University of Dublin

TRINITY COLLEGE

QuizFlash:

A React Native Flashcards App Using Adaptive Environments To Promote The Learning

Niall Mulloy
B.A.(Mod.) Business and Computing
Final Year Project April 2019
Supervisor: Dr. Lucy Hederman

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this project is, except where otherwise stated, entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

Name:

Date:

Acknowledgements

I would firstly like to thank my supervisor Dr. Lucy Hederman for her guidance, support and valuable advice throughout the year. I would also like to thank Swapnil Raj whose consultation and assistance greatly steered me towards the right direction. Lastly I would like to thank my parents for their support and encouragement throughout the duration of this project.

Abstract

This report aims to outline the research, requirements gathering, planning, development and implementation of Quizflash, a React Native flashcard app which allows students to quiz themselves in an adaptive learning environment that caters for their specific learning style. This involves implementing a dynamic user model based on the Leitner flashcard system that would analyse users' quiz answers and ask them questions they get wrong more frequently. Furthermore, the also allowed the user to select their preferred answer method for these quizzes based on the user's learning style from the VARK model

Table Of Figures

Figure 5.5: Survey Question about benefit of Swipe

Table Of Contents

Chapter 1 Introduction

The world of education has started to increasingly integrate smartphone technology into its learning process in recent years. It is now a common sight to see students in schools or universities using their phone as a form of study tool to disrupt the traditional and monotonous studying techniques that currently exist in society and make them more fun, engaging and effective.

However despite this trend, the author has found that the current offerings available for the studying of e-flashcards on a mobile device are underwhelming and inadequate. They tended to appeal to only one style of user, had no form of adaptability and lacked contemporary features that would engage a younger demographic.

The solution was to build a React Native flashcard app which allowed students to quiz themselves in an adaptive learning environment that caters for their specific learning style. This involved implementing a dynamic user model based on the Leitner flashcard system that would analyse users' quiz answers and ask them questions they get wrong more frequently. Furthermore, it also allowed the user to select their preferred answer method for these quizzes based on the user's learning style from the VARK model

1.1 Motivations

The inspiration for this came to the author last year while studying for exams. They found that when some form of recall was involved in their studying process, it greatly enhanced their comprehension of content and boosted their exam results significantly. As a result, they became fascinated with the concept of learning optimisation and wanted to know what makes some form of learning better than others.

The author began experimenting with a wide variety of study techniques and platforms in attempts to isolate variables that affected the process. Ultimately they found that while some common variables exist, students tended learn best when they know what type of processes work for them specifically. This was frustrating in some ways as the author had discovered that they greatly benefitted from the style of auditory learning but that created systems in education tended to favour those who had strong reading/writing learning styles. When they went looking they found there was a lack of both practical and online resources for those didn't conform to this one methodology of learning.

With this their recent interest in these adaptive learning techniques in conjunction with the rise of smartphone utilisation in education, the author decided to address the issue directly and create an app that they could use to create more optimal and personal study content.

1.2 Aims

The objectives of this project was to create an app for mobile platforms which allowed students to quiz themselves on sets of virtual flashcards in an environment that was most conducive for their learning. The author will fulfil this aim by:

- Creating an app that allows for the creating, editing, deleting and quizzing of virtual flashcards and their decks
- Designing models that assist and benefit the user's learning experience
- Implementing systems which allow users to assess their own learning styles and pick their answer methods accordingly

While there is a focus on a simple and accessible User Interface, this app is designed for students who have experience using swipe answer systems. Therefore, the app was created with these users in mind, taking inspiration from other apps they are likely to have had previous experience with.

The aim was also to design this platform using the React Native framework as to avail of its cross platform capability and because the majority of students own smartphones and use them some form of educational. The author also wanted to incorporate phone specific features into the app in the form of swipe and slide components.

1.3 Personal goals

The overall aims of the project were shown in the previous section. However the author was also looking to achieve some personal goals in the undertaking of this project. Despite having completed multiple Computer Science module involving software development, the author had never before designed and implemented a mobile application and had no prior experience with React Native framework and only very little with JavaScript. Consequently the personal goals for this project are:

- The creation of an app which functions on android and apple devices and has the extensibility to become a web app
- The learning and development of:
    - React Native
    - Redux
    - Expo
    - JavaScript
    - JSX

- The construction of an adaptive learning system and its user model that have a genuine effect on the learning process
- The development of software which resolves a real-world problem
- The establishment of a potentially commercially viable service

1.4 QuizFlash

This project, including the described aims, motivations and personal goals, culminated in the
creation of QuizFlash, a flashcard mobile app that promotes adaptive learning.

Fig 1.1: QuizFlash app logo

Chapter 2: Background & Research

2.1 Flashcards Overview

Flashcards are a common study tool in which pieces of paper or note cards have a term or question written on one side and a definition or answer written on the other. When an individual studies using these flashcards, they stereotypically look at the question or term on one side and try to recall the information on the other (Bryson, 2012) . For example, if an individual was learning German vocabulary terminology, one side of the card might read *'der Bruder'* and the other side *'brother'* (shown In Figure 2.1). The option of reading aloud the question and stating the question is entirely dependent on the user. Alternatively, you could have a photograph of a dog on one side and *'der Hund'* (its German translation) on the other.

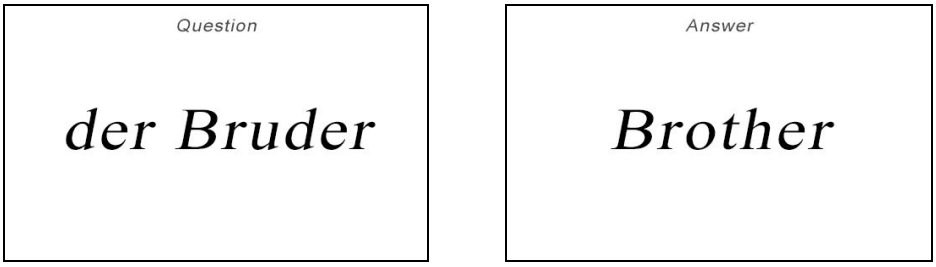


Fig 2.1 Flashcard example

It is a common study method used by over 65% of college students. 83.3% of these students stated they used them because they helped in the memorisation of large sets of information and/or found it an easy method of learning (Wissman, Rawson and Pyc, 2012).

n = 216

What Type Of Information Do You Use Flashcards To Study For:	**Percentage (%)**
Vocabulary	82.9
Factual Information From Notes (people/places/dates)	29.2
Key Concepts	28.2
Formulas	7.4

Difficult Information	3.7
Information From Study Guides	3.7

Fig 2.2 Table Describing What Flashcards Are Generally Used For
Source: (Wissman, Rawson and Pyc, 2012).

If we look at Figure 2.2, we can gauge the overwhelmingly predominant use of flashcards is to test vocabulary and that the utilisation of flashcards occurs most frequently when learning information with a short question and answer lengths.

2.2 Why Flashcards Are Better Than Other Forms Of Learning

A study conducted by Roediger and Butler (2011) examined the important role of retrieval practice in long term retention and came to the conclusion that the finding and retrieving of information from memory produces a significantly better retention rate than the continuous restudying of this same information in a passive manner (e.g reading, writing, transcribing etc.) for an equal amount of time. Even if the learner does not accompany the mental retrieval with an outward response, the act of testing recall in the form of flashcards still strengthens memory better than other forms of study (Carpenter, Pashler and Vul, 2006). This is due to the fact that the flashcards study process is a metacognitive control method (Kornell and Bjork, 2008). Metacognition can be defined as the monitoring and understanding of one's own learning process and having an awareness of an individual's thought process. The most successful form of metacognitive control relies on the steady improvement of retrieval fluency, cue familiarity and the rate of success for retrieval attempts (Kornell and Bjork, 2008). All these brain functions are being engaged when using the flashcard study method and therefore the assumption can be made that it is an effective technique. This is supported by an experiment performed by Karpickie and Roediger (2008) that found that students who used flashcards to repeatedly test their recall ability while studying a set of 40 Swahili-English word pairs, saw a 44% increase in their exam scores in comparison to those who use other methods to study (shown in Figure 2.3). This was also proven over a variety of practical domains including, but not limited to: the learning of vocabulary in native and second languages, face–name associations, general facts, text passages, word lists, and maps (Carpenter, Pashler and Vul, 2006). Finally the use of flashcards not only improves the recallability in the direction that was practiced (e.g., German to English vocabulary such as *'der Bruder*' to *'Brother'* in figure 2.1), but also in the direction that was not practiced (*'Brother'* to *'der Bruder*').

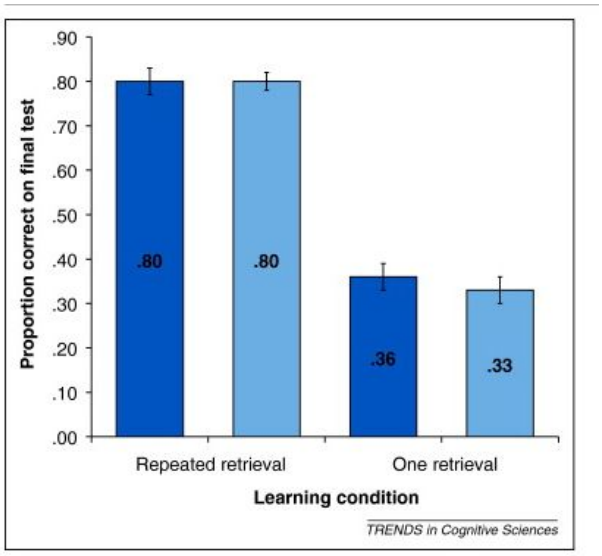


Fig 2.3 Graph Showing Repeated Retrieval v One Retrieval
Source: (Carpenter, Pashler and Vul, 2006)

2.3 Leitner System

This a flashcard study method created by German Scientist Sebastian Leitner in the 1970's for the effective studying of flashcards. It is a simple system implementation based on the principle of spaced repetition in which cards are viewed at increasing intervals depending on how frequently they are answered incorrectly. The system firstly splits the flashcards into boxes based on how well the learner knows them. The learner then iterates through the flashcards contained in these boxes based on what study session they are on. If they recall the answer to the flashcard correctly, then it is advanced to the next group. If they recall the answer to the flashcard incorrectly, then is relegated back to the first box. The result is that each succeeding group has a longer period of time before the learner is required to revisit the cards. (Godwin-Jones, 2010)

Example - Leitner System With 3 Boxes

Imagine there exist 3 boxes of flashcards called "Box 1", "Box 2" and "Box 3" (shown In Figure 2.4) The flashcards placed in Box 1 are cards that the learner frequently gets wrong and the flashcards placed in Box 3 are cards that they normally get right. They could choose to study the flashcards in:

- **Box 1:** once every *N* periods e.g once a day, once every 3 days
- **Box 2:** once every *2N* periods e.g once every 2 days, once every 6 days
- **Box 3:** once every *3N* days e.g once every 3 days, once every 9 days

If they answer a flashcard correctly while studying it in Box 1 it is promoted to Box 2. If they answer a flashcard correctly in Box 2 it is promoted to Box 3.

However if they answer a flashcard incorrectly in Box 2 or Box 3, it is demoted to Box 1. This forces the leaner to test themselves on that card more often.

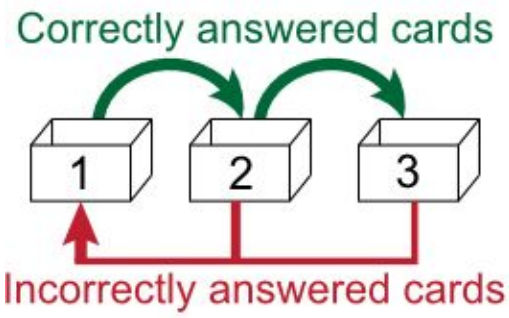


Fig 2.4 Leitner 3 Box System

This system is built on the fundamental principle that the learner has to focus on learning the more difficult flashcards more frequently so as to improve their recall ability on them. This premise is supported by research completed by Pyc and Rawson (2009) which concluded that more difficult but successful retrievals are better for memory than easier successful retrievals and therefore more is learned by the individual. The assertion is based on two key principles: the spacing of flashcards and the achieving of higher criterion levels.

2.3.1 Spacing of Flashcards

This is the premise that distributing learning events over a longer period of time is a significantly more effective studying method than massing them together in a short amount of time (Cepeda et al., 2006). Therefore when studying a set of flashcards, multiple shorter study sessions achieve better retention rates than a single large one. For example, six hours of studying spaced over four days is better than six hours of studying crammed into one day. This can be summarised as the longer the lag between studying, the greater the benefit for the student. In an experiment done by Nate Kornell (2009), he proved this by showing that 90% of students performed better when spacing their flashcard study sessions further apart instead of studying them in mass the day before the exam. This was contrary to the prevalent assumption held by 72% of students who believed that cramming was a more effective technique (Kornell,2009). This also applied for the spacing of flashcards within a study session, with Kornell verifying that the larger the spacing between repetitions of a given card, the higher the rate of retention. Essentially, this means that studying one large stack of flashcards is a more effective way of learning than studying many smaller decks with the same amount flashcards. This also was a commonly mistaken belief by students with 72% of them believing that they learn more using smaller decks of 5 to 8 flashcards (Wissman, Rawson and Pyc, 2012) This corroborated with a more scientific study conducted by Baddeley (1992) which came to the conclusion that

repetition occurring between spaced and timed breaks, where language is continuously reiterated in between these increased time lapses, enables the regeneration of neurochemical substances in the brain, which is the most significant method to improve a student's long term recall ability on a subject.

2.3.2 Achieving Of Higher Criterion Levels

This is the assertion that repeatedly recalling an answer correctly more frequently makes it easier to recall in the future. An experiment performed by Vaughan & Rawson demonstrates this by finding that by recalling a word pairing of Lithuanian to English flashcards more than once in a study session, improved its cued recall rate by a minimum of 20% (Vaughan and Rawson, 2011) (shown in Figure 2.3). If the learner recalls it four or five times, this percentage is raised to 40%.

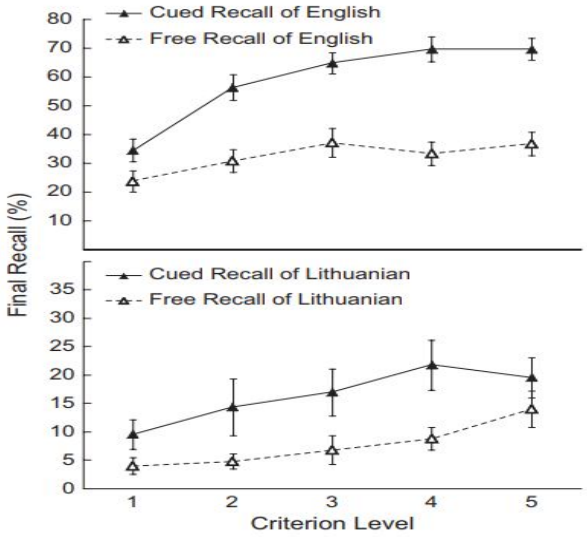


Fig 2.5 Line Chart Showing Recall Rates For Lithuanian & English Word Pairings
Source: (Vaughn and Rawson, 2011)

This corroborates with a range of different studies done by Pyc & Rawson (2009) which also confirmed this same hypothesis by demonstrating that when participants studied foreign language word pairs by practicing them until they were correctly recalled 1 to 10 times, it drastically improved final test performance from 12% to 79% in 13 groups across four experiments. Despite this information, 63% of learners still tended to remove a flashcard from the deck after only recalling it correctly once (Wissman, Rawson and Pyc, 2012). This indicates that a disconnect exists and that students do not comprehend the importance and advantages of reaching higher criterion levels while studying. Students reported learning to a predetermined level criterion (generally one), but appear to have no idea what an appropriate standard level of criterion is (Kornell,

2009) . Regardless, the conducted studies have proven that the increasing number of times a flashcard is correctly recalled while studying significantly improves the associative memory, the target memory and the cue memory.

The design of QuizFlash's flashcard selection system was heavily influenced by Leitner's system and while I will go on to discuss in later sections how the final system utilised by the app operates in a different manner, they do share a lot of key fundamental concepts, such as:

- The system should ask the user questions they get wrong more frequently
- The promotion of the spacing of cards within a study session
- The encouragement of larger deck sizes
- The promotion of higher criterion levels

2.4 Adaptive Learning System

Adaptive learning is an educational mechanism which utilises algorithms and previous information to provide an individually customised learning experience that is tailored to the student's needs (Kerr, 2015). Essentially, it is the way in which the learner interacts with the previous content and how these interactions dictate the nature of the materials provided to them. The process should be dynamic, interactive and automated to create a personalised learning experience for the user. To provide this customised experience, research was conducted into the learning styles of students.

2.4.1 Learning Styles & VARK

There exists a wide variety of learning styles and theories that look to explain the differences in individuals' learning methods. The majority of these theories suggest that all people can be segregated according to their style of learning. The VARK learning style model is one of these and was introduced in 2006 by Neil Fleming as an amendment to the traditional VAK model created in the 1920's. VARK is an acronym standing for: *Visual, Auditory, Reading/Writing* and *Kinesthetic/Tactile* (Prithishkumar and Michael, 2014). It is a model which categorises students into four different learning processes

- **Visual**: A student who retains information better when it is presented to them in a visual format e.g pictures, diagrams and charts
- **Auditory:** An auditory-dominant learner is an individual who prefers listening to what is being presented. They respond best to sound and voices e.g a lecture or group discussion. Hearing their own voice or repeating something back to themselves also assists their learning.
- **Reading/Writing:** A person who learns best through the repetitive process of reading and writing. Their most effective form of learning is to

assimilate information by taking down notes or writing out questions and answers

- **Kinesthetic/Tactile**: a kinesthetic or tactile learner is someone who absorbs information best through the form of physical experience. They respond well to an interactive and “hands on” approach that can involve the touching and feeling of objects or learning props.An example of kinesthetic or tactile learning would be dancing or gymnastics.

In terms of creating an adaptive learning system for the flashcards app, it’s impossible to create one that appeals to a kinesthetic learner as they can’t experience a practical and “hands on” approach for them to utilise through an app. Resultantly, the system will have be developed to focus on those who are some form of Visual, Auditory or Reading/Writing learners. While this is disappointing as the majority of users learn most effectively through kinesthetic learning (as shown in figure 2.6), this isn’t an issue for the app’s appeal as over 87% of individuals are Bimodal or Trimodal which means they have 2 or 3 preferences.(shown in Figure 2.7)

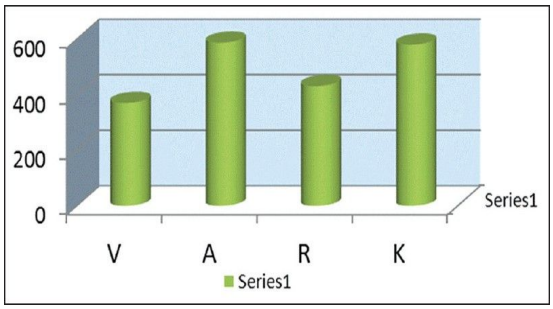


Fig 2.6 Shows The Distribution of VARK
Source: (Prithishkumar and Michael, 2014)

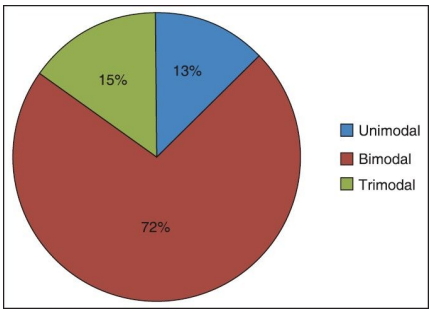


Fig 2.7 Pie Chart Showing The Percentage of Multimodal Users
Source: (Prithishkumar and Michael, 2014)

2.4.2 User Model

The user model describes the division of human to computer interaction occurring within the adaptive system. This describes the process of gathering the information, knowledge and preferences of the user (Luíz Faria, Vaz de Carvalho and Carrapatoso, 2008) and its main objective is to adapt the system to their needs based on this data. There were two options assessed for the creation of a user model within QuizFlash

2.4.2.1 Highly adaptive user models

These types of models attempt to represent one particular individual and as a result can have a very high amount of customisation occuring within the system. It looks to find a specialised solution for each type of user and while this results in a large amount of advantages for the learner in the form of high adaptivity, it does require a lot of information to be collected.

2.4.2.2 Dynamic User Model

Dynamic user models enable an up to date representation of users to be implemented without as much information being required. Changes in a user's interaction with the system and their learning progress are recorded and utilised to update the current goals and objectives of the app.

Ultimately the dynamic user model was selected as the process to use. It was chosen because :

- The system did not require a lot of information to function
- The need for simplicity due to time constraints
- Beyond the capabilities of the developer who was programming in a new format

2.4.3 Data gathering

Information about users needs to be gathered to implement this model and there exist three main methods for this to be performed:

2.4.3.1 Asking For Specific Facts When Interacting

The system prompts the user to enter in specific information outlining their preferences or needs into the user model so they can be fulfilled. Normally parameters can be changed or edited afterwards by the user

2.4.3.2 Learning Preferences Through Observation

The information is retrieved from the user's behaviour when interacting with the system. The application learns from observing interactions and analyzing:

- The methods users choose to accomplish tasks
- The combination of things they take an interest in
- Their interactions with the things they observe

2.4.3.3 Hybrid Approach

This approach is a combination of the ones above. Users have to answer questions and return feedback to set up some aspects of the model. Other facets of the model will be satisfied by observing their interactions within the system and adjusting the user model accordingly.

The process selected for the gathering of information was the hybrid approach as it was the most appropriate for the model design. The description of how exactly this will be described in the following section.

2.4.4 Finalization Of Adaptive Learning Systems

After conducting research on Leitner flashcard system in section 2.3, the VARK learning styles and a user model's necessary component an adaptive learning system was finalized. The system's two primary adaptability goals and objectives were outlined as follows:

- To select Flashcards that the user answers wrong more frequently. The information required for this process will be collected by observing the user's previous answer history and interactions with each flashcard.

- Allow for the user to choose the way the question or answer is displayed by the app. They should have the possibility to answer in the form of a visual, auditory or reading/writing method. The data needed to set up this procedure will be chosen by button selection each time a new flashcard is added to the system. However this process should be editable.

2.5 Mobile Learning

2.5.1 Overview Of Mobile Learning

Mobile Learning can be defined as the educating of students using some form of portable device, such as a tablet or smartphone, through the usage of learning materials in the form of applications, social interactions and online resources (El-Hussein and C. Cronje, 2010). The benefits are that it is flexible and allows the student to access educational frameworks at any time or anywhere. It provides a method for educational institutions to deliver education directly to their students. Students of Trinity College Dublin for example can currently use mobile apps to download course instruction, upload assignment to teachers and work in online social groups. The use of smartphones for educational purposes is a highly understudied topic even though their accessibility and potential has expanded rapidly in recent years. However it was found that within a college environment over:

- 75% students used smartphones during their breaks, meetings etc.
- 55% while simply just waiting around in college
- 45% of those asked used it directly for college related uses

(Dean, 2012).

This trend allows for the possibility of increasing study time if a student choose to use a flashcard based app during these downtime opportunities.

2.5.2 The Application Of Using Mobile Learning Flashcards In a College Environment

The usage of traditional flashcards was compared against the usage of mobile flashcards in an Introductory Psychology college class in Southwestern Oklahoma State University in an experiment completed by Burgess and Murray (Golding, Wasarhaley and Fletcher, 2012). It should be noted that the flashcard application used by the students did not incorporate any form of adaptive learning process and was a simple 'Yes/No' smartphone app. Before the study an initial survey was conducted and it was found that approximately 70% of the 415 members in the class used flashcards at some point during the module. This was comprised of 65.5% of students using written flashcards, while only 3.9% used a computer or smartphone flashcard applications. 6.5% of the participants who were found to use flashcards indicated that they sometimes use both. During the experiment four written exams were performed, in which traditional and mobile flashcards were encouraged to be used as a study technique before the exam. In some instances the teacher even created flashcards for his students to use.

- **1st Exam:** No form of flashcards were provided before the exam and as a result only 15% created their own traditional flashcards, while 0% created any using a flashcard app.
- **2nd Exam:** Both traditional and app flashcards were provided before the exam and flashcard usage rose to 74.2%. Of those who utilised flashcards, 54.8% used traditional flashcards and 19.4% used the flashcard app
- **3rd Exam:** Both traditional and app flashcards were once again provided before the exam and flashcard usage rose even higher to 90.6%. However, of those who utilised flashcards, all 90.6% used traditional flashcards and 0% used the flashcard app.
- **4th Exam:** Only mobile flashcards were provided before the exam and flashcard usage deceased to 40%. Of those who used flashcards, 29.1% created their own traditional flashcards and only 10.9% used the flashcard app.

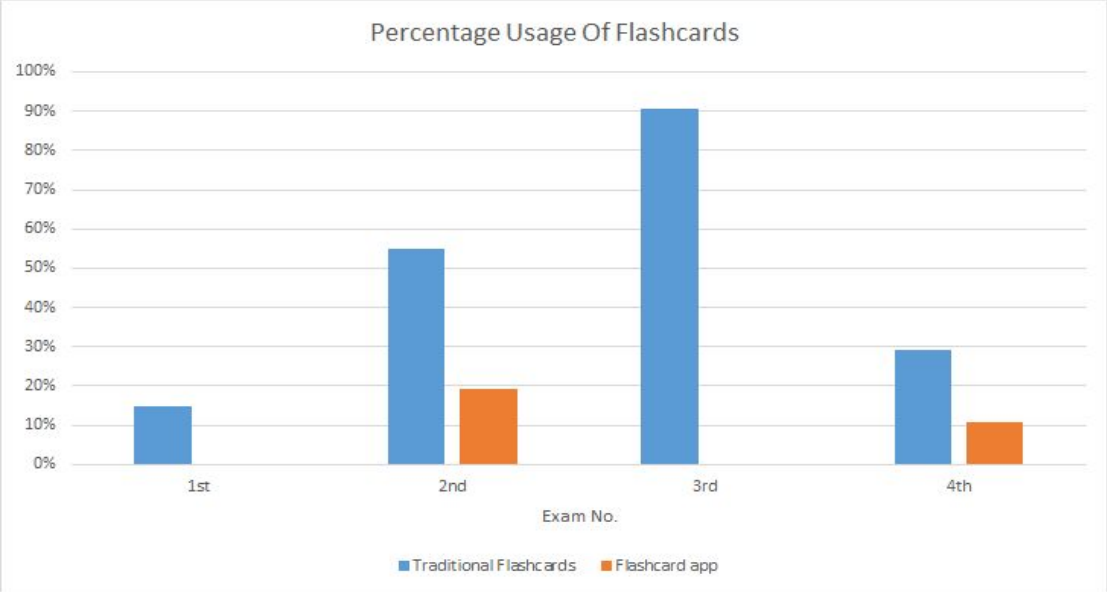


Fig 2.8 Bar Chart Showing The Percentage Use Of Flashcards during Exam

In terms of results, there were no significant differences in exam grades between those who used traditional flashcards and the app. However, it should be mentioned that those who used flashcards as a studying technique in the first exam had a much higher average than those who didn't. After the exams, another survey was conducted in which the students were asked the open-ended question on the advantages and disadvantages of using a flashcard smartphone app. These were the general consensus of the responses:

- **Advantages of using an app based flashcards:**
  - Portability
  - Ability to study almost everywhere
  - The availability on their smartphone was convenient

- **Disadvantages of using an app based flashcards:**
  - Technical issues, specifically draining too much of their phone's battery life
  - That they would be easily distracted by other things on their smartphones
  - They would forget about the app's availability

Conclusions that can be drawn from this study are that:

1. The usage of both traditional and app based flashcards increased when the students were provided with them. One of the reasons they didn't use flashcards in the other exams was that they thought the creation process took too long.
2. However, when just provided with the app based flashcards students were still more likely to create their own traditional based ones indicating that there

currently exists a disconnect in college students with regards to using their phone as a form of study tool. Many were simply just not aware of that they could use their phone as a viable type of study tool.
3. Students value their phone space and battery consumption. They do not want to use a flashcard app that takes up a lot of storage and drains their phone.
4. They were very easily distracted by entertainment or social media apps on their phone as the app simply wasn't engaging enough.

The learnings taken from this study used to assist the development of the app were that:
1. If flashcards are pre-loaded into the app, the students are more likely to use them
2. College students currently aren't aware that they can use their phone as a credible studying platform. One possible solution, (if this project ever evolves into a credible business case) is to work in collaboration with teachers and professors to promote the usage of QuizFlash. It should also be recognized that this trend is changing as discussed by the statistics in section 2.5.1.
3. The app developed should not consume many resources to minimize phone storage and battery consumption.
4. The app needs to have more contemporary features and animations to compete with the entertainment apps for the user's focus. The app could possibly be gamified to engage with the user even further.

2.6 E-Flashcards & Competition

Electronic flashcards are an approach to support students for active learning and individual feedback and they're different from traditional cards because they require the learner to type in the answer during retrieval, thus combining the advantage of writing with that of ready-made cards. They can also manage the learning process, utilising what is known about the transition from short-term into long-term memory. The main two competing companies in this field are:

2.6.1 Quizlet

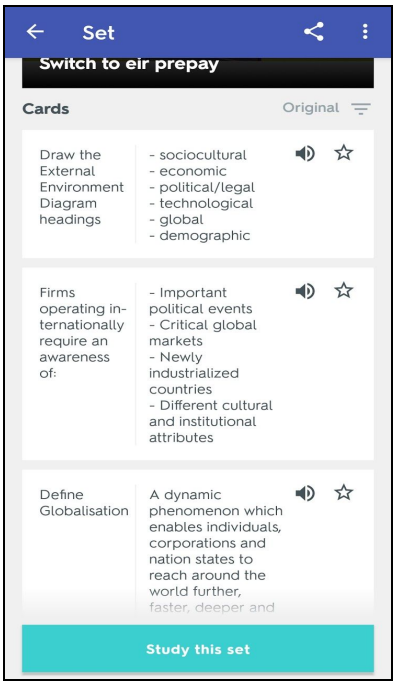


Fig 2.8 Quizlet ScreenShot

Quizlet is a mobile and web-based study application that allows students to study information via learning tools mainly in the form of e-flashcards. It is currently used by two-thirds of high school students and over half of university students in the United States.(Quizlet, 2018)

What They Do Well:

- Has a very sophisticated algorithmic system in its learn function to identify card selection methods
- Has 5 types of answer methods which all provide detailed feedback and history to the user on their progress
- Has a learning due date that assists the user work towards an exam and prompts the user on when they should use the app

What Can Be Improved:

- Very structured entry format that doesn't allow for 3rd party images, lists or any form of creative and adaptive input
- Lack of contemporary and engaging features such as a swipe function to quickly navigate through cards
- While extremely popular in the United States, it's not very well known in Ireland

2.6.2 Anki App

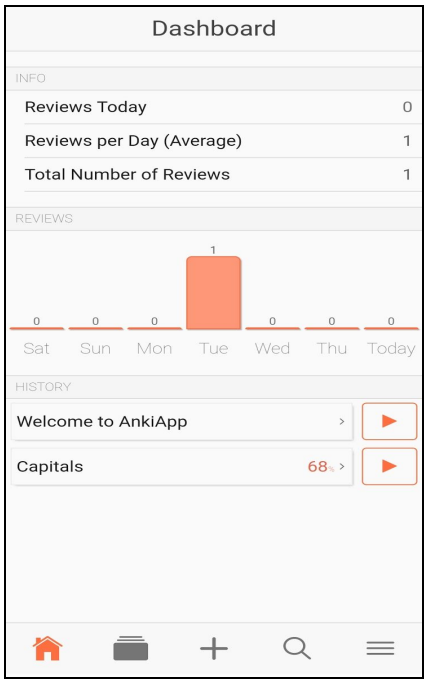


Fig 2.9 Anki ScreenShot

Anki is a spaced repetition e-flashcard system which allows a user to create, manage and review flashcards. Currently exist as an app for Android and iOS as well as a web interface that can be used to interact with the user's flashcard database.

What They Do Well:

- Highly customisable and allows for all types of answer formats to be utilised in their templates
- An impressive dashboard screen that provides a great overview to the user of their weekly breakdown
- Allows other users to see how frequently they've reviewed their flashcards and has a great online community presence

What Can Be Improved:

- Very complicated to uss due to the fact that it provides too much information at once and overwhelms new users
- Extremely difficult to navigate around. Extremely poor usability
- Very boring and does not engage with user well at all

2.7 Conclusion on Research

To conclude this chapter, the research was of great benefit to the author and laid out strong guidelines for the project to follow and most importantly a new perspective on how best to achieve the project's objectives was gained.

Chapter 3: Design

3.1 Introduction:

In this chapter, the author will examine and explain the project's design. More specifically this chapter will discuss the designs used during the development process and the consideration of concepts and technologies utilised throughout the project. The intention of this chapter is to clarify why some of the design decisions were made and to justify why these were the optimal solutions implemented to solve the issues experienced.

3.2 Requirements:

As there was no client for the project, the author had to set out his own requirements for the application. The functional and non-functional requirements were both established at the initial stages of the project and considered throughout the development process.

3.2.1 Functional Requirements

The functional requirements were defined at the start of the project and contained some essential features that set the foundation for the fundamental functionality of this app:

- **Allow the User To Take A Quiz** - The core basis of the app and the reason for its creation was to allow users to test their recall of a deck of virtual flashcards in the form of a quiz . After taking this quiz, they should be provided with their score to determine if their knowledge is progressing. The creation of this should be the primary goa.
- **Design Of Flashcards -** Flashcards are comprised of two parts, a face and a back:
    - **Face**: The front of the flashcard where the question is written

- **Back**: The back of the flashcard where the answer is written

The stereotypical way to use a flashcard is by reading the question on the face of the flashcard and then guessing the answer on the back. The flashcard is then flipped so the user can verify their and check if they were right. The aim of this requirement is to create a process that emulates this system as closely as possible.

- **Create An Adaptive System:** The system should implement different ways to answer quiz questions to improve the retention rates of those with different learning styles. Furthermore, based on the principles of the Leitner model, the app will look to employ algorithms to ask users questions they got wrong more frequently to improve their recall on harder flashcards by raising its criterion levels.
- **Design A Dynamic User Model:** A model which captures the interactions between the user and the system . This should be incorporated into the app to gather information and adapt the system to user's needs based on these interactions. The will gather information using the hybrid approach in the following two ways:
    - The app should pick questions based on their answer history
    - Accept answer methods the user has picked when adding the card.

- **Creating of Decks & Cards-** Users should be capable of creating decks, which are a grouping of virtual flashcards based on their topic e.g European countries and their capitals. Within these decks, they should then have the capacity to create an individual virtual flashcard and add it to the deck they just created. Each card should possess 4 main items:
    1. The card's question
    2. The card's answer
    3. The history the user has answering the card
    4. The method the user wants to answer by
- **Editing & Deleting of Deck & Cards -** This feature allows these decks or card to be edited or deleted by the users. They should be able to:
    - edit and delete individual cards from a selected deck
    - edit and delete a deck and all its cards
- **Informative Display Of Decks And Cards -** The system should create pages in the system that display the following two dashboards :
    - An overview of the created decks
    - An overview of the cards contained within decks

  Understanding how to navigate between It was pivotal then that the app displays them in a format that is cognisant and comprehensible to the user.
- **Engaging Features** - Based on the research, students are too easily distracted by other applications on their phones and find it difficult to

concentrate on studying when trying to use their phones when studying. As a result the system needs to be designed so it has one or two core features that engage them in the learning process and maintain their concentration levels for the entire length of the study session.

The app would also consist of 10 primary components, with each providing some form of assistance as to deliver these main functions. These sections are:

1. **Home Page -** should appear when users open the app, providing a layout of all the decks the user has created. Contains a search function, allowing the user to search through each deck.
2. **Flashcards -** There should exist two forms of flashcards:
    a. **Traditional Flashcards:** A double sided flashcard with the question presented in red text on the face of the card and the answer presented in green text on the back of the card. The default view presented on the screen of the traditional flashcard component, displays a view in which the flashcard's face is shown and the back is hidden. It's only when the face is pressed that the flashcard is flipped to display a view showing its back and hiding its face.

    b. Written Flashcards: A written input card, which has the exact same layout as the traditional flashcards except it has the addition of a text input field on the bottom of the component in which the answer can be entered

3. **Deck Addition Page -** Allows the user to add a new deck to the homepage
4. **Card Page -** An overview of all the cards contained in the deck as well as the study button which begins the quiz for the user.
5. **Deck Editing -** Where the user should be able to change the deck title and any other features of the deck
6. **Card Addition Page -** Allows the user to add a new virtual flashcard to the deck. Should also allow the user input in the form of buttons, selecting what format they would like the question to be presented in
7. **Individual Card Detail Page** - Just the virtual flashcard being presented by itself to the user
8. **Card Editing Page-** The component in which the user should be able to change an existing card's question, answer or answering style.
9. **Quiz -** A number of cards dependant on how many cards exist in the deck should be presented to the user in the form of a quiz and it should select these cards based on the their answer history. There should be many ways to answer the questions and this should be dependent on what the user selected when adding the flashcard

10. **Result** - Should detail the percentage of the questions answered correctly in the quiz as well as any other relevant information to the user.

3.2.2 Non-functional Requirements

A number of non-functional requirements were also identified as being key parts of the application.

- **Usability:** Making the app easy to use was a primary object of the design. The main usability requirement established at the beginning of the project was that the app employed a simplistic design and only incorporated a small number of meaningful application features which were easily identifiable and accessible. The app should also be designed tso hat user error is minimised and they're prevented from causing errors or inputting incorrect information.

- **Extensibility:** The application should be designed and implemented to allow for further development and the feasible inclusion of new features should be possible and not negatively affect the current system. This should be taken into account in the creation of the architecture, design and implementation so that a system is actively looking and considering its hypothetical future advancements.

- **Reliability:** The application should perform as intended and required in all situations. The user should be able to depend on the app to improve its study techniques and know that each time they go to use the application that it will perform as intended.

- **User Experience:** The app must achieve a number of user experience goals.At it's core the app should be encouraging as the main aim of the app is to encourage students to conduct more regular and effective study. Essentially students must enjoy the app or at the very least prefer it to other study alternatives and want to use it during their the study sessions. Additionally, the app should be aesthetically pleasing and have an engaging interface to be appealing and attractive to the core demographic of students aged 18-25.

3.3 Development & Planning

3.3.1 Development model

The project was developed using the incremental model as shown in figure 3.1. This model was chosen as it was adaptable and facilitated changes during the development process. Ultimately the model's iterative process allowed the author

to learn in the form of a structural mechanism. This proved essential as the author had never programmed in React Native before and had overestimated the project's capabilities when designing the requirements. The model also worked well in conjunction with React Native's live reload feature. This will be discussed in later sections.

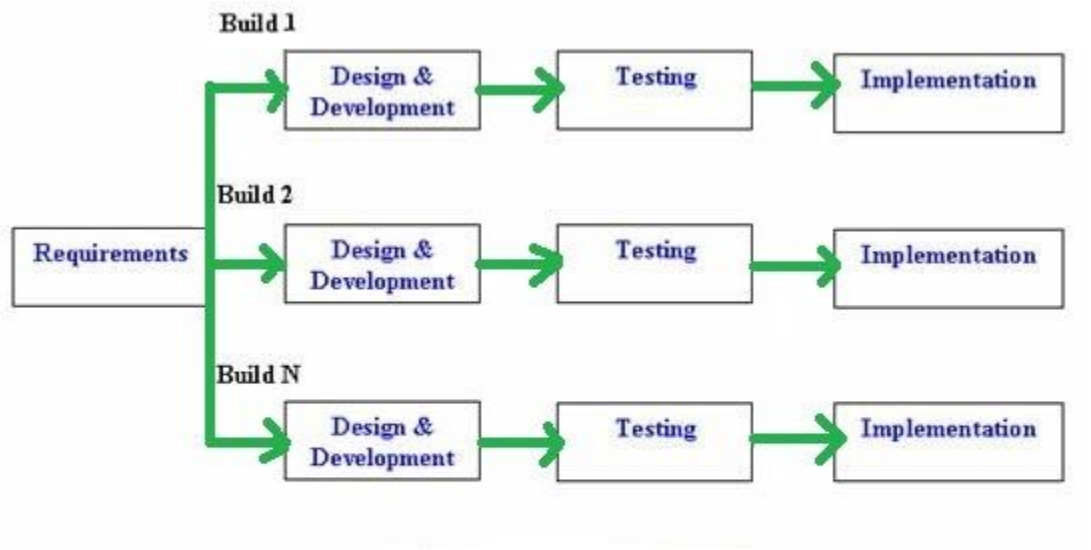


Fig 3.1 Incremental Model
Source:(Findnerd.s3.amazonaws.com, 2019)

Each of the components stated in the functional requirements in 3.2.1 became a core part of the app and after finalising the requirements needed, the model was used to split each component into three main concepts:

- Design
- Testing
- Implementation

The benefit of using this incremental method is being able to:

- Regularly test and interact with working software
- Focus, isolate and work on one component at a time
- Easily change the scope and requirements of a single component

3.2.2 Planning Method

The creation of a project management timeline occurred after the requirements had been completed and its aim was to ensure that a method for identifying project milestones and the time needed to achieve these milestones was set in place. The timeline set out a chronological sequence of events that need to occur by a certain date. (see figure 3.2). Furthermore, this also allocated time allowed for the testing of the system both internally

and externally. However the external testing was only planned to happen once all the components were incrementally implemented and after they had been tested initially and in an internally controlled environment.

Fig 3.2 Project Timeline

3.2.3 System Architecture

React Native splits the system into 3 separate layers as shown in figure 3.3:

1. **Front-end:** This layer comprises of the app's user interface and is completely rendered in the object-oriented language of JSX. This is the part of the system that is presented on the screen and interacted with by the user. It displays its user interface by retrieving information from its objects and controllers layer and formatting it using JSX to display it on the screen.
2. **Objects & Controllers:** This aspect of the system focuses on passing data between the front and back-end while performing some form of action or logic on the information. This layer is primarily comprised of all the functions and objects used throughout the development process and is programmed completely in the language of JavaScript.
3. **Back-end:** This layer determines how information is asynchronously stored by the app and the JavaScript library utilised to assist in the structuring of this information was Redux. It was chosen because of its capabilities to structurally create a system of how to maintain every components' many global states in one specific location. This was done by the employment of reducer and action functions which allowed for information to be easily be sent and received from the phone's storage space.

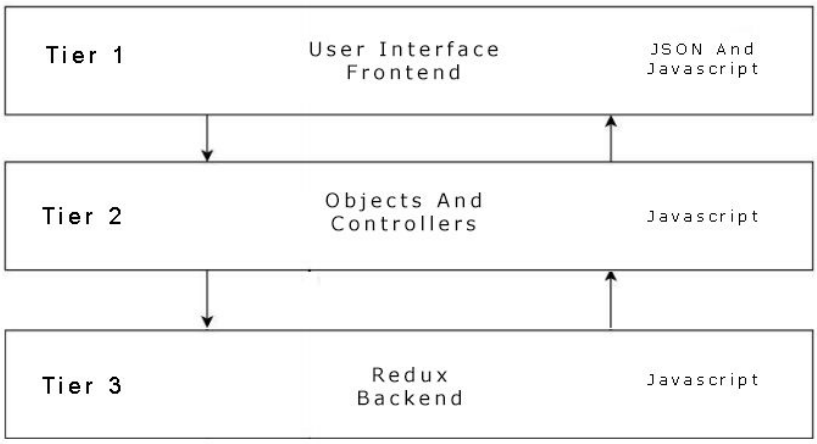


Fig 3.3 System Architecture

3.3 Influences on User Interface Design

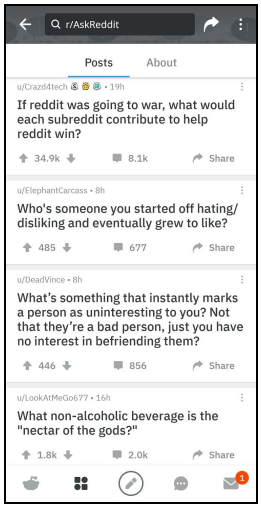


Fig 3.4 Reddit

Reddit

The inspiration for the basic FlatList component style layout of the system came from popular React Native apps such as Facebook and Reddit (see figure 3.4) . These apps are structured into card like sections that the user can easily navigate. A similar framework was utilised in the design of QuizFlash's UI and the benefit of adopting this approach is that it provides a strong and formatted design framework for the app to be built around.

Fig 3.5 Quizlet

Quizlet

The virtual flashcard layout and design was heavily influenced by the how flashcards are presented in the competing app of Quizlet (see figure 3.5). While lacking some engaging features and certain functionalities that exist in QuizFlash's system, it did have a simple and effective design layout that allowed users to be competently quizzed on a set of flashcards contained within a deck.

Fig 3.6 Tinder

Tinder

The swipe and slide features used in the quiz component of the app were directly inspired from the functionalities of the app Tinder (see figure 3.6). Within this tinder swipe system, a swipe to the right indicates a positive response and a swipe to the left indicates a negative response i.e. a right for 'Yes' and a left for 'No'. These features were taken to provide an intuitive and engaging method for users to navigate through a deck of cards and to successfully indicate to the system whether they answered a quiz question correctly or incorrectly.

3.4 Prototypes

The design of the UI began with the creation of low fidelity prototypes (see. Figure 3.7) in the form of rough sketches being done on paper. These were simply used as loose guidelines to follow when initially attempting to design and compile the system. The sketches laid out a plan of the how the home component, card component and quiz component should look and it outlined all the possible features that would be provided by the app. The advantage of using low fidelity prototypes such as these is that they can be rapidly created and quickly changed to match the scope of the requirements. They also provide a relatively clear idea of how the app should look,feel and ultimately be laid out. When being used in the introductory planning stages of Quizflash, they were found to be extremely useful because they::

- Focused on the development of core features

- Kept the design as simplistic as possible
- Clarified and defined the visual scope of the requirements

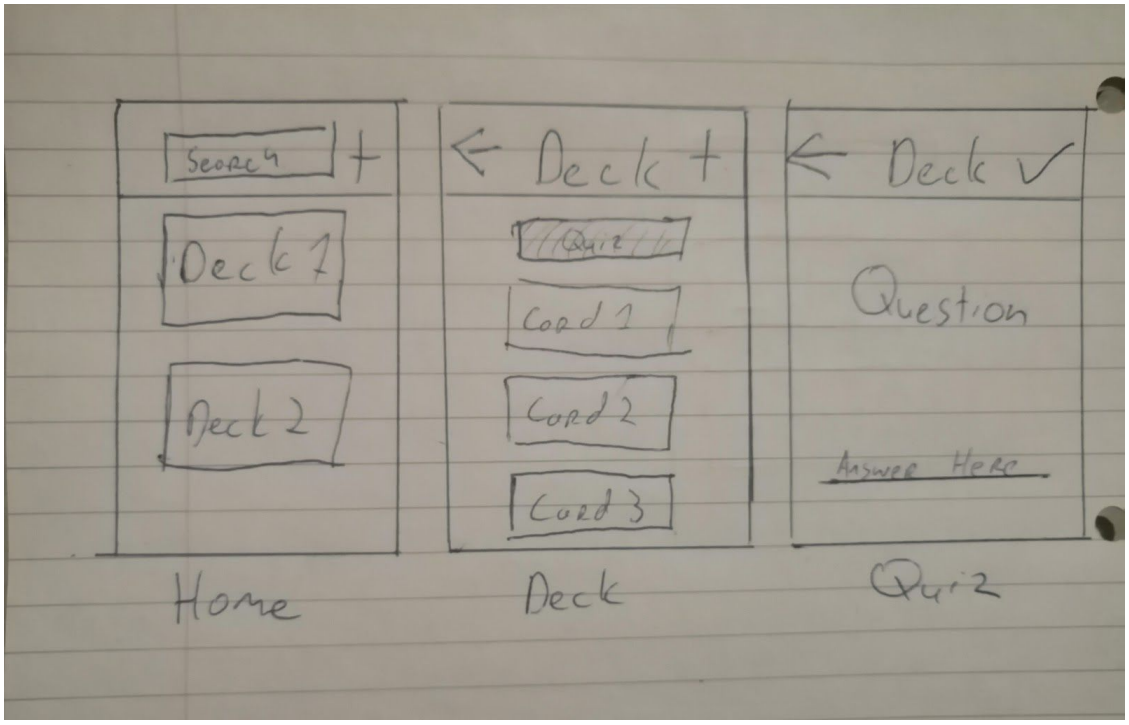


Fig 3.7 Low Fidelity Prototypes

These sketches were then transformed into high-fidelity prototypes (Figures 3.8-3.10) through the use of an online service called Marvel App. This was done to specify a more accurate presentation of what each page of the app would look like when the development process is finished. The previous components of Home and Card were developed further, while the design of the flashcard component were split into two new flashcard types:

- **Traditional Flashcards:** A double sided flashcard with the question presented in red text on the face of the card and the answer presented in green text on the back of the card. The default view presented on the screen of the traditional flashcard component, displays a view in which the flashcard's face is shown and the back is hidden. It's only when the face is pressed that the flashcard is flipped to display a view showing its back and hiding its face.

- **Written Flashcards:** A written input card, which has the exact same layout as the traditional flashcards except it has the addition of a text input field on the bottom of the component in which the answer can be entered

Moreover, the add card page and results page were both initially created during this stage as well and can be viewed along with the other high fidelity prototypes in Appendix 1

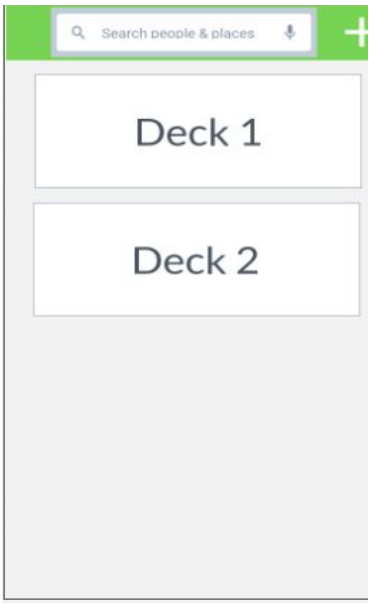


Fig 3.8 High Fidelity Prototype - Home Page

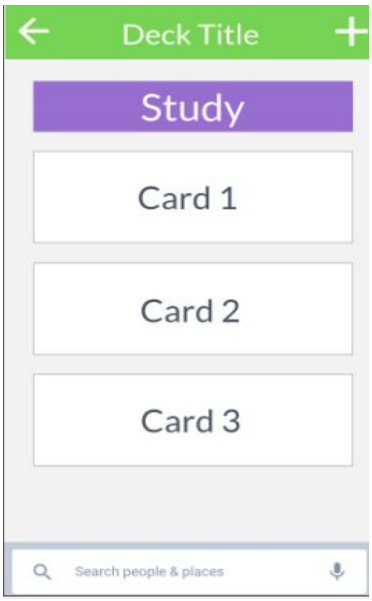


Fig 3.9 High Fidelity Prototype - Card Page

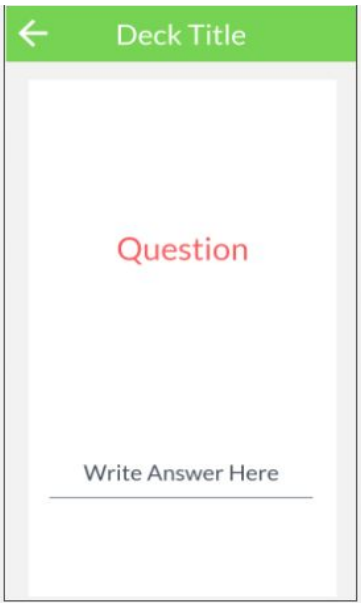


Fig 3.10 High Fidelity Prototype - Quiz Page

3.5 User Interface Design

The user interface (UI) is always the first layer created and implemented during each stage of the incremental model and every page's UI was iteratively edited and steadily improved to enhance the app's overall design and layout. The following design goals were outlined for the user interface at the beginning of the project:

- Users would find the application simple and easy to use
- The interface would be intuitive and not require much or any instruction on how to be used
- Users are able to move through the app easily and find it simple to navigate between decks and cards
- Decks and cards should be easily edited and/or deleted
- The quiz component of the app should be fun and engaging by incorporating the use of a swipe function that allows the user to progress through cards when taking the quiz
- The UI gives feedback to users in the form of their percentage score after the quiz's completion

- They should always have the option of returning to the previous screen as to encourage exploration through the app when first being used

The user interface was designed to mainly appeal to college students aged 18-25 and this resulted in creating an app that was both colourful and engaging while also maintaining a professional and pragmatic layout. The result was a bright and vibrant app that utilised many interactive features such as the swipe and slide functionalities in addition to utilising flip and show animations. However, the information was still presented in quite a layered and structured formal framework to promote a consistent and recursive learning environment in which students could flourish.

The human factors aspect of the app also placed a lot of emphasis on minimizing user error by not allowing the user to:

- Leave an answer or question to a flashcard empty
- Enter in a flashcard question that already exists in the deck
- Have two decks with the same name
- Receive a prompt before deleting a deck or flashcard

3.5.1 Home Page

When users open the app they are presented with the Home Page. The body of the page consists of a FlastList component with all the deck objects that the user has created so far and the number of cards contained in each deck. Each deck object in this FlatList component is a Touchable Opacity. A Touchable Opacity can be defined as a component wrapped around other Views to make them respond appropriately to touches or clicks by editing its *onPress* functionality. In this case, when the deck object is pressed it will navigate the app to the Card Page for the selected deck (see figure 3.11). These deck objects can be slid to the left to open a menu option where the user can choose to either edit or delete the deck (see figure 3.12). At the top of the home page there is a navigation bar which has two main components. It has a search bar which allows the user look for decks contained in the list. The other component is a plus icon which is a Touchable Opacity that when pressed takes the user to a new page with a single text input line in which they can enter a name to create a new deck.

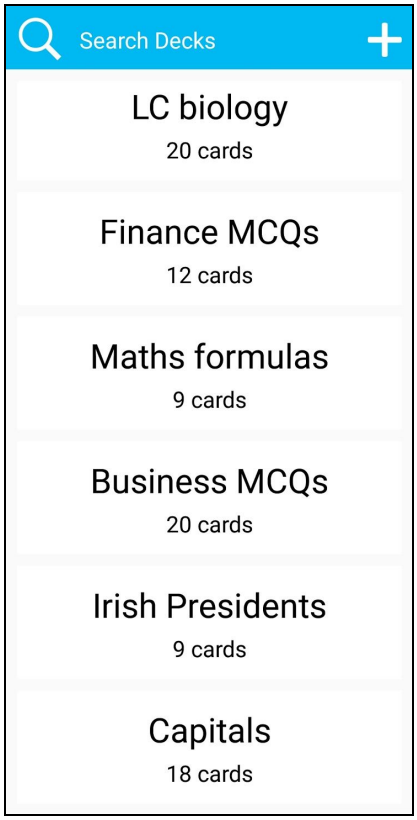


Fig 3.11 Home Page

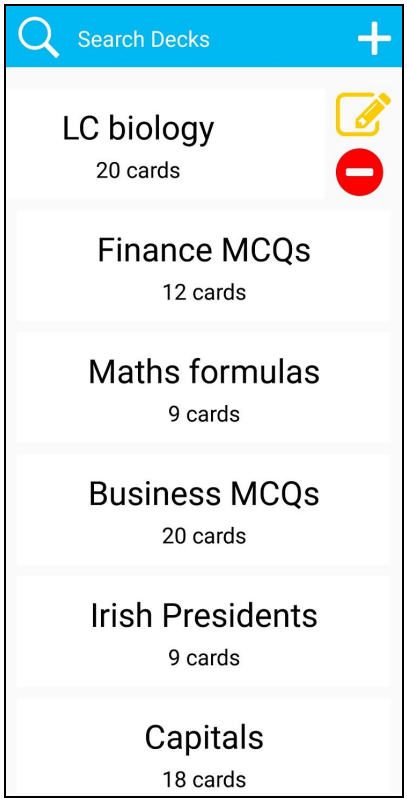


Fig 3.12 Home Page with Edit/Delete

3.5.2 Card Page

The body of the Card Page, in a similar and consistent design to the Home Page, contains a FlatList component of all the cards contained in the chosen deck. These card objects are Touchable Opacities which when clicked will show the user the chosen individual virtual flashcard which they can singularly inspect and examine (see figure 3.13). Likewise to the deck objects, when slid to the left the card object presents a menu option to either edit or delete the card (see figure 3.14). Above this list component, is an orange ‘QUIZ’ button which navigates the user to quiz page where the user will be tested on their knowledge of all the cards contained within the selected deck. At the top of the page is a navigation bar which is made up of three components. A back arrow icon which returns the user to home page. A text label which states the title of the deck and in brackets the number of cards contained in the selected deck. And finally a variation of a plus icon, which navigates the user to the Add Card Page where they can place a new flashcard into the chosen deck. The bottom tab bar comprises of a search bar which lets the user look for cards contained in the deck in the form of a text input function.

Fig 3.13 Card Page

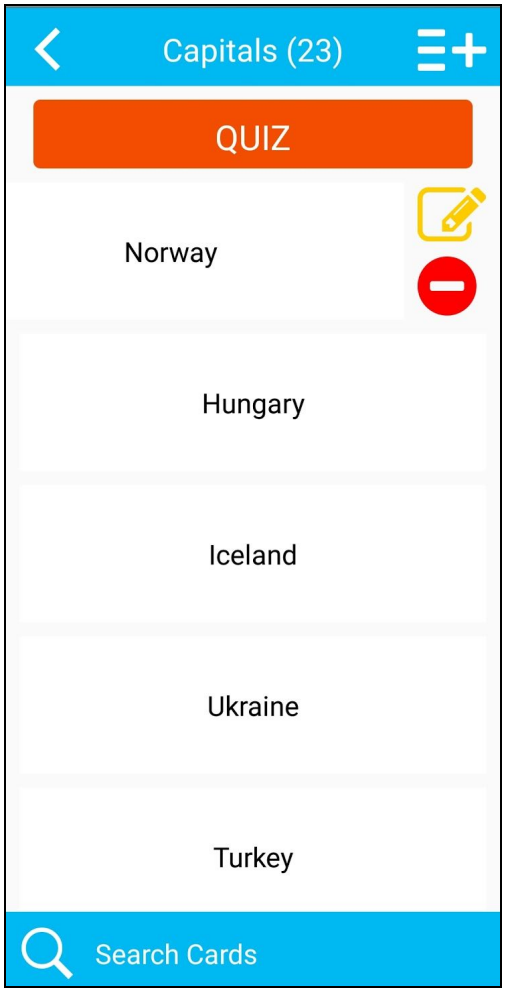


Fig 3.14 Card Page with Edit/Delete

3.5.3 Add Card Page

When adding a card to the deck, the user will be presented with the screen shown in figure 3.15. The body of the page has two main sections. The first section consists of a single line and multiline TextInput component in which the flashcard's question and answer are to be entered. This is indicated by the utilisation of placeholder texts that state *'Question'* and *'Answer'*. The other section is made up of two buttons, the *'traditional'* flashcard button represented by the speech bubble icon[1] and the *'written'* flashcard button represented by the pencil. These buttons are used to determine the answer method for the flashcard. A non-selected button will have an orange background and a white icon and border, while a selected button will have a white background and an orange icon and border. Both sections are wrapped in a KeyboardAvoidingView component which means the padding between them closes so they can be seen when

[1] Traditionally flashcards were spoken out loud while studying them which is why is represented by a speech bubble

the app uses the keyboard as displayed in figure 3.16. The navigation bar is made up of three components. A back arrow icon which is a Touchable Opacity that when pressed returns the user to the Card Page. A text label stating the deck title. And a tick symbol that is a Touchable Opacity component the user clicks when they're finished creating the card to add it to the deck. A card cannot be added to the deck if the question already exists in the deck or if any of the input text fields are empty.

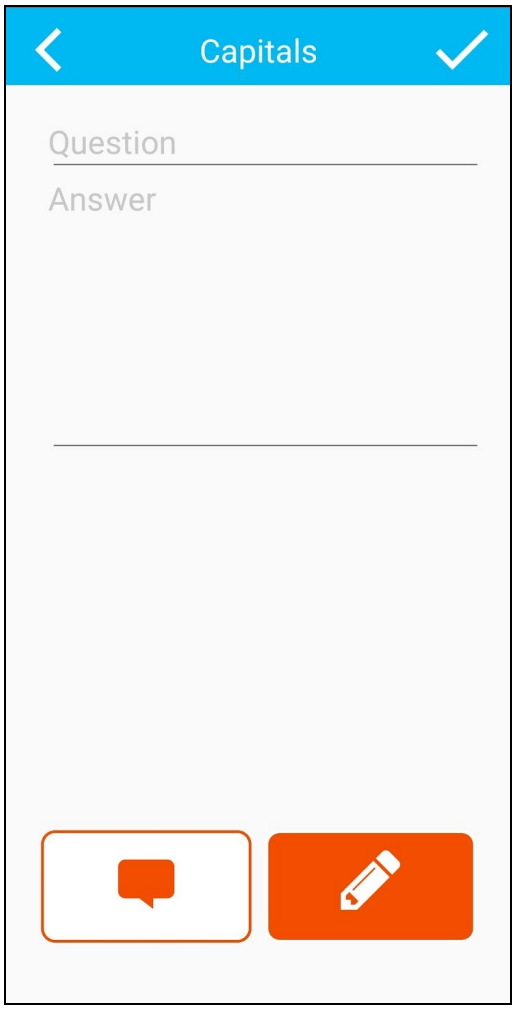


Fig 3.15 Add Card Page

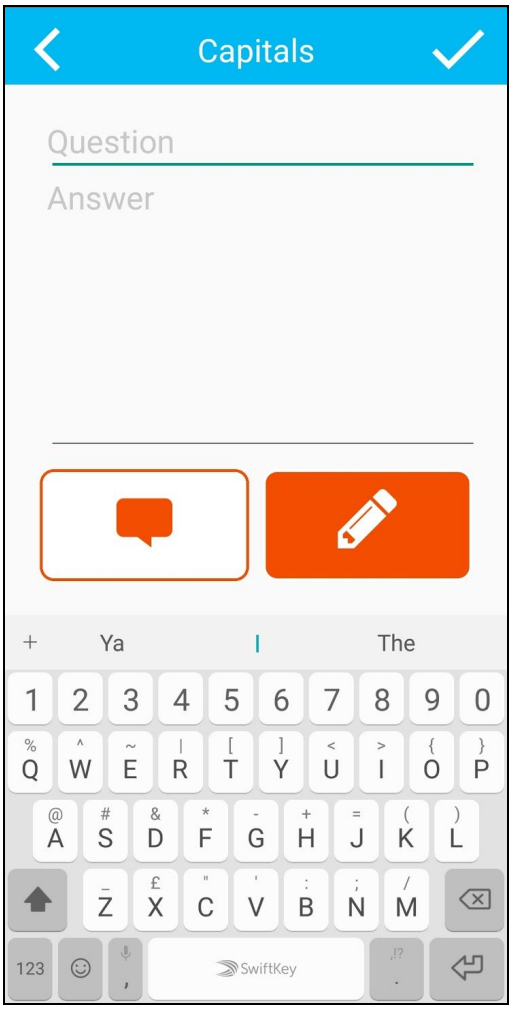


Fig 3.16 Add Card Page with Keyboard

3.5.4 Flashcards

As discussed in section 3.4, there exists two forms of flashcard designs:

- **Traditional:** This is comprised of a FlipCard component made up of two parts, a *face* and a *back*. The face is a FlatCard component containing a standard Text View label of the question in red (see figure 3.17) and the back is a FlatCard component containing a standard Text View label of the answer in green (see

figure 3.18). The default View presented on the screen of the traditional flashcard component, displays a View in which the flashcard's *face* is shown and the *back* is hidden. The *face* is a Touchable Opacity component that when clicked performs a flip animation to show the *back* and hide the *face* of the flashcard. The reverse is also true. The *back* is a Touchable Opacity component that when clicked performs a flip animation to show the *face* and hide the *back* of the flashcard. The *face* and *back* can be continuously clicked to toggle between being hidden and shown.

Fig 3.17 Traditional FlashCard Face

Fig 3.18 Traditional FlashCard Back

- **Written:** The design for the *'written'* is almost identical to the *'traditional'* one. It has two sections. The first section has the exact same FlipCard component as the *'traditional'* flashcard with the question in red (see figure 3.19) and the answer in green(see figure 3.20). The second section has a TextInput component in which the answer can be entered into and an orange box with a white tick icon

which is a Touchable Opacity component that when clicked submits the answer. This flashcard is wrapped in a KeyboardAvoidingView component so it can be seen when the keyboard is being used.

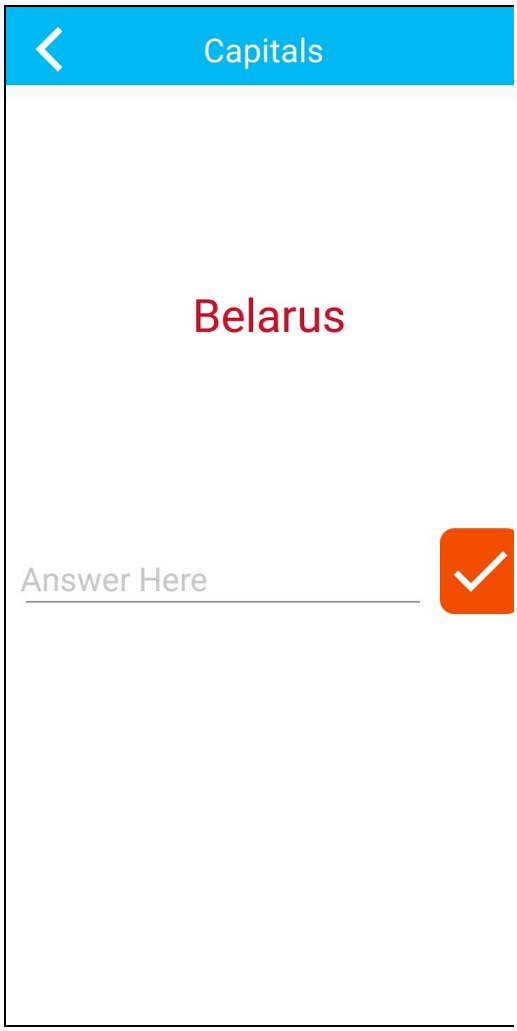


Fig 3.19 Written FlashCard Face

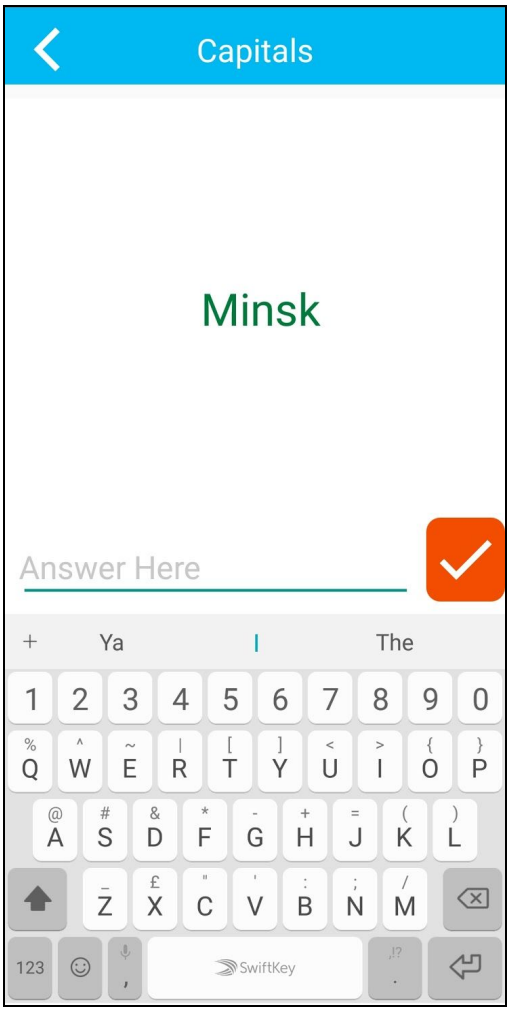


Fig 3.20 Written FlashCard Back with Keyboard

3.5.5 Quiz Page

The body displays a *'traditional'* or *'written'* flashcard based on which button the user has selected in the Add Card Page. The entire body of the Quiz Page is wrapped in a Swiper component which enables cards to be swiped either left or right. In terms of how to answer questions during the quiz there are two methods:

- **Traditional**: The user guesses their answer by saying it aloud or thinking it in their mind. Following this the flashcard is then flipped by pressing the *face's* Touchable Opacity component and the said or thought answer is compared with the actual answer on the back of the flashcard. If the said answer is correct and matches the actual answer, the user swipes the screen to the right (see figure 3.22). If the said or thought answer is incorrect the user swipes the screen to the left (see figure 3.21). This answering method mimics the way traditional flashcards are used and it similarly relies on the user being truthful because the system has absolutely no way to verify if the said or thought answer actually does match the correct one.

- **Written**: The answer would be submitted by entering it into the text input field. The system will then compare this inputted text with the flashcard's actual answer. If the inputted text is correct and they match character-by-character, the flashcard component is automatically and immediately swiped to the right and the user's quiz score is increased by one. If the inputted text is incorrect and it doesn't match the actual answer, the flashcard is flipped to display the *back* of the flashcard where the answer is displayed for 500 miliseconds. Once this time has elapsed, the flashcard component is automatically swiped to the left. Unlike the '*traditional'* answer method this system does have a way to verify the inputted answer and doesn't rely on the user being truthful.

Furthermore, the system allows for human error in the text input fields by ignoring capitalisation and trimming any blank spaces and punctuation surrounding the entered text input. Additionally the *'written'* answer method has an accelerator for more experienced users. If they think they know the answer and don't want to enter it into the text field, they can simply choose to answer the flashcard using the *'traditional'* answer method i.e saying the answer, checking it and swiping left or right appropriately. When viewing a flashcard object during the quiz, it will appear as it does in figure 3.21. It is is identical to the single flashcard view (see figures 3.17 to 3.18) except it has an additional Text View on the top right of the flashcard which lets the user know their score so far in the quiz. This text label is dynamically updated by the system every time the user swipes the flashcard to the right to indicate to the system that he said or thought the correct answer. When swiping the card left or right, the system provides feedback in the form of Text Overlay labels. When swiping left, the top left show a red Text Overlay label stating "NOT YET" indicating that the answer is incorrect (see Figure 3.22). When swiping right, the top right shows a green text overlay label stating "GOT IT" indicating that the answer is incorrect (see Figure 3.23). As these overlays get closer to their respectable edges their opacity gets higher and they fill more with colour. Once a swipe is performed, it is permanently recorded by the system and it cannot be reversed.The top of the page has a blue navigation bar which contains two components, a back arrow icon that when clicked navigates back to the card page and a text label specifying the name of the deck.

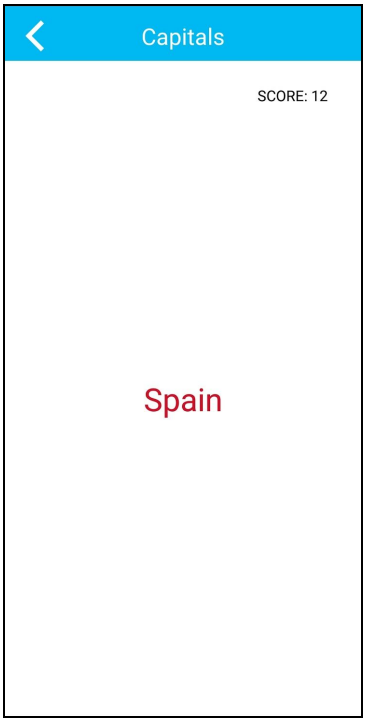


Fig 3.21 Standard Card View During Quiz

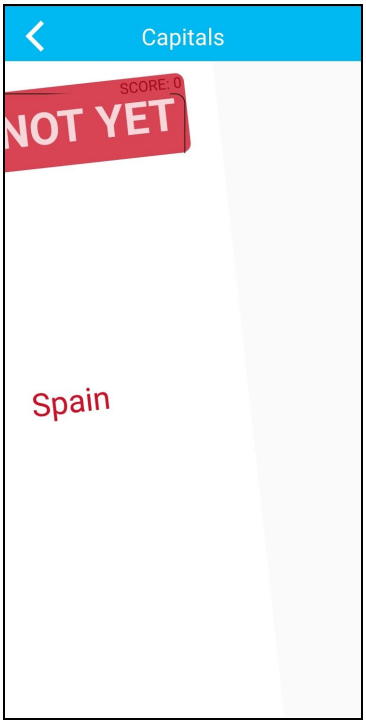


Fig 3.22 Quiz Page Swipe Left

Fig 3.23 Quiz Page Swipe Right

3.5.6 Results Page

The Results Page is only shown once all the questions in the quiz are answered. If the quiz is exited before completion, the answer history of the quizzed questions is recorded but the user will not be navigated to this Results Page and instead will return to the Card Page. The user cannot keep their progress and continue their exited quiz from where they left. They must start a new one to complete it successfully. The body of the page displays the percentage score achieved in the quiz and a circular coloured bar that is filled up based on the received percentage. If the score isn't 100%, the colour of the score and circular bar is red and a button allowing the user to take the quiz will be presented on the screen (see figure 3.24). If the score achieved is 100%, the colour of this score and circular bar is green (see figure 3.25). The navigation bar is comprised of a back arrow icon that is a Touchable Opacity component that when pressed will return the app to the Card Page and a standard Text View label of the deck name and the number of questions answered in the quiz.

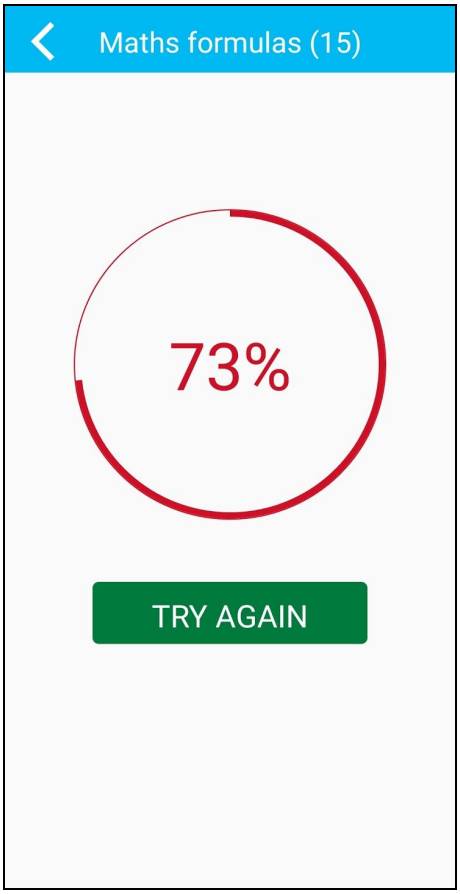


Fig 3.24 Results Page

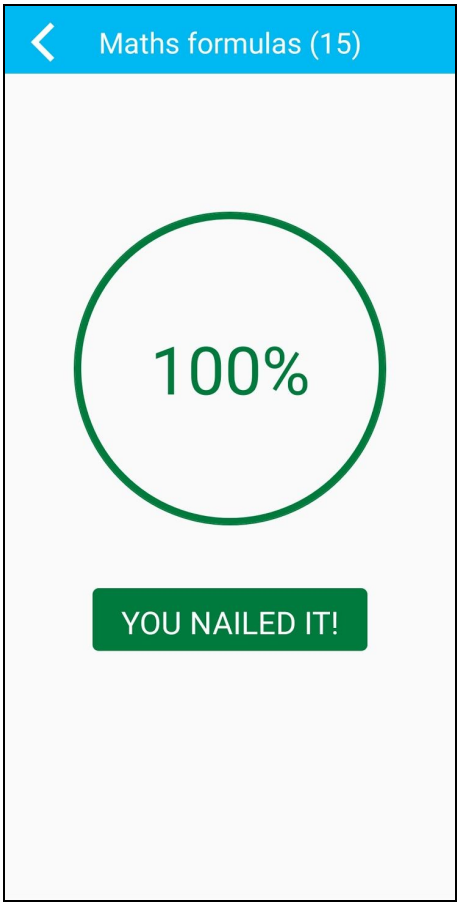


Fig 3.25 Results Page

3.6 System Design

This part of the report will now discuss the design of the three main functions of the app using Sequence and Use Case diagrams. Employing the use of the incremental model, each of the functions was developed individually as key part of a core component, and operate completely independently of each other.

3.6.1 Add Deck

The adding of a deck was the first fundamental function created for the app. This was the most logical functionality to develop first as adding the card and quizzing were both dependent on having a deck object to be added to. Using the iterative steps of the incremental model, the user interface was constructed before moving on to the application tier.

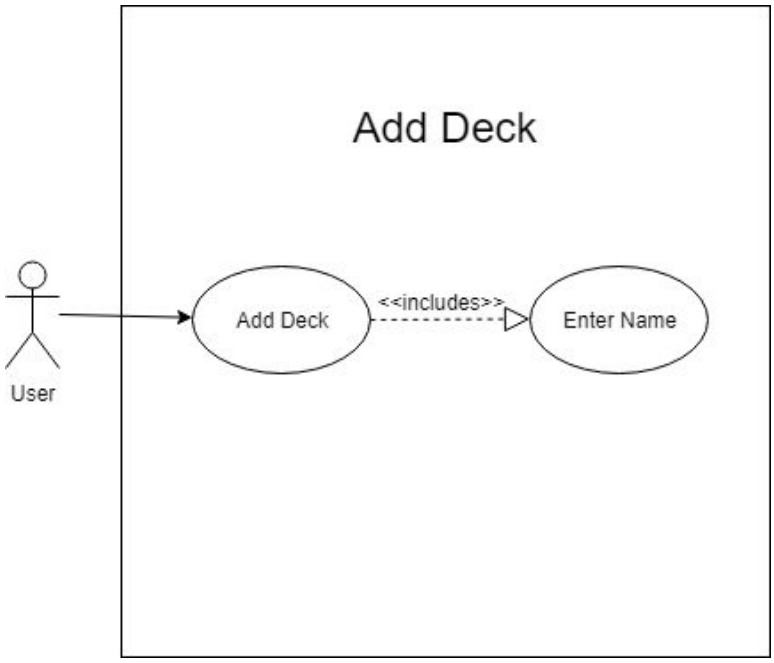


Fig 3.26 Use Case Diagram Add Deck

As shown in the Use Case Diagram (see figure 3.26), the user is only presented with one option in the Add Deck page. This is entering the name of the new deck. The function was designed this way, to limit the information to specifically just what was needed to create an identification key for the deck and to keep the action as simple as possible as to avoid any errors happening. As the name of the deck is an identification key used to locate it in the phone's storage, users are restricted to not creating decks of the same name to prevent duplication occuring in the back-end of the app.

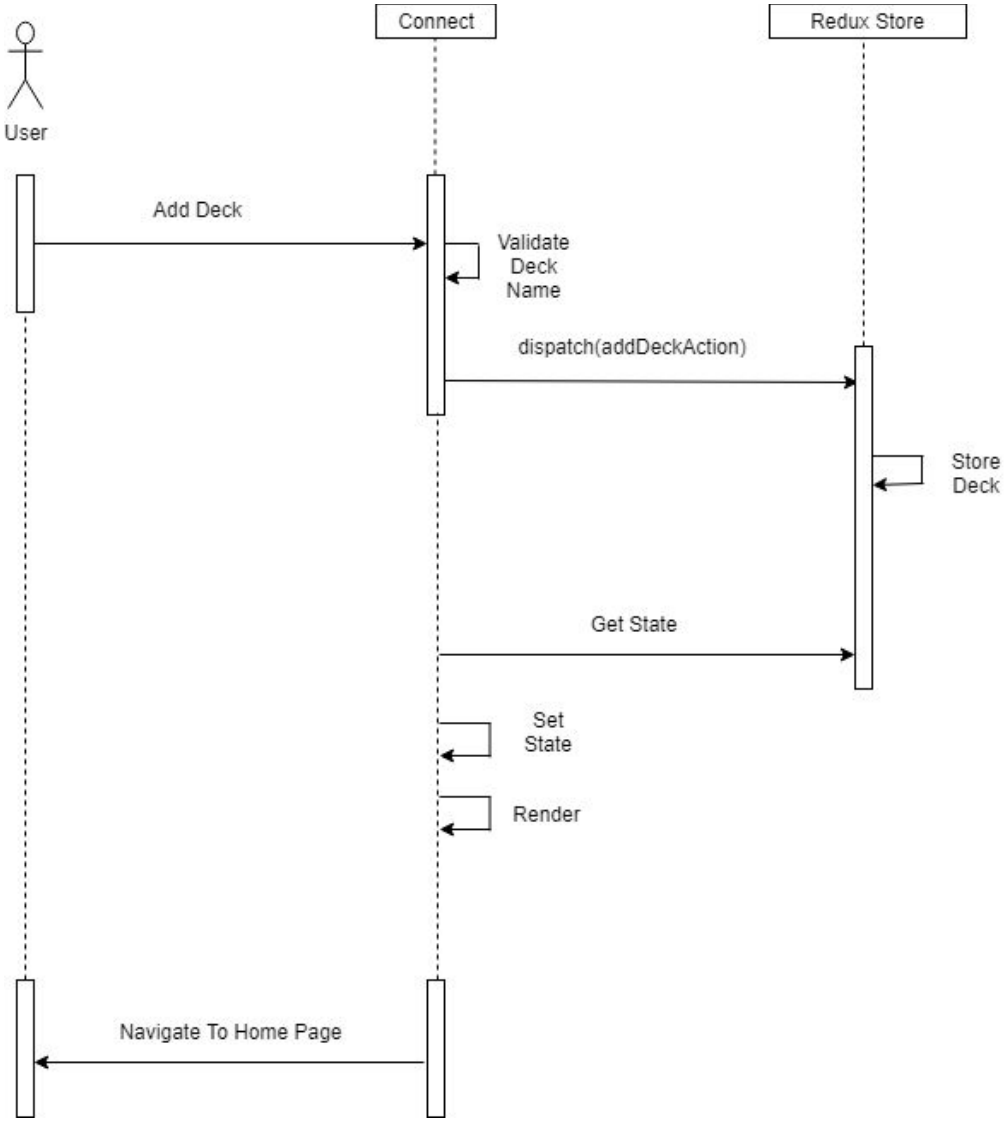


Fig 3.27 Sequence Diagram Add Deck

The Sequence Diagram (see figure 3.27) displays the series of events that occur when the name of the deck is entered and the submission button is clicked on the Add Deck Page. The text entry is firstly validated by the application tier to ensure duplication doesn't occur. When confirmed, the addDeckAction is dispatched to the Redux backend and the deck is saved in the async storage of the phone. The state of the component is then retrieved and set by functions in the objects and controllers layer. When the state of a component is set or reset in a React Native app, it always results in a re-rendering of the screen. This is what takes place in this bridge section of the app. However if the action is successful, the app is then instantly navigated back to the Home page where the new deck object is now present.

3.6.2 Add Card

After developing the add deck functionality, the action of adding a card to a deck was implemented. It was designed in such a way that it maintained a comparable look and feel with the Add Deck Page to promote consistency within the app and to simplify all actions for the user.

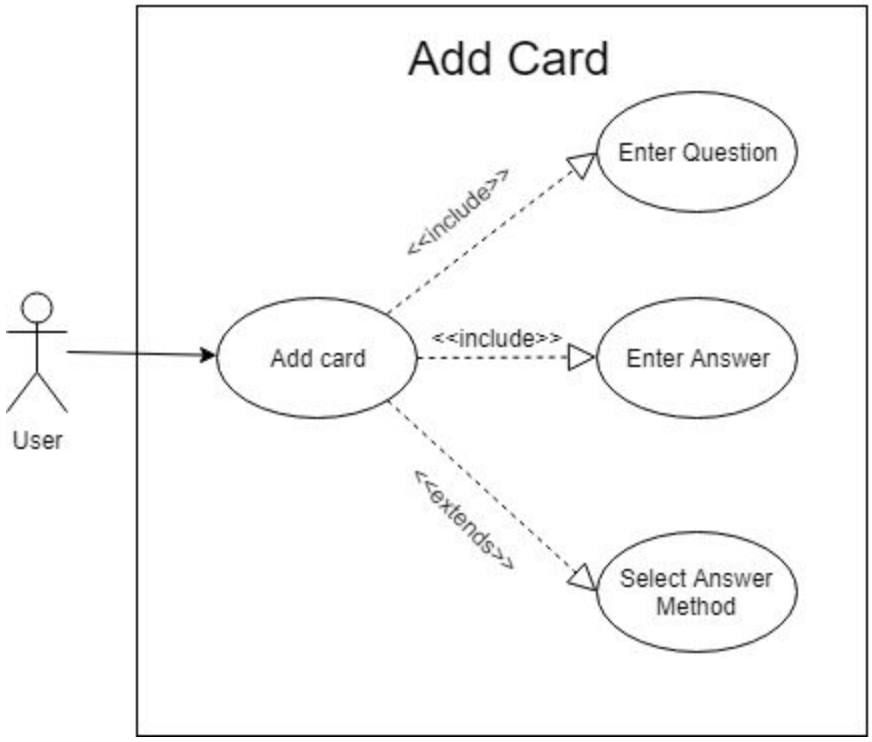


Fig 3.28 Use Case Diagram Add Card

The Use Case diagram (see figure 3.28) for the Add Card function is shown as a structured and straight forward set of actions to follow. The reasoning for this, is to once again maintain simplicity and to restrict the user from doing anything that the system isn't expecting. The function starts by the user inputting the question they would like to be asked in the first text input field on the Add Card page. Similar to the deck name, the question is used as an identification key to locate the flash card object in the phone's storage, meaning that a question cannot be duplicated in the same deck. Following this, the answer to the flashcard will be entered into a multiline text input. These two fields cannot be empty for the card to be successfully added to the deck. Finally, the user has the option to change the method of how they would like to answer the card. The options are the *'traditional'* method or the *'written'* method, with the initial default choice being set to the '*traditional'* method.

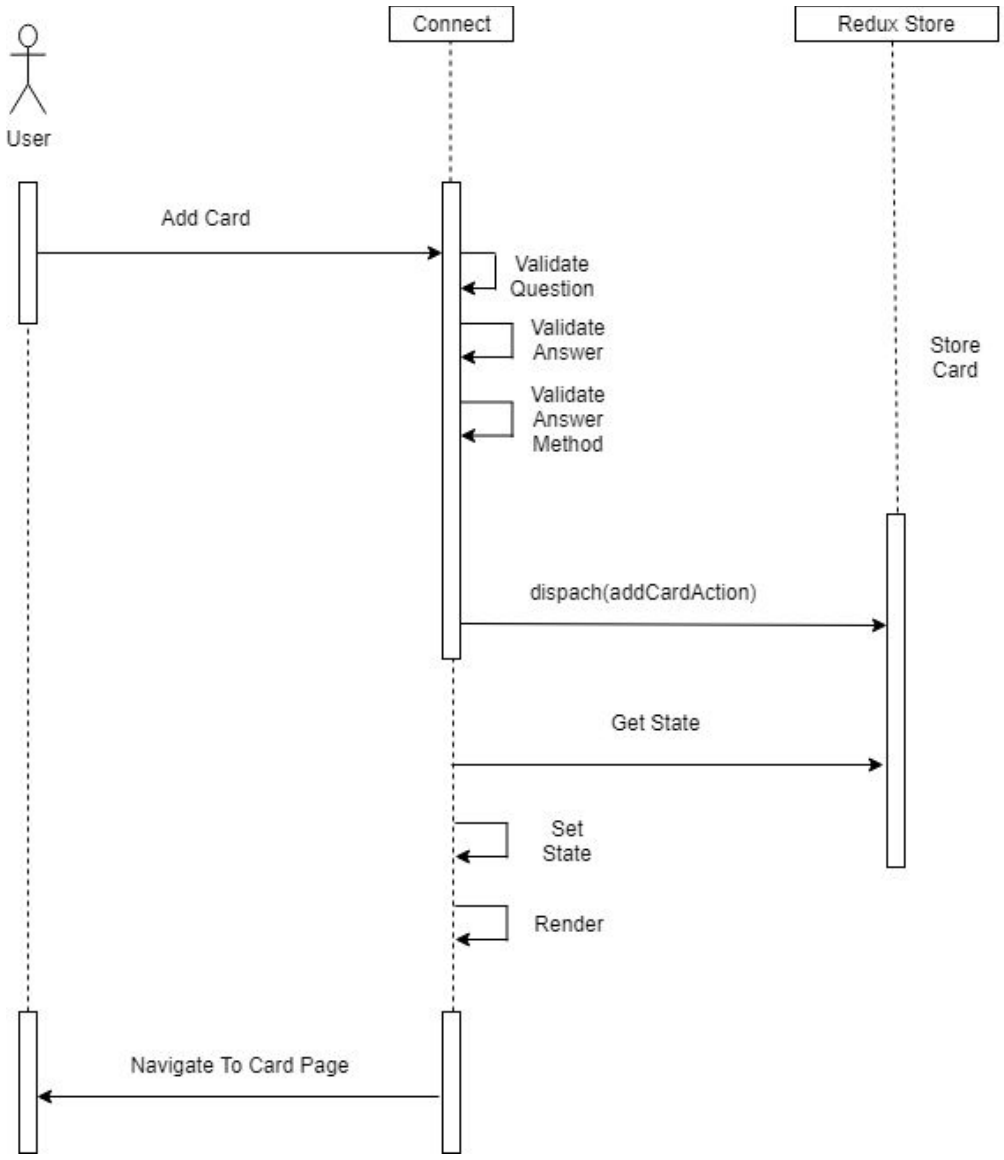


Fig 3.29 Sequence Diagram Add Card

When the card is successfully submitted, it triggers the sequence of events shown in the diagram above (see figure 3.29). Before dispatching the addCardAction to the Redux back-end, the question, answer and answer method must all be validated by the application tier. This dispatch action contains all the necessary information for Redux to store the card in the appropriate deck in the async storage. The state of the component is then updated and re-rendered. If all the actions are done correctly, the app is returned to the Card Page where the newly added flashcard can be viewed by the user.

3.6.3 Quiz

The last of the three major functions that was implemented in the development process, was the Quiz functionality. Construction of the function began once all other functionalities were complete and tested to a reasonable level. It operates quite differently to the other two main functions but still preserves the same look, feel and design consistency.

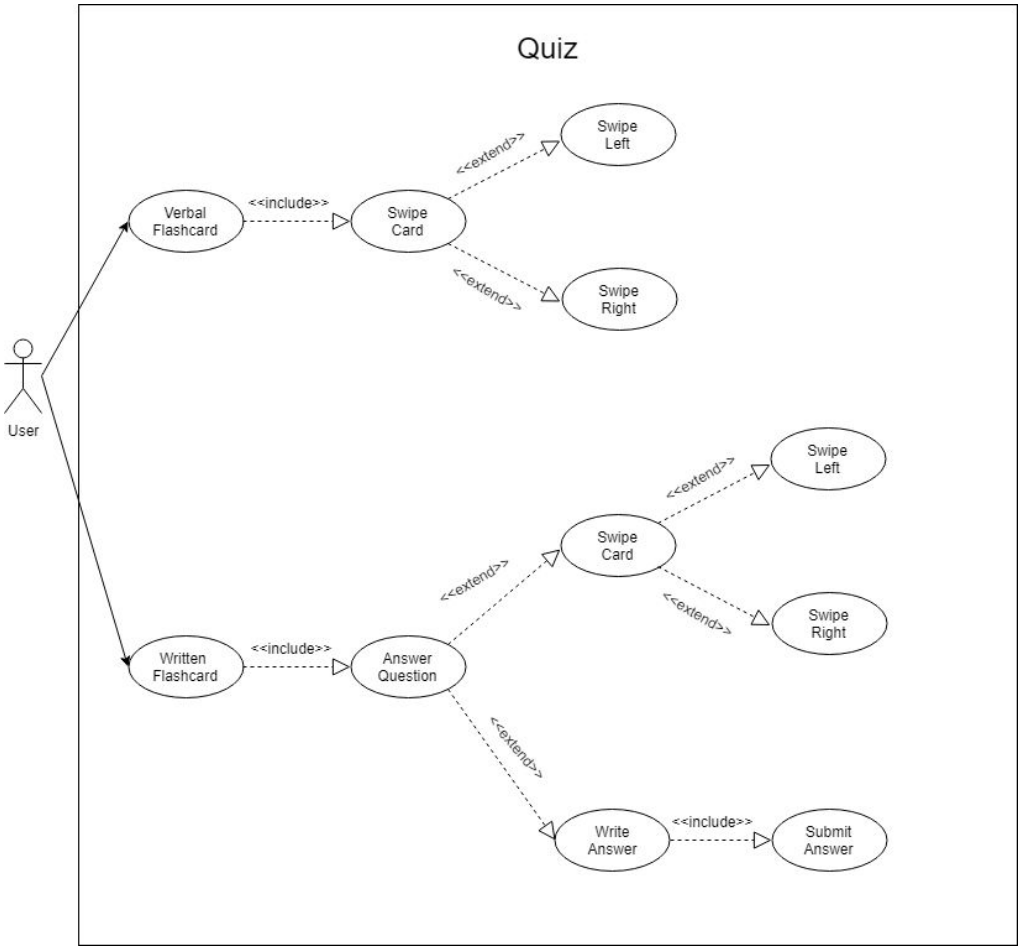


Fig 3.30 Use Case Diagram Quiz

The Use Case diagram (see figure 3.30) shows how a user can be presented with two answer formats during the quiz:

- **Traditional Format:** If the *'traditional'* format is shown to the user, they must swipe the card left or right to advance onto the next card or complete the quiz (see figure 3.28).
- **Written Format:** If the *'written'* format is shown to the user, they have two options to advance onto the next card or complete the quiz. They can either swipe the card left or right or they can enter an answer in the form of a text input which will be compared with the answer of the flashcard.

The answer text field can be empty to advance onto the next question, but it will be marked wrong in the quiz (see Figure 3.28).

The Sequence diagram (see figure 3.31) for the quiz action was separated into two distinct series of events:

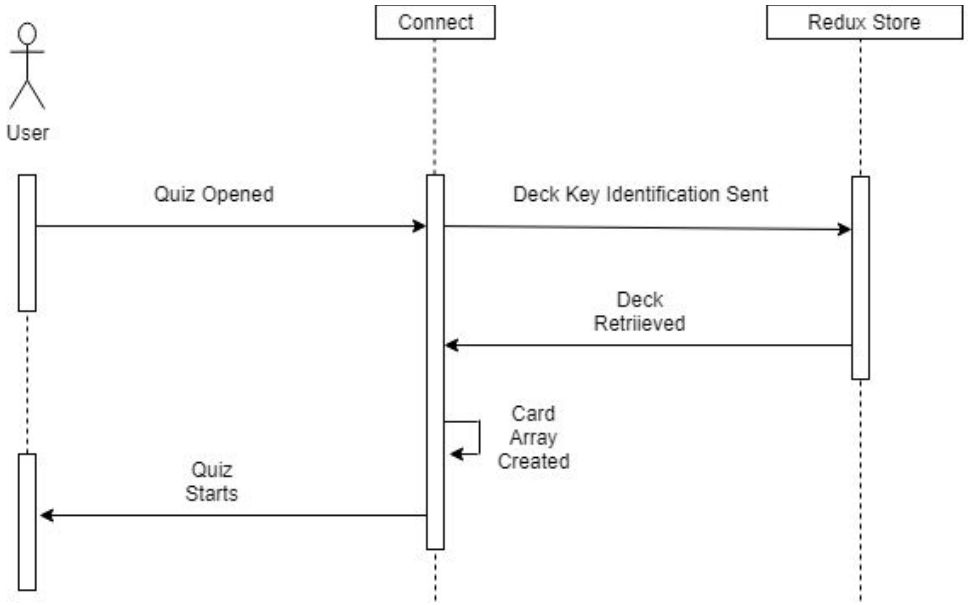


Fig 3.31 Sequence Diagram Quiz: Retrieve Deck

The first sequence consists of the deck identification being sent to the back-end to retrieve the deck. Once this is retrieved, a card array containing all the flashcards is created. When this is completed the quiz can begin.

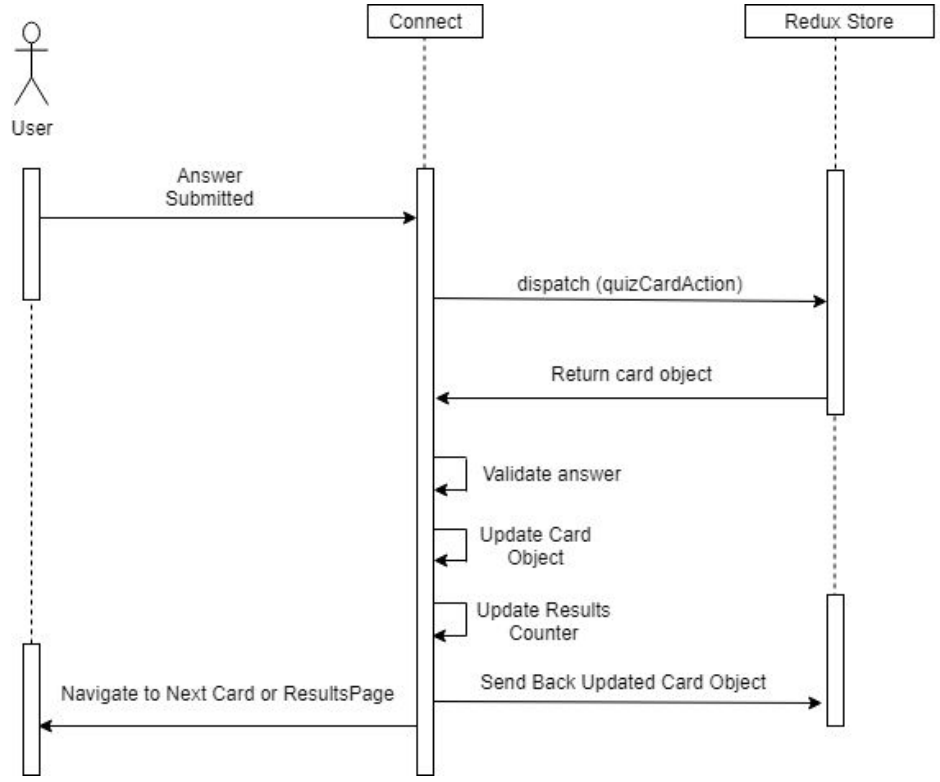


Fig 3.32 Sequence Diagram Quiz: Show Flashcard

The second Sequence diagram (see figure 3.32) occurs directly after the first and it illustrates how the flashcards are handled during the quizzing process. The answer is submitted in the form of a swipe or text input from the card. Following that, the quizCardAction is dispatched and the card object is retrieved from the Redux Store. The answer submitted by the user is then compared with the card object's answer to verify if it's correct or incorrect. After this verification, the card object's history and the results counter are updated accordingly and the altered card object is sent back to the phone storage. As the state of the component doesn't change by using the setState function, it isn't re-rendered. Why this is important will be discussed further on in section 5.5.3. When all these actions are completed the app is then navigated to the next card or the results page depending on if all the flashcards in the card array have been utilised. If they have and all the questions have been completed then the app is navigated to the app page.

3.7 Conclusion On Design

To summarise, using the information gained from chapter 2, ideas and solutions to possible problems were formulated and constructed and the author felt the system employed a highly usable and well thought out design that adhered to the scope of

requirements. Overall, the author was very happy with the layout of the app and thought not only did it provided a good framework for the functions to be implemented.

Chapter 4: Implementation

4.1 Choosing Technologies

Before the system was implemented, there were several choices that needed to be made regarding the technologies that would be used.

4.1.1 React Native

React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It's based on React, Facebook's JavaScript library for building user interfaces, specifically for mobile platforms. The reason I chose React Native over simply creating directly for Android or IOS can be outlined by these 3 main reasons:

- **Cross-Platform Development:** The main reason react native was chosen is that it supports cross-platform development. It has the ability to develop for both mobile operating systems and web development with just a single development language. Essentially the framework integrates the advantages of mobile app development with the native React's environment adaptability and power to construct a flexible app that can be run a variety of devices successfully.

- **Reusable Components**: React Native's building blocks are reusable components that compile directly to native. Components used in iOS or Android development have corresponding counterparts in React JS and it abides by the WORA principle of "Write Once, Run Anywhere". This allowed for a consistent look and feel to be achieved across all these platforms without needing to write endless amounts of code. This component-based structure also allowed the app to be built with a more systematic and agile approach to development rather than other hybrid frameworks.

- **Live Reload Feature:** The React Native development process allows for a live reloading feature which displays the updated UI content instantly whenever any script or code is saved. This significantly speeds up the development time, and worked particularly well in conjunction with the incremental model during the app creation process because being able to instantly view your code output and how it's affected by recent programming changes is extremely important when using this development model.

- **Speed and Performance:** React Native is light, extremely quick to work with, and does not consume many resources. The end product of the React Native app was fast, smooth and only 19mb in size. It was completed with very little performance-tweaking and the app makes full use of the GPU (Graphics Processing Unit), unlike native development, which is more CPU (Computer Processing Unit) intensive.When conducting research in section 2, it was found that one of the reasons students didn't use a mobile flashcard app was because it drained the battery on their phone and took up a large amount of storage space. Consequently, by using React Native it minimised these impacts by the app.

4.1.2 Redux

In React Native applications, there exists a unidirectional flow of information from parent components to child components. It cannot occur the other way around. This occurs in the form of props being sent to child nodes and using them in their functions. This can become very difficult when the application becomes too large to manage in the state of each component. The JavasScript library of Redux was used to solve this issue and act as a single large store for all the updated states of each component in the application. There exists three main aspects of Redux:

- **Actions:** Whatever process or function that is being used to update the component's state. These are created in a specific format for Redux.

- **Dispatch:** The operation of sending the action to the reducer.

- **Reducer**: Looks at the action and accordingly chooses what it needs to do to save the data in the store. At its core, the reducer is nothing but a file consisting of switch case statement and used for storing the data in a store and returning the updated state value from the store. So whenever the state is updated the value in the store gets updated too.

Redux proved very useful in:

- Keeping the back-end of the app simple, comprehensible and easy to maintain.
- Maintaining predictable state updates and assisting the author understand how the data flow works in React Native applications
- Employing ‘pure’ reducer functions which allows for the much easier testing of logic used on the state information and assists in the debugging process
- Centralising the state made it easier to implement logging changes to the information and the passing of data between pages and page refreshes

4.1.3 Marvel App

Marvel App is an online design tool that provides simple prototype development. It allows for the creation of screens directly on the Marvel website using a range of images from their collection. Images can also be imported from software such as Photoshop or Canva to be used when designing prototype templates. These prototype templates could then be exported to Dropbox and Google Drive. The main reason Marvel was utilised in the development process is that allows users to link the screens together. Gestures and actions indicating transitions between pages can then be added to make a prototype feel like it’s a real app that can be freely navigated. This was a great way to examine and review the flow and feel of the app without having to fully commit to developing all the processes and functions of an operational version. Marvel, and other tools like it, allow users to avoid the problem of sinking a huge amount of time into the app development with a design that doesn’t suit the project.

4.1.4 Expo

Expo is a toolchain library built around React Native that assists in the rapid and agile development of an app by providing third party user interface components and services in the form of node modules that can be quickly implemented. It also provides an Expo CLI development server that can run on a browser for a quick testing of the app via the scanning of a QR code by the Expo Client app or by emailing the developer a direct link to the JavaScript bundle on the development server.

4.1.5 Atom

Atom is the text and source code editor used to write the React Native app. It was chosen for its customizable view and its ability to support third party plugins such as EsLint that assisted in the development of the app.

4.1.6 EsLint

EsLint is a JavaScript linting utility plugin for Atom used to find problematic patterns of code that don't adhere to a pre-approved set of guidelines set by the plugin. It made the maintenance of code consistency and the identification of errors in the code considerably easier throughout the development process by highlighting these issues in red.

4.1.7 Github

GitHub is an online repository service for source code of a variety of programming languages. A GitHub repository was kept throughout the development of the project for version control and to mitigate the risk of data loss. Having implemented key functionalities, the author continuously pushed work to the master branch along with a brief description of changes made or updates made.

4.2 Feature Implementation

4.2.1 FlatList Implementation

A FlatList component is a performant interface for rendering simple 'flat lists', such as the lists used in to display the deck and card objects in sections 3.5.2 and 3.5.3. They are implemented in the project instead of normal list components for their additional functionality of having the props:

- **data:** This dictates what information is shown by the FlatList component.
- **renderItem:** This instructs the FlatList component on how this information is shown

4.2.1.2 FlatList Implementation On The Home Page

```
129          </View>
130
131          <FlatList
132            data={showingDecks}
133            renderItem={({ item }) => this.renderItem({ item, itemProps })}
134            keyExtractor={(item, index) => index}
135          />
136
137        </View>
```

Fig 4.1 Flatlist Component The Home Page

The body of the Home Page displays a flat list of all the current existing decks in the phone's storage by retrieving an array of deck objects from the phone's storage and passing them into the FlatList component prop of data (see line 132 in figure 4.1). If no deck objects have been created by the app, the FlatList component will be left empty. It then iterates through this array of deck objects contained in the data prop , identifying the necessary and relevant deck information to displays them within the FlatList component on the Home Page using the renderItem function (See figures 3.11 & 4.1).

```
25      renderItem = ({ item, itemProps }) => {
26        if (item.title !== null) {
27          return (
28            <Swipeable
29              style={styles.list}
30              onRightButtonsOpenRelease={itemProps.onOpen}
31              onRightButtonsCloseRelease={itemProps.onClose}
32              rightButtonWidth={60}
33              rightButtons={[
34 >              <View>⋯ {/* This contains the edit & Delete Functionalities
66                            Which can be opened when slid to the left*/}
67
68              </View>
69            ]}
70          >
71            <TouchableOpacity
72              style={styles.flatCard}
73              onPress={() => this.props.navigation.navigate(
74                'CardPage',
75                { deckTitle: item.title }
76              )}
77            >
78              <Text style={styles.deckTitle}>{item.title}</Text>
79              <Text style={styles.deckBody}>{item.questions.length} cards</Text>
80            </TouchableOpacity>
81          </Swipeable>
```

Fig 4.2 renderItem function On The Home Page

The renderItem function (shown in figure 4.2) works by first ensuring that the deck object title isn't null and that it exists and contains the appropriate information. As shown on lines 68 to 77, it returns a Touchable Opacity component that when clicked navigates the app to the Card Page and passes the prop of deckTitle into the navigation function to be used as an identification key when locating the deck in the phone's storage. The Text Views contained in the Touchable Opacity component generate their Text View labels from the deck props that were passed into the renderItem function (see itemProps on line 15 of figure 3.2) and show the title of the deck, as well as the number of cards it contains (see figure 3.11). The Touchable Opacity component is contained in a Swipeable wrapper component to allow a sub menu of edit and delete buttons to be opened and closed by sliding the FlatList component to the left and right. This will be discussed further when looking at the slide function in section 4.2.1.3.

4.2.2 FlatList Implementation On The Card Page

```
137           <FlatList
138             data={showingCards}
139             renderItem={({ item }) =>
140              this.renderItem({ item, itemProps, deckTitle: deck.title })}
141             keyExtractor={(item, index) => index}
142           />
```

Fig 4.3 Flatlist Component The Card Page

The body of the Card Page, which shows all the flashcards contained in the deck (see figure 3.13), operates in a very similar manner. The FlatlList component in figure 4.3 is only differentiated to figure 4.1 in the additional action of passing of the prop deckTitle to the renderItem function. This is performed as the title of the deck is used as an identification key to locate the correct deck in the phone's storage, where all these flashcard objects are stored

The *renderItem* function on the Card Page (see figure 4.4) is also very much alike to the *renderItem* function on the Home Page (see figure 4.1). The Touchable Opacity component is correspondingly contained in a Swipeable wrapper component to allow for a sub menu of edit or delete to be opened or closed by sliding the flatlist component to the left or right. However, it is different in the way the Text View of the FlatList component contains a Text label of the flashcard's question (see line 66 in figure 4.4) rather than the title of the deck and the number of card it contains (see figure 3.3). Furthermore, when the component is clicked it navigates to the CardView page passing in the operation of *'individual'* to signify the viewing of a single flashcard. This will be analysed further in section 4.2.6.

```
renderItem = ({ item, itemProps, deckTitle }) => (
      <Swipeable
        style={styles.list}
        onRightButtonsOpenRelease={itemProps.onOpen}
        onRightButtonsCloseRelease={itemProps.onClose}
        rightButtonWidth={60}
        rightButtons={[
          <View>{/* This contains the edit & Delete Functionalities
                    Which can be opened when slid to the left*/}
          </View>
        ]}
      >
        <TouchableOpacity
          style={styles.flatCard}
          onPress={() => this.props.navigation.navigate(
            'CardView',
            { operation:
               'individual',
               deckTitle,
               card: { question: item.question, answer: item.answer } }
          )}
        >
          <Text style={styles.cardTitle}>{item.question}</Text>
        </TouchableOpacity>
      </Swipeable>
```

Fig 4.4 renderItem Function On The Card Page

4.2.2 Search Implementation

4.2.2.1 Search Implementation On The Home Page

The search box is a TextInput component (see top of figure 3.11) which determines its value from the global state of the query variable (shown in Figure 4.5). This query state begins as a blank string, but is updated as the user enters in characters into the Text Input component.

```
        <Search />
        <TextInput
          style={styles.search} placeholder='Search Decks'
          placeholderTextColor='#FFFFFF'
          value={query} underlineColorAndroid="transparent"
          onChangeText={(query) => { this.setState({ query }); }}// eslint-dis
        />
```

Fig 4.5 TextInput Component On The Home Page

This query is then used to filter though the deck objects, looking for any titles with the characters in the exact same order as the entered ones to display the relevant deck objects in a FlatList component on the body of the Home Page(see figure 3.11) using the

code shown in Figure 4.6. It achieves this by changing the FlatList component's prop of *data* (shown in figure 4.1) to the value of the variable *showingDecks* which determines which deck objects contain these characters and will be rendered by the function. If the query is equal to the default value of an empty string, all deck objects are shown in the FlatList components on the body of the page.

```
let showingDecks;
if (query) {
  const match = new RegExp(escapeRegExp(query), 'i');
  showingDecks = decks.filter((deck) => match.test(deck.title));
} else {
  showingDecks = decks;
}
```

Fig 4.6 Search Query On The Home Page

4.2.2.2 Search Implementation On The Card Page

The search box on the Card Page is also a TextInput component (see Figure 4.7) that operates in the exact same manner. The only discrepancy is that It contains a few minimal stylistic differences (see bottom of figure 3.13).

```
<TextInput
  style={styles.search} placeholder='Search Cards' placeholderTextColor='#FFFFFF'
  underlineColorAndroid="transparent"
  value={query}
  onChangeText={(query) => { this.setState({ query }); }}// eslint-disable-line no-shadow
/>
```

Fig 4.7 TextInput Component On The Card Page

The query searching through these decks for flashcard objects (see Figure 4.6) is also identical except for the fact that it filters through the deck questions looking for characters that match a flashcard object's question or answer rather than a deck title.

```
let showingCards;
if (query) {
  const match = new RegExp(escapeRegExp(query), 'i');
  showingCards =
   deck.questions.filter((item) => match.test(item.question) || match.test(item.answer));
} else {
  showingCards = deck.questions;
}
```

Fig 4.8 Search Query On The Card Page

4.2.3 Slide, Edit And Delete Implementation

develop

4.2.3.1 Slide, Edit And Delete Implementation On The Home Page

The Swipeable component discussed in section 4.2.1 and displayed in figure 4.2 can be slid to the left by 60 pixels (indicated by the prop rightButtonWidth in figure 4.2) to display the View component of a menu (see figure 3.12) containing the following two buttons:

- **Edit**: A Touchable Opacity component that when selected, navigates to the DeckEdit page by passing the operation of *'edit'* in conjunction to the current title of the deck. (Shown in Figure 4.9)

‘

```
<TouchableOpacity
  onPress={() => {
    closeSwipeable(this.state.currentlyOpenSwipeable);

    this.props.navigation.navigate(
      'DeckEdit',
      { operation: 'edit', oldTitle: item.title }
    );
  }}
>
  <Edit />
</TouchableOpacity>
```

Fig 4.9 Edit Touchable Opacity Component For Deck Sub Menu

- **Delete:** A Touchable Opacity Component that when clicked asks the user whether they are certain they want to delete the deck (Shown in Figure 4.6) If they confirm *'OK'*, the deck is located and removed from the storage. Otherwise the action is cancelled.

```
<TouchableOpacity
  onPress={() => {
    closeSwipeable(this.state.currentlyOpenSwipeable);

    Alert.alert(
      'Are you sure you want to delete' +
      'this deck and all of its cards?',
      null,
      [
        { text: 'Cancel' },
        { text: 'OK',
        onPress: () =>
         this.props.removeDeck({ title: item.title }) },
      ],
      { cancelable: false }
    );
  }}
>
  <Remove />
</TouchableOpacity>
```

Fig 4.10 Delete Touchable Opacity Component For Deck Sub Menu

4.2.3.2 Slide, Edit And Delete Implementation On The Card Page

On the Card page, (Shown in figure 4.4), the component can also be slid to the left by 60 pixels to show a sub menu to edit or delete a flashcard object with the following two buttons:

- **Edit**: When selected, it navigates to the CardEdit page by passing in the operation of *'edit'*, the title of the deck and the flashcard's question and answer(Shown in Figure 4.11)

```
<TouchableOpacity
  onPress={() => {
    closeSwipeable(this.state.currentlyOpenSwipeable);

    this.props.navigation.navigate(
      'CardEdit',
     { operation:
       'edit',
       deckTitle,
       oldQuestion: item.question,
       oldAnswer: item.answer }
    );
  }}
>
  <Edit />
</TouchableOpacity>
```

Fig 4.11 Edit Touchable Opacity Component For Card Sub Menu

- **Delete:** When the Touchable Opacity is pressed, the app dispatches the removeCard action to the Redux back-end by passing in the identification keys of the deck title and question to accordingly locate and remove it from the phone's storage (Shown in Figure 4.12).

```
<TouchableOpacity
  onPress={() => {
    closeSwipeable(this.state.currentlyOpenSwipeable);

    this.props.removeCard({ title: deckTitle, question: item.question });
  }}
>
  <Remove />
</TouchableOpacity>
```

Fig 4.12 Delete Touchable Opacity Component For Card Sub Menu

4.2.4 Adding & Editing Of A Deck

When a deck name is entered into the TextInput component on the DeckEdit page, it is trimmed and removed of any blank spaces surrounding the text. The newly trimmed deck name is then inspected to validate that the user has entered in an acceptable deck name in the TextInput field (see Figure 4.13). If the TextInput field is left empty and then submitted, the app then prompts the user by telling them they must enter in a name to successfully continue with the adding or editing process. As deck names are used as identification keys, they all must be unique. Therefore, if the entered deck title already exists in the phone storage, the user is once again prompted and asked to enter in a new deck name.

```
48                  if (inputTrim === '') {
49                    Alert.alert(
50                      'You have to enter a name',
51                      null,
52                      [{ text: 'OK' }],
53                      { cancelable: false }
54                    );
55                    return;
56                  }
57
58                  if (deckTitles.indexOf(inputTrim) !== -1) {
59                    Alert.alert(
60                      'This name has been used',
61                      null,
62                      [{ text: 'OK' }],
63                      { cancelable: false }
64                    );
65                    return;
66                  }
```

Fig 4.13 Verification Of Inputted Text For Adding & Deleting Deck

If a unique name is successfully entered into the text field and submitted, the operation parameters are then assessed (see figure 4.14). These operation parameters are passed in when navigating from a previous page and are entirely dependent on which button was clicked i.e if the edit button is pressed on the sub menu from section 4.2.3.1, the operation is set to edit and if the plus icon is clicked in the navigation bar on the Home Page (see figure 3.11), the operation is set to add. Once the operation is determined, the appropriate action is dispatched to the redux back-end. The addDeck action only requires a trimmed and unique deck title to set up a new deck for the user (see line 69 in figure 4.14). However, the editDeck action requires the previous title of the edited deck as well the newly inputted deck title to locate the relevant deck object in the phone's storage and appropriately alter its name (see line 71 in figure 4.14). When either action is completed, the app is then navigated to the Card Page (see figure 3.1.3) of the newly added or edited deck.

```
68                  if (operation === 'add') {
69                    addDeck({ title: inputTrim });
70                  } else if (operation === 'edit') {
71                    editDeck({ oldTitle, newTitle: inputTrim });
72                  }
73
```

Fig 4.14 Operation Parameters For Adding & Deleting Of A Deck

4.2.5 Adding & Editing Of A Flashcard

The adding and editing of a flashcard operates in a very similar manner. The inputted question and answer are firstly trimmed before being validated as acceptable inputs (see figure 4.15). If either of the TextInput fields for the flashcard's question or answer are left blank, the user is prompted by a message that reminds them that these fields cannot remain empty for the process to be successfully completed. Following this, it is then verified that the entered question doesn't already exist in the current deck. This is done as the flashcard's question is used as a unique identification key and the system wants to avoid duplication occuring in the back-end of the app. However, the answer does not have any of these constraints and thus can be whatever the user chooses.

```
46              onPress={() => {
47                const questionTrim = question.trim();
48                const answerTrim = answer.trim();
49
50                if (questionTrim === '' || answerTrim === '') {
51                  Alert.alert(
52                    'Question or answer cannot be blank',
53                    null,
54                    [{ text: 'OK ' }],
55                    { cancelable: false }
56                  );
57                  return;
58                }
59
60                if (questions.indexOf(questionTrim) !== -1 && questionTrim !== oldQuestion) {
61                  Alert.alert(
62                    'This question has been added already',
63                    null,
64                    [{ text: 'OK' }],
65                    { cancelable: false }
66                  );
67
68                  return;
69                }
```

Fig 4.15 Verification Of Inputted Text For Adding & Deleting Of A Flashcard

If these parameters are successfully fulfilled, the operation criteria are evaluated. The add card operation can be activated by clicking the variation of the plus icon in the right of the navigation bar on the Card Page (see figure 3.13) and the addCard action is dispatched to the redux back-end once his is clicked (see lines 72 to 78 Figure 4.16). This action has the parameters of:

- **Title:** The title of the deck
- **Question:** The newly trimmed question
- **Answer:** The newly trimmed answer

- **History:** A record of the answer history for the flashcard. When a flashcard is added or edited it set to the default array of 5 correct answers and 5 incorrect answer. The usage of this will be further discussed in section 4.3.
- **Question Type:** The answer method the user wants to adopt when answering the flashcard during a quiz session.The options are '*stereotypical'* and *'written'*.

The editCard action has the exact same parameters as the addCard action except it has an additional one of *'oldQuestion*'. This extra parameter contains the previous question of the flashcard and is used to appropriately locate it in the back-end of the app so that the editCard action can be performed on the appropriate flashcard object. When either of these actions are performed, the app is then navigated back to the Card Page (see figure 3.13), where the newly added or edited flashcard can be viewed.

```
71              if (operation === 'add') {
72                addCard({ title: deckTitle,
73                  question: questionTrim,
74                  answer: answerTrim,
75                  history: [false, false, false, false, false,
76                    true, true, true, true, true],
77                  questionType: selectedQuestionType
78                 });//eslint-disable-line max-len
79              } else if (operation === 'edit') {
80                editCard({ title: deckTitle,
81                  oldQuestion,
82                  newQuestion: questionTrim,
83                  newAnswer: answerTrim,
84                  newHistory: [false, false, false, false, false,
85                    true, true, true, true, true],
86                  newQuestionType: selectedQuestionType,
87                }
88                );
89              }
```

Fig 4.16 Operation Parameters For Adding & Deleting Of A Flashcard

4.2.6 Single Flashcard View Implementation

```
299           {operation === 'individual' &&
300           <View
301             style={styles.viewPortQuiz}
302           >
303             {this.renderCard((compareCard(deck.questions, card)), true, navigation, 'individual')}
304           </View>
305           }
```

Fig 4.17 View Component For Single Flashcard Rendering

The single flashcard view function occurs when a user clicks on a flashcard's Touchable Opacity component contained within the FlatList component on the Card Page (see figure 3.13). Its functionality is to allow for the singular examination of a flashcard (see figures 3.16 to 3.20) without having to take a quiz. It achieves this by first evaluating the operation parameter (see line 299 In Figure 4.17). If this parameter is equal to *'individual'*, the renderCard function is implemented (see figure 4.17) with the following criteria:

- **Card:** The current flashcard object. (The equivalent of *this* in programming)
- **SingleCard:** A boolean value determining if the deck is rendering a single card component
- **Navigation:** The navigation parameters from the page
- **Operation:** Determines the operation selected

If all these criteria are successfully fulfilled, the function renders the CardFlip component (see figure 4.18). This is a component that contains two Touchable Opacities, a *'face'* component showing the question and a *'back'* component showing the answer (see figure 3.16). The initial default view is that the '*face'* component is displayed and the *'back'* component is hidden. However when pressed, the CardFlip component performs a flip animation that toggles between the two display states.

```
59        <CardFlip
60          style={styles.viewPort}
61          ref={(card) => { this.card = card; }}// eslint-disable-line no-shadow
62          duration={200}
63        >
64          <TouchableOpacity
65            style={styles.flipSide}
66            onPress={() => this.card.flip()}
67          >
68            <Text style={styles.face}>{card.question}</Text>
69          </TouchableOpacity>
70          <TouchableOpacity
71            style={styles.flipSide}
72            onPress={() => this.card.flip()}
73          >
74
75            <Text style={styles.back}>{card.answer}</Text>
76          </TouchableOpacity>
77        </CardFlip>
```

Fig 4.18 CardFlip Component For Flashcard generated by the renderCard Function

4.2.7 Quiz Implementation

The quiz function is implemented when the *'Quiz'* button is selected on the Card Page and it uses the same renderCard function shown in figure 4.16. However it instead passes in the operation of *'group'* or performs an operation to check it's not equal to

individual to indicate to the system that a quiz is occuring (see figure 4.19). As a result, an additional Text label stating the current quiz score is rendered by the function and is shown on the top right of the displaying flashcard (see figure 3.21).

```
55        { operation !== 'individual' && <Text
56          style={styles.score}
57        > SCORE: {this.state.score} </Text> }
```

Fig 4.19 Rendering Of Text Component Stating The Score

When the quiz function is implemented it returns a Swiper component (see figure 4.20) in which the renderCard function is used to satisfy its renderCard prop (see line 175 in figure 4.20) by generating the CardFlip component (see figure 4.18). This Swiper component is not to be confused with the Swipeable component seen in figures 4.2 and 4.4, as they perform completely different actions. For this Swiper component to function correctly, its prop of 'cards' needs to be satisfied by an array of card objects that the renderCard function can utilise.

```
166       return (
167         <View style={styles.container}>
168         {operation !== 'individual' &&
169             <Swiper
170               ref={swiper => {
171                 this.swiper = swiper;
172               }}
173               cards={cards}
174               cardIndex={this.state.cardIndex}
175               renderCard={this.renderCard}
176 ›             onSwipedLeft={() => {=}}
196 ›             onSwipedRight={() => {=}}
219 ›             onSwipedAll={() => {=}}
229 ›             overlayLabels={{=}}
267
268             />
269             }
```

Fig 4.20 Swiper Component Used During The Quiz Implementation

The array of card objects used is a global state variable whose parameters are first set when the initial component is mounted by the app (Shown in Figure 4.21). The first variable calculated when the component is mounted is the *'totalQuestions'* which sets the number of questions that will be asked during the quiz. This is decided by an if statement based on the deck size which states that:

- If the number of flashcards in the deck is less than 5, the quiz will have 10 questions
- If the number of flashcards in the deck is less than 10 but greater than 5, the quiz will have 15 questions

- Otherwise the quiz will have 20 questions

Once this is determined, two flashcard objects from the deck are chosen and pushed to a list containing these flashcard objects. The first object pushed is the first flashcard used during the quiz and the second object pushed is the second card. The second card is to act as a buffer or backup card so the deck will never be empty when pushing in more flashcard objects further on in the program. How these questions are selected by the *selectNextCard* function will be explained and discussed in section 4.3.The flashcard objects are pushed to the global state variable of the list instead of directly into the global state holding the cards array as to prevent the Swiper component from re-rendering and returning to its first quiz question over and over again.This will be discussed and explained further in the diffifulties encountered in section 5.3.3. Finally the global state variables of the ‘cards’ array is set to this list value and the total number of questions in the quiz is appropriately finalised, so that the quiz begins. When it starts, there exists only two flashcard objects in the cards array and the rest will be dynamically loaded by the program as the quiz continues.

```
26      componentDidMount() {
27        const { deck } = this.props;
28
29        let totalQuestions;
30          if (deck.questions.length < 5) {
31              totalQuestions = 10;
32          } else if (deck.questions.length < 10) {
33            totalQuestions = 15;
34          } else {
35            totalQuestions = 20;
36          }
37
38          this.state.list.push(deck.questions[generateQuestionNumber(deck)]);
39          this.state.list.push(deck.questions[generateQuestionNumber(deck)]);
40
41
42        this.setState({
43          cards: this.state.list,
44          total: totalQuestions,
45        });
46      }
```

Fig 4.21 Mounted Component

The are 3 possible methods of answering a question in the quiz and I will now discuss the events that occur when these methods are used:

1. Swipe Left:

```
178                onSwipedLeft={() => {
179                  quizCard({ title: deckTitle,
180                    question: cards[this.state.cardIndex].question,
181                    result: 'wrong'
182                   });
183                   for (let i = 0; i < cards[this.state.cardIndex].history.length; i++) {
184                     if (cards[this.state.cardIndex].history[i] === true) {
185                       cards[this.state.cardIndex].history[i] = false;
186                       break;
187                     }
188                   }
189                  this.state.cardIndex += 1;
190                  nextCardGenerator(this, totalQuestions, deck);
191                }}
```

Fig 4.22 OnSwipeLeft Function

The flashcard is swiped to the left by the user because when answering the question using the traditional method from section 3.5.5, the said or thought answer didn't match the actual one and they are indicating to the system that the question has been answered incorrectly. When this occurs, the quizCard action is dispatched to the redux back-end with the result of *'wrong'* (see lines 179 to 182 in figure 4.22). The answer history of the card is then appropriately updated and a single boolean state of true is changed to false (see figure 4.22). This raises the likelihood of the question being generated by the *selectNextCard* function. Subsequently, the cardIndex is increased and another flashcard object is added to the list array using the nextCardGenerator function (Shown in Figure 4.18). Another card object is only added to the list if the cardIndex is less than the total number of questions in the quiz.

```
358    function nextCardGenerator(card, totalQuestions, deck) {
359      if (card.state.cardIndex < (totalQuestions - 1)) {
360        this.state.nextQuestion = generateQuestionNumber(deck);
361        this.state.list.push(deck.questions[this.state.nextQuestion]);
362      }
363    }
```

Fig 4.23 nextCardGenerator Function

2. Swipe Right:

```
onSwipedRight={() => {
  quizCard({ title: deckTitle,
    question: cards[this.state.cardIndex].question,
    result: 'right',
    history: cards[this.state.cardIndex].history
    });
    for (let i = 0; i < cards[this.state.cardIndex].history.length; i++) {
      if (cards[this.state.cardIndex].history[i] === false) {
        cards[this.state.cardIndex].history[i] = true;
        break;
      }
    }
    this.state.score += 1;
    this.state.cardIndex += 1;
    nextCardGenerator(this, totalQuestions, deck);
}}
```

Fig 4.24 OnSwipeRight Function

The flashcard is swiped to the right by the user because when answering the question using the traditional method from section 3.5.5, the said or thought answer matched the actual one and they are indicating to the system that the question has been answered correctly. When this occurs, the quizCard action is dispatched to the redux back-end with the result of ‘right’ and the answer history of the card is appropriately updated and a single boolean state of false is changed to true, lowering the likelihood of the question being generated by the ‘*selectNextCard’* function. Subsequently, the score and cardIndex are increased and another flashcard object is accordingly added to the list array using the *nextCardGenerator* function (Shown in Figure 4.18).

3. Enter The Answer Via TextInput

```
function writtenAnswerChecker(inputtedAnswer, card, cardAnswer) {
  const inputtedAnswerLowerCase = inputtedAnswer.toLowerCase().trim();
  const cardAnswerLowerCase = cardAnswer.toLowerCase();

  if (inputtedAnswerLowerCase === cardAnswerLowerCase) {
    card.swiper.swipeRight();
  } else {
    if (card.state.isClicked === false) {
      card.card.flip();
    }
    card.setState({ isClicked: false });
    setTimeout(() => {
      card.swiper.swipeLeft();
   }, 500);
  }
}
```

Fig 4.25 writtenAnswerChecker Function

This method of answering a quiz question can only be employed if the user who created the has chosen the answer method of *'written'* when adding or editing the card as previously discussed in section 4.2.5. Instead of swiping left or right to indicate the answer, it is entered by the user into the TextInput component located in the *'written'* flashcard object (see figures 3.19 to 3.20).This answer is submitted by pressing the orange subission button and once submitted, the inputted answer is passed into the *writtenAnswerChecker* function (Shown in Figure 4.25) along with the flashcard object and the correct answer. The inputted answer and card answer are both altered in the form of removing all capital letters and trimming any blank spaces to ensure they are in the exact same answer format. Following they are compared and if the inputted answer equals the card's answer, the swipe right function is implemented. If the inputted answer does not equal the card's answer, the card is flipped to display the correct answer and after 500 milliseconds the swipe left function is implemented.

4.2.8 Result Implementation

```
            onSwipedAll={() => {
              this.props.navigation.navigate(
                'Result',
                { deck,
                numberOfCorrectAnswers,
                totalQuestions,
                key: operation === 'retry' ? key : navigation.state.key }
              );
            }}
```

Fig 4.26 OnSwipedAll Function

When the list object containing these functions is finally emptied and the quiz is finished, the *onSwipedAll* function (shown in figure 4.26) is utilised to navigate the app to the Results Page (see figure 3.23 & 3.24). On this page the score is calculated by dividing the number of correct answers by the total number of questions and the result is displayed on the screen in the form of a percentage. If all the questions weren't answered correctly, a button prompting the user to take the quiz again is presented. Otherwise a message congratulating the user on their score is shown by the system and they are navigated back to the Card Page.

4.3 How The Next Cards Are Selected

The process for selecting the next card during the quiz is based on the Leitner model and aims to select cards that are answered wrong more frequently (i.e. they have more false values in their flashcard history). The six step process for doing this is:

1. History Calculation:

Each flashcard object contains an answer history that the system uses to determine how frequently the question has been answered incorrectly in the last 10 attempts. This answer history is a boolean array attached as a piece of data to each individual flashcard object. This array is set to the initial value of 5 true and 5 false when the flashcard is first added to the deck by the addCard action (see figure 4.27). It is initially set to this original value so it has the most even likelihood of being chosen. However each flashcard's answer history can be altered by the swipe functions (see figures 4.22 and 4.24). If the flashcard object is swiped left, a truth value in this array is replaced with a false value. If there are no truth values available in the array then no change occurs. If the flashcard object is swiped right, a false value in the array is replaced by a truth value. Similarly, if there no false values available in the array then no change occurs.

2. Total Number Of Incorrect Answers In the Deck:

The number of times each flashcard object is answered incorrectly in the deck is tallied and returned. This is done by utilising the function getNumberOfIncorrectTimes, which examines a flashcard's answer history and adds one to a counter every time it finds the boolean value of false in this history. Once this function has cycled through the entire array it returns this counter.The totalNumberOfIncorrectAnswers function then calculates the entire number of incorrect answers in the whole deck by adding all these counters together and returning the value

3. Percentage Assignment:

A new array of the exact same length is created to store each question's percentage value of being chosen. To percentage value for a card to be chosen can be calculated by getting the number of times it was answered incorrectly and dividing it by the total number of times every question was answered incorrectly. These percentages are then mapped onto the array.

4. Range Assignment

These percentage values are then distributed between 0 and 1 by adding the previous range value with the next flashcard's percentage (see figure 4.27).

Example:

- Flashcard A has a percentage of .255
- Flashcard B has a percentage of .434
- Flashcard C has a percentage of .311

Value	Percentage	Range x	Range Y
Flashcard A	.255	0	.255
Flashcard B	.434	.255	.689
Flashcard C	.311	.689	1

Fig 4.27 Sample Ranges

5. Random Number Generator is Used:

A random number generator is used to select a value between the ranges of 0 and 1.

6. Question Retrieval:

If the randomly generated number returns a value in a question's range, that question is returned. From the example, if .353 is returned Flashcard B is selected. If .757 is selected then Flashcard C is returned.

The process works on the basis that questions that are are answered wrong more frequently will have larger ranges between 0 to 1. Therefore they have a greater chance of being selected by the random number generator and ultimately displayed to the user. Everytime a question is selected and answered, its answer history is updated and the six step process has to occur all over again with updated totals, percentages and ranges. This mean that it is a dynamic user model as it assess the user's interactions with the

quiz questions and updates this system to select the questions based on this information.

4.3.1 Example

There exists three flashcards in deck with the following information in figure 4.28

Flashcard	Card History
A	{true,true,true,false,true,true,false,false,true,true}
B	{false, false, false, true, true, false, true, false, true, false}
C	{true, false, true, false, false, true, false, false, true, true}

Fig 4.28 Sample Card History

1. **History Calculation:**
   - Flashcard A has been answered incorrectly 3 times
   - Flashcard B has been answered incorrectly 6 times
   - Flashcard C has been answered incorrectly 5 times

2. **Total Number Of Incorrect Answers In the Deck:**
   3 + 6 + 5 = 14

3. **Percentage Assignment:**
   - Flashcard A has a probability of 3/14 = .214
   - Flashcard B has a probability of 6/14 = .429
   - Flashcard C has a probability of 5/14 = .357

4. **Range Assignment**
   - Flashcard A has a range of 0 to .214
   - Flashcard B has a range of .214 to .643
   - Flashcard C has a range of .643 to 1

5. **Random Number Generator**
   Random number of .588 was generated

6. **Retrieval Of Question**
   .588 exists in flashcard B's range so Flashcard B is retrieved and its card history is altered, changing the percentages and ranges

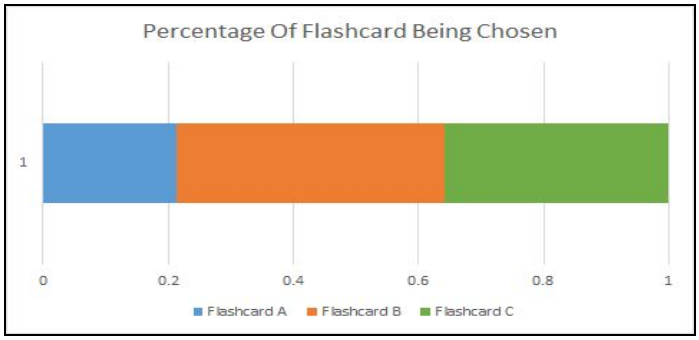


Fig 4.29 Sample Of Percentage of Flashcards Being Shown

4.3.2 Corrective Action

When examining the algorithm during the initial testing phase, it became clear to the developer that if a card was answered correctly ten times in a row, it would have a 0% of being selected by the system and be lost out of circulation. To account for this a corrective action was made of adding .05 to every value to ensure every card always had a chance of being selected, even if it was minimal.

4.3.3 More Sophisticated Example

The following is sample data taken from a quiz about capitals done by the author over two days. Figure 4.31 shows a breakdown of the calculations done on excel and figure 4.32 shows a bar chart depict the percentage ranges.

Flashcards	No. of times wrong	percentage	Corrective Measure	New Percentage	Range X	Range y
Ireland	0	0.000	0.050	0.024	0	0.024
Brazil	3	0.037	0.087	0.041	0.024	0.065
France	2	0.024	0.074	0.035	0.065	0.100
Spain	0	0.000	0.050	0.024	0.100	0.124
Germany	2	0.024	0.074	0.035	0.124	0.160
Italy	1	0.012	0.062	0.030	0.160	0.189
Netherlands	4	0.049	0.099	0.047	0.189	0.236
Latvia	7	0.085	0.135	0.064	0.236	0.301
Belarus	8	0.098	0.148	0.070	0.301	0.371
Lithuania	6	0.073	0.123	0.059	0.371	0.430
Serbia	5	0.061	0.111	0.053	0.430	0.483
Liechtensteir	4	0.049	0.099	0.047	0.483	0.530
Azerbaijan	5	0.061	0.111	0.053	0.530	0.582
Bulgaria	4	0.049	0.099	0.047	0.582	0.630
Luxembourg	2	0.024	0.074	0.035	0.630	0.665
Malta	8	0.098	0.148	0.070	0.665	0.735
Monaco	3	0.037	0.087	0.041	0.735	0.776
Slovakia	3	0.037	0.087	0.041	0.776	0.818
Turkey	4	0.049	0.099	0.047	0.818	0.865
Ukraine	4	0.049	0.099	0.047	0.865	0.912
Iceland	2	0.024	0.074	0.035	0.912	0.947
Hungary	5	0.061	0.111	0.053	0.947	1.000
Total	82	1	2.100	1.000		

Fig 4.31 More Sophisticated Example Calculation

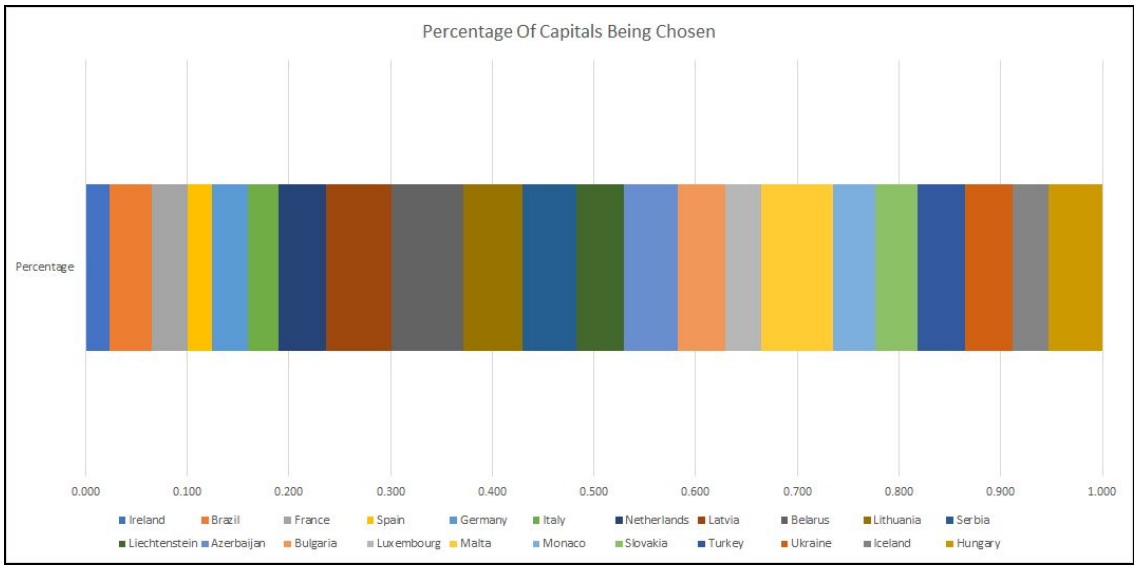


Fig 4.32 Sample Of Percentage of Flashcards Being Shown In Sophisticated Example

4.4 Conclusion On Implementation

Implementation is always is always the hardest part of any programming project and although the logic seems straightforward, implementing this system proved to be quite difficult at times. However creating an adaptive learning mechanism through the use of

dynamic user modelling and seeing it function properly on a new framework was extremely fulfilling. By separating the implementation process it into smaller step-by-step parts, the app was able to be developed into a relatively complex system that is capable of working in tandem together to satisfy a relatively challenging user need.

Chapter 5: Evaluation

5.1 Overview

The evaluation of QuizFlash consisted of the following:

- Internal Testing
- Design Evaluation
- User Testing
- Evaluation of Difficulties
- Overall Evaluation

The final evaluation of the application was only conducted after the other testing phases were completed and this involved the internal testing of all the primary features and their designs. Furthermore, the application was intermittently evaluated in accordance with the incremental model to ensure that development goals were satisfied and in line with targets set out at the beginning of the project.

The author ran into a number of difficulties during testing which will be discussed in detail
below. To conclude, an in-depth evaluation will be provided at the end of this section, outlining the successes and failures experienced during the development of QuizFlash.

5.1 Initial Testing

As discussed in section 3.3, the author developed the project using the incremental model. This resulted in testing occurring during the development and implementation of

every significant feature and more intense testing being conducted upon the completion of components. This testing occurred on the author's Android smartphone through the use of the Expo CLI development server and the Expo Client app. The app was run on the phone by using the Expo CLI server creating a QR code on the computer's browser that can be scanned by the Expo Client app to transfer the JavaScript Bundle to the smartphone via WiFi tunnelling protocol. This resulted in the benefit of testing occurring more frequently, more effectively and on the intended device for use. Additionally, React Native's live-reload feature meant that any saved changes made to the code could be instantaneously viewed on the smartphone, without the delays of configuration or Gradle builds of other development methods. This was an essential aspect of implementing the incremental model successfully and it contributed hugely to the development process, by allowing the author to visually interpret how their code changed the app. In the initial testing phase of the project, the two main forms of testing implemented during this phase were Testing to Break and User Scenarios, as discussed next.

5.1.1 Testing To Break

When conducting these initial tests, the author would regularly try to 'break' the system by attempting to do unexpected actions outside the system's parameters or stereotypical types of usage e.g. entering the incorrect answer into the *'written'* flashcard while simultaneously trying to swipe the card to the right or entering highly unusual question and answer formats. This was done to identify any flaws or issues that could be exploited.

5.1.2 User Scenarios

User scenarios are narratives which outline how a user might interact with an app or software application and were regularly incorporated throughout the development process. In these scenarios, specific tasks are characterised and narratives are written describing how a user might try to complete these tasks. These scenarios are advantageous in the development process as they enable developers to gain a new perspective on the possible wants and needs of potential users. By avoiding technical processes, it leaves these developers open-minded to identify and solve problems that they might not have seen otherwise.

Example

Hannah is a second year student in Law & Business in Trinity College who has an upcoming exam in the module Operations Management. She wants to learn the definitions of key terms while commuting on her 1 hour journey from Drogheda to Trinity and aims to create a deck of 20 flashcards on the app to

achieve this. She particularly struggles with definitions surrounding lean systems and wants the app to focus on these cards more than the others. She mainly uses an auditory style of learning when studying but would prefer to answer the flashcards in a writing style as she is uncomfortable with saying the answers aloud in a public setting.

Fig 5.1: Business MCQ

User Scenarios were useful in the specific development of QuizFlash because they identified more individualistic issues that otherwise would have gone overlooked on the system. Examples of problems recognised and solved by using this method are:

- Some flashcards having a 0% chance of being asked as they had been answered correctly 10 times in a row and thus had a value of 0 in the card history array.
- Flashcards not having their answer history set to a more optimal length of 10 and not being pre-set state of 50% wrong and 50% right.
- The app just instantly swiping the card to the left when the incorrect answer is submitted into the *'written'* flashcard'. The user didn't learn anything using this method as the flashcard was never flipped to display the correct answer at any point.

5.2 Design evaluation

The design goal of the user interface was to achieve a desired look and feel of elegance and simplicity, while maintaining a structured and pragmatic format. To judge whether this was achieved, a heuristic evaluation was performed. This form of evaluation assists in the identification of usability issues by making comparisons between the UI and the 10 design principles outlined by Jakob Nielsen outlined in his article *10 Usability Heuristics For User Interface Design* (1995).

5.2.1 Nielsen Heuristics

1. **Visibility of System status:** The system should utilise appropriate feedback to constantly keep the user informed about what is going on.
   Status: **Achieved**
   Description: Provides the user feedback during the quiz by showing them their score. The system also presents their result in the form of a percentage upon completion of the quiz and uses a green or red text label overlay depending on whether the flashcard is swiped left or right.

2. **Match Between Systems And The Real World:** The system should speak the users' language, with words and phrases that are familiar to the user rather than technical terms. Information should also be in a natural and logical order.
   Status: **Achieved**
   Description: Technical terms are not used throughout the app and the logical order of LIFO (Last In. First Out) exists in the FlatList functionality.

3. **User control and freedom:** Users should be able to freely navigate around the system. There should always be an option to ‘exit’, to avoid becoming stuck in an unwanted state.
   Status: **Achieved**.
   Description: The navigation bar has a return button on the top left of every page (except for the home page) to allow for the app to be freely navigated.

4. **Consistency and standards:** The system should establish standardised language and consistent interactions. User’s should not have to wonder whether different words or icons perform the same actions
   Status: **Achieved**
   Description: The system employs consistent language and interactions across all facets of the app. All buttons and icons perform the exact same function when clicked.

5. **Error prevention:** The system should be entirely error free.

Status: **Achieved**
Description: There are no errors to the developers knowledge in the current iteration of the app and user errors are restricted from occurring.

6. **Recognition rather than recall:** The user should not have to rely on memory in order to operate the system. Instructions and required information should be visible when needed.
   Status: **Failed**
   Description: In some cases, instructions on how the app should be used are not made available to the user and the design is not intuitive enough to guarantee that a new user can instinctively understand core concepts. This can be highlighted by 3 main examples within the app:

   - There is no indication that the flashcards are Touchable Opacity components that when clicked will toggle between the question and the answer

   - It is never signified at any point throughout the app that the quiz component uses a swiper function. Users are just left to figure it out. Additionally, an incorrect assumption was made. The author considered it common and innate knowledge that swiping right on a phone indicates a positive response and that swiping left on a phone indicates a negative response. New users who have no experience with similar systems (e.g Tinder or Jobr) could greatly struggle with this aspect of the design.

   - The menu for editing or deleting a deck/card can only be displayed by sliding the FlatlList component to the left by 60 pixels. However there is no way for users to know where this hidden menu is located (or that it even exists) when using the app for the first time. This facet of the decision has the potential to quickly frustrate them and reduces the app's usability.

7. **Flexibility and efficiency of use:** Accelerators should be able to increase the speed at which an experienced user can interact with the system.
   Achieved: **Somewhat**
   Description: Accelerators exist in the app in the form that experienced users can swipe on *'written'* flashcards to speed up their quizzing process. Yet, this is the only instance of it occurring as accelerators are not a prevalent concept used throughout the app. However the time required to use the app is already quite small and thus the number of instances where this acceleration could occur is minimal. Therefore this heuristic shouldn't be scrutinised too closely.

8. **Aesthetic and minimalist design**: Dialogue should not contain irrelevant or unnecessary information
   Achieved: **Achieved**
   Description: Designs were kept very minimal and simplistic and only content that seemed relevant to the user was displayed on each page.

9. **Help users recognise, diagnose, and recover from errors**: Error messages should be expressed in plain language, precisely indicate the problem, and constructively suggest a solution.
   Achieved: **Achieved**
   Description: Error messages are presented when an impermissible action such as the addition of a blank flashcard or the employment of a non-unique deck name occur. These error messages prompt the user of the correct action and how the issue can be overcome e.g Question can't be a blank field, please enter question

10. **Help and documentation:** Any help information should be easily located and should provide clear steps to be carried out.
    Achieved: **Failed**
    Description: As discussed during the recognition versus recall heuristic , not a single form of help, demonstration or documentation is provided to the user at any point. It certainly is a failing aspect in the design process of the app.

5.2.2 Assessment of Heuristic

Overall the app performed very well in the evaluation, with a large majority of the heuristics being achieved. However, some significant design issues do exist with the app that can't be ignored, specifically in the form of recall being used rather than recognition. When moving forward with the project these issues need to be acknowledged if usability wants to be improved upon. I will discuss how I will plan to address this further in section 6.x.x

5.3 User Testing

The application was planned to be distributed in the form of an online Qualtrics survey for user testing and evaluation of the app's usability and design. An online survey was chosen as the medium to do this because:

- It was the most convenient
- It allows for the survey to be sent to those who are unable to meet the developer in person.

- The results of the survey can be aggregated and analysed much easier than a survey conducted on different sheets of paper

The website Qualtrics was used specifically for its survey logic control feature which determines which questions were to be shown based on a user's previous answers. The aims and goals of the survey were established and I will explain how it aims to evaluate the app's usability on two main levels:

5.2.1 The System Usability Scale

The System Usability Scale (SUS) is an industry standard measurement tool used to give assessment on the usability of a system by taking into account its context and evaluating it on 3 main aspects:

- **Effectiveness:** The achievement of user objectives.
- **Efficiency:** How the resources are utilised to achieve these objectives.
- **Satisfaction:** The satisfaction of users.

Subsequently, the first part of the survey consisted of a ten-item SUS questionnaire in which each item has a total of five possible options ranging from '*Strongly Agree'* to *'Strongly Disagree'*. It is a standardised and widely adopted questionnaire which evaluates the usability of many different types of systems and has the added benefit of not needing large sample sizes as it can just adjust its confidence intervals instead(usability.gov, n.d.). A copy of the SUS questionnaire is enclosed in Appendix 2.

5.2.1.1 The System Usability Scale Evaluation Method

Once the questionnaire is completed, it returns a score ranging from 0-100. To calculate this score we use the following three steps:

1. For each odd numbered questions, subtract 1 from the score.
2. For each even numbered questions, subtract that score from 5.
3. Multiply the sum of all these values by 2.5 to determine the SUS rating.

The average SUS rating for a system is 68 (usability.gov, n.d.) and thus anything above that is considered above average and vice versa. Additionally, anything above 80.3 is regarded as a very good score and anything below 51 indicates that the system has usability issues. If we analyse this and consider the high usability goals of Quizflash , a score of at least 75+ would be a minimum requirement before a successful and complete launch of the app to the wider public.

5.2.1.2 Why The System Usability Scale Was Chosen

The SUS questionnaire was selected specifically for QuizFlash because it:

- Returns dependable and reliable results, regardless of sample size. This is advantageous as it can be difficult for a student to attract users to its surveys.
- Implements a user-friendly Likert scale on the survey. Not only does this regulate the response format but it also makes it easier to comprehend and analyse submitted information.
- Is a specialised and industry standard questionnaire created by experts. This means it can be relied on to effectively validate the system's usability rather than trying to create a personal set of standards.

5.2.2 Specific Questions On The Quiz Functionality

The Quiz Function is the main feature on the app and the feature of the app that users will be using the most often. As a result, the rest of the survey consisted of questions which specifically focused on improving the usability of this one core aspect. The reasoning behind this was that by concentrating all the efforts on optimising the app's USP (Unique Selling Point), it would satisfy the greatest number of existing users, while simultaneously attracting new ones. It would also provide the app with a strong basis of a *'usable'* foundation to be developed and improved in the future. This line of questioning was split into two parts and can be summarised as follows:

5.2.2.1 Questions Regarding The Layout

These questions aimed to comprehend if the structure of the quiz component was too complex for the average user and diminished their experience using the app. The questions asked are shown in figure 5.1 to 5.2:

Fig 5.2: Survey Question About Layout

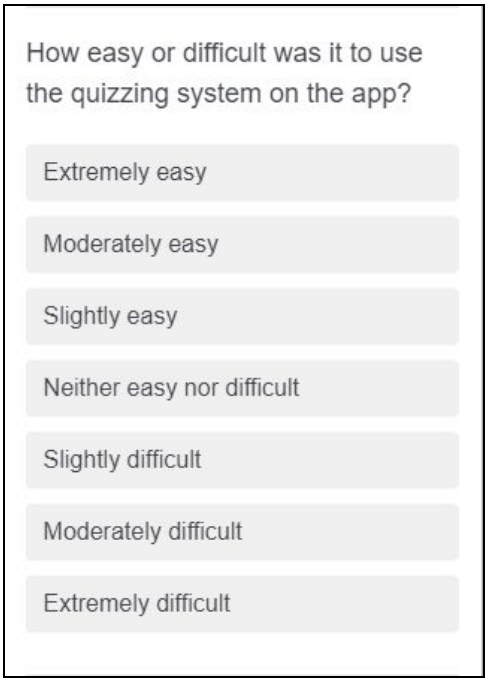


Fig 5.3: Survey Question About Quizzing System

5.2.2.2 Questions Regarding The Swiper Component

The objective of these questions was to identify if the implementation of a swipe functionality in the quiz component actually improved the usability of the app. The questions asked are displayed below in figure 5.3 to 5.4:

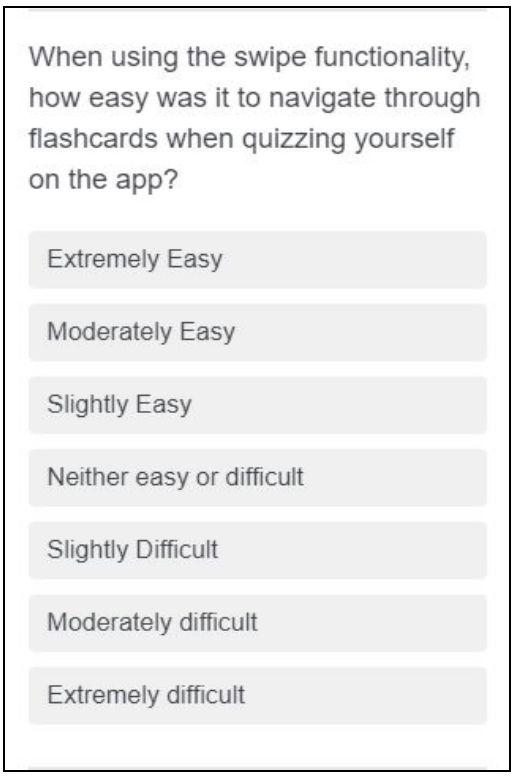


Fig 5.4: Survey Question About ease of swipe

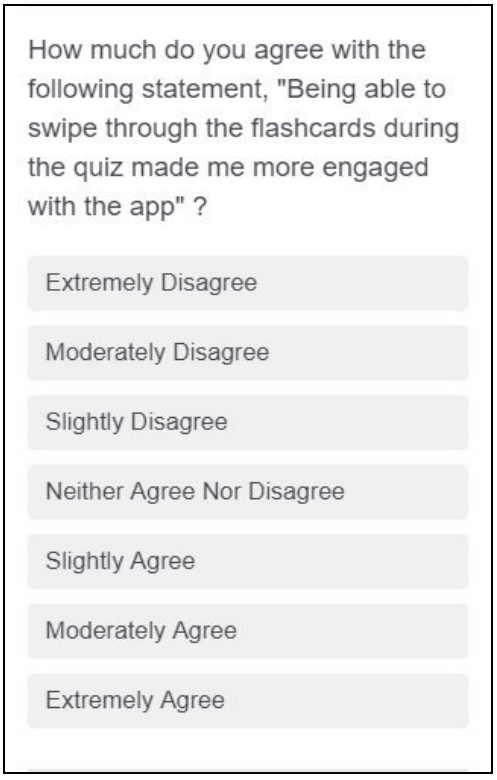


Fig 5.5: Survey Question about benefit of Swipe

5.3 Research Ethics

For this survey to be distributed, a research ethics application needed to be completed and approved by Trinity's Computer Science and Statistics Research Ethics Committee. Unfortunately, the application was submitted but not approved by the committee as it required the following amendments:

- The questionnaire utilised in the survey needed to be included on the application as an extension of the pdf file rather a html link
- The number of participants expected to be utilised in the survey was required to be included in the project description
- A clarification on how the data will be disclosed to the participant

These amendments were received on the 5th of April in the author's third correspondence with the committee regarding the application. Due to the time constraints, these amendments could not be successfully implemented and re-submitted to them before the project's deadline. As a result the survey was not conducted. A copy of this application's outline has been enclosed in the Appendix 3.

5.4 Evaluation Of Difficulties Encountered

In any software development project, there always exist several difficulties which are encountered during the creation process. This app was no different and the author

confronted a number of issues during development. While some could have been avoided through better planning or improved subject matter comprehension, others were complications that had to be overcome regardless of difficulty. In this section of the chapter I will go on to describe these issues and how they were overcome:

5.4.1 .Learning A New Framework

Problem

The author had very little knowledge in the field of mobile development and had no prior experience of using:

- The mobile framework of React Native to create the app
- The React Native toolchain of Expo to implement external libraries
- The JavaScript library of Redux to manage the back-end

As a result a large investment of time was required at the beginning of development to gain a firm grasp of their capabilities and how they could be best utilised to create a successful project. This mainly occurred in the form of online programming courses provided by the website Udemy and the construction of smaller and simpler applications using these unfamiliar tools (e.g a calculator and an album library). While experiences in web development provided a strong foundation to work from, the author found it challenging at times to find online support and tutorials to address more technological or specialised topics during development. This was because React Native is still a relatively new framework that was created in 2015 by Facebook and therefore only has a nominal amount of online assistance. Furthermore, there also have been many updates to the framework since and correspondingly many of these online tutorials are using older versions that have since gone obsolete.

Solution

This was overcome by the author meeting up with individuals who had relatively more experience using React Native and asking them more detailed and highly technical questions that were more specific to this project. The author also employed a lot of experimentation in the form of trial and error to inspire and generate ideas. This ultimately yielded the creation of viable and utilised solutions.

5.4.2 Redux Back-end

Problem

The strictness and complexity of Redux’s formatting for the back-end was a concept that the author found particularly hard to comprehend and implement into the system. It acts

so differently to other concepts used within the frameworks that many new and unforeseen challenges were presented and these hurdles were very difficult to overcome.

Solution

As discussed in section 5.3.1, an individual with more experience in this subject matter was consulted. Under his recommendation, other systems using this form of JavaScript library were inspected and examined. After studying these system for quite some time, patterns began emerging in the code which greatly improved the author's comprehension of the subject and assisted in the creation of a more personal and improved Redux back-end within their project.

5.4.3 Swiper Issues

Problem

This was the most technical problem experienced during the development process . The functionality of the swiper component within the app works by loading in a series of flashcard objects into the prop *'cards'*. These cards are then navigated through by being swiped left or right and it is only once all the cards have been completed that the app will be navigated to a new page. However, functions and actions are to be performed on the swiped card's states during the quiz functionality depending on whether they were swiped left or right. A primary example of this is the updating of the card's answer history to reflect the user's most recent answer. Changes to each flashcard's state could be done using a *setState* function, however whenever a *setState* function is called it results in an entire re-rendering of the page. This normally would be fine, except in this specific instance. The issue was that whenever the page was re-rendered it would result in a reload of the swiper component, meaning that it would simply just show the first card over and over again. Ultimately the screen would just infinitely stay on the first question of the quiz and it was impossible to complete.

Solution

There is a commonly held misbelief in React Native that these state variables are immutable and as a result can only be altered by the *setState* function. This is only partially true, as it is considered best practice within the documentation to treat these states as immutable despite the fact that they're actually not. They are treated as immutable variables because if a manual state mutation occurs, it will be overridden when a setState function is implemented. This can result in errors due to the way that states are processed in batches and thus mixing setState function with operators isn't advocated. However, this did generate the program's solution as it meant that as long as manual mutations were adequately managed and not mixed with setState functions they

could be implemented. This allowed for the re-rendering of the page and swipe component to not occur and the app operating as initially intended.

5.4.4 Scope Creep

Problem

This refers to the continuous and uncontrolled changes in a project's scope after it begins. In the introductory discussions of the app's development, the author was very naive in what could be achieved in this project's time span. Ultimately there were just too many objectives outlined for the app to reach that the overall focus of the project's core components suffered. It was only over time, specifically during the programming phase of the project, that these goals and expected requirements were narrowed down to more pragmatic and reachable targets.

Solution

There is no direct solution to this issue. However if the project moves forward (or the author takes part in a new one), a properly defined set of requirements and the setting of appropriate objectives *needs* to be done in the initial stages of the set up. On reflection, this issue could have been solved in this process if the author had conducted more research in the planning phase of the project in conjunction with properly analyzing their own skills and limitations

5.4.5 Cognitive Myopia

Problem

Throughout the process, the author had an inclination to only concentrate on the information immediately related to their own judgement and biases, while ignoring other prominent pieces of information. If we look at the main issues returned from the heuristic evaluation, it can be seen that at times the author found it hard to take the perspective of the average everyday user who might not be as 'tech savvy' and knowledgeable about these systems, which resulted in them creating a less usable system.

Solution

Similar to section 5.3.4, there is no direct solution to this problem. It can only be overcome through the development of skills which enable them to gain this perspective. It is crucial for developers to cultivate these skills so that systems can be built for users accordingly and that software usability is maximised. This is something the author felt

they improved on significantly while creating the app and throughout the process they began to think more and more like the average user, resulting in superior solutions being generated.

5.5 Overall Evaluations

The biggest issue in the project that could have been more easily addressed, or avoided completely was the approval of the ethics application. However, all the information gathered can be used in the near future and it definitely helped in gaining perspective on the project. A more accurate reflection on the evaluation of the project would be to realise that it was very successful and a high majority of the requirements were met by the final product in addition to very difficult and unforeseen obstacle being overcome during development. Overall, the author was extremely happy with the project's outcome of a fully functioning cross platform app that provides tangible methods for users to improve their learning process and felt it more than met the initial scope and expectations.

6 Future Developments

The majority of the functional & non-functional requirements outlined in section 3.2 were met. Although most of the project was completed, some of the issues encountered were not solved in time. On top of that, there is always room for improvement and innovation. This section is about future work that could be done on the project.

Even though the functional and non-functional requirements outlined earlier were met, as part of the evaluation stage the author has identified what steps could be taken to further improve the project. The following improvements were either identified during development or on completion of the project.

6.1 Finish User Testing

If the project were to continue, its immediate primary focus would be to gain approval from Trinity's Computer Science and Statistics Ethics Committee for the dissemination of the online Qualtrics survey. This would be the main objective in the near future as it would allow for the SUS rating of the app to be obtained, analysed and assessed. With this information alterations could be made and issues isolated to improve the overall user experience. The online assessment would also gauge opinions regarding the swipe feature utilised during the quiz function and if it is an engaging and viable component that enhances the overall usability of the app. In general, the completion of the user testing would enable the project to get its first form of external feedback which can only be beneficial for the improvement of the app.

6.2 Make An App Tutorial

It became increasingly obvious during the design evaluation phase in section 5.2 , that the app was constructed without the appropriate planning being conducted on how best to educate new users so that they can appropriately use the app. It was naive to believe that a system this complex could be utilised intuitively without any form of guidance being offered. Moving forward, this could be best addressed by creating some form of initial tutorial that could be implemented when the app is loaded up for the very first time. This tutorial would consist of:

- An explanation of the system's objectives and how it implements adaptive learning techniques such as the question selection method and the variety in answer type to create a personal environment that is tailored to their educational needs.

- A brief description of the VARK learning model with a questionnaire that would assist them in identifying their own primary learning style so users could choose the appropriate question or answer methods that is most conducive for their learning type.

- An exploration through the app, providing a walkthrough on how to perform the primary features of:
    - Creating,editing and deleting a deck
    - Creating,editing and deleting a card
    - How to quiz yourself using the app
    - How to implement the swipe functionality during a quiz

6.3 Conversion To A Web App

The React Native system is already a cross-platform app meaning that it was built to be used by both Apple and Android systems without any alterations. However, with further development it could also become a customised React Web app. For this to occur, it would involve the additional act of converting the framework from React Native to React Web when the app is being utilised as a website. All features of the app (e.g swipe, slide, add, etc) would be completely unaffected and it would operate exactly the same online as it does as an app. This is because the project is programmed in JavaScript which can run successfully on HTML5 browsers. The only foreseen error, the author thinks the system could experience is when the website is being viewed by a desktop computer. The spacing would be incorrect and the swiper functionality would not be as practical. To overcome this, a separate view for screens above a certain height and width (e,g 1024px and 768px) would be created to cater to their specific needs. To implement this successfully it would involve converting the JSX syntax to HTML and as they're already extremely similar languages, this would not be very time consuming and achievable in the near future. The only reason that it wasn't undertaken during this project was that it would also require the supplementary creation and maintenance of an online database, which was beyond the scope and capabilities of the project's requirements. However, the eventual goal is to construct an online Firebase database, which would allow for information to be shared and accessed across many platforms at any time through the use of an online log in system.

6.4 Additional answer methods

One of the key findings discovered when conducting research for this project was the different types of learning styles each person had and how these styles could be utilised to make environments most conducive to learning through the use of the VARK model. The initial aim of this project was to create a wide variety of answer methods, with a minimum of one for each learning style (with the exception of the Kinesthetic approach).

Due to time constraints placed on the project, the final result only had the implementation of two answer methods using *'auditory'* and *'reading/writing'* learning styles. When continuing on with the project, I would like to fulfill the original objective and add the following answer methods:

- **Image Integration:** This answer method would be most beneficial for *'visual'* learners, whose mind is best stimulated when presented with images. The answer method would function by the question being presented in the form of a picture, chart or graph and the answer being submitted in the *'traditional'* or *'written'* format.

    - **Traditional**: The answer is said aloud or thought in one's mind. Following this the flashcard is then flipped by pressing the *face's* Touchable Opacity component and the said or thought answer is compared with the actual answer on the back of the card. If the answer is correct, the user swipes the screen right. If incorrect the user swipes the screen left.

    - **Written**: The answer would be submitted by inputting text and then comparing it with the actual answer of the flashcard.

- **Acronyms:** This answer method would be most beneficial for *'visual'* and *'reading/writing'* learners. It would involve the question displaying each letter of the acronym on the left hand side of the screen and the corresponding answer being submitted in the form a text input on the right hand side of the screen.

- **Lists:** This is answer method would also be the most beneficial for *'visual'* and *'reading/writing'* learners. The question would be presented on the screen in the form of a standard Text View with the number of answers expected below it. The answers would then be entered individually submitted in the form of inputted text. If all the list items aren't independently entered correctly, the answer as a whole will be regarded as incorrect. However the possibility of implementing of some form of fuzzy systems may provide a more elegant solution to overcome this.

- **Voice Recognition:** This would be the most technically advanced and difficult answer method to implement, however it would also be the most practical and beneficial method for *'auditory'* learners. The question would be presented in the form of a standard Text View and then spoken aloud by the phone system. The user would then reply to the question by saying the answer out loud in response. Using voice recognition technology, the system would determine if the spoken answer is correct or not and then advance on to the next question. Systems similar to this do already exist in apps such as DuoLingo, so it is not an unfeasible concept as a future development prospect.

6.5 More Sophisticated Methods For Questions To Be Selected

While the current *selectNextCard* system is more than adequate for the project's requirements, the author would like to undertake an investigation into what is the best method for selecting the most appropriate flashcard when being quizzed. The objective of this study would be to isolate the relevant variables that impact these adaptive learning systems and use them to construct a more sophisticated and optimised algorithm to be implemented when selecting questions on the app. These variables could include, but are not limited to:

- **Spacing:** The length of time since the flashcard was last asked.
- **History:** The answer history of the flashcard.
- **Deck Size:** The number of flashcards in the deck.
- **Frequency:** How often the flashcard has been asked in the last 24 hours.
- **Answer Method:** The type of answer method is the flashcard.
- **Previous Card:** The difficulty and answer method of the previous question.

6.6 More Sophisticated Feedback

Currently, the system only provides three instances of feedback:

- A text label indicating the user's score
- Red and green overlay text labels when swiping left or right
- The final result in the form of a percentage

In the advancement of the project this aspect would looked to be developed so that it could provide the user with statistics and information that would assist them in identifying areas in which they struggle. Additionally, it could highlight possible methods they could adopt to improve this and give feedback on whether these methods are working. This data could include:

- **Mastered Flashcards:** Identifies which flashcards they have gotten repeatedly correct
- **Results Overview:** A line chart showing a summary of the scores obtained in the last day, week or month. It would illustrate the progress being made towards knowing all the flashcards in the deck.
- **Date Since Last Quiz:** The length of time since the last being quiz.
- **Questions or Topics Struggled With:** Identifies the types of questions or topics that the user frequently gets more wrong

This would all be displayed on the results page of each deck in the form of a customizable dashboard and would allow the user to isolate trends and tendencies in their learning patterns that they could use to study more effectively.

6.7 Feature Of Pre-loading Flashcards

As evidenced in section 2.5.2, students were much more likely to use mobile flashcards if they were made and pre-loaded into their phones. Based on this conclusion, I would look to implement a sharing flashcards function in which teachers or students could share flashcards with those in their class. As a result, students would have more motivation to use the system as they could spend less effort making flashcards and more time effectively learning them.. Furthermore, teachers would be incentivised to use the app as they could ensure that students are learning in a format that they have personally laid out for them and they would have more control over what information they absorb.

6.8 Teacher's Dashboard

The final plan for future developments in chapter 6 combines many of the ideas from section 6.5 and 6.6 and applies them more directly to a classroom environment in which teachers attempt to judge how effectively their students are performing and which concepts they struggle with. A teacher's dashboard would look to solve this issue by creating a new feature on the app which involves the following steps:

1. Students enroll in a class organised by their teacher
2. The teacher assigns flashcards, quizzes and other educational resources for the students to learn
3. The results and statistics for each individual student are recorded and analysed by the system. The teacher can review this information on their dashboard on an individual student-by-student basis or at an entire class wide level.

The applications and benefits for this feature if introduced to the app and applied within colleges or schools would be:

- **The identification of learning styles:** The learning styles of each individual student and the class as a whole could be assessed subject by subject and the teacher could make adjustments to their teaching styles accordingly to overcome these obstacles of learning.

  ### Example: Individual Student

  John's recent quiz scores indicate that he is struggling with ancient history and that he is a *'visual'* learner. His teacher reads this on his dashboard and as a result uses more *'visual'* resources and cues when addressing John during class.

He is also made a personal set of *‘visual’* flashcards depicting information and facts regarding ancient roman history to improve his comprehension on that particular subject matter

Example: Class

Throughout the class it is indicated by the dashboard that the *‘auditory’* style questions in the geography quizzes are being poorly answered. The teacher can address this directly by organising more in class discussions to improve their auditory skills or indirectly by deciding to adopting a more *‘reading/writing’* approach.

- **Identification of problem questions:** Teachers can instantly analyse questions or sets of questions which students have answered incorrectly and if any trends emerge on student or class level

  Example: Individual Student

  Tommy always struggles with accounting questions during the business quizzes. His teacher’s response is to actively get Tommy more engaged when solving accounting questions by asking him to talk through examples during class.

  Example: Class

  The class often gets the biology quiz questions regarding immunology wrong. The teacher decides to devote 10 mins of the following class to revise this topic and improve its comprehension.

- **Assessment of how much work a student truly does:** The teacher can judge and appraise how hard a student is actually working.

  Example:

  During a parent teacher conference, Sarah’s parents want to know why she is receiving a C grade in Chemistry when she averages B’s in every other class. The teacher can pull up her quiz frequency and show the parents that Sarah actively avoids the subject of chemistry by only performing one quiz a week.

6.9 Conclusion On Future Developments

Extensibility is a non-functional requirement that was always very important to the author right throughout the development. This is evidenced by the high number of feasible and credible ideas that were presented in this chapter, with some being very achievable in

the new future. Overall the author was excited to see how these future plans for the app develop.

Chapter 7. Conclusion

The author has benefited from many new experiences throughout the process. The main benefits that were gained through this experience were, an improved level of programming knowledge, working with a whole new development framework and a better understanding of the entire development process. The author learned how to create user models, adaptive learning systems and design a system which benefits the learners. Also, despite working on several software development projects throughout the past four years, this has been the author's first opportunity to have complete control over the entire project from start to finish.

Overall the author deems this project a success. The majority of the functional requirements outlined in Chapter 3 were implemented successfully and the learning opportunity that this project provided enabled the author to learn more about cross platform development using React Native and Expo ,along with significantly improving the authors level of JavaScript.

Bibliography

Baddeley, A. (1992). Working memory. *Science*, 255(5044), pp.556-559.

Bryson, D. (2012). Using Flashcards to Support Your Learning. *Journal of Visual Communication in Medicine*, 35(1), pp.25-29.

Carpenter, S., Pashler, H. and Vul, E. (2006). What types of learning are enhanced by a cued recall test?. *Psychonomic Bulletin & Review*, 13(5), pp.826-830.

Cepeda, N., Pashler, H., Vul, E., Wixted, J. and Rohrer, D. (2006). Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological Bulletin*, 132(3), pp.354-380.

Dean, J. (2012). Smartphone user survey: A glimpse into the mobile lives of college students. Retrieved from http://testkitchen.colorado.edu/projects/reports/smartphone/smartphonesurvey/

El-Hussein, M. and C. Cronje, J. (2010). Defining Mobile Learning in the Higher Education Landscape. *Journal of Educational Technology & Society*, 13(3), pp.12-21.

Godwin-Jones, R. (2010). Emerging Technologies: From Memory Palaces To Spacing Algorithims: Approaches To Second-Language Vocabulary Learning. *Language Learning & Technology*, 14(2), pp.4-11

Golding, J., Wasarhaley, N. and Fletcher, B. (2012). The Use of Flashcards in an Introduction to Psychology Class. *Teaching of Psychology*, 39(3), pp.199-202..

Hernández-Leo, D. (2013). Scaling up learning for sustained impact. Berlin: Springer, pp.629-630.

Karpicke, J. and Roediger, H. (2008). The Critical Importance of Retrieval for Learning. *Science*, 319(5865), pp.966-968.

Kerr, P. (2015). Adaptive learning:. *ELT Journal*, 70(1), pp.88-93.

Kornell, N. (2009). Optimising learning using flashcards: Spacing is more effective than cramming. *Applied Cognitive Psychology*, 23(9), pp.1297-1317.

Kornell, N. and Bjork, R. (2008). Optimising self-regulated study: The benefits—and costs—of dropping flashcards. *Memory*, 16(2), pp.125-136.

Luíz Faria, A., Vaz de Carvalho, C. and Carrapatoso, E. (2008). User Modeling in Adaptive Hypermedia Educational Systems. *Journal of Educational Technology & Society*, 11(1), pp.194-207.

Prithishkumar, I. and Michael, S. (2014). Understanding your student: Using the VARK model. *Journal of Postgraduate Medicine*, 60(2), p.183.

Pyc, M. and Rawson, K. (2009). Testing the retrieval effort hypothesis: Does greater difficulty correctly recalling information lead to higher levels of memory?. *Journal of Memory and Language*, 60(4), pp.437-447.

Pyc, M. and Rawson, K. (2009). Testing the retrieval effort hypothesis: Does greater difficulty correctly recalling information lead to higher levels of memory?. *Journal of Memory and Language*, 60(4), pp.437-447.

Quizlet. (2018). A new milestone for Quizlet: 50 million monthly learners. [online] Available at: https://quizlet.com/blog/a-new-milestone-for-quizlet-50-million-monthly-learners [Accessed 12 Apr. 2019].

Roediger, H. and Butler, A. (2011). The critical role of retrieval practice in long-term retention. *Trends in Cognitive Sciences*, 15(1), pp.20-27.

Usability. System Usability Scale (SUS). Available at: https://www.usability.gov/how-toand-tools/methods/system-usability-scale.html (Accessed: 023 March 2019).

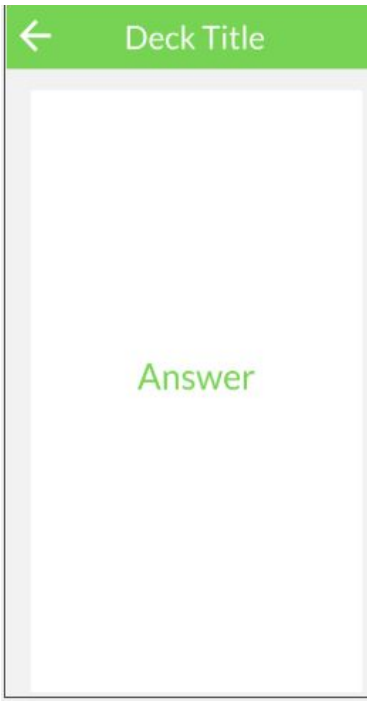
Vaughn, K. and Rawson, K. (2011). Diagnosing Criterion-Level Effects on Memory. *Psychological Science*, 22(9), pp.1127-1131.

Wissman, K., Rawson, K. and Pyc, M. (2012). How and when do students use flashcards?. *Memory*, 20(6), pp.568-579.

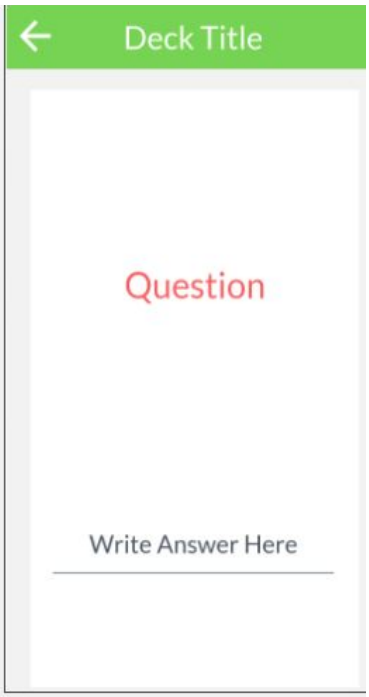
Appendices

Appendix 1: High Fidelity Prototypes

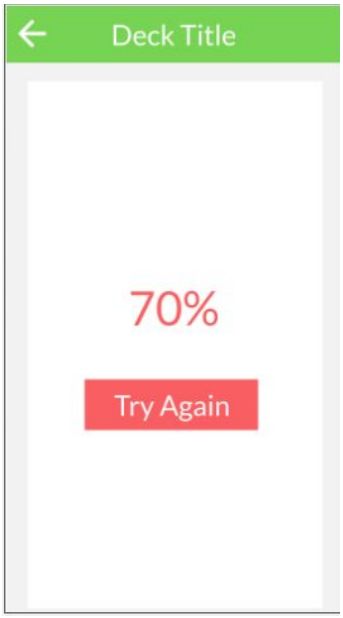
Flip Card Face Component High Fidelity Prototype



Flip Card Back Component High Fidelity Prototype



Written Input Card Component High Fidelity Prototype



Results Component High Fidelity Prototype

Appendix 3

Outline of the Research Proposal

Title of project: Adaptive Learning Flashcards App

Purpose of project including academic rationale

The purpose of the project is to design, create and evaluate an application that will allow users to design, create and structure electronic flashcards in a way that is most conducive for learning. They can them test themselves using these flashcards indicating which cards they've gotten right and which ones they've gotten wrong. It is adaptive in the way that it will test the users on the flashcards they more frequently get wrong. It will involve the creation of flashcard sets which the user can share externally with other users. This study will require participants to use the web application for a minimum of 10 minutes and to provide feedback on the web application. Gathering these users' evaluations and observations about the app is essential to identify areas of the application which require improvement.

Brief description of methods and measurements to be used

Participants will be required to test and review the app. This will involve spending 10 minutes using the app on the first day. Specifically, participants will be asked to study a set of pre-prepared flashcards and create a set of their own flashcards. Following this, the participant will be asked to fill out a Qualtrics survey (5 minutes), which will provide feedback about their experience, fundamentally based on the Standard Usability Scale (SUS). There are no anticipated risks for the participant. The website may record data about how participants have used the website but this data will be anonymous and not linked to the survey response.

Participants

Participants will be volunteer students from Trinity College who will be contacted via social media and email and recruited in a non-discriminatory manner.

Ethical considerations

The only ethical considerations are to do with volunteering and data collection. All participants are volunteers and therefore have the right to withhold information or refuse to participate in certain aspects of the study. All information gathered from each participant will be disclosed to that participant and they will be allowed to have it deleted. All information will be used for my final year project then destroyed. All participants will remain anonymous.