C0C251

Student ID: F027740

Raffl

A Raffle Ticket App

by

Samuel Precious

Supervisor: Dr Hossein Nevisi

Abstract

The work in this dissertation seeks to extensively document the journey undertaken in creating “Raffl”. An innovative mobile application, Raffl seeks to meld the excitement and intrigue of traditional raffle systems with the practicality of modern e-commerce. It will begin with in-depth research of existing solutions and design principles and a discussion of legal and ethical issues before delving into execution in the implementation phase. It will be structured by an altered agile development framework, which is reinforced by comprehensive user testing against clearly defined requirements.

The journey is concluded with a holistic view of the development of Raffl from design to deployment, presenting it as a novel contribution to an increasingly saturated market with real opportunities for further research and development.

Acknowledgements

I am grateful to Dr. Hossein Nevisi for his guidance and support on this project.

Contents

Table of Figures

1 - Project Description

1.1 Project Overview and Problem Statement

The purpose of Raffl is to gamify online shopping to address the stagnation and lack of innovation in the current e-commerce solutions while tackling a niche in the ever-saturated market. In the modern day, most e-commerce sites are functionally the same, with minor differences in design and little innovation. Raffl seeks to change this, bringing an air of excitement to the otherwise bland online shopping market.

Most e-commerce sites today have the same basic steps when acquiring a product: you browse products that interest you and immediately buy them at the specified price. eBay somewhat solves this problem by offering engaging and exciting auctions in which users can take part. However, these can still cause unnecessary stress for buyers who do not want to monitor the price of an item constantly. That's where Raffl comes in, with a fresh look at the shopping market and the option to make small investments in raffle tickets for the chance to win big. This unique approach provides incentives for both sellers and buyers. Sellers have the chance to earn increased profits, whilst buyers can win items at minimal cost to themselves. It should also improve user experience, making online shopping more fun.

1.2 Objectives

The main objectives of this project are as follows:

1. To develop a mobile application incorporating a raffle ticket system.
2. To employ advanced UI and UX principles for ultimate user satisfaction.
3. To choose and utilise cloud services and front-end development software.
4. To rigorously test against defined requirements to ensure high quality.

1.3 Scope and Limitations

The scope of the project is heavily shaped by the resources available for the solo project, weighing up concurrent priorities, limited budget, and effective time management (limitations). To avoid scope creep, it is important to form realistic objectives at the outset and continuously reflect on development efforts to stay focused on meeting requirements. Key features to be implemented are user registration, notifications, a functional raffle system and an avenue for users to list their own items. The project will not cover actual payment systems; instead, it will use a dummy interpretation for ease of testing as it is conceptual in nature.

1.4 User Journey

The app will be user-focused, with an emphasis on user-friendly and intuitive features, borrowing what works from the industry and fixing what doesn't. The envisioned user journey is that a user will download the app, register, log in, easily navigate to a raffle, buy tickets, list an item, upload shipping details, browse live raffles in a recommended feed, track the status of the raffles, and receive notifications for relevant status updates.

2 – Literature Review

This chapter provides an in-depth analysis of existing literature related to mobile raffle applications, competitive e-commerce platforms, technological advancements in app development, and user experience design. The aim is to research and identify existing solutions and gaps in the current market, which "Raffl" aspires to fill and to establish a foundation for design and development choices. Additionally, legal, and ethical implications will be discussed.

2.1 – Competition Analysis and Market Overview

This section seeks to explore the competitive landscape that Raffl faces within the e-commerce market, along with outlining the potential gap in the market that it is poised to fill. Moreover, I will develop a detailed overview of popular e-commerce apps, along with existing raffle ticket apps, highlighting their strengths and weaknesses. I will conclude by discussing the unique offerings that Raffl brings with it, given the gap analysis, and how its features can capitalise on these opportunities to differentiate itself from the competition.

2.1.1 – E-commerce Platforms Analysis

Conceptually, there are almost no e-commerce applications on the market that match Raffl's goals and intended features. Instead, listed below are some of the most widely used and relevant applications that can be analysed.

When considering what some of the largest online marketplaces are today, offering sellers a place to list and buyers the option to choose amongst an endless catalogue of items, it may seem counterintuitive to put Raffl on a similar level. However, in pursuing this analysis, I will be able to identify and learn from both the positive and negative elements of current e-commerce platforms and identify why someone should want to use Raffl's unique offerings over these platforms.

2.1.1.1 – eBay

At first glance, the primary competition for the app is likely to be eBay. This is due to its status as the most popular second-hand e-commerce platform. Unlike most e-commerce sites, eBay has the unique selling point of a hybrid auction-house system, allowing users to bid on items as well as (where sellers choose to) the option to purchase them directly. In many ways, the convenience provided by this mechanism has contributed demonstrably towards its success. eBay has also built a reputation that users can trust, and that provides seamless access to a global network of other users who are willing to purchase or sell items.

In terms of user experience and service, eBay has amassed considerable resources, generating $9.795 billion dollars in 2022 and has been in development since 1977 [1]. Developing as many features as eBay (many of which are displayed in Figures 1-to-8 below) with the modest budget of Raffl is, therefore, quite a challenging task and would also risk compromising on quality and user experience with limited time to effectively implement. This, however, is not necessarily a negative, as a common pitfall among mobile applications is feature overload; when too many features are implemented into applications, users may

judge that the cost of learning said features outweighs the benefit of using the system [2]. Moreover, in attempting to match the impressive number of additional features, I am likely to end up with longer development times and potential visual overload for users in the early stages of the app.

As a matter of fact, on eBay, it is debatable whether each page already has too much information, meaning simple tasks like viewing purchases take longer to find, detracting from the user experience. Perhaps an example of this is in Figure 1 (below), displaying the homepage of eBay, where a large blue ribbon has been placed front and centre to draw user attention to the sale. Here, a user might find it useful to be alerted to a sale in the 'fashion' category; however, the caption giving the names of 4 separate retailers along with a date and time (instead of, e.g., how many hours/days are remaining on the sale) can overload a user with too much information. To avoid this, Raffl will be developed as a simple, minimalistic app but will still strategically use colour, which can be used in the UI in a different way from eBay. This would help set Raffl apart from its biggest competitor while offering a refreshing experience for users.

On the other hand, it is also my intention that Raffl mirrors key aspects of eBay's business model by establishing a global reach and by building a platform that makes buying and selling second-hand items effortlessly. Nevertheless, Raffl will achieve this through a different vehicle by pivoting on the unique selling point of using a raffle ticket system as a means of e-commerce.

One reason why such a system could be advantageous over an auction system for a buyer is that it can appeal to users who might otherwise have been priced out. Moreover, since a raffle ticket is a fixed cost, users do not have to frequently monitor and adjust their bids for an item they want to buy. However, some users still may prefer direct transactions, where the price is in their control, which would not be fulfilled in a raffle ticket system. Furthermore, if a user loses an auction on eBay, they are not out of pocket, whereas if they lose a raffle listing, they lose money with nothing to show for it.

Though both eBay and Raffl will have a similar, user-focused model, one substantial difference is that eBay offers a website. Consequently, I will direct the bulk of the following analysis on the eBay mobile app, where I will investigate a subset of features seen below and compare them with Raffl. Below are all the pages of eBay, along with an analysis of their features.

Figure 1 – eBay Home

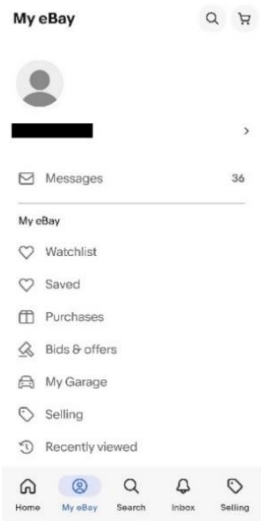


Figure 2 – eBay Profile

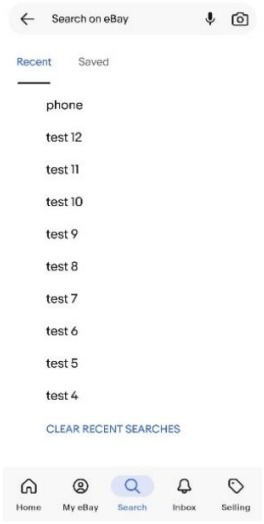


Figure 3 – eBay Search

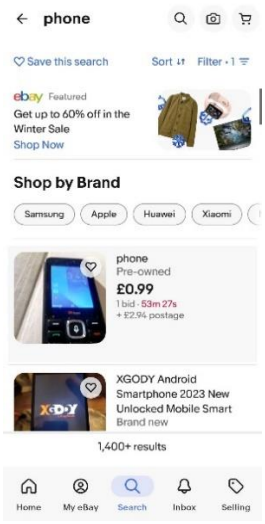


Figure 4 – eBay Results

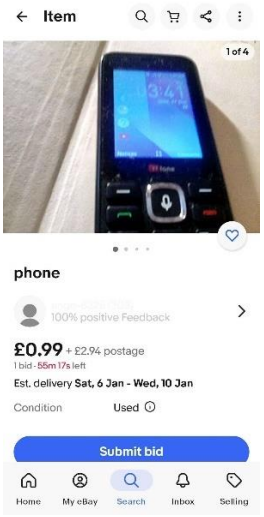


Figure 5 - eBay Listing

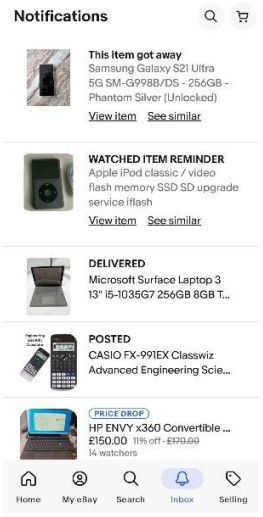


Figure 6 - eBay Inbox

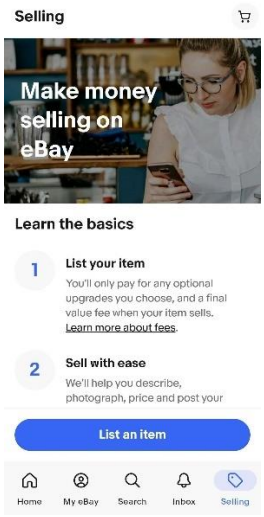


Figure 7 - eBay List #1

Figure 8 - eBay List #2

As the dominant small-time seller application, eBay undoubtedly has many standard features that are important to user experience, which will be seen in Raffl as well. For instance, sellers can list their own items, as shown in Figure 7, which are then accessible via the search and recommendation tabs, as in Figure 3 and Figure 4. Notifications are also versatile and separated into their own page (Figure 6), where users can get up-to-date information related to their activity. These systems are commonplace in industry, such as in Amazon and YouTube. For example, on YouTube, the concept is similar, but with videos, where creators can list videos that are accessible via the search engine and recommendation tabs. Users interested in said channels are then notified with status updates. Intuition is shaped by learning [3], and while such systems remain standard, adopting a similar structure in my app will be advantageous. This is because users have already learned them and will thus have a rough understanding of the system before even using it.

As mentioned before, some features offered by eBay will not be seen in Raffl. This is due in part to some features being unsuitable, whilst others are not possible within the current development timeframe. The main difference between Raffl and eBay is the way items are

purchased. In Raffl, the focus is on raffling tickets to sell items in place of a ‘buy it now’ or ‘auction’ approach to selling items. Though it is still possible to incorporate such features, I believe including these features risks overcomplicating the app, detracting from the more niche experience offered by a raffle ticket system.

Additionally, a key feature of eBay that will offer unique problems for Raffl is refunds. In most e-commerce applications, if users have problems with an item, they can send it back for a refund, but this isn’t an option with Raffl. Users buying tickets may be deeply disappointed if they do not receive what they were advertised after winning and are only refunded for the tickets they bought. A non-standard approach must be taken to resolve this, such as offering refunds in the form of revenue for sold items. For example, if ten tickets were sold at £5 each, and a user who bought one ticket never received their item, the profit earned for said item would go to the winner. This system would, however, be open to abuse and would require an operational support team to hold requests, which will not be possible due to the resource constraints of this project and its conceptual nature.

2.3.1.2 – Facebook Marketplace

Another app that could potentially provide fierce competition to Raffl is Facebook Marketplace. This platform benefits from up to 40% of Facebook’s 3 billion monthly users who use the platform to shop, as well as 250 million global sellers [4]. Unlike the more traditional e-commerce platforms out there, Facebook Marketplace is not a dedicated app and instead is implemented to complement Meta’s (was Facebook) social media platform, Facebook. This provides social media and local communities for transactions and aims for local reach within a community to sell items as opposed to the sparser geographical approach Raffl will take. Below is a subset of the features (shown in Figures 9-12) that will be analysed for inspiration and to identify potential gaps.

Figure 9 - FBMP Home

Figure 10 - FBMP Search

Figure 11 - FBMP Listing

Figure 12 - FBMP Messaging

The main selling point of Facebook Marketplace is the unique social media aspect offered by it. To list an item on the marketplace, users must have a Facebook account. This means that it’s easier to identify legitimate sellers based on profile content and creation date. Users are also easier to hold accountable for their actions, such as scams, as it is tied to their actual

profile and, as such, involve a level of personal risk. This is a double-edged sword, however, as there is significantly less privacy offered on the marketplace, especially since sales often take place through direct message chains (Figure 11 and Figure 12), leaving sellers open to harassment and social engineering, among other things.

Though FBMK benefits from a large social media platform to piggyback on, this could be substituted in seller reviews to enhance trust and accountability in raffles held. Whilst this would not introduce as much trust as Marketplace, it would avoid privacy invasions whilst also protecting sellers from harassment, so it is a relatively good middle ground. Raffl could also incorporate messaging as well as in place of a review system, as this would also add trust and give buyers the opportunity to know more about items up for grabs. Whilst both features would be beneficial to Raffl, they currently fall outside the intended specification for the product but are worth considering for future iterations.

Another big difference between Raffl and Facebook Marketplace is the scope of transactions. Facebook Marketplace focuses on a mostly local transaction system, as shown by the location setting beneath the search bar in Figure 10 (set as 'Croydon' in this example), where it is most common for users to pick up items in person. In contrast, Raffl aims for a wider reach, where postage will likely be the only method of acquiring an item. A wider range of people on the national and global market also means that sellers are more likely to appeal to a wider audience and sell more tickets. This gives them a higher potential profit and, therefore, more reason to switch to Raffl.

2.1.2 – Dedicated Raffle Card System

While in previous sections, we compared Raffl against current e-commerce platforms, the focus here will be on the existing solutions like the raffle ticket aspect of the app. This includes both a comparison of in-person raffles and a raffle-hosting website.

2.1.2a – In-Person Raffle System

In standard raffles, multiple prizes are often involved. People who show interest in said raffle can buy tickets for it, and if they win, they will immediately be rewarded with the prize allocated to their number. While not a strict rule, this is often a new item and is bought for the purpose of being put in the raffle. Raffl approaches this quite differently, as it tries to blend the world of standard e-commerce with the traditional raffle ticket system. Like auctions held on eBay, there will be just one item as opposed to the usual assortment of prizes. Furthermore, instead of being tied to a ticket at the beginning of the draw, Raffl will allocate tickets to users sequentially and randomly decide the winner when the draw ends. This results in the item always being won by a user, even if only one ticket is bought, unlike standard raffles.

Another big difference between in-person raffles and the system found in Raffl itself is the scope. Where in-person raffles will usually have a small target audience of locals in the area, the app aims to target global audiences. This allows access to a much larger pool of potentially interested buyers, which would correlate to an equally higher profit for sellers who are successful at gaining traction for their listing and selling tickets.

2.2.2b – Raffall

Next, I will investigate Raffall, an online implementation of standard raffle ticket systems. It enables large commercial clients to host private raffles, highlighted in Figure 15, often with a focus on charity events or conferences. After surveying the market for applications, I found that this is the most similar application to Raffl on the market to date, despite the lack of a mobile presence.

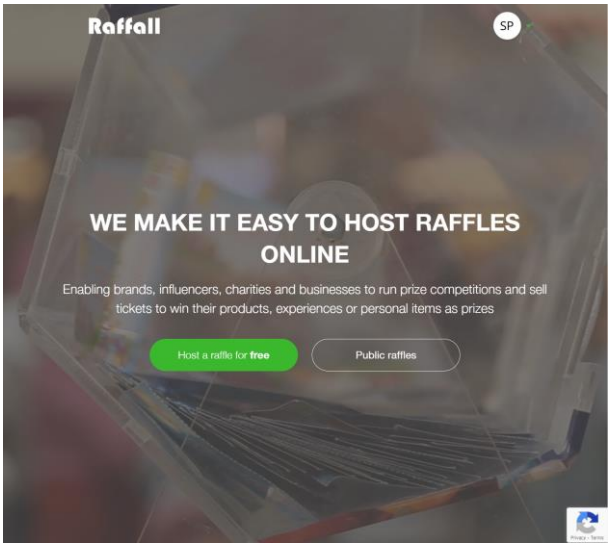


Figure 13 - Raffall Home

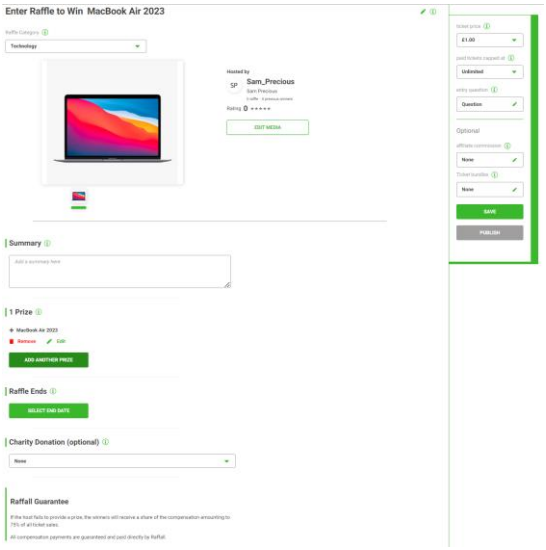


Figure 14 - Raffall List Item

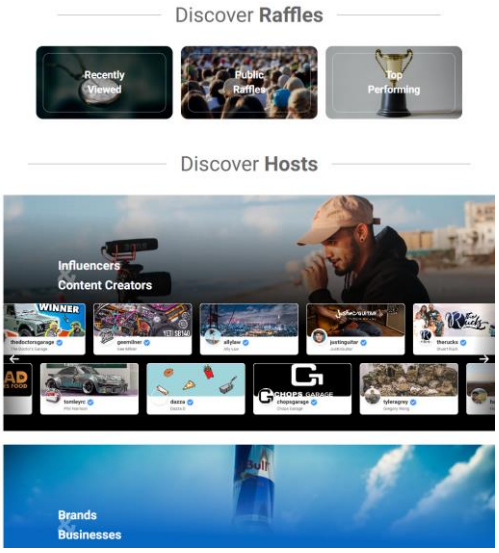


Figure 15 - Raffall Discover

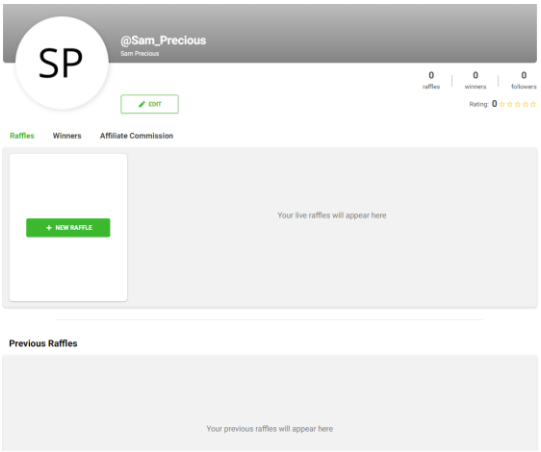


Figure 16 - Raffall Profile

A big difference between Raffl and Raffall is platform availability. Whilst Raffl is mobile-only, for reasons stated in 2.2.1, Raffall is web-based. It appears that Raffall used to have a mobile presence, appearing on the Apple App Store from 2016-2018, but no longer exists today[5] for unknown reasons. It never received an Android app either, so the target demographic is different to Raffl. Upon further research into the application, it can be noted that the business values statement included: ‘enabling brands, influencers, charities, and businesses to run prize competitions’ on the homepage shown in Figure 13. This reinforces

the fact that Raffall aimed to provide larger, often corporate clients and charities with the option to raffle off items at prices. This also takes the more standard raffle approach of multiple items per draw, which is vastly different to Raffl, which aims to offer raffles as an alternate means to sell single items they no longer want. Instead of using e-commerce competitors, users can do the same style of selling but bring an element of excitement to it. Raffall's priority is also clear in the discover page, where public raffles exist, but they are not the focus of the application, as shown by the UI, which seems to draw focus to influencers and big brands. By combining similar functionality to Raffall but with a more mainstream audience like eBay and Facebook, Raffl offers a unique point in the market that is not yet filled.

The core functionality of both applications, outside of target audiences, will be relatively similar. Raffall has the opportunity for sellers to list items, set the price of their tickets, and set the end date of their tickets and other features. They offer a limit to the number of tickets sold, but this is unlikely to make it to Raffl as I do not see any purpose in limiting sales for its audience. The other features mentioned, however, are very likely to make it to Raffl, as they are standard features you would expect from a paper-based raffle service. Raffall is another platform with user reviews, which could also appear in Raffl, as discussed earlier.

One final feature that can be seen on the listing page is the 'entry question' selection. This is for legal reasons and seems to exist as raffle ticket systems may be considered gambling and, as such, are not necessarily legal to operate without a license.

2.1.3 – Gap Analysis

This section aims to summarise the gaps I have identified in the platforms and applications that could be considered potential competitors and how Raffl aims to fill said gaps. It serves as a conclusion to the research done in previous sections and to justify the app's business case and prospective target consumers.

Standard e-commerce applications, such as eBay, exist, and there are also raffle hosting platforms out there, but none have tried to bridge the gap. That's where Raffl comes in; as an unconventional hybridisation of both e-commerce and raffles, it offers a new, refreshing approach for users to offload unwanted items at a profit whilst providing an exciting opportunity to win new items.

Moreover, Raffl aims to gamify the practice of obtaining new items you may desire. Like the lottery, this will stimulate the buyer with anticipation of winning, albeit with lower stakes. If the profit made by standard raffle stalls is anything to go by, selling your items with this method may bring in more profit than selling them for a fixed fee or for an auction online, offering a strong incentive for sellers.

While the primary standout feature of Raffl against e-commerce sites are its raffle tickets, it is not the only one. Drawing inspiration from the success of eBay and other sites, Raffl can implement similar features but streamline the approach with a simplified UI that is more in line with modern design expectations. This means that Raffl won't fall for the common pitfall of overwhelming users with too many features or a clunky UI and can fill that gap where many applications, such as eBay, fall short.

In terms of the raffle side of competition, the closest thing to Raffl is the website mentioned earlier, ‘Raffall.’ This, however, falls under more standard Raffle ticket approaches where it has a large prize pool and is intended as competitions for large clients, such as companies with a project to shift. There is no competition in terms of small, hosted listings by individuals, and this is a gap that Raffl capitalises on. Unlike local, community raffles, Raffl provides a wide reach and hosted ‘raffles’ are intended as a tool to offload unwanted items rather than just a means to make a profit for the sellers. This gives it a unique level of utility that current raffle platforms just don’t offer.

In summary, Raffl is not just another e-commerce app but is instead a bridge between the excitement of raffles and the utility of standard e-commerce apps. By combining the best aspects of raffles and e-commerce, Raffl can offer an exciting, streamlined experience in the digital marketplace that has not yet been seen. The app's distinctive approach to blending all these elements together equips it well to address all the unmapped needs previously discussed by simplifying the selling process, enhancing buyer engagement, and potentially increasing the profitability of sellers.

2.2 – Design Considerations

2.2.1 – User Interface (UI) Design

This section aims to cover user interface (UI) design, which is a vital aspect of the design process. This is the way in which the user interacts with the application and how information is presented to said user. UI design can significantly affect the performance and usability of applications, and studies have proven that a good UI design can increase the conversion rate of a website by approximately 200% [6]. Whilst this study was conducted on websites, it is still relevant to mobile development as consistency is integral across all platforms [[7], so by extension, if design is important for websites, that can also be applied to applications.

The usability heuristics discussed below are widely recognised across UI and UX design and provide a strong cornerstone for the design of a mobile application. As such, by applying these principles, I can develop an app that is intuitive and easy to use while meeting the needs of the users. The ten principles laid out by Nielson that I plan to adhere to are below and will be elaborated on in the requirements section.

1. Visibility of system status.
2. Match between system and real world.
3. Use control and freedom.
4. Consistency and standards.
5. Error prevention.
6. Recognition rather than recall.
7. Flexibility and efficiency of use.
8. Aesthetic and minimalist design.
9. Help users recognise, diagnose, and recover from errors.
10. Help and documentation.

Legibility is another important factor to consider when designing the UI. Therefore, font styles will be familiar and readable, avoiding confusion for the user. The font size will also be

suitable for the environment, where it will be visible for most users rather than alienating those with poor eyesight, increasing the accessibility for a broader audience. Ensuring colour compatibility will also improve text legibility, so it should be prioritised.

Good UI design is not only aesthetically pleasing, but users are also more likely to trust an app with a clean design that lacks ads, especially in e-commerce. Ensuring this would also help avoid user errors, such as accidental purchase of raffle tickets. Productivity is another stand-out feature of good UI design, as users can complete their tasks faster when the UI makes it intuitive, leading to greater satisfaction. In the context of Raffl, if users are able to list items with ease, they may be more likely to return and list more for the convenience factor.

An important aspect of UI design is deciding on colours that will be used in the app. Though seemingly inconsequential, research has shown otherwise, with a study from CCICOLOR that claims people make subconscious judgements about products within 90 seconds of initial viewing and that between 62 and 90% of that assessment is solely based on colour [8]. To aid the product in good colour design, I will use the common ’60-30-10’ rule. This is a design guideline commonly used in industry that emphasises key elements in the design, with the dominant colour being used for around 60% of the product, where it’s supported by the secondary colour for 30%. 10% of the colour is then left as the accent colour, with the goal of highlighting specific features or parts of the design.

2.2.2 – User Experience (UX) Design

This section aims to cover User Experience (UX) design. This is the process of aiding user satisfaction by enhancing the usability and accessibility between users and an app. Similarly to UI design, UX design has a huge impact on application performance, with a study by Forrester showing that for every $1 invested in UX, a return of around $100 can be achieved [9]. That’s not the only support for emphasising UX, as a survey from ‘Topal’ claims 90% of smartphone users say they’re more likely to keep shopping if they’re having a great user experience [10]. This is especially relevant for an application such as Raffl, as at its heart, it’s essentially a shopping app.

Accessibility is a huge part of UX design, and by working on this, I can improve utility for all users, including those with certain impairments. This leads to a more inclusive environment, making sure that no one feels left out who would want to use the application, and in turn, provides us with more possible consumers to work with. To ensure that my application adheres to accessibility standards, I will be utilising ideas from ‘Material Design 3,’ an open-source design system made by Google [11]. This includes a range of basic advice, such as ‘target spacing,’ ‘pointer targets’, and other seemingly simple things that one may overlook when creating an application, not understanding the importance of dealing with it.

To ensure accessibility in this application, I will focus on design aspects such as having sufficient contrast within the app. This can make sure that the app not only looks visually appealing but is clear and legible, with colours not blending in together. Failing to do so can harm all users, but specifically those who have visual impairments, such as partial colour blindness. I should also avoid using colour to convey information [12]. For example, I could use red to indicate danger in a certain task, but if I do, then I must ensure that the danger is

clear despite the colour. Otherwise, colourblind users would be at a disadvantage by not understanding this. In this example, I could overcome this by adding a conventional warning message along with the colour, so the danger is clear, regardless of colour.

While these features may seem good on paper, subjective opinions may differ from person to person, and strictly defining a good UX is difficult without putting it into action. Therefore, I will prototype the design to get an idea of how it feels as an end user. This will be done with initial paper prototypes for a basic look before moving on to more in-depth Figma prototypes.

2.3 – Legal and Ethical Considerations

This section covers the ethical considerations that must be considered when designing Raffl, along with the legal implications regarding bringing such an app to market. As it is not being brought to market, this is more of a hypothetical analysis of what would be required.

As the regulation surrounding raffles themselves is quite ambiguous, I will consider the UK Gambling Act [13] in the context of Raffl. Consequences of this act for Raffl would include a necessary age verification system to ensure that users are not under the age of 18 when participating. This could take many different forms; for instance, when signing up, users could be required to have ID verification before being allowed to buy tickets.

Individually hosted raffles that bring in a profit of less than £20,000 would also need to be registered with local authorities, with further complications with licensing if this profit is exceeded [14]. This would be especially frustrating for sellers entering this market, as it would be a huge inconvenience to start using the platform. This has, however, been circumvented by some entities, where instead of being classified as lotteries, they are classified as prize competitions. In the case of Raffall, they switched to this classification, including a random question when buying raffle tickets, to demonstrate the level of skill involved in participation [15].

Other considerations that are necessary to break into a market like this would be ensuring compliance with the Financial Conduct Authority (FCA), for example, in the UK market. The FCA sets out regulations to ensure that financial markets are fair and open and that firms act in the best interests of consumers [16]. The implications for Raffl would be to adopt transparent financial practices on the handling of payments, disclosing how funds are stored and processed from ticket sales. In addition, Raffl may be required to build robust systems to detect and prevent fraud, including (but not limited to) real-time transaction monitoring.

Similarly, the Data Protection Act (DPA) is about protecting the data privacy of consumers, laying down the rules about how organisations store and manage personal data [17]. For Raffl, this means using secure data storage systems, employing data encryption where possible, and carrying out regular audits on all these systems to prevent data breaches. Moreover, to guarantee transparency towards users, Raffl's user interface should comprise straightforward user consent agreements on data being collected and held, as well as potentially allowing users the freedom to manage the data through deletion requests, for example.

If Raffl were to arrive in a production environment, ethics must also be considered to ensure that people are not taken advantage of and that the application is not detrimental to society.

One such way an app like this may harm society is in its addictive potential. If some users get too invested in the thrill of raffles, countermeasures such as spending or time restraints could be incorporated. The goal of this would be to reduce usage for these users and give them time to cool off rather than getting carried away.

Another ethical qualm that many gambling apps fall into is what might be called ‘predatory’ promotions. These could be in the form of free tickets after a certain amount of play, the overlaying of endless pop-up adverts, or seemingly generous welcome deals that reel users in. While these might generate revenue for Raffl, they can be perceived as predatory promotional strategies and can mislead users about the nature of the app.

To conclude, there are several legal and ethical considerations that arise in the development of an e-commerce app, and many regulatory barriers could appear in the future. These would all have to be carefully considered and built upon in a full production release and would likely require experts in the field to ensure that both legal and moral grounds are appropriately handled. Furthermore, it is particularly important that legal compliance and ethical issues are addressed because of the risk of legal penalties, consumer mistrust, and reputational damage. Finally, in addressing these issues successfully, what once were weaknesses can then become strengths and competitive advantages to Raffl, such as building trust with users and differentiating from competitors who are keener to take advantage of advertising revenue and perceptively predatory promotional strategies.

3 - Technology Considerations

3.1 – Database Technology Considerations

When deciding what database technology to use, there are two important decisions to make. Firstly, I will decide to use SQL or NoSQL. After making this decision, I will then choose what platform to develop the database software on.

3.1.1 – SQL vs. NoSQL

The two main database technologies in use today are SQL and NoSQL, respectively. The aim of this section is to detail the differences between SQL and NoSQL before concluding with what would be a better choice in the use case of Raffl.

SQL, aka 'Structured Query Language', is a standard computer language used for querying and manipulating data in relational databases. This means that users can utilise SQL to interact with relational databases, using it to create, view, update and delete data. A relational database is a collection of data stored in any number of tables [18]. The term 'relational' indicates that the tables are related to each other in some manner. In the context of Raffl, there may be a user's table and a listings table. Every listing has a singular host attached to it, which is the user who is hosting said listing. Similarly, every user can host any number of listings. This relationship between tables is established through foreign keys, and in this case, a 'hostID' foreign key in the listings table would reference a 'userID' in the user's table. This relationship is known as a one-to-many relationship, but SQL supports many different kinds.

A common misunderstanding with NoSQL is that the language does not support SQL. This acronym actually means 'not only SQL' and is not a strict rule outlawing the use of SQL; instead, it varies with differing implementations. SQL works very well when data has a clear structure, as each table has fixed columns but an unlimited number of rows. This is not always the best way to store things, which can be seen in an app like Raffl. Each listing may have a name and a description, which SQL would handle well, however some listings may contain more optional data that doesn't necessarily exist in others. For instance, certain listings may have items associated with them, such as tracking numbers or a shipping address. You can still work with this data in SQL. You may want to create an entirely new table to reference tracking numbers, but this means you'll need to do another query, which will increase the time needed to retrieve data. You could also add a tracking number column and leave it blank for listings without one, but this would waste storage. This is a drawback that you don't have to face with the flexibility of NoSQL.

In industry, a combination of these systems would often work best. There are many different tasks that we want to perform with our data, and no 'one size fits all' solution, so using both SQL and NoSQL in different areas would be optimal. Given that Raffl is a relatively short project with a limited number of resources, I cannot justify developing both SQL and NoSQL solutions. As shown with the structure examples earlier, Raffl is likely to have some data that works well for SQL and some data that works better for NoSQL. As Raffl is being developed in a short timeframe and will not initially have all the features it is expected to have on launch, I have determined that writing a structured and well-defined schema is not a good

idea. In possible future iterations, this could be difficult to work with and may need data to be restructured entirely. This is a problem I will not face if I use a NoSQL database for Raffl.

To conclude, I have decided to use NoSQL for the development of Raffl, but that is not to say SQL would have been a bad approach. Both models offer different advantages, and both could be used for Raffl.

3.1.2 – Firebase vs. MongoDB

In the last section, I decided that NoSQL will be utilised over SQL, but there are still plenty of NoSQL solutions that could be used for Raffl. In this section, I will evaluate two possible solutions, 'MongoDB' and 'Firebase Firestore', along with a justification for which one I will use and why.

Firebase offers two database services with 'Cloud Firestore' and 'Realtime Database.' If I go with Firebase, I will be using Firestore for numerous reasons, such as extremely high uptime performance along with transactions that span the entire project rather than specific data subtrees [19]. The main advantage of Firebase is the suite of tools offered by Google that can be directly used with Firebase. This includes Googles authentication service, and Google cloud functions. That's not to say these features are unavailable on MongoDB, but the lack of direct implementation makes them harder to work with and less convenient, adding to the development time of an app that is already limited in this field.

When it comes to MongoDB, a big advantage over Firestore is the supported complex querying. Situations that can be solved in a query by MongoDB may require application-side code, such as filtering in Raffl. This would be bad not only for performance but also for cost as we would be requesting more data from our database. Another useful feature offered by MongoDB is its full-text searching. This enables more accurate searching, which is not possible with Firestore alone.

Overall, both services offer tempting features that Raffl could benefit from, but only one can be selected. I have decided that despite the benefits offered by MongoDB, the direct implementation of Google services, such as their authentication and cloud functions, make Firestore the more appropriate choice for Raffl. The biggest drawback of Firestore is the lack of full-text search functionality, but if I have time during the development of Raffl, I can get around this using a service such as Algolia in tandem with Firestore to make up for it.

3.2 – Platform and Technology Considerations

One of the first things to consider when developing a product is the platforms on which it will be available. Every platform will require a different approach to creating the application, from the design to the actual implementation. It is, therefore, important to consider what is most suitable for Raffl in terms of both developments and end-user experience. In this section, I will discuss why I have opted for cross-platform mobile development, along with comparing alternative approaches. It will also cover the technology I have decided to use for this project with relevant justifications.

3.2.1 – Web vs. Mobile Applications

In the UK, 97.8% of individuals use the internet as of 2023 [20]. As most internet devices have access to websites, this implies that a web-based application can reach a large, diverse user base that mobile-only applications cannot offer. This can simplify the process of development, as developing for the web will cover most devices.

Despite the web having a significant potential user base, this doesn't necessarily translate into actual users. According to Criteo, users view 4.2x more products per session on mobile apps vs mobile sites [21]. For an e-commerce app like Raffl, this implies that mobile apps will provide an increase in user exposure and, thus, engagement with the products listed. As Statcounter indicates, mobile devices have a higher internet share, at 59.01%, compared to desktops, which have just 40.99% [22]. I will prioritise a mobile user base. Therefore, to engage these users more effectively, it would make more sense to offer mobile applications instead of relying on web accessibility.

In an ideal world without time and resource constraints, I would develop both mobile and web applications separately; however, this isn't feasible for a product of this timescale. Therefore, as the information reviewed previously indicates clear advantages for mobile applications, I will focus on mobile app development with Raffl.

3.2.2 – Android vs iOS Development

One factor we should consider when deciding which platform we will deliver to on mobile devices is market share. Android is slightly less popular at 41.46% than Apple at 58.1% [23]. Whilst this statistic favours Apple, the gap between platforms is not very large, and excluding Android would still result in us losing almost half the user base. With fewer Android users, there will be fewer potential ticket holders, which would result in lower revenue for products. If products make less, then iOS users are also discouraged from listing them, and this could damage the app, so if possible, we would not like to exclude either platform.

Another point in favour of iOS is that there is far less hardware heterogeneity between devices. When developing for Android, we must consider aspect ratio differences, whereas, with iOS, all devices in the last few years have been released with an aspect ratio of 19.5:9. This means that when only developing for Apple, we can make a nice, consistent design without fear that a layout change will break it. If we develop for Android, however, we will need to consider compatibility across a huge number of devices and, as such, a UI that can keep up with this.

Security is another important factor when developing across platforms. Android is open-source and highly modifiable, which leads to many benefits, but unfortunately, it also opens the device up to more potential vulnerabilities. Apple have stricter management of their devices,[24] and as such, are much harder to crack. While disadvantageous to Android, it would still be good practice to implement good security within iOS, so it should not change anything in the grand scheme of things.

Ever since Windows 11, now android apps can be downloaded and ran on Windows devices. This is a big advantage for Raff, especially since we have decided to focus on mobile app development as opposed to web development, so Windows users will still have the option to run apps if they need to. While not many users will care about using apps on Windows, this doesn't require any extra effort from a development perspective and only provides benefits, so it is worth consideration.

From a development perspective, the biggest drawback to iOS is Apple's closed ecosystem. Developing and testing applications for Androids is very easy on both Windows and MacOS, though the same cannot be said for iOS devices. This is because Apple's firm grip on its ecosystem does not allow for iOS virtualisation or allow Xcode to run on Windows devices, unlike Android development, which is easy on both Windows and MacOS. This can technically be done with major workarounds but is difficult and does not provide the same experience as development on a MAC. As the development environment is a Windows device, and my primary phone is an Android, iOS development would cause more trouble.

The limitations outlined above are significant in terms of development for a single mobile platform, therefore I have opted for a cross-platform development approach. This means I will have to face the downsides of both systems, such as weaker security on Android and difficulty testing on iOS. This application isn't performing hardware-specific tasks, therefore the main difficulty will be an increased development learning curve, which can be justified by improved accessibility. I will develop for Android first, as it is the easiest with the tools available to me; however, using a suitable cross-platform development environment means this shouldn't impede the iOS user experience much. A cross-platform approach also means I won't have to allocate development time to 2 different applications, giving me more time and resources to refine a singular product.

3.2.3 – Choice of Development Framework

Now that I have decided on cross-platform development, an important decision will be what development framework to use. There are currently two dominant frameworks in the industry, 'Flutter' and 'React Native'. Here, I will provide a comparison of these frameworks, evaluating their strengths and weaknesses, along with their suitability for Raffl. By the end of the section, there will be a decision along with a justification for our chosen framework.

Flutter is an open-source framework from Google for cross-platform app development. It is relatively new[3], and while such systems remain standard, adopting a similar structure in my app will be advantageous.

Flutter is an open-source framework from Google for cross-platform app development. It has a widget-based approach, wherein widgets are used to describe the structure and layout of an application, acting as building blocks. In the context of Raffl, we would use a combination of

widgets to build our UI elements, i.e. a custom widget could contain our listing details, such as a separate image and text widgets for the user to view. Using Flutter will also require learning ‘Dart,’ a lesser-known programming language by Google.

Alternatively, I could use React Native, which is Facebook's own open-source framework for cross-platform app development. It uses a component-based approach, which, while much like Flutter's widget-based approach, has a few differences. One such difference is the handling of styling in our app, where React Native utilises CSS for our styles as opposed to Flutter, which utilises Dart for everything, including styling. React Native also uses JavaScript, a popular scripting language for development, meaning there is less to learn and will likely be more resources, such as libraries and a bigger community, to assist online.

Both Flutter and React Native share many useful features, such as ‘hot reload’, which is a big selling point compared to native Android Java development, which does not offer this feature. Hot reloading is where you can instantly apply changes while an application is running, and unlike instant reload, which aims to do the same thing, hot reload is faster and saves the current state, meaning the app isn’t restarted. As they are also backed by tech giants such as Google and Facebook, they can both be relied on for continuous updates and plenty of resources.

As for differences between the languages, a big selling point for Flutter, in terms of this project, is Firebase support. If I decide to use Firebase authentication and database services for Raffl, which is likely, then its ease of integration and performance when matched with Flutter is unparalleled. This is because both are Google products, so Flutter utilises plugins as opposed to third-party libraries required by React Native, resulting in better performance and smooth development [19]. We can also see that Flutter has rapidly been increasing in user adoption to the point where it has overtaken React Native in popularity [20]. This indicates a shift in industry trends and a sign of trust from users.

In conclusion, whilst both frameworks have their own pros and cons, they share many strengths, and either would be a good choice for Raffl. In some ways, I think React Native may be a better choice for Raffl, primarily due to my experience with JavaScript and CSS. However, overall, I think that Flutter aligns better with the needs of Raffl due to the Firebase support offered and the rapidly growing userbase, and therefore it will be the chosen framework for this project.

4 – Requirements

This section provides a comprehensive set of requirements for Raffl, the app I am in the process of creating. The aim is to understand the entire scope of the application whilst also breaking it down into requirements, which should give a sense of scale which will assist with project planning. I also detail why each requirement was made, along with the process taken to create them.

In future sections, I will cover the timeline of this project, where the recommendations created here were essential. This is because I am using an agile methodology, and these recommendations assisted me in breaking big tasks down into smaller chunks. I was then able to utilise this breakdown to create a plan for future sprints, and whilst my initial plan wasn't entirely accurate, it gave me a good idea of how much I should be accomplishing each sprint.

4.1 – Requirements Gathering

To identify potential requirements for Raffl, I first set out to understand how user interactions with the system would work. I did this by modelling a high-level, abstract use-case diagram with a tool known as Creately [27].

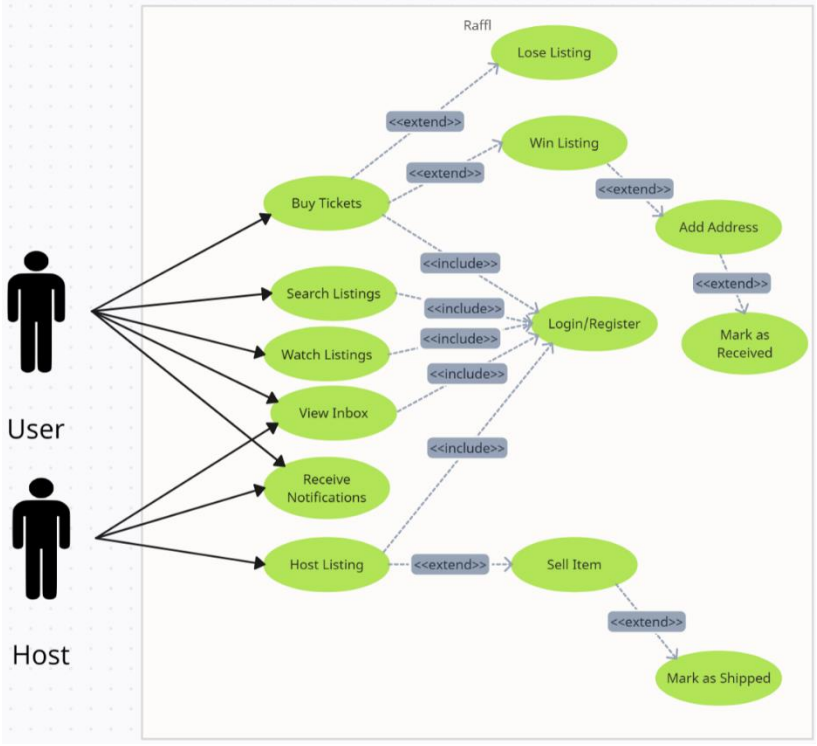


Figure 17 - Use-Case Diagram

To create the use-case diagram seen in Figure 17, I first thought about the possible interactions users may have with a system. Despite them having the same privileges, for simplicity, I separated standard users from hosts to better identify the needs that may differ for host users. I went for a relatively high-level look at all the features so I could search through these requirements and break them down into smaller, more digestible chunks when creating the requirements.

When drafting the initial requirements, I made sure all the features seen in this use case diagram were accounted for and possible within the scope of the requirements definition. This guideline ensured that I didn't forget any important details that could come back to haunt me later when it would be more difficult to adapt my product.

4.2 – Requirements Definition

When developing my requirements, one thing I made sure to cover were the ten usability heuristics that Neilson laid out [6]. I spoke about these in my literature review section, so I will not delve any deeper into them here, but below is a number list of said heuristics, which I refer to in many of my recommendations:

1. Visibility of system status.
2. Match between system and real world.
3. Use control and freedom.
4. Consistency and standards.
5. Error prevention.
6. Recognition rather than recall.
7. Flexibility and efficiency of use.
8. Aesthetic and minimalist design.
9. Help users recognise, diagnose, and recover from errors.
10. Help and documentation.

To craft my requirements, I am using the 'MoSCoW' technique [21]. This method for defining requirements was created by Dai Clegg and is a way of prioritising requirements for projects. This method has the following four categories:

- Must - Requirements integral and non-negotiable for project completion.
- Should – Requirements that would benefit the product and should be incorporated if possible.
- Could – Requirements that are nice but not that important and could be implemented if resources permit it.
- Would – Requirements that the project may benefit from but fall out of the current scope and may be useful in future versions.

To structure this, I have four subsections that cover all categories of MoSCoW'. Each section contains a set of requirements, along with an ID, a title and a description/rationale. Sections about 'must', 'should' and 'could' also contain a validation. This is because, unlike requirements in the 'would' section, many of these requirements will be included in our project, and we want to have a way to test whether this is valid. The structure of each requirement is shown below.

ID	**Title**
Description/Rationale – A description & justification of our requirement	
Validation – How we can test our requirement	

4.2.1 – Essential Requirements

This section details all the requirements that are essential to the completion of this project. They are the requirements that fall under the 'must' category in 'MoSCoW' and should all be implemented.

#1	Users can register with a unique email address
E-mails are less personal than phone numbers as you can easily create multiple. Furthermore, as no two emails are the same, this is a good way to uniquely identify an account, so it is a useful tool for identifying unique users. Email addresses must be unique, as we don't want two accounts with the same e-mail address.	
Register an account with an e-mail, which should succeed. Next, try to register an account with the same e-mail address, which should fail.	

#2	Users can sign into existing accounts with email
Users should be able to sign-in to previously created accounts with their email so they can access the application.	
Attempt to sign in from the sign-in page using an email address.	

#3	App works on Android and IOS
As mentioned in the literature review, both IOS and Android are popular, so to avoid losing a large chunk of the market, targeting both is imperative.	
An ideal testing solution would be to verify that functionality works on both Android and IOS, but as this will not only approximately double the testing time, I do not have the resources available to test it on an IOS device. Apps built on Flutter should automatically run on IOS, given there is no platform-specific code, so checking there is no platform-specific code to break it is the current workaround validation.	

#4	Users can differentiate between items of interest (wins, selling, etc…)
Separation of different items of interest is common (as seen in the platform analysis) and logical for users and assists user intuition.	
Check that we can view our wins, selling, and items we search for on different pages/sections of the app.	

#5	Password is secure
Password security is important to avoid compromising user accounts to hackers. By enforcing password security standards, we can attempt to prevent common hacking attempts such as brute-forcing passwords.	
Ensure that a password is eight characters long and has one of each of the following: lowercase character, uppercase character, and a special character, by trying to input invalid passwords. If passwords that fulfil this succeed, and passwords that don't fulfil this fail, this is successful.	

#6	Email is correct format
To avoid wasting resources, we should validate that an email address is in the correct format. If a user accidentally misses a character like @, this will also notify them and help them fix their mistake.	
Enter emails of invalid formats and make sure we get an error message.	

#7	Users have unique UID

Whilst emails are a unique identifier of user accounts, in future production builds, it may be possible to allow users to change their emails if they lose access. To avoid future problems with this, each user should have their own unique UID rather than relying on the unique property of an email address.
Check user accounts in the backend have unique IDs.

#8	Users can list own items
Users listing their own items is essential for an app like Raffl to function, as sellers are a necessary part of an e-commerce platform.	
Attempt to create an item listing with a user account.	

#9	Users can buy raffle tickets for items they do not own
Buying raffle tickets is core to the user experience and is the main selling point of our app, which all other features are built around, so this is essential. I do not want sellers to be able to buy raffle tickets, as this defeats the purpose and is open to abuse, so this should also be stopped.	
Attempt to buy a ticket for an item with a user account that does not own said item; this should succeed. Next, attempt to buy a ticket for an item with a user account that owns said item. This should fail.	

#10	Users receive currency for sold items
For users to sell their own items, they need compensation, and this is in the form of directly receiving all revenue for their items sold. Whilst in a production environment, tax, app commission, and other variables may affect this, this is disregarded when building our application.	
Ensure that after a user marks an item listed received, the seller's account is updated with a proportional amount of currency to the amount bought.	

#11	Users have an inbox with status updates
Popular apps such as eBay, Shpock, FB Marketplace and others all have some form of an inbox system. This is, therefore, not just a useful feature for users but should also help with intuition; if a user is looking for a status update, they are likely to look for an inbox page.	
Check an inbox page is available with useful status updates, such as ‘tickets bought for x credits’ or ‘you have won x item’.	

#12	Users get push notifications for more important updates
Another important feature in many mobile applications is push notifications. By incorporating these, we can let users know important updates immediately, such as their items selling for credits or winning an item.	
When important changes occur in the app, such as winning an item, or an item you have listed selling, check for push notifications saying as much.	

#13	Items should be clearly differentiated between own and others

When navigating to an item, there must be a clear differentiation between the items we are selling and other users' items. This is because we do not want confusion among users when viewing their own listings compared to similar listings if they happen to view another one.
Check listings that the current user is currently selling and compare them to listings other users are selling. There should be a noticeable difference between both pages.

#14	Winners of items can add address to listing
This is so the seller knows where to ship the item and should also be restricted to the user who has won said item, as other users don't have any right to the item. We also shouldn't allow users to add an address after they have already added their original address, as the seller may have paid for postage to their address already but have not yet marked it as shipped.	
After winning an item, check that other user accounts cannot submit the address. After this, check that the winner can submit the address. Finally, check that the winner cannot resubmit the address.	

#15	Sellers can mark items as shipped
When a seller has shipped an item, they should be able to mark this as shipped so the user knows that their item is on its way and isn't left wondering. This should only be possible after a user has submitted their address, and only sellers should have this ability.	
Check that a seller is unable to mark an item as shipped when a user address has not been received. Next, when the item has received an address, check that other users cannot mark the item as shipped. Finally, check the seller can mark it as shipped.	

#16	Winners can confirm an item has been received
After an item is marked as shipped, we do not know if the user has received it until we get feedback from them. This feedback is required to verify the transaction and give the credits to the seller. This should only be possible from the winning user.	
Check that a winner is unable to mark an item as received when an item has not been marked as shipped. Next, when the item has been marked as shipped, check that other users cannot mark the item as received. Finally, check the winner can mark it as received.	

#17	Users can add an image for items they are selling
Users should be able to see an image of an item they want to buy, as this both builds trust and avoids ambiguity. In a production environment, adding multiple images to a listing is standard practice; however, to stay within the free tiers of software used for this product, I am limiting this to singular items.	
When creating a listing for an item, ensure that we can submit an image during creation. Next, view the same listing and ensure that the image used is there.	

#18	Users can search for items
To find items users want to buy, the most reliable and common method in industry is searching, so we should allow users to search for listings they may want to buy.	
Try to search for an item and ensure that relevant results appear.	

4.2.2 – Desirable Requirements

This section details all the requirements that are heavily recommended for the completion of this project. They are the requirements that fall under the 'should' category in 'MoSCoW', and we want to implement most of these and, if possible, all of them.

#19	Users can filter items by various properties
When searching for items in a big project, there may be many that fall outside the range that a user is interested in; for instance, they may not be willing to spend much money. To help this, a filter system should be available.	
After searching for an item, check that we can filter on various properties, and the search results reflect our selection.	

#20	Users can sort items by various properties
Much like filtering items, some users may want items that fulfil various properties, such as listings ending soon. Adding a sort for these properties will allow users to access items they are interested in more easily and improve the convenience of Raffl.	
After searching for an item, check that we can sort on various properties, and the search results reflect our selection.	

#21	Relevant users can see status of items that have been sold
By seeing the status of items that have been sold (i.e. awaiting shipping number, currently shipping, awaiting address, etc.), users will have feedback and know what is going on with items that are relevant to them (i.e. items they won or are selling)	
For users who have won items and sellers of said items, check that we can see multiple different statuses across the won -> received lifetime.	

#22	The app is accessible where possible
To follow government regulations and ensure fairness to all people as much as possible, accessibility is an important requirement for the app to follow. This means the app should be easy to follow, with text that is readable.	
As this is subjective, I will ask users if they think it is accessible based on various criteria, such as text readability and understandability, while also making sure they don't think anything would be unclear to people with colour blindness.	

#23	Dangerous actions are obvious
Dangerous actions in the app should be obvious to help the user know what they are doing. For instance, we don't want to accidentally buy a ticket from a missed click, but we want this to be an obvious choice by the user to avoid mistakes. This also adheres to Neilson's 3rd usage heuristic (the system should be clear and allow for total control with the user), as the user is in total control and knows exactly what they are doing. This also aligns with Neilson's 5th usage ` (error prevention)	
Ensure that when buying tickets, it is clear what you are doing and impossible from a single miss-click.	

#24	Design is minimalistic

As seen in the 8th usability heuristic and discussed in our e-commerce platform analysis, minimalism is important for the design of our app.
As this is more of a subjective opinion, it will be handled during the interview, during which candidates will be asked if they think the design is minimal or not.

#25	Users can set ticket price
By giving users the ability to set their ticket price, they have more flexibility and control over their own products.	
Try and set the price of tickets to different values. Ensure that when buying said tickets, the correct number of credits are deducted from buyers.	

#26	Design is consistent
By instilling consistent design throughout the app, we can ensure a sense of familiarity for users whilst also making the app more satisfying to use. This should help user engagement and ensure we are adhering to the 4th usage heuristic that Neilson laid out (consistency and standards).	
As this view is also subjective, I will ask users their thoughts on the design and if they believe it is consistent.	

#27	Live countdown for listings ending
To ensure the 1st of Neilson's usage heuristics (visibility of system status), we can give users appropriate feedback in the form of countdowns for listings that will help them make informed decisions. For instance, if a user knows they only have 15 seconds left on a listing they want to buy tickets for, they must hurry. If this information wasn't live, users may miss out on important information, and sellers may make less profit, which, in turn, decreases total user satisfaction overall.	
Ensure that pages such as the listing page and search page have a timer that ticks down as time goes on.	

#28	App is intuitive
The app should be intuitive as it is crucial to keeping users engaged, especially in terms of first impressions [22]**, which are** heavily influenced by a user's intuition. This also helps with Neilson's 2nd usability heuristic (match between system and real world), as if we incorporate intuitive features, such as icons that are immediately recognisable due to real-world implementation (e.g. a house icon for our home page), then users will immediately understand what that button does. It also aligns with Neilson's 6th usability heuristic (recognition rather than recall), as users quickly recognise what to do rather than having to recall it from memory.	
Ensure that in-app icons are like real-world equivalents. As much of this is subjective, ask participants if they think the app is intuitive.	

#29	Navbar is interactive
The navbar should respond to user clicks for responsiveness, as discussed in user design. For example, the navbar should highlight the currently selected tab.	
Check that the navbar responds to user inputs. For instance, make sure the home icon is highlighted when we are on the home page.	

#30	App has user-specific recommendations
By providing useful recommendations, users may discover items they otherwise would have missed, giving users more opportunities to use Raffl as intended and giving sellers more potential profit.	
Ensure that recommendations correlate to user interests somewhat, based on the user's activity.	

#31	Users can reset their password if forgotten
If a user forgets their password, they should be able to reset it to get back into their account. This ensures that a user doesn't lose all their account history and possibly access to essential data just because they have forgotten their password.	
Attempt to reset an account's password from the login page. Check that an email is received to reset the password of said account, and use the link in the email to reset the account password. Finally, check that the new password works.	

#32	Application is easy and efficient to navigate
The application should be easy to navigate and be efficient in doing so. For instance, it shouldn't take ages to get to where we want to go and having shortcuts (i.e. a back button and a navbar) would help users save time. It would also align with Neilson's 7th usability heuristic.	
As this is subjective, I will leave this to the opinion of users who test the application during the interview.	

#33	Mistakes are clear for users
This is not the same as making dangerous actions obvious. If a user makes a non-dangerous mistake, such as not conforming to password requirement standards, this should be obvious to them. This can be done with colours such as red but must also be clear in other ways so colourblind people are not confused, such as an extra error message. This should be clear as to what the user is doing wrong so they can rectify their mistake and adheres to Neilson's 9th usability heuristic (help users recognise, diagnose and recover from errors).	
As this is subjective, I will leave this to the opinion of users who test the application during the interview.	

4.2.3 – Possible Requirements

This section details all the requirements that would be beneficial to our project's competition but don't have that great an impact based on their perceived value (time to implement based on the benefit they give us).

#34	Users can see analytics of listings
Getting insight into item analytics, such as views or other tickets purchased, helps Raffl's users in 2 ways. Firstly, it gives buyers a bigger picture and helps them decide if they want to buy tickets for said item. It also gives sellers information about how well their items are doing so they can use that information for future items.	
Check the listing page of our items and see if there is any user insight. Also, check the search results to see if we can get any insight here.	

#35	Users can add tracking information to listings
If users can add tracking information to their listings, users will be able to track said items via postal services, gaining knowledge on when their items are coming, giving winners the opportunity to prepare for it (i.e. if they see an item is arriving on Wednesday, they will know to be home to answer the door).	
As a seller, when marking an item as shipped, check to see if we can add postal details to said item. When viewing an item marked as shipped by a buyer, ensure that we can see these details.	

#36	Users can view sold items
If users can view sold items, they can use said knowledge for insight into future decisions, such as knowing what items to sell and how many raffle tickets tend to sell for certain items.	
Search for items on the home page, and check if there is an option to view sold items. If this item exists, make sure it works.	

#37	Search functionality is broad and doesn't rely on exact inputs
In many basic text searches, you need to directly match the name of a product in order to find said item, but this rigid approach is often inconvenient in real-world scenarios. By having flexible search functionality, users are more likely to discover items they are interested in rather than just direct matches. It also means if a user makes a typo in their search, they will still find relevant products.	
Try to search for items that exist in the app but under slightly different names (i.e. with typos), and ensure that they still appear.	

#38	Users can search for items by respective tags
Whilst item tags are hidden on the back end, allowing our users to search for items using these will enable users who are interested to find categories of items they are interested in.	
When searching for an item, try to search using broad tags such as 'tech' and ensure tech-related items without that name appear.	

#39	Help page is available in app
To adhere to the 10th usability heuristic laid out by Neilson, a help page could be incorporated into our app. This would ensure users who don't fully understand the app are able to learn to use it, avoiding frustration causing guesswork.	
Check if there is a help page within our app, and if so, check that it details how to use the application fully.	

#40	Users can watch items
If users can watch items they are interested in, they will be able to go back to said item with ease and interact with it (e.g. if they decide to buy more tickets closer to the end date of said listing). This feature would also require users to have some method to filter on watched items, such as another page or under our filters.	

Check if there is an option to watch an item on listing pages. If it is possible to watch an item, ensure that we can view watched items independently of others.

#41	Users require email confirmation
If we force users to confirm their emails, it reduces the ability for malicious users to create fake spam accounts and stops users from accidentally creating accounts with a typo in their email.	
Attempt to create an account with a unique email, and instead of logging us in, the app notifies us that we must verify our email. Next, try to log into our account without verifying the email; this should fail. We can then check if we have received an email verification link, and if so, verify it through the link. Finally, we should be successful when trying to log into our email account.	

#42	Users can directly purchase items
In eBay, users are not forced to auction off their items, and instead can choose to outright sell them for money. If time permits, this may be a useful feature; however, it should be considered more deeply, as it may just overcomplicate the app.	
Check if users can list an item without raffling it off. If possible, check that users can buy said item directly.	

#43	Recommendation system is driven by machine learning
In many popular applications, such as YouTube, recommendations are driven by machine learning. This means that, in time, the system learns what individual users desire and can offer them more reliable recommendations. In a small test app such as Raffl, it would be hard to optimise such a system without many users or training data, but it may be worth implementing more as a proof of concept rather than a useful feature.	
This is a hard requirement to test and would require analysis of the backend and knowledge of how recommendations work in the app. To confirm this, evidence of such a system would be required.	

#44	Users can log in with phone number
All accounts will have an e-mail linked to them, but if we give users an optional option to log in with their phone number, this allows for more flexibility in user sign-in and gives the user more choices. It also helps if a user forgets what e-mail they signed up with, as chances are they only have one phone number.	
Create a user account with a phone number if possible. Login to user account with said phone number instead of email. If successful, this is verified.	

4.2.4 – Future Requirements

This section details all the requirements that fall under the ‘would’ category. Most of these requirements would be quite beneficial to this application, but under the resources available, they have fallen just outside the scope of Raffl. In future iterations, these features could be implemented.

#44	Postal tracking is available within app
In conjunction with requirement #32, implementing postal tracking within Raffl would be beneficial to users, as instead of having to go to a postal service website, they could directly witness this within their application. While due to time restrictions, this is not present in Raffl, it would be useful to incorporate it in a possible future to compete with apps such as eBay, which have similar systems.	

#45	Optional website alongside mobile application
Raffl could also have an optional website to reach a wider audience. This would not only open Raffl up to laptop and desktop users but also give mobile users a second option if they desire it. As this would require a fair amount of extra work and is not possible within the current development timeframe, I am not doing this. Despite this, Flutter has the option for website deployment, but this feature isn't perfect yet. In future builds of Flutter, if this feature is improved, it may be hassle-free to offer Raffl as a website option as well.	

#46	Winners can leave reviews for users they have received items from
To instil trust in users and match other popular applications available, the option to leave reviews for the experience a user has had dealing with another individual would be useful. It would help build trust for reputable sellers whilst also weeding out bad sellers. This is not being incorporated due to time restraints.	

#47	Users can get refunds for unfair services
If a user is scammed on many websites, such as eBay, there is a big support network to ensure everyone is treated as fairly as possible. In an application like Raffl, without a proper support network, implementing such a feature without major downsides (i.e. winner abuse) would be impossible, so it is not being considered for this application. In a possible future build of Raffl, with a functional team behind it, this could be a valuable addition.	

#48	Users can message sellers
By implementing messages to other sellers and taking a page out of eBay and Facebook Marketplaces books, users would be able to get clarification about items for which they want to purchase raffle tickets. This would give them a more informed decision and should lead to user satisfaction on both ends. This is not being incorporated due to time restraints.	

#49	Users can message sellers
By implementing messages to other sellers and taking a page out of eBay and Facebook Marketplaces books, users would be able to get clarification about items for which they want to purchase raffle tickets. This would give them a more informed decision and should lead to user satisfaction on both ends. This is not being incorporated due to time restraints.	

#50	Two-factor authentication protects user accounts
If an optional mobile or email 2fa was implemented into Raffl, this would provide users concerned with extra security reassurance whilst protecting user data better. This is another feature that would be useful to implement if time allows, and as such, could be possible in future updates to Raffl.	

5 – Design

This section details the design process for Raffl, including both UI design and database design. These designs were built from the requirements section to make sure the project meets its specifications. The UI design sections are sequential, starting with basic wireframes before evolving into more complex Figma designs. I also include my rationale for design choices in the app with colour selection among other things. Finally, I end with the database design section, which covers the structure of my NoSQL database within Raffl.

5.1 – Paper Wireframe Prototypes

The first step in my design process is using paper wireframe prototypes for an initial look at my design, which is what this section covers. I got inspiration from this after studying Human-Computer Interactions (HCIs) which put emphasis on the prototypes. Paper wireframes are useful, as they are relatively fast to make and can help identify any glaring issues with designs.

Many pages in my app are likely to follow a similar design. Below are 4 of the primary pages I expect to include in my application:

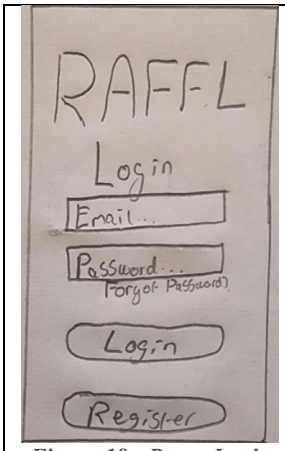


Figure 18 - Paper Login

The first page is the Login page, and is relatively simplistic, perhaps taking inspiration from industry standards. However, with a seemingly common design, users should instantly understand how this app works and intuitively be able to use it, supporting requirement #28.

The 'RAFFL' banner at the top is unlikely to stay in future designs and is more of a placeholder for a logo. As is standard practice, users can enter their email and password to log in, along with the option of switching to a register page if they do not have an account.

NB: I have not yet made a register page for my application; I will experiment with this in my Figma design prototypes.

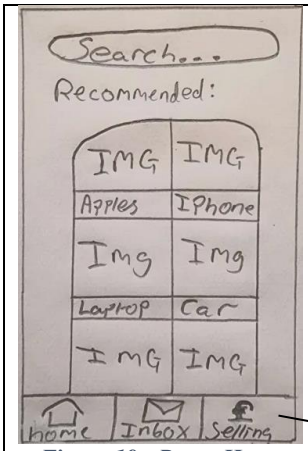


Figure 19 - Paper Home

The home screen took inspiration from popular e-commerce platforms, such as Amazon and eBay, where recommendations are on the home feed, immediately drawing attention to products more aligned to user preferences. In addition, by offering a Navbar at the bottom, users can see recommended items based on their activity, along with a search bar at the top to further browse through the catalogue of items sold on Raffl.

There is also a NavBar that is on all future wireframes and gives users easy access to all pages the user may want to navigate. It uses labels such as 'home' and an easily recognisable icon to create user-friendly navigation.

The 'selling' page can be accessed using the home button on the bottom navigation bar.

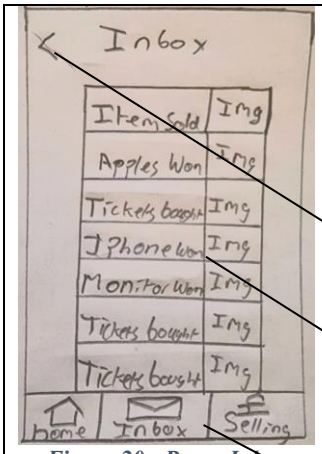


Figure 20 - Paper Inbox

The inbox screen contains all status updates from the app, including items that have been sold, tickets purchased, raffles won, and items shipped. In the final implementation, I plan to have all push notifications displayed here (such as Item Sold), but not all items in here are necessarily push notifications.

This page includes a back button, following the convention of returning to the previous page.

Notifications are interactive, and clicking on a notification should send us to the listing page related to our item. This is to offer a shortcut to the user and uphold requirement #33 (the app should be efficient and easy to navigate.

This page can be accessed using the inbox button on the navbar.

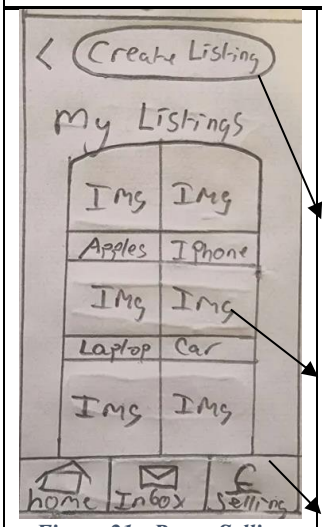


Figure 21 - Paper Selling

The 'selling' page follows the themes present on pages like the home screen. This aligns with requirement #26, which suggests that the design should be consistent across pages. The primary difference is that it shows listings that this user is currently hosting rather than listings that the user may be interested in.

The other difference here is the 'create listing' button, which brings us to a separate page for hosting a listing of an item. This page has not been implemented and will be investigated further in our Figma design.

Clicking this image, or the car text below it, will bring us to the item listing page. Another page that is not in the paper wireframes but will be investigated further in the Figma prototypes.

As before, this entire page can be accessed using the selling button on the navigation bar.

5.2 – Visual Design Considerations

Now that the wireframe prototypes have been made, I have a good idea of how I want my Figma prototype to look. The wireframes are useful as a starting point for the structure of the design; however, they do not capture all possible considerations necessary when designing said application. These considerations, such as colour schemes and logo designs are what this section aims to cover.

5.2.1 – Colour Scheme

This section aims to tackle the colour scheme choices used in Raffl. It underlines how I decided on what colour to use, along with the proportion of that colour that will be used within our application.

In the literature review of this report, I stated the importance of choosing a good colour scheme and how much a user's perception of an app can be altered by the colours involved in each application. I first needed to select the primary, secondary, and tertiary colours for my mobile application. When creating an app with minimalism in mind, I knew I didn't want any flashy, over-the-top colours, so I started with a primary colour of plain white. When choosing

a secondary colour, I wanted another similarly basic colour and landed on grey. I decided on grey as I wanted Raffl to have a reputation for sophistication [23].

After landing on my primary and secondary colours, I decided to use ‘Coolors’, the colour pallet generator, to create a colour pallet that I liked for Raffl. I did this by inputting the colour white and locking it before randomising the colour pallet until I got one that I liked with some form of grey as its secondary colour. After multiple attempts, I landed on this colour pallet that I really liked:

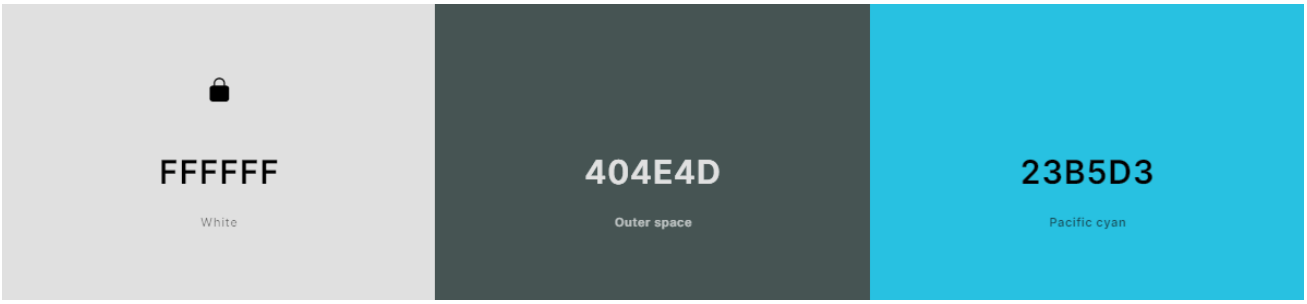


Figure 22 - Coolors colour selection

As per the generated colour pallet above, I will try and perform the following colour distribution across my app:

- Primary Colour: White (FFFFFF) – 60%
- Secondary Colour: Grey (404E4D) – 30%
- Tertiary Colour: Cyan (23B5D3) – 10%

It is worth noting that this distribution is more of a suggestion than a rule, as I will not sacrifice visual elements to increase/decrease the distribution of colours.

5.2.2 – Logo and App Icon

This section covers the choice behind my logo and app icon design for Raffl. It covers the process I used to create them, along with the final product.

As I am not all too familiar with graphic design, and this is a solo product, I was unsure how to proceed with the development of my Logo/Icon. After surveying the web for open-source logos that could match the theme of my logo, I had little success. I then had the idea to use newly developed Generative AI tools, which have seen increasingly large traction, leading me to the ‘Bing Image Generator’ [24].

I started with the prompt: “create me a logo for an app called 'Raffl'. It is a raffle ticket-themed auction app. As such, I want the logo to include a raffle ticket, and I want it to be relatively minimalistic without too much detail. I also want you to use a grey colour in the logo with RGB code 404E4D.” While not necessarily bad, these logos had noticeable errors, were quite complex, and were in a difficult format to extract as a logo.

Figure 23 - Logo Batch #1

Figure 24 - Logo Batch 2

After subsequent attempts, I came up with a prompt that got me a logo I liked: ‘Make me a logo. It is a raffle ticket, and it has the name 'RAFFL' in it, so ‘raffle’ is capitalised but without the E. I want it to be quite minimalistic and have a grey colour scheme (with the colour code 404E4D)’. Here, I was clearer about the logo name and tried to be more specific about my design requirements. Consequently, I think both the bottom images were suitable logos for Raffl, and I ended up choosing the bottom left logo for its minimalism.

After some Photoshop adjustments to align the logo and remove the background, the resulting logo can be seen in Figure 25. I felt this captured the style that I am going for in Raffl whilst also pulling off the simpler and cleaner look that Bing Chat had failed to materialise in earlier tests.

Figure 25 - Final Logo

With a similar process, I moved on to creating an app icon. This also took multiple attempts, until I fell upon a prompt producing the 4 icons in Figure 26:uas it wasn’t putting a raffle ticket in the background despite specifying one. ‘Design me an android app icon, that is a big grey raffle ticket with the letter R inside. The background should be white, and the letter R should be white. Make it minimalistic, and don’t forget the raffle ticket.’

Figure 26 - Icon Batch

Figure 27 - Final Icon

This was the final icon design, and unlike many AI-generated images, it seemed to be relatively accurate to the prompt, with no noticeable mistakes.

5.3– Figma Prototyping

After taking the HCIs module at Loughborough University, I gained insight into how valuable Figma is as a tool to create designs for mobile applications. This is because you can easily edit and adjust details to your liking and get a solid idea of what you want your application to look like before you begin building it. It can also help to identify key features that may need to be added. This section details the usage of Figma in my development and will provide an analysis of all the current designs.

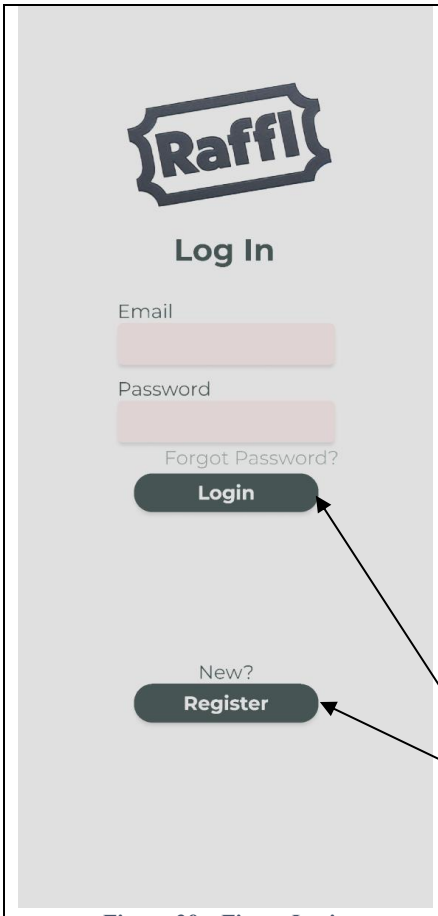


Figure 28 - Figma Login

The login screen in Raffl is very similar to the screen seen in the wireframe prototypes. The main changes made were the implementation of the app logo, the 'new' text above the register button, and the text being separated from boxes for a cleaner aesthetic.

The register screen is very similar to the login screen but without a 'forgot password' button. This is to ensure consistent design and not overcomplicate things. If I decide to add a phone number later, the register screen may change.

Users can access the home page with the higher 'login' and 'register' buttons, respectively, and switch between each other's buttons further down.

In both screens, users can enter their email and password into their corresponding boxes, and if there are any issues (i.e. password too short), these boxes should outline in red to warn users.

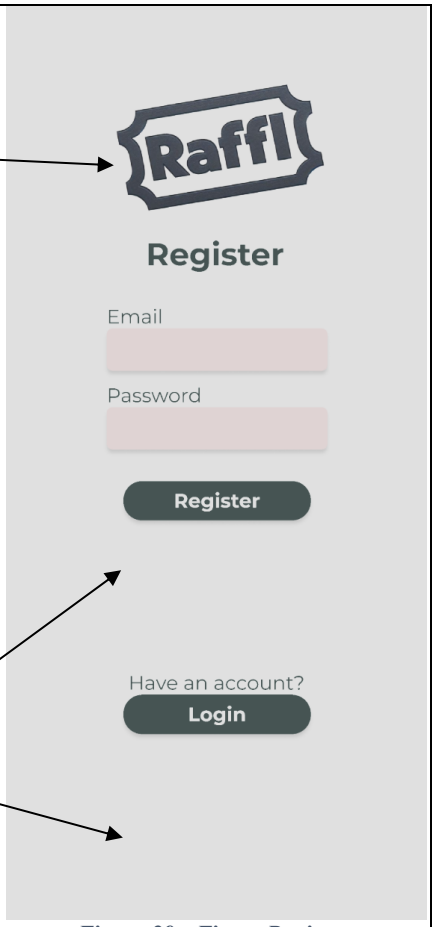


Figure 29 - Figma Register

Original Home Screen

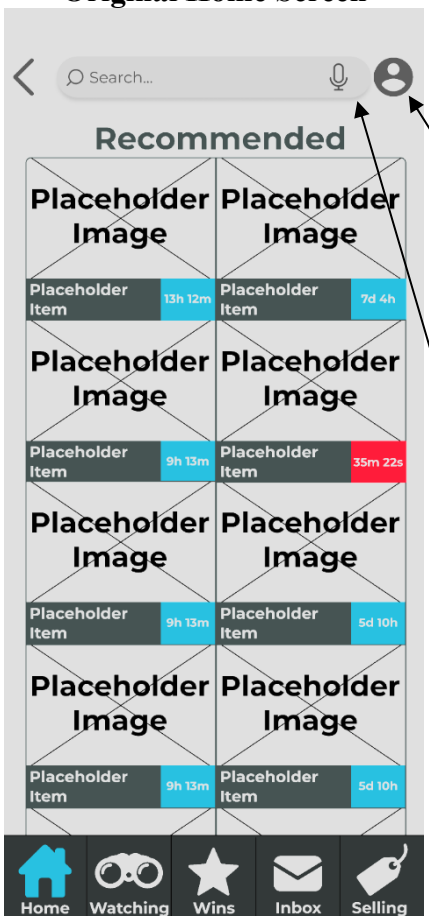


Figure 30 - Figma Old Home

The home screen for Raffl went through 2 iterations. The leftmost iteration is most like my paper prototypes but has been updated to include a button to access the profile page and a back button to navigate to earlier pages. This was updated to be more in line with other pages and to help if we navigate to the home page from any screen other than our login/register screens. There is also a microphone icon for voice recognition and accessibility.

This design also includes a live timer that counts down to the end of this listing. After laying the text out on two lines to improve spacing, I determined this still looked quite cramped, so I moved to the more open, less space-efficient design seen on the right. This was to reduce visual clutter, but it also gave us the advantage of adding item stats so users could see how listings were performing. Each tile also acts as a button, navigating us to the listing page of the same item.

New Home Screen

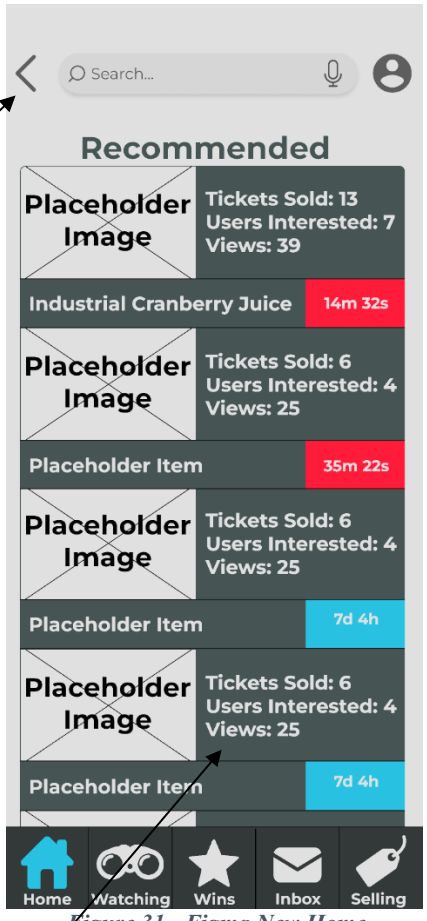


Figure 31 - Figma New Home

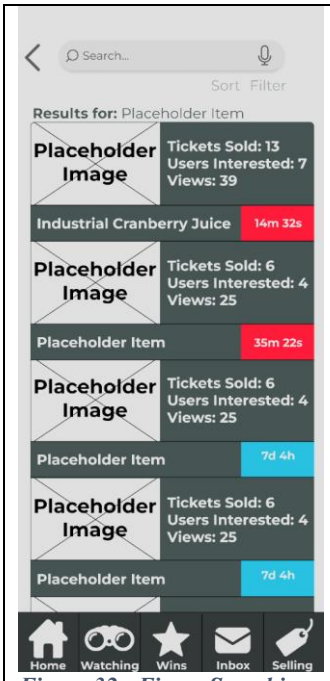


Figure 32 - Figma Searching

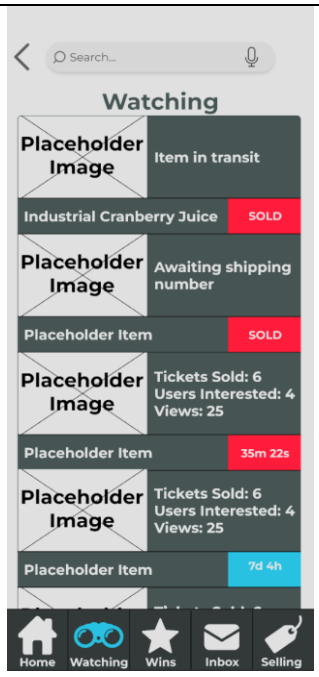


Figure 33 - Figma Watching

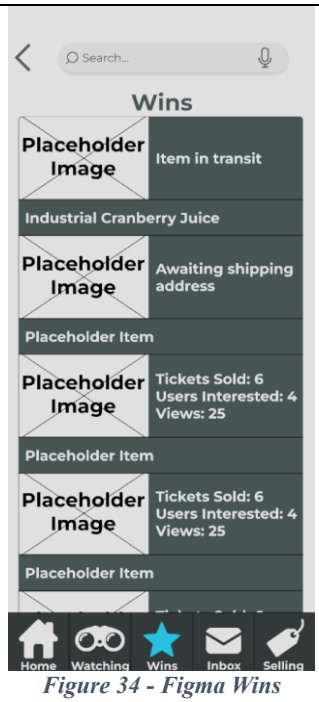


Figure 34 - Figma Wins

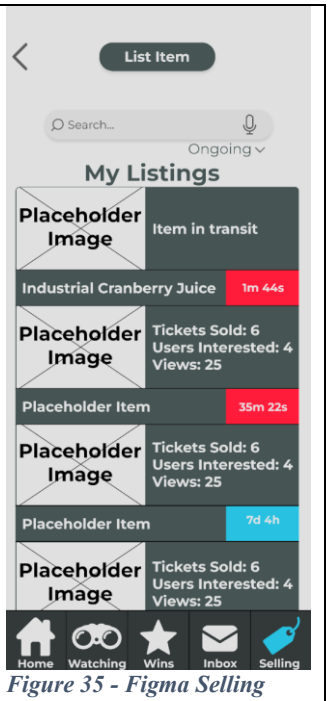


Figure 35 - Figma Selling

I have grouped the above pages together as they are all very similar to each other, barring minor differences. These similarities exist as an attempt to achieve consistent design across the app. Similarities include the layout of listing results, the bottom navigation bar (aside from its status), the back button and the search bar.

When looking at the navigation bar, you can see that, in Figure 33-35, our navigation bar is highlighted in blue. This is to indicate that we are currently on the selected page. The search screen is accessible via the search bar on the home page, not on the navbar, and as such, no asset is highlighted. When designing the navbar, I tried to make the icons logical whilst also providing a page title to make it clear for users to see what the icon does. This is relatively objective for some icons, such as the house for home and the mail for the inbox; however, the other buttons are less common in the industry, so I selected what I thought would be intuitive for users. I did this with a price tag for selling, binoculars for watching, and a star for wins.

Though it may not look like it, the search screen (Figure 32) is the most different to all the other screens. This is because we are filtering all results in our database rather than the subsets that we see on other screens. On this page, we have a ‘sort’ button, which can be used to sort by certain criteria, e.g. ‘ending soonest’. We also have the ‘filter’ button, where we can filter on various criteria, such as the price range for our tickets. I chose this layout as it is so common in the industry, with leaders such as Amazon and eBay having similar designs. It also provides great utility for users as they can easily find what they want by narrowing down items with specific criteria. The search screen also has a tag at the top, ‘results for:’, that reminds the user what they have searched for, which should highlight any typos or mistakes they may have made.

Our watch screen (Figure 33) and wins screen (Figure 34) are both very similar to each other. Both screens have a search bar that should filter through results locally on this page, which, unlike our home page, filters locally through items users are watching and have won, respectively, rather than the home page where we search the entire database. In the wins screen, Raffl does not show a 'SOLD' tag, as all items here have been sold, so the data would be redundant. For the items that are sold, instead of the statistics, we can also see the status of an item, such as 'item in transit' or 'awaiting shipping number', to give the user more relevant information.

Finally, on our selling screen (Figure 35), we have two additional properties not seen on the other screens. We have our 'list item' button, which is like the 'create listing' button found in our wireframes, and we have a filter that is currently on 'ongoing.' This is because it provides users easy access to differentiate between their sold listings, where they can view statistics and see what performs well, and current listings, where they can see how their currently listed items are doing. I have also kept the search bar, which was not seen on the paper wireframes, as I believe it still offers utility here whilst keeping consistency with the rest of the app.

Ongoing Listing Screens

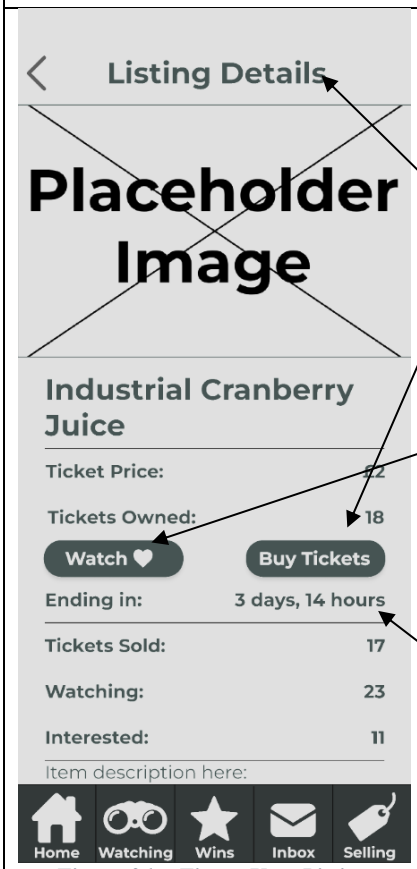


Figure 36 – Figma User Listing

These are the pages for the listings that are still ongoing. I have separated standard users and hosts, as different users should have different privileges. To make this change clear, the title of the page switches from 'Listing Details' to 'My Listing' if we are the host. These pages differ, as hosts cannot buy their own tickets, adhering to requirement #9. This button will allow users to buy tickets for an item they are interested in, giving them the opportunity to win it. I have also removed the ability for hosts to watch their own items, as these items will already land on their selling page, and I don't want to overcomplicate things.

Other than the changes mentioned above, these pages are identical. There is a live countdown timer on both pages that updates in real-time, conforming to requirement #27. The description length could potentially expand the size of the page beyond borders, so this whole section above the navbar and below the title is scrollable. This page can be navigated to from one of the many shortcuts found in Raffl, such as inbox notifications, or on the primary search page.

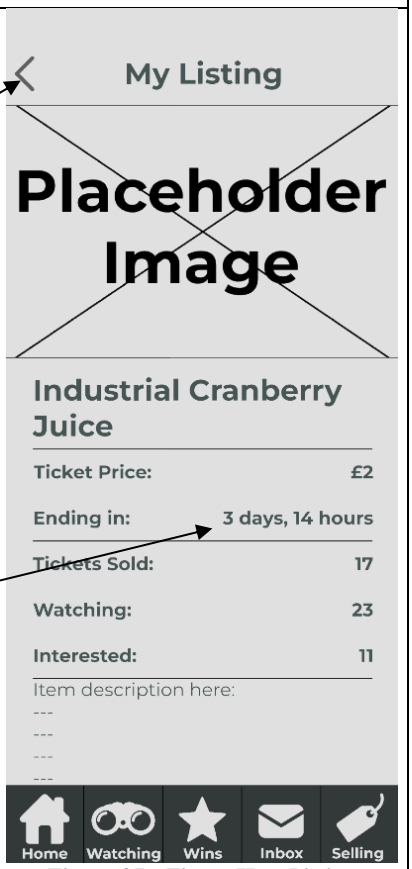


Figure 37 - Figma Host Listing

Completed Listing Screens

Figure 38 - Figma User Won

Figure 39 - Figma User Lost

Figure 40 - Figma Host Sold

Figure 41 - Figma Host Unsold

Figures 38-41 demonstrate the four possible page states for a listing that has finished, i.e. the end date of the listing has passed. 'Users Won' (Figure 38) is reachable by winners, who will also have been alerted by a push notification. 'Users Lost' (Figure 39) is accessible to any user who is not the host and has not won the item; they don't necessarily need to have bought a ticket. On the 'Host Sold' page (Figure 40), hosts can see their successful auctions. Finally, our 'Host Unsold' (Figure 41) page is for hosts who have listed an item but no users bought any tickets, so the listing has finished but not sold.

The main difference between these pages and their unfinished counterparts is that as these listings are over, there is no option for new users to buy tickets. Users still have the option to watch and un-watch said item, as it may still be of interest to them for future reference (i.e. identifying how many tickets were sold for x item), so this button has been left in. The item description for most pages has been changed to their status (i.e. awaiting user details or adding shipping details) for all users besides users who did not win, as the status of the item is not of interest to them at this point. The 'ending in' field has also been eliminated, as this date has passed and is no longer of interest to users.

The general structure of these pages, including the ongoing listings (Figure 36-41), is very similar. Like most pages in Raffl, we have the standard structure of a bottom navbar and a title to the page with a back button for navigation. These encapsulate the core of this page, which contains details of the current listing we are viewing. Users can scroll through this content to ensure they don't miss anything. The start of this subpage is the 4:3 image, which is consistent across all listings to ensure a consistent viewing experience for users. An image the host has uploaded will be shown here in place of the current placeholder. This is followed by the listing name, the ticket price, and various statistics about the ticket. Most of the statistics are common sense, however the user's interested statistic is not. This indicates the number of individual users who are interested enough to buy tickets. For instance, if a user

buys three tickets and another user buys two tickets, we have five tickets sold, but only two users are interested. Finally, we have either the description of the product or the status of the item depending on the status of the item and its relationship to our user.

For users who have won, this page has been altered slightly to give users a message of congratulations for their win, as well as the status of the item below. If users need to act, this will be contained within the status message. As an example, if a user needs to submit a shipping address, the status will say ‘awaiting shipping address’ and a button will appear to allow users to add said address.

The ‘Users Lost’ page (Figure 39) is accessible for any user who has not won the item and does not necessarily mean they had any interest or bought any tickets. This page has had minimal changes compared to others but does have a message to inform users that they have not won this item, to avoid any possible confusion.

We then have the host pages (Figure 40-41). If a host has not managed to sell any tickets, this page will reflect that to said host with a message where the description would be. If a host has sold any tickets (which guarantees a winner), then this page will instead be updated, much like the ‘Users Won’ page, where the host will get the status of the item, along with any action they need to take.

Users can navigate to the ‘create listing’ screen from the ‘selling screen,’ and the purpose of this page is for users to create their listings. This means that users can enter their name, tags for the item, the end date this listing is ending, the description, the image and the ticket price. The end date will likely be a DD/MM/YYYY format to stick to the current UK standards and will result in a calendar dropdown upon selection. This page is also scrollable for devices where it does not fit on screen.

After submitting an image using the ‘add image’ button, this icon will update and be replaced with the new image that the user has uploaded to indicate that the upload was a success. The user will then be able to replace this image by clicking the same button again and going through the same process.

The create listing button will then push the listing to the database, providing that no fields are blank. The exception to this rule is the ‘tags’ field, as it does not necessarily need to contain content for the listing to be valid, as this is only for recommendation purposes and searching.

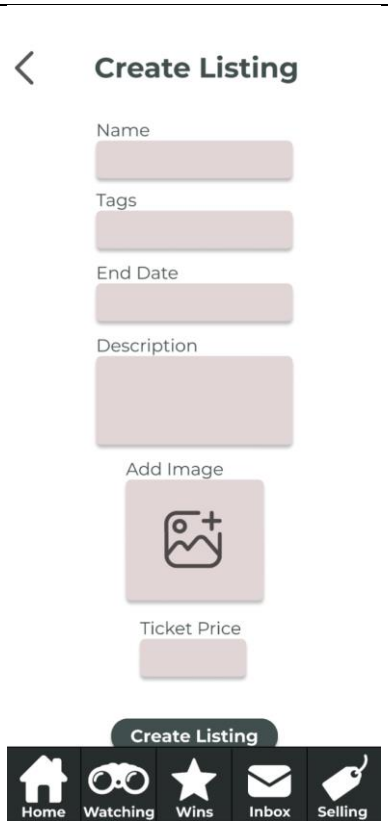


Figure 42 - Figma Create Listing

Figure 43 – Figma Inbox

This is the inbox screen for Raffl, and where we can see all relevant status updates for items, such as 'tickets bought' or 'item won'. These are technically 'notifications' for the items a user is interested in, though not all of these are important enough to constitute a push notification. If a user buys tickets, they may want that in their inbox to view the history but do not need a notification pop-up telling them this. Conversely, if a user wins an item, having a push notification to alert them is likely to be useful.

Like most pages, it has the default structure with the title and back arrow at the top, along with the navbar at the bottom. The main contents of the page are a scrollable list, of all the notifications the user has received, in descending date order so the latest item is displayed first. These items are also clickable and should send you to the related listing details page upon tap.

Each inbox item now comes with a timestamp to give the user more information about when this notification was sent. Instead of cramming the status update details into 1 line, I have spread them over 2. The first line is the name of the listing that this notification is related to, i.e. 'apple,' whilst the second line is the details of the notification itself, i.e. 'tickets bought.' I have also separated the inbox items to make the screen look less compact and more open to fall in line with the more minimalistic design approach I am aiming for. The final change is minor but has to do with the newly rounded corners to fit the more modern aesthetics and match recent apps.

We can navigate to this page by clicking the inbox button, which is indicated by the mail symbol in the navbar. This is highlighted in blue to indicate we are on this page.

The final page in Raffl is my profile page. This page is a relatively basic page and does not have too much information. It has the standard structure with a title, a back button and a bottom navbar.

The contents of this page consist of a profile icon, that is currently to fill blank space and could be replaced by a profile picture if user accounts are expanded in future iterations. We are then faced with the email of the account we are currently logged in at, along with how many credits we have.

Credits are my placeholder solution for cash transactions. As this is a prototype app and is not meant for production, I wanted to simulate transactions rather than use a banking service. To do this, I have virtual credits that are added when users 'add credits' using the add credits button. Alternatively, users will receive the credits generated by listings that they sold, but only after a user has confirmed their item as received. These credits will also be deducted from users when they buy tickets at the cost of '(ticket price * ticket amount).

Finally, users can log out of their current account using the 'log out' button, which will also send them back to the login screen.

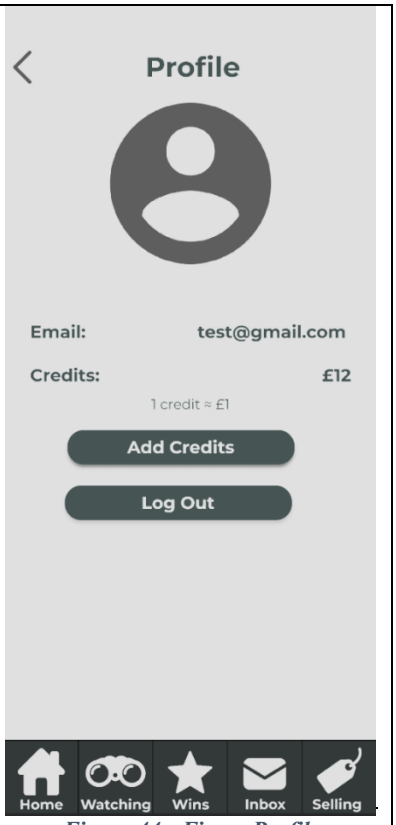


Figure 44 - Figma Profile

5.4 – Database/Storage Design

Whilst NoSQL databases do not have a rigid schema like their SQL counterparts, this does not mean that there is no definable structure in which the data is stored. This section details how the database has been designed and the final structure found within Raffl.

Approaching a NoSQL database was a relatively new challenge for me. Whilst I had experimented in the past, such as briefly using one in an earlier project for university, they were all very surface-level implementations and didn't dive deep into the capabilities of a NoSQL database. When defining an SQL database, it is recommended that the database is fully visualised before starting your project to ensure that data is structured in a way that works with other tables and doesn't need a major redesign later down the line. This is not strictly true in NoSQL databases, which are more flexible.

I decided, when designing my NoSQL database, to have a rough idea of all the data I would need to represent and how I could do so in a good manner. To do this, I made a loose structure of all the data I would need and how I would represent it. I split it into two categories: 'User' and 'Listings'. The 'User' category would involve all the things necessary to represent a User in Raffl, including how many credits a user has, the items a user is watching, and more. On the other hand, the 'Listings' category involved all the data regarding individual listings, including the User ID of its host and all other data required. After ensuring I had a good amount of the data structure mapped out and that no future changes would require the data to be restructured, I got to work on creating Raffl.

During the creation of Raffl, there was occasionally data that I hadn't realised I would need to account for, such as storing a user's 'Notification Token' in the database, and I needed to make minor changes to my structure, but the overarching structure stayed consistent throughout the design of my app. Below, I cover the final layout of my database and will make notes on each data point, specifying how I came up with it and if it was realised from the beginning or during the development of Raffl. The following subsections detail the 'UserData' and 'Listings' collections, respectively, and cover all the data required for Raffl.

5.4.1 – User Data Collection

The ‘UserData’ collection contains all data related to a user of Raffl. This section covers all the data that is stored in the ‘UserData’ collection whilst providing an analysis to justify why it is stored like this.

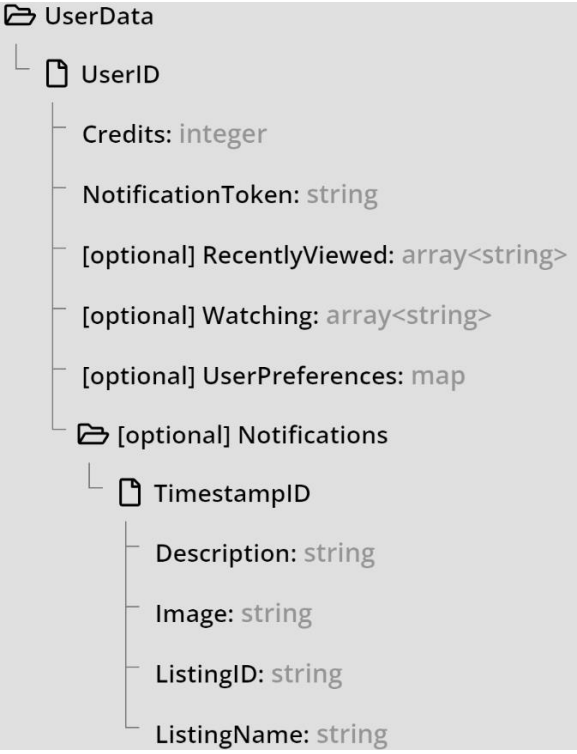


Figure 45 - User Data Collection

The ‘UserData’ collection has a unique document for every user that exists in Raffl. As there is as many users as there is UserData, and user IDs are unique, it makes logical sense to give documents the same ID as the user, as defined by the Firebase authentication service. This gives us the ability to quickly query the database, for anything related to our current user with ease.

Every document for users contains both the credits, and the notification token for a user. This is the most logical place to store them, as they are easily accessible with our user ID and can be efficiently queried. The initial design of this data store did not include the notification token, as I was unaware how firebase notifications were handled, so was added later down the line.

There are also optional fields within a user’s data document. The first of these is the ‘RecentlyViewed’ field, which contains the Listing ID’s for the two listings that a user has visited. Then there is the ‘Watching’ field, which contains all the Listing ID’s for listings that a user is watching. Finally, we have the ‘UserPreferences’ map, which contains a selection of tags, along with a numerical value to indicate a user’s preference towards said tags. This is used in an algorithm to calculate our user’s recommended results and tailor their feed, and as such, it will be covered in depth in the implementation section. Of all these fields, as I wasn’t originally planning a ‘RecentlyViewed’ category, and I was unsure of how I was going to implement my recommendations, these sections did not exist in the initial draft. This contrasts with the ‘Watching’ field, which was there from the start.

This collection also has a child in user notifications, as this is how they are stored. This collection is for all the entries on the inbox page for a user and is how Raffl permanently stores push notifications for a user. They have been indexed with an ID that is the timestamp they were created in, as this is not only guaranteed to be unique but also saves us from storing the timestamp whilst also making it incredibly easy to sort by the date created. Each notification stores the description of our notification, such as ‘congratulations, you have won,’ the listing ID, and the image and name of the listing to which this notification relates.

5.4.2 – Listing Collection

This section is almost identical to the previous section but instead aims to cover and justify the structure of the 'Listings' collection within Raffl's Firestore database.

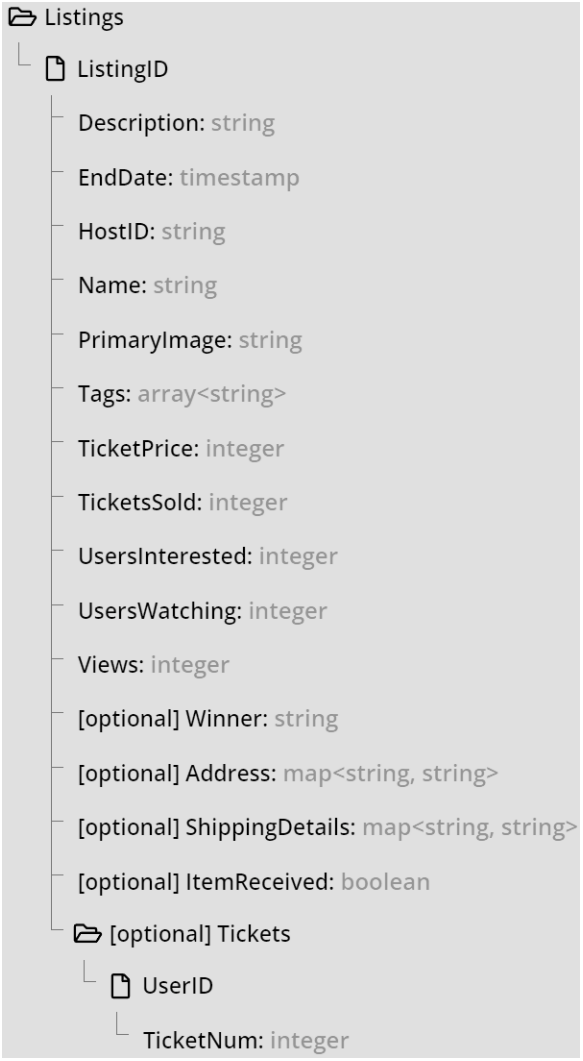


Figure 46 - Listing Collection

The 'Listings' collection stores data for every listing found within Raffl. Each document is stored with a unique 'Listing ID' that is created randomly upon generation of the new listing.

Every listing includes a certain amount of descriptive information for the item listed. These include a name and a description to give users an idea of what they are buying, along with a URL to the image of said item, its end date, and the price of each ticket. A host ID is a reference (like a foreign key in SQL) to the user running this listing. Tags are also included in this, though they are hidden from interested users and are instead used for both search results and the recommendation engine.

The other mandatory fields found in a listing are the stats of an item. These metrics aim to give hosts insight into an item they have listed and give potential buyers an idea of how popular an item is. The 'views' metric is an integer to represent how many non-host views a listing has received, while the 'tickets sold' metric indicates how many tickets have been sold for an individual item. There is also a 'users watching' metric for how many users are watching, and finally, a 'users interested' metric, which represents how many unique users have bought a ticket, so essentially, how many interested parties a listing has.

The optional fields found in a listing are all related to its status and are added in sequential order to listings after they end. First up, there is the 'Winner' field, which contains the ID of the user who has won this raffle. This field will only be created if tickets are sold; otherwise, there will be no winner. Next, the address field, which is a map containing address attributes such as a postcode and address line, can be created. This can only be created by a user after they have won an item and cannot be modified. Following this, there is the 'ShippingDetails' field, which the seller should add after a user has won an item. This includes the tracking number and courier but can be left blank if these are not included. Finally, we have the 'ItemReceived' field, which the winner can check once shipping details have been added. This confirms that a winner has received their item, and the transaction is complete.

There is also an optional collection in listings, and that is 'Tickets.' This collection is relatively simple and contains a document for every user who has bought tickets for this item.

It is indexed by the user's ID. This simple collection just stores the number of tickets a user owns, and as such can be updated when they buy more.

5.4.3 – Image Storage

The Firebase Firestore database does not support image storage, and as such, other means are necessary to store images. The most obvious solution is Firebase's in-house storage solution, which supports image storage. The images hosted here can be accessed via a simple URL, and as such, can be saved in our Firebase database under the 'primaryImage' String. This string can then be read into Raffl, and directly used to display images.

Name	Size	Type	Last modified
1708613203868	920.65 KB	image/jpeg	Feb 22, 2024
1711640376249	407.77 KB	image/jpeg	Mar 28, 2024
1711640414391	312.4 KB	image/jpeg	Mar 28, 2024
1711640499106	292.02 KB	image/jpeg	Mar 28, 2024
1711640821381	294.28 KB	image/jpeg	Mar 28, 2024

Figure 47 - Firebase Image Storage

6 – Implementation

This section details the implementation of Raffl and shows the final app that was developed as a result. I will also demonstrate how some of the application was made, such as the coding behind it when showing features under the screen's subsection, though will not cover this for every section as there is too much code to fully handle.

6.1 – Project Structure

This section underlines the overall structure of Raffl, defined within flutter, and how all the elements of it have been handled.

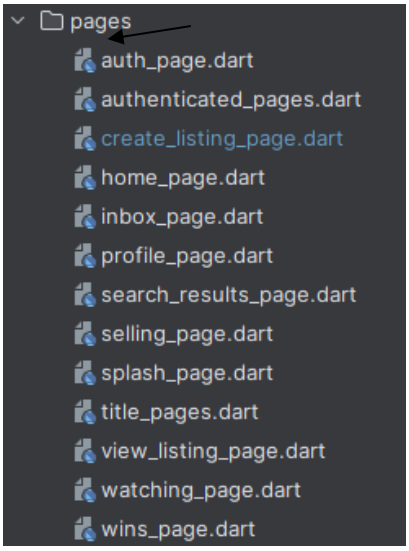


Figure 48 - Raffl Page Directory

The pages directory in Raffl contains all the individual screens of Raffl, where each page corresponds to a full-screen widget. One of these pages stands out, and that is the 'authenticated_pages' file. This encapsulates all pages that require authentication (i.e. every page except the login and register page, which are represented within auth_page).

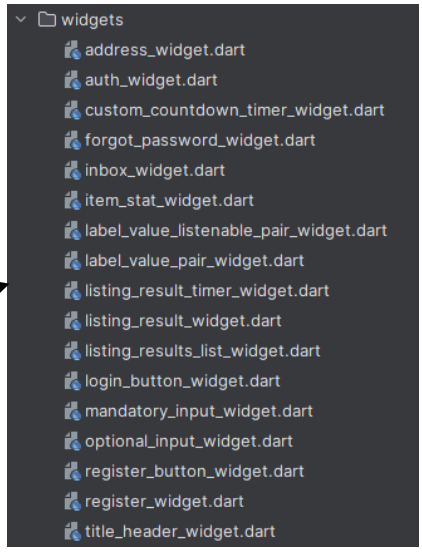


Figure 49 - Raffl Widget Directory

Similarly to this, the widgets directory contains all the custom widgets within Raffl that don't constitute a whole page. Many of these are common blueprints that are reused throughout the app to enforce consistent design.

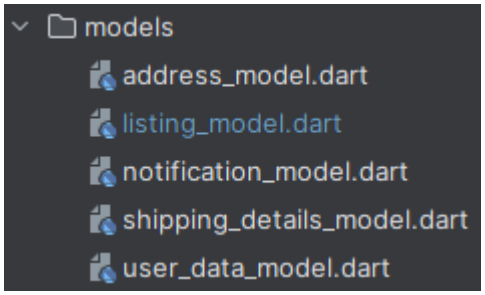


Figure 50 - Raffl Model Directory

The model's directory stores all of the models within Raffl. Models represent our data and act as a blueprint for creating objects. For instance, the UserDataModel defines all the possible attributes for our user data and can map it to an object stored locally within the application. Objects can also be created using this model and then used when writing to the database.

This directory contains each repository required for Raffl. These files are responsible for directly communicating with the databases, such as Firestore and Algolia. They utilise methods from our models to convert items to and from each format and handle all the transaction logic within Raffl.

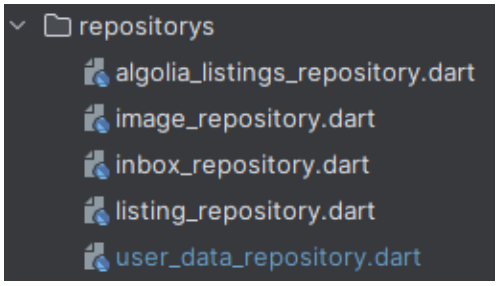


Figure 51 - Raffl Repository Directory

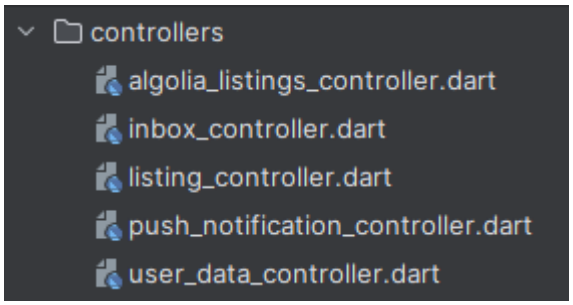


Figure 52 - Raffl Controllers Directory

Controllers in Raffl act as middlemen between the user interface and data repository classes. They handle all the requests and the appropriate repository to route them to. These requests include creating/modifying data and retrieving data where they can be read by Raffl.

The functions directory in Raffl only contains one dart file, the 'access_auth' file. This handles the authentication logic required to login to our user accounts through Firebase authentication.

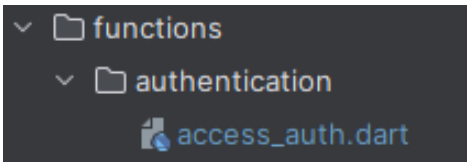
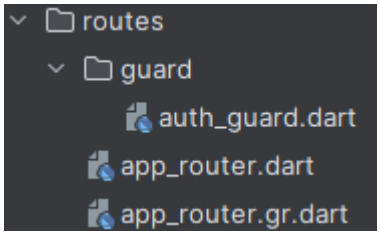
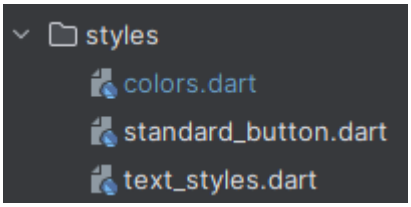


Figure 53 - Raffl Functions Directory



The routes directory in Raffl, handles routing to all the possible accessible pages. This uses the 'auto_route' package which provides a good solution to guarding certain pages that require authentication and simplifies the routing process.

Here all of the style choices of our app are contained, so if anything changes, we can easily modify attributes such as colours, and the text styles within our app without manually changing everything.



There are two important files that do not have their own directory but instead within the overarching 'lib' directory that contains the above directories. These are the 'main.dart' and 'pubspec.yaml' files, respectively. The 'main' file is the entry point for Raffl and contains the main() function, which is the first function to run and kickstart the application. The 'pubspec' directory, on the other hand, contains all the dependencies required to run Raffl, such as 'auto_route', which was mentioned earlier.

6.2 – Interface and Code Overview

This section covers the separate screens found within Raffl, along with explanations of how functionality within these screens was implemented. There is too much code to completely cover, but the more important coding details will also be included here.

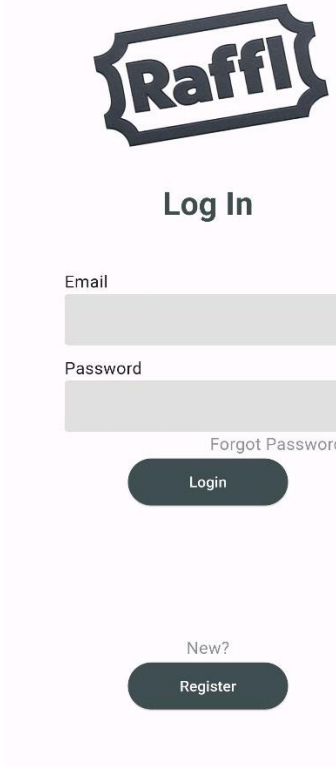


Figure 54 - Raffl Login

The login screen and register screen are nearly identical to their initial designs in Figma. This is because I was happy with how they came out and didn't want to stray far from the design. Both the log-in and register pages were implemented slightly differently compared to other pages in Raffl. I chose to have them both under an 'auth_page' rather than two separate pages, as I wanted to make transitioning between the pages seamless and not require a whole page regeneration as they are so similar. As such, elements are swapped out when pressing the bottom 'Register' and 'Login' buttons to match the screen type, and various functionality (i.e. password formatting) is only enabled on the relevant page. The login screen is the first page you are greeted with when accessing Raffl.

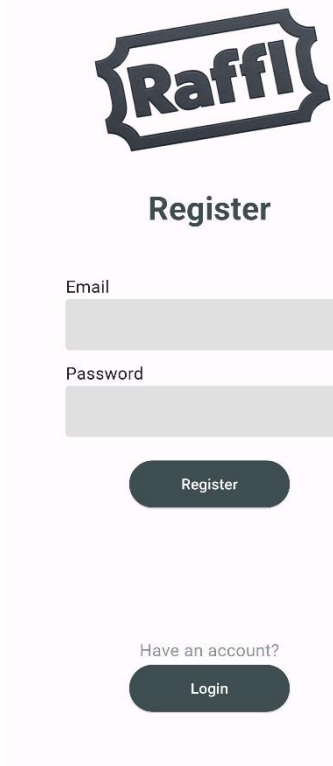


Figure 55 - Raffl Register

```
validator: (value) {
    if(!widget.login){
      if (value == null || value.length < 8) {
        return 'Enter 8 characters minimum';
        //The rest uses RegEx to verify certain criteria
      } else if (!RegExp(r'^(?=.*[A-Z])').hasMatch(value)) {
        return 'Must contain 1 uppercase character';
      } else if (!RegExp(r'^(?=.*[a-z])').hasMatch(value)) {
        return 'Must contain 1 lowercase character';
      } else if (!RegExp(r'^(?=.*\d)').hasMatch(value)) {
        return 'Must contain 1 number';
      } else if (!RegExp(r'^(?=.*?[!@#\$&*~]).{8,}$').hasMatch(va
        return 'Must contain 1 special character';
      }
    }
  return null;
}
```

Figure 56 - Password Syntax Checker

When registering, we have logic to ensure the password is valid using regular expressions.

```
GestureDetector(
  onTap: () async{
    await FirebaseAuth.instance.sendPasswordResetEmail(email: emailControll
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Reset link sent to Email'),
      ),  // SnackBar
    );
  },
  child: Align(
      alignment: Alignment.bottomRight,
      child: Text(
        'Forgot Password?',
        style: myTextStyles.fadedText,
      )  // Text
  ),  // Align
):  // GestureDetector
```

Figure 57 - Forgot Password

The forgot password button uses Firebase to send a reset email to our users.

E-mail verification is usually required to login, and is demonstrated later in the testing section, but under debug builds, to make it easier for experimentation and using multiple accounts, this functionality has been disabled and users are directly logged in upon registering.

```
UserCredential userDetails = await FirebaseAuth.instance.createUserWithEmailAndPassword(
  email: email,
  password: password,
);
await userDetails;
/*
await userDetails.user!.sendEmailVerification(); //Sends email verification to user
ScaffoldMessenger.of(context).showSnackBar(
  SnackBar(
    content: Text('Awaiting email verification.'),
  ),
);*/
```

Figure 58 - Email Verification

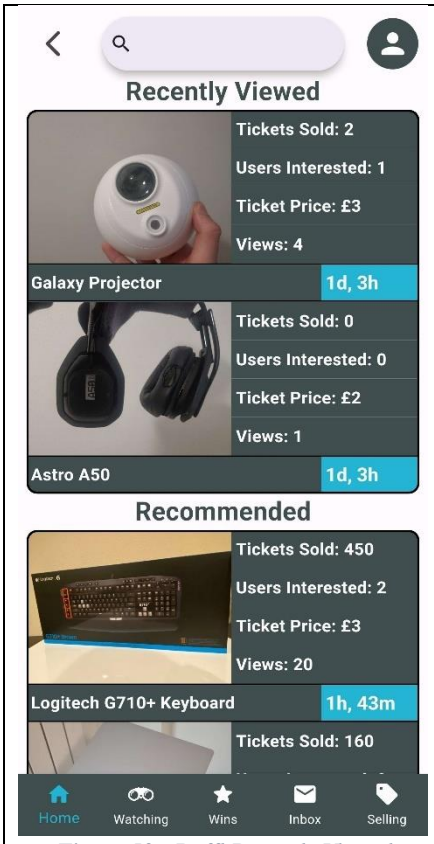


Figure 59 - Raffl Recently Viewed

Here, we have the home screen. As you can see, this is a bit different to the original Figma design. I decided adding a 'Recently Viewed' section to the home page would help users better navigate to products they are interested in whilst also making the page less bland. This page is scrollable, so you can view both the recommended and recently viewed items with ease. Neither category will contain items hosted by our current user, as they can easily be found in the 'selling tab.' The top of the page has a navigation bar, which can be used to search through the entire Raffl catalogue, along with a profile button to get to the profile page and a back button to navigate back through the app. The back button is not on the home screen in Figma prototypes but has been added for consistency, as it still has utility if we want to go back to an earlier page after navigating through the app.
I also added the 'ticket price' information to listing widgets, as this is important for users who won't want to navigate to a listing page to find out.

The biggest visual draw seen on this screen is the widget containing my listing and its details. This widget is relatively consistent across 'Raffl' and can be seen on many screens, albeit sometimes with slightly different properties. As Flutter takes up a large amount of screen space, I will not show the entire widget but instead show sections of it and describe others. This is

```
return GestureDetector(
    //Constructs 'ListingResultWidget' for each separate listing
    child: ListingResultWidget(
        name: name,
        endDate: endDate,
        primaryImageUrl: imageUrl,
        ticketsSold: ticketsSold,
        usersInterested: usersInterested,
        views: views,
        ticketPrice: ticketPrice),  // ListingResultWidget
    onTap: () => AutoRouter.of(context)
```

Figure 60 - Listing Gesture Detector

done with the user of a 'FutureBuilder' in Flutter, which is a widget that builds itself based on the latest snapshot interaction with a 'Future' [FLUTTER SOURCE]. Futures are essentially objects that represent a value that is not yet available but will be in the future. This allows our page to build before our query returns all our results and then builds it as specified. This snapshot is then plugged into a 'ListView.builder', which creates a scrollable grid of widgets defined by the size of our snapshot. The widgets generated by the ListView builder can be seen in Figure 60 above and are a 'ListingResultWidget' that has been encapsulated in a gesture detector. The purpose of the gesture detector is to direct us to the listing page to which this item corresponds. As for the 'ListingResultWidget,' this is a custom widget that is set out to display the results of an individual listing.

The code block for Raffl's 'ListingResultWidget' is too big to be shown without bloating the page, as Flutter widgets take up a lot of space, but it is essentially a widget that utilises the 'Flex' widget, which tells the widget what proportion of the available screen space it can take up. This ensures the layout is relatively consistent across devices while also ensuring we have a 4:3 slot for our images. This widget also contains the custom 'ListingResultTimerWidget.' This is shown in Figure 61 and is what displays our timer. It does this by sending a 'ValueChanged' listener to a custom countdown timer widget.

```
return Container(
  color: containerColor,
  child: Padding(
    padding: const EdgeInsets.only(left: 4.0, right: 4.0),
    child: Align(
      alignment: Alignment.centerLeft,
      child: DefaultTextStyle(
        style: TextStyle(
          color: primaryColor,
          fontSize: 18,
          fontWeight: FontWeight.bold,
        ),  // TextStyle
        child: CustomCountdownTimer(
          endTime: widget.endTime,
          lessThanHour: handleLessThanHour,
        ),  // CustomCountdownTimer
      ),  // DefaultTextStyle
    ),  // Align
  ),  // Padding
);  // Container
```

Figure 61 - Listing Result Widget

```
Color containerColor = Colors.blue;
bool isLessThanHour = false;
void handleLessThanHour(isLessThanHour) {
  //Changes widget to red colour if we have less than an hour left
  if (isLessThanHour) {
    WidgetsBinding.instance.addPostFrameCallback((_) {
      if (mounted) {  // Add this check
        setState(() {
          containerColor = warningColor;
        });
      }
    });
  }
}
```

Figure 62 - Custom Countdown Timer

This custom countdown timer widget will modify the 'handleLessThanHour' listener, changing it to false when the time drops below an hour, which flags the widget and tells it to change the timer colour to red if the time drops below an hour. The CustomCountdownTimer widget also displays the time in the most relevant format, such as 'hours and minutes' or 'minutes and seconds,' ensuring that users always have an accurate insight into the status of an item and exactly when it is ending.

To display the 'RecentlyViewed' section, Raffl queried the UserData collection in the database to retrieve the corresponding 'RecentlyViewed' field. This field is updated when a user views any item they do not host, where the 'updateRecentlyViewed' function is fired. This retrieves an array of the current documents that a user has recently viewed and checks if our newly viewed document is in this array. If it is in this array, Raffl will ensure that it is in position 0, indicating that it is the most recently viewed item, and if not, it will swap the order of the array to ensure this. Otherwise, Raffl would push the newly viewed listing onto the front of the array and push the final listing off the back. Finally, if the 'RecentlyViewed' array

has been altered in any way, this new array is pushed to the database, updating the previous values.

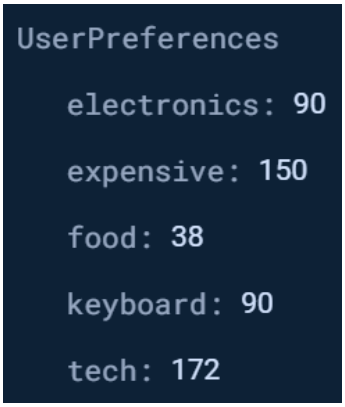


Figure 63 - User Preferences

After retrieving the recently viewed listings that a user had looked at, Raffl proceeds to calculate the recommended listings for a user. In the Firestore, each user listing has a set of preferences, which are updated when buying tickets for an item. A multiplier for all tags on the said item is applied based on the number of credits the item costs, and the corresponding value in the database is updated. This can be seen in Figure 63, where different tags have different levels of preference towards each other. To calculate the recommended items for a user, a blacklist is generated for items we do not want, and this is initially filled with the listings in, recently viewed, as seen in Figure 64. This is to avoid having a listing re-appear on the homepage in both the recommendations and the recently viewed, as this would be redundant. Next, the user's preferences are recovered from the database. We then get a total range based on all the multiplier values, where each individual preference is mapped to part of the range. For instance, from Figure 63, electronics would represent the first 90 digits in the range, expensive the next 150 and so on. Then, a while loop is entered, which checks that we are not at the maximum recommendation size (which has been set to 4 but could be any number) and also checks that we still have preferences in our map. Within this while loop, a random number in our current total range is selected, where we enter a for loop to calculate what preference this number represents. This tag is then removed from our user preferences, along with its range, to ensure that we get diverse recommendations and don't target the same tag again. Then a random listing with this preference tag is retrieved using 'getRecommendedListingByTag' and given it, the blacklist to ensure that we do not accidentally retrieve this listing again. We have reached the end of the initial while loop, and we repeat this until we have enough recommendations or until we run out of preferences for the user to pick from.

```
while(integerPreferencesMap.length > 0 && recommendationsList.length < recommendationsLength){
  print("Total range: ${totalRange}");
  Random random = new Random();
  //Random number corresponding to a preference we have is selected
  int randomTagNum = random.nextInt(totalRange);
  print(randomTagNum);
  int currentRange = 0;

  for (var entry in integerPreferencesMap.entries) {
    int multiplier = entry.value;
    currentRange += multiplier;
    //Calculate what preference our current random number is pointing to
    if(randomTagNum < currentRange){
      String selectedTag = entry.key;

      //Remove current tag and its range from the preferences so we don't select it again
      integerPreferencesMap.remove(selectedTag);
      totalRange -= multiplier;
      //Gets a listing with the corresponding tag that is NOT blacklisted
      ListingModel? randomListing = await listingRepository.getRecommendedListingByTag(selectedTag, blacklist);
      if(randomListing != null){
        recommendationsList.add(randomListing);
        blacklist.add(randomListing.getDocumentID()); //To avoid re-selecting the same listing
      }
      break;
    }
  }
```

Figure 64 - Recommendations Algorithm #1

```
//If we still need to fill space in the recommendations but haven't yet, select random listings as follows
while(recommendationsList.length < recommendationsLength){
  ListingModel? randomListing = await listingRepository.getRecommendListingRandom(blacklist);
  if(randomListing != null){
    recommendationsList.add(randomListing);
    blacklist.add(randomListing.getDocumentID()); //To avoid re-selecting the same listing
  }
}
```

Figure 65 - Recommendations Algorithm 6#2

If we run out of preferences, a random selection of listings will be selected until we have populated the recommendations list in its entirety.

```
Future<ListingModel?> getRecommendedListingByTag(String tag, List<String> blacklist) async {
  Filter hostFilter = Filter("HostID", isNotEqualTo: FirebaseAuth.instance.currentUser!.uid);
  Filter tagsFilter = Filter("Tags", arrayContains: tag);
  final snapshot = await db.collection("Listings").where(Filter.and(hostFilter, tagsFilter)).get();
  List<ListingModel> listings = snapshot.docs.map((e) => ListingModel.fromFirestore(e, 0)).toList();
  if(listings.isNotEmpty){
    return await selectListing(listings, blacklist);
  }
  return null;
}
```

Figure 66 - Recommended Retrieval

To retrieve a random recommended listing from the Firestore database, a query was constructed to get a random listing with a filter to avoid our host documents and get listings that include the tag that we are filtering on. Firebase doesn't like querying on the document ID and other filters at the same time, so all listings with this tag are retrieved on the user side and manually filtered using 'selectListing' which is a relatively simple function, which gets a subarray that only contains documents that are not in the blacklist and returns a random document from this selection.

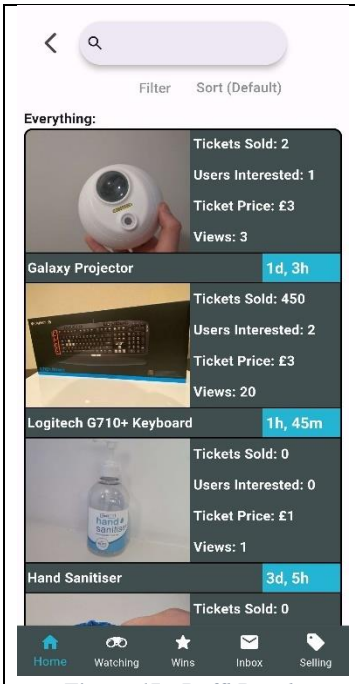


Figure 67 - Raffl Results

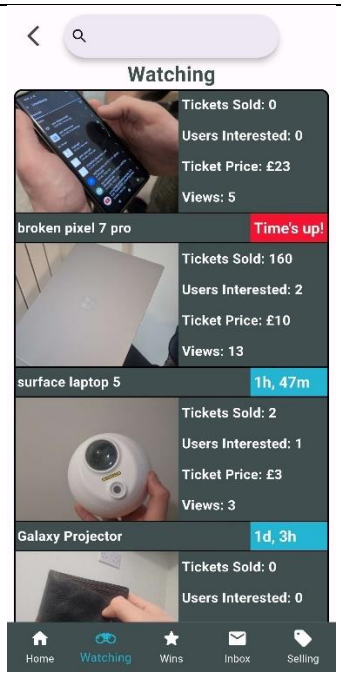


Figure 68 - Raffl Watching

Figure 69 - Raffl Wins

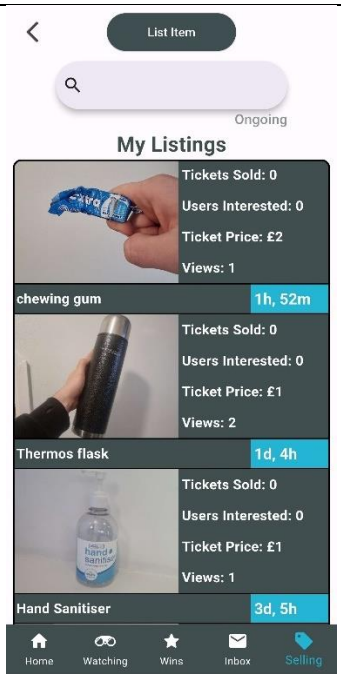


Figure 70 - Raffl Selling

The screens above Figure 66-69 show all other pages with a similar configuration to our home page. Most of these pages act the same, aside from the ‘Searching’ screen, which has a different way of querying. Figures 67-69 use the standard text filtering method, where our page is updated in real time based on the user input. I do this to avoid re-querying the database, as this is a small subset of all possible products on Raffl and thus does not require complex querying. Of these pages, Figure 68 stands out slightly, as it does not have a timer on the listings. This is because all listings in the ‘Won’ category will have finished, so the timer would be redundant. There is also a filter in Figure 69, currently set to ‘ongoing,’ and when flipped, it will display all the listings that a user has that have ended. Figures 67-69 directly query from Firebase, as there is no need for anything more powerful, especially since the results filtering is done locally. Figure 70, for example, shows how we retrieve all the documents for winners, which is just a simple query to our Firestore DB. You can also observe the ‘ListItem’ button on our selling screen, which takes us to the ‘create listing’ screen later down the line.

```dart
Future<List<ListingModel>> getWins(String userID) async {
  final snapshot = await db.collection("Listings").where('Winner', isEqualTo: userID).get();
  final listing = snapshot.docs.map((e) => ListingModel.fromFirestore(e, 0)).toList();
  return listing;
}
```

Figure 72 - Wins Retrieval

```dart
Future<List<ListingModel>> getSearchResults(String searchQuery,
    String? optionalSort, bool soldItems, RangeValues priceRange) async {
  int currentTime = DateTime.now().millisecondsSinceEpoch;

  AlgoliaQuery query = algolia.instance.index('listings_index').query(searchQuery);
  // Add price range filter
  print("Price ranges:");
  String filterQuery;
  if(soldItems){
    filterQuery = "EndDate <= ${currentTime}";
  }else{
    filterQuery = "EndDate > ${currentTime}";
  }

  filterQuery += " AND TicketPrice:${priceRange.start.toInt()} TO ${priceRange.end.toInt()}";
  query = query.filters(filterQuery);
  print(query);

  final snapshot = await query.getObjects();
  final searchResults = snapshot.hits.map((e) => ListingModel.fromAlgolia(e)).toList();
  if (optionalSort != null) {
    print("Sorting results with ${optionalSort}");
    if (optionalSort == 'Cheapest Tickets') {
      searchResults.sort((a, b) => a.ticketPrice!.compareTo(b.ticketPrice!));
    } else if (optionalSort == 'Ending Soonest') {
      searchResults.sort((a, b) => a.endDate.compareTo(b.endDate));
    } else if (optionalSort == 'Most Sold') {
      searchResults.sort((a, b) => b.ticketsSold!.compareTo(a.ticketsSold!));
    } else if (optionalSort == 'Most Viewed') {
      searchResults.sort((a, b) => a.views!.compareTo(b.views!));
    }
  }
  return searchResults;
}
```

Figure 71 - Search Results Query

On the search results page, querying is done with a service called ‘Algolia’ instead of ‘Firebase Firestore.’ This is because Algolia supports full-text search, unlike Firebase, and on this page, Raffl is searching across the whole database rather than a small subset of it, so I wanted a powerful search to capture exactly what the user is after rather than missing substantial parts. When querying Firebase, if the search text doesn’t directly match part of the text in a listing, it will not be contained with results, so without directly requesting the right data, users may miss out on relevant important results, where they do not with this approach. Algolia is, however, very costly to run as it clones the results in the database that we are searching for, including every update. It also just generally costs a lot more per search request than a service like Flutter, so it is used sparingly throughout the app. To avoid incurring costs, I have stuck within the free tier and have also limited the data indexed by Algolia to only what is directly necessary. Figure 71 shows all the data Algolia indexes for an example listing, such as the tags (for searching) and then all the other attributes which are required for display in the scrollable search listing widget shown on the search page (e.g. name, views, etc). The end date is stored as an integer value that represents the milliseconds since the UNIX 1970 epoch, rather than the Firebase timestamp and is automatically converted between them by it.

objectID	"f8LxD00knidxjer9u02h"
Name	"Logitech G710+ Keyboard"
HostID	"XQTDfgB94Nfw2jle4XtCa2VMwMj2"
path	"Listings/f8LxD00knidxjer9u02h"
Tags	["keyboard", "tech", "expensive", "electronics"] 4 items
EndDate	1714312219233
PrimaryImage	"https://firebasestorage.googleapis.com/v0/b/raffl-2ec83.appspot.com/o/images%2F1713707352495?alt=media&token=54d6ccc4-9d72-43d2-887e-7fc5b043f430"
Views	17
TicketsSold	450
UsersInterested	2
TicketPrice	3

Figure 73 - Algolia Listing Example

All code written was backed up to GitHub, which I primarily used for its version history, and as a generic safety net to keep data safely backed up and accessible from any other machine. This also provided the ability to roll-back changes and identify where bugs occurred, with great issue tracking features. Whilst this is currently a solo developer project, if this changes in future builds, it opens the opportunity for the many collaborative benefits provided by GitHub.

Ongoing Listing Screens

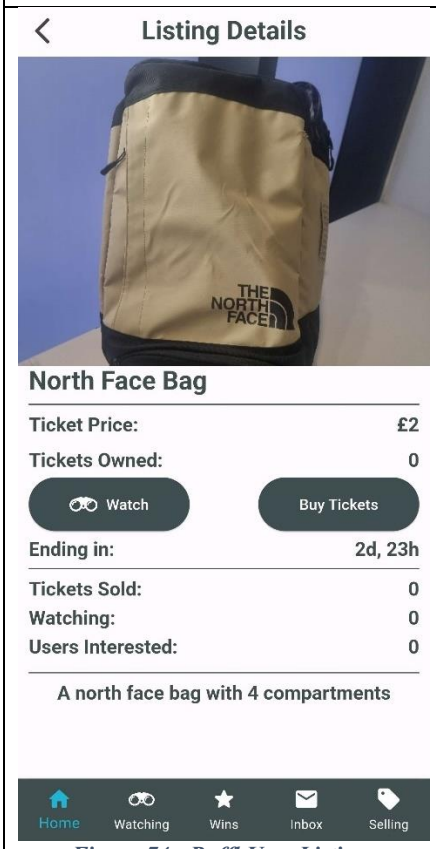


Figure 74 - Raffl User Listings

The final screens for ongoing listings are very similar to the ones seen in Figma. Minor changes include changing the watch logo to the navbar equivalent for consistency and changing the image to the 4:3 standard to fall in line with the rest of the app. Unfortunately, due to a bug with the navigation bar, this is not as responsive as I originally intended, and this will be covered further in the testing section. When a listing ends, this page automatically refreshes to ensure users are not stuck on the original version of the page. The timestamp uses the 'CustomTimerWidget' seen earlier, which updates the page in real-time.'

Figure 75 - Raffl Host Listings

Upon initialisation of these screens, various details about the item are recovered, such as how many items have been sold, how many users are watching said item and so on. As users update these statistics, i.e. buying a new ticket, these are updated on the database, but to avoid unnecessary requests, values such as how many users are watching are locally updated. While this may be out-of-date information, unnecessary queries have been deemed too expensive to be worth the benefit.

When users go to buy a ticket, an alert dialogue pops up that will inform them of how many credits they have and have an input box allowing them to buy as many tickets as they want. If users do not have enough credits, they are informed of how much they need. In this example, a user has one credit, and the ticket price is two credits. A user tries to buy two tickets, totalling four credits but needs three more, so they are informed as much by an error message. This ensures that the mistake is clear to the user, and they can easily recover from it, adhering to requirement #33.

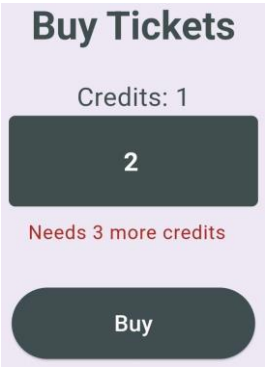


Figure 76 - Buy Dialog

When a user purchases a ticket, Raffl uses a Firebase transaction to ensure that this action is successful. Transactions are ACID-compliant, which implies many different properties. One of these is that all mutations to the document are atomic, so if one of these mutations has

```
buyTickets(String documentID, int ticketAmount, int ticketPrice) async {
  String uid = FirebaseAuth.instance.currentUser!.uid;
  //We make the transaction atomic so it is all or nothing
  return FirebaseFirestore.instance.runTransaction((transaction) async {
    final userDataRepository = Get.put(UserDataRepository());
    bool transactionSuccess = await
    userDataRepository.subtractCredits(transaction, (ticketAmount * ticketPrice));
    if(transactionSuccess){
      int ticketNum = await getTickets(documentID);
      if(ticketNum != 0){
        updateTicketNum(transaction, documentID, ticketAmount, uid);
      }
      else{
        createTicketNum(transaction, documentID, ticketAmount, uid);
      }
      updateTicketsSold(transaction, documentID, ticketAmount);
      return true;
    }else{
      print("Transaction unsuccessful");
    }
    return false;
  });
}
```

Figure 77 - Ticket Purchase Transaction

occurred, then all of the ones specified by the transaction have. Otherwise, none of them have. In the context of this transaction, if the user tries to buy an item and their credits are successfully reduced, then they are guaranteed to have bought a ticket. Vice versa, if the user's credits have not successfully been reduced due to any number of possible errors, then the user will not acquire their tickets and will instead get an error message. The transaction first attempts to subtract credits from a user, and then after this is successful, it either creates or updates the user's ticket count based on whether they already have tickets. Finally, the 'ticketsSold' field is updated to indicate tickets have been sold. If any one of these three steps fail, then the transaction will be aborted, and none of the actions will be completed.

Completed Listing Screens

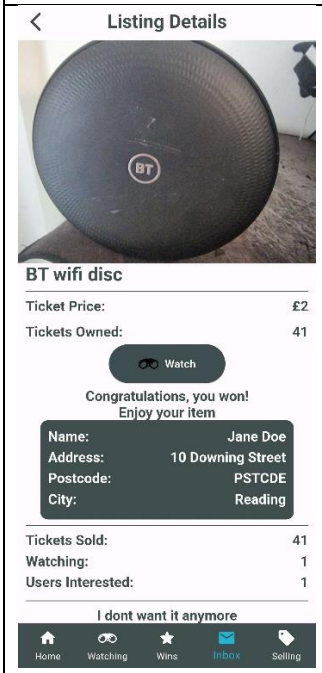


Figure 78 - Raffl User Won

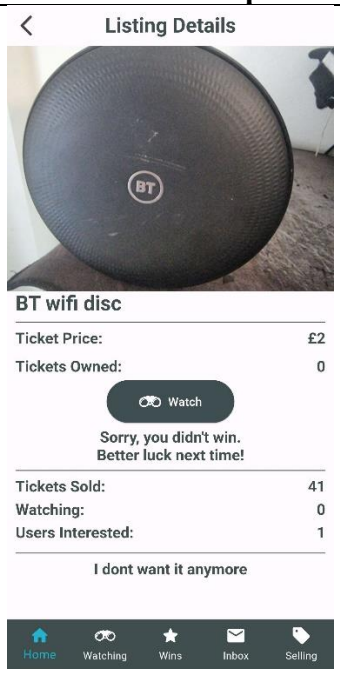


Figure 79 - Raffl User Lost

Figure 80 – Raffl Host Sold

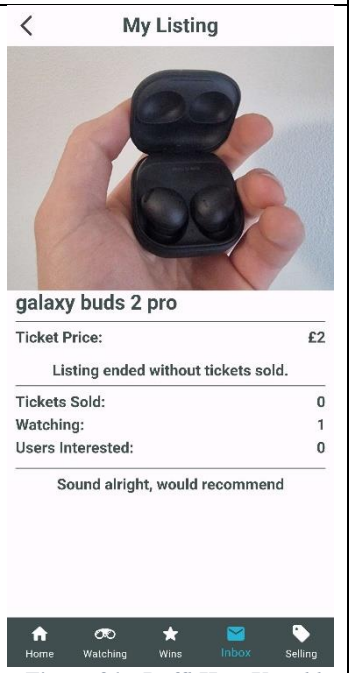


Figure 81 - Raffl Host Unsold

The alternate screens for our completed listings are very similar to the Figma designs shown earlier. Like before, you can see the interactivity with the 'watch' button, as it is highlighted

black for users who have selected it. One big difference now, however, is that the description is persistent across listing details for both consistency and for users who may still be interested in reviewing it at that stage. All these screens are also run from the same ‘view_listings_page’ dart file as there is so much shared content, it was more efficient to just update small parts of the page and lock parts of the page under certain circumstances (i.e. no buy button for hosts when the listing is ongoing, or any user when the listing has ended).

The solution for winners and sellers who have sold their items is a page that slowly updates. After a user wins, the host sees a message about ‘awaiting user shipping details’ until the user adds them with a new ‘add address’ button on their respective screen. After receiving an address, the host can mark the item as shipped and has the option to add the courier name and tracking number if applicable. When this happens, the winner can see the updated shipping details, if applicable, and be informed that the item is currently being shipped. They then have the choice to mark the listing as received for when it arrives, upon which credits generated on this listing will be released to the host, and finally, the page will update to inform the host.

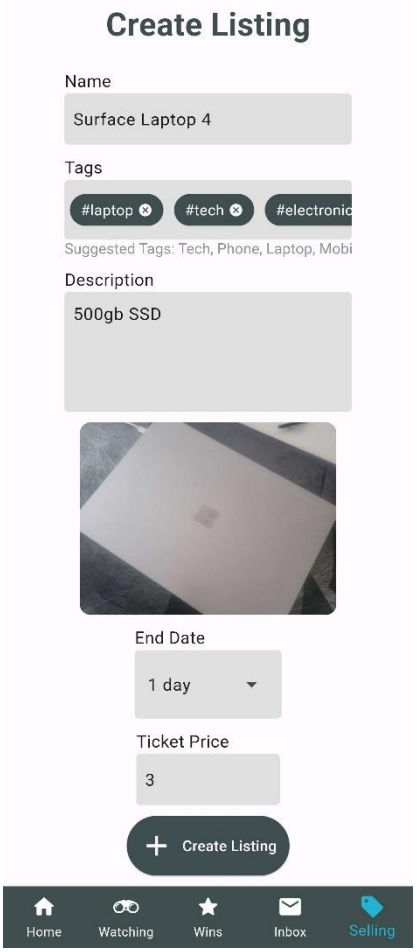
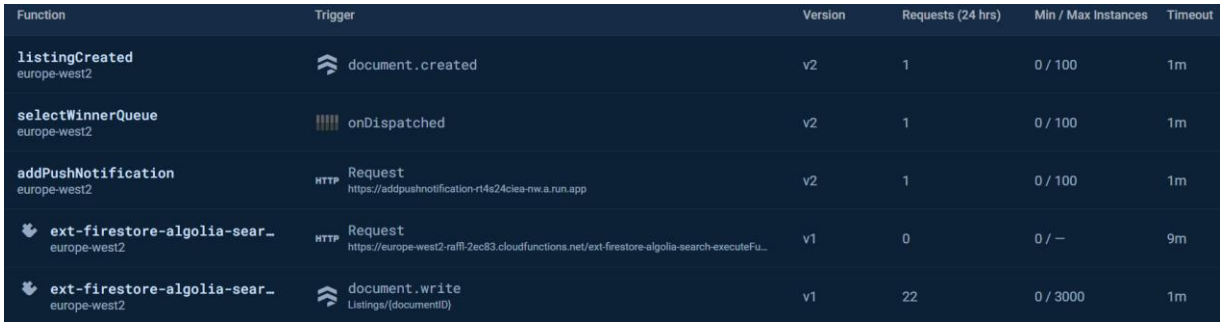


Figure 82 - Raffl Create Listing

This is the create listing page for Raffl. It is relatively different from the Figma wireframe and has had several changes. For one, the tags section has been more understood and acts as a scrollable list of tags that you can optionally add. Below these tags is a scrollable list of tag suggestions so users have an idea of what tags they could add to their items. The date selection is also different, where now, instead of a random calendar date, the listing ends in a set number of days. This is to enforce simplicity whilst also closely matching similar apps like eBay.
The image widget here, starts off like the stand in shown in the Figma wireframe. This is just an ‘add image’ button, and then switches to the image the user has uploaded, after they upload it. All fields other than the tags are mandatory, including the image. This enforces consistency across listings, and also stops users from making mistakes, such as accidentally not including a description.
Upon adding all the details, the active user can press the ‘create listing’ button. This then creates a listing in the Firebase backend with these details, which kicks off some cloud functions (covered later) and then refreshes us with the new ‘selling’ screen, which should include the newly listed item.



Function	Trigger	Version	Requests (24 hrs)	Min / Max Instances	Timeout
listingCreated europe-west2	document.created	v2	1	0 / 100	1m
selectWinnerQueue europe-west2	onDispatched	v2	1	0 / 100	1m
addPushNotification europe-west2	HTTP Request https://addpushnotification-rt4s24ciea-nw.a.run.app	v2	1	0 / 100	1m
ext-firestore-algolia-sear… europe-west2	HTTP Request https://europe-west2-raffl-2ec83.cloudfunctions.net/ext-firestore-algolia-search-executeFu…	v1	0	0 / –	9m
ext-firestore-algolia-sear… europe-west2	document.write Listings/{documentID}	v1	22	0 / 3000	1m

Figure 83 - Firebase Cloud Functions

Figure 82 shows all the cloud functions used by Raffl. The bottom 2 are outliers for Algolia, which allows Algolia to pull directly from the Firestore database whenever a listing is updated and automatically generated by installing the relevant Algolia extension and configuring the settings appropriately. The rest of the functions here were manually created using TypeScript before being pushed to the Raffl project on Firestore.

First off, we have the 'addPushNotification' function. When this function is invoked, it takes an input containing the user ID along with details such as the image, listing ID, name and description tied to this notification. After receiving this information, a notification item will be created to display on Raffl's inbox page through Firestore. After this, Firestore is queried for the notification token, and if it exists, a push notification is sent to the corresponding user, who will end up on their device.

```
const userDoc = await userDataDocument.get();
const notificationToken = userDoc.data()?.NotificationToken;
if(notificationToken){
    console.log("Preparing push notification");
    const pushNotification = {
        notification: {
            title: listingName,
            body: description,
        },
        apns: { //Apple handles images differently
            payload: {
                aps: {
                    alert: {
                        title: listingName,
                        body: description,
                    },
                    'mutable-content': 1,
                },
                'media-url': image,
            },
        },
        android: {
            notification: {
                imageUrl: image
            }
        },
        token: notificationToken
    };
    //Send push notification to device
    await admin.messaging().send(pushNotification);
    console.log("push notification sent");
}
```

Figure 84 - Push Notification Creation

```
exports.listingCreated = onDocumentCreated(
    {document: "Listings/{docId}",region: "europe-west2"}, async (event) => {

    const snapshot = event.data;
    if (!snapshot) {
        console.log("No data associated with the event");
        return;
    }
    console.log("Document Creation Noticed");
    const endDate = snapshot.data().EndDate;
    const endDateJS = endDate.toDate();
    const documentId = snapshot.id;
    const functionName = "selectWinnerQueue";
    const queue = getFunctions().taskQueue(functionName);
    const targetUri = await getFunctionUrl(functionName);
    const payload = documentId;

    const enqueuePromise = queue.enqueue({payload}, {
        uri: targetUri,
        scheduleTime: endDateJS,
    });
    await enqueuePromise;
    console.log("Payload sent");
    return null;
});
```

Figure 85 - Listing Created Function

First off, we have 'listingCreated,' which is a function that is automatically invoked any time a listing is created in Firebase. It does this with an 'onDocumentCreated' function call that watches the 'Listings' collection, which is triggered any time a document is created. Upon triggering, this function retrieves the end date for this listing, along with its document ID. It then queues the 'selectWinnerQueue' function for the timestamp specified by the end date, which means when this date is hit, the 'selectWinnerQueue' function will be invoked automatically.

```
exports.selectWinnerQueue = onTaskDispatched({region: "europe-west2"}, async (request: any) => {
    const listingId = request.data.payload;
    const db = admin.firestore();
    const listingDocument = db.collection('Listings').doc(listingId);
    const listingSnapshot = await listingDocument.get();
    //Create notification to tell user they have won:
    let listingName = "";
    let image = "";
    let hostID = "";
    //Recovers information from snapshot
    const listingData = listingSnapshot.data();
    if (listingData) {
        listingName = listingData.Name;
        image = listingData.PrimaryImage;
        hostID = listingData.HostID;
    } else {
        console.log('No data in document!');
    }
    const ticketsCollection = listingDocument.collection('Tickets');
    const ticketsSnapshot = await ticketsCollection.get();
    let nonWinningUsers = new Set<string>(); //Array of our users who are interested (watching or have tickets) but did not win

    if (ticketsSnapshot.empty) {
        console.log('User has no tickets');
        //Notification for listing with no tickets sold
        createNotification(db, hostID, "Sorry, no one bought any tickets.", image, listingId, listingName);
    } else { // Select winner for current auction
        let ticketArray: string[] = []; //Array to contain a user for every ticket they own (i.e. a user with 5 tickets in the array 5 times)

        // We add an individual value into our array for all tickets, to randomly select one after
        ticketsSnapshot.forEach((ticketDoc) => {
            const ticketData = ticketDoc.data();
            nonWinningUsers.add(ticketDoc.id);
            for (let i = 0; i < ticketData.TicketNum; i++) {
                ticketArray.push(ticketDoc.id);
            }

        });
        //Selects the winning ticket at random
        const winner = ticketArray[Math.floor(Math.random() * ticketArray.length)];
        await listingDocument.update({Winner: winner});
        nonWinningUsers.delete(winner); //Removes the winner from the nonWinningUsers set
        console.log("The winner is: ", winner);
        //Notification for winner
        createNotification(db, winner, "Congratulations, you have won, please add your address.", image, listingId, listingName);
        //Notification for host
        createNotification(db, hostID, "Congratulations, your item has a winner. Awaiting shipping address.", image, listingId, listingName);
    }
    // Informs all other interested participants that they didn't win
    for (let user of nonWinningUsers) {
        createNotification(db, user, "Sorry, you didn't win this item. Better luck next time!", image, listingId, listingName);
    }
});
```

Figure 86 - Winner Selection Algorithm

The select winner queue function is invoked the second a listing ends and, as such, is responsible for selecting the winner of the current listing whilst informing all interested parties. First off, if there are no tickets sold for this listing, it exits early after informing the user that no one bought any tickets for their item. Otherwise, the function continues, where all users are compiled into a list based on how many tickets they own to get a weighted average when selecting the winner. All users are also added to a set for non-winning users. When a random number is then selected based on the list length, the corresponding winner is removed from the array of unsuccessful participants, and the listing updates to declare this user as a winner. All interested parties are then given a notification of their status, where winners are told they have won, hosts are told their item has a winner, and the participants who did not win are given the unfortunate notification.

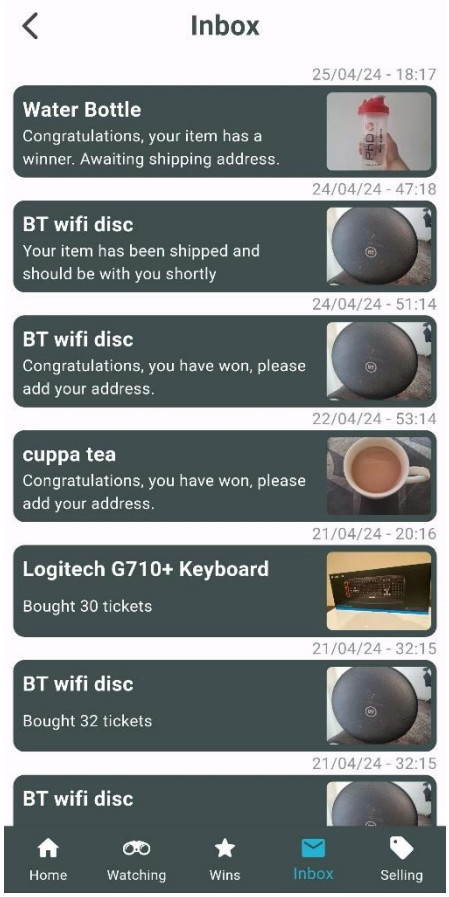


Figure 87 - Raffl Inbox

As the Figma design for the inbox screen was perceived well and all necessary information was incorporated, the final screen on Raffl was nearly a 1:1 replication, aside from adding curved corners to the internal image. On this page, the inbox results are ordered from ascending to descending date order, where the most relevant options to the user are shown at the top of the list. This is a scrollable list, and each inbox entry references an item, which can then be navigated to via tapping on the item. This fulfils requirement #32 in that the application is easy and efficient to navigate, as this is intuitive and adds a nice shortcut for the user to access the listing of interest.

Each inbox entry is a description of a status update at the specified timestamp, along with the name and image of the item in reference. Inbox entries are generated in 2 ways, cloud functions, such as when users win, and directly from the app, for non-push notification inbox entries, such as 'bought X tickets.' These entries are generated with their ID as the timestamp, to make ordering much easier without fancy queries or local sorting.

```
const currentTimeInMilliseconds = Date.now().toString();
userDataDocument.collection('Notifications')
.doc(currentTimeInMilliseconds).set(notificationEntry).then(() => {
    console.log('New notification successfully written!');
}).catch((error) => {
    console.error('Error writing new notification: ', error);
});
```

Figure 88 - Inbox/Notification Firebase Creation

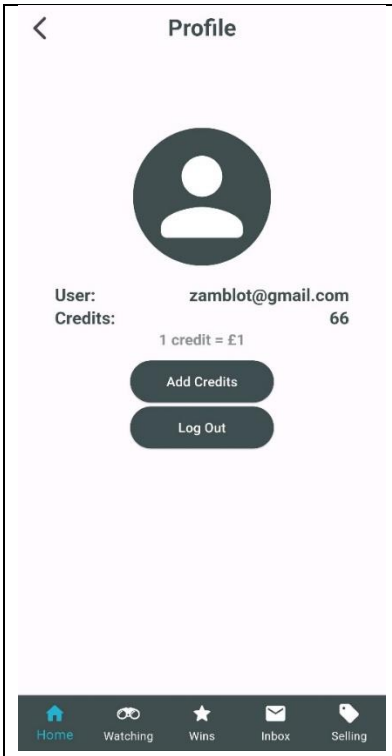


Figure 89 - Raffl Profile

The profile page is the last page covered on Raffl and is arguably the most simple page. This is very similar to the initial Figma wireframe design, as there is not much content to put here.

As credits were a necessary placeholder due to real monetary transactions not being included in this build of Raffl, I needed an extra page to host them. Therefore, in a potential production release of Raffl, this screen would likely not exist, or at the very least, it would be substantially different. This page does, however, have the side goal of giving users the option to sign out, which is necessary for switching between accounts and would have to have been incorporated somewhere anyway.

Upon clicking 'add credits,' an alert dialog, much like the one for buying tickets on the 'view listings' page for unfinished listings, will appear. This dialogue gives users to add a positive number of credits to their account balance, where it will be promptly updated in real time thanks to the use of Flutters listenable values.

7 - Testing

This chapter details the testing of Raffl. This includes testing all requirements, which is done manually and shown with proof, before giving the app to users for some less formal testing and to get hands-on insight into what possible consumers think about the app.

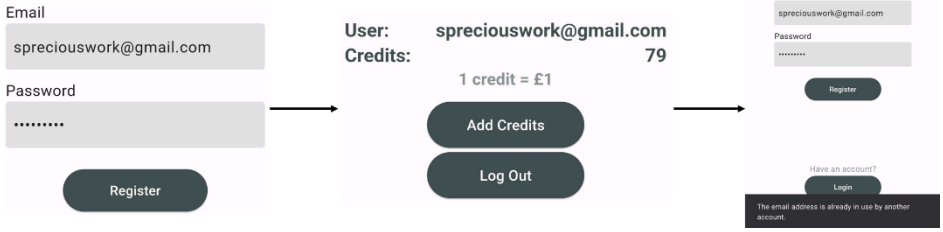
7.1 – Requirements Testing

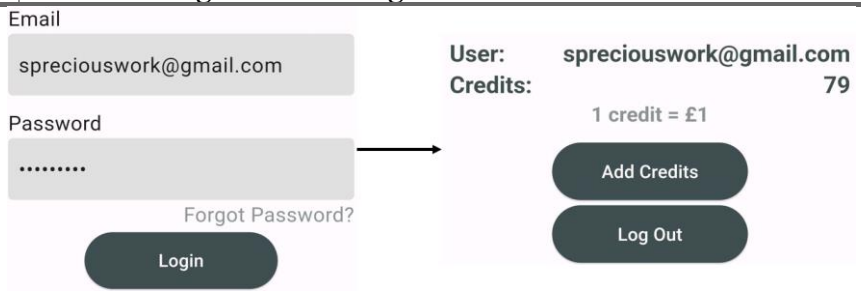
Here, the culmination of all requirements specified in section 4 is tested, and photographic evidence of success is provided if applicable.

7.1.1 – Essential Requirements

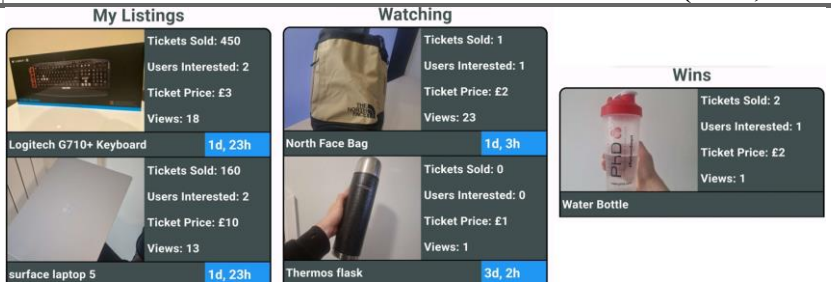
Requirement format:

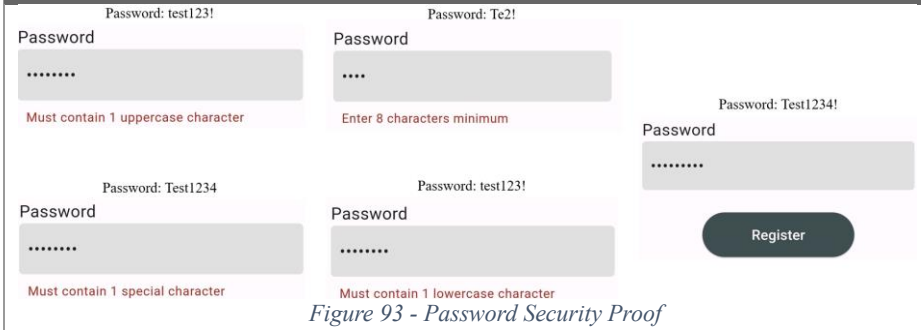
ID	Title	
Proof/Explanation – Proof of requirements working or an explanation as to why they're not working		**Status –** Pass/Fail

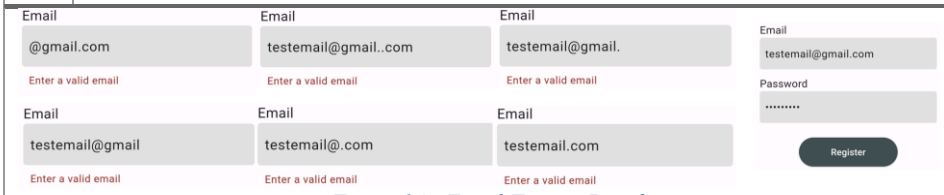
#1	Users can register with a unique email address	
 *Figure 90 – Unique Email Proof*		**Pass**

#2	Users can sign into existing accounts with email	
 *Figure 91 - Email Sign-in Proof*		**Pass**

#3	App works on Android and IOS	
As per the validation explanation, I do not have the tools to test IOS, but I can confirm here that no android specific code was written, and the app code was entirely written through Flutter, so this requirement should pass.		**Pass**

#4	Users can differentiate between items of interest (wins, selling, etc…)	
	 *Figure 92 - Proof of Item Differentiation*	**Pass**

#5	Password is secure	
	 *Figure 93 - Password Security Proof*	**Pass**

#6	Email is correct format	
	 *Figure 94 - Email Format Proof*	**Pass**

#7	Users have unique UID	
	29VvmFqR8uW7CCBCyG7YB3zyk933 4Wdow9HpFQOXel4KHq0fBXTsR3O2 UspbXJ5vD4bZAxylBCo7fJ5oaHU2 WGNF5OK0nbh1BKRHneDDUcZFVz82 XQTDfgB94Nfw2jIe4XtCa2VMwMj2 kHym7GwagdbP0JUqJCkWFUvxZ323 oBpxOsi7cLc6WLDGR65Ad73lr5n1 *Figure 95 - Unique UID Proof*	User IDs are all unique on the backend and generated through Firebase, where uniqueness is guaranteed.	**Pass**

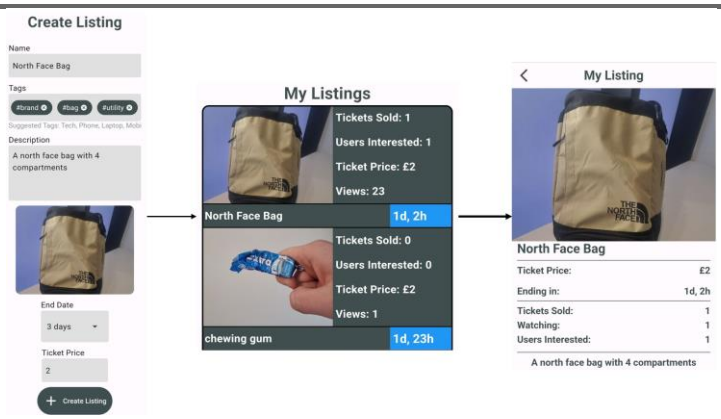
#8	Users can list own items	
		**Pass**

Figure 96 - Listing Creation Proof

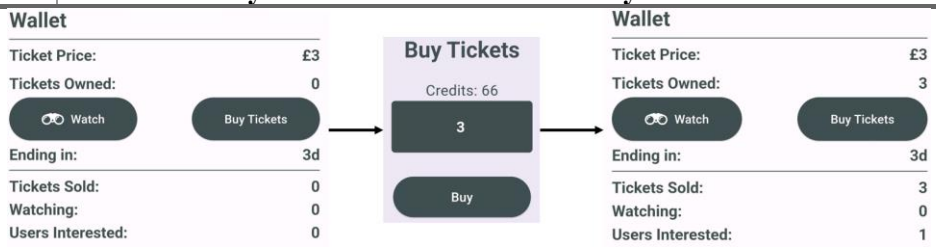
#9	Users can buy raffle tickets for items they do not own	
		**Pass**

Figure 97 - Raffle Ticket Purchase Proof

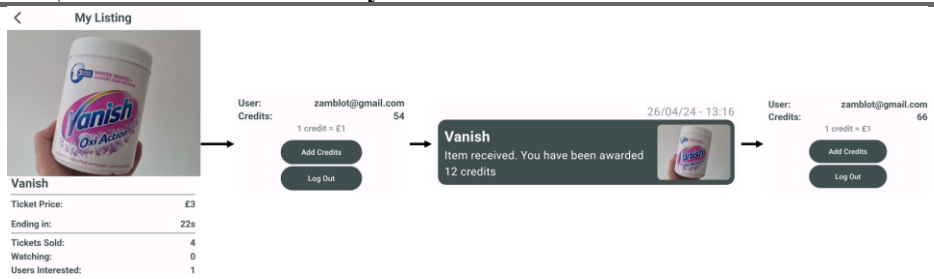
#10	Users receive currency for sold items	
		**Pass**

Figure 98 - Currency Received Proof

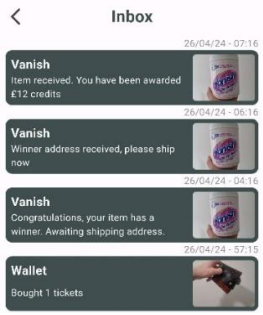
#11	Users have an inbox with status updates	
		**Pass**

Figure 99 - Inbox Proof

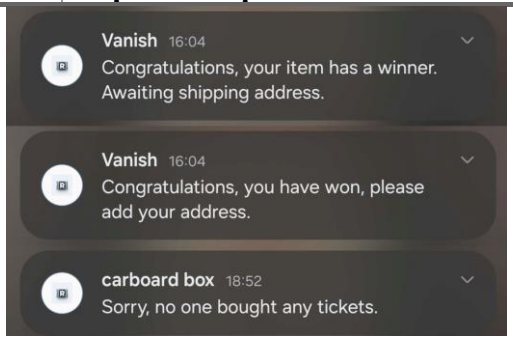
#12	Users get push notifications for more important updates	
		**Pass**

Figure 100 - Notification Proof

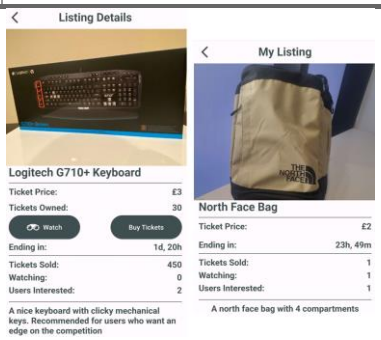
#13	Items should be clearly differentiated between own and others	
		Pass

Figure 101 - Page Differentiation Proof

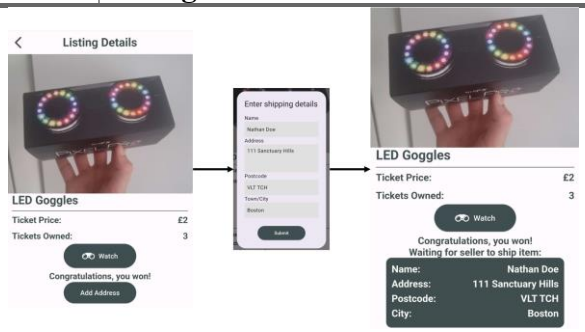
#14	Winners of items can add address to listing	
		Pass

Figure 102 - Address Submission Proof

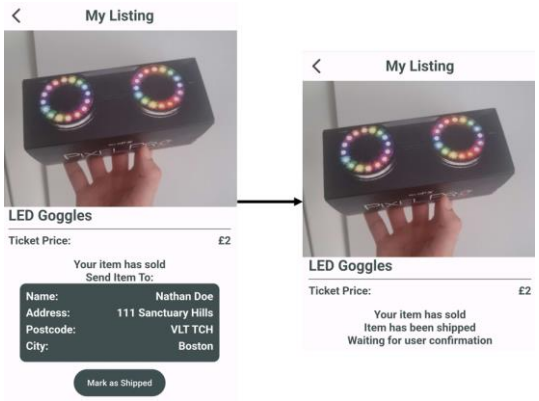
#15	Sellers can mark items as shipped	
		Pass

Figure 103 - Mark Shipped Proof

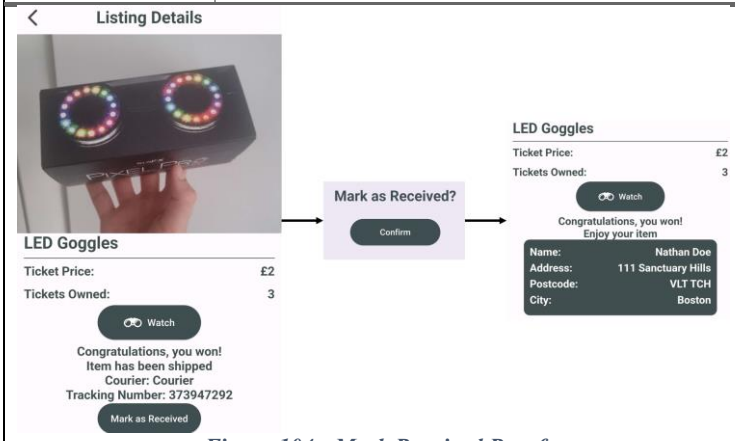
#16	Winners of items can add address to listing	
		Pass

Figure 104 - Mark Received Proof

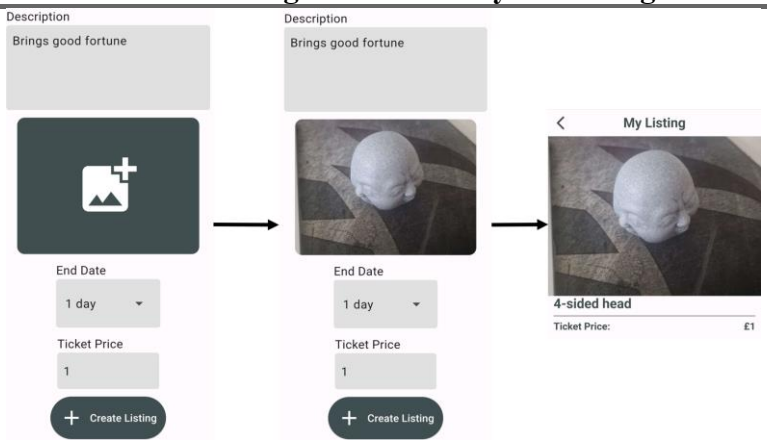
#17	Users can add an image for items they are selling	
		Pass

Figure 105 - Image Addition Proof

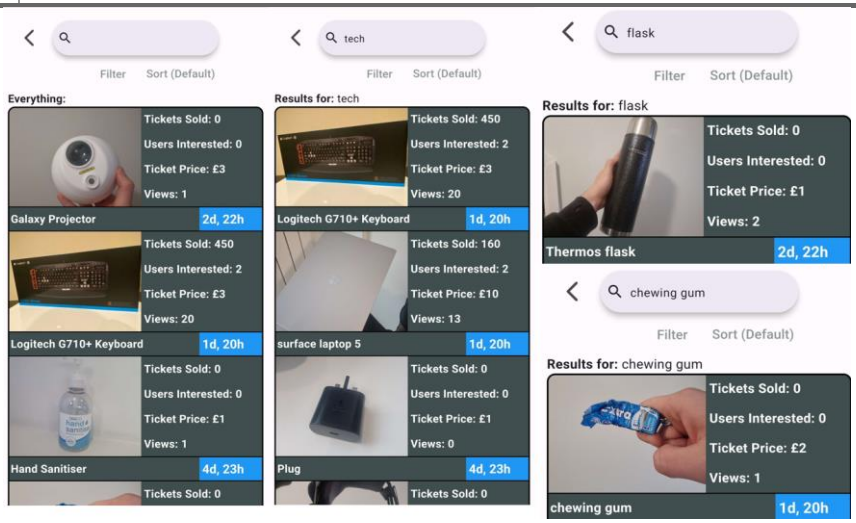
#18	Users can search for items	
		**Pass**

Figure 106 - Item Searching Proof

7.1.2 – Desirable Requirements

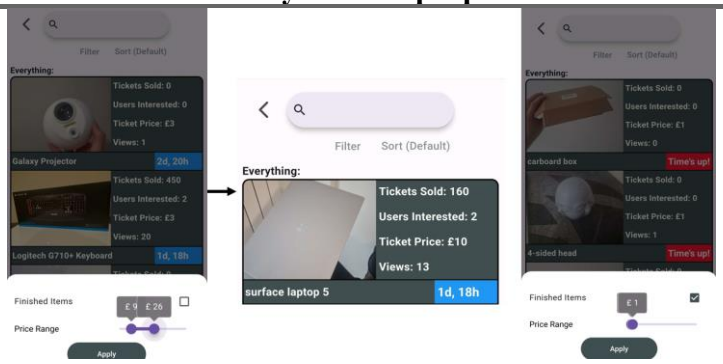
#19	Users can filter items by various properties	
		**Pass**

Figure 107 - Filter Proof

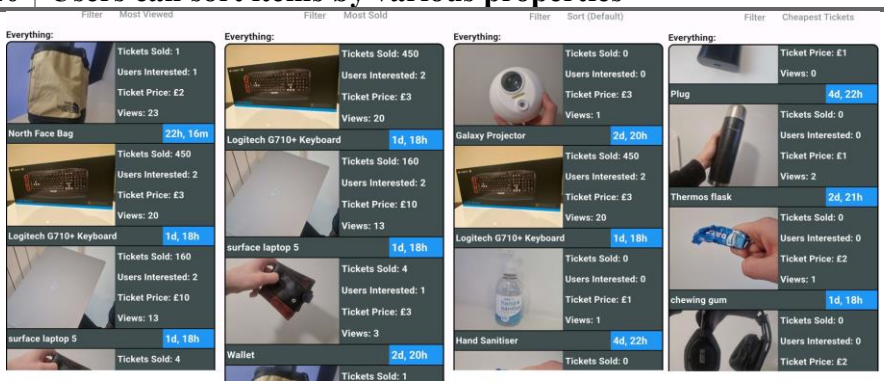
#20	Users can sort items by various properties	
		**Pass**

Figure 108 - Sort Proof

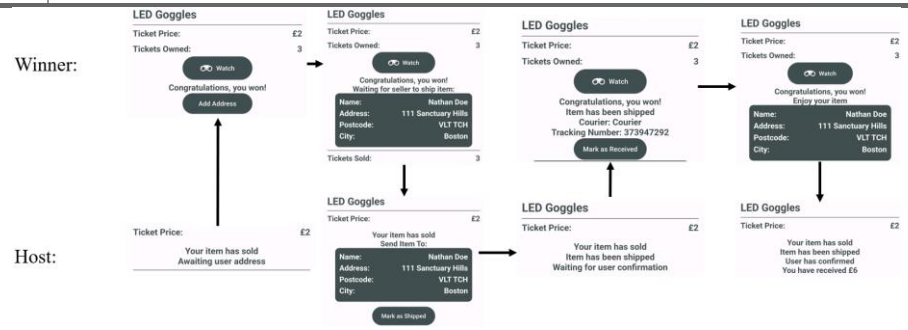
#21	Relevant users can see status of items that have been sold	
		**Pass**

Figure 109 - Status Lifecycle Proof

#22	The app is accessible where possible	
This is subjective, so it will be verified with user feedback rather than proof later.		**N/A**

#23	Dangerous actions are obvious	
		**Pass**

Figure 110 - Dangerous Proof

#24	Design is minimalistic	
This is subjective, so it will be verified with user feedback rather than proof later.		**N/A**

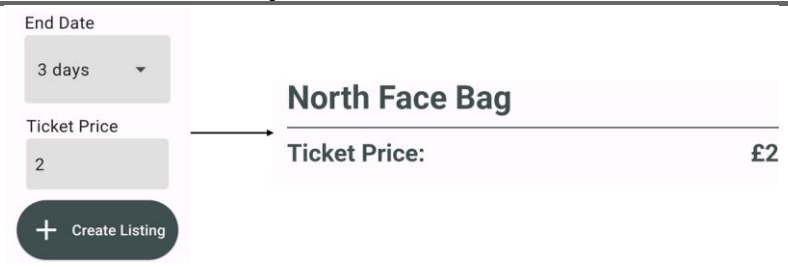
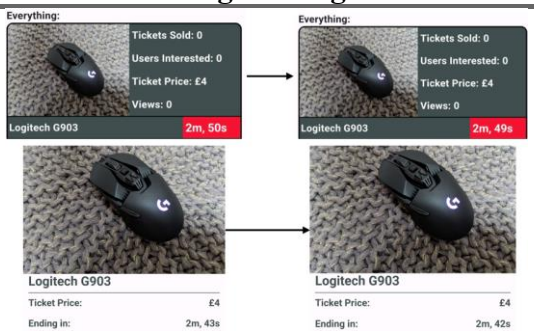
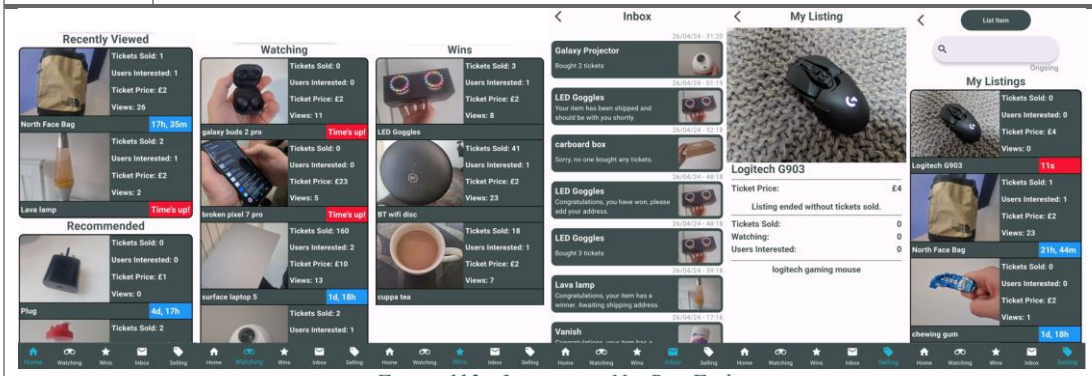
#25	Users can set ticket price	
		**Pass**

Figure 111 - Price Set Proof

#26	Design is consistent	
This is subjective, so it will be verified with user feedback rather than proof later.		**N/A**

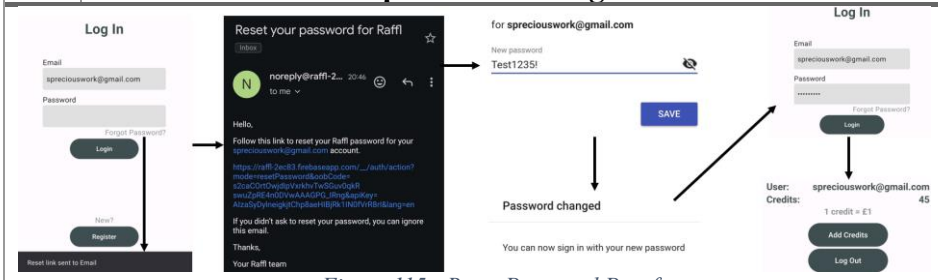
#27	Live countdown for listings ending	
 *Figure 112 - Live Countdown Proof*		**Pass**

#28	App is intuitive	
This is subjective, so it will be verified with user feedback rather than proof later.		**N/A**

#29	Navbar is interactive	
 *Figure 113 - Interactive NavBar Fail* Unfortunately, this objective was not passed. This is because while the navbar is interactive while switching between tasks, it does not respond when switching to other screens without using it. Fixing this would require a fair bit more work than expected and did not fall within time restraints.		**Fail**

#30	App has user-specific recommendations

 *Figure 114 - Recommendations Proof*	After buying tickets for multiple tech related listings, the recommendations displayed a strong bias towards the tech and similarly related items than the average listing. Note that the finished listings are included here, as in this debug build of Raffl, I determined it would help for the purpose of testing features like this, but it could easily be changed in a production release with a single line of code.	**Pass**

#31	Users can reset their password if forgotten	
 *Figure 115 - Reset Password Proof*		**Pass**

#32	Application is easy and efficient to navigate	
This is subjective, so it will be verified with user feedback rather than proof later.		**N/A**

#33	Mistakes are clear for users	
This is subjective, so it will be verified with user feedback rather than proof later.		**N/A**

7.1.3 – Possible Requirements

#34	Users can see analytics of listings	
 *Figure 116 - Analytics Proof*		**Pass**

#35	Users can add tracking information to listings	Pass

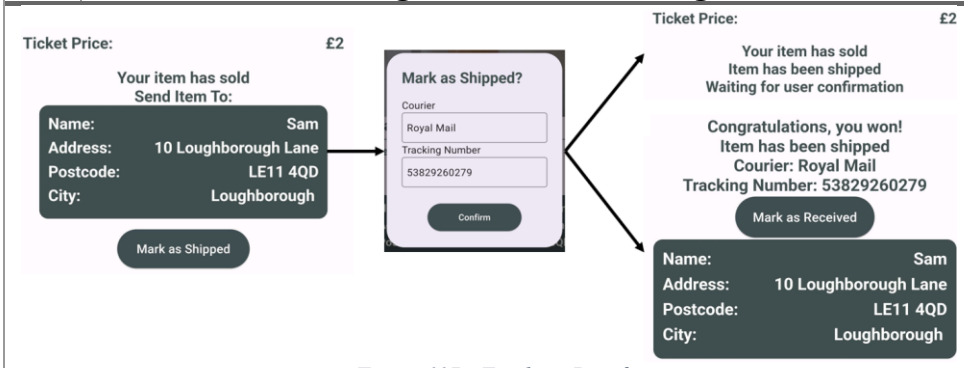


Figure 117 - Tracking Proof

#36	Users can view sold items	Pass

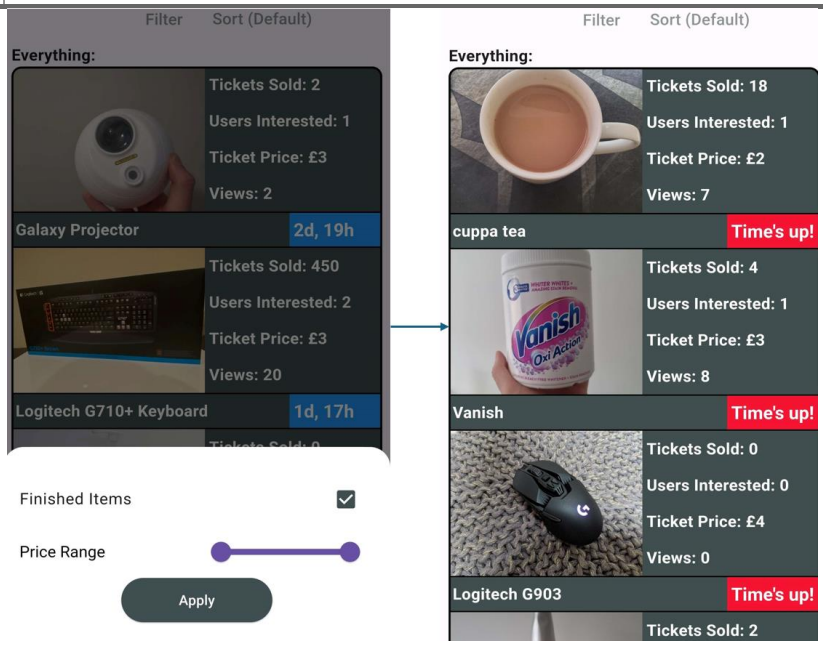


Figure 118 – Sold Filter Proof

#37	Search functionality is broad and doesn't rely on exact inputs	Pass

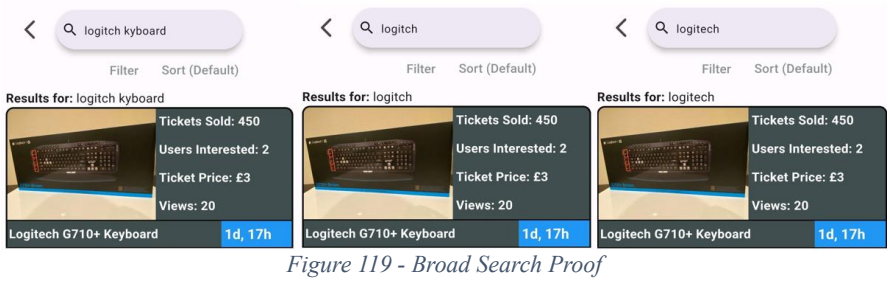
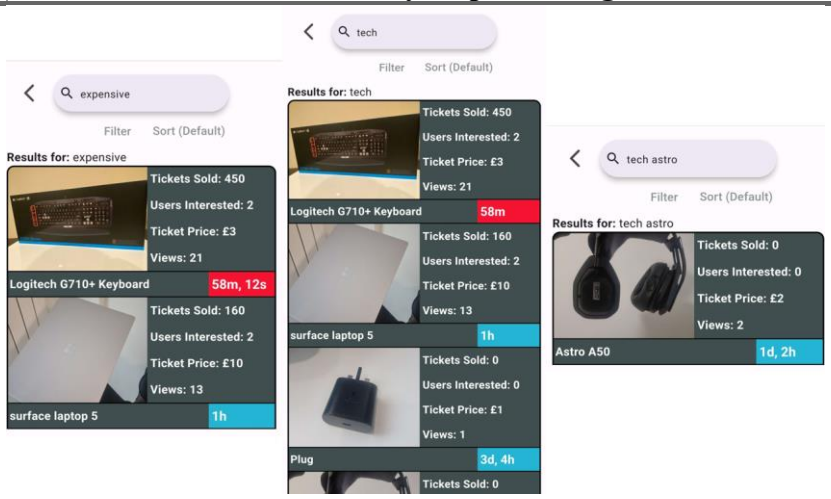
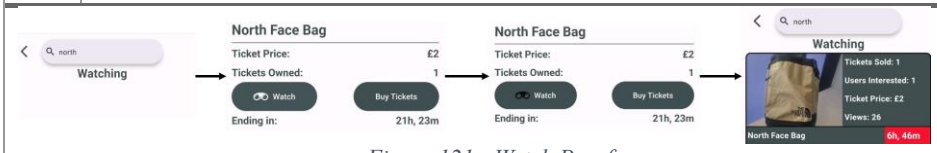
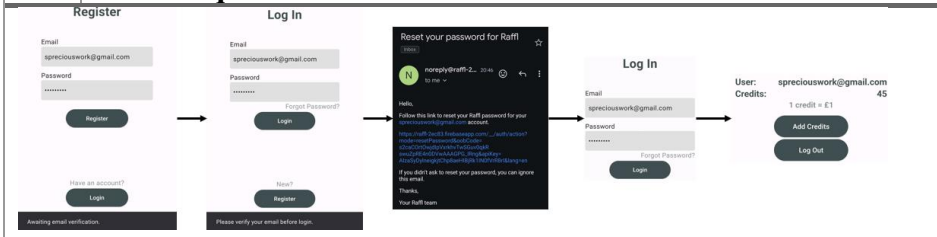


Figure 119 - Broad Search Proof

#38	Users can search for items by respective tags	
 *Figure 120 - Tag Search Proof* In the third search 'tech astro,' there is no tag for 'Astro' but, the combination of the tag tech, and the name astro means the smart search offered by Algolia puts this information together and figures this out.		**Pass**

#39	Help page is available in app	
This requirement was not met, as it did not fall within the strict time restrictions.		**Fail**

#40	Users can watch items	
 *Figure 121 - Watch Proof*		**Pass**

#41	Users require email confirmation	
 *Figure 122 - Email Confirmation Proof*		**Pass**

#42	Users can directly purchase items	
This requirement was not met, as it was not possible to implement Raffl's simple design without overcomplicating it.		**Fail**

#43	Recommendation system is driven by machine learning	
This requirement was not met, as both time and budget restrictions did not allow for this.		**Fail**

#44	Users can log in with phone number	
This requirement was not met due to strict time constraints		**Fail**

7.2 – User Testing

This section seeks to document several interviews I conducted in the process of user testing on my application. The feedback gained from the user testing process is crucial for enhancing and refining the app's design and functionality. Firstly, I will detail the methodology used to extract comprehensive feedback from the participants. Secondly, I will translate this feedback into a list of recommendations, then draw these together in summary and conclude on future directions.

Methodology

The method of testing I chose was to interview a diverse set of candidates, conducting interviews under controlled conditions and with direction observation and providing a unique build of Raffl that enabled effective feedback to be made on the app still under development. This included disabling email verification and reducing listing durations (from days to minutes) to facilitate a meaningful and thorough exploration of the app's capabilities. The idea is to allow participants to interact with the app as if it were real-world usage in a condensed timeframe so that realistic conclusions can be drawn on the app's performance and features.

In addition, to extract as much useful feedback as possible from participants, it was important to formulate the questions posed to them in a way that enabled them to share confidently their own experiences and give sufficient detail to identify areas for improvement. To achieve this, I made sure to craft my questions in an open-ended way and employed active listening during the interviews to maximise feedback from the participants. Also, I referred to several issues I had already raised in the requirements chapter, such as unfinished requirements, in the questions to direct feedback on areas I knew I might be lacking.

User Experience and Feedback

Before giving users access to Raffl, all three participants were asked if they were interested in raffles and if they had previous experience with them. This is to gauge any preexisting biases or general perceptions towards raffle systems, as well as to gain some historical context to each participant's previous experiences with them. Consequently, in response, all participants were quick to share the positive associations they had with raffles, noting especially the fondness of experiences from childhood with them and the excitement derived from the lower-risk but exceptionally rewarding ticket draws. Another response I noted was that raffles were often most associated with charity events, and this lens often gave them a positive reputation.

Having conducted the initial inquiries, I now introduced each candidate to the app itself, along with a brief explanation of a few actions they could attempt to do, such as 'list an item' or 'buy tickets for an item,' so participants would have an idea of where to start. Following this, the participant's initial interactions with the app immediately led to a variety of suggestions and feedback on features:

- **Candidate 1:** Recommended implementing content moderation tools for fraud detection. Also, expressed the desire to have seller ratings and product condition labels.

- **Candidate 2:** Suggested providing additional support mechanisms like FAQs and live chat customer support to provide immediate assistance to users. Also, I suggested adding account login integration, such as with Google or social media, to make it easier to register and log in.
- **Candidate 3:** Echoed the suggestion of having ratings of sellers and requested features such as in-app tracking and allowing multiple image uploads for a listing.

To get a sense of how users felt about the app, they were then asked how they felt Raffl compared to similar e-commerce sites, such as eBay and Facebook Marketplace. This question received a myriad of different responses:

- **Candidate 1:** Suggest the app was intuitive and easy to use. Preferred the accountability of eBay with seller reviews.
- **Candidate 2:** Liked the convenience of raffles over auctions, as they didn't have to actively watch an item, along with how easy it was to list items by comparison.
- **Candidate 3:** Enjoyed the excitement of raffles, stating it felt 'more fun than transactional' unlike traditional e-commerce applications

Finally, I asked users questions based on the four requirements that required user feedback to get a sense of whether they had been achieved:

- **#24 – Is the design was minimalistic?** – All participants agreed that the design was minimalistic, with 1 candidate elaborating on their reasoning by stating the minimal colours and lack of clutter confirmed this.
- **#26 – Is the design is consistent?** - This response also met a resounding yes from all candidates without further elaboration.
- **#32 – Is the app easy and efficient to navigate?** – All candidates agreed, with one user mentioning that they were able to perform all tasks with relative ease and no confusion.
- **#33 – Are mistakes, such as bad password formatting, made clear by the app?** – Candidates confirmed that this was also true without further elaboration.

Feedback Analysis

By documenting and reviewing the outcomes of the interviews, I can draw four key themes for areas of improvement:

	Feedback	Recommendation for Development
1	**Content moderation and security:** address the need to handle explicit content and maintain a user-friendly environment.	Implementing content moderation and a fraud detection system to align with the need for a secure and trustworthy user environment.
2	**UI enhancements:** usability of watch function and improvements to the process of listing items.	Allow users to upload multiple photos per listing... transform watch button to toggle between 'watch' and 'unwatch' to ensure clarity and improve user control...

3	**User Support and Accessibility:** More accessible support and immediate assistance features are needed.	Add FAQs and live chat support.
4	**Seller accountability:** sellers should have more accountability for their actions.	Add seller reviews.

Conclusion and Future Directions

Overall, the interviews provided meaningful feedback, which can be actioned to improve user experience and further develop app functionality in preparation for full deployment. Future testing will continue to iterate on these improvements, ideally pushing Raffl towards becoming a popular choice for e-commerce and raffle-based platforms.

8 – Execution Timeline

This chapter documents the project management approach taken and the execution timeline created in developing Raffl. In particular, an explanation along with an evaluation is given for the adaptation of an agile framework to accommodate for the fact that the project is being completed by myself alone. Continuing from this, key milestones of the project are initially set out in a Gantt chart, enabling a more flexible and responsive project execution while moving from sprint to sprint. Finally, a detailed breakdown of sprints completed during the project is provided, summarising the main areas of focus for each sprint and conveying a holistic view of the entire development process.

8.1 – Agile

At the outset, Raffl was not a straightforward development project, and therefore, development required detailed planning to meet the deadline. This project was developed alongside various, sometimes conflicting, priorities. Subsequently, it was hard to predict the total development timeline accurately. To mediate this potential setback, I employed the agile methodology for its flexible, iterative development process, accommodating for changes in scope.

Some adjustments were made to the traditional agile model to make it suitable for this project. For one, agile is usually done in large teams, but as this involved a solo developer, a slightly different approach was taken. First, daily standups were cut out, as there were no other developers to catch up on progress with. This also meant there was no separate ‘product owner’ or ‘scrum master’ role, and instead, all responsibilities were compiled into one. Therefore, the responsibility of both project specification and development fell on a single individual.

A knock-on effect of the single developer structure with this project meant sprints could not accurately represent a fixed amount of work due to external responsibilities. These responsibilities came in the form of other projects, such as exams, and could take large chunks of development time away from the project. To account for this, certain sprints were longer; for example, the 3rd sprint is about 50% longer than previous sprints to account for a 2-week exam gap. Sprints were also on the longer end of what is typically expected, at around one month per sprint, as an attempt to reduce variation in overall development time, taking wider timeslots as averages.

First, a basic Gantt Chart was developed, which vaguely specified what would be accomplished over each sprint and how it would be done. These tasks were spread out until the project deadline and roughly encapsulated the end goals of the project. As each sprint began, more information came to light, which was used to update and flesh out the goal of the sprint. This was achieved by improving the details of the sprint whilst adding manual tests that would need to be passed to consider each individual goal a success. While writing individual unit tests would be preferable, as we could guarantee reliability and that sections are not broken throughout, the initial time investment to learn a testing framework and implement such functionality was too great.

Whilst the project had an initial vision along with a clear understanding of its target audience through extensive research, adjustments were made along the way. The iterative approach of agile meant that whilst there was a comprehensive list of requirements and a solid set of

initial designs, these would change over time and grow based on fresh insights. These new requirements and design changes would eventually be reflected by features in future sprints and ensured that the app wasn't limited by a rigid set of initial criteria proposed.

8.2 – Sprint Breakdown

This section gives a detailed breakdown of all sprints, where each subsection gives an explanation pre-sprint about the goals of said sprint and how they will be achieved, followed by the Gantt Chart for this individual sprint. When the sprint is finished, a small conclusion is then written up, which details how the sprint went, along with any tasks that remain unfinished and are pushed to the next sprint.

8.2.1 – Sprint Mapping

When mapping out the timeline of all sprints, the first thing to account for was how much time was available for development. Using this, along with external life events that are likely to cause disruption, I developed an early iteration of the Gantt Chart, seen in future sections, that detailed what was necessary to implement over certain time periods.

As the initial Gantt Chart is relatively long and can be seen in greater detail across future sprint sections, it is not included here. Instead, I will go over the time allocated to each sprint and explain why I have chosen to do it this way. As mentioned in the previous section, most sprints aimed for approximately one month of development time and are laid out as follows:

- Sprint 1 (October 25th – November 24th) – This sprint is given the average, month-long timeframe as there are no special occurrences. This sprint is mainly allocated for research.
- Sprint 2 (November 27th – December 22nd) – There is a brief gap from the last sprint to account for HCIs coursework. This sprint also ends early to avoid overlapping with Christmas. It is centred around initial design along with minor implementation
- Sprint 3 (December 26th – February 13th) – This is the longest sprint. January exams mean a large proportion of development time is lost over this month, so an extension is in place, so this sprint still produces valuable work. The first bulk of implementation is expected here.
- Sprint 4 (February 14th – March 14th) – Average sprint without many special occurrences, so standard timeframe. Focused on implementation.
- Sprint 5 (March 15th – April 11th) – Finished slightly early to give final sprint more time before deadline. Final implementation expected here.
- Sprint 6 (April 12th – April 30th) – Final sprint, tying up loose ends and finishing write-up

8.2.2 – Sprint 1

Sprint 1 prioritises research and has a focus on understanding what should come out of this project and how it can be accomplished. As this was not development based, there are no manual tests involved.

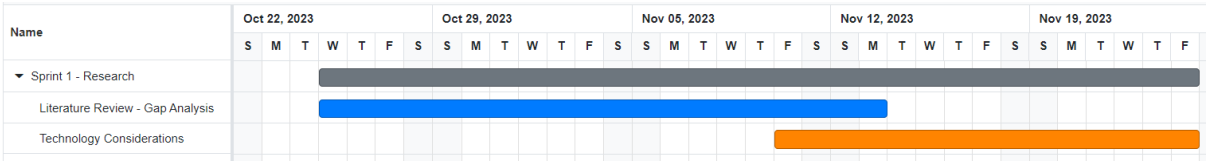


Figure 123 - Sprint 1 Chart

Overall, this sprint went well and was achieved according to plan. After this sprint, adjustments were made to both the literature review and the technology considerations involved, but the substantial bulk was achieved here.

8.2.3 – Sprint 2

Sprint 2 focuses on the design and requirements of Raffl, along with the early implementation of the features involved. As there is actual code here, the following tests are done:

1. User registration possible
2. User login possible
3. Logout via profile if possible

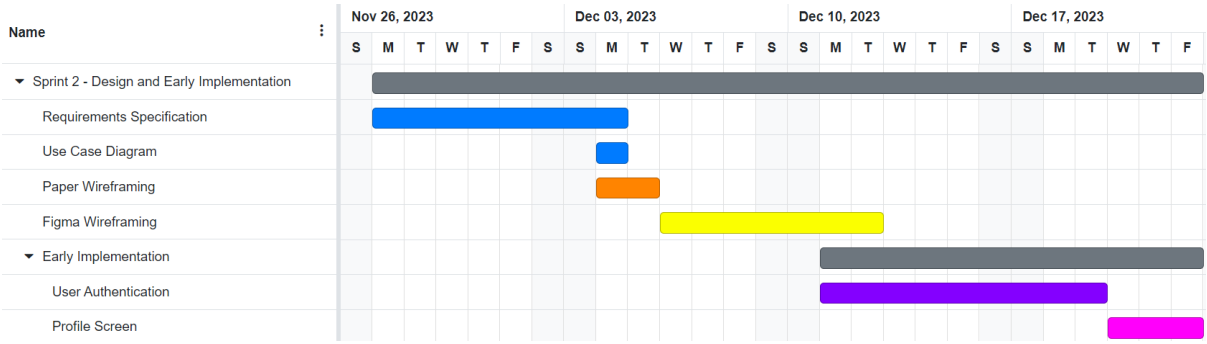


Figure 124 - Sprint 2 Chart

After drafting the requirements specification and making a use case diagram was done, the design sections became a priority. This required paper wireframing before moving on to more advanced Figma wireframing. Whilst I tried to initially capture as many aspects of both the requirements and design as possible, they are subject to change for future sprints if the need arises. An underestimation of getting set up with Flutter and learning the basics, paired with a slight overlap with HCI coursework, resulted in delays with the early implementation, so the criteria to pass the manual testing here was unsuccessful.

8.2.4 – Sprint 3

Sprint 3 is entirely based on implementation from a surface level, though this does not mean that there was no documentation or tweaks of the non-technical aspects. Whilst the coding criteria as defined by the Gantt Chart remained constant, minor tweaks to both the design and requirements were made throughout, when deemed necessary. The manual tests for this sprint are defined as follows:

1. User registration possible
2. User login possible

3. Logout via profile if possible
4. Credits incrementable from the profile page
5. Basic listing creation functionality exists
6. Search listing functionality exists
7. A separate accessible page for each listing exists
8. Ticket purchase functionality exists
9. Inbox/notifications are saved to the backend

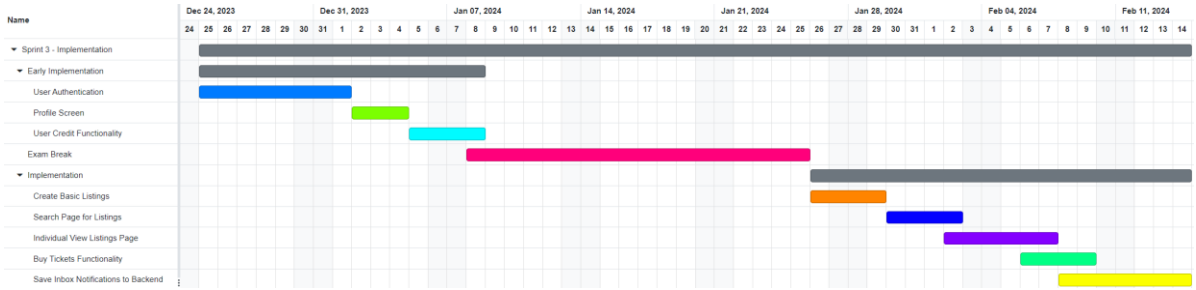


Figure 125 - Sprint 3 Chart

This sprint was relatively successful, where both the user authentication and profile screen were implemented before the exam break. Whilst the user credit functionality wasn't finished yet, this was quickly recovered within a day of the break ending, and progress was back on track. This continued through until February 14th, when all testing requirements for this sprint were successfully passed, and the sprint was deemed a success. During this time, changes were made to the requirements doc, where the full-text search implementation was added after I was unsatisfied with the results of the basic Firebase searching. The design was also updated, and the grid layout seen in the old home screen (Figure 30) was updated to the new one (Figure 31). This change was seen across all screens that had the old grid search layout, such as the search screen and wins screen.

8.2.5 – Sprint 4

In sprint 4, the focus on implementation is carried over from before, as there is still a lot to do here, and the requirements were in a satisfactory position. The full-text search requirement, realised in the previous sprint, was also added here along with Algolia, as I determined it to be the most suitable tool for the job at hand. I had the following mandatory testing requirements to consider the sprint a complete success:

1. The inbox page can be accessed
2. Winners are automatically selected upon listing ending
3. Statistics of listings can be viewed
4. Search results don't require complete accuracy (i.e. typos)
5. Push notifications are received for all participants involved in the transaction
6. Search results can be ordered and filtered
7. Functional selling page is accessible
8. Functionality to watch items
9. Functional watching page is accessible

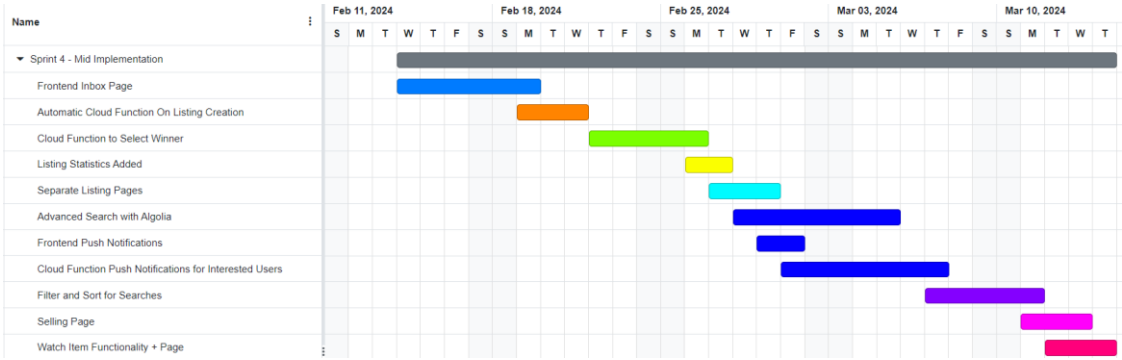


Figure 126 - Sprint 4 Chart

Sprint 4 started great but had some hiccups along the way. Implementation of the inbox page, along with a satisfactory design, was completed slightly ahead of schedule; however, due to bugs and time taken to learn typescript, the initial cloud functions took longer to develop than expected. Algolia was successfully implemented, and though it took a while to set up, it ended up working much better than Firebase for searching, as expected. Due to external factors, such as upcoming coursework, both the selling and watching page, along with the watch item functionality, were delayed until the next sprint. When attempting the testing requirements, tests 1-6 passed without a hitch, but as the functionality hadn't been implemented, requirements 7-9 failed and were pushed to the next sprint.

8.2.6 – Sprint 5

The aim of sprint 5 was to finalise the implementation of Raffl whilst also writing about said implementation in this report. It includes the three tasks that didn't make the cut last sprint, along with some new ones that ended up completing the app. The main implementation goals here are to implement the final pages and generally polish the design on all Raffl screens to make it more in line with the Figma wireframes. I also added a 'recently received' task here to make the home page more interesting and offer further utility to users. The tests to pass this sprint are as follows:

1. Functional selling page is accessible
2. Functionality to watch items
3. Functional watching page is accessible
4. Functional wins page is accessible
5. Responsive filtering on side pages works (e.g. instant search on wins page)
6. Screens are visually satisfying and, like Figma wireframes

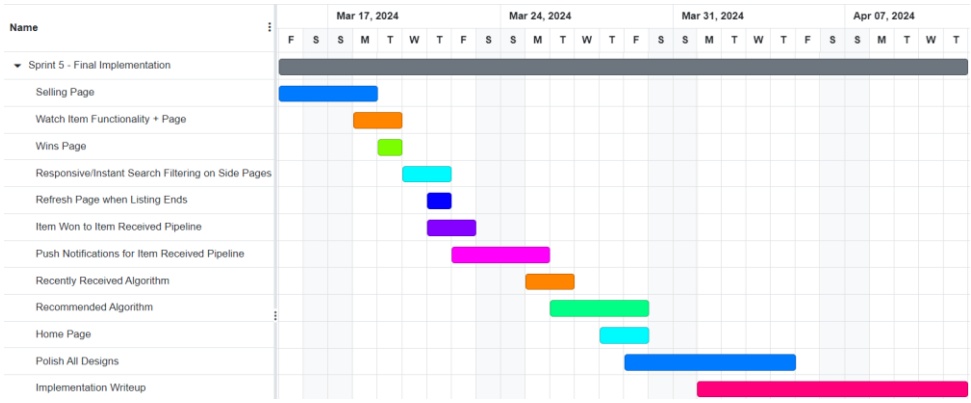


Figure 127 - Sprint 5 Chart

Sprint 5 was a very successful sprint, with all tasks completed within the timeframe, leaving ample time for testing Raffl, polishing the report, and completing any incomplete sections. Throughout this sprint, as the plan had been fully realised at this point, no further development was required on past sections of the report, and the requirements and design sections didn't have any changes. All requirements tests here have successfully passed, and by extension, all the application implementation is completed. Furthermore, the implementation has completely been written up at a level that I am satisfied with.

8.2.7 - Sprint 6

Sprint 6 is the shortest sprint yet and centres around the final phase of production, along with the completion of the write-up. In this section, I aim to retest all requirements to ensure nothing has broken along the way. After this, I aim to test the subjective requirements, mainly in terms of the UI design, rather than the usual short tests at the end of each sprint. I then plan to write the rest of this section, making sure all the information is well presented, before finally moving on to the conclusion.

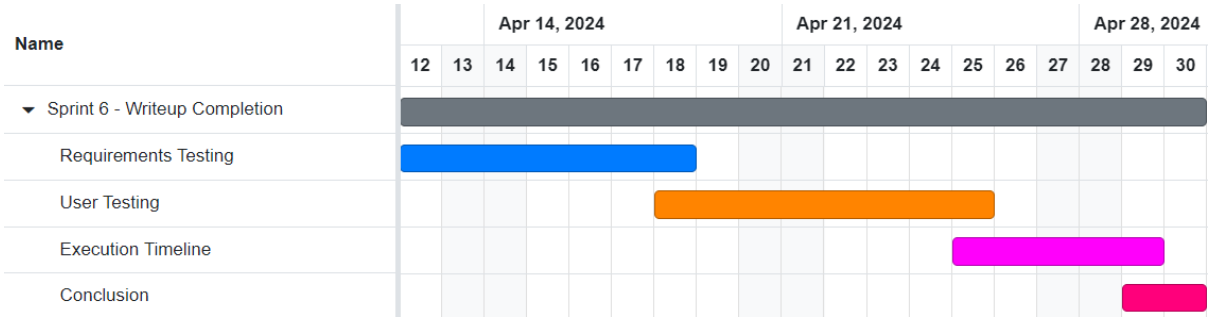


Figure 128 - Sprint 6 Chart

This section has gone well so far, and unlike previous sections, has not concluded yet. This is because the conclusion is done next and still needs some finishing touches. The requirements all tested well, as seen in the requirements section as most have passed, and the user feedback received has been invaluable for reflection and to consider with future projects.

8.2.8 – Sprint Conclusions

Overall, I believe the agile approach was the right methodology to follow here, as it got the product where it needed to be and allowed for numerous upgrades and changes that those rigid methodologies, such as waterfall, would not be able to capture. Delays and setbacks happened, as they always do, but in the end, there were no huge changes to the project throughout, nor did I have to heavily modify the specifications. Some sprints were, however, more challenging than others, as unexpected setbacks or code that took longer to complete than expected required working overtime to correct and get back on track.

9 – Conclusion

This chapter aims to provide a conclusion to this report, including a description of how to deploy Raffl if you choose, along with a reflection on the process and, finally, some changes I could make in future developments of Raffl.

9.1 – Deployment

This section details the deployment of Raffl on your own device for testing purposes. It is worth noting that this section has been written with Android in mind and may have to be adapted for other devices. Note that I have included the same testing build provided to users in user testing to disable email verification and set listings to end in minutes as opposed to days. This is to allow easier testing and enables multiple accounts to be made with ease to buy tickets for separate items.

The easiest way to run raffle, is through sending the APK included to your android device. Ensure that 'install from unknown sources' is checked in your settings, and after this you should be able to click the APK file and install said application. It is as simple as that and doesn't require any additional setup.

If you wish to build and run Raffl directly from its source code, the process will be slightly different. This would require an installation of Android Studio and your Android device to be configured to allow USB debugging (found in the developer options). Next, after installing all the Flutter package on Android Studio, connect your android device to the Flutter build, and finally, run the build using the run button in the top of the screen.

9.2 – Reflections and Future Directions

When developing Raffl, many things went well, and I was particularly happy with the result. As demonstrated by my implementation section, I stayed relatively on track throughout the entirety of the development, besides some minor setbacks, and as a result, the plan was followed, and the initial specification was entirely captured. By extension, almost all requirements were completed aside from a few of the more time-consuming 'could' requirements. This indicated that the methodologies followed, such as Agile, were a solid fit for such a project.

The user testing section exhibited a significant level of interest as the feedback was overall positive, suggesting that this product has the potential to be successfully introduced to the market. However, it would still need work in its current state to be marketable. To do this, I would need to consider future additions and changes to improve Raffl's user experience.

One such change would be adding user reviews, a very popular feature that could greatly benefit customers. Initial research indicated that this would be useful, and this was exacerbated by overwhelming support for it with user feedback. Doing this would also bring Raffl more in line with popular e-commerce applications, like eBay, ensuring that it can stay competitive in the market.

Another area of research that could be explored would be machine learning recommendations. As these are relatively time-consuming to set up and require lots of data to train, I did not feature it here; however, with an improved AI system that could accurately personalise the user experience to be more in line with their preferences, engagement would

be driven up. If leveraged effectively, the machine learning algorithms could also collect analytics and give valuable insight into user behaviour to improve the app further. Doing this effectively may require significant funding, so it may be better in the hands of a big company like eBay or Facebook. The nature of raffle tickets requires significant demand, so the marketing only a big company can provide may be most beneficial here.

Another avenue that could be explored is the adoption of smart contracts on a blockchain. Smart contracts are an emerging area of interest in blockchain technology and are essentially self-executing contracts, which have their terms hard-coded into the blockchain. For example, if I used the Ethereum blockchain, I could create a smart contract for Raffl by directly writing the terms of the contract, including randomisation of winner selection, into Ethereum's proprietary programming language, Solidity. This would enable the entire ticket sales process for the raffles and winner selection to be transparently recorded and made easily accessible and verified by anyone on the blockchain. This would help to assure users that the choice of winners for raffles is fair and impartial, building a stronger reputation for Raffl and differentiating itself further. Moreover, as a self-executing contract, if contract terms are not met, then all transactions are reversed, allowing a large portion of the payment system to be automated by the blockchain. Nevertheless, potential downsides to this addition could be the running costs of dealing with payments through the blockchain, as well as the specific security risks that come along with dealing with cryptocurrencies. Furthermore, blockchain is still a relatively new concept that many users might struggle to fully understand, resulting in more mistrust that would defeat the purpose of its inclusion.

In conclusion, the process of developing Raffl provided me with valuable skills and practical knowledge in the context of app development. This not only included experience necessary to plan and research for a potential application but also offered me hands-on experience with technologies typically used in the industry, including both Flutter and Firebase. By extension, the insights I gained here have been invaluable for any development projects I may undertake in the future.

10 – References

[1] MacroTrends, ‘EBay Revenue 2010-2023’. Accessed: Jan. 07, 2024. [Online]. Available: https://www.macrotrends.net/stocks/charts/EBAY/ebay/revenue

[2] S. F. Verkijika, ‘Download or swipe left: The role of complexity, future-oriented emotions and feature overload’, *Telematics and Informatics*, vol. 60, p. 101579, Jul. 2021, doi: 10.1016/J.TELE.2021.101579.

[3] R. M. Hogarth, ‘Intuition: A Challenge for Psychological Research on Decision Making’, *Psychol Inq*, vol. 21, no. 4, pp. 338–353, Nov. 2010, doi: 10.1080/1047840X.2010.520260.

[4] Capital One, ‘Facebook Marketplace Statistics (2024): User & Revenue Data’. Accessed: Jan. 4, 2024. [Online]. Available: https://capitaloneshopping.com/research/facebook-marketplace-statistics/

[5] AppAdvice, ‘Raffall - The Competition App! by Raffall Limited’. Accessed: Dec. 07, 2023. [Online]. Available: https://appadvice.com/app/raffall-the-competition-app/1079900801

[6] R. Kaiser, ‘49 UX/UI Statistics for 2022 - Wilderness Agency’. Accessed: Dec. 07, 2023. [Online]. Available: https://www.wildernessagency.com/ux-ui-statistics-for-2022/

[7] N. Jakob, ‘Ten Usability Heuristics’, *Heuristic Evaluation*, 2005, Accessed: Nov. 18, 2023. [Online]. Available: https://www.su.se/polopoly_fs/1.220913.1422015209!/menu/standard/file/10%20Heuristics%20for%20User%20Interface%20Design_%20Article%20by%20Jakob%20Nielsen.pdf

[8] S. Singh, ‘Impact of color on marketing’, *Management Decision*, vol. 44, no. 6, pp. 783–789, Jun. 2006, doi: 10.1108/00251740610673332/FULL/XML.

[9] A. Hogan and D. Laufer, ‘The Six Steps For Justifying Better UX | Forrester’. Accessed: Dec. 10, 2023. [Online]. Available: https://www.forrester.com/report/The-Six-Steps-For-Justifying-Better-UX/RES117708

[10] Leadpages, ‘UX Statistics - 53 Data Points to Know for 2023 | Leadpages Blog’. Accessed: Dec. 10, 2023. [Online]. Available: https://www.leadpages.com/blog/ux-statistics

[11] Google, ‘Accessibility – Material Design 3’. Accessed: Dec. 10, 2023. [Online]. Available: https://m3.material.io/foundations/accessible-design/accessibility-basics

[12] B. Fontenelle, ‘7 Best Practices For Accessible App Design | ArcTouch’. Accessed: Dec. 10, 2023. [Online]. Available: https://arctouch.com/blog/accessible-app-design

[13] UK Parliament, ‘Gambling Act 2005’, 2005. Accessed: Dec. 12, 2023. [Online]. Available: https://www.legislation.gov.uk/ukpga/2005/19/contents

[14] Gambling Commission, ‘Fundraising and lotteries’. Accessed: Dec. 12, 2023. [Online]. Available: https://www.gamblingcommission.gov.uk/public-and-players/fundraising-and-lotteries

[15] Raffall, ‘Why do I have to answer a question before entering a competition? – Raffall’. Accessed: Dec. 12, 2023. [Online]. Available: https://help.raffall.com/hc/en-gb/articles/360018871340-Why-do-I-have-to-answer-a-question-before-entering-a-competition

[16] Financial Conduct Authority, ‘About us | FCA’. Accessed: Dec. 13, 2023. [Online]. Available: https://www.fca.org.uk/about

[17] UK Parliament, ‘Data protection: The Data Protection Act - GOV.UK’, 2018. Accessed: Dec. 13, 2023. [Online]. Available: https://www.gov.uk/data-protection

[18] L. Rockoff, *The Language of SQL - Larry Rockoff - Google Books*, 3rd ed. 2021. Accessed: Apr. 17, 2024. [Online]. Available: https://books.google.co.uk/books?hl=en&lr=&id=0NnPEAAAQBAJ&oi=fnd&pg=PT17&dq=SQL&ots=gD4Q-COWOk&sig=R4X49pu4RrX-hkhIfxYbDq9_e-8&redir_esc=y#v=onepage&q=SQL&f=false

[19] Firebase, ‘Choose a Database: Cloud Firestore or Realtime Database | Firebase Realtime Database’. Accessed: Dec. 13, 2023. [Online]. Available: https://firebase.google.com/docs/database/rtdb-vs-firestore

[20] Statista, ‘Internet penetration in the UK 2023 | Statista’. Accessed: Jan. 02, 2024. [Online]. Available: https://www.statista.com/statistics/1124328/internet-penetration-uk/

[21] Criteo, ‘Data & Trends’. Accessed: Jan. 02, 2024. [Online]. Available: https://www.criteo.com/blog/category/data-trends/

[22] Statcounter, ‘Desktop vs Mobile Market Share Worldwide | Statcounter Global Stats’. Accessed: Jan. 02, 2024. [Online]. Available: https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/#monthly-201806-202312

[23] Statcounter, ‘Mobile Operating System Market Share United States Of America | Statcounter Global Stats’. Accessed: Jan. 02, 2024. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/united-states-of-america/#yearly-2009-2023

[24] NordVPN, ‘Android vs. iOS: Security comparison 2023’. Accessed: Jan. 02, 2024. [Online]. Available: https://nordvpn.com/blog/ios-vs-android-security/

[25] J. Pandya, ‘Firebase Flutter vs Firebase React Native- A Detailed Comparison’. Accessed: Jan 27, 2024. [Online]. Available: https://www.expertappdevs.com/blog/firebase-flutter-vs-firebase-react-native

[26] ‘Cross-platform mobile frameworks used by global developers 2022 | Statista’. Accessed: Dec. 12, 2023. [Online]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[27] Creately, ‘Creately App’. Accessed: Apr. 30, 2024. [Online]. Available: https://app.creately.com/d/start/dashboard

[28] S. Madsen, ‘How To Prioritise Requirements With The MoSCoW Technique’. Accessed: Feb. 24, 2024. [Online]. Available: https://www.knowledgehut.com/blog/agile/how-to-prioritise-requirements-with-the-moscow-technique

[29] Microsoft, ‘Characteristics of a Good Mobile App | Microsoft Power Apps’. Accessed: Feb. 24, 2024. [Online]. Available: https://powerapps.microsoft.com/en-us/what-makes-a-good-app/

[30] R. Fleck, ‘How to use color to evoke powerful emotions in your design | Dribbble Design Blog’. Accessed: Jan. 5, 2024. [Online]. Available: https://dribbble.com/stories/2020/07/14/color-and-emotions

[31] Microsoft, ‘Image Creator from Microsoft Designer’. Accessed: Feb. 24, 2024. [Online]. Available: https://www.bing.com/images/create