the Implementation section of this report.

4.1 Requirements

Requirements lay out the functionality of the application and what must be in place to achieve said functionality. Requirements are often split into two categories: functional and non-functional requirements (Altexsoft, 2023). Functional requirements define the actions the application must enable the user to perform. A failure to meet these requirements means that the application cannot perform the necessary actions required for the application to work as intended.

Non-functional requirements, on the other hand, describes how the application should work and perform. Rather than defining the essential functionality, they describe how the functionality is delivered. Requirements layout a path for development and define what Q-Up must be able to do and how it should do them.

4.1.1 Functional Requirements

The functional requirements of the Q-Up app were as follows:

1. Users must be able to check-in/log-in to a facility.
2. The app must be able to retrieve facility and attraction data from the cloud.
3. Users must be able to view attractions in a facility on both a map and a list.
4. Users must be able to queue virtually for an attraction.
5. Users must be able to view their queue status at any time, with a queue-time estimate.
6. Users must be able to leave the queue at any time.

Further sections will go into more detail as to how these requirements were achieved and the technology used to achieve them.

4.1.2 Non-Functional Requirements

The non-functional requirements of the Q-Up app were as such:

1. Users should be able to check-in into the facility anonymously/without signing up with an email.
2. Users should be able to scan a QR code on-site to check-in.
3. Leased user-IDs must only persist for a defined period of time, after which they are revoked.
4. Attractions should be categorized by type/status.
5. The app must sync with the cloud regularly to provide up-to-date information regarding queue times.
6. The app should send push notifications to the user to notify them about their queue position.
7. Users must be removed from a queue if they do not check into the attraction after a defined period of time.
8. Users should be able to receive directions to an attraction using the map.
9. Disconnecting from the internet or closing the app must not cause the user to lose their user ID.

4.2 Architecture Model

Figure 2 shows the application and cloud architecture for a single facility. The major components and processes will be explained in detail in the System Design section, with aim to how each component helps to achieve the application's requirements. Details about how each feature was implemented are available in the Implementation section of this report.

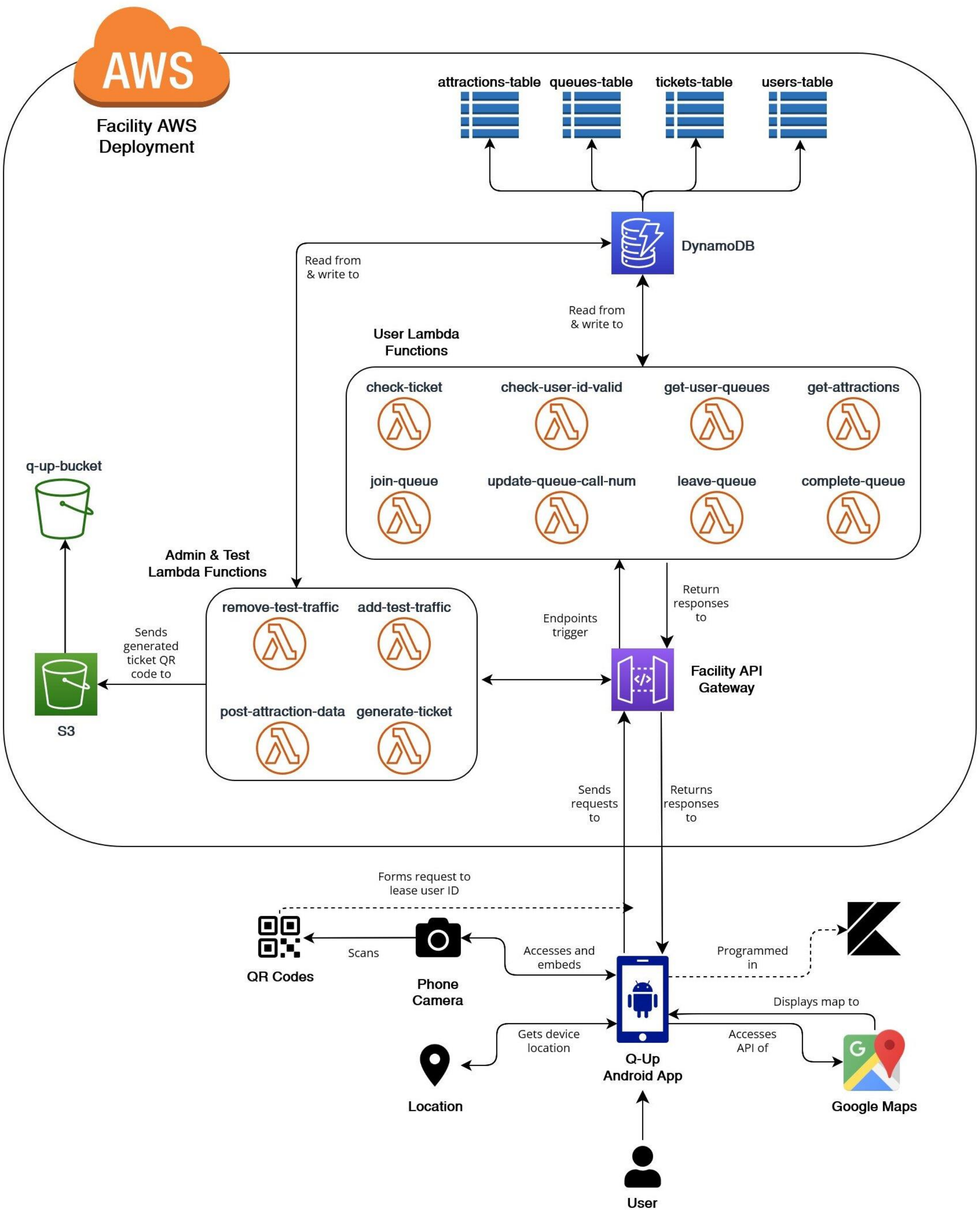


Figure 2 - Q-Up application and cloud architecture overview

4.3 System Design

Expanding on the architecture model from the previous section, further detail will be given as to how certain components operate and help to achieve the application requirements. This section will look at an overview of the Cloud infrastructure, including the API Gateway REST API, User and Admin Serverless Lambda Functions, DynamoDB and the storage of data, Anonymous log-on with S3 and QR Codes, Security, Attraction Maps, Accessibility, and App Installation. More detail as to why specific technologies were chosen will be explained in the Technology section of this report.

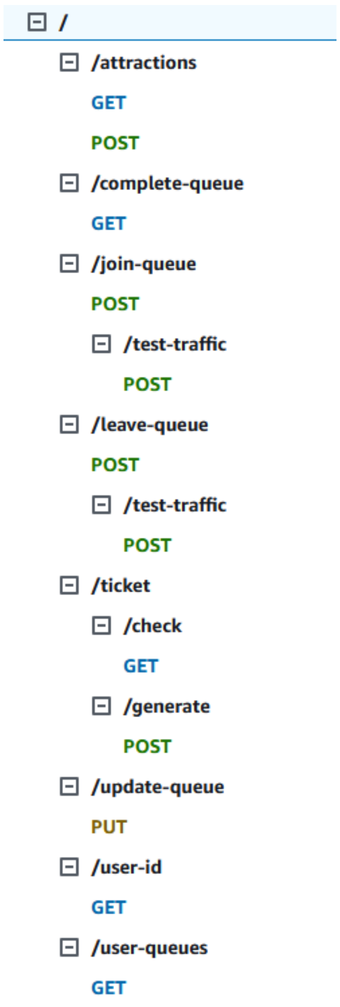
4.3.1 Cloud Infrastructure Overview

The Amazon Web Services (AWS) infrastructure is a key part of the overall application architecture, being the backend to the front-end Android application. Facility and attraction info, attraction queues, user IDs and the QR codes to obtain such user IDs are provided through AWS. Each facility registered with the service is allocated their own cloud infrastructure and resources, isolating their data and functionality from other facilities.

The highly distributed nature of cloud architectures such as this allows for information to be accessed over the internet and resources to be deployed locally to suit the location of the facility, enabling worldwide operation. Along with providing many of the resources to fulfil the functional requirements, the on-demand nature of cloud computing enables for data to be accessed at any time the visitor wants.

4.3.2 REST API

Representational State Transfer (REST) APIs allow for information to be consistently and uniformly transferred between client and server (user and host) using requests formed from HTTP verbs, such as GET for retrieving information and POST for sending data. The REST API enables users to access information from the backend AWS services without granting them direct access (Amazon Web Services, 2023). API Gateway is an AWS service that provides a REST API through which services can be accessed on endpoints. Each facility has their own gateway, which grants access to that facility's resources. The API Gateway acts as the point of communication through which all information to and from the application and the cloud passes through. The structure of the Q-Up application's REST API can be seen in Figure 3



```
⊟ /
  ⊟ /attractions
      GET
      POST
  ⊟ /complete-queue
      GET
  ⊟ /join-queue
      POST
    ⊟ /test-traffic
        POST
  ⊟ /leave-queue
      POST
    ⊟ /test-traffic
        POST
  ⊟ /ticket
    ⊟ /check
        GET
    ⊟ /generate
        POST
  ⊟ /update-queue
      PUT
  ⊟ /user-id
      GET
  ⊟ /user-queues
      GET
```

Figure 3 - The API structure of the REST API

4.3.3 Serverless Functions

To connect the API endpoints to the services themselves, there needs to be a compute-based resource that will execute the requests represented by each endpoint. Lambda is an AWS service that allows for the execution of code functions without needing to set up servers. Driven by events, Lambda functions execute upon defined triggers (Amazon Web Services, 2023). The lambda functions in the cloud infrastructure are tied to specific API endpoints, which act as the catalyst for the retrieval of information and execution of specific functions and are key to providing the functional requirements of the application.

4.3.3.1 User Lambda Functions

The User Lambda Functions are a grouping of serverless Lambda functions that standard application users, such as visitors to a facility, execute through the Android application. The User Lambda Functions all perform read and/or write operations to the DynamoDB tables. Each Lambda function's purpose is listed below:

- **check-ticket**
    - Check that the scanned entrance ticket QR code is valid (is past the activation time, has not expired, and has not been used yet), and provide the user with a user ID and facility information. The corresponding ticket in the tickets table is updated to be marked as used, and a record for the leased user ID is created in the users table.
- **check-user-id-valid**
    - Checks if the specified user ID is still valid (if it exists in the users table and is not past its expiry time), returning a Boolean of true if its valid, and false if not. This check is performed automatically by the Android application upon the launch of the application using the user ID stored in the app's DataStore, removing the stored ID if invalid.
- **get-attractions**
    - Gets and returns all the attractions in the attractions table. This function does a lookup for queue table entries for each attraction and appends the number of users in queue for said attraction to the API response.
- **get-user-queues**
    - Gets all attraction queue entries in the queues table for a specified user ID. This function also does a lookup for the number of users ahead of the specified user in the queue and appends the result to the API response.
- **join-queue**
    - Creates a queue entry for a specified user and attraction.
- **update-queue-call-num**
    - Updates the call number value of a specific queue entry to a specified number. The purpose of this value is explained in section 4.3.4.2.
- **leave-queue**
    - Removes a queue entry for a specified user and attraction.
- **complete-queue**
    - Completes and removes a queue entry for a specified user and attraction. This is triggered when an attraction entrance ticket is scanned.

4.3.3.2 Admin & Test Lambda Functions

The Admin & Test Lambda Functions are a grouping of serverless Lambda functions that are only available to the admin user with AWS credentials, preventing standard users from executing them. These functions are not executed from the Android application and are accessible as API endpoints only. Each Lambda function's purpose is listed below:

- **add-test-traffic**
    - Creates dummy traffic for an attraction within a specified user ID range. This was used for testing the functionality of the queue time estimates and the notification sent based on the user's distance from the attraction.
- **remove-test-traffic**
    - Removes dummy traffic for an attraction within a specified user ID range. This was used for testing the same functionality as above.
- **post-attraction-data**
    - Creates a new attraction entry in the attractions table.
- **generate-ticket**
    - Generates a facility entrance QR code and corresponding tickets table entry.

4.3.4 Data Storage and Attraction Queues

The facility data – attractions, users, and tickets – and attraction queues are provided through DynamoDB, a serverless, high performance NoSQL database. Fast and flexible, DynamoDB can provide for both structured data like the attraction information and for data that needs to be regularly updated, such as the attraction queues (Amazon Web Services, 2023). Each facility has its own tables for their specific data and queues. The database is the most crucial part of the cloud infrastructure, as the storage and retrieval of data and queues were vital to the fulfilment of the functional requirements.

4.3.4.1 Attractions Table

The Attractions Table stores information about the attractions in the facility. The table design is as follows:

- **id** – Internal ID of the attraction for tracking queue entries. This is the partition key of the table.
- **name –** Name of the attraction.
- **description –** Description of the attraction.
- **type –** The type of attraction it is (e.g. rollercoaster, exhibit, show, etc.).
- **status –** The current status of the attraction (whether it is Open, Closed, or undergoing Maintenance).
- **cost -** The entrance cost for the attraction,
- **length –** How long the attraction is in seconds,
- **lat –** The map latitude position of the attraction’s map marker
- **lng –** The map longitude position of the attraction's map marker
- **avg_capacity –** The average amount of people per the time frame defined in *length* that the attraction receives.
- **max_capacity –** The maximum capacity of the attraction per run

4.3.4.2 Queues Table

The Queues Table stores information about the virtual attraction queues in the facility. The table is designed as such:

- **attractionId** – Internal ID of the attraction for the queue entry is for. This the partition key.
- **userId** – ID of the user the queue entry is for. This is the sort key.
- **time** – A timestamp of when the queue entry was created. This is formatted as a ULID (universally-unique lexicographically-sortable identifier). ULIDs and the reason for their usage is detailed in the Technologies section of this report.
- **callNum** – A numeric value that denotes how many times the application has notified the user about updates regarding their queue entry. The meaning of each call number value is as follows:
    - **0 –** The user has not been called for the attraction queue yet.
    - **1 –** The user has been called to make their way towards the attraction.
    - **2 –** The user’s entrance ticket has been generated.
    - **3 –** The user has been reminded after 10 minutes to enter the attraction and have been told they have 10 minutes remaining to scan their ticket and enter the attraction or they will be removed from the queue.
    - **4 –** The user has been reminded after another 5 minutes to enter the attraction and have been told they have 5 minutes remaining to scan their ticket and enter the attraction or they will be removed from the queue.
    - **5 –** The user’s queue entry has expired due to not scanning their ticket. On next refresh, their entry will be removed.
- **lastUpdated** – A UTC timestamp that tracks the last time a *callNum* update has occurred to know how long it has been since the last *callNum* update.

The Queues Table also has a Global Secondary Index. This is a separate way of filtering the table entries that can treat values – other than the primary partition and sort keys – as partition and sort keys, allowing for more complex queries to be made (Amazon Web Services, n.d.). This Global Secondary Index, *user-queues-index*, treats the *userId* as the partition key, and the *attractionId* as the sort key. This allows for the table to be queried to get all of a specified user's queue entries. Without this, the computation to get every attraction a specific user has queued for would be slow, as the function handling the request would need to check every existing attraction ID to ensure all user queue entries are being returned.

4.3.4.3 Tickets Table

The Tickets Table stores information about the generated facility entrance ticket QR codes. The table design is as follows:

- **ticketId** – A unique identifier code for the ticket. This is generated as a UUID (universally unique identifier), which is guaranteed to be unique, preventing overlapping IDs and DynamoDB errors. This is the partition key of the table.
- **startDate –** A UTC timestamp that tracks when the facility entrance ticket code becomes active.
- **startDate –** A UTC timestamp that tracks when the facility entrance ticket code expires.
- **activated** – A Boolean that denotes whether or not the facility entrance ticket has been scanned yet.

4.3.4.4 Users Table

The Users Table stores information about the leased user IDs. The table design is as follows:

- **userId –** The leased numeric user ID. When the user ID is leased and the table entry created, the Lambda function scans the table for the current highest user ID and creates the next highest one. This is the partition key of the table.
- **leasedAt** – A UTC timestamp that tracks when the user ID was leased.
- **expiresAt** – A UTC timestamp that tracks when the leased for the user ID expires. This is set to be the same expiry time as the facility entrance ticket code that the user ID was leased from.
- **ticketId** – The ID of the ticket that was scanned to lease the user ID.

4.3.5 Anonymous Log-on

To prevent false requests from outside traffic, visitors must verify their attendance at the facility before being able to access information from other AWS services. The generated QR codes serve as single-use entry tickets that allow visitors to obtain a user ID and enter virtual queues without having to create an account with Q-Up or the facility they are entering, providing for the authentication and security aspects of the functional requirements. The generated facility entrance ticket QR codes are stored in a global S3 bucket, with a separate directory for each facility. The leased IDs only remain valid for a set length of time, also fulfilling some of the non-functional requirements.

4.3.6 Security

Security is an important aspect of the system design to help prevent malicious or outside actors from altering data or taking actions that may prevent the virtual queuing system from being used. Through the facility entrance ticket QR codes, users are authenticated with one-time use entry codes. Each user gets a single code to use, either when they enter the facility or as part of their overall booking with the facility operator. As these codes are one time use, it prevents users that are not visitors to the facility gaining user IDs and sending false traffic to attraction queues. Additionally, the API endpoints and connected Lambda functions that make significant changes to attraction data or queues without requiring an input user ID are locked behind the requirement to have approved AWS credentials, which are not distributed to visitors.

However, the current security implementation has flaws. Firstly, while the using the application requires a user ID only obtainable through scanning a one-time use QR code, the user-accessible API endpoints are open, not requiring any API key to access. This means that outside actors who know the API endpoint for a facility and how the requests are structured can manufacture API requests and send false data to attraction queues. This security issue could be fixed by implementing Cognito as part of the AWS infrastructure. Cognito, similarly to the current implementation of user IDs, allows for the creation and leasing of anonymous identities without the need for any form of sign-up or log-in (Amazon Web Services, 2023). Unlike the current implementation however, these anonymous identities can have Roles attached to them, granting access to resources through the identities, and blocking anyone who doesn’t have a Cognito-distributed identity.

Additionally, while Lambda and DynamoDB automatically scale resources to match demand, the scaling does not happen instantly, meaning large surges in traffic such as that from a DDOS (direct denial-of-service) attack can render services unavailable to users. A Web Application Firewall could be included as part of the infrastructure connected to the API Gateway to protect the infrastructure from DDOS attacks and other malicious traffic (Amazon Web Services, n.d.)

4.3.7 Attraction Maps

To visually display the layout of the facility and the attraction markers, Google Maps is embedded into the app. The attraction information retrieved from the cloud database contains information to used by Google Maps, including latitude and longitude coordinates, which is then used to display markers on the map. The markers use a custom-designed map marker icon viewable in Figure 4.

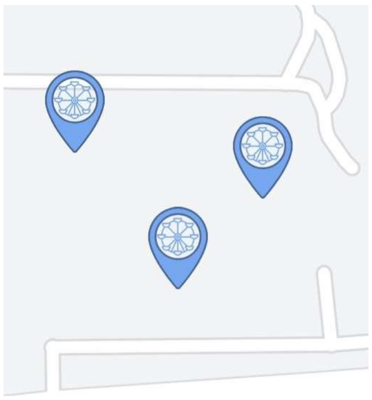


Figure 4 - Custom map markers on the in-application embedded Google Maps attraction map

4.3.8 Accessibility

Attraction-based facilities are often popular tourist destinations, attracting visitors from around the world. To facilitate this global reach, strings in the app are extracted to a string resource file in the Android environment that makes translation of the app much easier and more consistent.

4.3.9 App Installation

The singular Q-Up Android application is obtainable as an APK package from the projects GitHub repository. In an actual business deployment of the app, the app would be obtainable from the Google Play Store. The app is relatively small in size at just under 60 MB, allowing for the app to be easily obtainable over metered and congested networks, such as those often found at attraction-based facilities. This was not as small as desired but was the result of the packaging of required dependencies and assets.

4.4 User Stories

User stories are another way of explaining the applications functionality from the perspective of the user, with a focus on the value the product and its features will provide to its users rather than the technological requirements (Rehkopf, User stories with examples and a template , 2023). They are non-technical and focus on what the users should be able to achieve with the application. They are an essential part of agile software development, which will be explored in more detail in the Methodology section of the report and help to focus development around the most important functionality. The following user stories outline the key functionality of the application for the various users and were used to guide development:

- As a visitor, I want to be able to view the attractions of the facility.
- As a visitor, I want to be able to join a virtual queue for an attraction.
- As a visitor, I want to be able to see my estimated queue time at any time.
- As a visitor, I want to be able to view the layout of the facility.
- As facility staff, I want to be able to update the status of an attraction.

4.5 Risk Analysis

There were two main points of risk for the development of this application. Firstly, there was not a huge amount of documentation or prior examples of queuing systems such as the one required available throughout my initial research for this project. However, an example implementation of a customer queue for a call centre using DynamoDB and Lambda functions posted to the AWS Blog showed a queue system similar to the one required for the Q-Up application (i.e. allows for the user's position in the queue to be retrieved) being possible (Basak & Evarts, 2022). This blog post proved essential in de-risking this aspect of the project, and served as a foundation from which the queueing system and minimum viable product were achieved.

Secondly, were this application to be launched as a business, the market for virtual queueing systems and companion apps for attraction-based facilities is competitive, with companies like Attractions.io providing a service similar to what Q-Up would provide, Accesso providing alternative solutions to the same issues, or large facilities such as Disney World developing their own, in-house solutions. Q-Up, to compete, would have to focus on providing services to smaller facilities that cannot afford Attractions.io or Accesso's solutions and winning over consumers in the name of simplicity through a single app for all facilities.

5. Methodology

In this section, the Agile software development methodology will be explored along with the associated concepts of test-driven development, version control, and continuous integration and continuous development, and what the application of these concepts is.

5.1 Agile

Agile is a project and development management methodology that divides the work required into phases, or sprints, that take place over a defined period of time with a set number of objectives (Atlassian, 2023). Emphasising iterative development, Agile allows for feature-focused development with clear outlined tasks, while also having the flexibility to create new tasks and objectives for the next cycle. Ideally, each sprint should produce a version of the application with some level of significant change from the previous. Using an Agile methodology helped to keep the project on track and focused on the implementing the features required to meet the functional requirements of the ideal application and the minimum viable product.

For this project, rather than approaching agile with set time-based sprints, I took it with a feature-based sprint approach, where sprints are done on a feature-by-feature basis with the objective of having each sprint having a major feature completed by the end. This approach was taken mainly with regards to the fact that I had multiple other modules to be doing work and assignments for throughout the semester and may not be able to get significant work done during time-based sprints depending on work that had to be done for other modules. These feature-based sprints are detailed in the Implementation section of this report.

5.2 Test Driven Development

Test drive development is an iterative approach to software development that involves writing tests before writing the code functions used in the test. This approach requires thought around the expect functionality and result of a function before the function itself is written, with tests defining the expected results and the written code then matching those results to pass the test (TestDriven.io, n.d.).

This process ensures that functions perform as defined when they are first ratified, and that tests are not being written in a way that makes them easy to pass or manipulated to suit the outcome of the functions. Given the API call nature of the Q-Up application, it may be difficult to test some of the important functionality, but I aimed to employ this methodology where possible. An illustrated example of the process can be seen in Figure 5 below.

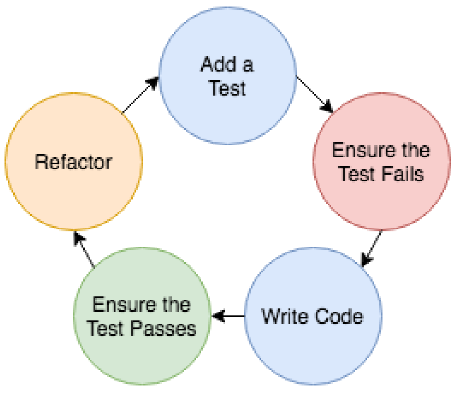
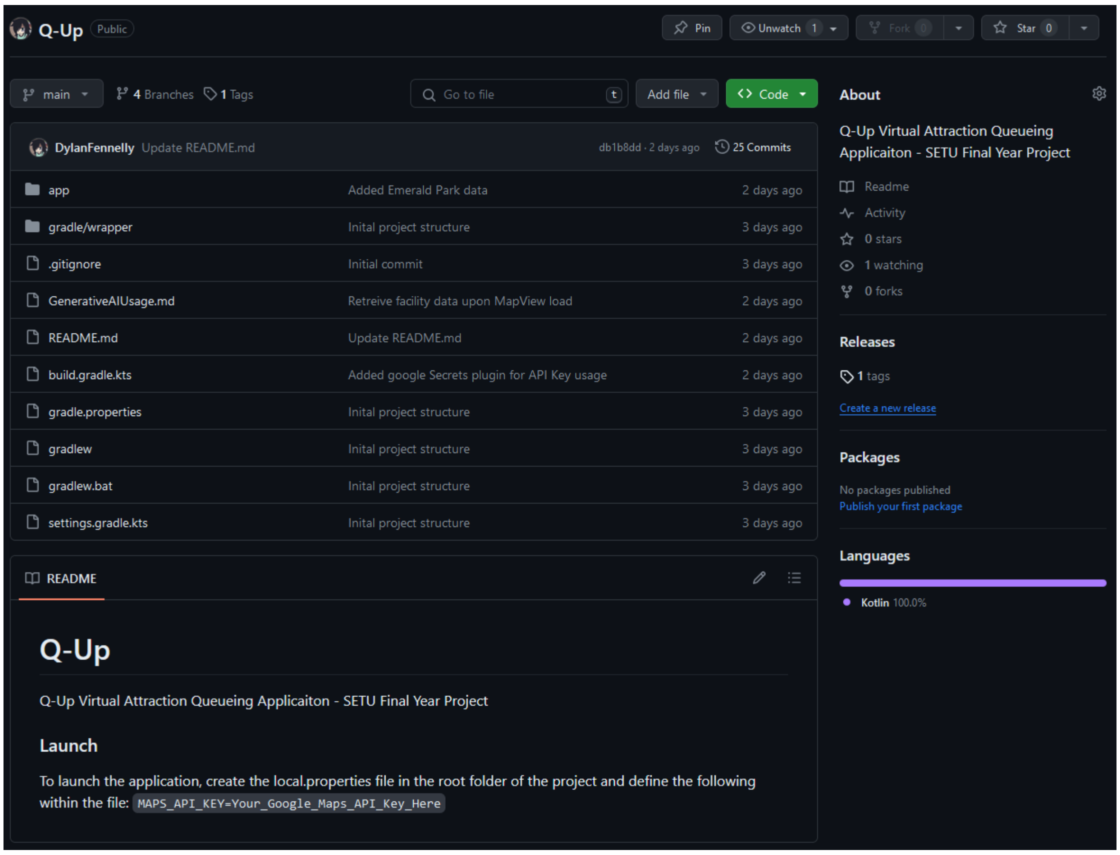


Figure 5 - Test Driven Development cycle (TestDriven.io, n.d.)

5.3 Version Control

Version control is the process of tracking and managing changes to source code through a central repository. As changes are committed to the repository, each iteration of the project before each code commit is saved, allowing for older versions to be reference or rolled-back to at any time (Atlassian, 2023). It is an essential part of iterative development and assists with the creation of successful builds. Git and GitHub were used in this project for version control: Git as the version control system and GitHub as a storage for the Git code repository (Figure 6). In this project, changes could not be directly committed to the main branch and had to be merged through an approved pull request. This prevents unapproved code from becoming part of the deployed and released application.



DylanFennelly Update README.md db1b8dd · 2 days ago 25 Commits

app	Added Emerald Park data	2 days ago
gradle/wrapper	Inital project structure	3 days ago
.gitignore	Initial commit	3 days ago
GenerativeAIUsage.md	Retreive facility data upon MapView load	2 days ago
README.md	Update README.md	2 days ago
build.gradle.kts	Added google Secrets plugin for API Key usage	2 days ago
gradle.properties	Inital project structure	3 days ago
gradlew	Inital project structure	3 days ago
gradlew.bat	Inital project structure	3 days ago
settings.gradle.kts	Inital project structure	3 days ago

README

Q-Up

Q-Up Virtual Attraction Queueing Applicaiton - SETU Final Year Project

Launch

To launch the application, create the local.properties file in the root folder of the project and define the following within the file: `MAPS_API_KEY=Your_Google_Maps_API_Key_Here`

About

Q-Up Virtual Attraction Queueing Applicaiton - SETU Final Year Project

Readme
Activity
0 stars
1 watching
0 forks

Releases

1 tags

Create a new release

Packages

No packages published
Publish your first package

Languages

Kotlin 100.0%

Figure 6 - Q-Up GitHub repository

A link to the GitHub repository for the Q-Up application is available in the appendices section of this report.

5.4 CI/CD

Continuous Integration and Continuous Delivery (CI/CD) is a development process through which the speed and efficiency of software development and deployment can be massively increased. Continuous Integration entails the integration and merging of code changes into a central main branch, much like what was described in the Version Control section, while Continuous Delivery involves the automation of application build and testing to deliver usable versions of the application (GitLab, n.d.). Continuous Integration helps to streamline development and was used as part of this project to assist with development. However, Continuous Delivery was not applied to this project.

6. Technologies

It is important for any project to choose the right technologies to develop the application with, taking mind to the features and limitations of multiple options and choosing the correct options for the project. The following is a list of all the technologies used by the project, and why the technology was chosen over alternatives.

6.1 Android

Android is an open-source smartphone orientated operating system developed by Google, as is the main development platform for the Q-Up application. It was chosen over its main competitor, iOS, primarily for ease of access and cost. To develop iOS applications, access is required to a MacOS X computer for the Xcode IDE and to deploy the app on the App Store (Patel, 2023), which I do not have access to nor the funds to acquire. While there are workarounds that would not require purchase a Macintosh machine, such as renting a Mac in the cloud, these solutions are not ideal compared to natively running an operating system such as Windows.

Additionally, I am unfamiliar with the primary development languages for iOS, Swift and Objective-C, and running other languages (such as Java) requires a large amount of translation and heavily impacts performance (DevMountain, n.d.). In the long term, were this product to go to market, it would be absolutely required for an iOS version of the application to be developed, but for the purposes of this project, just an Android version suffices.

6.1.1 Jetpack Compose

Jetpack Compose is a UI toolkit for Android that simplifies the process of creating modern, intuitive, and reactive user interfaces (Richardson, 2020). It moves away from the old XML layouts towards and reduces the amount of coupling – the dependency of one class or module of another class or module – the application has to do as UI changes and updates are made, simplifying development and improving performance. Jetpack Compose was used for all the UI development in the Q-Up Android application, which helped to reduce development time spend on user interfaces.

6.1.2 DataStore

Along with Jetpack Compose, Google introduced a new way of persisting data for an application: DataStore. A replacement for the old SharedPreferences, DataStore stores values as simple key-value pairs, where an input key corresponds to a set value. Compared to SharedPreferences, DataStore does not block the UI – prevent the app from doing anything else – when making changes to the stored data, and allows for dynamic and continuous reading of values (Marosfalvi, 2021). DataStore is used to store the user ID and facility information when a facility entrance ticket code has been scanned.

6.2 Google Maps

Google Maps is a key component of the application, displaying the attractions visually in the facility on a map and providing directions between locations. While AWS does provide its own mapping service in Amazon Location, Google Maps integrates natively with Android and its location services and has been suggested over the alternatives by my supervisor. To use Google Maps, I had to sign up for Google Maps platform with Google Cloud and obtain a Google Maps API key.

6.3 Amazon Web Services

Amazon Web Services is a cloud computing platform that provides on-demand access to a large number of services, covering a variety of computing areas from compute, storage, and databases, to networking, content delivery, and security and authentication, all while being scalable and elastic (Amazon Web Services, 2023). While Microsoft Azure and Google Cloud Platform are good choices for cloud service providers, AWS was chosen for the development of this project primarily for familiarity, as I have extensive experience working with AWS services, and also because of the greater amount of documentation afforded by AWS being both more mature than the alternatives and having the highest market share (Felix Richter, 2023).

6.3.1 API Gateway

API Gateway is AWS's HTTP and REST API service. It allows for the creation of API endpoints that can then be accessed over the internet. By using an API, it allows for the Android app the access information from other AWS services without requiring direct access to the service. For the free tier, API Gateway allows for one million REST API calls per month, more than enough for the purposes of this project (Amazon Web Services, n.d.)

6.3.2 Lambda

Lambda is a serverless code execution service provided by AWS. It allows for the execution of code functions without the need to configure servers. Lambda functions are triggered by events occurring; in the case of the Q-Up application, Lambda functions trigger when specific API endpoints are accessed, connecting the API Gateway to other AWS services.

6.3.3 S3

Simple Storage Service (S3) is an AWS service that allows for the storage of any type of data in virtually unlimited storage containers (buckets) that can then be accessed from anywhere. S3 is an object storage solution, meaning that every file is stored as a singular, whole object (Amazon Web Services, n.d.). If only a single character of a text file is changed, the whole object must be updated. This is compared to block storage, where a file is divided into blocks and only the block containing the change is updated. As such, this makes S3 unsuitable for dynamic data, such as the attraction information and queues. S3 was instead used to store the generated facility entrance ticket QR codes.

6.3.4 DynamoDB

DynamoDB is a serverless, NoSQL database with high performance, low latency, and can be easily scaled to match demand. DynamoDB was used for the storage of attraction, user, and ticket information, and the management of attraction queues. The tables were configured with the on-demand capacity and pricing model, with boundless and dynamic scaling to match demand and charges only for the resources used (Amazon Web Services, n.d.). While the provisioned model is more cost effective, the amount of resources and throughput available to the tables is set and static, making it unsuitable for the unpredictable and 'spiky' traffic that would be received as part of this application.

Compared to other AWS database services, DynamoDB offers a free-tier allowance and is overall cheaper than Aurora, and offers higher performance and scalability for use in a serverless environment than Amazon RDS, particularly for a worldwide deployment (Achinga, 2023). While a SQL language would suit the structured nature of the attraction data, schemas allow for NoSQL data to conform to a structure the same as it would in an SQL database. Graph Databases, such as Amazon Neptune, were considered for the storage and management of user queues but were deemed not suitable for the lack of a need for the network-node like, many-to-many connections and better performance offered by DynamoDB (Amazon Web Services, n.d.).

6.3.5 CloudWatch

CloudWatch was used to monitor the performance of applications and the log outputs of Lambda functions for debugging and development. Alarms can be created in CloudWatch to trigger events upon certain metric thresholds, such as increasing capacity of a service if its usage metrics are too high. Most of the services used are serverless and are automatically scaled by AWS, and as such this feature was not necessary.

6.4 Boto3

Boto3 is the official Amazon Web Services Software Development Kit (SDK) for Python, enabling for Python code to interact with AWS services (Amazon Web Services, 2024). It was used extensively in the Lambda functions to read and write data from the DynamoDB tables and S3.

6.5 ulid-py

Universally-Unique Lexicographically-Sortable Identifier – or simply ULID – are an alternative to Universally Unique Identifier (UUID) that are lexicographically sortable, meaning that despite being a string, ULIDs will always sort in order or newest to oldest (Feerasta, 2019). ULIDs were used as part of the queues table as a timestamp to ensure that queue entries could always be easily and quickly sorted by time to calculate estimated queue times. ulid-py was the library used to implement these ULIDs into the Lambda functions (Hawker, 2020).

6.6 Segno

Segno is a Python QR code generator that allows for the creation of QR codes that can then be saved as images (heuer, 2024). Segno offers much in the way of customization of QR codes, with different sizes, border widths, and colours. Segno was used for the generation of the facility entrance ticket QR codes by the appropriate Lambda function.

6.7 Languages

Two main programming languages were be used throughout the development of the Q-Up application: Kotlin for the Android app, and Python for the AWS Lambda functions.

6.7.1 Kotlin

Kotlin is an open-source Java Virtual Machine language developed by JetBrains. It is fully interoperable with Java, with modern syntax to create more concise and efficient code (Kotlin, 2023). While Android apps can be developed in many languages, Kotlin has been the preferred language for Android development since 2019 (Lardinois, 2019) and is the language in which Jetpack Compose, the new UI toolkit for Android, is developed.

6.7.2 Python

Python is a popular open-source programming language with easy-to-read syntax and is extensively versatile through its underlying design and community support (Python, 2024). While Lambda supports many languages, including TypeScript and Go, Python was chosen for its simplicity, speed (Schmidt, 2023), versatility to perform non-standard functions (like generating QR codes), and my former experience and familiarity using Python and Boto3 in previous college modules.

7. Tools

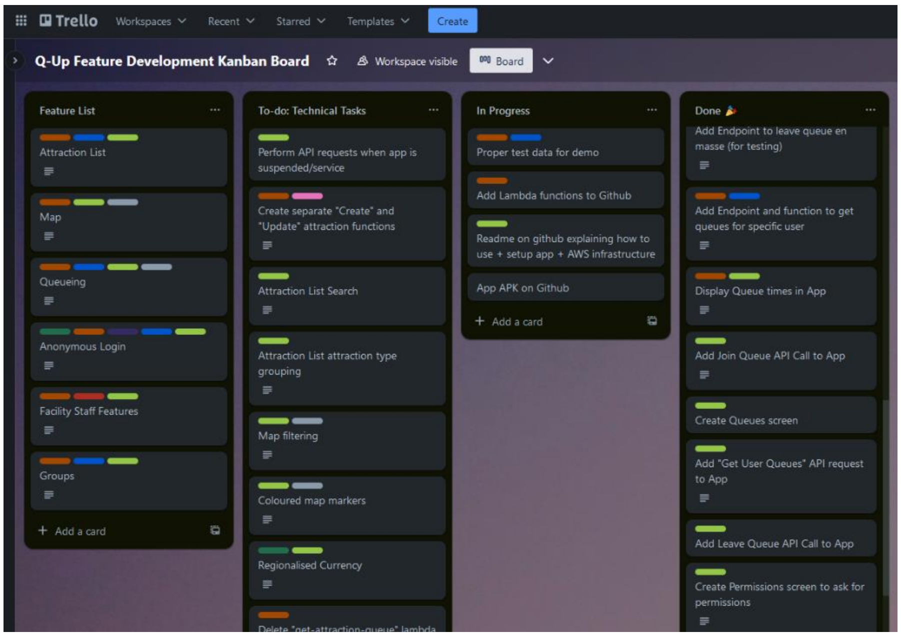
Various miscellaneous tools were used throughout the development of the Q-Up application. This section will briefly outline a number of these, and to what purpose they serve.

7.1 Git and GitHub

As previously stated in the Methodology section of this report, Git was used in this project to provide version control and development history, while GitHub was used to store the repository, manage the creation of branches, and pull requests of feature branches into main.

7.2 Trello

Tello is a tool to track tasks and progress in an Agile development environment on a kanban-style board. Kanban, literally meaning 'signboard' in Japanese, visually represents work and progression by placing them into categories – in-progress, to-do, complete, abandoned – whatever categories are needed (Rehkopf, What is a kanban board? , 2024). Trello was used extensively throughout the development process to help break down features into individual tasks, and then tracking the progress of those tasks as development progressed. Jira was also considered for as a task-tracking tool, but with the feature-based sprint Agile approach taken, a Tello kanban board was more appropriate and much simpler to use than Jira.



Q-Up Feature Development Kanban Board

Feature List
- Attraction List
- Map
- Queueing
- Anonymous Login
- Facility Staff Features
- Groups

To-do: Technical Tasks
- Perform API requests when app is suspended/service
- Create separate "Create" and "Update" attraction functions
- Attraction List Search
- Attraction List attraction type grouping
- Map filtering
- Coloured map markers
- Regionalised Currency
- Delete "get-attraction-queue" lambda

In Progress
- Proper test data for demo
- Add Lambda functions to Github
- Readme on github explaining how to use + setup app + AWS infrastructure
- App APK on Github

Done 🎉
- Add Endpoint to leave queue en masse (for testing)
- Add Endpoint and function to get queues for specific user
- Display Queue times in App
- Add Join Queue API Call to App
- Create Queues screen
- Add "Get User Queues" API request to App
- Add Leave Queue API Call to App
- Create Permissions screen to ask for permissions

Figure 7 - Trello feature development kanban board

7.3 Android Studio

Android Studio was the development integrated development environment (IDE) used for the creation and development of the Android Application. Designed with Android in mind, Android Studio comes with many tools to assist with the development of Android applications, such as easily creating app icons and a fully-featured Android emulator that can emulate a large number of devices and Android versions (Android, 2024). Considering the main Q-Up application was developed in Android, it simply made sense to use Android Studio.

7.4 GIMP

GNU Image Manipulation Program (GIMP) is a powerful open-source image editing application that provides much of the functionality that can be seen in Photoshop for free. GIMP was used in this project for the creation of the custom map marker icons visible in Figure 3, as well as the project poster for the Final Year Project Expo.

7.5 Postman

Postman is an application and platform used for the creation and testing of APIs, allowing for requests to be created and formatted with different bodies, authentication keys, and HTTP verbs, and allows for created requests to be saved and categorised into folders. It was used extensively for the creation and testing of the backend REST API.

7.6 Mermaid.js

Mermaid.js is a JavaScript based chart and diagram creation tool that allows for the programmatic creation of flowcharts, sequence diagrams, and more. This was used to create the flow diagrams in the Implementation section of this report.

8. Implementation

In this section, each development of each feature will be detailed. As previously stated, the project was developed using a feature-based sprint Agile approach, where the development was split up by feature with the aim of each sprint resulting in a finished feature of the application.

8.1 Attraction List

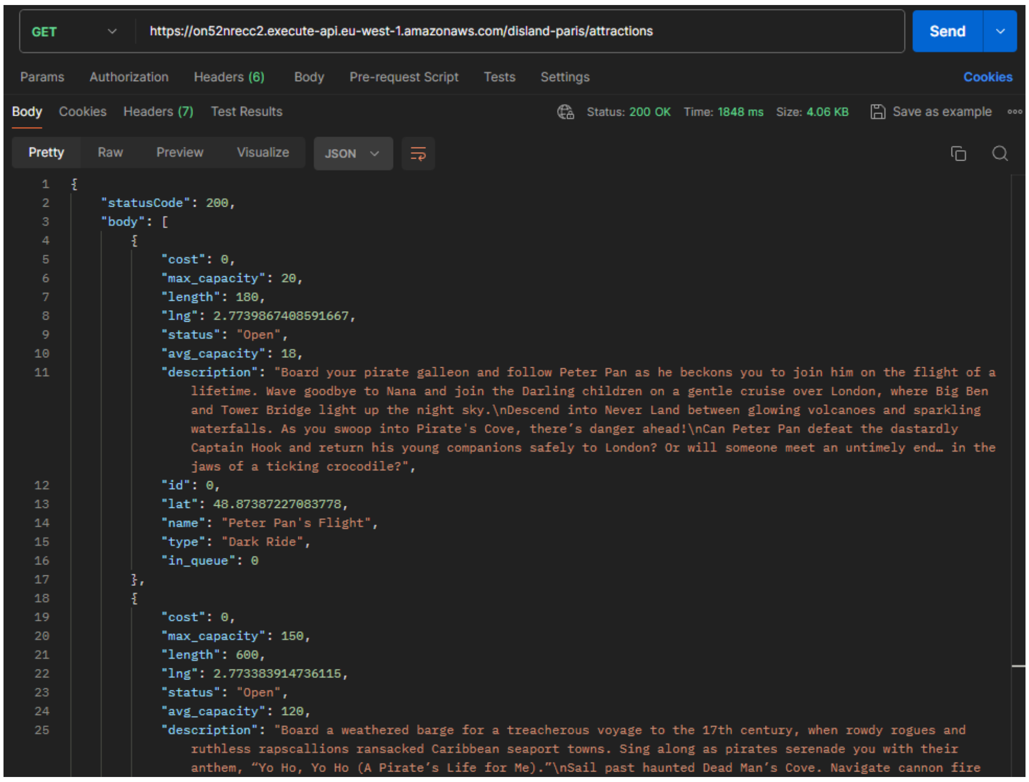
The goal of this feature-sprint was to have the Android application pulling attraction data from DynamoDB through the REST API and displaying the information in a list. Within this feature-sprint, the Attraction Map and Attraction Detail screens were also developed. These features were essential to have developed, as they are core to the main functionality of the application and the minimum viable product and are needed to for the development of further features, such as the virtual queueing.

8.1.1 Attraction List

The Attraction List displays all the attractions in the facility in a scrollable list. Starting from the base prototype application, which simply displayed hardcoded data on a map, multiple developments had to be made in both the backend AWS resources and frontend Android application.

8.1.1.1 Amazon Web Services

In AWS, multiple vital resources were created, including the API Gateway REST API and the attractions DynamoDB table, *attractions-table*. Two Lambda functions were created for the development of this feature: *get-attractions* to retrieve the attraction data from the DynamoDB, and the admin-only *post-attraction-data* to speed up adding data to the attractions table though an API request instead of through the AWS web console.



GET https://on52nrecc2.execute-api.eu-west-1.amazonaws.com/disland-paris/attractions

Status: 200 OK Time: 1848 ms Size: 4.06 KB

```
{
    "statusCode": 200,
    "body": [
        {
            "cost": 0,
            "max_capacity": 20,
            "length": 180,
            "lng": 2.7739867408591667,
            "status": "Open",
            "avg_capacity": 18,
            "description": "Board your pirate galleon and follow Peter Pan as he beckons you to join him on the flight of a
                lifetime. Wave goodbye to Nana and join the Darling children on a gentle cruise over London, where Big Ben
                and Tower Bridge light up the night sky.\nDescend into Never Land between glowing volcanoes and sparkling
                waterfalls. As you swoop into Pirate's Cove, there’s danger ahead!\nCan Peter Pan defeat the dastardly
                Captain Hook and return his young companions safely to London? Or will someone meet an untimely end… in the
                jaws of a ticking crocodile?",
            "id": 0,
            "lat": 48.87387227083778,
            "name": "Peter Pan's Flight",
            "type": "Dark Ride",
            "in_queue": 0
        },
        {
            "cost": 0,
            "max_capacity": 150,
            "length": 600,
            "lng": 2.773383914736115,
            "status": "Open",
            "avg_capacity": 120,
            "description": "Board a weathered barge for a treacherous voyage to the 17th century, when rowdy rogues and
                ruthless rapscallions ransacked Caribbean seaport towns. Sing along as pirates serenade you with their
                anthem, “Yo Ho, Yo Ho (A Pirate’s Life for Me).”\nSail past haunted Dead Man’s Cove. Navigate cannon fire
```

Figure 8 - The result of the 'get-attractions' function in Postman

8.1.1.2 Android Application

On the Android side of things, three major dependencies were added to the application: Retrofit2 to handle the REST API requests, the underlying OkHttp that Retrofit2 relies on, and Kotlinx Serialization to send and decode JSON bodies to the REST API. Retrofit2 was chosen for my former familiarity with using the library before and its ease-of-use.

Before the API requests could be implemented into the application, a separation the UI and Data layers needed to occur. In Jetpack Compose, the application should be split into layers to control the flow of data and better prepare the foundation of the app for future development (Android, n.d.). The UI layer controls the visuals of the app – how the app looks and interprets user input – and the Data layer handles the processing of data to and from the UI layer (Figure 9). Its important for the concerns of both layers to be separated from each other, as failure to do so can result in issues when attempting to expand the application later on in development. The base prototype application did not have this separation of the layers, and as such had to be done to ensure smoother development later on.

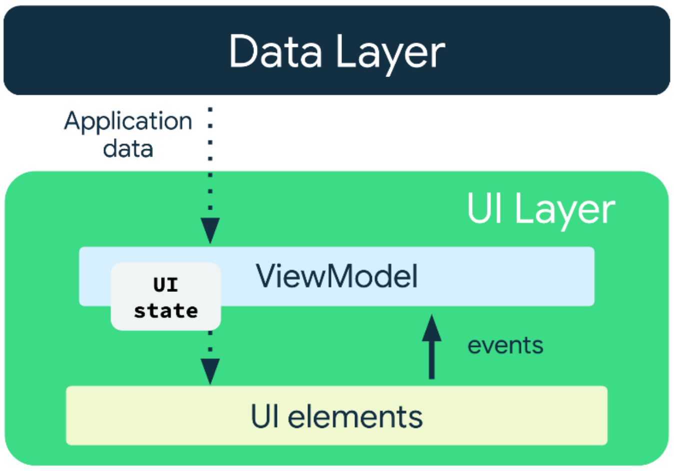
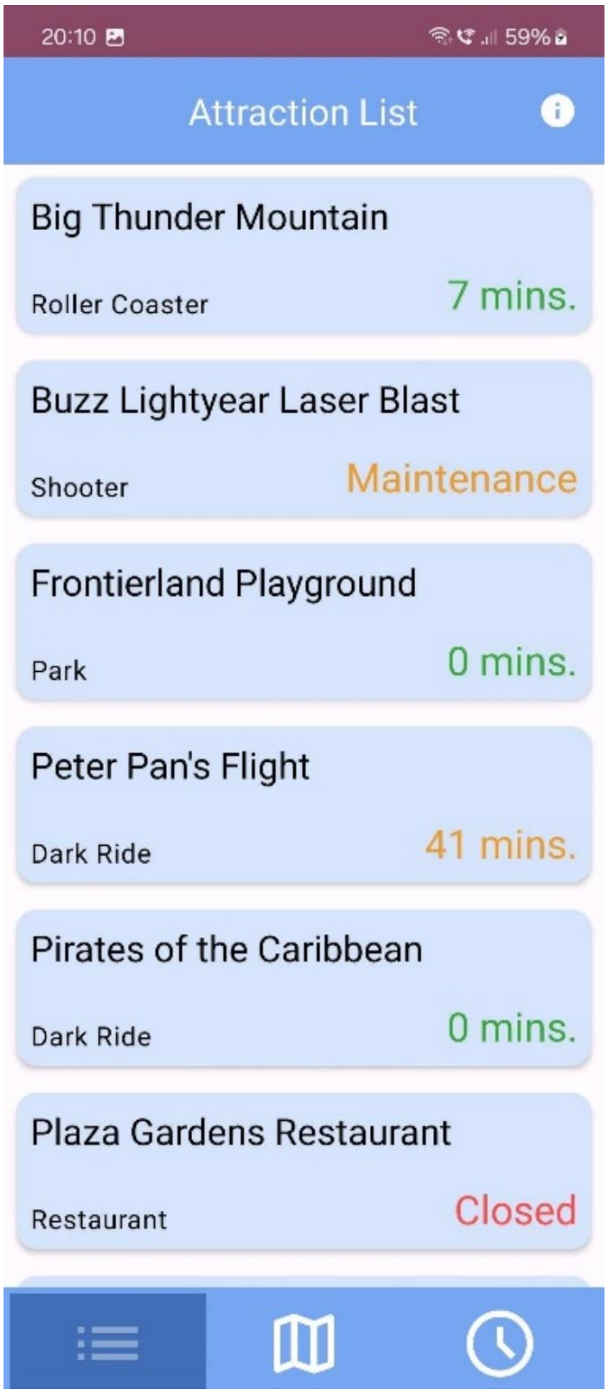


Figure 9 - The separation of concerns user UI and Data layers

Once the UI and Data layers were appropriately split from one another, I began work on adding the API request to the app. A dependency for the OkHttp Logging Interceptor was also added at this point to assist with debugging the API requests and crashes related to it. With the attraction data class modified to match the format obtained from the API request and the data handled in the appropriate ViewModel, I was then able to work on the UI development of displaying the facility’s attractions as a list of cards, with each attractions name, type, and current status displayed. This status would later be replaced with estimated queue time for open attraction, as can be seen in Figure 10.



Big Thunder Mountain
Roller Coaster
7 mins.

Buzz Lightyear Laser Blast
Shooter
Maintenance

Frontierland Playground
Park
0 mins.

Peter Pan's Flight
Dark Ride
41 mins.

Pirates of the Caribbean
Dark Ride
0 mins.

Plaza Gardens Restaurant
Restaurant
Closed

Figure 10 - The Attraction List in the Android application

8.1.2 Attraction Map

Displaying the attractions in the facility on a map was not too difficult to do once the API request was in place, as I already had an attraction map displaying the locations of hardcoded attractions in place from the prototype application. All that needed to be done was to use the data obtained from the API instead of the hardcoded data, which was removed at this point.

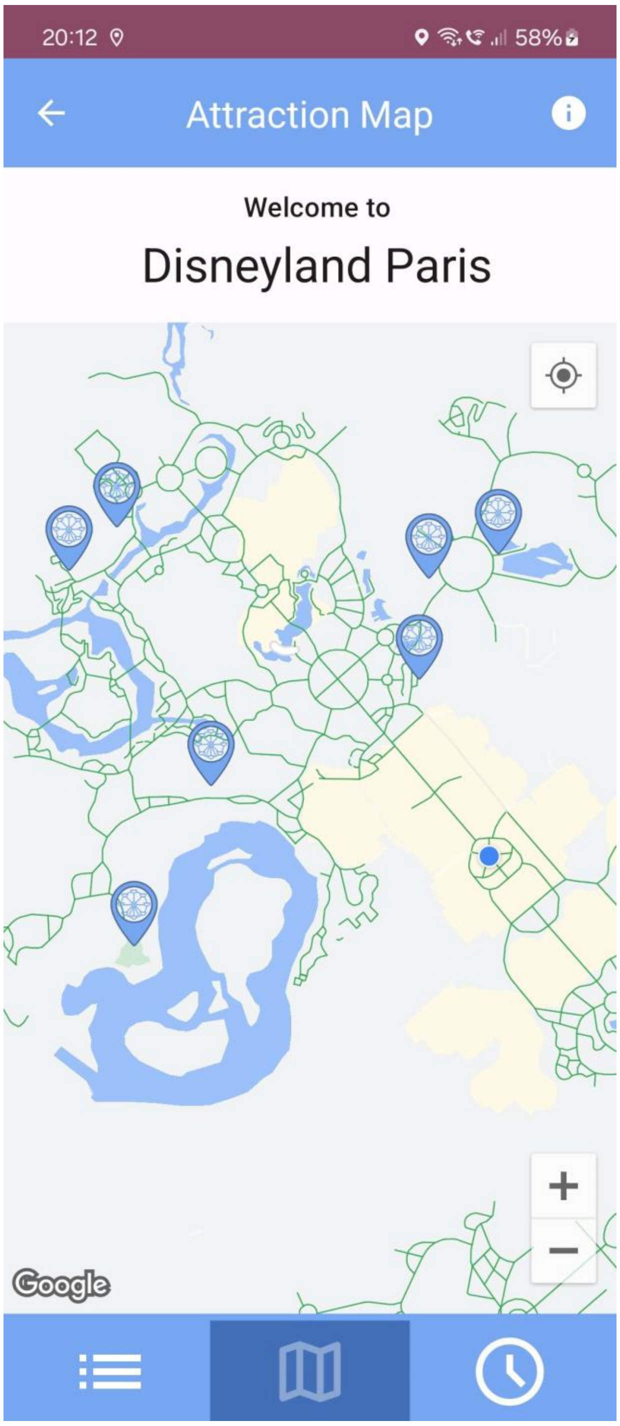
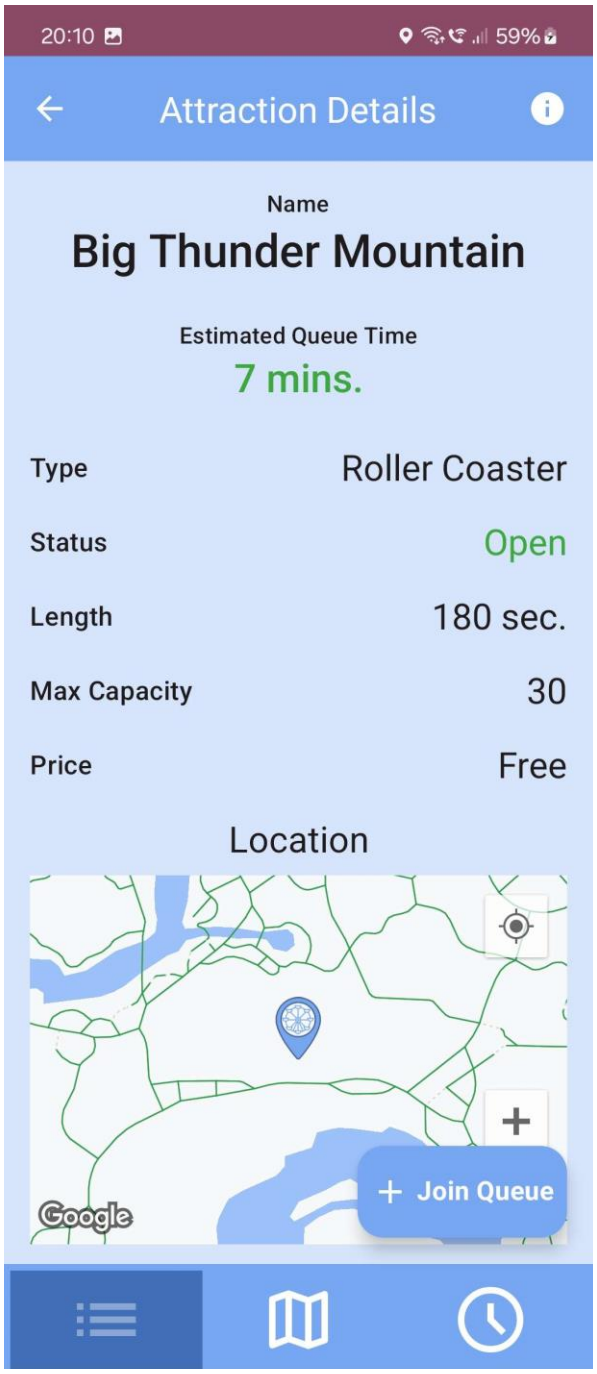


Figure 11 - The Attraction Map in the Android application

8.1.3 Attraction Details

After the attraction list was created, I worked next on creating a screen to display all of the information about a specific attraction, as the attraction list only displays the name, type, and status (later estimated queue time). Since the data was already retrieved, all that needed to be done was to create a view to display the information in a scrollable, easy to read format. I had some difficulty with navigating to a screen with an argument parameter (attraction ID) in the navigation route using the NavController but managed to get the problem sorted with debugging and logs. It was also at this point that a change was made to the attraction data class, *attractions-table*, and the associated Lambda functions to change the value type of attraction length from a float to and integer to better facilitate future use in queue time estimation.



Name

Big Thunder Mountain

Estimated Queue Time

7 mins.

Type	Roller Coaster
Status	Open
Length	180 sec.
Max Capacity	30
Price	Free

Figure 12 - The Attraction Details screen in the Android application

8.2 Queueing

The objective of this feature-sprint was to implement the virtual queueing system and all the required components of the system: estimated queue times; the ability for a user to join, leave, and check the status of their queues; notifications to update the use about their active queue entries and sending the first 'go to attraction' notification based on the user's current location; and the ability for a user's queue to be completed through an attraction entrance ticket. The implementation and completion of this feature was vital to the development of the project and for meeting the specifications of the functional requirements and defined minimum viable product. Figure 8 below shows the final flow and logic of the virtual queuing system.

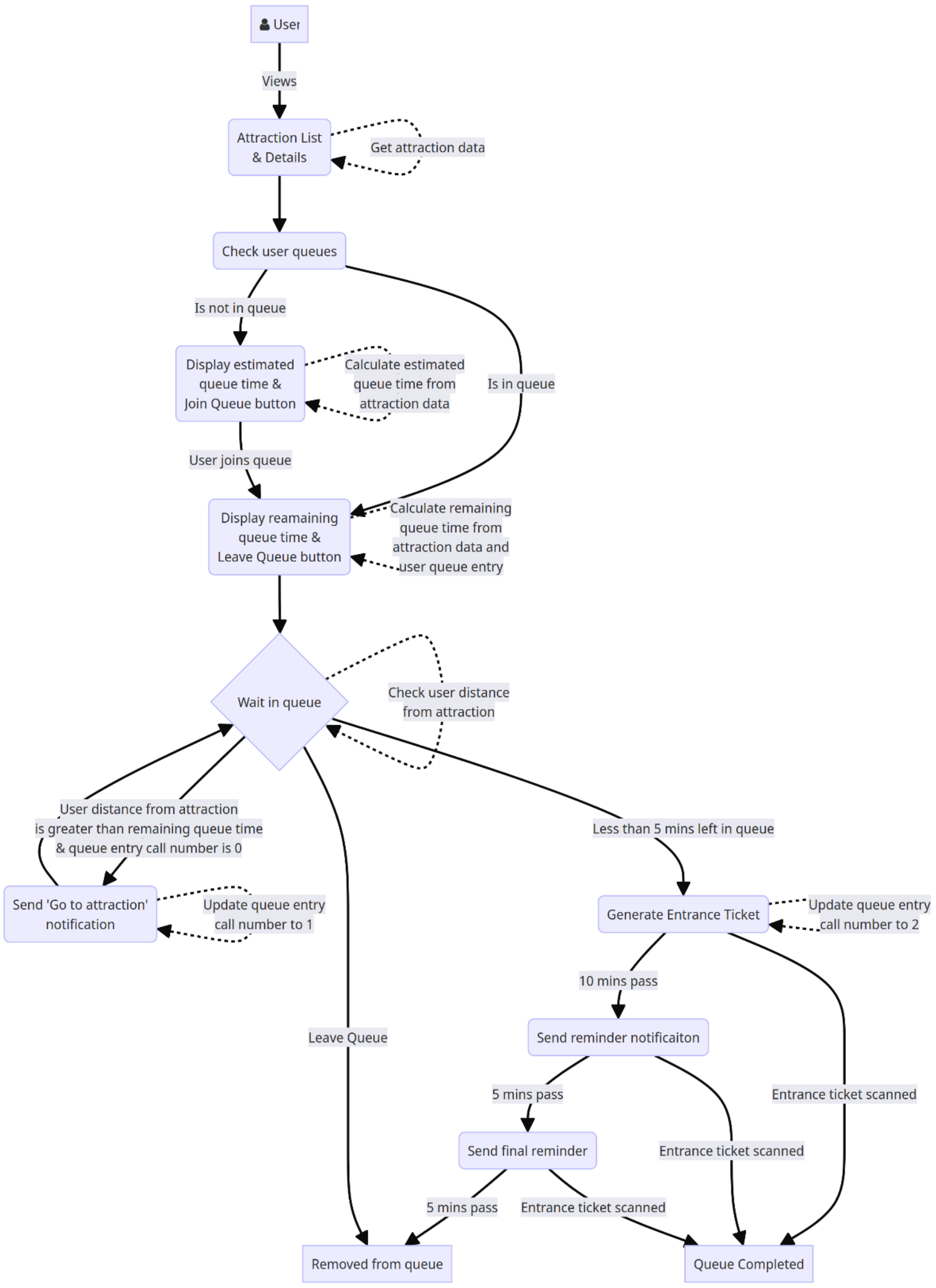

User
Views
Attraction List & Details
Get attraction data
Check user queues
Is not in queue
Is in queue
Display estimated queue time & Join Queue button
Calculate estimated queue time from attraction data
User joins queue
Display reamaining queue time & Leave Queue button
Calculate remaining queue time from attraction data and user queue entry
Wait in queue
Check user distance from attraction
User distance from attraction is greater than remaining queue time & queue entry call number is 0
Send 'Go to attraction' notification
Update queue entry call number to 1
Less than 5 mins left in queue
Generate Entrance Ticket
Update queue entry call number to 2
Leave Queue
10 mins pass
Send reminder notificaiton
5 mins pass
Send final reminder
Entrance ticket scanned
Entrance ticket scanned
Entrance ticket scanned
5 mins pass
Removed from queue
Queue Completed

Figure 13 - Queueing flow diagram and logic

8.2.1 Join and Leave Queue

The ability to join and leaves queues are baseline vital functionality to the virtual queueing system. The implementation of these features required changes to both the backend AWS infrastructure and the frontend Android application.

8.2.1.1 Amazon Web Services

In AWS, a new DynamoDB table was created to store information about queues, *queues-table*. For simplicity of configuration and deployment, a single table approach was chosen with the attraction ID and user IDs being combined together as the partition and sort keys to create a composite key that allows for each user to queue for any attraction – but only once per attraction. The REST API was expanded with new endpoints and connected Lambda functions: *join-queue* to enter a specified user ID into a queue for an attraction, and *leave-queue* to remove a queue entry. The *get-attractions* function was also updated to check the *queues-table* for queue entries for each attraction, sum the number of entries, and append the result to each attraction. This was important for the later development estimated queue times, as this value is required for the calculation.

Two admin-only functions, *add-test-traffic* and *remove-test-traffic* were also added at this point to add and remove entries to the *queues-table* en masse. The functions were vital for testing later queuing features, such as the estimated queue time and the notifications.

8.2.1.2 Android Application

In the Android application, changes had to be made to the attraction data class to account for the new *in_queue* attribute that tracks how many users are in queue for each attraction. The new requests were added to the Retrofit service, which were then hooked up to the appropriate ViewModels to be callable in the user interface. A button was added to the attraction details screen that allows for the user to join a queue, sending a request to the REST API and adding their entry to the *queues-table*.

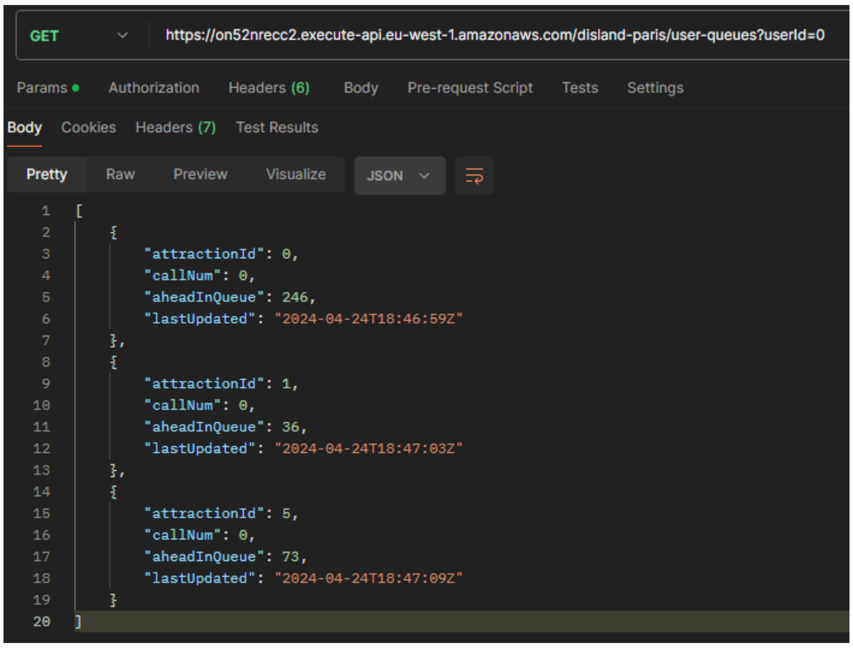
However, at this point there was no way to check what attraction queues a user was currently in, meaning that button could not be flipped to a 'leave queue' button after the user has joined an attraction queue. The ability to get a specific user's queues had to be implemented before this functionality could be completed.

8.2.2 Your Queues

To complete the previous functionality and to display to the user what queues they have currently entered, a way to get only the queue entries a user has entered into was required. This required the creation of a new resource in the cloud along with new screens and logic in the Android application to add the 'your queues' screen and complete the join and leave queue functionality.

8.2.2.1 Amazon Web Services

In AWS, one new Lambda function and corresponding API endpoint was created: *get-user-queues*. This function takes in a parameter of the user ID and gets all queue entries for that user in the *queues-table*. The Global Secondary Index *user-queues-index*, discussed in Section 4.3.4.2, was created and used as part of this function to ensure that the table is able to be queried by just the user ID. At this point, the request returned a list of queue entries for the user, specifying the attraction ID and the number of people ahead of the user in the attraction queue, but two other variables were added later on as can be seen in Figure 14.


GET https://on52nrecc2.execute-api.eu-west-1.amazonaws.com/disland-paris/user-queues?userId=0

```
[
    {
        "attractionId": 0,
        "callNum": 0,
        "aheadInQueue": 246,
        "lastUpdated": "2024-04-24T18:46:59Z"
    },
    {
        "attractionId": 1,
        "callNum": 0,
        "aheadInQueue": 36,
        "lastUpdated": "2024-04-24T18:47:03Z"
    },
    {
        "attractionId": 5,
        "callNum": 0,
        "aheadInQueue": 73,
        "lastUpdated": "2024-04-24T18:47:09Z"
    }
]
```

Figure 14 - API result for 'get-user-queues' function in Postman

8.2.2.2 Android

On the Android side of things, a new data class had to be created to handle the data received from the get-user-queues request: QueueEntry. The new API request was added to the Retrofit repository and added to the ViewModels in such a way that whenever a request was made to get attraction data, the request to get user queue entries would occur immediately after. Logic was then added for the attractions screen that would find if there were any queue entries corresponding to the selected attraction's internal ID, and display the leave queue button if so. This then allowed for the user to leave queues, completing the join and leave queue functionality. A message banner was also added to the attractions list and detail screen denoting if the user has queued for that attraction.

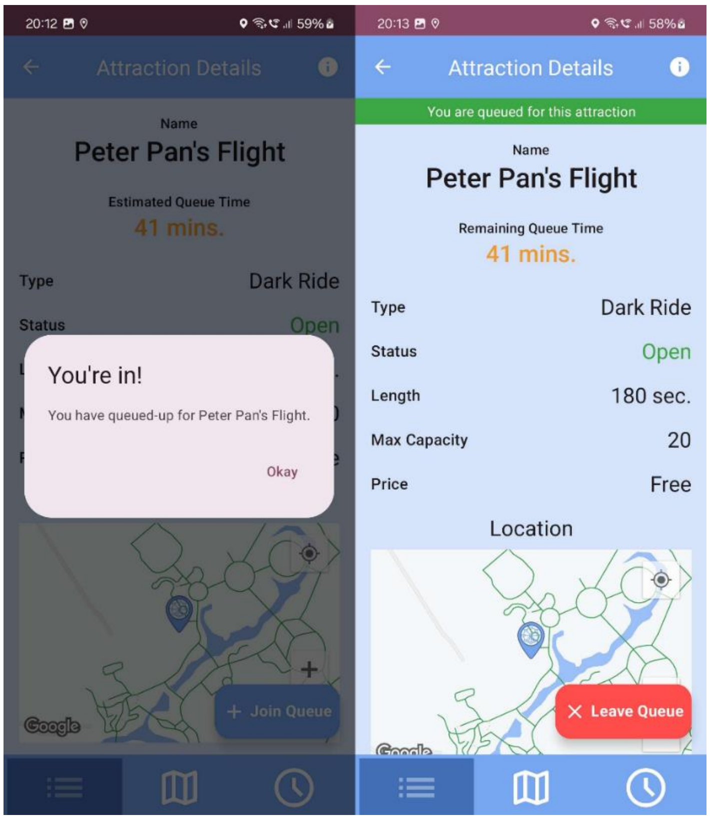


Figure 15 - Join queue alert message (left) and leave queue button (right) in the Android application

Now that the user was able to join and leave queues and their current queue entries retrievable, creating the 'your queues' screen to display all the queue entries was simple. Inversing the logic to connect queue entries to attractions from the attraction details screen so that corresponding attraction data for each queue entry was obtained, the user's queue entries were then displayed a list similar to the attraction list. Unlike the attraction list, the entries in this list are expandable, with two buttons being revealed upon expansion: 'leave queue' to leave the attraction queue, and 'attraction details' to go the corresponding attraction's details page. At this point, the estimated queue time was not implemented, meaning these entries simply displayed the attraction name.

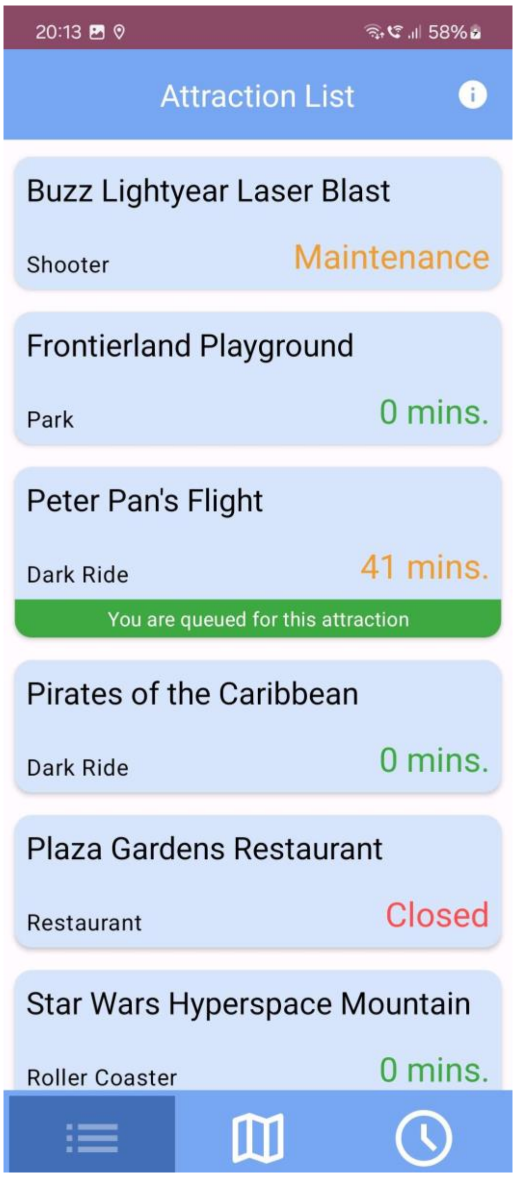


Figure 16 - Queued attractions denoted with banner on Attraction List

8.2.3 Estimated Queue Time

Estimated queue time displays how long a user entering the queue at the moment of viewing would approximately have to wait until they are entering the attraction. For this, the attraction data class and Dynamo DB table had to be updated again to include attributes for the average capacity (for the calculation) and maximum capacity (for the user's information). Once these changes were made, the logic connecting the attractions and queue entries was extended to enter in information about each attraction and the current queue status into a function to calculate the estimated queue time – or the remaining queue time if the user is queued for the attraction. The queue time calculate is as follows:

$$Queue\ Time = \frac{(Num.users\ in\ queue/Average\ capacity) * Attraction\ time}{60}$$

where:

- *Num. users in queue* is the number of users currently queued for the attraction – or number of users ahead of specified user if currently in queue for the attraction.
- *Average capacity* is the number of people that enter the attraction per run on average.
- *Attraction time* is how long the attraction takes to complete per run in seconds.
- *60* the devisor to convert the seconds of the calculation to minutes.
- *Queue Time* is the calculated estimated queue time in minutes, rounded to the nearest minute.

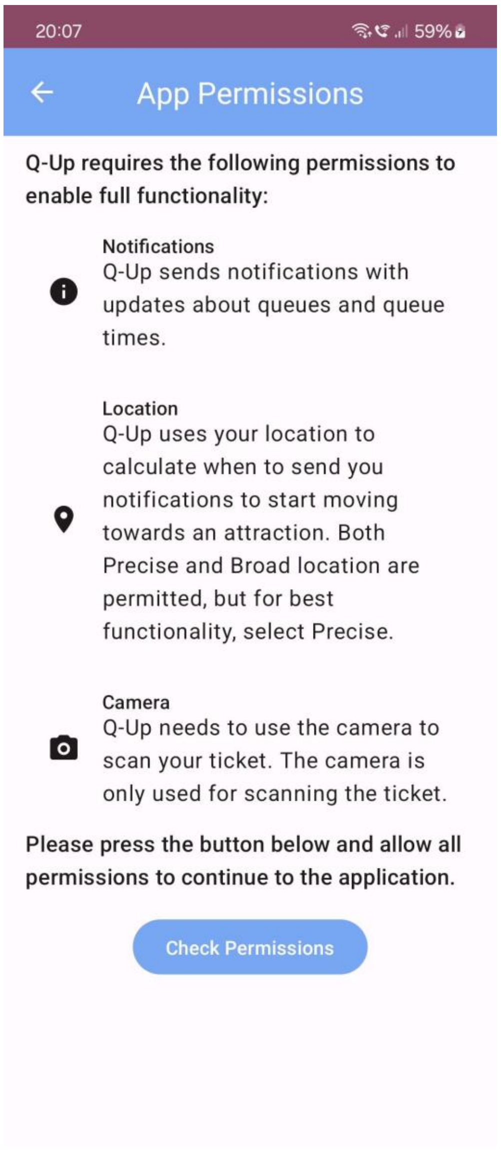
With this calculation and logic added, the estimated queue times were then added to attraction list, details, map and queues screen. If an attraction is not open (is under maintenance or closed), the estimated queue time is not displayed, with the current status displayed instead. The ability to refresh the API data by pulling down on each screen was also implemented at this point, which was vital in testing that the estimated queue time calculation was working correctly.

8.2.4 Permissions

The Android application requires multiple system permissions to perform its functionality, including notifications, user location, and the camera. In Android, permissions must be explicitly requested by the application before using them. To manage permissions before attempting to use them and make the addition of future permissions easier, a screen was added to request the required permissions. This screen is only shown if the app does not have all the permissions is needs. The permissions screen lists all the permissions required by the application, and for what purpose it needs access for.

When the 'check permissions' button is pressed, the app will request all permissions required. If all are granted, the application continues. Otherwise, if one or more permissions were not granted, the app notifies the user that one or more permissions were refused and prompts them to go to the app's settings to grant the required permissions, with a button that will take the user directly to the app's settings. Once the user returns to the app, it will automatically check if the required permissions have been granted and perform the required request from the user again if necessary.

The management of user permissions was very tricky and required extensive work to ensure all permissions were being requested correctly without the user interface logic skipping one or more of the permissions requests, and to ensure that the permissions process is user friendly and easy to understand.


20:07

59%

App Permissions

Q-Up requires the following permissions to enable full functionality:

Notifications
Q-Up sends notifications with updates about queues and queue times.

Location
Q-Up uses your location to calculate when to send you notifications to start moving towards an attraction. Both Precise and Broad location are permitted, but for best functionality, select Precise.

Camera
Q-Up needs to use the camera to scan your ticket. The camera is only used for scanning the ticket.

Please press the button below and allow all permissions to continue to the application.

Check Permissions

Figure 17 - Permissions screen in Android application

8.2.5 Notifications

Notifications are an important aspect of the Android application, used to notify the user about updates to their queue entries. To ensure that notifications were being sent correctly, with each type of notification only being sent once and after the appropriate time frames had passed, changes had to be made to the AWS backend infrastructure to facilitate the required functionality of the Android app.

8.2.5.1 Amazon Web Services

In the *queues-table*, two new attributes had to be introduced: the call number, which tracks how many times a user has been notified regarding the queue entry, and a last updated timestamp, which tracks the last time an update was made to the queue entry. Both of these values are important in ensuring that the correct notification messages are being sent at the correct times to the user. The meaning of each call number was detailed in Section 4.3.4.2.

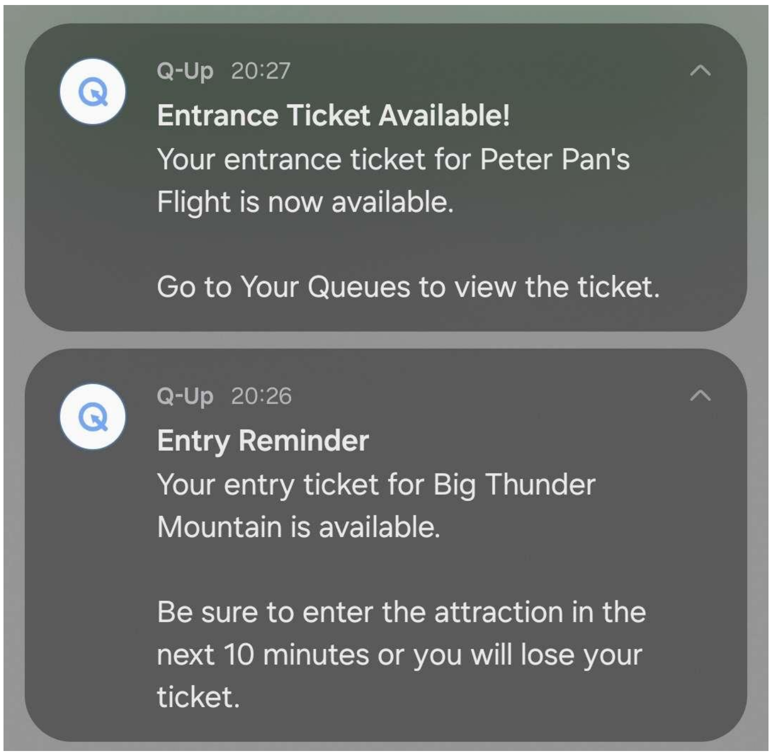
In addition to these changes, a new Lambda function and connected API endpoint was added: *update-queue-call-num*, used to update the call number of a queue entry by the Android application each time a notification was sent to the user. This function also updates the 'last updated' timestamp automatically.

8.2.5.2 Android

With the permissions screen created, the permission to send notifications to the user was added to the permissions check and the QueueEntry data class updated to account for the new fields added to the DynamoDB table. New logic was then added to the function that gets the user's current queue entries to check each entry's call number and last updated time. Depending on the call number, estimated remaining queue time, and the amount of time between now and the last call number update, different notifications are sent:

- If the amount of time the user has left to queue is less than 5 minutes and the queue's call number is 0 (hasn't been called) or 1 ('Go to Attraction' notification sent), send "Entrance Ticket Available!" notification and update call number to 2.
- If call number is 2 and time between now and last update is more than 10 minutes, send "Entry Reminder" notification and update call number to 3.
- If call number is 3 and time between now and last update is more than 5 minutes, send final "Entry Reminder" notification and update call number to 4.
- If call number is 4 and time between now and last update is more than 5 minutes, send "Ticket Expired" notification and update call number to 5.
- If call number is 5, remove user from attraction queue.

These checks are essential to ensure that the user receives the correct notifications for their queue status and also does not receive the same notification more than once. Tapping the notifications will bring the app to the foreground in the last state it was open in. At this point, the 'Go to Attraction' notification was not implemented yet as the location permissions had not been requested, and the attraction entrance tickets were not generated yet.



Q-Up 20:27

Entrance Ticket Available!

Your entrance ticket for Peter Pan's Flight is now available.

Go to Your Queues to view the ticket.

Q-Up 20:26

Entry Reminder

Your entry ticket for Big Thunder Mountain is available.

Be sure to enter the attraction in the next 10 minutes or you will lose your ticket.

Figure 18 - Two different types of notifications in the Android application

8.2.6 Location-Based ‘Go to Attraction’ Notification

The location based ‘Go to Attraction’ notification is a notification sent to the user depending on distance from the attraction, with the aim that once they receive the notification and start walking towards the attraction, they will arrive at the attraction as their attraction entrance ticket becomes available.

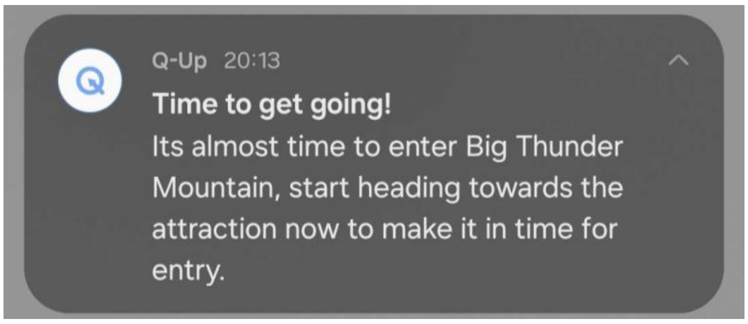
To implement this, location permissions had to be added to the permissions screen and checks, and a function created in the appropriate ViewModels to get the users current location on request. The app allows for the user to permit either precise or approximate location data but advises the user to allow for precise location data for improved functionality. This function call was then added to the user queues and notifications logic, getting the user’s current location at the time the logic is processed. The user’s current distance from the attraction is calculated as such:

$$Distance\ from\ attraction\ in\ mins = \frac{Distance\ from\ attraction}{100} * 2$$

where:

- *Distance from attraction* is the straight-line distance between the user and the attraction in meters.
- *Distance from attraction in mins* is the estimation of the number of minutes it would take for the user to walk to the attraction. This is based on an assumption that the average adult walks at a speed of 3 miles per hour (Cronkleton & Bubnis, 2019), meaning they would complete 100 meters in 70 seconds. Considering that in most facilities, a visitor would be unable to walk in a straight line to their destination, this assumption is upped to 120 seconds per 100 straight-line meters, or 2 minutes per 100 meters. The result of this calculation is then rounded to the nearest minute.

The notification logic calculates the distance from the attraction in minutes, then adds it to a base of five minutes to determine what estimated queue time must be remaining for the ‘Go to attraction’ notification to be sent. For example, if a user is 400 meters away from an attraction, they are calculated to be eight minutes away from the attraction. Added to a base of five, this results in 13 minutes. If the user has less than 13 minutes left in their queue, the ‘Go to attraction’ notification is sent. This calculation is performed each time the app data is refresh, ensuring that the user’s current location is always being used.


Q-Up 20:13

Time to get going!

Its almost time to enter Big Thunder Mountain, start heading towards the attraction now to make it in time for entry.

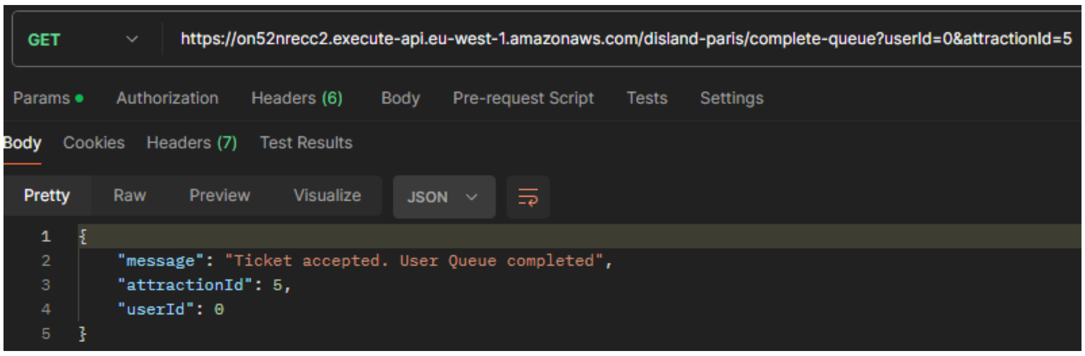
Figure 19 - 'Go to Attraction' notification in Android application

8.2.7 Attraction Entrance Ticket

The attraction entrance ticket is a QR code generated in the Android application that permits the user entry to an attraction they have queued for. It was the final component required for the virtual queuing system and completed the minimum viable product. This feature required one new resource in AWS, along with a new dependency and screen in the Android application.

8.2.7.1 Amazon Web Services

One new Lambda function was created for the completion of attraction queues, *complete-queue*. The endpoint for this function is embedded in the attraction entrance ticket QR codes along with the attraction ID the queue entry is for and the ID of the user. The endpoint is accessed and function triggered upon scan of the QR code, which checks the queue entry to ensure that it has not expired (its call number is not equal to 5), and returns a response, denoting that the ticket has been accepted. The user's queue entry is then removed from the queues table.

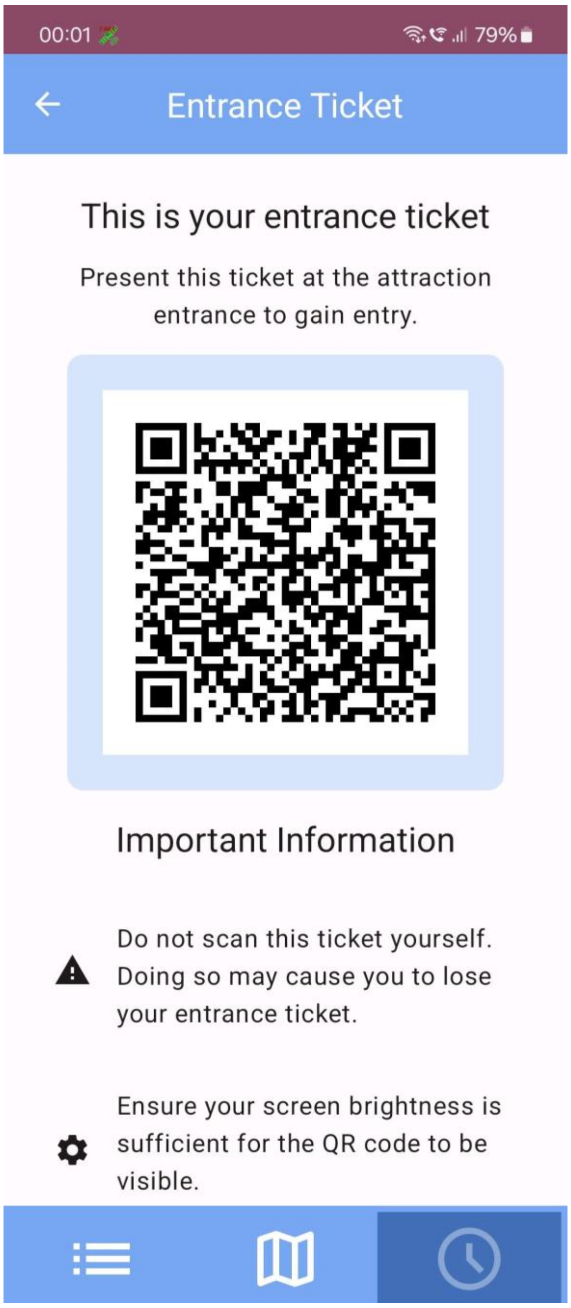


```
{
    "message": "Ticket accepted. User Queue completed",
    "attractionId": 5,
    "userId": 0
}
```

Figure 20 - API response of 'complete-queue' function in Postman

8.2.7.2 Android

In Android, a new dependency was added for the creation of the QR codes: ZXing. When a queue entry has a call number greater than 2 and less than 5, a banner with the message "Entrance ticket available" is displayed on the queue list item, with a new button appearing in the expandable menu to view the entrance ticket. This button takes the user to the ticket screen, where their attraction entrance ticket is displayed. A check is performed to ensure that the queue entry is still valid, then the QR code generated with the API endpoint, attraction ID and user ID embedded onto it.


00:01 79%

Entrance Ticket

This is your entrance ticket

Present this ticket at the attraction entrance to gain entry.

Important Information

Do not scan this ticket yourself. Doing so may cause you to lose your entrance ticket.

Ensure your screen brightness is sufficient for the QR code to be visible.

Figure 21 - Attraction entrance ticket in Android application

Ideally, these tickets would be scanned by a companion application used by facility staff to ensure that the tickets are valid and for the correction attraction. Due to the lack of hardware and the focus of this project being on the singular full-stack application, the tickets are currently scannable by anyone by a standard phone camera.

8.2.8 Auto-Refresh

To ensure that attraction data and queue time are up-to-date, the app regularly performs the API requests while open. This occurs every 60 seconds and ensures that information, particularly that which is related to the queue times, is always reasonably up-to-date.

It was originally planned for a foreground service to be utilized so the app could refresh and update information even while the app was in the background, but to implement this service a major restructuring of how the application handles data would have been required, which was not possible at this point in the project's development due to time constraints.

8.3 Anonymous Login

The objective of this feature sprint was to implement the final functional requirement of the application allowed for users to check-in to a facility. The non-functional requirements defined that this should be done anonymously, without an email or other form of sign up required, ideally through a QR code or NFC scan. Figure 21 below shows the final flow of how a user anonymously checks-in to a facility.

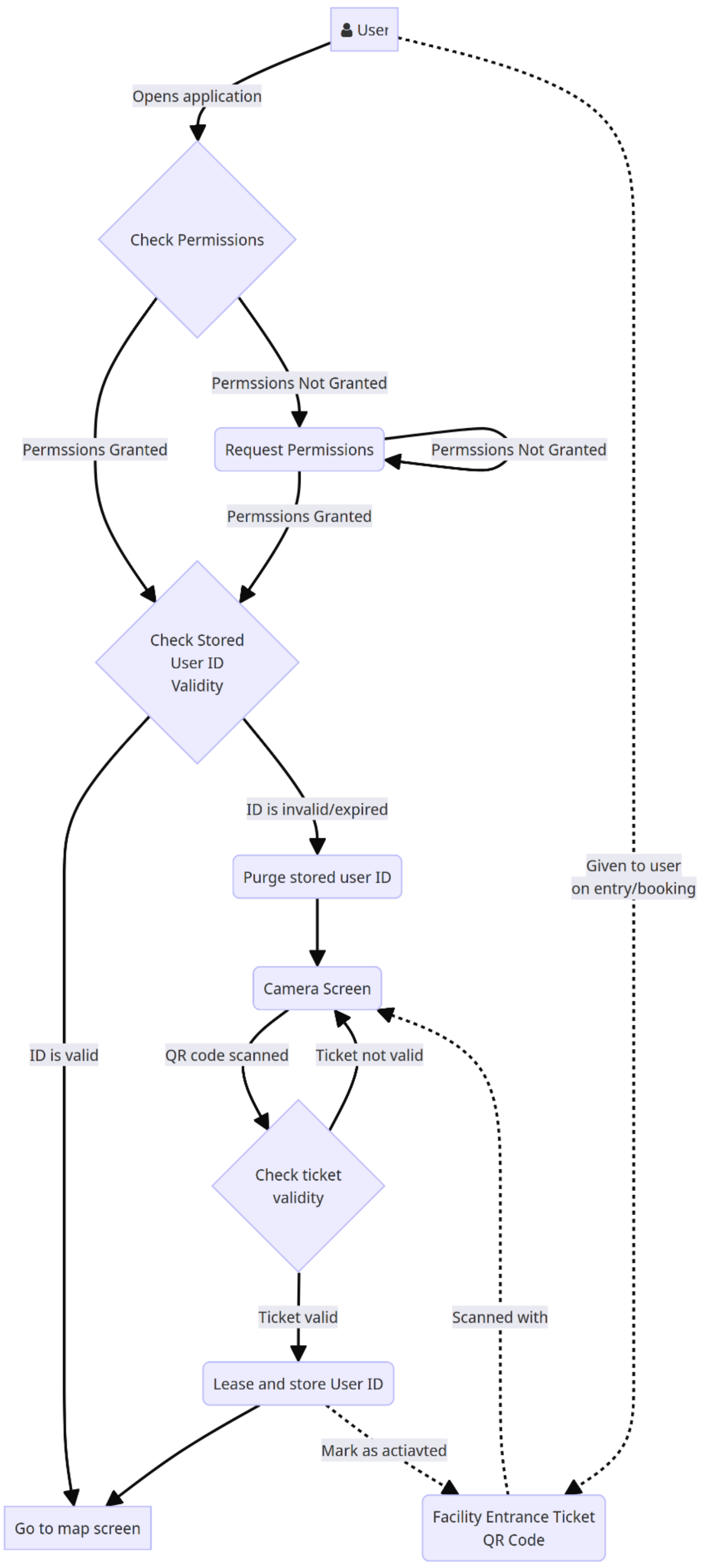
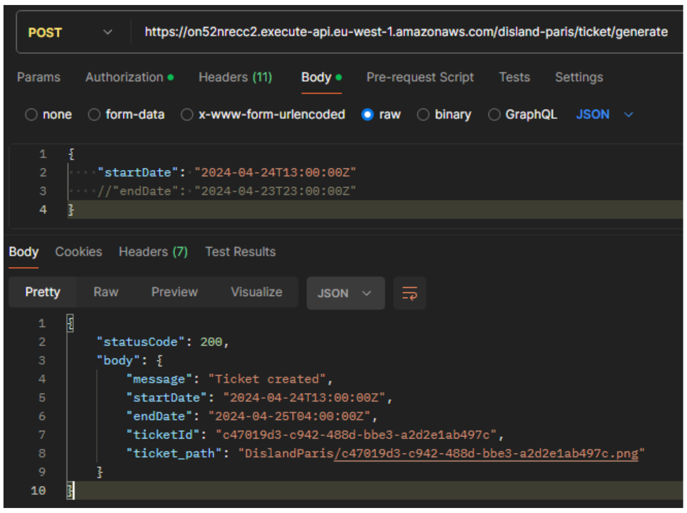


Figure 22 - Flow diagram of how user logs in to facility anonymously

8.3.1 Facility Entrance Ticket QR Codes

The facility entrance ticket QR codes are codes distributed to users that allow them to access the virtual queueing system. These codes are to be created by the facility when a user enters the facility or creates their booking with them. The QR codes are embedded with the endpoint required to check the validity of the ticket, as described in Section 8.3.2 along with the ticket ID. To create these tickets, a new admin-only API endpoint and function was created, *generate-ticket*, which generates the ticket using a Universally Unique Identifier (UUID) to ensure each ticket's ID is unique from one another, along with activation and expiry times that mark when the ticket becomes available and no longer available to use. These values can be specified when the ticket is created, or can be automatically created, with an activation time of the current time and an expiry time of 15 hours afterwards by default.



POST https://on52nrecc2.execute-api.eu-west-1.amazonaws.com/disland-paris/ticket/generate

```
{
    "startDate": "2024-04-24T13:00:00Z"
    //"endDate": "2024-04-23T23:00:00Z"
}
```

Body (Pretty, JSON)

```
{
    "statusCode": 200,
    "body": {
        "message": "Ticket created",
        "startDate": "2024-04-24T13:00:00Z",
        "endDate": "2024-04-25T04:00:00Z",
        "ticketId": "c47019d3-c942-488d-bbe3-a2d2e1ab497c",
        "ticket_path": "DislandParis/c47019d3-c942-488d-bbe3-a2d2e1ab497c.png"
    }
}
```

Figure 23 - API result of 'generate-ticket' function in Postman and the generated QR code

To store the generated tickets, an S3 bucket was created with a directory for each facility containing the facility entrance ticket QR codes for that facility, along with a DynamoDB table to store ticket information, *tickets-table*.

8.3.2 Camera and QR Scanner

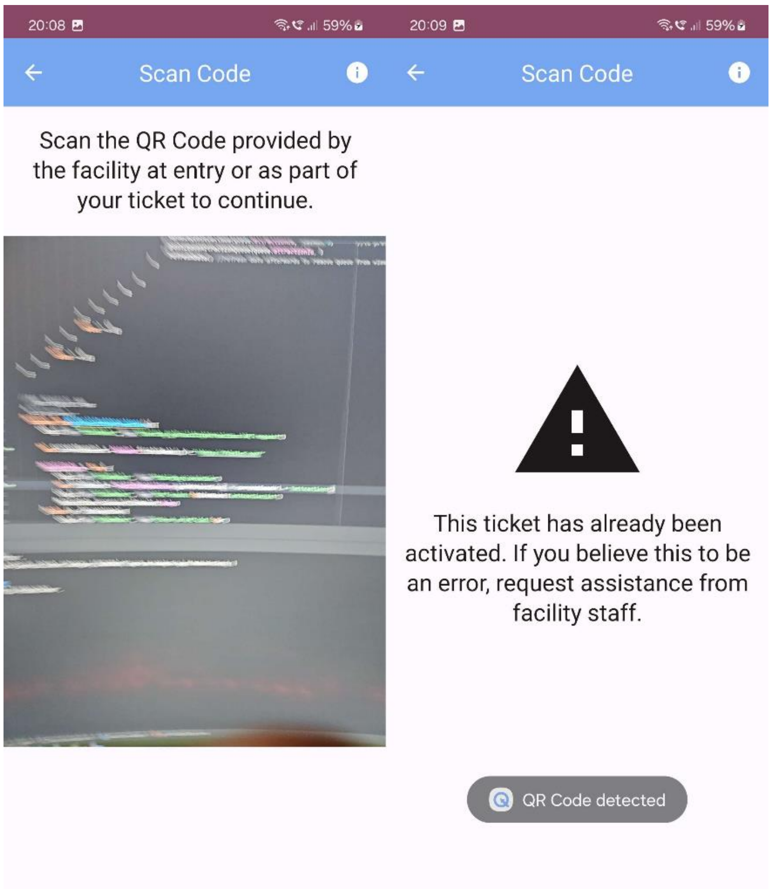
The camera and QR scanner are important for the ability to scan the facility entrance ticket QR codes. The development of this feature primarily concerned the Android app, but a new API endpoint had to also be created to ensure full functionality.

8.3.2.1 Amazon Web Services

One new API endpoint and Lambda function was created to check the validity of scanned tickets: *check-ticket*. This takes in a ticket ID from the corresponding entry in the *tickets-table*, checks to see if the current time is before the ticket's expiry date, after the ticket's activation data, and if the ticket has been activated or not. Depending on the result, a different status code is sent, denoting different the different reasons the ticket is invalid.

8.3.2.2 Android

Camera permissions were added to the permissions check in the Android app, along with the Google MLKit Barcode Scanning library as a dependency to create the QR scanner. Opening the camera screen, a live feed of the device's back camera is displayed. When a QR code is detected, the camera feed pauses, and the URL obtained from the facility entrance ticker QR code used to execute the ticket validity check. If the result is successful, the app continues on to the map screen. Otherwise, an error is displayed with a message denoting why the ticket is invalid.



Scan the QR Code provided by the facility at entry or as part of your ticket to continue.

This ticket has already been activated. If you believe this to be an error, request assistance from facility staff.

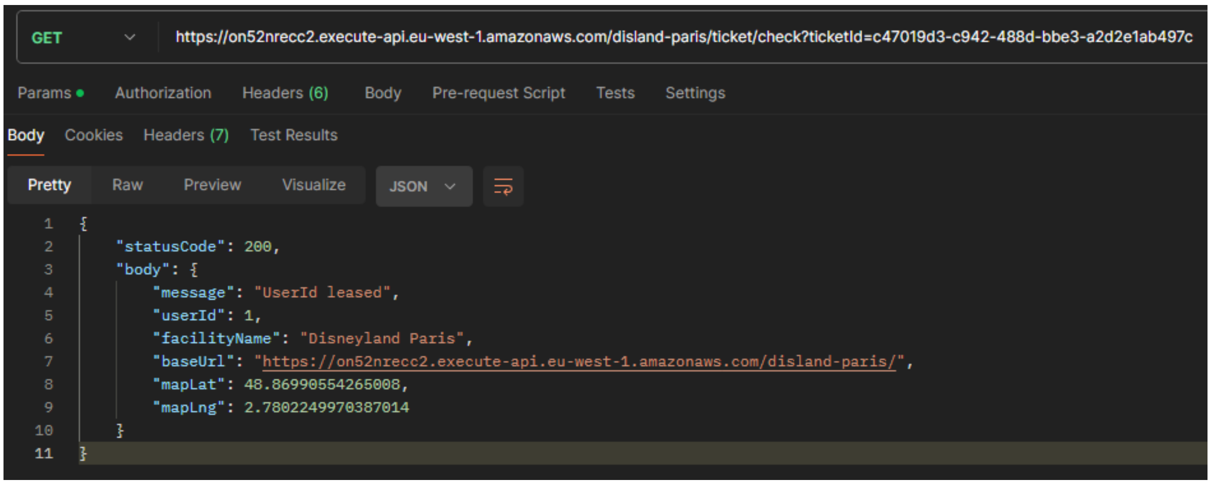
Figure 24 - The camera screen (left) and a 'ticket already activated' error (right) in the Android application

8.3.3 Leased User ID

Now that the tickets can be generated and scanned, the next major step towards completing the anonymous login functionality was to leased user IDs to users when they successfully scan a valid ticket. As with many features before, this involved the creation of the final required AWS resources, along with some new logic in the Android application.

8.3.3.1 Amazon Web Services

Firstly, a new DynamoDB table was created to store leased user IDs: *users-table*. The *check-ticket* Lambda function was altered to, upon the successful validation of a ticket, generate a new user ID, place this new entry into the *users-table* along with the time it was created at, the time the lease expires, and the ticket ID that the user ID was leased from. To generate the user ID, the function checks the *users-table* for the current highest user ID, and increments by one to lease the next user ID. The function then returns the leased user ID, along with facility information, such as the facility name, the base URL for making requests for information in the facility, and the map coordinates for displaying the in-app maps.

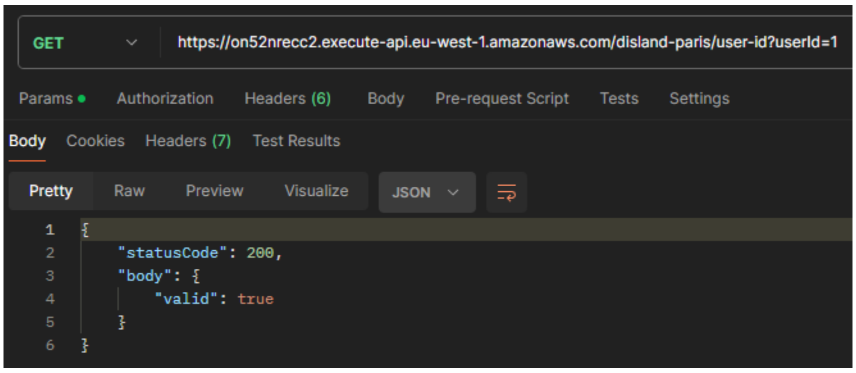


```
https://on52nrecc2.execute-api.eu-west-1.amazonaws.com/disland-paris/ticket/check?ticketId=c47019d3-c942-488d-bbe3-a2d2e1ab497c
```

```
{
    "statusCode": 200,
    "body": {
        "message": "UserId leased",
        "userId": 1,
        "facilityName": "Disneyland Paris",
        "baseUrl": "https://on52nrecc2.execute-api.eu-west-1.amazonaws.com/disland-paris/",
        "mapLat": 48.86990554265008,
        "mapLng": 2.7802249970387014
    }
}
```

Figure 25 - API response of 'check-ticket' function for valid ticket in Postman

The final Lambda function and API endpoint, *check-user-id-valid*, checks if an input user ID is valid. It does this by retrieving the corresponding user ID lease from the *users-table*, and checking if the ID firstly still exists and that the current time is not past the expiration time. If one of these conditions fail, the endpoint returns ‘false’. Otherwise, if the ID is valid, it returns ‘true’.



```
https://on52nrecc2.execute-api.eu-west-1.amazonaws.com/disland-paris/user-id?userId=1
```

```
{
    "statusCode": 200,
    "body": {
        "valid": true
    }
}
```

Figure 26 - API response of 'check-user-id-valid' for a valid user ID in Postman

8.3.3.2 Android

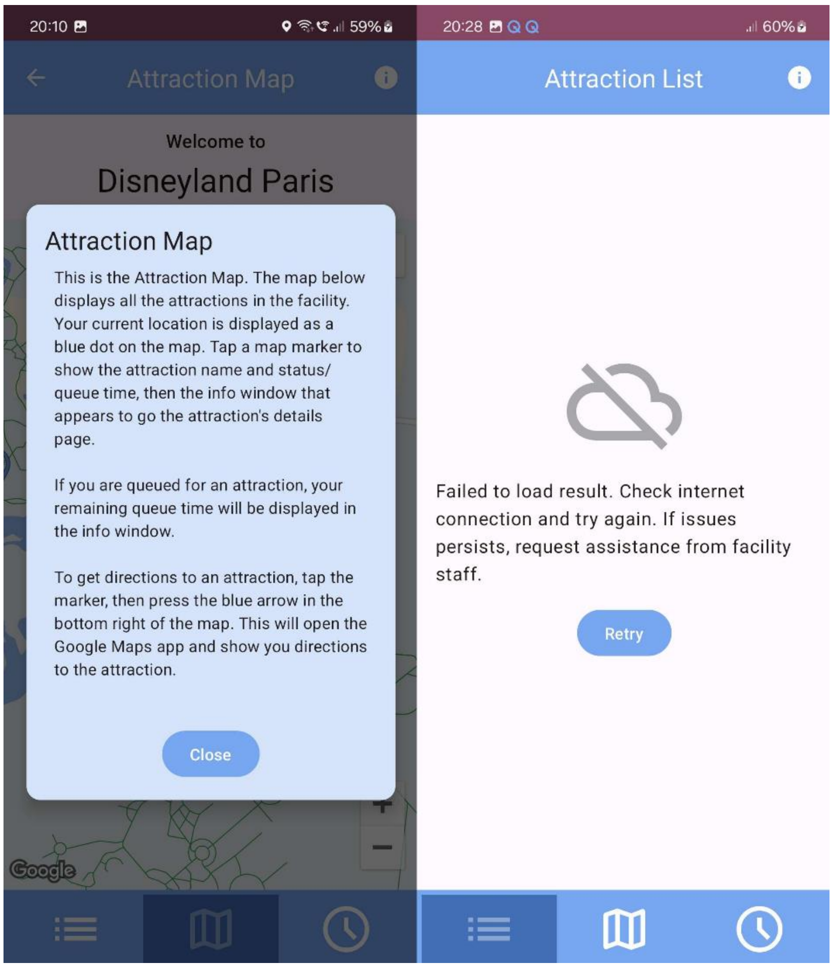
In the Android application, a new request was added to the Retrofit service along with the response body. When the request is made after scanning a QR code, the application receives the response data, and uses the received user ID, base URL, and other facility information to display data and make requests. However, this data was not persisted in any way at this point and was lost when the application is closed – not ideal with one-time use entry codes and queues tied to the user IDs. Before this feature sprint could be completed, persistence would have to be added to the application.

8.3.4 DataStore

To ensure that leased user ID and facility information are not lost upon the app being closed, a preferences DataStore was introduced to store and persist the information received from the *check-ticket* request after the facility entrance ticket QR code was scanned. To ensure that expired or invalid credentials are not being held on to, the app checks the validity of the stored user ID each time it is launched with the *check-user-id-valid* endpoint. If the user ID is still valid, the app continues to the map screen as normal. If tje lease on the ID has expired or the ID no longer exists, the app purges the stored data in the DataStore, returning them to their initial default values, and goes to the camera screen to scan a new facility ticket entrance ticket.

8.4 Quality of Life

Now that all the functionality requirements of the application were met, one final feature-sprint was conducted to make miscellaneous improvements to the user experience of the Android application. Improvements in this sprint included the addition of an information button on each view to display information about purpose of the screen and its functionality, the ability to refresh the app data after a network error has occurred (Figure 27), improvements to how the app navigates between screens, displaying the users current location on the map, along various improvements to the underlying code.



Attraction Map

This is the Attraction Map. The map below displays all the attractions in the facility. Your current location is displayed as a blue dot on the map. Tap a map marker to show the attraction name and status/ queue time, then the info window that appears to go the attraction's details page.

If you are queued for an attraction, your remaining queue time will be displayed in the info window.

To get directions to an attraction, tap the marker, then press the blue arrow in the bottom right of the map. This will open the Google Maps app and show you directions to the attraction.

Failed to load result. Check internet connection and try again. If issues persists, request assistance from facility staff.

Figure 27 - The information pop-up (left) and the network error message (right) in the Android application

9. Future Project Developments

From this point, there is much potential for future developments of the Q-Up project. There are a number of features that I would have liked to have implemented but was not able to due to time constraints, and aspects of the implementation that would need to be improved or developed were this application to go to market. The following is a list of both those features and additional functionality required to get the app ready for business:

- A companion application for use by facility staff and a web portal for facilities to managed facility data and view queue information. The current application is very user focused in its design, and only offers simple and non-user-friendly ways to access and update information. A full companion app that allows for facilities to update attraction information and statuses, add new attractions, and send facility-wide notifications to users would be required for the deployment of this application in an attraction-based facility long term.

- Analytics about visitor habits and queue times. Currently, the average attraction capacity is a hardcoded static value that does not take into account the volume of visitors to a facility or attraction. This would ideally be dynamic and based on visitor flow and patterns, along with providing other analytics about visitor behaviour and queuing in the facility.

- Allowing users to join queues as groups. Currently, if a group of people are attending the facility together, there is no way for them to all be in the same queue as a unit together. Each visitor in the group has to queue separately with their own queue entry, which is not ideal for large groups attempting to enter a busy attraction together or for children who do not own a phone. Functionality that allows for users to enter a group together with just a singular queue entry and attraction entrance ticket would be ideal for facilities that often see large groups of people attending together or many parents with children, such as amusement parks.

- While the application is generic and can be used for any number of facilities, further functionality to allow for facilities to theme the attraction with colours, assets, and terminology would help greatly with the single-app, software-as-a-service approach the application is striving to achieve.

- New features and improvements for the application such as using a foreground service to allow the app to perform data refreshing in the background, a search bar that allows the user to search for attractions in the facility and filter by attraction type, and attraction recommendations for nearby attractions that can be queued for and enjoyed while waiting in queue for another attraction would all go a long way in making the application more useful with better functionality for the users. All of these features were original planned for the project but were cut due to time constraints.

While I am happy with how the final project turned out, many of the developments listed above would be great improvements for the project as a whole, and ultimately would be required were the application to go to market and try to compete against current competitors in the virtual queueing space.

10. Conclusion

Throughout the planning and development of this application and the experience of the Final Year Project, I have learned much about software development and project planning. Being in the Cloud and Networks stream provided me with a large basis in cloud computing, and this project allowed me to combine that knowledge and desire to work with cloud resources with the general programming, design, and project skills obtained over my four years in this course. I obtained a lot of valuable experience in full-stack software development: working from the creation of the backend database and compute resources in the cloud, the frontend Android application – which in of itself is comprised of its own frontend user interface and backend data processing and storage – and creating the REST API that allows for the two main aspects of the application to communicate with each other.

While much of this project was based in areas I had prior experience with, it also forced me to work with entirely new areas, such as Android permissions management, Google Maps and user location, and Agile development with a Kanban board. While I was familiar with Agile development methodologies, I have gained a newfound appreciation for Agile and the benefits it brings to project management. The process helped to keep the project on track and focused on getting the major features and functionality implemented.

I ran into several setbacks throughout the development process that caused time to be lost on solutions or alternatives. Some of these setbacks occurred too late in the development process to provide proper solutions for and had to be abandoned. Such was the case for the background application refreshing. From this, I learned a valuable lesson about de-risking application features, particularly those that I have no prior experience of. I also learned a lesson in managing project scope and feature creep. As the nature of this application hands itself to the addition of many features to improve the user and facility operator experiences, a careful eye had to be kept on the scope of these features to ensure that development remained on-track towards an application that provided a virtual queuing system first and foremost.

Overall, the development of the Q-Up application and process of the Final Year project have been extremely valuable for the development of my programming, design, and project management skills, and I believe will serve me well as a great experience as I begin my career in the tech industry.

11. References

accesso. (n.d.). *A Better Way to Wait* . Retrieved from accesso: https://www.accesso.com/solutions/virtual-queuing

Achinga, C. (2023, July 28). *Amazon RDS vs DynamoDB: 12 Differences You Should Know*. Retrieved from Cloud Academy: https://cloudacademy.com/blog/amazon-rds-vs-dynamodb-12-differences/

Altexsoft. (2023, Nov 30). *Functional and Nonfunctional Requirements: Specification and Types*. Retrieved from Altexsoft: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/

Amazon Web Services. (2023). *Allowing unauthenticated guest access to your application using Amazon Cognito*. Retrieved from docs.aws.amazon.com: https://docs.aws.amazon.com/location/latest/developerguide/authenticating-using-cognito.html

Amazon Web Services. (2023). *Amazon DynamoDB features*. Retrieved from aws.amazon.com: https://aws.amazon.com/dynamodb/features/

Amazon Web Services. (2023). *AWS Cloud Development Kit features*. Retrieved from Amazon Web Services: https://aws.amazon.com/cdk/features/

Amazon Web Services. (2023). *AWS Cloud Products*. Retrieved from Amazon Web Services: https://aws.amazon.com/products/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re%3AInvent=*all&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all

Amazon Web Services. (2023). *AWS Lambda*. Retrieved from aws.amazon.com: https://aws.amazon.com/lambda/

Amazon Web Services. (2023). *What is a RESTful API?* Retrieved from aws.amazon.com: https://aws.amazon.com/what-is/restful-api/

Amazon Web Services. (2024, April 19). *boto3*. Retrieved from PyPI: https://pypi.org/project/boto3/#description

Amazon Web Services. (n.d.). *Amazon API Gateway pricing*. Retrieved from Amazon Web Services: https://aws.amazon.com/api-gateway/pricing/

Amazon Web Services. (n.d.). *Amazon DynamoDB features*. Retrieved from Amazon Web Services: https://aws.amazon.com/dynamodb/features/

Amazon Web Services. (n.d.). *AWS WAF features*. Retrieved from Amazon Web Services: https://aws.amazon.com/waf/features/

Amazon Web Services. (n.d.). *Using Global Secondary Indexes in DynamoDB*. Retrieved from Amazon Web Services: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html

Amazon Web Services. (n.d.). *What Is a Graph Database?* Retrieved from Amazon Web Services: https://aws.amazon.com/nosql/graph/

Amazon Web Services. (n.d.). *What's the Difference Between Block, Object, and File Storage?* Retrieved from Amazon Web Services: https://aws.amazon.com/compare/the-difference-between-block-file-object-storage/

Android. (2024, April 18). *Meet Android Studio*. Retrieved from Android Developers: https://developer.android.com/studio/intro

Android. (n.d.). *Guide to app architecture*. Retrieved from Android Developers: https://developer.android.com/topic/architecture

Atlassian. (2023). *Jira | Issue & Project Tracking Software*. Retrieved from Atlassian: https://www.atlassian.com/software/jira?tab=tab-track

Atlassian. (2023). *What is Agile?* Retrieved from Atlassian: https://www.atlassian.com/agile

Atlassian. (2023). *What is version control?* Retrieved from Atlassian: https://www.atlassian.com/git/tutorials/what-is-version-control

Attractions.io. (2021, January). *Virtual queuing - the future of theme park experiences?* Retrieved from Attractions.io: https://attractions.io/learn/virtual-queuing-the-future-of-theme-park-experiences

Basak, D., & Evarts, T. (2022, Dec 7). *Displaying position in queue for chat customers*. Retrieved from AWS Blog: https://aws.amazon.com/blogs/contact-center/displaying-position-in-queue-for-chat-customers/

Cronkleton, E., & Bubnis, D. (2019, March 14). *What Is the Average Walking Speed of an Adult?* Retrieved from Healthline: https://www.healthline.com/health/exercise-fitness/average-walking-speed#average-speed-by-age

DevMountain. (n.d.). *What Languages Are iOS Apps Written In?* Retrieved from DevMountain: https://devmountain.com/blog/what-languages-are-ios-apps-written-in/

Disney World. (n.d.). *Virtual Queues*. Retrieved from DisneyWorld.co.uk: https://www.disneyworld.co.uk/guest-services/virtual-queue/

Errigal Software Solutions. (2023). *Tailored Network Monitoring Solutions*. Retrieved from Errigal.com: https://errigal.com/solutions/

Feerasta, A. (2019, May 23). *The canonical spec for ulid* . Retrieved from GitHub: https://github.com/ulid/spec

Felix Richter. (2023, August 8). *Amazon Maintains Lead in the Cloud Market* . Retrieved from Statista: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

GitLab. (n.d.). *What is CI/CD?* Retrieved from About Gitlab: https://about.gitlab.com/topics/ci-cd/

Hawker, A. (2020, September 15). *ulid*. Retrieved from PyPI: https://pypi.org/project/ulid-py/

heuer. (2024, February 8). *Segno - Python QR Code and Micro QR Code encoder¶*. Retrieved from Segno.ReadTheDocs: https://segno.readthedocs.io/en/latest/

Kotlin. (2023, October 27). *Get started with Kotlin*. Retrieved from KotlinLang.org: https://kotlinlang.org/docs/getting-started.html

Lardinois, F. (2019, May 7). *Kotlin is now Google's preferred language for Android app development*. Retrieved from TechCrunch: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/?guccounter=1

Ledbetter, J., Mohamed-Ameen, A., Oglesby, J. M., & Boyce, M. (2013). Your Wait Time From This Point Will Be . . . Practices for Designing Amusement Park Queues. *Ergonomics in Design The Quarterly of Human Factors Applications 21*, 22-28.

Marosfalvi, G. (2021, August 11). *The new way of storing data in Android — Jetpack DataStore*. Retrieved from Medium: https://medium.com/supercharges-mobile-product-guide/new-way-of-storing-data-in-android-jetpack-datastore-a1073d09393d

Patel, B. (2023, November 7). *5 Best Ways to Develop iOS Apps on Windows*. Retrieved from Space Technologies: https://www.spaceotechnologies.com/blog/develop-ios-apps-on-windows/

Pearce, P. L. (1989). Towards the better management of tourist queues. *Tourism Management Volume 10, Issue 4*, 279-284.

Python. (2024). *About Python*. Retrieved from Python.org: https://www.python.org/about/

Rehkopf, M. (2023). *User stories with examples and a template* . Retrieved from Atlassian: https://www.atlassian.com/agile/project-management/user-stories

Rehkopf, M. (2024). *What is a kanban board?* . Retrieved from Atlassian: https://www.atlassian.com/agile/kanban/boards

Richardson, L. (2020, August 28). *Understanding Jetpack Compose — part 1 of 2*. Retrieved from Medium: https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050

Schmidt, T. (2023, March 13). *Supported Languages at AWS Lambda*. Retrieved from AWSFundamentals: https://blog.awsfundamentals.com/supported-languages-at-aws-lambda#heading-nodejs

SETU. (2023, September 1). Academic Regulations for Undergraduate and Taught Postgraduate Programmes 2023-24: SETU Waterford. Waterford, Ireland.

TestDriven.io. (n.d.). *What is Test-Driven Development?* Retrieved from testdriven.io: https://testdriven.io/test-driven-development/

12. Appendices

Q-Up App demonstration:
https://www.youtube.com/watch?v=Q6tVN1AfIZ8

Q-Up App GitHub repository:
https://github.com/DylanFennelly/Q-Up