EXPLORING THE USE OF MOBILE SENSORS

Project report

Abstract

Researching and implementing modern mobile sensors to create an app which not only makes a task simpler, but also enhances the user experience by providing an additional layer of entertainment or novelty to make the task more interesting.

Alfie Strickland
N0733719

Table of Contents

Terminology & abbreviations

- Park(s): theme park(s)
- Coaster(s): roller coaster(s)
- Credits / points: the total number of roller coasters someone has been on
- NoSQL: not only SQL (structured query language)
- UX: user experience
- UI: user interface
- SDK: software development kit
- QR: quick response
- NFC: near field communication

1.INTRODUCTION

In a survey of 124 roller coaster enthusiasts, 98 people (79%) reported that they track the number of roller coasters they have been on (appendix 2). Naturally, this has caused a demand for automated services to make the task of tracking the total ridden rides and listing them quick and easy, this can be verified by another question in the survey where, out of those 26 that said they didn't track their coasters, 19 answered that an app to simplify the process could convince them to start doing it. These numbers combined with a passion for roller coasters and the theme park industry inspired an idea to create a new and improved service to make recording coaster credits easier for everyone. Using personal knowledge and research into the industry, available technology and most efficient methods of data capture, a new service will be designed, implemented and tested throughout this report, eventually creating a service to hopefully rival the leading services currently being used.

1.1 Plan

Research into the available sensors will be used to gain insight as to what additional features could be implemented in the app. The results would then be used to ask the intended audience of the app which sensors they would want to see, and which ones they would be most comfortable with, resulting in a clear idea of which sensors should be used and what they should be used for.

Design will take place once research has mostly, if not completely concluded. This will enable the chosen sensor functionality to be designed along with the interface of the app. The app should be designed to be simplistic as possible as its main goal is to be accessible and easy to use as possible.

Implementation will take the design plans and use code to make them into a real app. During this stage, any results from the previous stages will be followed as closely as possible in order to make sure the app stays on track and does not get carried away in implementing new and possibly pointless features for novelties sake.

2.CONTEXT

2.1 Background

There is a community of people known as "enthusiasts" who track the number of rollercoasters they have been on, referred to as "credits" or "points". Each rollercoaster is considered to be a credit and the total number of credits they have is used to either keep a personal record, or to boast and compare among friends and others online. The most committed enthusiasts are willing to travel the globe in order to visit the latest rides so that they can collect new credits to add to their total. There are many different ways to record credits and it is up to each individual to choose which method is best for them. Methods range from a handmade pen & paper style list and tally, to electronic tally /

counting apps, to purpose-built applications and websites which can be used to compare credits with strangers around the world.

2.2 Existing solutions

2.2.1 Pen & Paper

Possibly the cheapest / quickest method to keep a record. Users can keep a tally / list of their total using pen & paper. The method or format would differ from user to user but there is no need for any technology which may be easiest for some users and therefore the preferred method. The drawback to this method is that due to the lack of technology there is no standardisation of input meaning that if not properly maintained the records could become difficult to follow and confusing. Another drawback is that over time, If the user meets the storage limit (ran out of space on paper / pages in book) it may not be possible to extend that storage space, without manually copying all the records by hand. This drawback can be overcome via technology as more storage can be allocated with relative ease and data can easily be copied over to other devices etc.

2.2.2 Self-created Spreadsheet

The digitalized version of pen & paper (see Figure 1), this method has similar drawbacks. As there is no standard way of doing it the format / method would be down to the user which will require some planning and forward thinking. Perhaps the most common method would be the use of spreadsheet software such as Microsoft excel or Google Sheets.

#	Date Ridden	Coaster	Park	Material
1	10/1/1988	Scooby Doo	Carowinds	Wood
2	1992	Gold Rusher	Carowinds	Steel
3	?	Unknown	NC State Fair	Steel
4	1994	Space Mountain	Magic Kingdom	Steel
5	1994	Big Thunder Mountain Railway	Magic Kingdom	Steel
6	1994	Hurler	Paramount's Carowinds	Wood
7	1994	Thunder Road (1)	Paramount's Carowinds	Wood
8	1994	Thunder Road (2)	Paramount's Carowinds	Wood
9	1995	Carolina Cyclone	Paramount's Carowinds	Steel

Figure 1: Example of a user created spreadsheet (Hawkins, n.d.)

With the right technical knowledge, this method can be very efficient using formulas to calculate total numbers, and validation to preserve the format of each record.

The main benefit of this method over pen & paper is that it overcomes the storage issues. This is achieved as the file formats used by spreadsheet software are very small in size relative to the storage capabilities of modern computers and even mobile devices. Data can also be copied very easily onto other devices which is good for accessibility, files could also be stored in the cloud which allows the user to edit the records from multiple devices without having to worry about keeping the file up to date over all their devices.

2.2.3 Website

Using an online service is appealing to users who do not have the ability or time to create and maintain their own spreadsheet. It allows them to set up an account and let the website to take care of the tedious parts for them. However, web-based services such as coastercount.com do have their drawbacks. For example, as the service is based online, it requires an internet connection, this means that some users may not be able to access it while they are out and visiting theme parks. This would then require them to remember the rollercoasters they have been on so that they can update their profile when they get home or have internet access which could be off putting for some people as they want to update their profiles on the go, as soon as they are off a ride.

Online services have their advantages, though. They can improve user experience by allowing features such as a friend list so that users can view their friends profile, and possibly search for other

people in their area who share the same interests to make new friends. This can attract users towards a service and away from others, which is crucial in building a core user base. The issue with allowing people to have public accounts to show off their score is that, inevitably, some people will try to cheat to artificially inflate their score. This is made easy as there is no validation built into the site, instead admins are responsible for manually reviewing accounts which appear to have unreasonably high / rapidly increasing scores. This would require a lot of time and maintenance to keep track of accounts if there are hundreds registered.

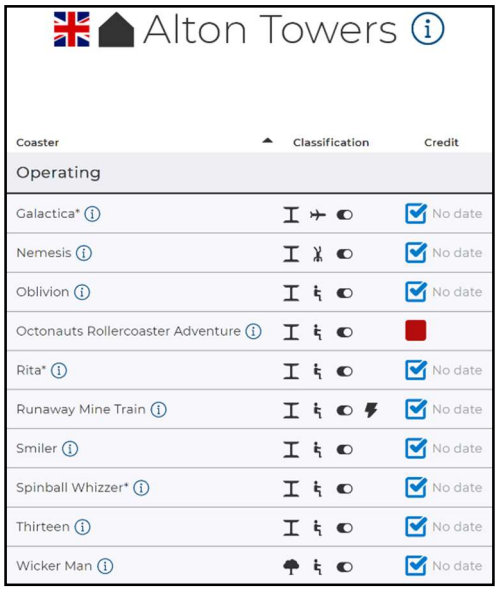


Figure 2: Method of adding credits (Thumann, n.d.)

The method of adding a coaster to your account is long and tedious, it requires searching through continents; then countries; then theme parks; and then you can tick off different coasters (see figure 2).

For each theme park, each ride is listed with some info and a checkbox used to add it to your account. Clicking the info button allows the user to view detailed information about each ride using the Rollercoaster Database, a comprehensive online resource that holds details on every roller-coaster in the world.

Figure 3: Adding coaster to an account (Thumann, n.d.)

Once the user has ticked off a coaster, a menu appears (Figure 3) where they can choose the date they visited it. This is a good feature as it allows users to keep track of when they first rode a coaster. However, it is a long process and users may not always remember the date if they are adding a coaster they visited a long time ago.

Figure 4: Leader boards (Thumann, n.d.)

Another UX Feature of the site is the leader boards (Figure 4). This displays the highest scoring accounts either globally or among friends. There are options to filter down the leader boards into country / sex / age etc. however they are locked behind a paywall.

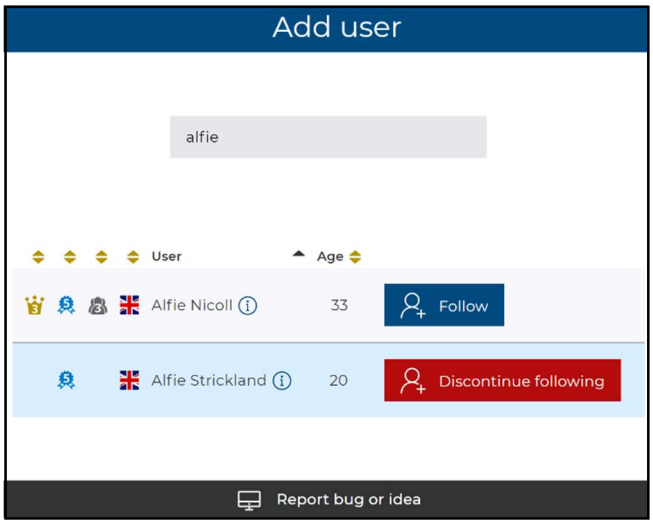


Figure 5: Searching for friends (Thumann, n.d.)

Coaster-count.com also offers a friends system (Figure 5), users can add other accounts to their friends list which enables them to view their accounts and compare them via the leader boards. This is a good feature to enable a competitive aspect to the service, however there should be an option to opt out of being discoverable to others if a user wishes to not have people looking at their accounts.

2.2.4 Mobile App

A mobile app has the same benefits of a website as its online so the user does not have to worry about storage etc, however the additional advantages are that it is easier to use on a mobile meaning it is more likely to be used on the go, while the user is at a theme park. This allows them to update their records as soon as they come off a ride, meaning there is no need to remember or make note of the rides until they are home / have access to a computer.

Mobile apps also have a similar downfall to websites as they usually require an internet connection to have full usability of the service. For example, an app could use a local storage system meaning that the user account is updated without internet connection, but they cannot see their friends accounts or vice versa until they have an internet connection.

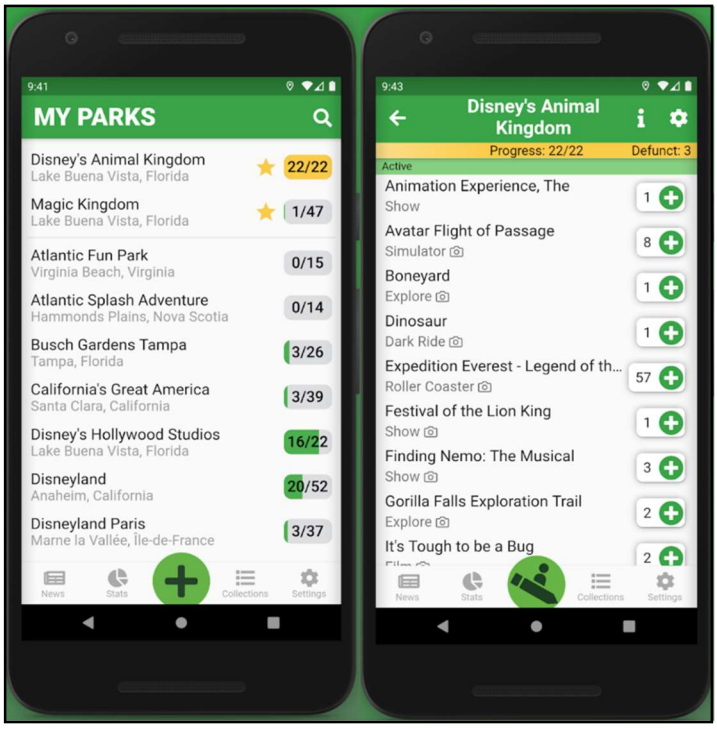


Figure 6: Theme parks and rides per park (thePARKSMAN, 2018)

One app, Log Ride, has similar functionality to the coaster count website. It allows users to browse through different theme parks until they find the one they want to add, and from there all rides at that park are displayed and the user can add the ride and also change how many times they have been on it (see Figure 6).

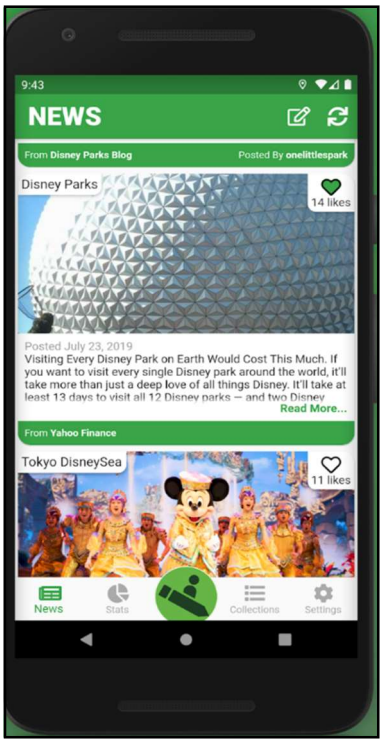


Figure 7: News page (thePARKSMAN, 2018)

The app has other UX features such as a news page (see Figure 7) which displays related news such as details for new rides or updates for different theme parks across the globe. This is a good feature as it gives another reason for enthusiasts to use the app regularly if they are not visiting parks and updating their accounts, meaning that the app will still receive traffic during the off-season where theme parks are closed for the winter.

The news articles that appear on this page are user generated. Users can submit titles, brief descriptions and links to external news sources. This means that users are more likely to create these news items as they do not have to write the whole story themselves, only link to stories they've seen online.

Another feature is the Stats page (Figure 8), where it shows a summary of the users visited attractions and parks. In the attractions section, the number of active / defunct attractions is shown as a ratio in the form of a bar which changes accordingly. Top attractions are also shown, this lists the top 5 attractions visited in order of the number of times they have been visited. Finally, the number of different types of ride the user has visited is displayed at the bottom and can be ordered by checks, ride type, or experiences.

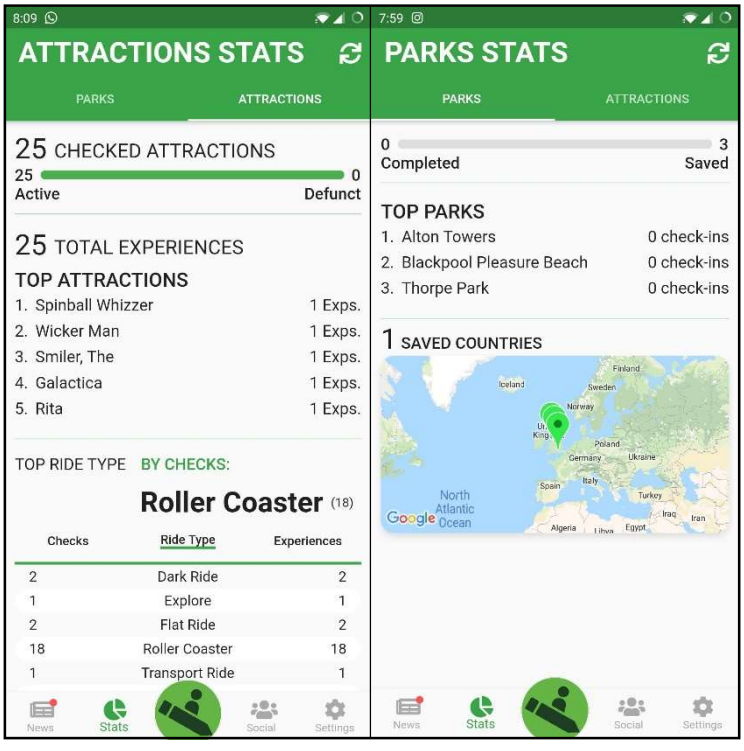


Figure 8: Stats page (thePARKSMAN, 2018)

The parks page shows the number of theme-parks the user has completed (all attractions visited) and the total number of parks they have visited. The page also shows the users top parks, where the users parks with the highest number of visited attractions are listed. The most interactive part of the page is at the bottom where the number of countries the user has visited is displayed along with a map that pinpoints the theme-parks the user has visited, the map can be opened which allows the user to scan through the different locations.

2.2.5 Conclusion

The big issue with existing solutions is that they lack validation (see table 1). It is possible, and easy, to lie about places a user has visited. This is a problem for online services such as apps and websites especially if they include features such as leader-boards and other methods of comparing scores among different account as it is difficult to identify which accounts are legitimate and which are not. The only verification found in the above solutions is the manual account checking from coaster-count.com where site admins check accounts that seem untrustworthy and terminate accounts which turn out to be lying.

Methods to counteract this issue would be to implement a system of adding coasters to an account which requires the use to physically be at the location of the ride to enable the ability to add that coaster. Mobile sensors can be used to achieve this. For example, GPS can be used to verify the user's location so that a check can be ran to ensure that a user Is at a similar GPS co-ordinate to the ride they are trying to add. However, there are a few issues with this too as it is possible to use GPS spoofing apps on android devices meaning the user can pretend to be in different locations at will. As the user only needs to be near the ride to add it to their account, they do not necessarily have to ride the rollercoaster to add it to their account, but instead add it from just standing within a few metres of it.

Another method would be the use of a physical medium, such as a QR code or NFC tag placed at a ride exit. Once scanned the app recognises the ride that the code / tag is associated with and adds that ride to their account. This has the benefit over the GPS method as it is placed at the ride exit it would require the user to go on the ride in the first place.

Key	
	- Poor
	- Okay
	- Good

Table 1: comparison between the different method's characteristics

	Pen + Paper	*Spreadsheet*	*Website*	*App*
Ease of use	Easy to start, user can use preferred format.	Requires some technical knowledge, easy once set up.	Takes some getting used to, main flow could be quicker.	Very easy to add credits once theme park is found.
Portability	Becomes increasingly difficult over time.	Can be accessed via different platforms, including mobile.	Requires internet connection, possibly of poor GUI for mobile devices.	May require internet connection, optimized for mobile devices.
Storage / archiving	Becomes increasingly difficult overtime, more maintenance required as list expands. Will take time and effort to copy data by hand.	Requires small amount of computer storage, easily expandable. Easy to archive with simple technical skills.	Taken care of by website, unable to obtain personal copy of data.	Taken care of by app, unable to obtain personal copy of data.

UX features	No other features provided other than a list / tally.	User can implement own features such as graphs etc with the right skills.	Leader boards of other user accounts, friend's system, some features behind paywall.	Analysis of user stats, related news articles, map of visited parks, and more currently in development,
Validation	None.	None.	None. However, accounts can be monitored by admins and removed if fount to be fraudulent.	None.

2.3 Development options

2.3.1 Front end

Android SDK by Google allows android apps to be written in many languages such as Java, C++ and Kotlin. This is good as it will potentially avoid the need for learning a new programming language which can be time consuming and frustrating. The android SDK provides a tool where elements such as buttons, image placeholders, etc. can quickly be dragged and dropped into a preview of the app (figure 9). This means that the main layout of applications can be developed quickly and easily, allowing more time to be spent on developing more intricate parts of the app, however extra time is spent on styling the elements to make them look more modern for a better looking overall app. The main benefit of using stock android SDK is that most other tools (such as a backend) are developed for it over other SDKs as it is the basic SDK for android development.

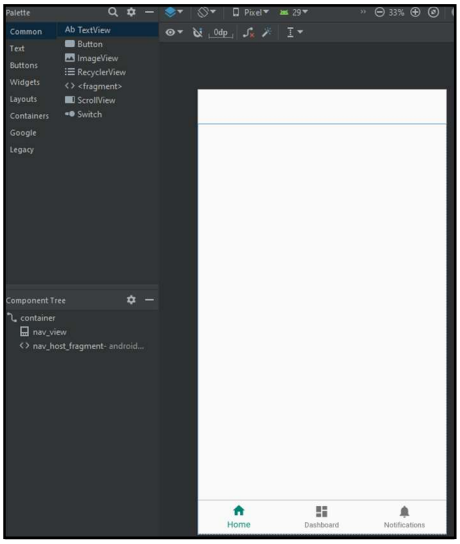


Figure 9: Preview of app layout, and droppable components (top left)

Flutter UI is an open source SDK also by Google which boasts its speedy development time and flexibility. features such as hot reload benefit development as it allows changes to be made and tested quickly without having to recompile the app each time. Flutter also provides a more modern looking UI over the basic android SDK, this would allow for the creation of better looking apps, in a shorter time, which will be an advantage as it would help to captivate an audience for the app which will be necessary to draw them away from existing services. Flutter also has a range of online videos demonstrating features and explaining their use, made by the developers themselves so they are reliable and very useful when getting to learn the SDK, this will be helpful during development as it can provide quick solutions to any troubles encountered.

Table 2: Comparison between SDKs

	Android SDK	*Flutter SDK*
Speed of development	Quick to get basic page design due to drag + drop page elements	Hot reload feature helps to quickly experiment, build UIs and fix bugs
Development tools	Android virtual device emulator to test app without the need for a physical device	Hot reload to apply changed to app without having to wait to recompile / build code
Documentation	Good documentation, plus video tutorials from a range of people	Good documentation, plus official video guides from the flutter developers

2.3.2 Backend

This project would require some form of database in order to store user accounts and the data each one holds.

There are many different options to choose from when selecting a backend. Parse is an open source platform owned by Facebook used to create servers to host databases. Parse servers are meant to be mounted on an express app, a web framework for Node.js. parse allows user management which allows each user to create their own account which has its own data stored in a database also hosted by parse. This would be ideal for the intended solution as it will require each user to have private access to their own account and the data on that account needs to be hosted online in order to implement some of the desired functionality.

Another option would be to use Firebase, a backend service provided by google which has mostly the same features as parse. It allows for easy implementation of authentication from different accounts such as Google, Facebook, Twitter etc through its Firebase authorization service. This means that new users do not have to create an account when using the app, they can simply sign in using their already existing accounts from other sites. Using Firestore, a NoSQL database service provided by Firebase; these user accounts can store their own data meaning that users individual data can be stored independently from each other, preventing users from seeing other people's data. Firebase also has its own set of plugins designed to work with flutter, allowing for quick and easy implementation of different services with a wealth of documentation to help.

2.3.3 UX Design Guidelines

The app will follow Googles material IO design guidelines. This is due to the guidelines focusing on making the apps feel familiar to the user and making the app as accessible as possible. Material IO achieves this by using theming rules in order to create familiar but unique designs for app components such as buttons. Accessibility is addressed in Material IO via the use of hierarchies, the general rule for hierarchies is that the more important an action / component is, the closer it should be to the top / bottom of the page; and items of similar importance should be placed side by side. Accessibility also involves the colour of items, and more importantly the contrast between different colours. Contrast between foreground/ background elements is important as it allows users to see

things easier such as text and icons, meaning that users can follow the flow of the app easy as it is clear where buttons and text are. Another reason that contrast is important is that it helps the visually impaired for example people that suffer from colour blindness would find it difficult or even impossible to discern between some colours with a low contrast ratio. One method to combat this would be to implement different settings to toggle the colour scheme of the app, although it would be much easier to stick to a high contrast theme from the start. Material IO makes choosing a colour scheme easy by offering a colour palette tool where a primary colour can be picked from a colour chart and other secondary etc. colours with good compatibility will be listed, this can then be coded into the app at an early stage using the primary / secondary colour variables which can be used throughout development to quickly build a consistent UI.

3.NEW IDEAS

3.1 High level description

The new method of adding rollercoasters to the users account will involve some form of readable data at the location of the rollercoaster, this could be a QR code or NFC tag for example. Whilst on the rollercoaster users will have the option to read their heartrate from their own smartwatch, this data will then be taken and stored alongside the record of their visit to the rollercoaster (This feature has to be optional as not all users will own a smartwatch). Once rides have been added to the users account, they can view them on their own user profile, users can also view other people's profiles using a search function. A friends' system will also be used to make it easier for people to view specific profiles regularly without having to search for them each time. Leader boards will be implemented in order to show the top users in different categories such as global, country & friends.

3.2 Aims & Objectives

This project aims to use the different sensors that modern mobile phones provide to improve upon the implementation of existing services.

The aim will be achieved by investigating into the different possible methods / sensors and weighing up the benefits and drawbacks of each. The most suitable method will be implemented into an android app, the app will also aim to match the level of extra features that the existing methods provide in order to prove that it is a viable service.

The project will also include research into which methods the potential users are most comfortable / familiar with, this will have influence on the direction of the project as the aim is to not only find the most reliable method, but the most suitable, which involves the opinion of users.

3.3 Development Methodology

This project will follow the waterfall methodology, where requirements will be gathered at the beginning of the project through methods such as analysis of existing solutions, and creating surveys for the intended user base to complete for analysis of what they really want from the app. The results from this research can then be interpreted into lists of requirements. Once requirements are determined, the design phase will begin. This is where interfaces mock-ups will be created in order to satisfy the listed requirements, and APIs will be decided in order to meet functional requirements. Once there is a complete design plan for the app, the implementation phase will involve turning the design mock-ups into code. This will feature small functional tests along the way to ensure parts of the app work individually and that the app still runs without crashing to help development move along before getting to the end of the development cycle and realising there are multiple errors which will be difficult and time consuming to debug. The verification stage will involve performing larger tests to ensure features work together as a whole rather than individual components. This is

to ensure that the app will function wholly as intended and is not just a few standalone features together in an app. This stage is to also check against the requirements in order to judge how the app meets users' needs and if implementation followed what it was set out to achieve from the beginning. User testing will also take place in order to get real and unbiased feedback from potential users. This feedback will be used to identify areas for improvement and judge how well the design of interfaces considered for the end user. The maintenance phase will take place, time permitting, in order to iron out any bugs that the user found whilst using the app. In a real world project this phase would be ongoing for a longer period of time depending on the scale of the app, maintenance may not just be performed on the app but the backend as well, to enable the database to cater for the capacity of users.

3.4 Test plans

Testing of the application will be required during and after development. Tests during develop will be smaller functional tests to ensure that the newly implemented code works and does not crash the entire, or parts of, the app. This enables requirements to be ticked off and therefore allow other parts of the app that rely on the completed part to begin development. Testing after the apps development is complete will consist of user tests and more vigorous edge case testing. This will allow judgement to be made on the quality of the implementation of the app and also help to identify any harder to find bugs which are caused via user misuse or long term use. The user tests will also provide opportunity to gain feedback on the usability and design of the app, which can give information about what needs to be considered in future work, or where the app has fallen short of what it had set out to achieve.

3.5 Research

In order to prioritise features and find out what potential users really want in an app like this, a survey was created. This allowed participants to choose which features they would like to see; and at some points give their own ideas and opinions as to what would make a good app. The survey was closed at 124 responses. To make sure the participants of the questionnaire were potential users of the app, the survey was posted online to the r/rollercoasters subreddit, a forum for rollercoaster enthusiasts to share news; and other theme park-based media.

To aid the decision between the use of QR codes or NFC tags, a question for each was used to gain an understanding of how familiar the participants were with each technology.

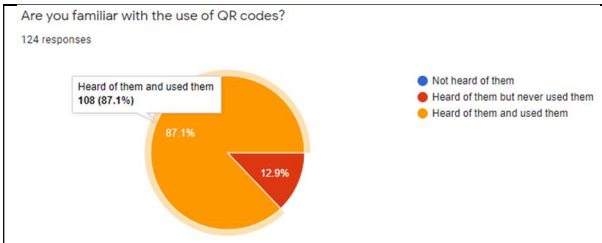
<table>
<tr><td> 

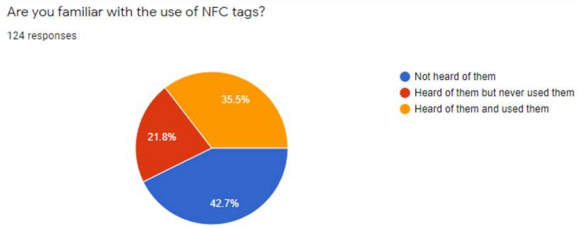
Figure 10: Survey question on QR codes</td><td>The responses to the familiarity of QR codes show that most people (108/87%) had heard of them and used them in the past, whilst the remaining participants said they had heard of them but never used them. No participants said that they have not heard of QR codes.</td></tr>
<tr><td> 

Figure 11: Survey question on NFC tags</td><td>The familiarity of NFC tags questions had a much more split response. 35.5% said that they had heard of them and used them, 21.8% said they have heard of them but not used them, and 42.7% said they have never heard of them.</td></tr>
<tr><td colspan="2">Judging between the two questions, it is clear the participants are more familiar with the concept and use of QR codes. This will influence the design of the system as it should use the method that users are most comfortable with to help encourage use of the app and improve the users experience.</td></tr>
</table>

The question regarding categorisation was designed to help the designing of the backend database as well as the front end, as the database will need to store the theme park each rollercoaster belongs to in order to implement categorisation.

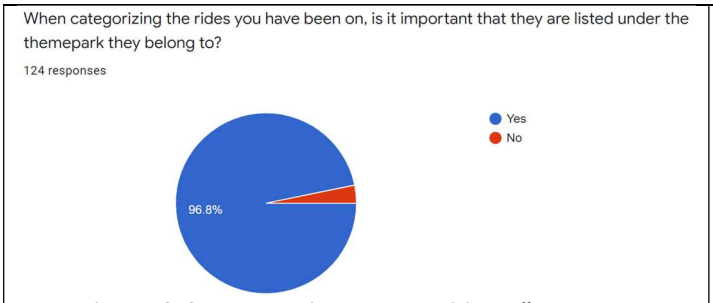
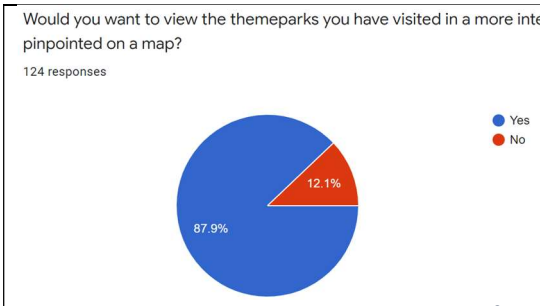
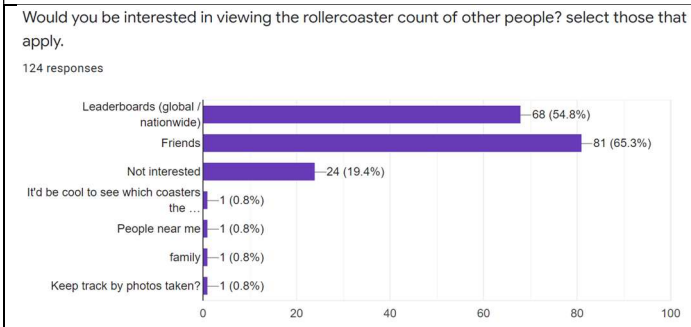
<table>
<tr><td> 

Figure 12: Survey question on categorising rollercoasters</td><td>96.8% of participants answered that they would want their rollercoaster credits to be categorised by theme park. This is a clear indication that the rollercoasters must be sorted into theme parks.</td></tr>
<tr><td colspan="2">This question shows that categorisation is very important to users. This means that a method of grouping rollercoasters by the theme park needs to be designed and implemented into the front and backend of the system.</td></tr>
</table>

The final questions were designed to help identify and prioritise any extra features that would make the app a more appealing service to its target audience.

 *Figure 13: Survey question on map feature*	The majority (87.9%) of participants say that they would want a feature that shows their visited theme parks on a map. This means that the map feature should be implemented, however it is not crucial to the functionality of the app so it could be put on hold until the other more vital features are complete.
 *Figure 14: Survey question on viewing other users' stats*	For this question, 65.3% of participants say they would like to be able to view their friends' stats, and 54.8% said they would be interested in viewing stats of strangers in their country / worldwide. Only 19.4% said they were not interested in seeing others' stats meaning that this should also be implemented, but perhaps focusing on the friends feature first before the global + nationwide features.

Additional comments were left on the original post of the survey which provided extra ideas as to what features could also be implemented in order to create a more user-friendly app. One user commented that one feature preventing them from investing in existing apps is the lack of an import / export feature where the users list of rollercoasters could be exported into a spreadsheet. This could possibly be popular among other potential users; however, it is not a crucial feature the applications objectives.

Other users have also commented that the reason they dislike other services is that they have too much unnecessary functionality, which makes it harder to complete the main flow of the service. This will impact the requirements collection as non-necessary functionality will be pushed aside in order to allow more time to develop and test the core features.

3.6 User stories

User stories are short statements to describe tasks a user seeks to accomplish when using the software. This helps to gain a solid list of the goals the users want to complete and therefore allows requirements to be based on these goals and planning user interfaces and high-level functionality.

1. User wants to add a roller coaster they have just been on to their list of ridden rides.
2. User wants to view theme parks they have visited.
3. User wants to view roller coasters they have ridden.
4. User wants to view the roller coasters they have ridden in a specific park.
5. User wants to see the total number of roller coasters they have ridden.
6. User wants to see the total number of parks they have been to.
7. User wants to access their account from different device.

3.7 Requirements

3.7.1 Functional:

The functional requirements are prioritised using the MoSCoW rule. The requirements under “Must” are those which the system needs in order to succeed. The requirements that fall under “Should” will be implemented if possible, but the project’s success does not rely on them. The requirements under “Could” will be implemented if there are no time constraints, and no other more important requirement is reliant on them. The “Won’t have” requirements are those that provide little to no value to the system and will be the first to get dropped if necessary.

The system must:

1. Take an input of rollercoaster name and the theme park it is based in (using QR / NFC); and add it to the users account.
2. Display the list of users visited rollercoasters and theme parks.
3. Allow users to create their own account.
4. Keep track of total number of users visited rollercoasters / theme parks.
5. Read recent heartrate data from a smartwatch and store it alongside each visited rollercoaster.

The system should:

6. Display leader boards of user profiles who have the highest number of rollercoasters on their profile, can be filtered by country/ friends etc.

The system could:

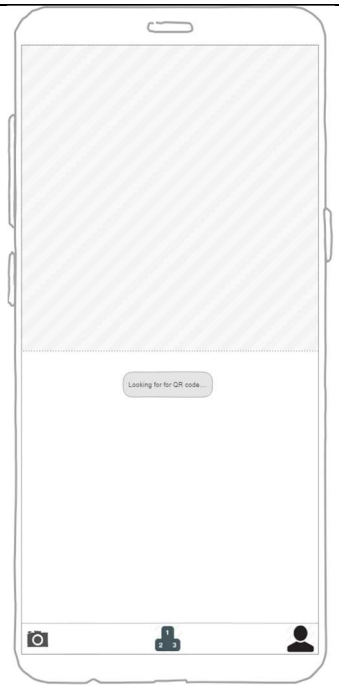
7. Include a friend’s system where users can search for other users and request to add them to their friends list.
8. Display the location of each theme park using google maps.

3.7.2 Non- functional:

- Users accounts can be accessed via different devices.
- Each user account stores the theme parks and optional heart rate data for each coaster they have visited.
- Once QR code is scanned, the user’s profile will be updated as soon as internet connection is available.
- Categorize users visited rides by the theme park they are based in.
- User data loads quickly and is readily available without having to re-load it multiple times.

3.8 Initial designs

3.8.1 Mock-ups

<table>
<tr>
<td> 
</td>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Log in page, the first page users will see. Page consists of only a logo and a sign in button, on first sign in, a user account will be created to meet requirement 3, and user story 7.</td>
<td>Profile page, on this page the user can view their own stats (requirement 2+4, user story 5+6); and browse through the theme parks they have visited (user story 2).</td>
<td>Profile page, when a user selects a park they have been to, the list of rollercoasters they have been on will appear. Alongside some additional data such as the recorded heart rate (user story 3).</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
<td> 
</td>
</tr>
</table>

Leader boards page, this page will display the users with the highest coaster count in the selected area (requirement 6). The spinner in the top right allows the user to choose from global / country / friends.	QR scan page, this page will allow the user to scan QR codes at attractions to then add that specific ride to their account in order to achieve requirement 1 and user story 1.	Add friends page, this page features a search bar where users can search for their friends' profile. Once the friend has been found they can add that person to their friends list, and they will then appear on the leader boards page under the friend selection this page was designed in order to satisfy requirement 7.

3.8.2 Comments

The aim of the mock-ups was to create a design which resembled other popular apps, for example the profile page took inspiration from the Instagram profile page where the username, followers and number of posts are displayed in one section at the top of the page, and the rest of the profile content is displayed beneath. The reason for this is to allow users to feel familiar with the app and decrease the amount of time it takes to learn where things are. This also helps the user to avoid feeling frustrated when they cannot find something and therefore be more likely to continue using the app. The Navigation bar is also used for this purpose as many popular apps use it like Instagram, twitter, and snapchat. These apps also use icons to differentiate between the items on the bar instead of text, this could increase usability as it would be functional across different languages, although it can be difficult to find an icon which adequately conveys the message required without confusing the user. This is avoided when using text as more accurate words can be used rather than leaving it up to user interpretation.

4.IMPLEMENTATION/INVESTIGATION

Development started with the investigation into which SDK to use. Although Flutter looked very promising and offered different advantages including faster development and easy modern UI, there was some issues with versions of different APIs such as firebase. For this reason, flutter was substituted for the native android SDK.

Firebase was picked to be used as the backend due to it being a Google product. This would mean that it should be easier to combine with other Google products such as android. Firebase also has very clear and detailed documentation, allowing for easy planning of how the databases and user accounts would work.

4.1 Firebase

To implement Firebase functionality into the app, the app must first be registered to a firebase project. This is quick and easy to achieve by following the firebase setup guide, and adding some dependencies into the apps "build.gradle" files. Once implementation is complete, it can be tested by running the app and firebase will tell you if it has received communication from the app.
The firebase console is where the different firebase functions can be created and edited, for example the to enable Google sign in, it needs to be activated under the authentication tab. Different sign in methods could be applied such as Facebook or Twitter, however only google was applied as it's an android app, most if not all users should have a Google account whereas not all may have Facebook or Twitter.

4.1.1 Firestore database

4.1.1.1 Design

Designing the Firestore database is a crucial step in meeting the requirements of the system as the correct data needs to be stored, and it needs to be stored in the right place. However, designing for a NOSQL database is more flexible than designing for an SQL database as not all documents in a collection have to contain the same data. This means design changes are easy to implement down the line without having to restructure the whole database like in SQL.

The initial design of the database (Figure 15) used a collection of users where each document in the collection would have a unique ID which would be generated from the Google authentication service. Each of these documents would have data fields, storing user data such as the number of roller coasters and theme parks the user has visited, each theme park would be stored as a map in this document and the map would have <string,int> pairs as its entries with the string being the roller coaster name and the int would be the users heartrate recorded for that ride.

```
Users{
        USER_ID{
                Total_coasters: int,
                Total_parks: int,
                Total_friends: int,
                Park_1: map<coastername,heartrate>
                Park_2: map<coastername,heartrate>
        }
}
```

Figure 15: Initial database design

4.1.1.2 Implementation

To implement the Firestore database, under the database tab in the firebase console, collections and documents can be manually created. This can be used to create test data to check functionality of the app and can also be used to view updates made in-app happen in an easy to understand format. The implementation of the original database design can be seen in Figure 16.

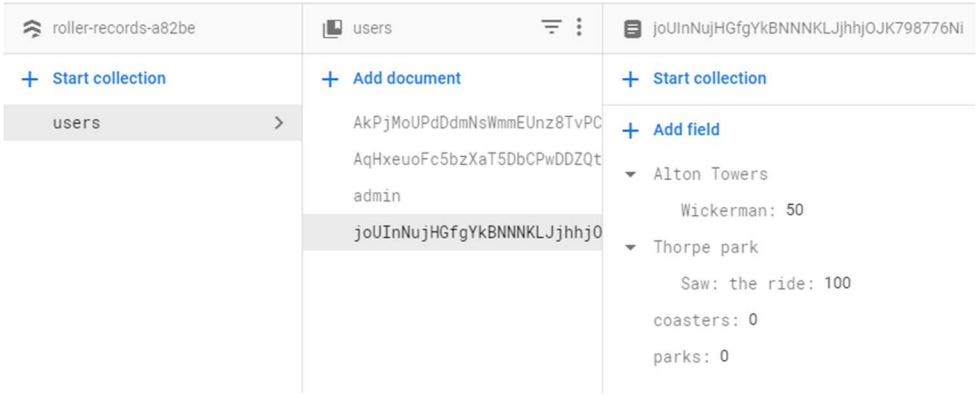


Figure 16: Initial implementation of database

4.2 Creating user accounts

4.2.1 Implementation

The first stage of development focused on setting up the user accounts. This involved creating a sign in page which would be the first thing the user sees on first use of the app. This screen would allow users to sign in with a google account, by either choosing one that is already associated with the device or by adding a new one. Once the user has selected an account to use, they are signed into firebase using Google authentication and their account is given a document inside the Firestore database users collection using a uniquely generated user ID. This enables the app to store / get any user data in / from a unique document, separately from all other user accounts. The sign in activity consists of two functions; the onCreate function sets up the variables used for sign in and also performs a check to see if the user is already signed in, if the user is signed in, then the sign in page is dismissed and the app will redirect to the next part of the app. However if the user is not signed in

then a onClickListener is set on the sign in button to trigger the sign in process. The onActivityResult function handles the authentication and database account creation. Once the account is authenticated a check is performed to see if the user has an account document in the database (Figure 17), if they do already have a document, the app proceeds to the next stage with no further action. However, if the user does not already have a document, one is created, and the necessary data fields are inserted. After this is complete the app will continue to the next activity.

```
final DocumentReference documentReference = db.collection( collectionPath: "users").document(mAuth.getCurrentUser().getUid());
documentReference.get().addOnCompleteListener((task) → {
        //if user document already exists, open activity, else, create document then open activity
        if (task.isSuccessful()){
            DocumentSnapshot documentSnapshot = task.getResult();
            if(documentSnapshot.exists()){
                startActivity(new Intent(getApplicationContext(),ProfileActivity.class));
            }
            else{
                Map<String,Object> user = new HashMap<>();
                user.put("parks",0);
                user.put("coasters",0);
                documentReference.set(user);
                startActivity(new Intent(getApplicationContext(),ProfileActivity.class));
            }
        }
```

Figure 17: Code to handle creation of new account

4.2.2 Testing

To test that the sign in functionality works, the firebase console was used to verify that an account was added to the list of accounts in the authentication tab and that a document was created with an ID mathching that of the user accounts UID.

After testing both the user account and corresponding document were visible in both the firebase authentication tab (Figure 18) and database (Figure 19). The “admin” document was added to the collection in order to prevent the “users” collection from deleting when user accounts were deleted during testing as firebase automatically removes empty documents and collections.

Identifier	Providers	Created	Signed In	User UID ↑
alfiestrickland99@gmail.com	G	Apr 13, 2020	Apr 13, 2020	AkPjMoUPdDdmNsWmmEUnz8Tv...
alfie.sspams@gmail.com	G	Apr 1, 2020	Apr 11, 2020	AqHxeuoFc5bzXaT5DbCPwDDZQt...

Figure 18: New accounts listed in authentication tab

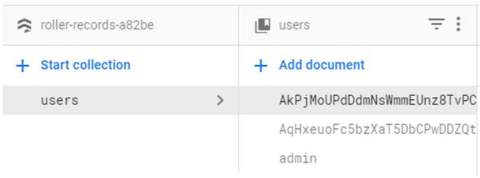


Figure 19: New accounts appearing in database

4.3 Application Navigation

The design for the rest of the app featured a navigation bar for the three elements of the app: the profile, QR scan, and the leader boards. To implement this, a tabLayout was created (Figure 20) to hold three selectable tabItems for each of the three elements, each of these tabs would correspond to a fragment so when a tab is selected its corresponding fragment would come into view. Each fragment would then hold the functions it needs and its own layout.

```
<com.google.android.material.tabs.TabLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    app:tabTextColor="@android:color/black"
    android:id="@+id/tabLayout"
    >

    <com.google.android.material.tabs.TabItem
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/QR"
    android:text="Add Coaster"
    />

    <com.google.android.material.tabs.TabItem
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/PR"
        android:text="Profile"
        />

    <com.google.android.material.tabs.TabItem
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/LB"
        android:text="Leaderboards"
        />
```

Figure 20: Layout file to implement tab layout

```
@Override
public Fragment getItem(int position) {
    switch(position) {
        case 0:
            return new fragqr();
        case 1:
            return new fragpf();
        case 2:
            return new fraglb();
        default:
            return null;
    }
}
```

Figure 21: Switch statement to map tabs to fragments

In order to map the tabItems to the fragments, a pageAdapter class was created where a switch statement maps the value of the tabItem (0/1/2) to the fragment classes (fragqr,fragpf,fraglb) (Figure 21).

Now that the navigation for the app was implemented, the main functionality of the app could be implemented into the newly created fragments. The designs for the profile fragment contains some basic user data; the user name, total number of coasters, and total number of parks. This was easily implemented as TextViews in the fragments layout file, with added code (Figure 22) to then fetch the users data from the databse / Google account, and use the .setText() method to change the text to the fetched data. This functionality was implemented in the onCreate method so that each time the fragment is created, the data is also updated, allowing for a more accurate reading of the userdata at a given time.

```
//setup ui using firebase data:
uName = view.findViewById(R.id.usrName);
uName.setText(firebaseAuth.getCurrentUser().getDisplayName());
final TextView score = view.findViewById(R.id.scoreNum);
final TextView parks = view.findViewById(R.id.parksNum);
uName.setText(firebaseAuth.getCurrentUser().getDisplayName());

documentReference.get().addOnCompleteListener((task) → {
        if(task.isSuccessful()){
            DocumentSnapshot documentSnapshot = task.getResult();
            score.setText(""+documentSnapshot.get("coasters"));
            parks.setText(""+documentSnapshot.get("parks"));
        }
});
```

Figure 22: Code to update profile with database values

4.4 QR Scanning

4.4.1 Scanning library options

Before implementing the section of the profile page which would display the users ridden coasters, there needed to be a method of adding that data to the account in the first place. This began the work on the QR scanning functionality. Investigation into the best method for scanning QR codes returned two different options, a library called “ZXing” and “mobile vision api”.

“ZXing is a barcode image processing library implemented in Java, with ports to other languages. It has support for 1D product, 1D industrial, and 2D barcodes.” (Google, n.d.). Google has used ZXing before in their own apps for use on android, however, it is an older library and could possibly have outdated documentation for more current android versions.

Mobile vision API is a more modern implementation with a wide range of tools which include face detection, barcode scanning and text recognition from live a live camera feed in real time or from static images saved on a device. The barcode API reads different barcode formats including 2d barcodes such as QR codes which is what the application was designed to scan, there are also parsing features built in which enables the QR codes to be parsed into formats such as URL’s, SMS,

phone numbers and more, meaning that in the future the app could be expanded to suit different requirements or implement new features.

The application will use the mobile vision API due to it being more modern than ZXing, although ZXing is more streamline and perhaps more suited for this applications purpose, the mobile vision API will be more designed towards modern android versions which is why google use it in their modern products like Google lens.

4.4.2 Design

The original implementation of the QR scanner had the camera preview in the fragment which would then allow users to scan codes without having to open a new activity. However, there was some issues with the scanner scanning constantly when a QR code is in sight, meaning that the code that ran whenever a QR code was found would run multiple times per second. To avoid this, the QR scanner was instead implemented into a new activity which could close on QR detection, preventing the scanner from being on and able to scan again. The QR fragment was still kept in the app but instead features instructions on how to use the scanner and a button to launch the scanner activity.

The QR scan activity has a simple layout which consists of a surfaceView to hold the devices camera preview, and a textView to display basic instruction on how to user the scanner. In order run specific code when the scanner detects a barcode, the receiveDetections function is used. In this function an array is created to hold the data, and then a check is used to see if there is any data in that array in order to proceed with the function. a vibrator is used to notify the user that the code has been scanned successfully, while the app deals with the data provided by the code. The format for the code should be “Themepark,Rollercoaster” (Figure 23), this is taken as a single string by the scanner and then separated into 2 strings, one for either side of the comma.

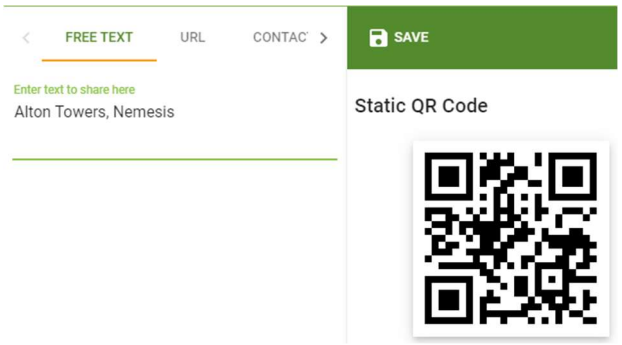


Figure 23: Example QR code with the intended format

The implementation of this function caused the discovery of an error in the layout of the firestore database, as there was trouble implementing a map with the field name of the scanned themepark, and the coaster as a key value. This is where the database changed into its current state of each user document containing a collection called “parks” which would hold a list of documents which are named as themeparks, and each document would contain fields acting as the name of the coaster and its value would be the heart rate data (Figure 24).

```
Users{
        USER_ID{
                Total_coasters: int,
                Total_parks: int,
                Total_friends: int,
                Parks{
                        Park1{
                                Coastername: Heartrate(int),
                                Coastername: Heartrate(int),
                        }
                        Park2{
                                Coastername: Heartrate(int)
                        }
                }

        }
}
```

Figure 24: Updated database design

4.4.3 Implementation

With the new database layout, the app gets a documentReference for the document name of the current scanned park, and creates a map consisting of a string and object, where the string is the coaster name and the object is the heart rate value (uses hard coded data at this stage). The documentReference uses the .get() method to retreive the data inside that document. If the

document exists, it means that the user has already visited that themepark before, so the newly created map is added to the document, and the users total number of coasters is incremented by 1. However, if the document does not exist, it means that the user has not been to that park before, so the else condition is triggered meaning that the same thing happens but the total number of theme parks is also incremented by 1. After this is complete the camera stops and the activity is closed, returning to the scan fragment (Figure 25).

```
Vibrator vibrator = (Vibrator)getApplicationContext().getSystemService(Context.VIBRATOR_SERVICE);
vibrator.vibrate( milliseconds: 1000);
String[] vals = (qrCode.valueAt( index: 0).displayValue).split( regex: ",");
//vals[0] == theme park || vals[1] == coaster

documentReference = db.collection( collectionPath: "users").document(firebaseAuth.getCurrentUser().getUid()).collection( collectionPath: "parks").document(vals[0]);
final Map<String, Object> map = new HashMap<~>();
map.put(vals[1],80);    //change 80 to heart rate
documentReference.get().addOnCompleteListener((task) -> {
        final DocumentSnapshot documentSnapshot = task.getResult();
        //if theme park already exists, add coaster to its document, else create document and add coaster
        if(documentSnapshot.exists()){
            documentReference.update(map);
            //increment coaster field
            documentReference = db.collection( collectionPath: "users").document(firebaseAuth.getCurrentUser().getUid());
            documentReference.update( field: "coasters",FieldValue.increment(1));
        }
        else{
            documentReference.set(map);
            //increment park + coaster field
            documentReference = db.collection( collectionPath: "users").document(firebaseAuth.getCurrentUser().getUid());
            documentReference.update( field: "coasters",FieldValue.increment(1));
            documentReference.update( field: "parks",FieldValue.increment(1));
        }
});
//stop camera & return to activity
cameraSource.stop();
finish();
```

Figure 25: Algorithm to handle the addition of a new coaster

4.4.4 Testing

The success of the QR scan can be seen in the firebase database(Figure 26) by testing a generated QR code that follows the correct comma separated value format. The themepark should be added as a new document and the ride should be added as a field inside that document. After testing with different themeparks and rollercoaster names, the database has correctly updated with the values being put in the correct places.

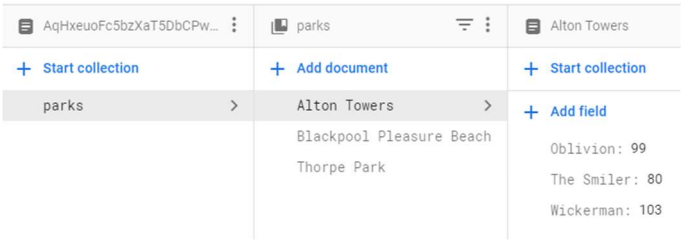


Figure 26: Results of testing algorithm

4.5 User Profile

Now that the database is full of data to display, progression can be made on the code to display that data in the user profile. The decision to use an expandable list view in order to display the data was influenced by the results of the survey, as responses highlighted that the users would want to see the theme parks they have been to and it is important to them that the coasters are grouped by the theme park they are at. An expandable list view would enable the app to display the list of themeparks as the parent list, and then the list of coasters as a child list under the theme park they belong to.

4.5.1 Implementation

To create the expandable listview, a custom listview adapter is required. The list adapter takes a list<string> and map<string,list<rideData>, where the list<string> is the list of theme park names, and the map consists of theme park names as the key and rideData as the object. rideData is a custom class (Figure 27) designed in order to help with the reading of firestore data, the class has a string and object member variable so that each field in the themepark document will be stored as one rideData object: the name variable will store the field name (roller coaster name) and the heartrate variable will store the field value (the heartrate).

```
public class rideData {

    String name;
    Object heartRate;

    public rideData(String name, Object heartRate) {
        this.name = name;
        this.heartRate = heartRate;
    }

    public String getName() { return name; }

    public Object getHeartRate() { return heartRate; }
}
```

Figure 27:Custom rideData class

The list adapter iterates through the list and map, in order to create parent and child views for each park and coaster beneath it. The view displays the read data by altering the textViews in the layout files for the parent & child list items(Figure 28). getGroupView() is responsible for setting the parent (theme park) textView, it creates a string equal to the string at the current list position and then sets the textView in the list item to the value of that string. getChildView() is a similar function for the child list items, where a rideData object, string, and object is created to hold the data of the map at the current position. The textViews then use the .setText method to change the text to the values of the new variables.

```
@Override
public View getGroupView(int groupPosition, boolean isExpanded, View convertView, ViewGroup parent) {
    String parkName = (String) getGroup(groupPosition);
    if(convertView == null){
        LayoutInflater inflater = (LayoutInflater)context.getSystemService(context.LAYOUT_INFLATER_SERVICE);
        convertView = inflater.inflate(R.layout.list_parent, root: null);
    }
    TextView parkTXT = (TextView)convertView.findViewById(R.id.TVParent);
    parkTXT.setText(parkName);
    return convertView;
}

@Override
public View getChildView(int groupPosition, int childPosition, boolean isLastChild, View convertView, ViewGroup parent) {
    rideData coasterData = (rideData) getChild(groupPosition,childPosition);
    String coasterName = coasterData.getName();
    Object heartRate = coasterData.getHeartRate();
    if(convertView == null){
        LayoutInflater inflater = (LayoutInflater)context.getSystemService(context.LAYOUT_INFLATER_SERVICE);
        convertView = inflater.inflate(R.layout.list_child, root: null);
    }
    TextView coasterNameTXT = (TextView) convertView.findViewById(R.id.child_park);
    TextView coasterHRTXT = (TextView) convertView.findViewById(R.id.child_hr);
    coasterNameTXT.setText(coasterName);
    coasterHRTXT.setText(""+heartRate);
    return convertView;
}
```

Figure 28: List adapter functions to change text to database data

Testing of the listview was first tested using hardcoded data, to check that the listView works when provided with the correct data and to avoid any potential bugs in the code when getting the data from the database. Once the listView was observed to work with the hard coded data, an algorithm needed to be implemented in order to read the Firestore data into the correct data structures.

The algorithm (Figure 29) needs to complete 2 things: get the document names into a list of strings, and for each document, insert each field into a rideData object and map this as a value to a string of the documents name.

```
listParks = new ArrayList<>();//list of document id's in user.parks
listCoasters = new HashMap<>();//<String,map<rideData> where string = document id in user.parks, <rideData> = field of document (String,obj)

/*
* 1. get list of user.park documents and insert into listParks
* 2.for each document in user.parks, convert each field into rideData obj & insert into listCoasters with document ID as key
*/

db.collection( collectionPath: "users").document(firebaseAuth.getCurrentUser().getUid()).collection( collectionPath: "parks")
        .get()
        .addOnCompleteListener((task) -> {
                if(task.isSuccessful()){
                    listParks.clear();
                    for(QueryDocumentSnapshot document : task.getResult()) { //for each document in parks:
                        listParks.add(document.getId());
                        Log.d(TAG,  msg: document.getId() + " => " + document.getData());
                        Map<String, Object> a = document.getData();
                        Iterator it = a.entrySet().iterator(); //iterator for each item in document
                        List <rideData> tempList = new ArrayList<>();

                        while (it.hasNext()){   //while there is items in document:
                            Map.Entry pair = (Map.Entry)it.next(); //create pair of objects for each item
                            //Log.d(TAG,pair.getKey() + " " + pair.getValue());
                            rideData tempRide = new rideData( name: ""+pair.getKey(),pair.getValue()); //convert pair into rideData object
                            tempList.add(tempRide); //add rideData object to list
                            it.remove();
                        }

                        listCoasters.put(document.getId(),tempList); //add item to map (String of doc name, list of rides)
                    }
```

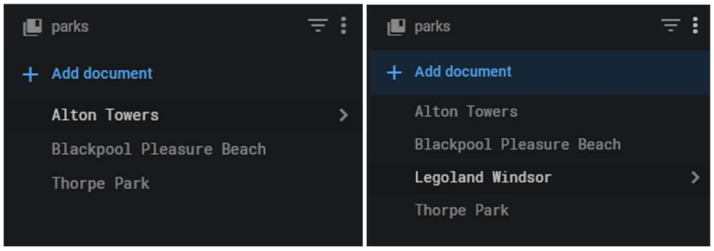
Figure 29: Algorithm to read data into correct data structures

First, all data from the users "parks" collection is obtained from the database and stored in a variable "task.getResult". A for loop is then used to iterate through each document in the collection. Then, the name of the document is put into the list of parks, then a map is created to store the contents of the current document and an iterator is also created in order to iterate through the map. The while loop iterates through the map until the end, and each iteration creates a pair of objects to store the field and value. This pair of objects is used to construct a rideData object which is then added to a temporary list of rideData objects. At the end of the document, the temporary list of objects is put into the listCoasters map as a value, along with the document id as the key. This then allows the hardcoded test data to be removed and the new listParks list and listCoasters map can be used to construct the expandable listview (Figure 30).

```
expandableListAdapter = new MyExpandListAdapter(getActivity(),listParks,listCoasters);//create list adapter with data from firebase
expandableListView.setAdapter(expandableListAdapter);
```

Figure 30: Code to create expandable list view with database data

One idea was to have the theme parks list in alphabetical order, this would benefit the user at it would be easier to navigate through the list to find the park they are looking for quickly. This could be achieved by performing a sort operation on the listParks list. However, on further investigation, Firestore already stores the list of theme park documents in alphabetical order, meaning that the created listParks list is already stored in the correct order. This can be seen in Figure 31, where Legoland Windsor is inserted between Blackpool Pleasure Beach and Thorpe Park, instead just being added to the bottom of list as the most recent addition.



=

Figure 31: Adding Legoland to check the automatic sorting

4.5.2 Testing

Using the app, the listview can be tested by opening the profile page and comparing the listed data to the data in the database. Figure 32 shows the (parent) list of theme parks. When a theme park is selected, the (child) list of roller coasters for that theme park drops down and displays the coaster name and heart rate (Figure 33). To check that this displayed data is correct and read correctly, it can be checked against the raw data in the database (Figure 34).

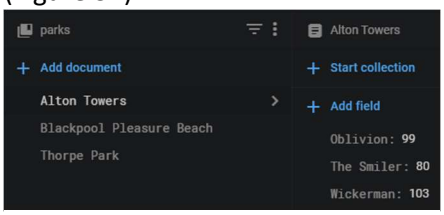


Figure 34: Comparison of database data

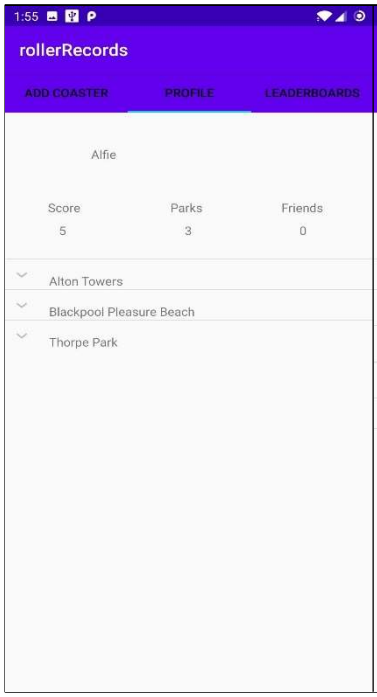


Figure 32: Testing the list of theme parks

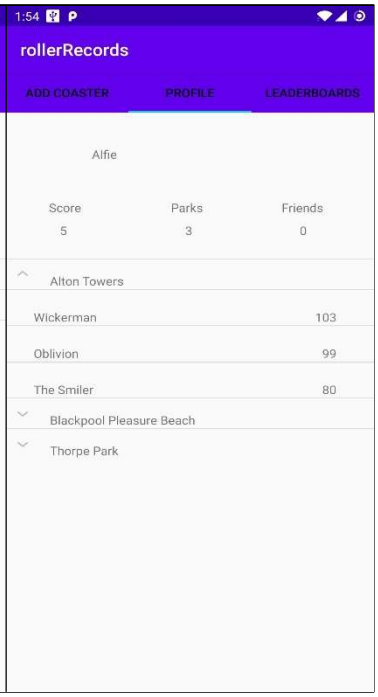


Figure 33: Testing the list of coasters

4.6 Heartrate implementation

4.6.1 Investigation

The investigation into how to obtain the users heartrate resulted in the design decision that the user will have to manually record their heartrate on their smartwatch whilst riding the roller coaster. This instance of heartrate data would then be uploaded automatically to the Google fit store, where all google fit data is kept (Figure 35). This store can be accessed using a variety of different fit APIs, but most importantly the History API. The history API allows for access to all different types of fitness data in the fitness store, this means that huge amounts of data can be obtained in the app and can be manipulated to suit various different needs.

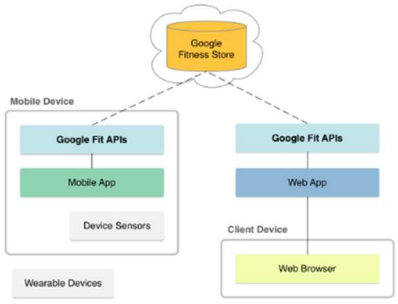


Figure 35: Google fit platform diagram

To enable the app to use this API, The app needs to request the neccesary permissions to access the body sensors. This can be achieved through the normal android permission granting procedure, however the app alson needs OAuth permissions in order to access the heart rate data type inside the Google Fitness store. These permissions can be granted by calling functions in order to check that the users Google account has granted the authorization. Once the app has requested all the required permissions, providing the user has accepted, the app is free to access the requested data at will.

4.6.2 Implementation

Before work could begin on implementing the Fit API, the permissions needed to be requested from the user. This included asking for permission to access the body sensor, and the OAuth permission to access the heart rate data. The app will request the body sensor permission alongside the camera permission, which takes place once the user has successfully logged into the app, there is no specific

reason for this choice other than it helps keep the permissions in one place and is therefore less annoying for the user. The OAuth permissions are requested once the user launces the QR scan activity, this is to help the user understand what the data is being used for, and as the data is only accessed when a QR code is scanned so it makes sense to request the permission in the activity where it is used.

```
public void setFitnessOption(){
    fitnessOptions = FitnessOptions.builder()
            .addDataType(DataType.TYPE_HEART_RATE_BPM,FitnessOptions.ACCESS_READ)
            .build();
}

public void requestGoogleFitpermission(){
    GoogleSignInAccount account = GoogleSignIn.getAccountForExtension( context: this,fitnessOptions);
    GoogleSignIn.requestPermissions(
             activity: this,GOOGLE_FIT_PERMISSIONS_REQUEST_CODE,
            account, fitnessOptions
    );
}
```

Figure 36: Functions to request OAuth permissions

To request permissions, the fitness options need to be set. This involves creating a builder where any datatypes the app needs to use are specified and the access it requires (read / write). These options are then used when the permissions are requested in order to ask for the specific permissions needed in the GoogleSignIn.requestPermissions() method (Figure 36).

Permission checks can be performed using GoogleSignIn.hasPermissions(), this enables the API functionality to only take place if the app has decided that the user has given permission. In the context of the app, the API is used when the QR scanner has detected, so all code is written in the barcode detectors “receive detections” function.

```
if(isGoogleFitPermissionGranted()){
    //get heartrate here:
    //set timeframe for heartrate data
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(new Date());
    long endTime = calendar.getTimeInMillis();
    calendar.add(Calendar.HOUR, amount: -1);
    long startTime = calendar.getTimeInMillis();

    //set read request to read HR data in timeframe
    DataReadRequest readRequest = new DataReadRequest.Builder()
            .aggregate(DataType.TYPE_HEART_RATE_BPM,DataType.AGGREGATE_HEART_RATE_SUMMARY)
            .setTimeRange(startTime,endTime,TimeUnit.MILLISECONDS)
            .bucketByTime( i: 1,TimeUnit.HOURS)
            .build();

    final GoogleSignInAccount account = GoogleSignIn.getAccountForExtension(getApplicationContext(),fitnessOptions);
```

Figure 37: Setting up variables for accessing Fit store

```
//access history client
Fitness.getHistoryClient(getApplicationContext(), account) HistoryClient
        .readData(readRequest) Task<DataReadResponse>
        .addOnSuccessListener((OnSuccessListener) (dataReadResponse) → {
                //handling getting heartrate value:
                List<Bucket> buckets = dataReadResponse.getBuckets();
                if(buckets.isEmpty()){
                    Log.d(TAG, msg: "dataset is empty for " + account.getDisplayName());
                }
                Bucket bucket = buckets.get(0);
                DataSet dataSet = bucket.getDataSet(DataType.AGGREGATE_HEART_RATE_SUMMARY);
                List<DataPoint> dataPoints = dataSet.getDataPoints();
                Log.d(TAG,dataSet.getDataPoints().toString());
                Object hr = new Object();
                //get single hr reading
                for(DataPoint dp : dataSet.getDataPoints()){
                    for(Field field : dp.getDataType().getFields()){
                        hr = dp.getValue(field);
                    }
                }
```

Figure 38: Code to get the heart rate BPM value

Figure 37 shows the code to set up the time scale variables to read the data from, as the app only needs the most recent heart rate data it only needs a small time scale to read from so it has been set to the past hour. The data read request is then used to specify which data should be requested from the fit store, for this app only the heart rate BPM is needed, however here many different data types can be requested as long as the permissions have previously been granted.

The code in Figure 38 shows the use of the History API. The response from the fit store is stored in a list of buckets which is then split into a list of data points, as buckets consist of different data points. The variable hr is used to store the final heart rate and eventually used to put the heart rate into the fire store database. The code loops through the datapoints in the dataset and sets the hr value to the last data point, which is the minimum heart rate. This is because there is no documentation to show how to access the max or average heartrate directly.

The hr variable is then used in place of the static data in the code where the app writes the QR data to the database (Figure 39).

```
documentReference = db.collection( collectionPath: "users").document(firebaseAuth.getCurrentUser().getUid()).collection( collectionPath: "parks").document(vals[0]);
final Map<String, Object> map = new HashMap<String, Object>();
//create map of coaster name & heartrate value
map.put(vals[1], hr.toString());    //map.put(coaster,heart rate)
```

Figure 39: Using the hr variable to write heartrate value to database

4.6.3 Testing

Testing this code follows the same method as testing the QR scanner, although heart rate data is needed in the fit store. To add data to the fit store, on a smart watch, the “read heart rate” button in the google fit app will take a short reading of the wearers current heart rate and send it to the fit store. After doing this, and checking via the Google fit mobile app that the data has been stored, a QR code can be scanned and then the Firestore database can be consulted to see the new record with the heart rate value(Figure 40).

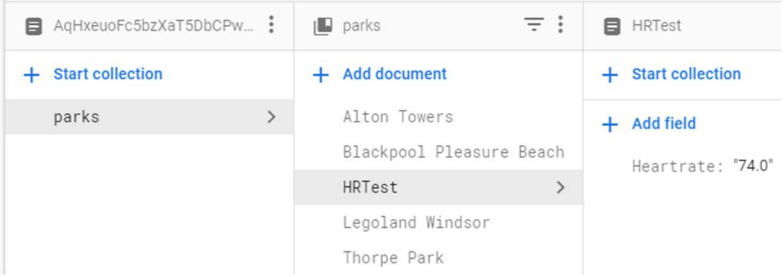


Figure 40: result of testing heart rate value

5.RESULTS AND DISCUSSION

5.1 Testing

5.1.1 Functionality testing

Although some limited testing has been carried out during development to test functions independently, more rigorous testing needs to be complete to test that different functions work together as a whole. As the app has a fairly simple main flow it is easy to test how well, or if, the different components of the app work together and achieve the intended result; the main flow is as follows: 1. Sign in, 2. Scan a QR code, 3. Heart rate is captured, 4. Database is updated, 5. Database changes can be seen in user profile.

For a fair and accurate test, a fresh account was used so that no test data would influence the performance of the app and the app would be acting as if it were a fresh install on a user's device.

Test 1: Sign in functionality

Testing	Method	Expected results	Actual results	Success? (Y/N)
New account document with unique id will be created in database on first sign in with that account.	Load up the app on the sign in page, chose an account to sign in with.	New account document will be added to the "users" collection, inside the users document should also be fields for total ridden coasters and total visited parks.	New account document appears in database, with the required fields value set to 0.	Y
Signing into an account which has signed in previously.	Clear cached data from the app and launching it again to prompt the sign in page, and then selecting an account which already has a document in the database.	App will continue as normal and no changes will occur in the database.	App proceeds to next activity and no changes happen in the database,	Y
User does not need to sign in each time app is launched.	Close app while an account is signed in and relaunch app.	Sign in page will be bypassed as the app remembers there is an account already signed in.	Sign in page is bypassed and main activity is launched.	Y

Test 2: QR scanning

Testing	Method	Expected results	Actual results	Success? (Y/N)
When scan activity opens, camera viewfinder will appear.	Give app camera permissions and open the scanner activity.	View finder will show a live feed of the device's camera.	View finder appears and live camera feed works.	Y
Camera can recognise a QR code.	Open scanner and point at a generated QR code.	Scan activity will close, and device will vibrate to confirm code is scanned	Activity closes and device vibrates.	Y

Test 3: Capturing heart rate

Testing	Method	Expected results	Actual results	Success? (Y/N)
App has access to Google fit store.	Record a heart rate, then scan a QR code to run the apps code to get the heart rate and then log the bpm value in the console logcat.	A heart rate value will appear in the log cat console.	Heart rate value is logged in console.	Y

Test 4: Database updating

Testing	Method	Expected results	Actual results	Success? (Y/N)
"parks" collection is created in user document.	Scan a QR code on a new account for the first time.	"parks" collection is added to user document, containing a document with the ID of the specified theme park.	"parks" collection is created, and it contains a document of the specified theme park.	Y
New rides belonging to an existing theme park are added to the existing document and do not create a new document.	Scan a few QR codes with the same theme park but different ride names and observe changes in database.	Each ride will become a field inside the existing theme park document.	Each new ride is added as a field in the existing theme park document.	Y
New theme parks are added as a new document in the "parks" collection.	Scan QR codes with different theme park names, but the same ride name and observe changes in database.	After scanning, the new theme park will appear as a document in the "parks" collection.	New theme parks are inserted into the "parks" collection as documents.	Y
Total theme park counter & total ride counter updates on scanning of QR code.	Generate a QR code with a new theme park and ride and scan it, observing changes in database.	Both counters should increment as it is a new theme park and a new ride.	Both counters increment.	Y
Total ride counter updates.	Generate QR code with existing theme park but new ride and scan it, observing changes in database.	Only total ride counter should update as the park has already been counted for.	Only ride counter increments.	Y
No counters update when the same code is scanned multiple times.	Generate a QR code, scan it more than once and observe database changes.	No counters will update as the park and ride has been added already.	Ride counter increments.	N
Heart rate value writes to database as value for the roller coaster field.	Add heart rate data to fit store, then scan QR code to add new ride, observe changes in data base.	New ride will be added as a field per the QR code, with the heart rate bpm (matching the	Heart rate value is stored in the correct ride field.	Y

		console logged value) as the value.		
Scanning a QR code when there is no recent data in the google fit store will write “no data” as the value instead of a bpm.	Scan a QR code without recently recording a heartrate, observe database changes.	New ride field will have “no data” as its value.	New ride field has “java.lang.Object@bc1b6a9” as its value	N

Test 5: Reading database data into user profile

Testing	Method	Expected results	Actual results	Success? (Y/N)
User profile displays accurate total coaster & park count.	View profile page and check total coaster & park values against database.	Values should be the same as database.	Values are the same as database.	Y
User profile updates Total coaster & park count after scanning QR code.	Scan a QR code to add new park & coaster then go back to profile to see if values have updated.	Values increment after scanning.	Values incremented after scanning.	Y
User profile lists all users visited theme parks.	View profile page to check the list of theme parks matches the list of documents in the “parks” collection.	List of parks on user page should match the list of document IDs in the parks collection.	List of parks matches the list of documents.	Y
User profile lists all relevant coasters as a child list under the corresponding park in the list of parks.	Select a theme park from the list on the user profile and check the list of coasters against the list of fields in the database document.	List of coasters should match the fields in the database document.	List of coasters matches the fields in the document.	Y
After scanning a new theme park QR code, the park is listed in the user profile.	Scan a QR code for a new theme park, check user profile to see it if is listed.	New theme park is listed in alphabetical order.	New theme park is listed in alphabetical order.	Y
After adding a new ride for an existing theme park, it is listed under said park in the user profile.	Scan a QR code for a new ride, check user profile to see if it is listed in the correct park.	New ride is listed under the correct park.	New ride is listed under the correct park.	Y
User can scroll through list of theme parks when it becomes too large to fit on one screen.	Add large number of parks to database, and check user profile to see if list becomes scrollable.	List of parks will become scrollable and user can scroll through list.	List of parks becomes scrollable and user can scroll through list.	Y

5.1.2 Accessibility testing

Accessibility testing is required to ensure that the app is usable by all intended users. Accessibility can cover many different factors in software engineering as a whole, however this report will focus

mainly on the UI as the app is not directly intended to suit a specific need, it will follow the rule that if a person can ride a roller coaster, they should be able to use the app. Although this rule could be applied in a larger project, for the scale and timeframe for developing this app it is unlikely, or impossible to cater for more severe needs such as blindness / physical disabilities.

One method of testing the Accessibility of the app is by using an app provided by google called Accessibility Scanner. This app scans screens and highlights any areas where there may be an accessibility issue and gives reasons why and suggests how to correct it. The benefit of this is that it can provide some feedback which may get discovered in user testing but gives a more accurate and concise explanation using the proper terms rather than an inexperienced user suggesting things which can get misunderstood or misconstrued in communication.

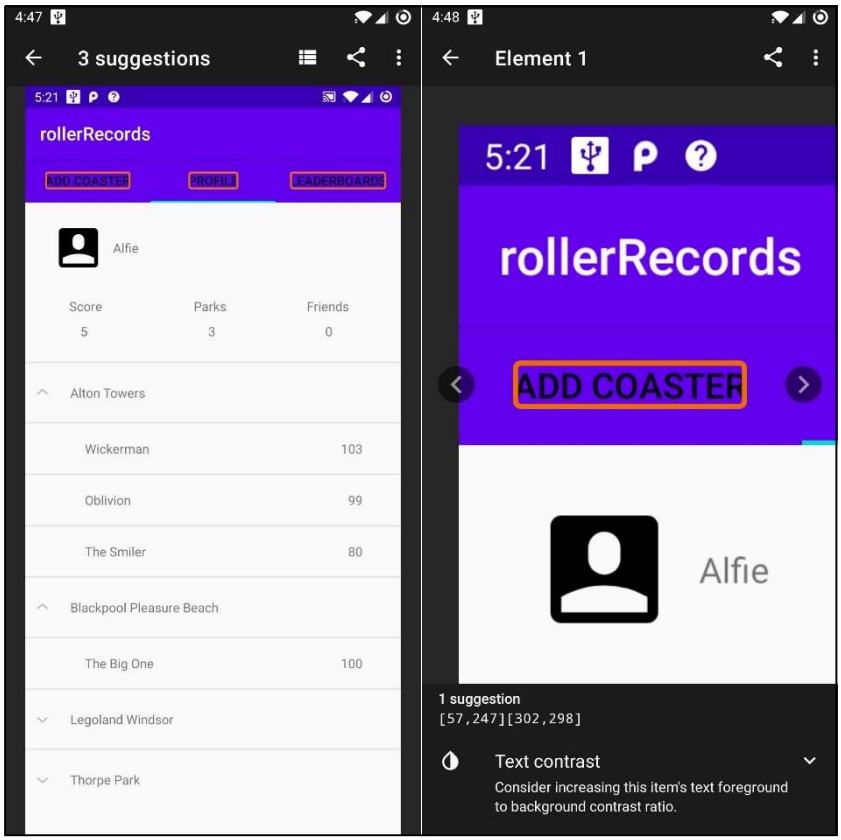


Figure 41: Results from accessibility scanner

Figure 41 shows the results from the accessibility scanner. The accessibility scanner has highlighted that the tabs for the different pages have a low background to foreground ratio. This means that the text might be too difficult for some users to read as the black text does not stand out too well against the purple background. This is an easy fix as it only requires changing the text colour to something brighter such as a white or light grey.

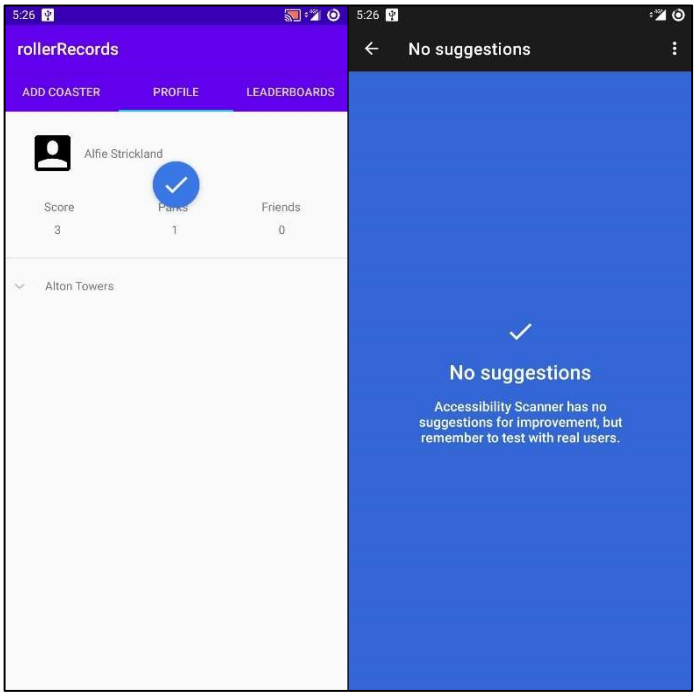


Figure 32: Accessibility scanner results after changes

Figure 42 shows the accessibility scanners results after changing the text colour to a light grey. Light grey was used as it has a high contrast ratio against the purple background, and it also helps differentiate between the (grey) tabs and the (white) title. The accessibility scanner shows that there are no further suggestions to the UI of the app, meaning that it should be accessible for the vast majority of people.

6.CONCLUSIONS AND FUTURE WORK

6.1 Meeting project aims

The Aim of this project was to implement mobile sensors in a new app to improve upon the implementation of other existing services. The roller records app achieves this by first using the QR scanning functionality, which is an improvement on existing services as it provides a fast method of adding coasters to the account with little to no user input. The QR scanning also makes it more difficult for users to lie about the coasters they have been on as they cannot simply scroll through a list of coasters and add them to their account at will, the full implementation of this app would have the QR codes situated at the rides they represent in real life meaning that the user does actually have to go on the ride to add it to their account. This is an improvement as no other alternative service has any functionality to prevent the users from lying, aside from account moderation as a preventative measure but this relies solely on users manually reporting accounts, and moderators manually reviewing those reports and taking appropriate action.

Another sensor the app uses is the heart rate sensor. This sensor is used to log users heartrate alongside the record of each ride, although this feature is novel in the current implementation, it could be used in future versions of the app to implement score boards and leader boards to view who has the highest / lowest heart rate for a particular ride. This feature is an improvement on existing services as it provides another dimension to the app other than just logging rides, users may find it interesting or entertaining to view others heart rate especially the heart rate of friends / family.

6.2 Legal, Social, Ethical & Professional issues

When comparing the final product against the potential issues that were identified, no additional issues manifested during the progression of development. This could be due to the fact the project had no real problematic issues from the start, and development managed to stay on track and achieve what was set out from the beginning. However, issues predicted during planning were taken care of as follows:

Type of issue	Issue from plan	Addressing the issue
Legal	Security of user information (passwords)	Using the google sign in feature meant that the app did not have to handle any storing of user passwords. Access to the account relies on access to the users Google account meaning the responsibility of storing passwords and maintaining account security is passed on to either the user, or Google.
Social	Accessibility of the app.	Accessibility has been tested and accounted for in the design of the product. However, this cannot account for all disabilities or users due to limitations of technology and the scale of the project.
	Users concerned about the use of heart rate data.	When the app asks for permission to the heart rate data, a short explanation is given to show what data is accessed.
Ethical	Users altering their heartrate artificially (drugs etc)	There is no real way to completely control how a user is going to use the app, however a pop-up warning in the form of a dialogue box could be used to warn users not to attempt to increase / decrease their heart rate. However, with the current implementation of the app, there is no real competitive reason to increase / decrease heart rate as there is no friends system / leader board functionality. The implementation of this would cause more concern for this issue as then people would have more reason to want to

		attempt to change their heart rate.
Professional	Use of app causing a disruption at theme parks.	The app use of QR codes means that adding a ride is very fast. This should cause minimal disturbance to the park operations as people may not even have to stand still to scan a code, modern mobile cameras should be good enough to identify and scan a QR code whilst the user is on the move. This app would also require permission from the theme park to operate as intended so it would not be a problem if they have endorsed and implemented it in the first place.
	Application should reflect the wants of potential users and not just the desires of the designer.	Through the design phase, the use of a questionnaire helped in making the requirements meet the wants and needs of the users and not the designer. The responses helped shape the functionality of the app (e.g. QR scanning over NFC tags) and also shaped features such as the categorization of rides and other features which did not get to be implemented in time such as the friends system and maps to show ridded rides.

6.3 Synoptic assessment

This project has helped to learn how to design and implement a larger scale project. With little experience in creating mobile apps prior to this, it has been challenging to create the app from scratch and investigate the best way to approach different situations. One of the largest tasks was learning to use the Google fit API, as there was little user created tutorials and even those provided by google were outdated meaning that learning how to use the API relied on reading the provided documentation and relying on previous knowledge and skill to adapt it into functionality for the app. Other skills developed from previous work was also used in this project. Using logic to develop algorithms was needed to implement the list feature in the app, the data base needed to be read in a specific way to construct the required data structures, and then those structures needed to be looped through in a specific way to create the expandable list view.

More knowledge specific to creating android apps have also been gained during this project. Smaller things such as getting to grips with android studio will be helpful in future projects making android apps. Future apps could now get off the ground quicker as the basic steps required to set up an app

such as implementing the backend or implementing the different activities or fragments layout files have become easy.

Firebase will definitely be used in future apps where possible and necessary. This is due to the well-designed documentation and set up guides which make it quick to implement the different features such as the authentication and database.

6.4 Direction of project in the future

The Roller records app could take a few different directions; one direction being a more social media-esque app, where the app focuses more on implementing friends lists and “feeds” / “time lines” where friends newly ridded rides would appear similar to a Facebook post or tweet. Another direction the app could take is a more simplistic logging app that just focuses on the uses rides, similar to how it is currently but perhaps more features to filter the ridden rides into their more detailed categories such as the manufacturer or material of ride (steel / wood / hybrid). This may be favoured by users as some did state during the research phase that the other existing solutions “did too much” and they just want a simpler logging app.

One main change the app would need if it were to become a real commercial app is the QR codes. As of now anyone could generate a QR code to add a ride to their account as it uses a simple comma separated string. To avoid users creating their own codes, the QR codes would need some form of encryption or ID based system to create a code that is incomprehensible unless you use the app.

Another major change the app needs before it could become a viable alternative to existing services, is updated UI. The current UI works and is useable, however for people to really consider using it, it would need far more modernisation and design work to create a visually pleasing colour scheme and interface and to keep the user interested in the app.

To enable the more potential users to use the app, an IOS version of the app would also need to be created. As the app in its current state would not be usable on an IOS device, either a native IOS app would be needed, or a recreation in an SDK such as flutter would enable ports to be made onto both Android and IOS devices from one code base.

6.5 Evaluation of entire project

Overall, this project has provided a valuable learning experience and insight into what planning and research needs to be created and carried out in order to make a product which not only meets the requirements of an individual but the wants and needs of a larger user base. Working on the mobile app has been an enjoyable experience and has potentially inspired a future career in mobile app development. App development seems more enticing than pc or web development as it is easier to be more creative with the different tools and features a modern mobile is equipped with, there are so many different sensors that could be incorporated into an app to simplify or enhance day to day tasks, plus making personal apps which can be used instead of other commonly used apps is rewarding and can also be shared with others easily.

Thanks to the MoSCow methodology the project prioritised requirements and therefore managed to achieve all the requirements listed in the “must” section, this means it has incorporated the base features it needs to provide the service it was meant to. However due to time constraints caused by unexpected delay in parts of the project the “should” and “could” requirements were not met. This will not impact the core functionality of the app, though, as these requirements are described as so to make sure they are not crucial to the more core requirements of the app.

Overall, the project has achieved a sufficient stage whereby the app meets the necessary requirements to fulfil its purpose. Its current stage would also be a good point to release a beta version of the app to a limited audience in order to perform a mass testing stage and collect data on the use of the app, and its performance. Feedback could also be obtained from the users to implement changes to the app in terms of UI design to help create a better-looking app that the wider public find appealing.

Bibliography

Google, n.d. *zxing.* [Online]
Available at: https://opensource.google/projects/zxing
[Accessed 20 February 2020].

Hawkins, J., n.d. *trackrecord.pdf.* [Online]
Available at: jonathanhawkins.net/trackrecord.pdf
[Accessed 5 December 2019].

thePARKSMAN, 2018. *LogRide.* Orlando, Florida: s.n.

Thumann, V. S. &. T., n.d. *Coaster Count.* [Online]
Available at: https://coaster-count.com
[Accessed 18 December 2019].

Appendices

1. Survey of questions

Roller Records

survey to find out what potential users of the app want

* Required

1. Do you currently track the rollercoasters you have been on? *

   Mark only one oval.

   - ◯ Yes *Skip to question 3*
   - ◯ No *Skip to question 2*

Untracked

2. Would a mobile app that simplifies tracking ridden rides change your mind? *

   Mark only one oval.

   - ◯ Yes
   - ◯ No

Skip to question 3

General

3. Are you familiar with the use of QR codes? *

   Mark only one oval.

   - ◯ Not heard of them
   - ◯ Heard of them but never used them
   - ◯ Heard of them and used them

4. Are you familiar with the use of NFC tags? *

Mark only one oval.

- ◯ Not heard of them
- ◯ Heard of them but never used them
- ◯ Heard of them and used them

5. When categorizing the rides you have been on, is it important that they are listed under the themepark they belong to? *

Mark only one oval.

- ◯ Yes
- ◯ No

6. Would you want to view the themeparks you have visited in a more interactive way such as pinpointed on a map? *

Mark only one oval.

- ◯ Yes
- ◯ No

7. Would you be interested in viewing the rollercoaster count of other people? select those that apply. *

Check all that apply.

- ☐ Leaderboards (global / nationwide)
- ☐ Friends
- ☐ Not interested

Other: ☐ _____

2. Survey response raw data

Timestamp	Do you currently track the rollercoasters you have been on?	Would a mobile app that simplifies tracking ridden rides change your mind?	Are you familiar with the use of QR codes?	Are you familiar with the use of NFC tags?	When categorizing the rides you have been on, is it important that they are listed under the themepark they belong to?	Would you want to view the themeparks you have visited in a more interactive way such as pinpointed on a map?	Would you be interested in viewing the rollercoaster count of other people? select those that apply.
2/5/2020 21:27:59	No	Yes	Heard of them and used them	Heard of them and used them	Yes	Yes	Not interested
2/5/2020 21:28:11	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Not interested
2/5/2020 21:38:11	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/5/2020 21:44:41	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Not interested
2/5/2020 21:44:44	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/5/2020 21:45:33	No	Yes	Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 21:50:01	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/5/2020 21:59:25	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide)
2/5/2020 22:08:18	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Not interested
2/5/2020 22:13:19	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 22:15:12	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends, Keep track by photos taken?
2/5/2020 22:21:45	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide)
2/5/2020 22:22:02	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/5/2020 22:27:44	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 22:29:01	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Not interested
2/5/2020 22:31:54	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 22:32:38	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Friends
2/5/2020 22:35:50	No	Yes	Heard of them and used them	Heard of them but never used them	Yes	Yes	Friends
2/5/2020 22:37:22	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide)
2/5/2020 22:39:44	No	Yes	Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/5/2020 22:41:41	No	No	Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/5/2020 22:51:10	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide)
2/5/2020 22:52:29	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/5/2020 22:52:59	Yes		Heard of them but never used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 23:03:07	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends, family
2/5/2020 23:07:47	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Friends
2/5/2020 23:10:06	No	Yes	Heard of them but never used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 23:11:30	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 23:18:19	No	Yes	Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 23:20:11	No	Yes	Heard of them and used them	Heard of them but never used them	Yes	Yes	Friends
2/5/2020 23:22:59	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 23:24:05	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/5/2020 23:37:01	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/5/2020 23:45:07	Yes		Heard of them and used them	Heard of them but never used them	Yes	No	Leaderboards (global / nationwide), Friends
2/5/2020 23:46:46	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Not interested
2/5/2020 23:50:44	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Friends
2/6/2020 0:00:30	No	Yes	Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 0:01:50	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 0:04:09	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 0:23:27	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 0:31:30	Yes		Heard of them and used them	Heard of them and used them	No	Yes	Leaderboards (global / nationwide)
2/6/2020 0:35:02	No	Yes	Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 0:37:32	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/6/2020 0:45:07	No	Yes	Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 0:48:38	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 0:48:58	Yes		Heard of them but never used them	Heard of them but never used them	Yes	No	Not interested
2/6/2020 1:04:17	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 1:19:33	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 1:20:34	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 1:20:43	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 1:21:03	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 1:24:35	No	No	Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 1:33:38	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 1:34:12	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 1:34:51	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 1:37:32	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Not interested
2/6/2020 1:37:39	Yes		Heard of them and used them	Not heard of them	Yes	No	Leaderboards (global / nationwide), Friends
2/6/2020 1:39:01	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 1:41:27	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Leaderboards (global / nationwide), Friends
2/6/2020 1:52:07	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 1:55:23	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Not interested
2/6/2020 2:02:52	No	No	Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), People near me
2/6/2020 2:10:28	No	No	Heard of them but never used them	Not heard of them	No	No	Leaderboards (global / nationwide)
2/6/2020 2:12:45	No	Yes	Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 2:17:50	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 2:22:34	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 2:24:48	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 2:31:40	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/6/2020 2:38:42	No	Yes	Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 2:44:53	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Not interested
2/6/2020 3:09:39	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 3:38:03	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 3:41:30	No	No	Heard of them and used them	Heard of them but never used them	Yes	No	Friends, Not interested
2/6/2020 3:44:32	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 3:50:31	No	Yes	Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 3:53:57	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 4:04:36	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Friends
2/6/2020 4:11:11	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 4:11:53	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 4:12:46	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 4:22:27	Yes		Heard of them and used them	Heard of them and used them	Yes	No	Friends, It'd be cool to see which coasters the most people have ridden
2/6/2020 4:22:40	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 5:03:53	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 5:44:26	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/6/2020 5:57:02	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 6:20:39	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 6:25:56	Yes		Heard of them but never used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 6:52:13	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 6:57:05	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 8:26:59	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 10:22:38	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 10:40:20	No	No	Heard of them and used them	Not heard of them	Yes	Yes	Not interested
2/6/2020 10:52:45	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends

2/6/2020 11:31:19	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 11:38:16	No	Yes	Heard of them but never used them	Not heard of them	Yes	Yes	Friends
2/6/2020 11:50:27	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 12:13:14	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/6/2020 12:34:49	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 13:11:30	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 13:46:06	Yes		Heard of them and used them	Heard of them but never used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 15:00:07	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 15:19:04	Yes		Heard of them and used them	Heard of them and used them	No	No	Friends
2/6/2020 15:26:11	No	Yes	Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 16:09:21	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 16:24:28	Yes		Heard of them but never used them	Not heard of them	No	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 16:44:44	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 16:50:31	No	Yes	Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 17:05:33	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/6/2020 17:40:59	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 17:48:23	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 19:12:52	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/6/2020 21:11:56	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/6/2020 21:48:29	No	Yes	Heard of them and used them	Heard of them and used them	Yes	No	Leaderboards (global / nationwide)
2/6/2020 23:11:45	No	No	Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/7/2020 0:12:35	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/7/2020 0:21:00	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Friends
2/7/2020 1:32:17	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends
2/7/2020 1:37:04	Yes		Heard of them but never used them	Not heard of them	Yes	Yes	Not interested
2/7/2020 1:53:43	Yes		Heard of them and used them	Not heard of them	Yes	Yes	Leaderboards (global / nationwide), Friends
2/7/2020 2:46:07	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Leaderboards (global / nationwide)
2/7/2020 3:47:50	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Not interested
2/7/2020 4:36:39	Yes		Heard of them and used them	Heard of them and used them	Yes	Yes	Friends, Not interested
2/7/2020 9:21:40	No	Yes	Heard of them and used them	Not heard of them	Yes	Yes	Friends