University of Glasgow | School of Computing Science

A Junior Tennis Diary

Andrew Lau - 1105510L

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 27, 2015

Abstract

Understanding and gauging long term progress is often difficult for junior tennis players from the perspective of the player and also their coach. An Android application called *Tennis Tracker* was developed to solve this problem. *Tennis Tracker* allows players to record their day-to-day thoughts and comments on training and also record results of matches they have played with the expectation that these will be viewed over a long period of time. The ability for coaches to view their players' diary and result entries is also included to facilitate more frequent feedback from the player's coach. *Tennis Tracker* is developed to support the Android 5.0 operating system and is designed for devices capable of running Android 5.0.

Acknowledgements

I would like to thank my project supervisor Dr. Simon Rogers for his guidance, knowledge and support throughout the project. I would also like to thank Julie Gordon for participating in the evaluation and helping define the requirements for the system. Finally, I would also like to thank the participants who took part in the evaluation of the system.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: ____________________ Signature: ____________________

Contents

Chapter 1

Introduction

1.1 Aims

The aim of this project is to develop an Android mobile application which allows junior tennis players to record and track their progress over time using diary and result entries. To do this, there will be two main features – recording of diary entries, and result entries. Within the diary entries, players should be able to record information such as their diet, sleep patterns and comments on training. Using these features, the user will be able to look back on results of previous matches and also the diary entries during that time to understand how they might improve. It is essential that the application improves on the currently existing solutions for recording progress in tennis matches but also specifically meet the needs of junior tennis players. This will be provided through the ability for players and coaches to interact. By allowing this interaction, it not only increases the level of communication between coach and player, but also allows crucial feedback to be given to the player on what to improve – the ultimate aim of using the application.

To summarise, the application should have the following set of base features:

- Allow players to record and view previous diary entries
- Allow players to record and view previous result entries
- Increase interaction between player and coach by allowing them to view diary and result entries their players enter, and also facilitate the asking of questions

1.2 Background

1.2.1 Long-term tracking

In the long term, tracking how specific aspects of a tennis player's abilities has improved can be difficult due to the limited amount of information available from looking at match scores alone. While there are existing applications which allow players to record scores [6][7], the ability to complement these with additional information is lacking. Adding the options to record text diary entries and answer questions asked by their coach increases the level of detail of information available. This potentially allows coaches to determine how to improve performance based on content of the diary entries and the answers to questions they have asked. For example, coaches could use the diary entries to demonstrate a correlation between certain diets and sleep patterns, and overall scores in tennis matches. If the coach has any further queries that arise from looking at this extra information, they can directly question the player. They may ask questions to understand how the player thinks they have performed or questions relating to diet and sleep in order to better understand the next steps to take in training. An existing web application called ‘MyTrainingDiary’ illustrates the demand for this kind of application with testimonials from multiple high profile tennis players [1].

1.2.2 Why Mobile?

There has been a significant increase in the use of mobile devices over the last decade, with the mobile market being dominated by two operating systems – iOS and Android. Currently, Android has a significant share of the market with 76.6% of mobile devices running Android [2]. This makes it an ideal platform to develop for as it has the potential to reach a very large user base if the application is published on the Google Play Store.

An iOS version could potentially be developed, however the first version of the application will be targeted towards Android users. This decision was made mainly due to the significant market share of Android devices and also the time investment it would require to develop for an additional platform.

The incorporation of data capture into our everyday lives has been increasing significantly, so much so there is now the notion of the ‘quantified self’ [3]. This describes a movement to integrate self data capture in everyday aspects of life – or in this case, tennis training, to allow improvements over time. An existing example of data capture in tennis is the use of sensors in tennis rackets by *Babolat* to capture information such as power, impact locator, type and number of strokes [4]. In theory the data from these tennis rackets could be added to *Tennis Tracker* to allow a more comprehensive record of a tennis player’s activities. This demand for capturing data presents an ideal opportunity to create a mobile application facilitating this.

1.3 Motivation

Due to the rapid rate at which junior tennis players progress, it is often difficult to gauge the player’s performance over time. At the time of writing, although there exists applications which allow tennis players to record and view their progress, these are not aimed at mobile users. These applications also do not promote the on-going interaction between player and coach off-court. *Tennis Tracker* aims to solve this by providing a mobile application which allows junior tennis players to record their day-to-day training activities while also allowing coaches to view this information as it is added by the player.

Other mobile applications only target a subset of the features offered by *Tennis Tracker*, namely recording diary entries or recording results of matches, but not both. Some applications are also web based and cannot easily be used on a mobile device. By developing a mobile application, it allows users to access the application at any time making it significantly more convenient than using a web application developed specifically for web browsers. We are also targeting a growing market. Smartphone usage is on the rise for those under the age of 18 [5] – a typical age for junior tennis players. The number of mobile users in the demographic of coaches has also increased exponentially since 2007 [5]. This presents mobile as an ideal platform to develop for in order to capitalise on this increased growth.

This project also provides an ideal opportunity to learn the technologies used to develop Android applications, namely the official IDE, Android Studio, Android-specific Java, and also the application structure of Android development. By the end of the project, a high level of competency should have been gained in the development of Android applications.

1.4 Report Content

The following is an outline of the structure of the report.

- Chapter 2 gives a detailed account of the requirements gathering process together with a list of the finalised requirements.
- Chapter 3 discusses of the architecture and user interface design of *Tennis Tracker*.
- Chapter 4 discusses the implementation process of *Tennis Tracker* in detail.
- Chapter 5 describes the evaluation performed, feedback received, and actions taken.
- Chapter 6 concludes the report, details future work and discusses lessons learned.

Chapter 2

Requirements

2.1 Requirements Capture

2.1.1 Initial Requirements

When initially presented with the project, there were a number of basic requirements. These were as follows:

- Allow recording of results of matches
- Allow recording of diary entries
- View diary and results entries over a long period to gauge improvement
- Develop the application for iOS or Android

With these requirements as a starting point, several requirements gathering techniques had to be utilised in order to ensure all the needs of the client were elicited.

2.1.2 Meetings with supervisor and client

The main source of requirements came from frequent meetings with the project supervisor. This allowed a refinement of the initial requirements and also gathering of the further, more detailed requirements for the application.

Following several meetings with the project supervisor, a meeting with the client – Julie, a local tennis coach – was set up. This meeting allowed valuable insight into information which tennis coaches may find useful when coaching junior tennis players. The result of this was the further reinforcement of requirements and also an incremental refinement of the already existing requirements.

2.1.3 Research

In order to understand the scope of the platform chosen, significant research was conducted on the capabilities of the latest version of Android – version 5.0 [9]. As this version was released shortly after starting the requirements process, it was important to incorporate the potential changes in this new version in the final version of the requirements. An understanding also had to be built up of tennis and how junior tennis players train. The gathering of this information allowed features of the application to be incorporated into what was learned. In order to understand the current landscape of the existing solutions, several applications were looked at. These are discussed below.

2.1.4 Existing applications

In order to determine the need for an application which tracks and records progress, a search was conducted to identify applications which provided the features of *Tennis Tracker*. There are some existing Android applications that meet this criteria, however none of them specifically target junior tennis players and interaction between the coach and the player. These applications also do not allow the user to record tennis diary entries and therefore do not provide the player with comprehensive solution to recording their tennis activity. One web-based application provides some similar features to *Tennis Tracker* and provides a suitable starting point for developing a mobile application.

Tennis Math
Tennis Math [6] allows the user to enter results much like *Tennis Tracker* and also allows tracking of matches in real time. This involves the user entering the details of a match while it is going on. This application is specifically focused on the recording of results and does not allow users to enter details about training. This is not the most practical solution since scores cannot be input individually by the player as this would require constant interaction with the application while playing the tennis match. The coach could input this information during the match, however, with each coach potentially coaching multiple players, this is unfeasible.

Tennis Stats
Tennis Stats [7] allows the user to enter results of matches while the match is going on in a way which is very similar to the features in Tennis Math. This application is also focused on the recording of results only and does not promote tracking of progress over time. As this application is focused on recording results, it was used to understand how scores are typically laid out in mobile applications.

MyTrainingDiary
MyTrainingDiary [1] is a web-based application which allows players to record diary entries of their training activities much in the same way as *Tennis Tracker*. It also provides the ability for players to interact with their coach. There are a number of disadvantages to this application, however. Firstly, a web-based application means tennis players must be connected to the internet at all times and secondly, the application requires the player and coach to use a web browser on a desktop computer or laptop. Using the application on a mobile device is difficult if it is not optimised for use on mobile devices – buttons are small and hard to press, and the application is not responsive resulting in scaling issues across the different resolutions of mobile devices.

2.2 Functional Requirements

The following list of requirements were produced following the initial meetings with supervisor and client, research on the latest Android version, and by looking at existing applications. As there is a limited timeframe available for implementation, requirements were prioritised using the MoSCoW method [8].

'Must have' requirements represent requirements that need to be implemented in order to have a minimally functional application. All other requirements are optional. The final requirements of the system following several iterations are detailed below:

2.2.1 Must Have

Table 2.1: Must Have Requirements

FR	Requirement	Description
1	Allow players to record and view diary entries	The player must be able to enter text diary entries which are tagged by date. Once entered the player must be able to view these diary entries in chronological order.
2	Allow players to record and view result entries	Similar to recording and viewing diary entries. Players should be able to see all of their recorded results entries after they have entered them.
3	Allow players to add comments on their performance when recording result entries	When players are adding a result, there should be a page asking them to enter a comment on their performance.
4	Allow users to create user accounts / Log into existing account	Users should be given the option to create a user account when they first start the application. If the user has already created an account, the application should allow the user to log into this using their registered username and password. User accounts should also allow the user to reset their password.
5	Store diary and result entries in the cloud	Once a user account has been created, while the user is logged in, any diary or result entries that the player enters should be saved to that account.
6	Profile pages	Players and coaches should have a profile page showing information about them. This information could include, but is not limited to, games played, won, country and other details about the player.

2.2.2 Should Have

Table 2.2: Should Have Requirements

FR	Requirement	Description
1	Allow coach to ask their players questions	Coaches should have the option to add questions, which the player can answer when adding a result entry. Once the questions have been answered, the coach should be able to see the answers.
2	Allow player to answer questions when recording a result	When recording a result, there should be a page which shows the questions asked by the coach.
3	Allow coach to add players they are coaching	By selecting from players with an account on *Tennis Tracker*, coaches should be able to 'connect'. Players should be notified of this.
4	Allow coach to view diary entries from the players they coach	By selecting from players added, the coach should be able to view all diary entries from these players.
5	Allow coach to view result entries from players they coach	By selecting from players added, the coach should be able to view all result entries from these players.
6	Allow players to view profile pages of other players	There should be an option to search for other players in the system. Selecting these players should show the profile page of the player.

2.2.3 Could Have

Table 2.3: Could Have Requirements

FR	Requirement	Description
1	Notifications	Notifications could be sent in a variety of situations. Some examples of these would be when coach adds a player to their list of players, when a player adds a result, when a player adds a diary entry, and when players answer a question on the results entry page. When these events occur, the relevant users should be informed. Interacting with the notification on the device should take the user to the relevant section in the application. For example, if a player has entered a diary entry, tapping on that notification should take the coach to that diary entry for the player.
2	Allow player to edit previously entered diary and result entries	When viewing previous result and diary entries, there should be an option to change text, date and also delete the diary entry.
3	Ask questions to specific players	Along with asking questions to all players, the coach should also be given the option to select which players they wish to ask questions to. If the coach is asking a specific player a question, only that player should be able to see and answer that question.
4	Searching previous diary entries	After entering diary entries, players should be able to search for keywords in previously entered diary entries. This will filter to only show the diary entries with those keywords.
5	Tagging diary entries to allow filtering	To improve the searching functionality, players could be given the option to tag diary entries when they first record them. This would allow quick filtering of all diary entries related to training, sleep, diet etc.
6	Sorting diary entries by date	Allow filtering options for selecting diary entries by date or sorting in chronological or reverse chronological order.
7	Support for tablet sized screens	While the application is designed for mobile phones, the layouts could be changed to accommodate the extra space provided by tablets.

2.2.4 Would Like to Have

Table 2.4: Would Like to Have Requirements

FR	Requirement	Description
1	Visualisations showing data collected by application	Graphs could be created using the data collected from diary entries, results, and performance.
2	Recommendations based on stats	Based on previous results, the application could recommend which players to play next or what aspects of training the player should focus on.
3	iOS port of application	A native application for iOS could be developed to allow users of iOS to also use the application. This would allow cross-platform interaction between Android and iOS users.

2.3 Non-Functional Requirements

Table 2.5: Non-Functional Requirements

NFR	Requirement	Description
1	Android mobile device support	As Android devices have a wide range of different screen sizes and resolutions, the application should adapt to these differences.
2	Android 4.2.2 Support	Supporting devices from 4.2.2 upwards ensures that the application is compatible with the majority of Android Mobile devices. While supporting older versions of Android is possible, devices running version 4.2.2.+ represent 68% Android devices with market share of previous versions decreasing rapidly [10].
3	Android 5.0 Support	Support for the latest version of Android (Android 5.0) ensures that the newest features and technologies are included in the application. This also ensures that the application will be usable by users of the newest Android mobile devices. As of March 2nd, 2015, 3.3% of Android devices run Android 5.0 [10].
4	Offline support	While the first use of the application requires the user to be connected to the internet, a network connection will not always be available. The application should be functional even without a wifi or network connection. When a network connection is available, any data stored locally on the device should be synced. Allowing some of the features of the application to be used without an internet connection will ensure that the application is useful to users in all situations.
5	Concurrent user support	The application is made with other users in mind and should allow more than one user to use the application concurrently. Concurrent use of the application is handled by Parse [11]. This is limited by the max number of requests the free tier in Parse can process. At the time of writing, this limit is 30 requests per second [11].
6	Extensibility	The application should be designed in a way to allow the adding of features in the future. As Parse supports multiple platforms, it allows applications on platforms other than Android to be implemented while using the same backend setup.

Chapter 3

Design

This section discusses the design process for *Tennis Tracker* and will be separated into two main sections – architecture and user interface. Architecture will discuss the general structure of the application including how backend elements interact. User interface will discuss the user interface design process in detail from initial paper prototypes to final implementation.

3.1 Application Architecture

The application makes extensive use of the Model-View-Controller architecture. This is one of the most commonly used architectures for Android applications and represents the foundation of many applications which revolve around user interaction.

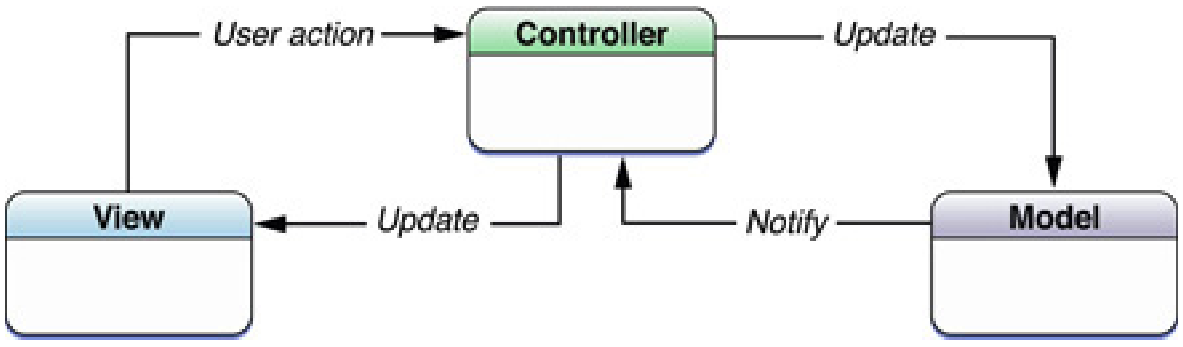


Figure 3.1: Model-View-Controller

Model

The backend of the application is handled by Parse [11]. Use of Parse is described in more detail in the implementation section. All data processed within the application is written and retrieved to cloud storage on Parse. This data is then passed to the appropriate classes in order to display.

View

The view represents what the user will see on their screen. The user can interact with the view and these actions will be interpreted by the controller. Updates to the view are in turn handled by the controller and conveyed.

The views in the application are represented using XML layout files [12]. These describe the main components which make up that that particular view. In most cases, the components of the screen are defined in the layout file, however these may also be added programmatically. Each screen in the application has a layout file and a corresponding class handling the logic, with any changes in the model being processed by the controller and then passed to the view to display.

Controller

The controller processes the changes in the model and propagates these to the relevant views. Similarly, any interaction with these views is also handled by the controller, with any updates being passed to the model. Each view has a corresponding controller class which handles the logic – this is typically referred to as a View-Controller pair as these components are dependent on each other.

3.2 Application Structure

3.2.1 Parse

Parse [11] is used as the cloud backend component for *Tennis Tracker*. All data used by the application is stored in Parse with the option to retrieve data using Parse requests. Also, in order to provide offline access, data needs to be stored persistently on the device. The `LocalDatastore` [13] functionality will be used to cache this data on the device so that future launches of the application query the data stored locally instead of in the Parse cloud.

Parse Login Structure

Tennis Tracker uses what is known as `ParseUser` objects to store information specific to each player in the system. When a user creates a new account on *Tennis Tracker*, a new `ParseUser` object is created with the user defined credentials, and stored in the Parse datacloud. As *Tennis Tracker* requires a `ParseUser` to function, it is necessary to check for an existing `ParseUser` before starting the main activity of the application. This involves using a `DispatchActivity` [14] to protect the application. Only when there is a valid `ParseUser` does the application proceed to launch the main activity. A high level representation of this structure is shown below. As we can see from figure 3.2, the application is protected by a dispatch activity. *Tennis Tracker* uses a similar structure for log in.

When the user launches the application, they are directed to a dispatch activity. This activity's purpose is to check whether there is a valid `ParseUser` instance available. If not, the user is directed to the login activity for the application. If yes, the user is directed to the initial starting activity.

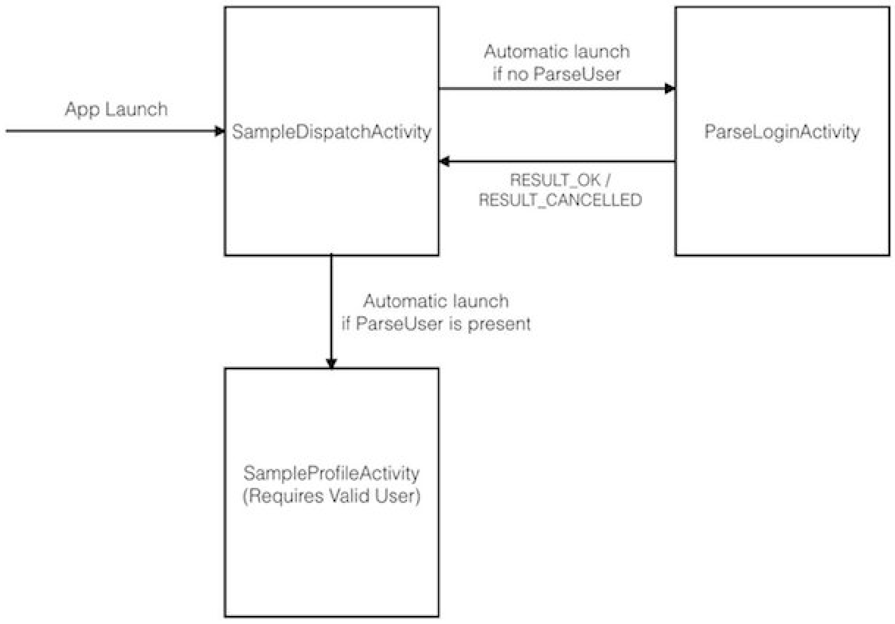


Figure 3.2: Parse Login with Dispatch

3.2.2 Navigation Drawer

The way in which the user navigates around the application is an important aspect and as such, two main navigation methods were considered: a tab bar with multiple buttons, each representing a different page, and a navigation drawer, with a list of the options available. There are some notable differences between these two options. When using tabs, the options are immediately accessible to the user and only requires a single tap on the screen to move to the desired page. Using a navigation drawer requires the user to activate the drawer before they can move to the desired page. It was ultimately decided to use the navigation drawer for three reasons:

1. Android documentation recommends use of the navigation drawer when there are more than three top level views [15].
2. With a navigation drawer, there is more scope for extensibility as tab bars are recommended to be limited to three tabs. The navigation controller allows access to many more than three views.
3. Tab bars are permanently on screen, therefore reducing the amount of usable space for the user. This is an important consideration due to the wide range of screen sizes available on Android with some as small as 3.5”.

3.3 User Interface Design

3.3.1 Designing for Android 5.0

Android 5.0 was released shortly after beginning the project and as a result, changes had to be incorporated into the design stage of the application to match the changes in the updated user interface guidelines.

Tennis Tracker is designed with Material Design guidelines in mind. Material Design is Google's new visual design language introduced with Android 5.0 [16]. One of the main goals is to provide an underlying system which allows for a unified experience across multiple platforms and device sizes. By implementing these guidelines, it allows *Tennis Tracker* to be potentially expanded to multiple platforms. This design language simulates interaction with paper in the real world and attempts to translate this to digital while emphasising the use of animations to provide context to the user. When designing *Tennis Tracker*, these guidelines were taken into consideration and where possible, follow the rules set out by Google. There are many guidelines defined in the specification for Material Design [16]. The main principles are listed below:

Material is the metaphor. This refers to the interaction of the user with what's on the screen. The interaction should be 'grounded in tactile reality'. The interactions should be similar to what users are used to in reality.

Bold, graphic, intentional. Emphasis should be placed on the main elements of design – typography, grids, space, scale, color, and use of imagery.

Motion provides meaning. Motion is meaningful and appropriate, serving to focus attention and maintain continuity. Feedback is subtle yet clear. Transitions are efficient yet coherent. A focus on animations to guide the user to the important aspects of the interface.

With the release of Android 5.0, an increased emphasis has been placed on effective design and also design which adheres to the guidelines set out by Google. Therefore, it is important not to overlook these aspects when aspiring to develop an application which succeeds in all areas.

As Material Design was introduced in Android 5.0, previous versions of Android feature different styling and fewer animations. An emphasis is placed on animation, transitions and shadows in Android 5.0. Incorporating these key features was a priority when designing *Tennis Tracker* in order to follow the guidelines set out by Google and also embrace the new design style of Android 5.0.

3.3.2 Colours

Colour in Material Design

In Material Design, there are new guidelines for colours [17]. Google provides colour palettes with colours specifically selected to work well together, consisting of a primary and secondary colour palettes. These colour palettes have colours designed specifically for separate components such as text fields accents. It is recommended that colour selection be limited to three colour hues from the primary palette and an accent colour from the secondary palette. *Tennis Tracker* implements these guidelines by using different shades of colour selected from primary and secondary palettes to provide a distinction between accents on fields and the main UI colour.

Status bar colours

According to the Material Design guidelines, the status bar should almost always have a clear delineation from the main toolbar. This has been reflected in *Tennis Tracker* in a subtle change of the status bar colour to a darker shade. The action bar uses a primary 500 colour whereas the status bar uses a darker primary 700 colour.

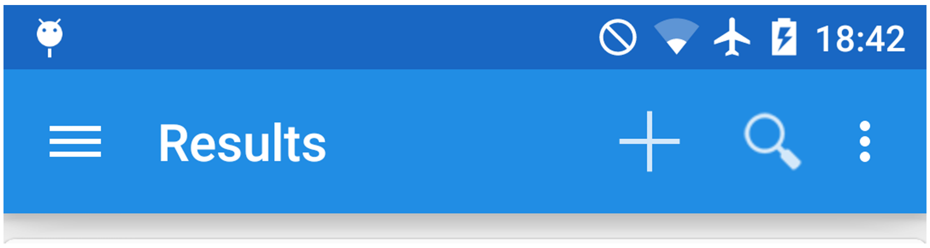


Figure 3.3: Darker colour status bar in Android 5.0

3.3.3 Animations

An emphasis has been placed on meaningful transitions and animations in Android 5.0 design [18]. When layouts change or elements are rearranged, the transitions used should not confuse the user but rather aid them through the process.

Fragment Transitions

The default transition from fragment to fragment uses no animation and simply cuts between screens – specifically what the guidelines advise against. In order to alleviate this issue, a custom animator has been used to provide a more gradual transition between fragments. An example of this transition is in the players page for coaches. When a coach selects a player while using the custom animator, the page will gradually fade to show the details for that player.

3.4 Prototypes

In order to reach the design of the final application, several possible user interface designs were considered. The best features from these initial designs were then carried over to the final implementation.

3.4.1 Paper Prototypes

To begin, some initial paper prototypes were sketched in order to map out the different features in the application according to the requirements. These designs were then iterated and refined. This resulted in more detailed prototypes of each of the individual pages in the application, as well as designs for the coach version of the application. The initial sketches for both the player and coach versions of the application are shown in figures 3.4 and 3.5.

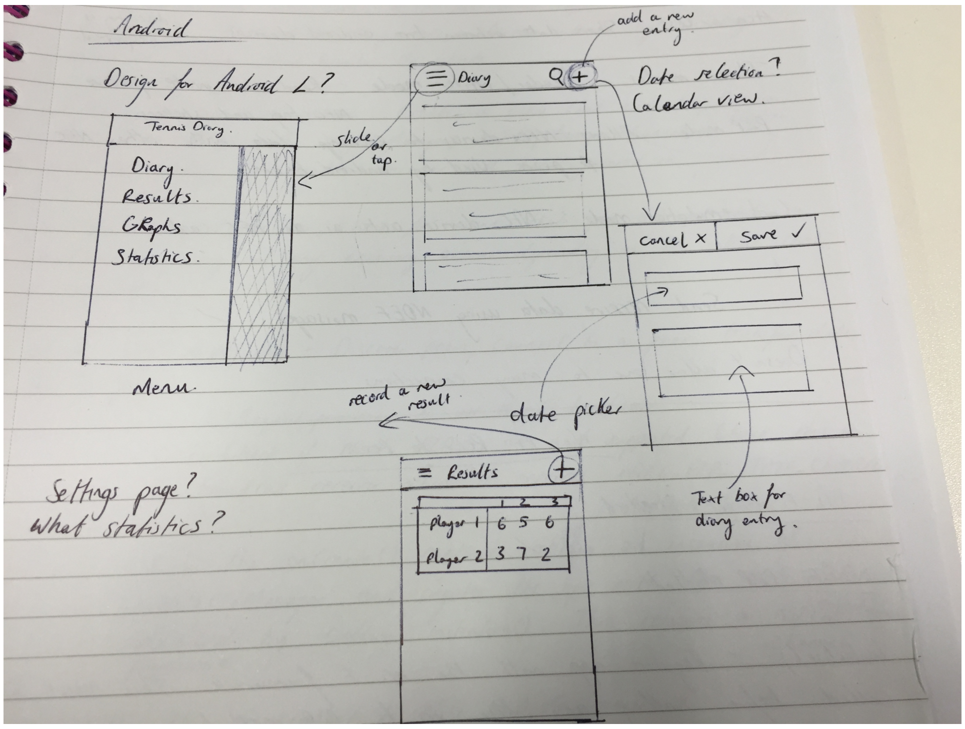
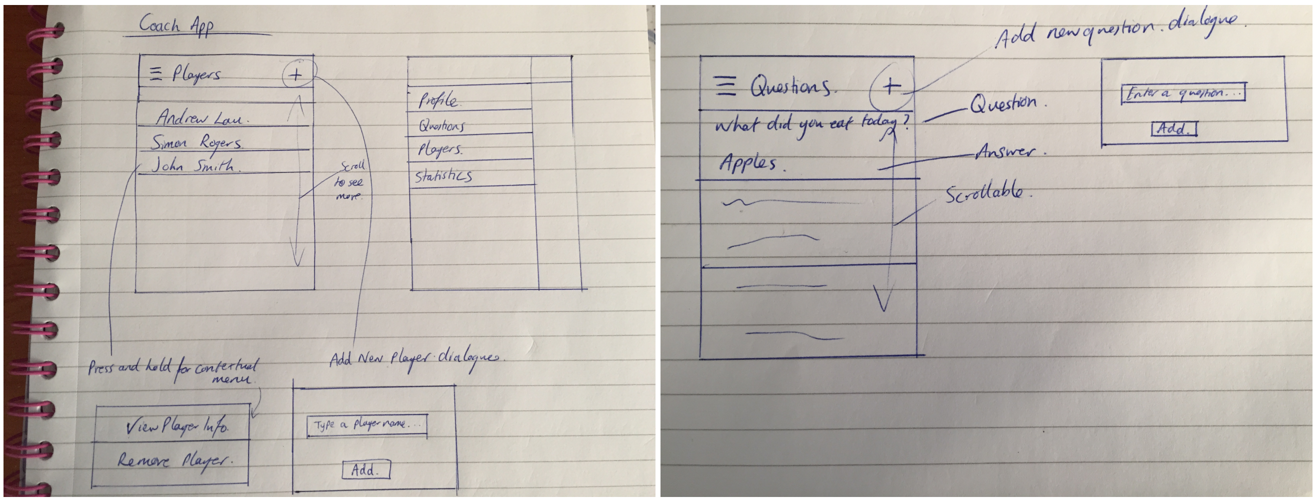


Figure 3.4: Initial Paper Prototype – Player

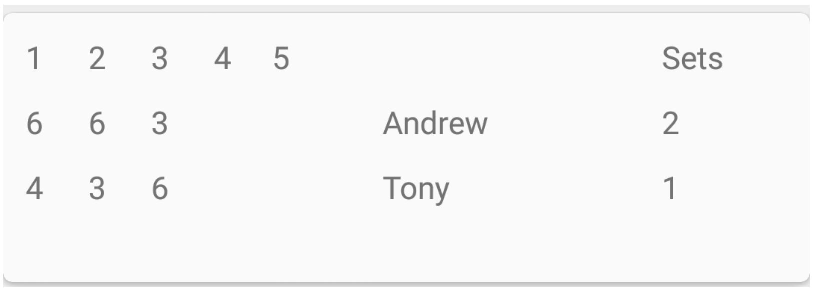


(a) Players page and menu paper prototype (b) Questions page paper prototype

Figure 3.5: Initial Paper Prototype – coach

Score card design

In order to display the scores entered by the user in the results page, score cards had to be designed to show all the necessary information in a format that is familiar and also easy to understand. As shown in the initial paper prototype for the scorecard, a minimalistic design was decided on, showing only the necessary information to the user. During development, this was iterated to include information such as tiebreak scores and number of sets each player has won. An image of the final scorecard is shown below.

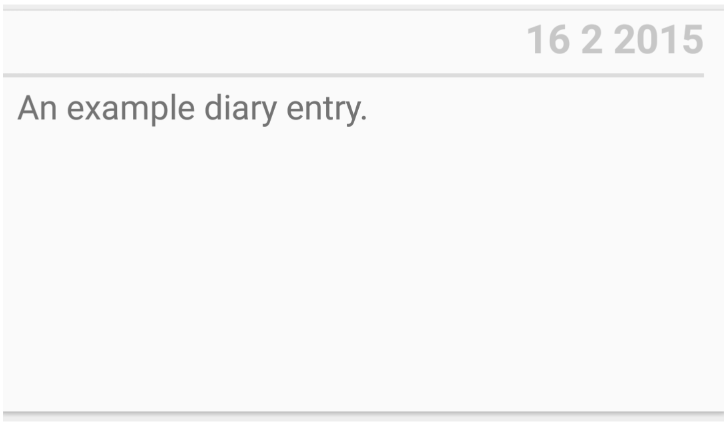


1	2	3	4	5		Sets
6	6	3			Andrew	2
4	3	6			Tony	1

Figure 3.6: Final scorecard design

Diary card design

At this stage, it was decided to use `CardView` [19] widgets to represent each of the diary entries. Introduced in Android 5.0, `CardView` allows a consistent look and feel across all diary entries while also retaining the Material Design look and feel. Previous versions of Android also supported cards using third party libraries. Support for `CardView` is provided using the `AppCompatv7` compatibility library. A single diary entry card can be seen in figure 3.7.



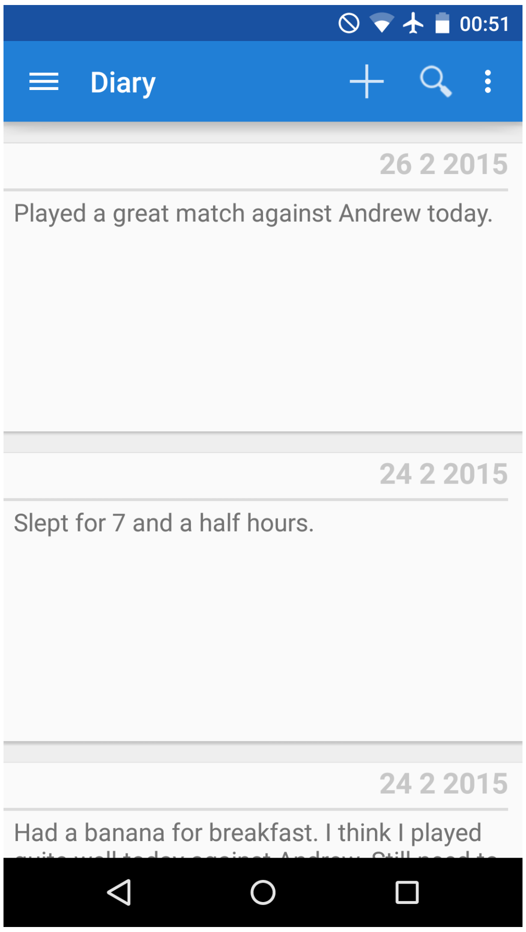
16 2 2015

An example diary entry.

Figure 3.7: Final diary card design

3.5 Final Implemented Design

The initial design shows many similarities with the final application. For example, the positioning of buttons and layout of diary and result entries has remained consistent. The final design for the diary page can be seen in figure 3.8a. As application design was continued well into the implementation process, certain design elements decided on at the design stage of the project have been iterated extensively, resulting in significant changes. One example is seen in the final design for the coach questions page shown in figure 3.8b. As a result of feedback from evaluation, the decision was made to add tabs to the questions page to enable a separation between questions for all players and questions for specific players. Similarly, tabs are used when viewing the diary, results and questions of players in the players page. As users were often unaware of the swipe gesture to navigate between views, this was added to improve the user experience and provide explicit direction to the user.

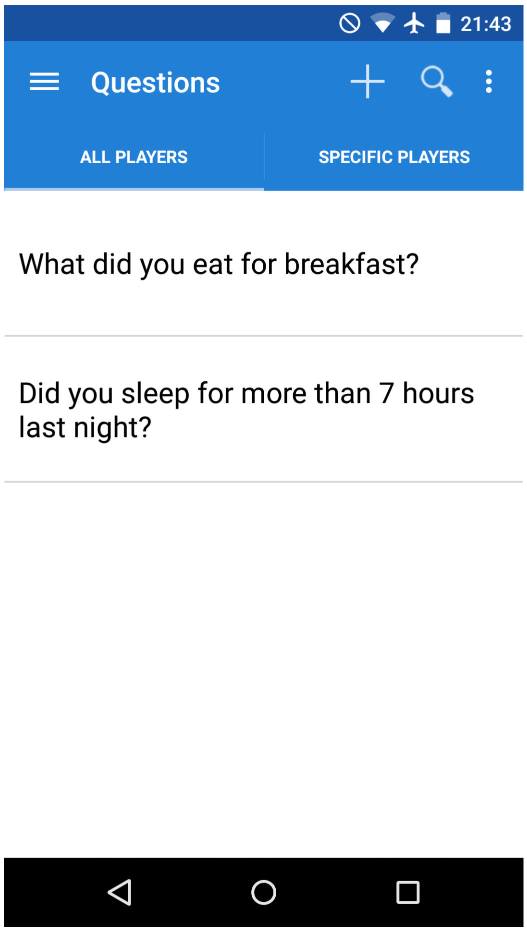


26 2 2015
Played a great match against Andrew today.

24 2 2015
Slept for 7 and a half hours.

24 2 2015
Had a banana for breakfast. I think I played

(a) Final diary page design



What did you eat for breakfast?

Did you sleep for more than 7 hours last night?

(b) Final questions page design

Figure 3.8: Final implemented designs

3.6 Design Decisions

3.6.1 Search for players

Following on from evaluation with the client, it was decided to add a page which allows searching for other players. The design for this page features many similarities with Google's stock Android applications such as the Google Play Music application, which features searching. These applications were used as a starting point in designing the search interface for *Tennis Tracker*. As shown in the screenshots of the application, a search icon has been added to the action bar. Tapping on this transitions the user to another screen which they can search from. In order to increase the speed at which users can search for players, the search box starts in an expanded state as opposed to hidden. A screenshot of this page is shown in figure 3.9.

3.6.2 Cloud Backend

To enable storage of data in the cloud, Parse was used. Parse [11] is a cloud backend solution which allows the storage and retrieval of data through requests and queries. A cloud backend was selected due to the fact that interaction between users and the ability to store data which can then be accessed anywhere was a requirement of the application. If a local database had been chosen to store the data, allowing this functionality in the future would have been challenging, requiring a restructuring of the backend of the application. Support for additional

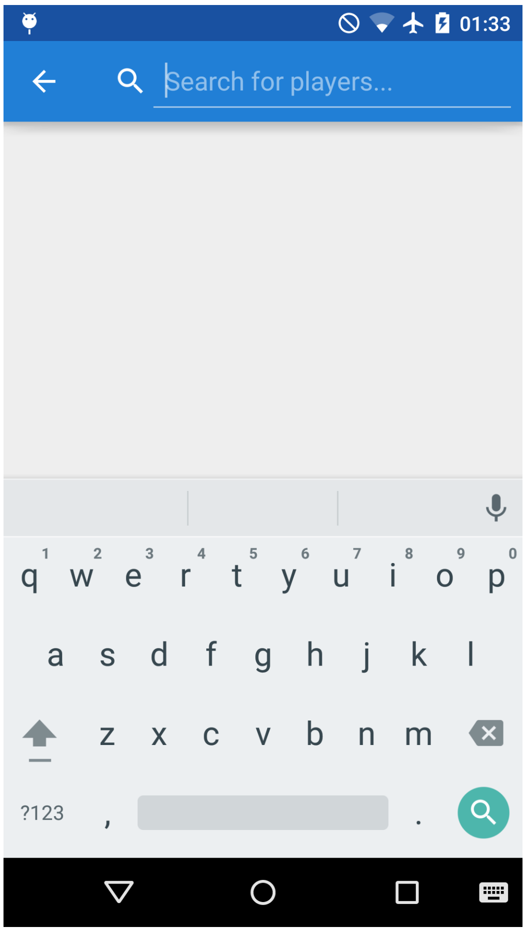


Figure 3.9: Player search page

features such as push notifications and user accounts was also desirable and necessary to the core functionality of *Tennis Tracker*. These features are provided natively by Parse. Extensibility was also a major factor in the choice to use Parse. As the Parse SDK provides native support for multiple platforms such as Android, iOS and Web, different applications utilising the same cloud backend can be developed.

3.6.3 Support for previous versions of Android

As Android 5.0 was released at the start of the development cycle, it was decided to design and implement for Android 5.0 devices. This decision meant that the majority of the existing Android user base would not be able to use the application. As of March 2nd, 3.3% of Android devices are running the latest version of Android – Lollipop [10]. In order to provide compatibility for users of previous Android versions, the Android Support Library was used. This is a library provided by Google and used to support previous versions of Android while also allowing the latest features on the devices which support them. Specifically, this support is provided through backward-compatible versions of Android framework APIs [20]. The decision to support previous versions of Android introduces a number of complications. As Lollipop introduces a new design language, designs for previous versions of Android do not incorporate new design methodologies and therefore need to be adapted to match previous design standards. Supporting a wider range of devices also entails supporting a much wider set of screen sizes, resolutions, and densities. Design in Android has evolved to now provide tools to support this diversity, with many components in the Google support libraries implementing Material Design.

3.6.4 User Interface Iteration

As Google promotes user interface design as one of the most important features of an application developed for Android 5.0, a significant portion of development time was allocated to development and iteration of the user interface. While a preliminary user interface design was completed at the design stage, design iteration was continued well into the implementation stage. By continuing to iterate the design, it allowed for a sufficient number of iterations necessary to improve the application to the standard described in the Google guidelines and also allowed for feedback from evaluation to be continuously incorporated.

3.6.5 Tablet Support

Tennis Tracker has been designed specifically for phone-sized screens, however, the UI elements scale to fill the screen if the application is run on a tablet screen. Applications which are optimised for tablets typically make use of the extra screen space and releasing both a tablet and phone application would entail two separate UI layouts – effectively doubling the length of the implementation and design stages. Tablets are also commonly used in landscape mode and therefore a dedicated landscape layout would also need to be created.

3.6.6 User Experience

As the application targets a potentially wide range of users with varying knowledge of mobile applications, it was important that functions in the application were obvious and logical in the steps it takes you through. Due to this, several user experience enhancements were added to aid the user when using the application. A number of the improvements are described below. In addition to these enhancements, Google's Android Design principles were followed closely in order to ensure the application meets the high user experience bar set by Google [21].

Navigation drawer peek

When using the navigation drawer, the drawer 'peeks' out if the user touches the left edge of the screen – within 20dp (density independent pixels) of the left edge. This is useful for new users as it promotes "accidental discovery" and "provides richer feedback" to the user [15].

Navigation drawer swipe gesture

In addition to pressing the navigation drawer icon, the drawer is also accessible using a swipe from the left edge of the screen. This makes it easier for the user to navigate the application when using the device with one hand as the navigation drawer icon at the top left of the screen is often difficult to reach on devices with a large screen size.

Toasts

Toasts are messages which temporarily show on the screen when events happen. The use of toasts provides feedback to the user when they perform actions, eliminating any confusion and ambiguity when the user triggers an event. An example of a toast displayed when the user enters a result entry is shown in figure 3.11.

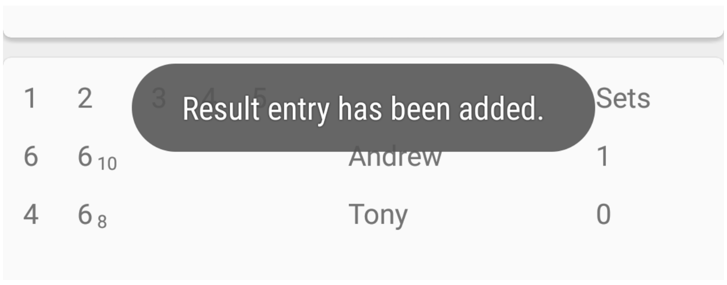


Figure 3.10: Toast message shown when a result entry is added

Use of toasts is not limited to messages describing user-triggered events, they can also be used for unobtrusive labelling of buttons. For example – when pressing and holding on icons on the action bar, a toast is displayed to inform the user of the function of that button.

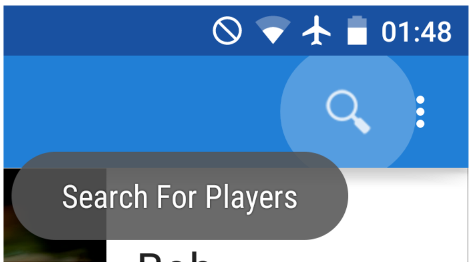


Figure 3.11: Toast message showing function of search button

Chapter 4

Implementation

This section will discuss the implementation details of *Tennis Tracker*. *Tennis Tracker* is implemented in Java – the native programming language for Android applications. The following section will discuss the implementation details for prominent sections of the application.

4.1 Technologies

The development of *Tennis Tracker* used the following main technologies.

4.1.1 Android

Tennis Tracker is an Android application developed for Android 5.0 mobile devices with compatibility support for Android versions down to 4.2.2 through use of the `AppCompatv7` compatibility library [20].

4.1.2 Android Studio

The IDE used for development of *Tennis Tracker* was Android Studio. At the time of writing, Android Studio is the official IDE for Android development and is based on IntelliJ [22]. This is an upgrade over the previous Eclipse-based Android development environment and features a Gradle build system to allow easier dependency management across projects, among other improvements.

4.2 Parse

Tennis Tracker uses Parse – a comprehensive cloud backend solution for web and mobile applications [11]. Parse allows data storage, user account management, push notifications and analytics of applications among other features. *Tennis Tracker* makes extensive use of all of these main functions.

In order to determine the most suitable service, several options which potentially provide the functionality needed for the application were considered. A search for ‘backend cloud services for mobile applications’ returns many results. However, three services were selected as possibilties and are discussed below.

Parse

From reading the documentation and analysing the available feature set, it is clear that Parse is the most comprehensive solution for a cloud backend. Parse [11] allows creation of user accounts, data storage, and multiplatform support. It also features a robust API which should ease the process of integrating its features into the application.

Backendless

Backendless [23] is very similar to Parse in terms of feature set, however not as well-known and less support is available via official documentation.

Appcelerator Cloud Services

Inspection of the documentation provided by Appcelerator [24] reveals a comprehensive documentation and support compared to Parse. Browsing the website gave no clear information on what is supported and what features are available for "free". It is possible that only paid tiers are available, but the information conveyed on the website makes this unclear.

Based on the feature set and comparing with other solutions, Parse was selected as the best service to use when implementing the application as it provides the main features necessary for implementing the main requirements.

4.2.1 Working with Parse

Parse provided the backend solution to *Tennis Tracker*. The following section will detail how *Tennis Tracker* interacts with the Parse backend for data storage and notifications [14].

Initialization

In order to start using Parse, it has to be initialised. To do this, we call the following function:

```
Parse.initialize(this, "appID", "masterKey");
```

The above line of code initializes Parse by passing it the IDs associated with the application. The IDs used are specific to *Tennis Tracker*.

Logging in with Facebook or Twitter

Tennis Tracker allows users to create an account using Facebook or Twitter. As many mobile users have Facebook or Twitter accounts, the option to create an account using publicly available information on Facebook and Twitter is beneficial. This streamlines the account creation and login process for users and potentially opens up the application to more users as sign up and log in is quicker.

Storing Data

Storing data on Parse involves creating a Parse object, populating it and then saving it to Parse. In order to create an object, we call the initialiser for ParseObject.

```
ParseObject entry = new ParseObject("Entry");
```

In this case, a ParseObject with name 'Entry' is being created.

We then add the fields with the values to the ParseObjects. This works similarly to a Map as we have a key which maps to a value.

```
entry.put("Text", diaryText);
entry.put("Date", diaryDate);
entry.put("createdBy", ParseUser.getCurrentUser());
```

Once the object has been created, we store it in the Parse database by calling `saveInBackground()`

```
diaryEntry.saveInBackground();
```

Retrieving Data

Retrieving data follows a similar process to storing data – once again done through Parse requests. There are a number of stages we must go through to retrieve and process the data for display in the diary page. The request and processing is shown in its entirety below.

```
query.findInBackground(new FindCallback<ParseObject>() {
                   public void done(List<ParseObject> diaryEntries,
                       ParseException e) {
                       if (e == null) {
                          Log.d("Entry", "Retrieved " + diaryEntries.size() + "
                              entries");
                          for (ParseObject parseObject: diaryEntries){
                             String diaryText = parseObject.getString("Text");
                             String diaryDate = parseObject.getString("Date");
                             diaryLayout = (LinearLayout)
                                 rootView.findViewById(R.id.diary_layout);
                             CardView card = (CardView)
                                 inflater.inflate(R.layout.diary_card,
                                 diaryLayout, false);
                             TextView textView = (TextView)
                                 card.findViewById(R.id.diary_card_text);
                             TextView cardDate = (TextView)
                                 card.findViewById(R.id.diary_card_date);
                             textView.setText(diaryText);
                             cardDate.setText(diaryDate);
                             diaryLayout.addView(card,0);
                          }
                       } else {
                          Log.d("Entry", "Error: " + e.getMessage());
                       }
                    }
                 });
```

```
                }
```

Once we have performed the request, we must loop through the objects returned to extract the relevant objects and information. For diary entries, we are interested in the text and the date. Using the `getString` methods, we can retrieve these from the Parse object. As we need to add to the diary page, we need to retrieve the relevant layout by calling `findViewById` on the `rootView`. `findViewById` needs to be called on the layout which the target ID exists in. For example, `R.id.diary_layout` is defined within rootView, so this view must be retrieved from this layout. Calling `findViewById` on the wrong layout results in a null pointer exception. A predefined `CardView` layout is used to enable consistency across all diary entries and to avoid dynamic definition of the attributes for the card. In order to use this layout, it must first be inflated. Each of the components can then be set individually as desired. As shown, the text and date fields of the card view are set to the text and date retrieved from Parse. This process is repeated for all diary entries retrieved.

Push notifications

The application makes use of Parse's push notification functionality. This provides a simplified way of targeting notifications at specific devices and user accounts. The following section explains how push notifications are handled in the console of Parse and also how push notifications are created in *Tennis Tracker*.

In order to use push notifications with the application, the application needs to be subscribed to a notification channel. Notification channels make use of the `publisher-subscriber` model. In this case, publishers send notifications for subscribers to receive.

```
    ParsePush.subscribeInBackground("", new SaveCallback() {
        @Override
        public void done(ParseException e) {
            if (e == null) {
                Log.d("com.parse.push", "successfully subscribed to the broadcast
                    channel.");
            } else {
                Log.e("com.parse.push", "failed to subscribe for push", e);
            }
        }
    });
```

This subscribes the application to the broadcast channel, allowing push notifications to be sent either from the Parse console or from within the application. The Parse console uses rules to determine who receives notifications. As this method of sending notifications is basic, creating and sending notifications in-app was chosen instead.

To send a push notification, it first has to be created:

```
ParsePush push = new ParsePush();
```

Once created, parameters must be set in order to determine who receives the push notification:

```
push.setChannel(ParseUser.getCurrentUser().getObjectId());
```

This sets the channel of the push notification to the objectId of the user who is currently logged into the application. The objectId in this case is the unique identifier of the user. This push notification will be sent to all users subscribed to this channel. In this case we are setting the channel to be the user so that their coach receives the notification.

Similarly to set a message for the push notification the following method is called:

```
String userName = ParseUser.getCurrentUser().getString("name");
 push.setMessage(userName + " " + "just added a diary entry");
```

When the notification is received, the notification text will be `userName + " " + "just added a diary entry"`. An example of this notification is shown in figure 4.1.

Finally, to send this notification we call sendInBackground();

```
push.sendInBackground();
```

This sends the push notification while continuing with the main tasks in the thread, hence, ‘InBackground’.

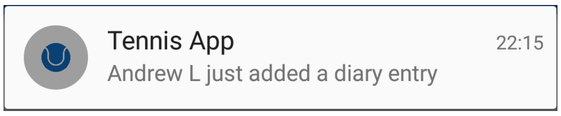
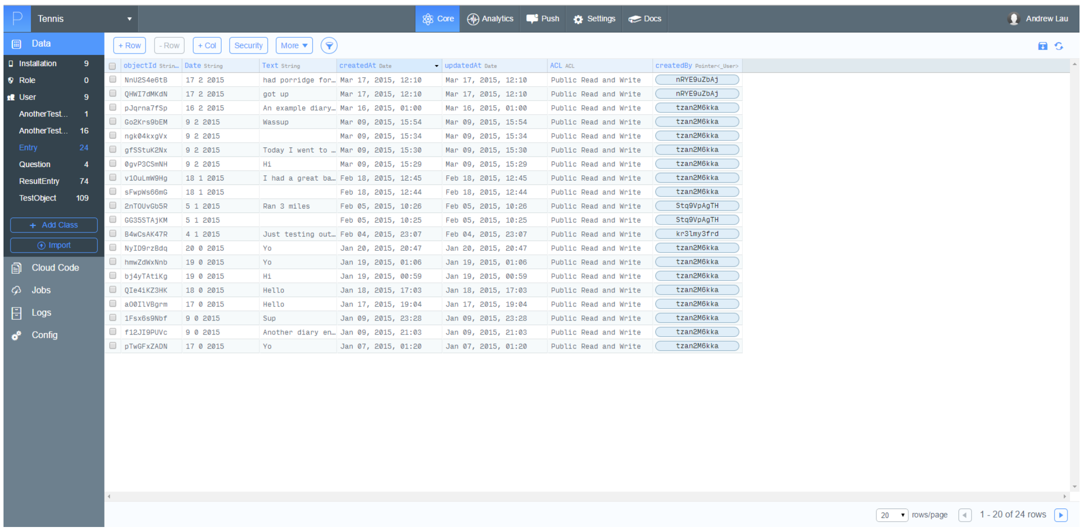

Tennis App 22:15
Andrew L just added a diary entry

Figure 4.1: Add diary entry notification

Parse Console

Parse provides a comprehensive set of tools which allow developers to analyse usage of their application. This analysis is handled from the Parse Console. From the console we can see various statistics such as number of users and number of requests made to Parse from the application.

A component of the Parse console, Parse Core, allows the developer to view and manage all of the data that is being stored in Parse. From here the data can be directly manipulated by editing the fields in the table shown. A screenshot of Parse Core is shown in figure 4.2.



objectId String	Date String	Text String	createdAt Date	updatedAt Date	ACL ACL	createdBy Pointer<_User>
NnU2S4e6tB	17 2 2015	had porridge for…	Mar 17, 2015, 12:10	Mar 17, 2015, 12:10	Public Read and Write	nRYE9uZbAj
QHWI7dMKdN	17 2 2015	got up	Mar 17, 2015, 12:10	Mar 17, 2015, 12:10	Public Read and Write	nRYE9uZbAj
pJqrna7fSp	16 2 2015	An example diary…	Mar 16, 2015, 01:00	Mar 16, 2015, 01:00	Public Read and Write	tzan2M6kka
Go2Krs9bEM	9 2 2015	Wassup	Mar 09, 2015, 15:54	Mar 09, 2015, 15:54	Public Read and Write	tzan2M6kka
ngk04kxgVx	9 2 2015		Mar 09, 2015, 15:34	Mar 09, 2015, 15:34	Public Read and Write	tzan2M6kka
gfSStuK2Nx	9 2 2015	Today I went to …	Mar 09, 2015, 15:30	Mar 09, 2015, 15:30	Public Read and Write	tzan2M6kka
0gvP3CSmNH	9 2 2015	Hi	Mar 09, 2015, 15:29	Mar 09, 2015, 15:29	Public Read and Write	tzan2M6kka
v1OuLmW9Hg	18 1 2015	I had a great ba…	Feb 18, 2015, 12:45	Feb 18, 2015, 12:45	Public Read and Write	tzan2M6kka
sFwpWs66mG	18 1 2015		Feb 18, 2015, 12:44	Feb 18, 2015, 12:44	Public Read and Write	tzan2M6kka
2nTOUvGb5R	5 1 2015	Ran 3 miles	Feb 05, 2015, 10:26	Feb 05, 2015, 10:26	Public Read and Write	Stq9VpAgTH
GG35STAjKM	5 1 2015		Feb 05, 2015, 10:25	Feb 05, 2015, 10:25	Public Read and Write	Stq9VpAgTH
B4wCsAK47R	4 1 2015	Just testing out…	Feb 04, 2015, 23:07	Feb 04, 2015, 23:07	Public Read and Write	kr3lmy3frd
NyID9rzBdq	20 0 2015	Yo	Jan 20, 2015, 20:47	Jan 20, 2015, 20:47	Public Read and Write	tzan2M6kka
hmwZdWxNnb	19 0 2015	Yo	Jan 19, 2015, 01:06	Jan 19, 2015, 01:06	Public Read and Write	tzan2M6kka
bj4yTAtiKg	19 0 2015	Hi	Jan 19, 2015, 00:59	Jan 19, 2015, 00:59	Public Read and Write	tzan2M6kka
QIe4iKZ3HK	18 0 2015	Hello	Jan 18, 2015, 17:03	Jan 18, 2015, 17:03	Public Read and Write	tzan2M6kka
aO0IlVBgrm	17 0 2015	Hello	Jan 17, 2015, 19:04	Jan 17, 2015, 19:04	Public Read and Write	tzan2M6kka
1Fsx6s9Nbf	9 0 2015	Sup	Jan 09, 2015, 23:28	Jan 09, 2015, 23:28	Public Read and Write	tzan2M6kka
f12JI9PUVc	9 0 2015	Another diary en…	Jan 09, 2015, 21:03	Jan 09, 2015, 21:03	Public Read and Write	tzan2M6kka
pTwGFxZADN	17 0 2015	Yo	Jan 07, 2015, 01:20	Jan 07, 2015, 01:20	Public Read and Write	tzan2M6kka

Figure 4.2: Parse Core Console

Push notifications can also be triggered from the Parse Console. While currently all notifications are triggered by the user from within the application, the console allows notifications to be sent at any time to any user based on a set of rules. Any users meeting the criteria defined by the rules receive the notification.

Parse allows developers to directly edit data which has been previously stored, as well as add data to the existing entities. This is done from the Parse Core Console. As shown in figure 4.2, there are several options available such as adding new rows and columns. This view also allows new classes to be added, reducing the need for creation code which is redundant following the first run of the application.

4.3 Layouts

When developing Android applications, the user interface is defined in XML layout files [12]. These files determine the structure and position of various UI elements on screen. An example of a layout implementation is shown below for the navigation drawer layout. Some elements in layouts are predefined, however elements can also be dynamically added to layouts after they have been inflated. Inflation of a layout is the process of instantiating its layout XML file into its corresponding view objects [12].

There are some notable implementation details which can be defined in this layout, most importantly the root element allows precise positioning of the several UI elements using layout "weights". Weights allow UI elements to be sized relative to the other elements in the frame. Attributes of these layout elements are defined using the namespace. The namespace is defined by:

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

This allows the Android namespace to be accessed so that default attributes of layouts can be set. The Android namespace references from the `android.R` resource class which consists of the default built in Android resources.

4.4 Activities

An activity typically represents a screen of content in the application. In *Tennis Tracker*, there is one main activity which is started when the application is launched – `MainActivity`. This activity is responsible for defining the logic for the navigation drawer in the application and allows access to all pages in the application. Another activity, `AddResultActivity` handles the adding of results in the player version of the application. Components of these activities are discussed in detail below.

4.4.1 MainActivity.java and AddResultActivity.java

The implementation of the main activity is in `MainActivity.java`. This activity handles the main structure of the application and is also the parent activity of all fragments in the navigation drawer. This means that if the fragments in the navigation drawer choose to use callback methods, then implementation will be handled by this activity. The activity is also responsible for the passing of data between these child fragments. When activities are started, they go through the activity lifecycle. The lifecycle represents the stages that activities go through when being run as part of an application.

The first stage in the lifecycle is the `onCreate` method which handles the creation of the Activity for the first time. The following line initialises the layout of the `MainActivity`. Instantiation of `AddResultActivity` is similar.

```
    setContentView(R.layout.activity_main);
```

Instantiation code for Parse and the action bar is also handled in the `onCreate` method. As this is the first activity which launches, the navigation drawer is instantiated as part of the `onCreate` method. The logic for switching between pages is also handled in this activity and is discussed in the following section.

Activities also have two methods – `onPause()` and `onResume()` – for handling the pausing and resuming of the activity, i.e. when the user exits from the application and returns to it at a later time.

4.5 Navigation Drawer

4.5.1 Navigation drawer layouts

Implementation began by creating a navigation drawer based project. This involved creating an initial activity for the navigation drawer. The layout XML file for this shown below.

Navigation drawers are implemented using the `ListView` widget. The parameters for this list view are shown in the layout file. `layout_width` has been set to 320dp to comply with Google's recommendation for navigation drawer maximum widths and to stay consistent with other native Android applications which use a navigation drawer.

```
<android.support.v4.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/drawer_layout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
   <!-- The navigation drawer -->
   <ListView
      android:id="@+id/left_drawer"
      android:layout_width="240dp"
      android:layout_height="match_parent"
      android:layout_gravity="start"
      android:choiceMode="singleChoice"
      android:divider="@android:color/transparent"
      android:dividerHeight="10dp"
      android:background="#FFFFFF"/>
</android.support.v4.widget.DrawerLayout>
```

Within the `ListView`, there are `ListView` items. These items require a separate layout in order to customise the colour, size, and other parameters. For the navigation drawer `ListView` item, a custom layout has been created to improve upon the original `ListView` item definition. This is shown below.

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/text1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
```

```
    android:textAppearance="?android:attr/textAppearanceListItemSmall"
    android:gravity="center_vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:textColor="@color/selector_white_black"
    android:background="?android:attr/activatedBackgroundIndicator"
    android:minHeight="?android:attr/listPreferredItemHeightSmall"/>
```

Another notable parameter is `android:id="@+id/drawer_layout"`. This allows the drawer layout to be referred to programmatically in the corresponding activity Java class. `android:id="@+id/content_frame"` refers to the content which changes when a new page in the navigation drawer is selected.

4.5.2 Switching between pages in Navigation drawer

The `selectItem` method is used to switch between fragments in the navigation drawer. This uses a nested if statement to start fragments depending on the position selected in the menu.

```
private void selectItem(int position) {
      if (userType.equalsIgnoreCase("player")) {
         Fragment fragment = null;
         if (position == 0){
            diary = false;
            result = false;
            fragment = new ProfileFragment();
            FragmentManager fragmentManager = getSupportFragmentManager();
            fragmentManager.beginTransaction().replace(R.id.content_frame,
                fragment).commit();
         }
```

As there are two versions of the application, a user type check needs to take place before populating the navigation drawer with items. This ensures that the correct menu items are displayed depending on the type of user logged in. When the user signs up for an account, the user type is recorded depending on which account type they select. This type is added to the `ParseUser` object as a field.

```
      if (checkedCoach) {
         user.put("Type", "Coach");
      }
      else if (checkedPlayer) {
         user.put("Type", "Player");
      }
```

This is then retrieved and stored on application start:

```
   userType = ParseUser.getCurrentUser().getString("Type");
```

4.6 Fragments

Fragments are portions of the user interface in Activities and are used extensively in *Tennis Tracker*. The most prominent use of fragments is within the navigation drawer with each fragment representing a page in the drawer.

Implementation of fragments is comparable to activities as they follow a very similar lifecycle. Each fragment layout is accompanied by a corresponding class where logic is defined.

4.6.1 Dialogue Fragments

A subclass of fragment, dialogue fragments are pop-up dialogues which ‘float’ on top of the active activity. Dialogues are used throughout *Tennis Tracker* to prompt input from the user. One example of this is when prompting the user to enter a diary entry. As the input for a diary entry is relatively simple, creating another activity is not necessary and input can be prompted through a lightweight dialogue. This is in contrast to the result entry process which takes the user through a multi step process.

4.6.2 Callback methods

In order to enable fragment to fragment communication or fragment to activity communication, callback methods need to be utilised. Callback methods allow fragments to call methods implemented in its parent activity. This can be useful for passing data between a fragment and its parent activity. To facilitate this interaction, an interface must be defined in the fragment class along with the definition of the desired method.

```
public interface DiaryInputListener{
   public void onDiaryInput(String diaryText, String diaryDate);
}
```

The activity used for the callback must then be specified in the callback method.

```
@Override
public void onAttach(Activity activity){
   super.onAttach(activity);

   try{
      mCallback = (DiaryInputListener) activity;
   } catch (ClassCastException e){
      throw new ClassCastException(activity.toString() + "must implement
          DiaryInputListener");
   }
}
```

Finally, this method is used to pass data between the fragment and the parent activity.

```
mCallback.onDiaryInput(editTextStr, concatDate);
```

As shown, the callback method `onDiaryInput` is called using the input listener `mCallback`. The `onDiaryInput` method is implemented in the parent activity class hence by passing `editTextStr` and `concatDate` as parameters, these are passed to the parent activity to handle.

4.6.3 New Instance

By default, every fragment uses a no-argument constructor for instantiation. In the case that we need to pass additional information to the fragment, we cannot use this no argument constructor. Instead, a static factory method may be used to set the desired arguments for retrieval in the fragment.

In the `PlayerInfoFragment`, we need to pass the player name so that we can retrieve the diary and result entries specific to that player. To enable this, we define a `newInstance` method for the `playerInfoFragment` as follows:

```
public static PlayerInfoFragment newInstance(String playerName) {
   PlayerInfoFragment fragment = new PlayerInfoFragment();
   Bundle args = new Bundle();
   args.putString(PLAYER_NAME, playerName);
   fragment.setArguments(args);
   return fragment;
}
```

As we are setting the `PLAYER_NAME` attribute of `args` when instantiating, this can then be retrieved further in the fragment lifecycle. To retrieve the `PLAYER_NAME` attribute, we retrieve the arguments first and then call `getString` to retrieve the variable.

```
mPlayerName = getArguments().getString(PLAYER_NAME);
```

In the case that no arguments need to be passed, fragment instantiation defaults to use the no-argument constructor.

4.6.4 The ViewPager Widget

The `ViewPager` widget is used in several points of the application to allow multiple screens of content to be displayed to the user with a swipe transition between screens. The widget emulates a transition similar to swiping across home screens on Android and allows the user to be guided through a series of steps. The following section discusses how use of a `ViewPager` is implemented in `PlayerInfoFragment`. This fragment displays diary and result entries of players to coaches in separate pages. The implementation of the other `ViewPager` instances in the application are similar in structure. As `ViewPagers` instantiate a fragment depending on what page the user is on, we can make use of our previously created fragments. In particular the implementation of `DiaryFragment` and `ResultsFragment` can be reused from the player side of the application. The `ViewPager` is instantiated by using a custom adapter class. This class defines which fragments to instantiate on which page and also defines the number of pages to create. The relevant methods are shown below.

```
      @Override
      public int getCount() {
         return NUM_ITEMS;
      }

      @Override
      public Fragment getItem(int pos) {
         System.out.println("Get item is being called");
         switch(pos) {
            case 0: return DiaryFragment.newInstance(playerName);
            case 1: return ResultsFragment.newInstance(playerName);
            case 2: return PlayerInfoQuestions.newInstance(playerName);
            default: return DiaryFragment.newInstance(playerName);
         }
      }
```

Once the adapter has been defined, the `ViewPager` descibed in the layout file can then be instantiated by assigning the adapter to the `ViewPager`:

```
mPager = (ViewPager) view.findViewById(R.id.playerPager);
String thePlayerName = getArguments().getString(PLAYER_NAME);
mPagerAdapter = new MyAdapter(this.getChildFragmentManager(),
    thePlayerName);
mPager.setAdapter(mPagerAdapter);
```

4.7 Transitions and animations

Tennis Tracker makes use of custom animations to provide an improved user experience. As described in the design section, animations are important for guiding the user through separate sequences in the application and helps inform users of where they are in the application. In order to provide one of these animations, custom animators had to be defined. By controlling the alpha level of the fragment on screen, a fading out effect was achieved.

```
<alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.1"
    android:duration="300"
 />
```

Similarly, a fade in effect was achieved by reversing the alpha values.

```
<alpha
    android:fromAlpha="0.1"
    android:toAlpha="1.0"
    android:duration="400"
/>
```

4.8 Up navigation

In Android, screens which are not the main entrance to the application or the main activity should provide a way of navigating to the logical parent screen [25]. As the application uses multiple activities, this functionality had to be incorporated. When the user presses the add button to add a new result entry, another activity is started – `AddResultsActivity`. From this activity, the user will see a back button in the action bar. In order to allow this back button to return to the previous activity, the functionality of the button has to be defined in the following method:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch (item.getItemId()) {
        case android.R.id.home:
            finish();
            return true;
    }
    return super.onOptionsItemSelected(item);
    }
```

As shown above, the back button is handled in the `R.id.home` case. Calling `finish()` on the activity effectively dismisses the activity, and as the previous activity is still on the backstack, we are returned to that activity.

4.9 Back button functionality

While adding a result, the default behaviour of the system-wide back button is to return to the parent activity. This functionality is desired when the activity is first opened, however as `AddResults` uses a `ViewPager` when on a page other than the first page, returning to the previous page is expected. To allow this functionality, the back button functionality for `AddResultsActivity` has to be overridden.

```
    @Override
    public void onBackPressed() {
       if (mPager.getCurrentItem() == 0) {
          // If the user is currently looking at the first step, allow the system
             to handle the
          // Back button. This calls finish() on this activity and pops the back
             stack.
          super.onBackPressed();
       } else {
          // Otherwise, select the previous step.
          mPager.setCurrentItem(mPager.getCurrentItem() - 1);
       }
    }
```

As shown above, in order to return to the previous page in the ViewPager, the `setCurrentItem` method of the `ViewPager` needs to be called. As we are returning to the previous page, we set the page to `currentItem - 1`. This is not the only instance in which we need to override back button behaviour. In the players page of the coach version of the application, there is also a need to override the functionality of the back button.

4.10 Action Bar

The bar which appears at the top of every window in the application is the action bar. The action bar can be used for a variety of functions. The main uses in *Tennis Tracker* are discussed in this section.

4.10.1 Searching from the action bar

To enable search from the action bar, the `SearchView` widget was utilised. This was defined as part of the menu in `PlayerSearchActivity`:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.andrew.tennisapp.PlayerSearchActivity">
    <item
       android:id="@+id/actionBarSearch"
```

```
      android:title="Search"
      app:actionViewClass="android.support.v7.widget.SearchView"
      app:showAsAction="always"/>
</menu>
```

Once this has been defined, the menu is inflated and relevant attributes are set:

```
      getMenuInflater().inflate(R.menu.menu_player_search, menu);
      MenuItem searchViewItem = menu.findItem(R.id.actionBarSearch);
      SearchView searchView = (SearchView) searchViewItem.getActionView();
      searchView.setIconifiedByDefault(false);
      searchView.setQueryHint("Search for players...");
      SearchManager searchManager = (SearchManager)
         getSystemService(Context.SEARCH_SERVICE);
      final SearchView searchView2 = (SearchView)
         menu.findItem(R.id.actionBarSearch).getActionView();
      searchView2.setSearchableInfo(searchManager.getSearchableInfo(getComponentName()));
      searchView2.setIconifiedByDefault(false);
```

By default, the search menu item appears as an icon and is not expanded. To force this view to always expand, the `setIconifiedByDefault` attribute was set to `false`. When this activity is started, expanding the search view now requires no user interaction.

4.10.2 Action bar tabs

The `ViewPager` in the `PlayerInfoFragment` uses tabs which are displayed below the action bar. These tabs are independent of the `ViewPager` widget and are actually an extension of the action bar. In order to link the tabs and the pages in the `ViewPager`, we must use click listeners on the tabs and adjust the page that is displaying depending on which tab has been clicked.

To enable tabs, `actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS)` is called. This overrides the parent activity's navigation mode to display tabs on all child fragments. Consequently, as this affects all fragments, tabs need to be disabled for fragments which do not need them.

A tab listener is then used to switch between the pages of the `ViewPager`:

```
      ActionBar.TabListener tabListener = new ActionBar.TabListener() {
         public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
            mPager.setCurrentItem(tab.getPosition());
         }
      }
```

4.11 Second Implementation Cycle

Following on from evaluation with the client, a number of additional features were identified and prioritised. It was decided that a second implementation cycle was necessary to fulfil these additional requirements. During this implementation phase, user interface enhancements were also implemented following feedback from evaluation. The following additional features were implemented.

4.11.1 Asking questions to specific players

In addition to asking questions to all players, the ability to specifically select players to ask questions to was added. This required many changes to the backend structure as well as the UI. In order for questions to be displayed to the correct players, additional information had to be added to the question object when originally storing this object in the Parse cloud. In particular an attribute to identify the coach who asked the question and the player who the question is intended for had to be added:

```
    if (player.equalsIgnoreCase("all players")) {
        //put the coach pointer to 'askedBy field'
        question.put("askedBy", ParseUser.getCurrentUser());
        addQuestion(questionText);
     }
     else {
        //put the player name to the 'playerAsked' field on Parse
        question.put("playerAsked", player);
     }
```

When asking the question, the coach has the option of selecting "all players". In this case, a pointer to the coach's `ParseUser` object is added to the question to allow all the players which a coach coaches to be asked this question. If a specific player has been selected, then this player's name is assigned to the `playerAsked` attribute of the question object. When retrieving these questions, the player's name is looked for in the `playerAsked` field of the question objects stored in Parse and only the questions with the current player's name in the `playerAsked` field will be retreived and subsequently displayed for the player to answer.

4.11.2 Viewing profile pages of other players

To add the ability to view other player's profile pages, an additional activity had to be added. `PlayerSearchActivity` defines the logic for searching and retrieving players, as well as showing profile pages when a name is selected.

To search for players, a query is made to Parse:

```
  ParseQuery<ParseUser> userQuery = ParseUser.getQuery();
      userQuery.fromPin("theUsers");
      userQuery.whereEqualTo("Type", "Player");
      userQuery.findInBackground(new FindCallback<ParseUser>() {
         public void done(List<ParseUser> users, ParseException e) {
            if (e == null) {
               for (ParseUser user: users) {
                  playerNames.add(user.getString("name"));
               }
            } else {
                Log.d("User", "Error: " + e.getMessage());
            }
         }
      });
```

The above query returns `ParseUsers` of type "Player" from the Parse datastore. `userQuery.fromPin("theUsers")` specifies that the query should use the local datastore instead of Parse datastore in the cloud and specifically retrieve results from the subset of results returned by the query pinned as "theUsers". To enable users to search, the results of this query are applied as an adapter to a `ListView`. This `ListView` then dynamically updates as the user types using an `OnQueryTextListener`:

```
        SearchView.OnQueryTextListener textChangeListener = new
            SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextChange(String newText) {
                if (searchView2.getQuery().toString().isEmpty()) {
                    listNames.clear();
                    myAdapter.notifyDataSetChanged();
                }
                else {
                    // this is the adapter that will be filtered
                    listNames.clear();
                    for (int i = 0; i < playerNames.size(); i++) {
                        if
                            (playerNames.get(i).toString().toUpperCase().contains(searchView2.get
                            {
                            listNames.add(playerNames.get(i).toString());
                        }
                        myAdapter.notifyDataSetChanged();
                    }
                }
                return true;
            }
```

To update the list view dynamically, the list of names being displayed needs to be updated constantly as the user types. This is done by filtering an adapter by searching for the substring of characters entered by the user over the list of player names returned by the query. Only the names with matching substrings are retained. To reflect this change, `notifyDataSetChanged()` is called and the `ListView` is updated.

When the user selects a player, the view needs to be updated to show the profile page of that player. As there is already a fragment class defined for displaying profile pages, we simply need to create a `newInstance` method for the profile fragment class, taking the player's name we wish to view as the parameter. Passing the player name as a parameter allows the profile page of the player to be viewed.

```
        fragment = ProfileFragment.newInstance(playerName);
        FragmentManager fragmentManager = getSupportFragmentManager();
        fragmentManager.beginTransaction().setCustomAnimations(R.animator.fade_in,
            R.animator.fade_out).replace(R.id.content_frame, fragment,
            TAG_FRAGMENT).addToBackStack("TAG_FRAGMENT").commit();
```

Once we have created the `newInstance` of the `ProfileFragment`, we must use the `fragmentManager` to switch to this fragment. As shown above, a custom animator is being used to simulate a fading animation when the user selects a player and switches fragment. The fragment is tagged to enable the back button to be overidden and take the user back to the desired page in the hierarchy.

4.11.3 Tiebreak score entry

To expand on the score entry for results, the ability to add tiebreak scores was added. A number of changes were necessary to implement this. To begin, the layout structure of the score cards was altered to allow tiebreak scores to be assigned. Text fields were added in a subscript style to allow these scores to be displayed alongside the main set score.

```
            <TextView
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:textSize="9sp"
               android:paddingLeft="2dp"
               android:layout_marginTop="2dp"
               android:id="@+id/player1set1tie" />
```

The positioning of the tiebreak score numbers was stylised with the addition of `paddingLeft` and `marginTop` with font size also being reduced.

In order to allow the user to enter tiebreak scores, the application automatically determines when to prompt the user for a tiebreak score. In the case that either player has won the set by 7-6, this indicates that a set was played with a tiebreak. At this point, the application inserts an additional score input line to prompt for the tiebreak score. The default layout for the set score entry page defines a single set score input row and therefore every time the page is loaded, the page prompts the user for a score for a single set. As there is the option for the user to add additional sets, a set score row also had to be dynamically defined when users press the 'Add set' button. When the user presses this button, a button listener triggers the dynamic addition of a set score row to the layout. As there are two separate types of set score row – predefined and dynamic – separate logic for dynamically adding tiebreak score rows had to be defined.

As dynamically added spinners have no `id`, they must be tagged so that the desired score can be retrieved:

```
setSpinner.setTag("P1S" + setCount);
```

By keeping a counter for the number of sets, the tag can be dynamically allocated as more sets are added.

Each spinner is assigned a listener to keep track of when the values are updated. Using this in conjunction with the tag, the score selected in the spinner can be linked to a variable which keeps track of its value:

```
         @Override
         public void onItemSelected(AdapterView<?> parent, View arg1, int arg2,
             long arg3) {
            if (parent.getTag().toString().equalsIgnoreCase("P1S2")) {
               P1S2 = Integer.parseInt(parent.getSelectedItem().toString());
            }
         }
```

When a value of a spinner is updated, the scores selected in the other spinners are checked to determine if it meets the criteria for inputting a tiebreak score:

```
            if ((P1S2 == 7 && P2S2 == 6) || (P1S2 == 6 && P2S2 == 7) ) {
               if (setsTie.get("set2") == null) {
                  setsTie.put("set2", "");
                  addTiebreakRow(parent.getTag().toString());
               }
            }
```

The logic for determining if the scores entered meet the tiebreak criteria is similar for the predefined spinners, however this resides in the class in which the layout is originally inflated.

Chapter 5

Evaluation

5.1 Evaluation with client

A user evaluation was conducted with the client once all basic requirements had been implemented. This allowed for feedback to be gathered on the implemented requirements to determine if any changes were needed to meet the originally defined requirements. This also allowed the client to suggest and prioritise features to be implemented in the next implementation phase. In conjunction with requirements validation, feedback on usability was also gathered while the client used the application.

5.1.1 Evaluation Procedure

To evaluate the application, the client was asked to perform several tasks. These tasks were selected to provide extensive coverage of the features implemented so far. The think-aloud evaluation protocol was used and hence the client was asked to make comments while using the application. Observations were also made while the client used the application to determine any usability issues and problems with the user interface.

The tasks performed by the client were as follows:

- Create a user account for both a Player and Coach
- On the player account, add a diary entry
- On the player account, add a result entry
- On the coach account, add the player that was just created
- On the coach account add questions for the players to answer
- On the coach account, view the diary and result entries for one of the players in the players list

5.1.2 Results

The client was encouraged to think out loud when performing the tasks listed above. This enabled the identification of several usability issues and also cases where the implementation did not match the user's expectation. These issues have been broken down into the different sections of the application below.

Profile page

The client liked the idea of the profile page and suggested that different information was displayed on the profile page. However, concerns were also expressed with who can see this information. A possible solution to this would be to restrict the profile page to only be visible once player and coach have agreed to share information with each other. When the coach selects a player to connect with, the player would receive a notification informing them that a coach wishes to connect. They would then have the option of accepting – allowing the coach to see their details, or decline.

Results Page

While entering set scores on the results page, the client displayed some difficulties when trying to select a score due to the size of the selectable area of the dropdown boxes. To solve this, the size of the dropdown boxes could be increased. It was also pointed out that a '0' score has been omitted from the score dropdown selection.

The client experienced some usability issues when answering questions on the question page of the results entry. When answering a question, the back button had to be pressed to dismiss the keyboard to see the rest of the questions as it was obstructing the main view. The application also unexpectedly crashed when the client pressed the on-screen back button. This was caused by the wrong activity parent being defined and was corrected by defining the appropriate activity parent in the manifest XML file.

Diary Page

When adding a diary entry, the client was unsure whether they had input a diary entry and therefore accidentally input a blank diary entry. A solution to this would be to add a 'toast' message when the user presses the 'add' button when adding a diary entry. In the second implementation cycle, validation was added to ensure no blank diary entries are input.

Feature suggestions

A number of potential additional features were also suggested during the think-aloud evaluation. These were:

- Asking questions to specific players in addition to asking every player that the coach coaches
- Including a bank of questions that the coach can ask to avoid ethical issues as this is an application aimed at junior tennis players
- Viewing profile pages of other players registered in the system
- Add option for parental consent for junior tennis players
- Coach to coach interaction – possibly in the form of an instant messaging system
- Asking questions before and after score entry

5.2 Evaluation with Supervisor

In addition to evaluation with the client, a final evaluation was also performed with the supervisor of the project. This allowed key functionality to be tested and feedback for future developments to be gathered. As the supervisor is familiar with tennis terminology, they were an ideal candidate for testing and verifying technical features such as set score entry. This also allowed for requirements validation and any changes to the requirements to be suggested.

During the evaluation, the tasks listed below were performed. The tasks were chosen to provide a comprehensive coverage of the application's features. The list of tasks below are split into separate tasks for the coach and player versions of the application.

- Create a new player account
- Create a new coach account

Coach

- Add the player you just created to the players list
- Ask that player a question
- Add a few more questions for all players in general
- View the questions that have been added in the sections on the questions page

Player

- Add a result for a player, answering the questions that the coach has asked
- Add a couple of diary entries to the diary page
- Add a result which has a tiebreak score

Once done, log out of the player

- Log back into the coach, search for a player using the player search function
- View the result and diary entries for the player that has just been added
- View the questions for that player that has just been added and the answers to these questions in the questions tab in the players page

Some miscellaneous tasks were also included to test some typical functions which may be used:

- Send a password reset to the player accounts you have created
- View the notification that has been sent to the coach account following the adding of a diary entry.

5.2.1 Evaluation Procedure

The evaluation consisted of three main parts. The supervisor was asked to perform a series of tasks to enable coverage of the application's main features. While performing these tasks, the think-aloud evaluation protocol was used – the participant was asked to 'think out loud' and notes were taken as the tasks were performed. Observations were also performed on the participant's actions to gauge overall usability. Following on from the evaluation, a short interview was conducted, asking open ended summative questions to understand the participant's overall thoughts on the application. Finally, the participant was asked to complete a NASA TLX form to understand subjectively, their cognitive load while performing the tasks.

5.2.2 Evaluation results

Think aloud and observation

Through a combination of think-aloud and observation the following feedback was gathered:

- When entering the password, the participant was not aware of the password criteria – passwords need to be 6 or more characters
- Attempts were made to interact with the profile page when first logged in, the participant commented that the profile page should be editable
- When asked to log out, it was unclear how to. The participant was unaware of the secondary menu
- On the coach application, after adding a player, asking questions to that player is not clear
- The application unexpectedly crashed when adding a question aimed at all players
- The participant was unsure how to ask questions after searching for a player and commented that there should be options to ask players questions when on that player's profile page
- Tiebreak score entry should prompt for entry when the user enters a score of 7-6 or 6-7. The participant was unsure how to enter tiebreak entries
- When following the task to view profile pages with asking questions, the participant assumed that this could be done from the profile page

5.2.3 Post task interview

In order to provide a summative overview of the participant's thoughts on the application as well as provide a chance to suggest additional features, a post task interview was conducted. This involved asking participants a series of open-ended questions. The questions asked were as follows:

1. Do you see anything that could be improved with the player side of the application? What about the coach?
2. Any features that given more time, you would like to see implemented?
3. In terms of UI and usability, how did you find the application to use?
4. At any point were you unsure of what to do? Or confused about how the application handled some action?

5. Anything you would change based on this?

As these questions were asked to all participants who participated in the evaluation, the results are summarised at the end of this chapter following a description of other evaluation procedures.

5.2.4 Results

A series of actions were generated from the feedback gathered from evaluation. Actions have been suggested for each feedback point and where possible, solutions have been proposed.

- To improve the clarity of the password criteria, a hint could be displayed when the user focuses on the password field. This could similarly be applied to other fields with criteria such as the email address input field.
- In order to clarify the features of the application, on application start a quick page-by-page tutorial could be displayed. This helps to eliminate any ambiguities.
- The profile page could be made more interactive by adding buttons linking to the questions page for that player. This could provide a more centralised way of accessing features related to players.
- Update the set score entry to automatically prompt the user to enter a tiebreak score when they have entered a score of 7-6 or 6-7.

5.3 User Interface Evaluation

In order to assess the user interface of the application, an evaluation was carried out with students at the University of Glasgow. In total, fourteen participants carried out the user interface evaluation. These participants were asked to use the application and provide feedback on the user interface with no previous insight into tennis or the functions of the application necessary. The participants selected were not in the intended user group of junior tennis players due to ethics approval issues, however user interface evaluation does not need to be performed with users in the intended user group in order to be effective.

5.3.1 Evaluation Procedure

Participants were asked if they would like to participate in the evaluation and upon consent, were briefed on the project and the evaluation. An informal interview was carried out while the participants were using the application in order to understand if any aspects of user interface was positive or negative. Participants were also asked a series of questions following the evaluation in order to gather summative feedback. These points were documented in the post-task interview results section. In order to gain an understanding of the cognitive load on participants while using the application, the participants were additionally asked to complete a NASA TLX form following the evaluation. This enabled a quantitative comparison between participants who used an older version of the application and those who used the latest version. By comparing these, the effect of changes in the user interface on cognitive load can be gauged. Seven participants used an older version of the application, while seven used the latest version. The results from this are documented in the following section.

5.3.2 Evaluation results

A summary of the points raised by participants is detailed below:

- Three participants noted that the drop-down boxes for set entry are too small and difficult to select the desired number accurately
- One participant suggested adding images to the menu items to make it more visually appealing
- Many of the participants tried interacting with the profile page and suggested the tiles be editable
- One participant suggested moving the "logout" button to the menu to make it more visible
- Most participants praised the clean design of the application
- One participant experienced a crash when adding a result entry

Following on from the evaluation, a number of additional features were implemented in order to solve the main issues highlighted. These are discussed below.

- In order to make it easier to input set scores, the UI for set score entry was updated to increase the size of the drop-down menus.
- The colours and sizes of the "Add set" and "Next" buttons have also been adjusted to make these functions clearer to the user.
- To increase the user's awareness of the various actions being performed by the application, a number of "toast" messages were added. These are temporary messages which appear at the bottom of the screen to inform the user that an action has taken place.
- Player Info was updated to include performance comments made by the user and also answers to questions.
- Tabs were added to the player info page to make the different options more visible.
- Results entry and logout page crash was fixed by adding an extra check for an instance of the local datastore.

5.3.3 Post task interview results

- Several participants noted that it would beneficial to have more interaction on the profile page. Suggestions were made to include editable statistics and profile picture.
- Two participants were unsure of the purpose of the profile page and suggested that there should be actions on this page.
- When adding players, participants suggested that players should be notified and given the option to confirm that they know the coach who has added them.
- One participant suggested allowing colours of diary entries to be changed. This would allow quicker identification of topics in diary entries through colour codes.
- Participants unfamiliar with Android were not aware of how to open the navigation drawer menu. It was suggested to open the navigation drawer automatically when the application starts to inform users of this.
- A player leaderboard was suggested for the coach application to allow coaches to see which players are performing the best.
- A way for the player to message their coach was also suggested.

5.4 NASA TLX (Task Load Index)

In total, fourteen participants completed a NASA TLX [26] form following the evaluation to measure the level of cognitive load they experienced when using the application. NASA TLX is an assessment tool which allows the rating of several factors when performing tasks such as mental, physical and temporal demand. Participants completed the paper based version of the form, however, a web based version is also available. An image of this form is shown in figure 5.1.

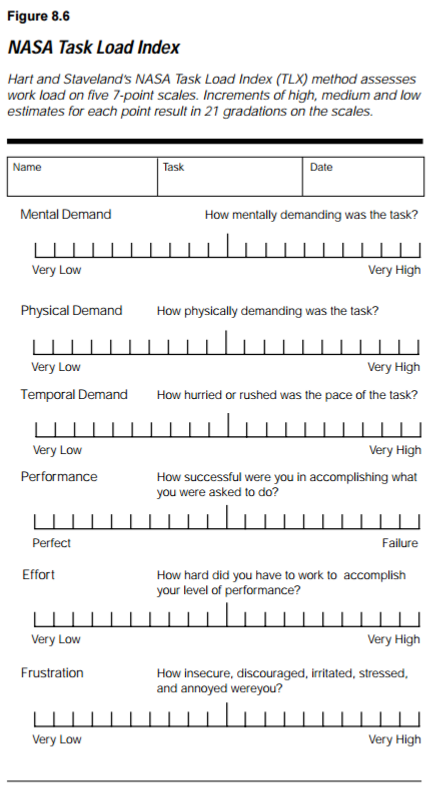


Figure 8.6

NASA Task Load Index

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

Name	Task	Date

Mental Demand — How mentally demanding was the task?
Very Low — Very High

Physical Demand — How physically demanding was the task?
Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?
Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?
Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?
Very Low — Very High

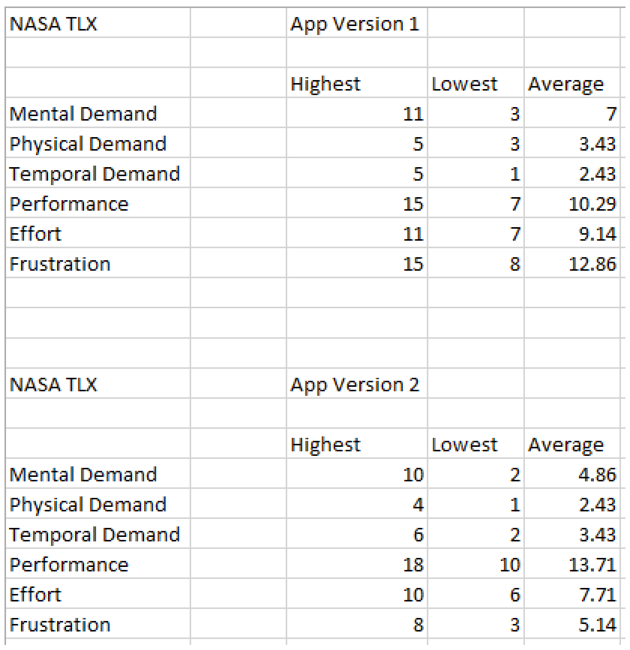
Frustration — How insecure, discouraged, irritated, stressed, and annoyed wereyou?
Very Low — Very High

Figure 5.1: NASA TLX form completed by participants

5.4.1 NASA TLX Results

This section details the results from the NASA TLX evaluation. An average of each of the sections in the TLX are displayed in the table shown in figure 5.2.

The results have been split between participants who used version 1 of the application and version 2 of the application. Version 1 represents an older version of the application without usability enhancements such as toasts and additional validation. Version 1 also omits the use of tabs when viewing diary, results and questions for a player on the coach application. Version 2 is the final version of the application with all usability enhancements in place. From the TLX results, we can immediately see some striking differences between version 1 and version 2. Participants on average performed significantly better with version 2. This can be attributed to the enhancements to version 1 added in version 2. This result is expected as many of the changes in version 2 were added in response to feedback when using version 1 of the application. One other notable difference is the increased frustration of participants when using version 1. This is understandable as many participants in the evaluation pointed out problems they had with the user interface and appeared visibly confused when asked to perform some tasks such as viewing the result entries of a player from the coach application. By adding tabs, this presented a visual indication to users that the player's results were available on this page. Nearly all participants did not



NASA TLX	App Version 1		
	Highest	Lowest	Average
Mental Demand	11	3	7
Physical Demand	5	3	3.43
Temporal Demand	5	1	2.43
Performance	15	7	10.29
Effort	11	7	9.14
Frustration	15	8	12.86
NASA TLX	App Version 2		
	Highest	Lowest	Average
Mental Demand	10	2	4.86
Physical Demand	4	1	2.43
Temporal Demand	6	2	3.43
Performance	18	10	13.71
Effort	10	6	7.71
Frustration	8	3	5.14

Figure 5.2: NASA TLX results

know to swipe the screen to view the results when there were no tabs. It is important to mention that NASA TLX is a subjective evaluation methods and any results gathered could be biased. This could be rectified by increasing the number of participants carrying out the evaluation, however this is resource intensive as more participants would have to be recruited.

5.5 Heuristic Evaluation

It was decided to utilise heuristic evaluation. This allowed the user interface and experience to be thoroughly evaluated. In order to carry out this evaluation, Nielsen's ten heuristics [27] were used as it allowed the evaluation of the application against recognised usability principles.

Visibility of system status

Upon loading the application for the first time, the user is immediately presented with the login screen. The fields displayed are labelled with "hints" to inform the user exactly what the system is expecting. Buttons are clearly labelled indicating which actions they perform. When a user is signing up for an account, the fields displayed also use hints to guide the user. If the user enters incorrect inputs, they are notified through the use of temporary toast messages. Every field in the login and sign up forms have validation, therefore suitable messages will be displayed to the user depending on which input is incorrect. The appropriate changes can then be made to the inputs. Transitions and animations used also keep the user informed. The use of a slide-up animation when the user successfully creates an account and presses the "Create account" button, or successfully signs in, make it clear to the user where in the application they are transitioning to. Also, as this transition is standard in Android applications, these transitions should be familiar to Android users.

Match between system and the real world

As the application targets tennis players, the words, phrases and concepts should be familiar with all users in the target user base. When viewing previously entered results, the cards are designed to follow the layout of a standard tennis scorecard, therefore should be understandable for users. Statistics shown on the profile pages for players also draw on this previous knowledge, showing statistics familiar to tennis players. As this is an Android application, layouts and commonly used widgets should be familiar to all users of Android as the application follows design conventions and guidelines set out by Google. One caveat to this may be due to the relatively recent release of Android 5.0, however the functionality of main system widgets have remained consistent with only visual changes in most cases.

User control and freedom

The use of a navigation drawer allows users to easily navigate from page to page in the application. In the case that the user selects the wrong page, they can easily navigate to the desired page using the navigation drawer. If users accidentally select a function such as “Add a result”, they have multiple ways to exit this dialogue. They can use the system-wide back button to dismiss the screen and return to the previous screen, or they can press the in-application back button to perform the same action. The use of a contextual back button – only displaying when needed - affords the user increased flexibility and freedom in the application.

Consistency and standards

The application uses a consistent colour scheme throughout. Upon starting the application, the user is made familiar with the application-wide colour scheme through colouring of the status bar and buttons. The action bar is displayed at the top of each page to inform the user of the current page and also to promote consistency and structure. The application has also been designed to follow Google’s Material design guidelines where possible. This ensures a consistent look and feel across the application while also drawing on the user’s potential previous experience of using applications designed for Android 5.0.

Error prevention

In order to prevent actions such as accidental logouts when the logout button is pressed, they are presented with a dialogue box which prompts the user to confirm the action. This ensures that the user is in complete control over their actions and has the option to cancel actions which they no longer wish to perform. Other error-prone operations include entering diary entries. In order to mitigate the error rate, the user’s input is validated when they press the ‘add button. If the user has entered no text in the text field, then a toast message is displayed informing the user of this error and suggesting a possible action while also preventing the user from recording a blank diary entry. The system will only allow an entry to be successful if the text box is non-empty.

Recognition rather than recall

All buttons and actions are labelled clearly in the application, reducing the need for the user to remember how they function. For icons in the action bar, the icons chosen are commonly used to represent the action. For example, a magnifying glass is used for the profile search function. Users of Android are also likely to use their familiarity with the operating system to navigate around the application with ease. The navigation drawer used is a very commonly used design pattern and therefore users of Android are likely to have experience with this navigation method. The idea of a slide-out menu is also not confined to Android – iOS applications also commonly use a slide-out menu to allow navigation between the main features of the application. The use of standard widgets should also allow little deviation from what the user is familiar with from the Android platform, therefore requiring little recall.

Flexibility and efficiency of use

The user has the option of using a gesture to open the navigation drawer. This may allow the user to navigate faster as they can open the menu by swiping in from the left edge, a significantly increased area versus tapping a button. Fitts' laws dictates that "the time required to rapidly move to a target area is a function of the ratio between the distance to the target and the width of the target" [28], therefore allowing this swipe gesture should decrease the amount of time needed to navigate around the application.

Aesthetic and minimalist design

The application features a minimalist design. As Google guidelines promote minimalism, by following these, the resulting application features screens which only display information which is necessary to the user. When the user moves to another section in the application, the information displayed is adapted to be appropriate for the screen the user is on. As screen space is a vital resource on mobile devices, this was prioritised. Cluttering the screen with additional unnecessary information potentially diminishes the usability and user experience. Functionality is only useful if the user can use it with little cognitive load. Frustrating the user ultimately results in alternatives being sought. Bold colours are also used to promote an aesthetically pleasing, clean design.

Help users recognize, diagnose, and recover from errors

The application shows error messages in a number of situations – in all of these cases, messages contain no error codes and only use plain English. An example of this is when the user does not enter the same password in both fields when signing up for an account. This message is displayed as a toast and is therefore unobtrusive, and requiring no action from the user. The message fades from the screen after a few seconds, allowing the user to make the necessary corrections.

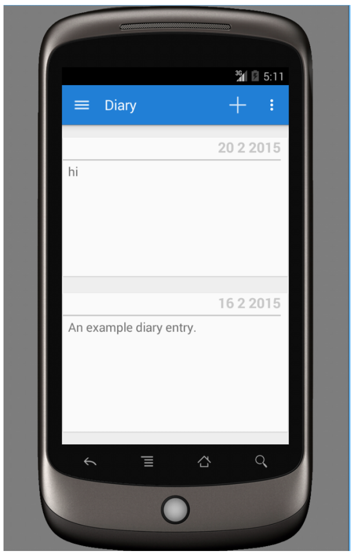
Help and documentation

The application features a dedicated help page detailing how to use the main functions in the application. Even though this help page exists, the application was designed to be easy to use and require minimal documentation in order to use effectively. While this goal has been largely met, a number of issues were identified in evaluation which prompted the need for more detailed documentation and guidance to the user. To reduce the need for this, hints are occasionally provided to the user in the form of toast messages.

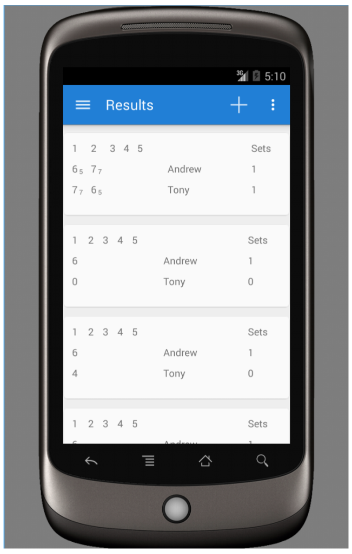
5.6 Compatibility testing

In order to verify the compatibility of the application with different versions of the Android operating system. The application was run using the simulator built into Android studio. The simulator facilitates the discovery of compatibility problems with Android versions without running the application on a physical device. However, as physical devices running the Android 5.0 and Android 4.2.2 operating systems were available, compatibility tests with these versions was conducted by installing the application on the physical device. Different screen sizes and resolutions were also tested to determine the versatility of the defined layouts. Typically layout sizes are defined by "dp" – density independent pixels. One density independent pixel is equivalent to one physical pixel on a 160dpi screen [29]. This enables user interface elements to be displayed correctly on screens with different densities. As Android supports a wide range of different screen resolutions and densities, different assets should be provided for each of the different classes of density ranging from ldpi to xxxhdpi.

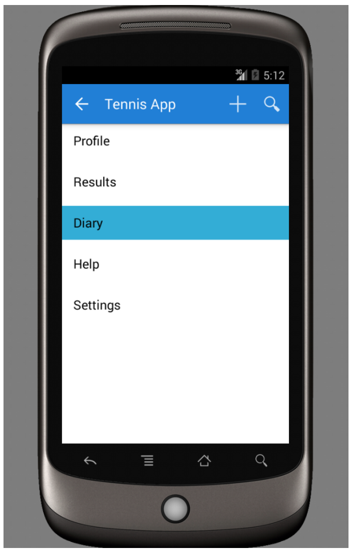
The application was tested with a low density, small screen size device on the emulator to determine how the application's UI scales. To also test the compatibility with previous Android versions, the operating system selected was 4.4.2. The device used to test was a Nexus One with a screen size of 3.7 inches and a density of 264dpi – significantly lower than the screen resolution of the Android 5.0 device tested. Images of the application running on this device are shown below.



(a) Diary page on Nexus One



(b) Questions page on Nexus One



(c) Menu on Nexus One

Figure 5.3: Application running on Nexus One (Android 4.4.2)

As shown in the screenshots above, running the application on smaller screens results in less content being shown vertically. However, the elements scale similarly horizontally when compared to larger devices. A notable difference is the menu now uses all the screen space and does not reveal part of the underlying view. This can be attributed to the considerably lower resolution horizontally. As the device is running an Android version lower than 5.0, there are some differences to the UI. Most notably, the status bar is no longer coloured and UI elements such as `DatePicker` feature different styling. This can be seen in figure 5.4. The application was also run on a physical device running Android 4.2.2, with a screen size of 4.7 inches. This resulted in little discernible difference compared to the Nexus 5, with a screen size of 5 inches.

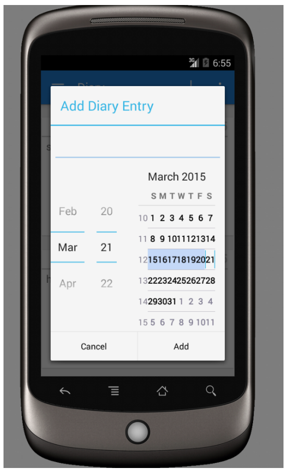
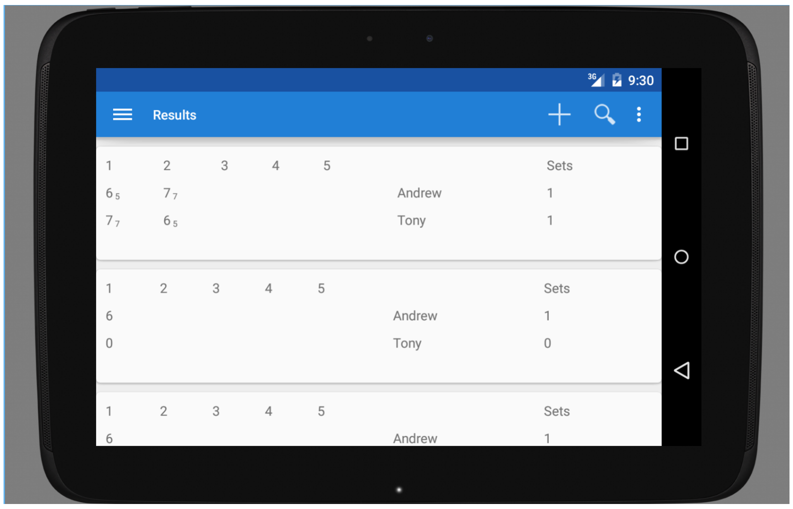


Figure 5.4: Nexus One UI styling

Large screen devices were also tested – namely 7 inch and 10 inch tablets. The application was run on a Nexus 10 on the emulator to understand how the application scales with large screen devices. This device was run in landscape mode as tablets are commonly used in a landscape orientation.



1	2	3	4	5		Sets
6_5	7_7				Andrew	1
7_7	6_5				Tony	1

1	2	3	4	5		Sets
6					Andrew	1
0					Tony	0

1	2	3	4	5		Sets
6					Andrew	1

Figure 5.5: Nexus 10 results

Once again, we see that the interface scales correctly for the larger screen in the landscape orientation. In order to take advantage of the additional screen space, separate layouts could be defined specifically for tablet devices and also for landscape orientation. The application was also run on a physical 7-inch yielding similar results to the 10-inch tablet.

Chapter 6

Conclusion

6.1 Future work

Continued development of the application is an aim. There are a number of requirements which, given more time, would likely have been included in the original implementation and scope.

Expanded application feature set

There are a number of features which could be implemented in future versions of the application. Some of these changes were identified during the evaluation stage but were not implemented for the final release.

- **Communication system for coaches.** Expanding on the idea of communication between player and coach, coaches could be allowed to communicate directly, facilitating discussion on information stored on *Tennis Tracker*. A communication system for coaches would be handled by Parse. This would be similar to the page for adding players, however now coaches would be listed instead of players. Upon selecting a coach, another screen would show the messages between the two coaches. The messaging screen would feature a layout very similar to popular messaging applications such as Facebook Messenger and WhatsApp. Messages being sent would be stored in Parse to enable the other party to retrieve the message.
- **Customisable profile pages.** Many participants in the evaluation attempted to interact with the profile page. Adding an option to edit the profile page by tapping on the various sections could provide an improved user experience as more feedback is provided to the user. Tapping on the sections would ideally open a contextual menu with options to edit, depending on which section is selected.
- **Tennis Social Network.** Combining the features of the communication system and profile pages, a social network system could be built to allow players to compare statistics on matches played and share tennis related posts.

Tablet optimised application

While the functionality of a tablet application would remain largely constant, tablet applications require significant adjustments in terms of layout and efficient screen space usage. A tablet application would take advantage of the additional screen space and increase the scope for more features based on typically increased processing power and battery life.

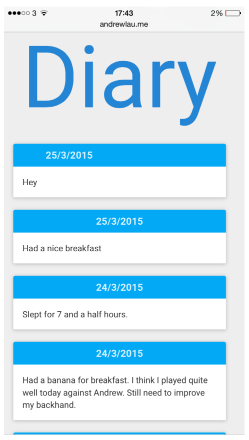
iOS application for iPhone and iPad

The application could be ported to the iOS platform. This would entail an application redesign to adhere to Apple's user interface guidelines. Although the data stored on Parse can still be used across multiple platforms, code would need to be rewritten to interact with the Parse backend. As iOS applications are typically implemented in Objective-C [30], developing an iOS application presents a steep learning curve. The application may also be implemented using the Swift programming language [31] – Apple's new programming language designed to work alongside existing Objective-C code. This programming language offers a number of improvements over Objective-C such as a more concise syntax, and therefore could lessen the learning curve needed to develop an iOS application.

Web application

A web application could also be developed. By taking advantage of the features offered by the the Polymer library [32], Material design look and feel could be brought to the web version of *Tennis Tracker*. This allows a level of consistency to be achieved between the Android and web applications. Developing a web application also has the advantage of being compatible with any mobile device running a supported browser, including iOS devices. While there would be a decrease in performance compared to a native application, web applications can potentially be used anywhere without the need to install, provided a compatible browser is used.

A proof of concept web application which allows users to create accounts and view and add diary entries was developed. The application was developed with Parse to demonstrate the potential for a cross platform application suite. By using Parse as the backend, data can be accessed from multiple platforms and devices. To provide unified user interface design, elements which implement Material Design were also used in the web application. This helps to provide a level of consistency for the user when switching between platforms. Screenshots of the diary page in a proof of concept version of the web application running on the Safari browser on iOS and desktop browser are shown in figures 6.1a and 6.1b. By using the Polymer library, the application also features animations similar to those in the Android application.



Diary

25/3/2015

Hey

25/3/2015

Had a nice breakfast

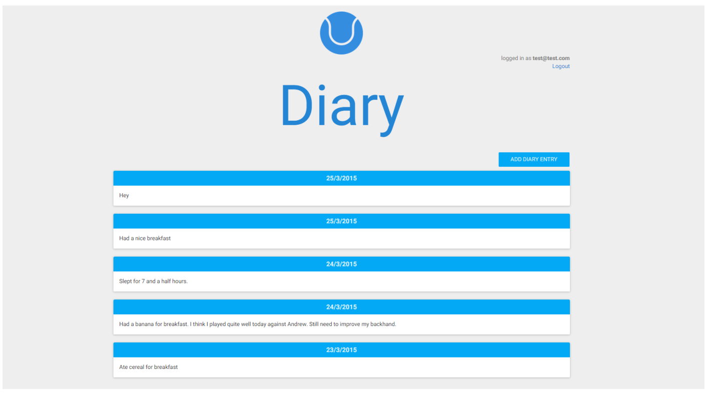
24/3/2015

Slept for 7 and a half hours.

24/3/2015

Had a banana for breakfast. I think I played quite well today against Andrew. Still need to improve my backhand.

(a) Web application running in iOS browser



Diary

25/3/2015

Hey

25/3/2015

Had a nice breakfast

24/3/2015

Slept for 7 and a half hours.

24/3/2015

Had a banana for breakfast. I think I played quite well today against Andrew. Still need to improve my backhand.

23/3/2015

Ate cereal for breakfast

(b) Web application running in desktop browser

Figure 6.1: Cross platform web application

By using responsive design elements, the web application is able to scale to a wide range of screen sizes. This improves the usability and scalability of the application as responsive design is resolution and density independent. Logic which interacts with the Parse database is implemented using JavaScript as the Parse SDK provides native support for JavaScript. The application could be extended to include the full feature set of the Android application in the future. As the backend structure of the application is already defined in the Android application, further development of the web application would mainly consist of JavaScript requests to retrieve and store data in Parse along with a combination of user interface design and front-end logic.

Cross platform interactivity between Android, iOS and Web applications

To increase the number of potential users of the application, cross platform support could be added. As all of these applications use Parse as their cloud backend component, all data stored in the cloud is accessible from all versions of the application. This enables users to use different versions of the application interchangeably. In order to achieve consistency across all applications, user interface design should follow Google's Material Design guidelines. Concurrent use of the application is potentially limited by the number of requests supported in the free tier of Parse – currently 30 requests/sec [11]. However, exceeding this request rate requires a fairly large userbase of active users. Separate Parse applications could be created to decrease the number of requests made, however cross platform interactivity would not be possible with this approach as each application would use a different database.

6.2 Lessons learned

As a minimal level of Android development knowledge was possessed when starting the project, competency in Android has been achieved with a wide range of Android concepts being covered throughout the project, many of these requiring significant time investment to understand fully. These concepts have been covered in detail and I am now comfortable with utilising many advanced features of Android. An increased understanding of the user interface design process for Android applications has also been gained, especially for applications utilising Material design. More experience in the requirements elicitation process has also been gained through a combination interaction with the client and consequent requirements refinements. Overall, this has enabled me to achieve a greater understanding of the whole software development process.

Due to the learning curve, some design decisions made do not reflect best practices in Android development. One example of this is the use of fragments to represent each page in the navigation drawer. The official Android documentation recommends the use of fragments for navigation drawer pages, however, in the application the processing on each page is intensive and thus should ideally be separate activities instead of fragments. Given the option to redefine the navigation drawer, activities would have been preferred over fragments and it promotes separation of concepts and overall cleaner code.

6.3 Final Words

This project has provided a comprehensive coverage of development on Android and also using the Parse backend solution for application development. The aim of the project was to develop an Android application which allows junior tennis players to record and track their progress over time. The *Tennis Tracker* application meets all of the requirements defined in the must have and should have categories, with most of the requirements in the could have category also implemented. Feedback received during evaluation has also mostly been implemented with significant progress in future developments. The Android application is intended to be released on the Google Play Store alongside a fully featured web application.

Bibliography

[1] MyTrainingDiary Testimonials. *http://www.mytrainingdiary.org/testimonials.php*

[2] IDC on Android Market Share. *http://www.idc.com/prodserv/smartphone-os-market-share.jsp*

[3] Technori - the quantified self. *http://technori.com/2013/04/4281-the-beginners-guide-to-quantified-self-plus-a-list-of-the-best-personal-data-tools-out-there/*

[4] Babolat - Play Pure Drive. *http://www.babolat.co.uk/product/tennis/racket/babolat-play-pure-drive-102188*

[5] Ofcom - smartphone usage. *http://consumers.ofcom.org.uk/news/a-nation-addicted-to-smartphones/*

[6] Tennis Math. *http://www.tennis-math.com/*

[7] Tennis Stats. *https://play.google.com/store/apps/details?id=com.kau.jonathan.tennisstats&hl=en*

[8] MoSCoW Prioritisation. *http://www.dsdm.org/content/10-moscow-prioritisation*

[9] Android 5.0. *http://www.android.com/versions/lollipop-5-0/*

[10] Android Marketshare. *https://developer.android.com/about/dashboards/index.html*

[11] Parse. *https://parse.com/*

[12] Layouts in Android. *http://developer.android.com/guide/topics/ui/declaring-layout.html*

[13] Local Datastore. *http://blog.parse.com/2014/04/30/take-your-app-offline-with-parse-local-datastore/*

[14] Parse Android Documentation. *https://www.parse.com/docs/android_guide*

[15] Android Navigation Drawer. *https://developer.android.com/design/patterns/navigation-drawer.html*

[16] Material Design Guidelines. *http://www.google.co.uk/design/spec/material-design/introduction.html*

[17] Material Design Colours. *http://www.google.co.uk/design/spec/style/color.html#color-ui-color-application*

[18] Material Design Animations. *http://www.google.com/design/spec/animation/authentic-motion.html*

[19] Android CardView. *https://developer.android.com/training/material/lists-cards.html*

[20] Android Support Library. *http://developer.android.com/tools/support-library/index.html*

[21] Android Design Principles. *http://developer.android.com/design/get-started/principles.html*

[22] Android Studio. *http://developer.android.com/tools/studio/index.html*

[23] Backendless. *https://backendless.com/*

[24] Appcelerator. *http://www.appcelerator.com/platform/apis/*

[25] Providing Up Navigation. *http://developer.android.com/training/implementing-navigation/ancestral.html*

[26] NASA TLX. *http://humansystems.arc.nasa.gov/groups/tlx/*

[27] Nielsen's Heuristics. *http://www.nngroup.com/articles/ten-usability-heuristics/*

[28] Fitts' law. *https://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/fitts.html*

[29] Android Documentation - Supporting multiple screens. *http://developer.android.com/guide/practices/screens_support.h*

[30] Objective-C. *https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC*

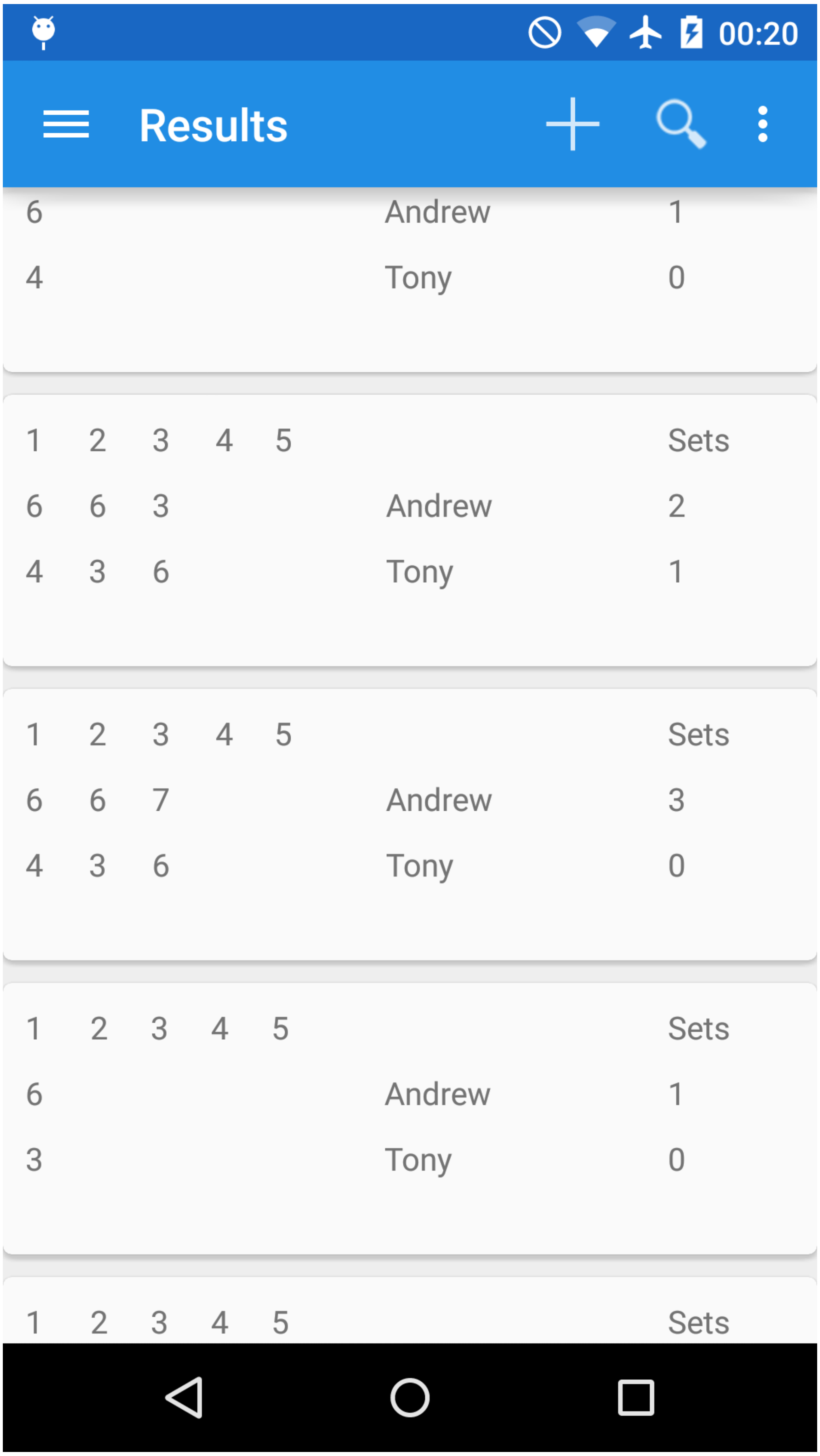
[31] Swift Programming Language. *https://developer.apple.com/swift/*

[32] Polymer. *https://www.polymer-project.org/0.5/*

Appendices

Appendix A

Screenshots



6					Andrew	1
4					Tony	0

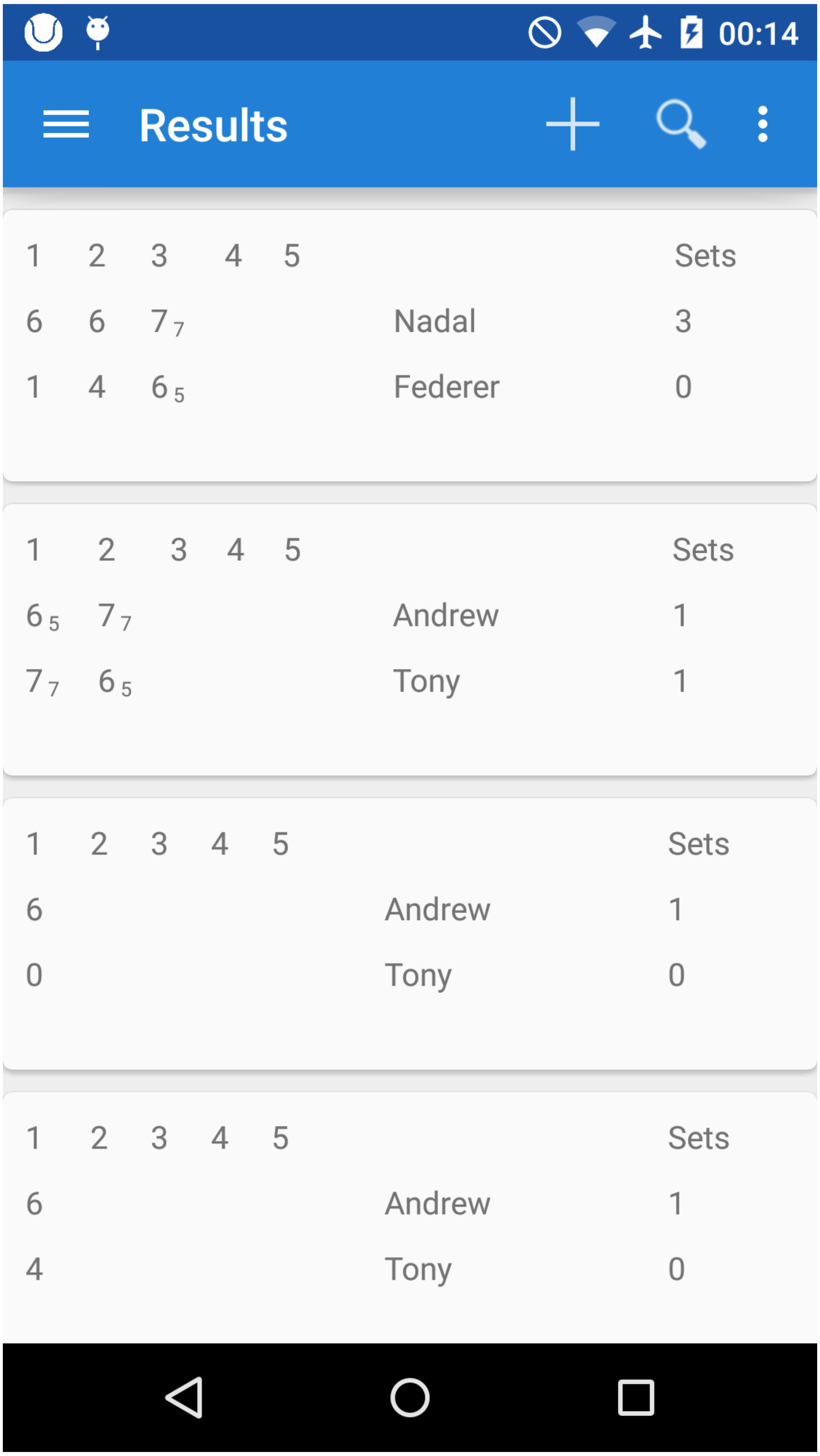
1	2	3	4	5		Sets
6	6	3			Andrew	2
4	3	6			Tony	1

1	2	3	4	5		Sets
6	6	7			Andrew	3
4	3	6			Tony	0

1	2	3	4	5		Sets
6					Andrew	1
3					Tony	0

1	2	3	4	5		Sets

Figure A.1: Results without tiebreak



1	2	3	4	5		Sets
6	6	7_{7}			Nadal	3
1	4	6_{5}			Federer	0

1	2	3	4	5		Sets
6_{5}	7_{7}				Andrew	1
7_{7}	6_{5}				Tony	1

1	2	3	4	5		Sets
6					Andrew	1
0					Tony	0

1	2	3	4	5		Sets
6					Andrew	1
4					Tony	0

Figure A.2: Results with tiebreak

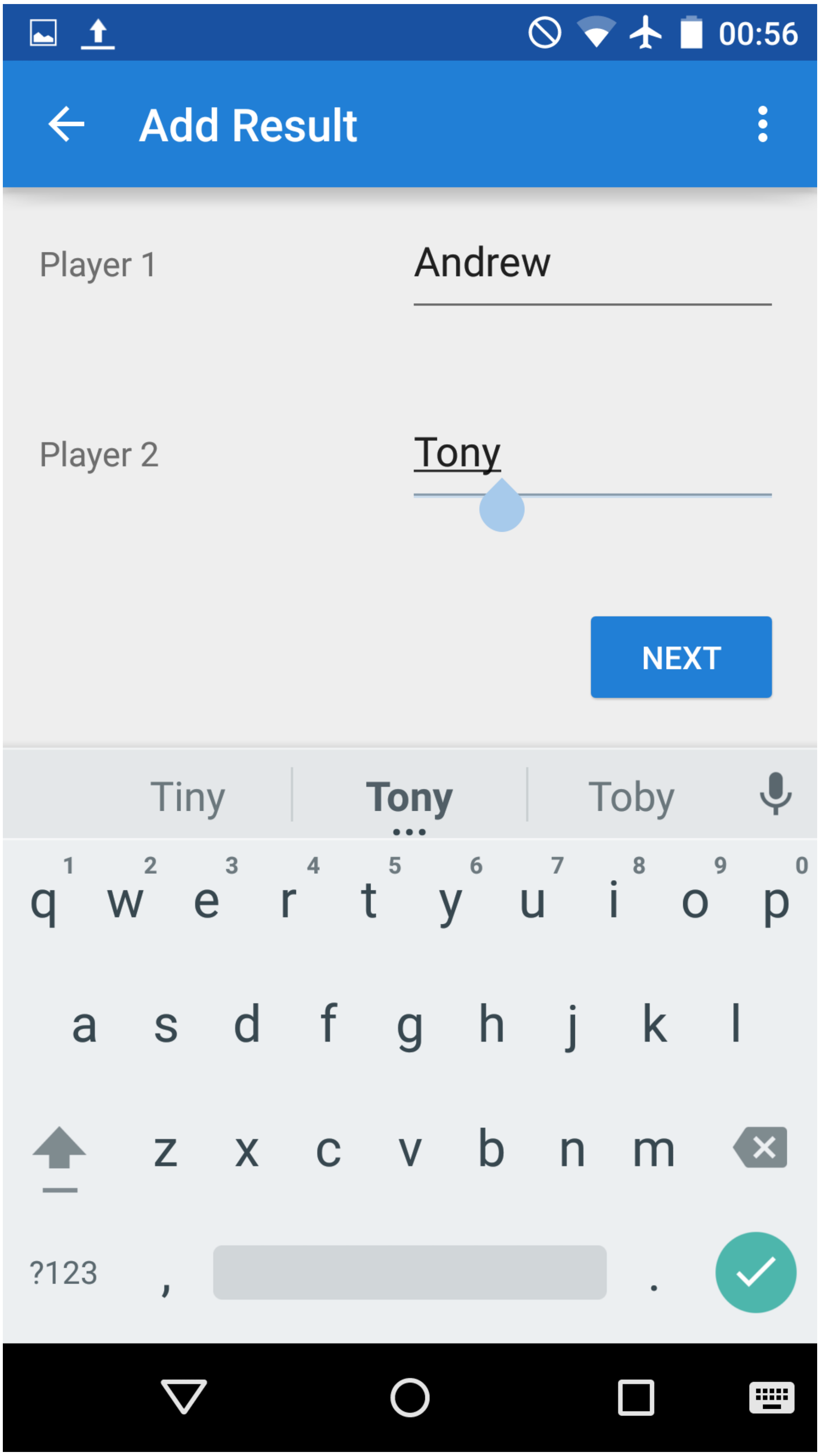


Figure A.3: Add results – Player names

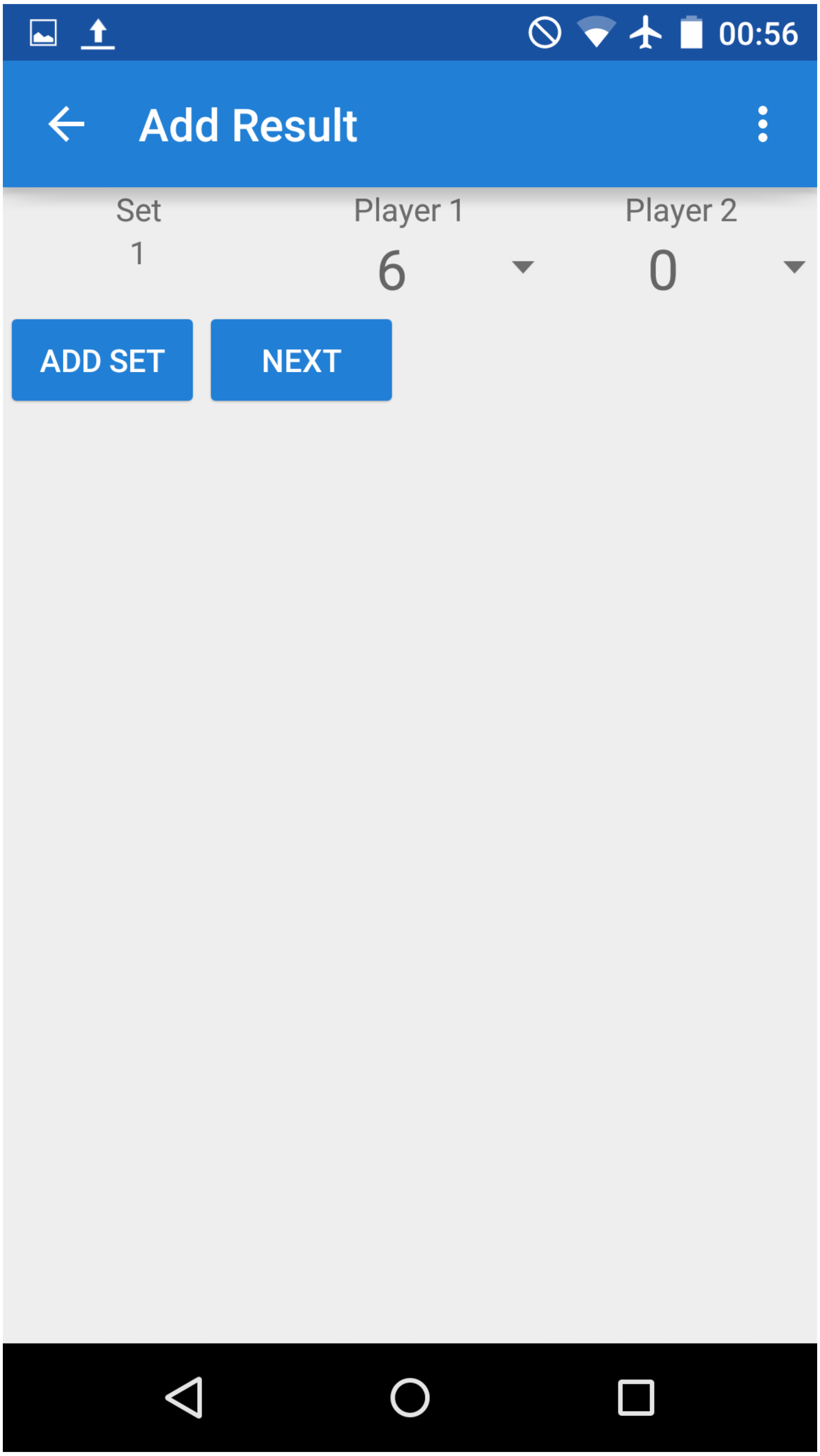
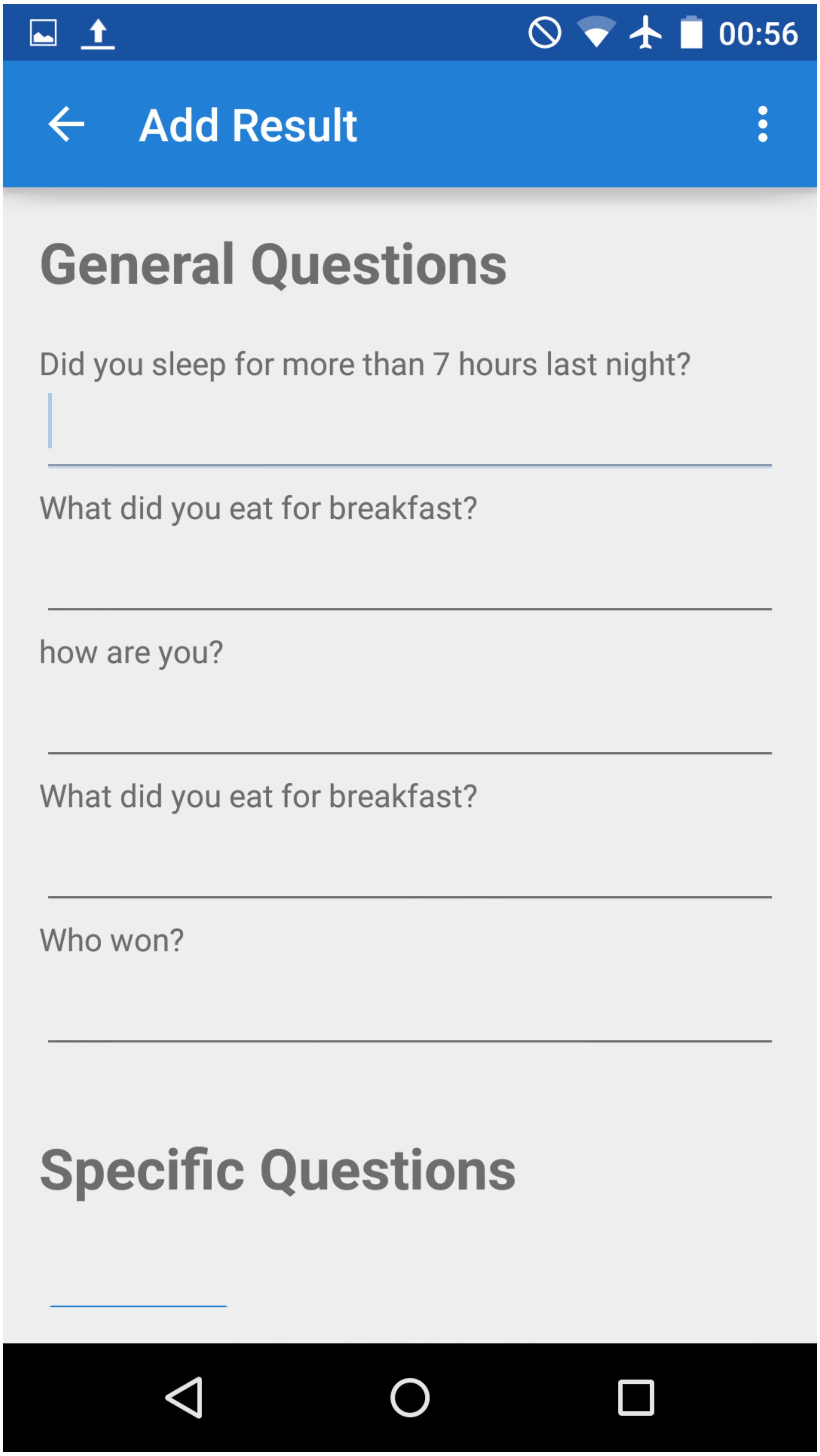


Figure A.4: Add results – Set scores



General Questions

Did you sleep for more than 7 hours last night?

What did you eat for breakfast?

how are you?

What did you eat for breakfast?

Who won?

Specific Questions

Figure A.5: Add results – Questions

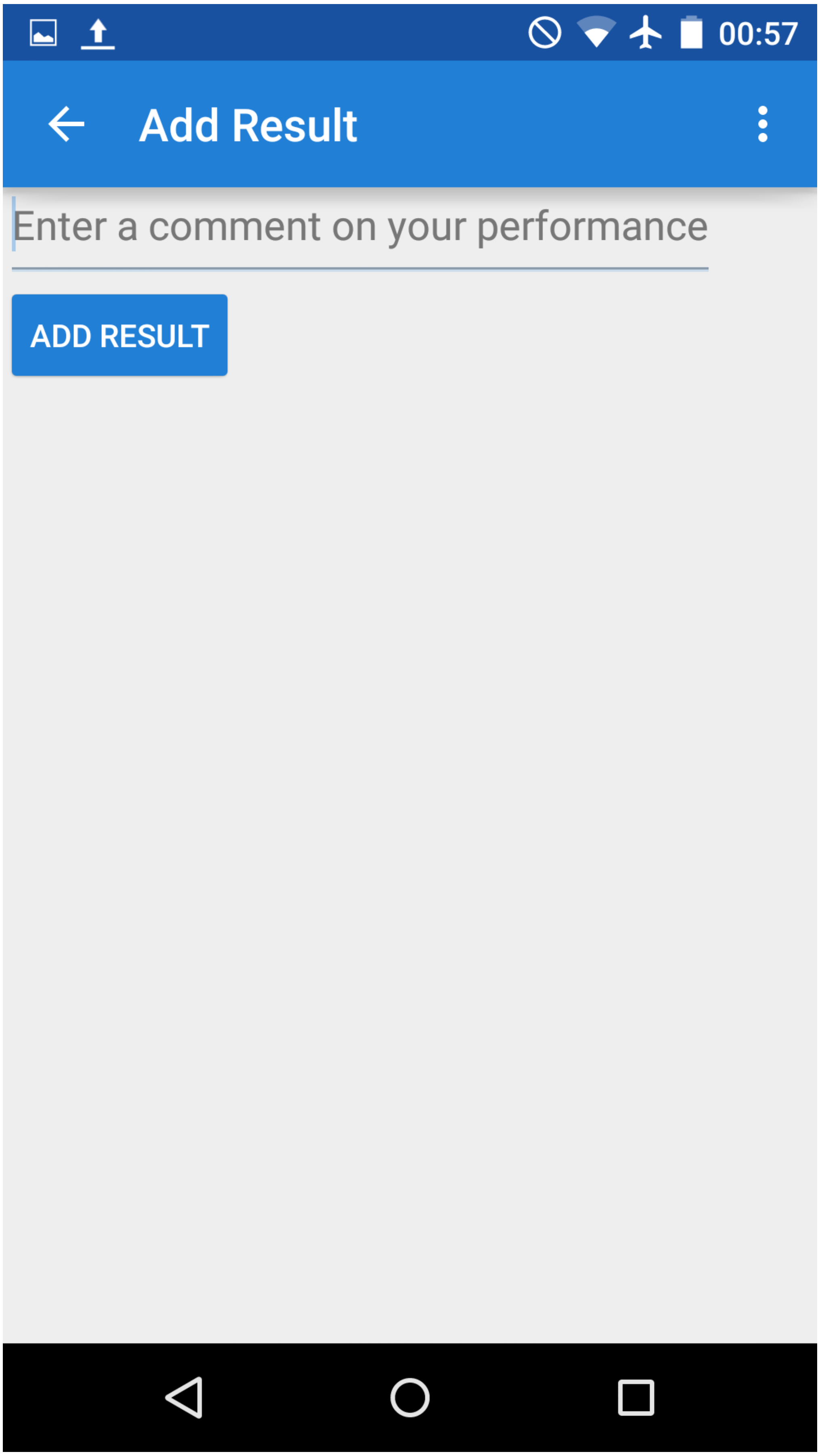
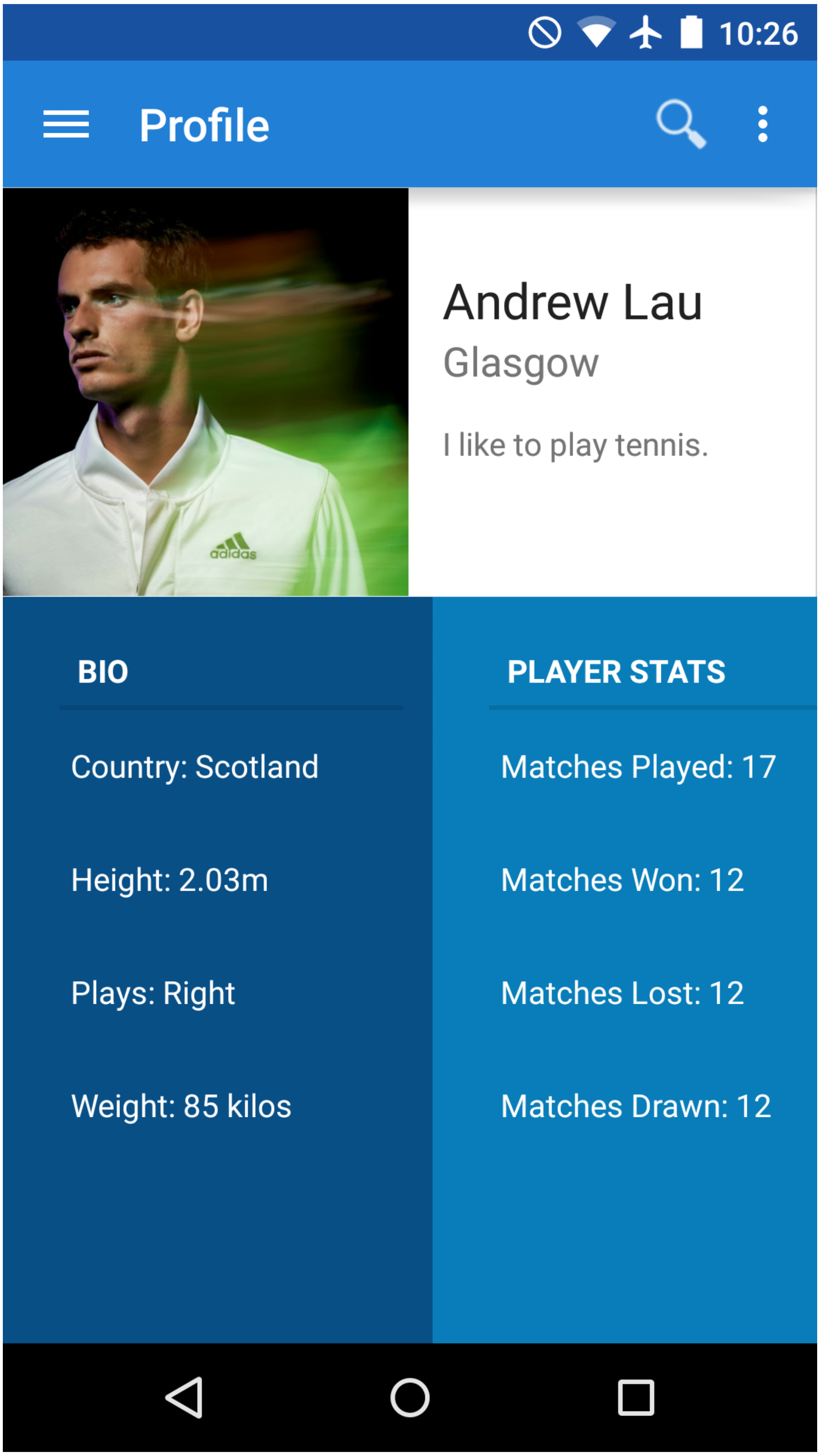


Figure A.6: Add results – Performance



10:26

Profile

Andrew Lau

Glasgow

I like to play tennis.

BIO	PLAYER STATS
Country: Scotland	Matches Played: 17
Height: 2.03m	Matches Won: 12
Plays: Right	Matches Lost: 12
Weight: 85 kilos	Matches Drawn: 12

Figure A.7: Profile Page

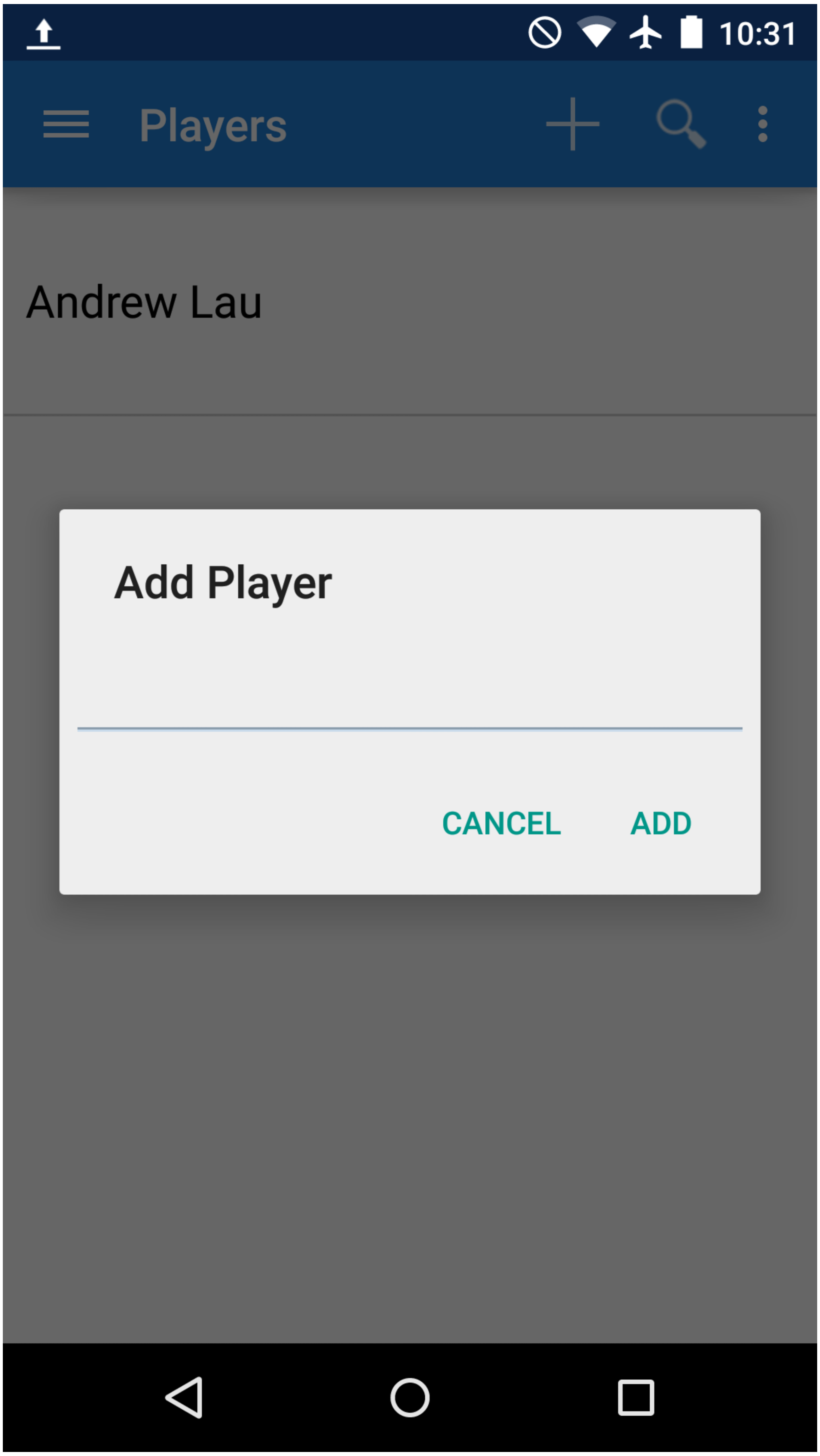


Figure A.8: Adding a player

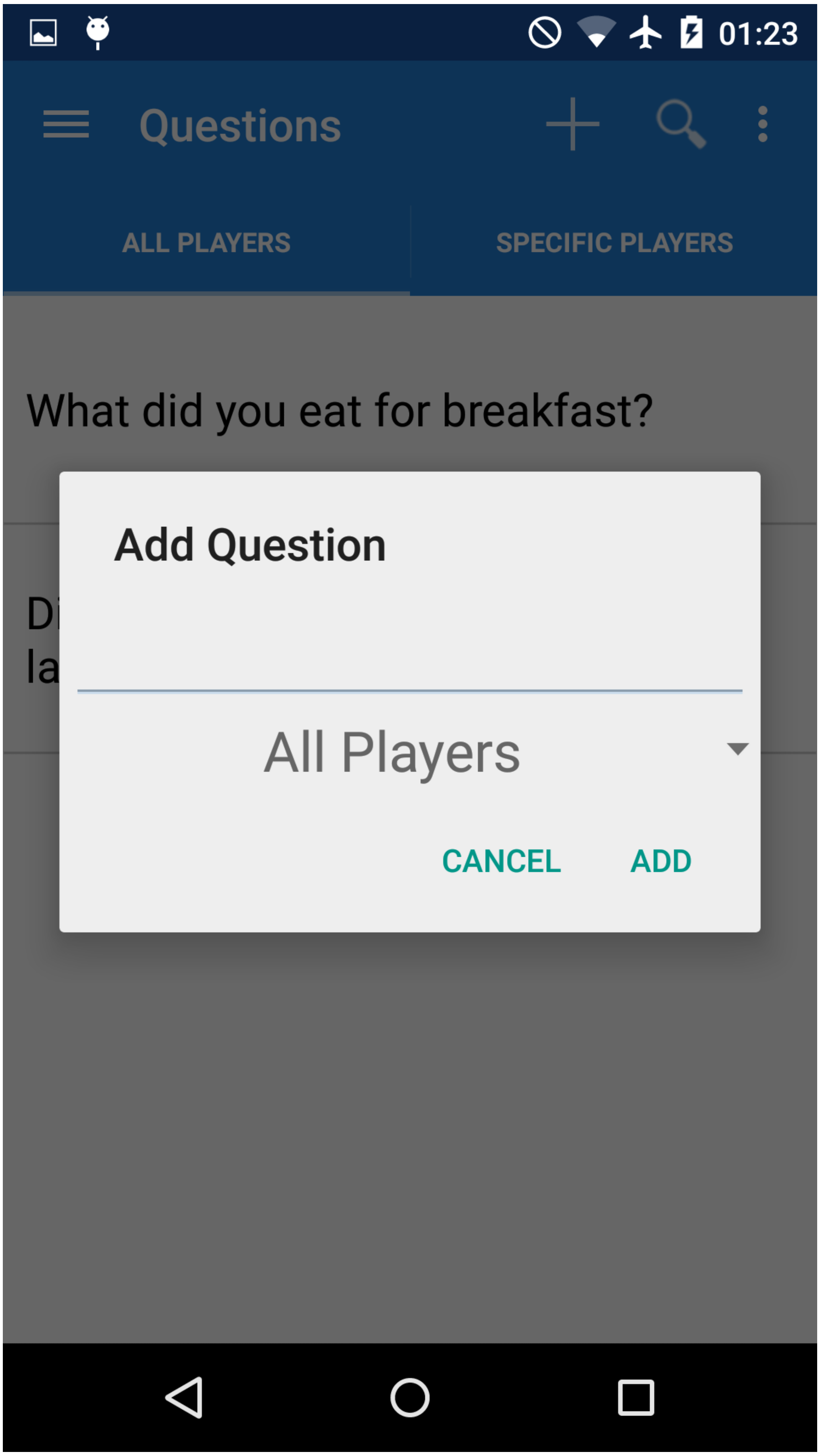


Figure A.9: Adding a question

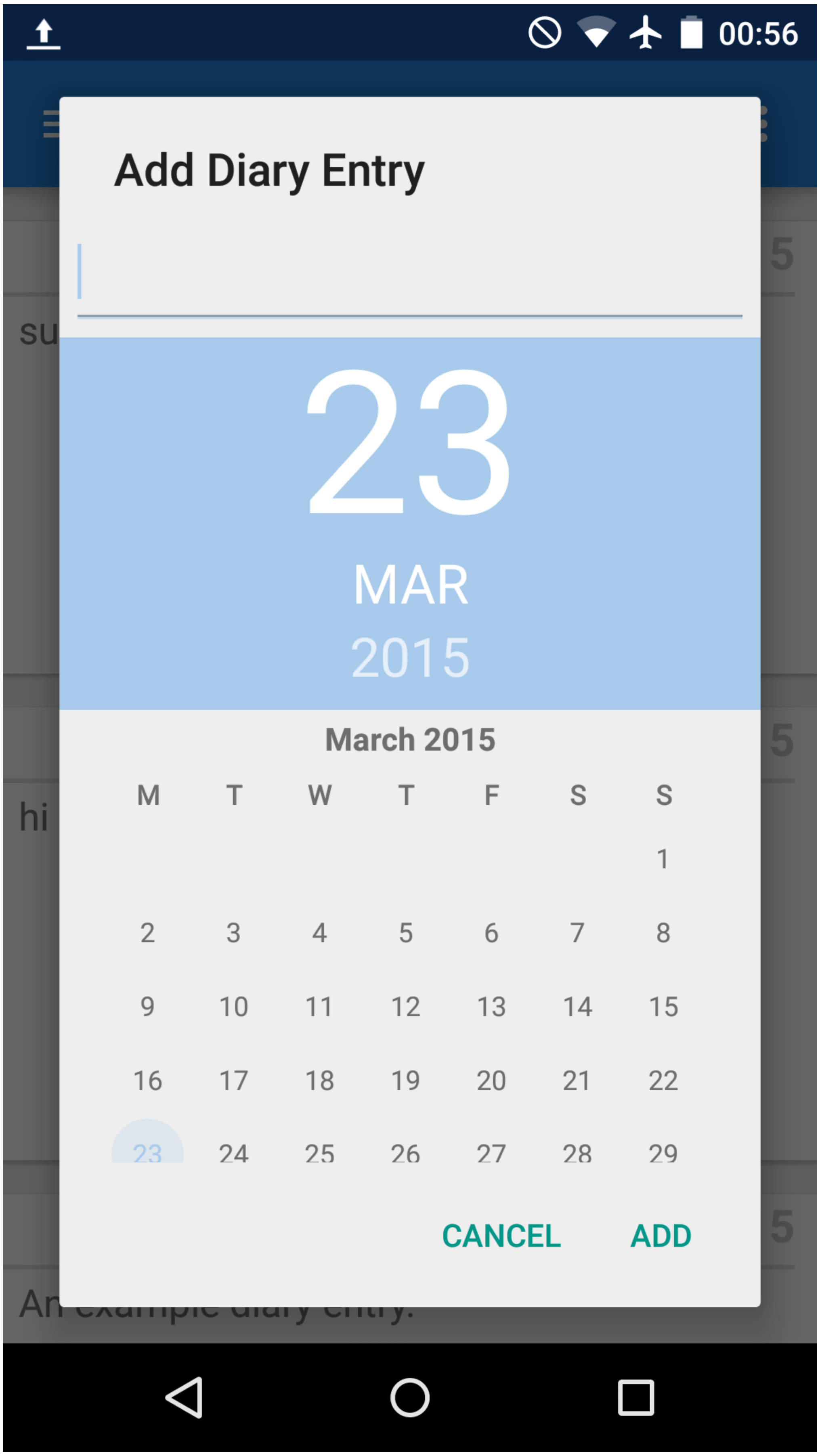
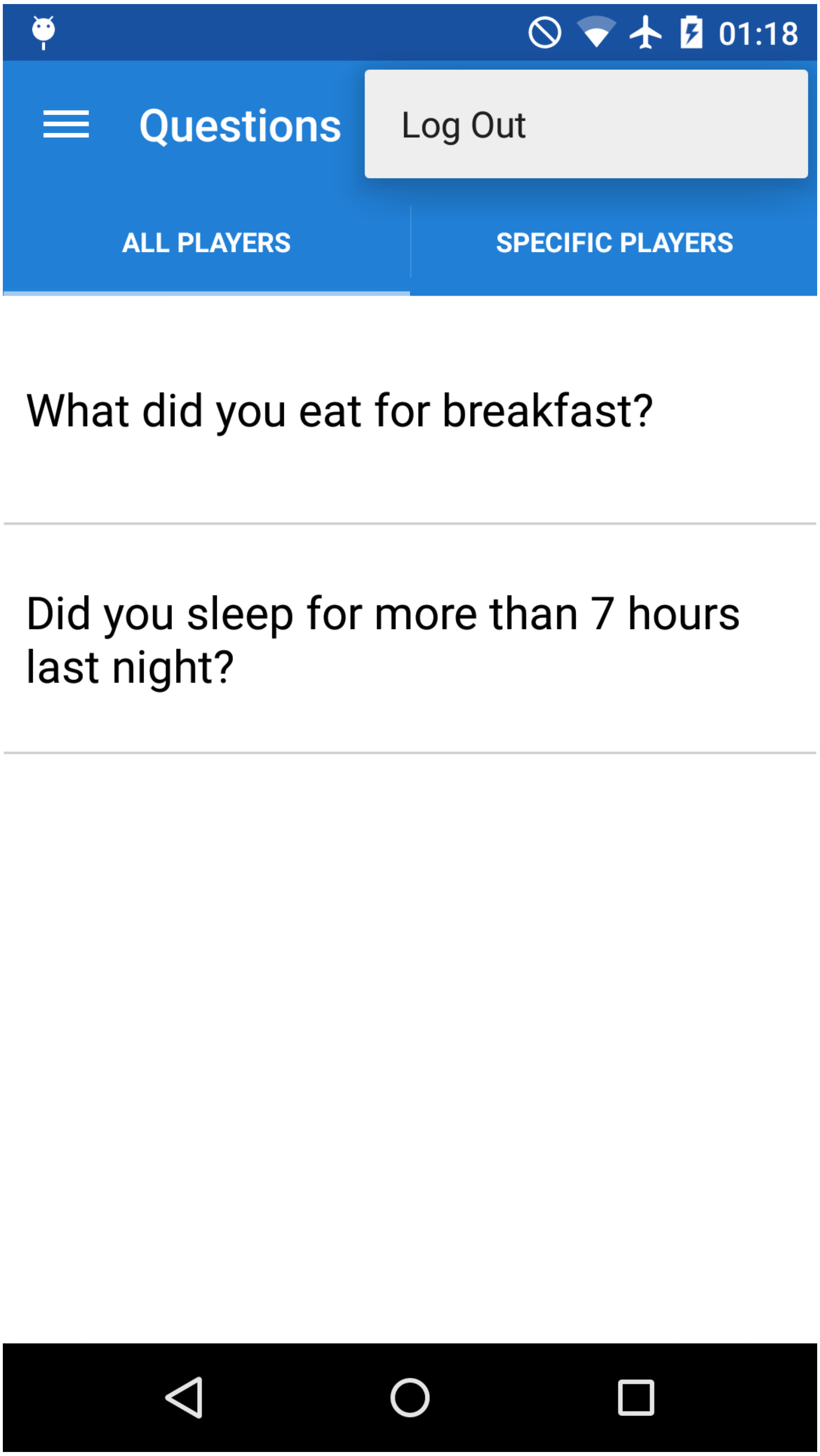


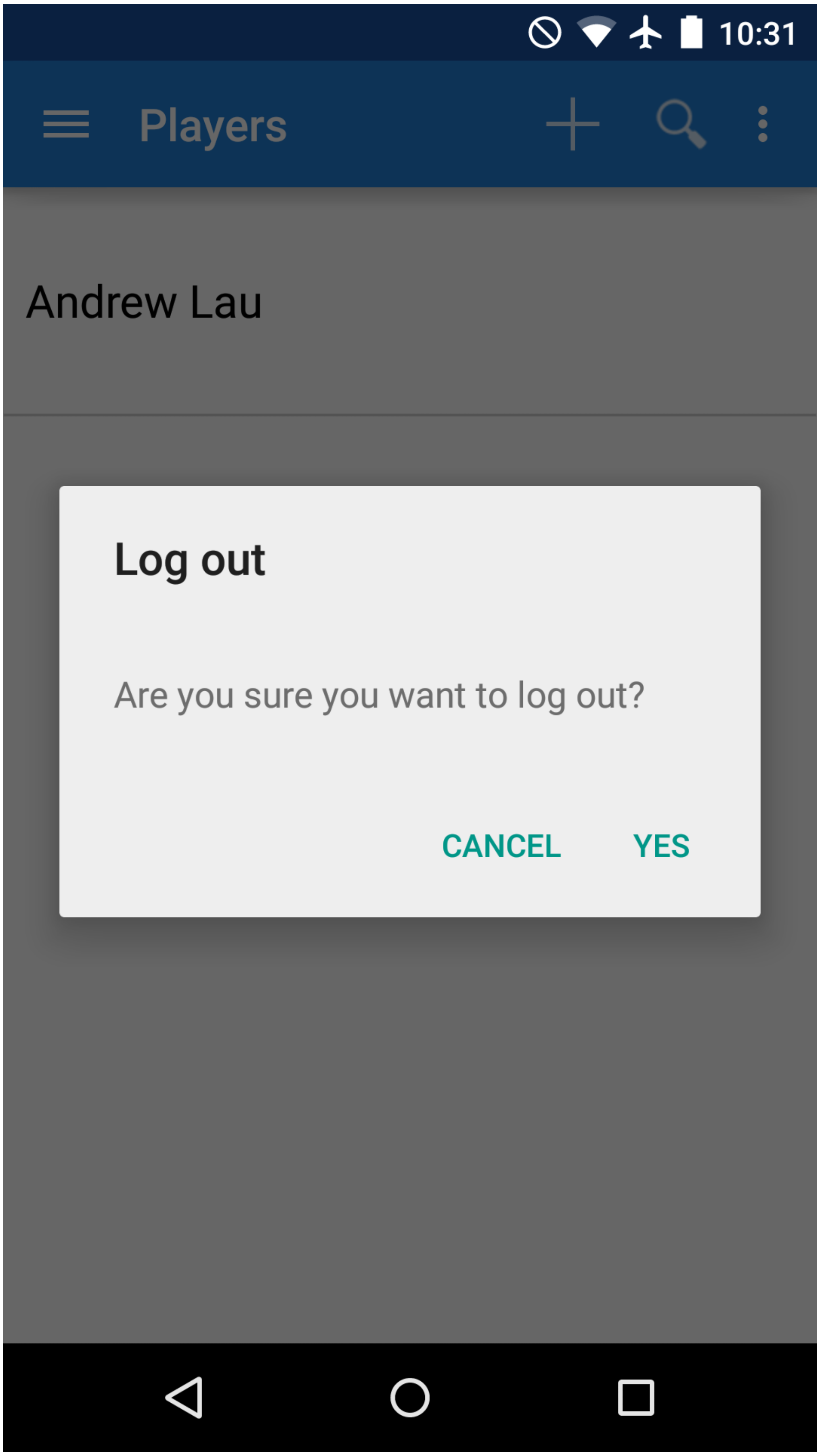
Figure A.10: Adding a diary entry



What did you eat for breakfast?

Did you sleep for more than 7 hours last night?

Figure A.11: Logout in menu


10:31

Players

Andrew Lau

Log out

Are you sure you want to log out?

Figure A.12: Logging out

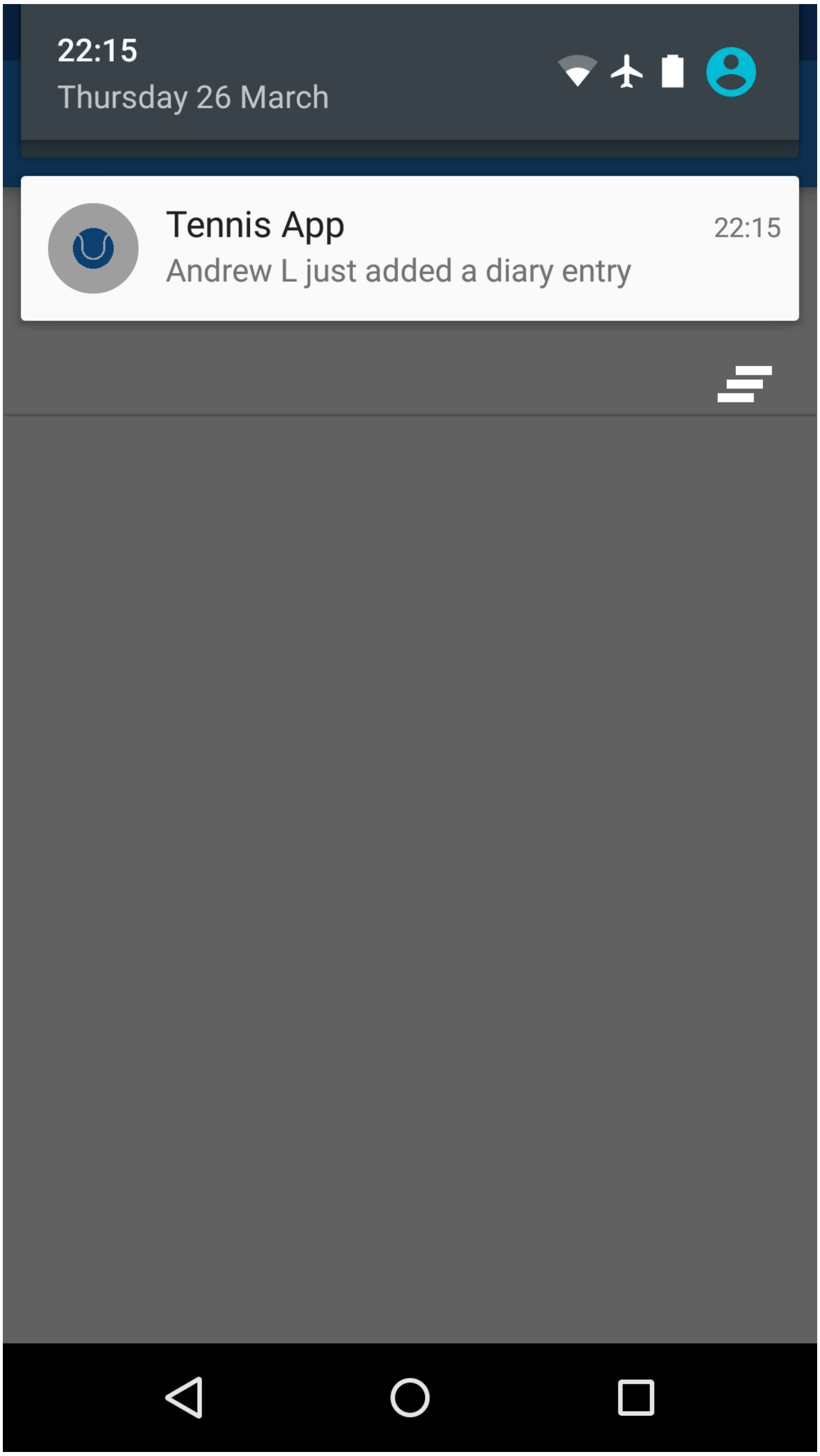
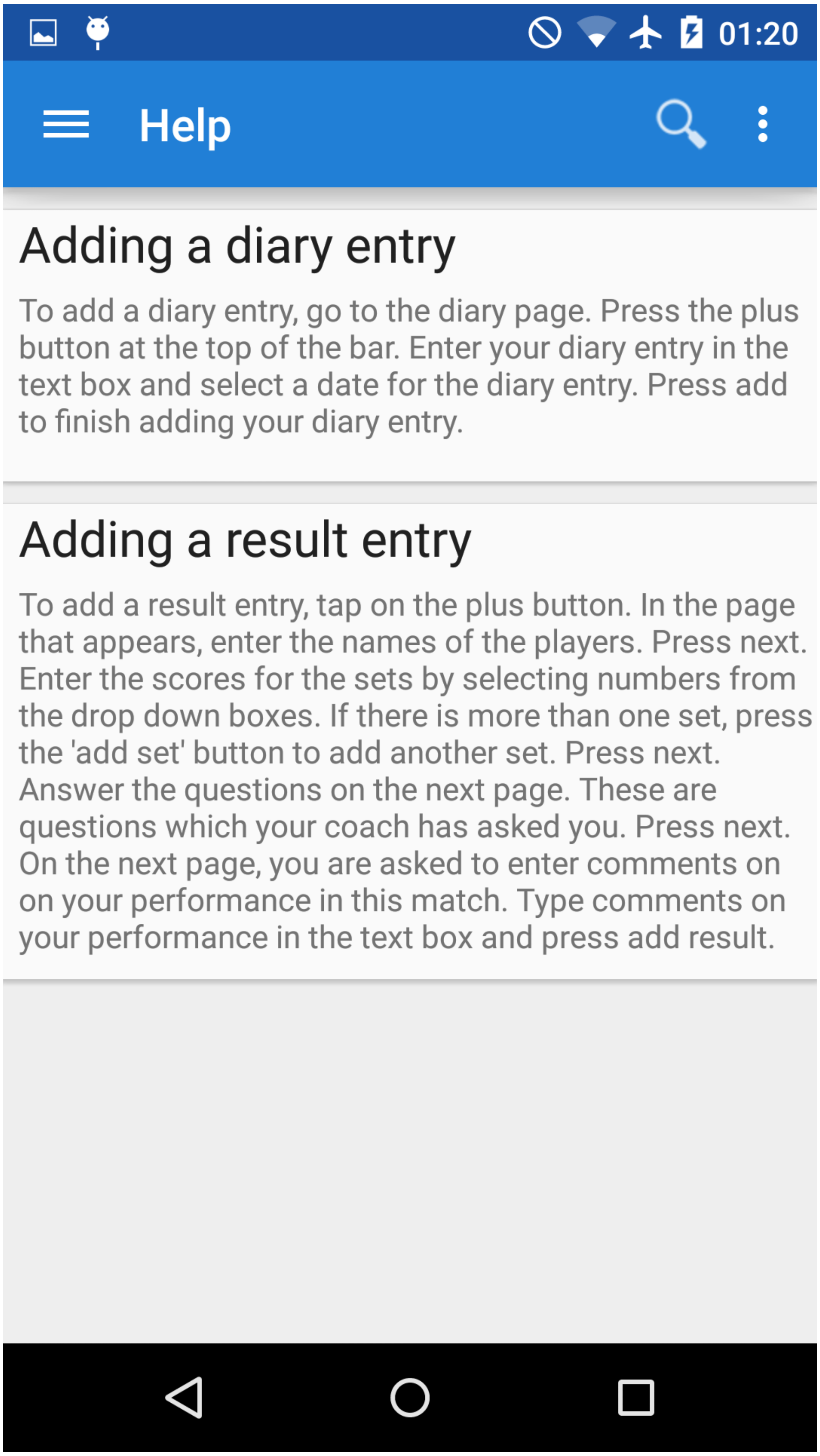

22:15
Thursday 26 March
Tennis App
22:15
Andrew L just added a diary entry

Figure A.13: Notification



Adding a diary entry

To add a diary entry, go to the diary page. Press the plus button at the top of the bar. Enter your diary entry in the text box and select a date for the diary entry. Press add to finish adding your diary entry.

Adding a result entry

To add a result entry, tap on the plus button. In the page that appears, enter the names of the players. Press next. Enter the scores for the sets by selecting numbers from the drop down boxes. If there is more than one set, press the 'add set' button to add another set. Press next. Answer the questions on the next page. These are questions which your coach has asked you. Press next. On the next page, you are asked to enter comments on on your performance in this match. Type comments on your performance in the text box and press add result.

Figure A.14: Help page

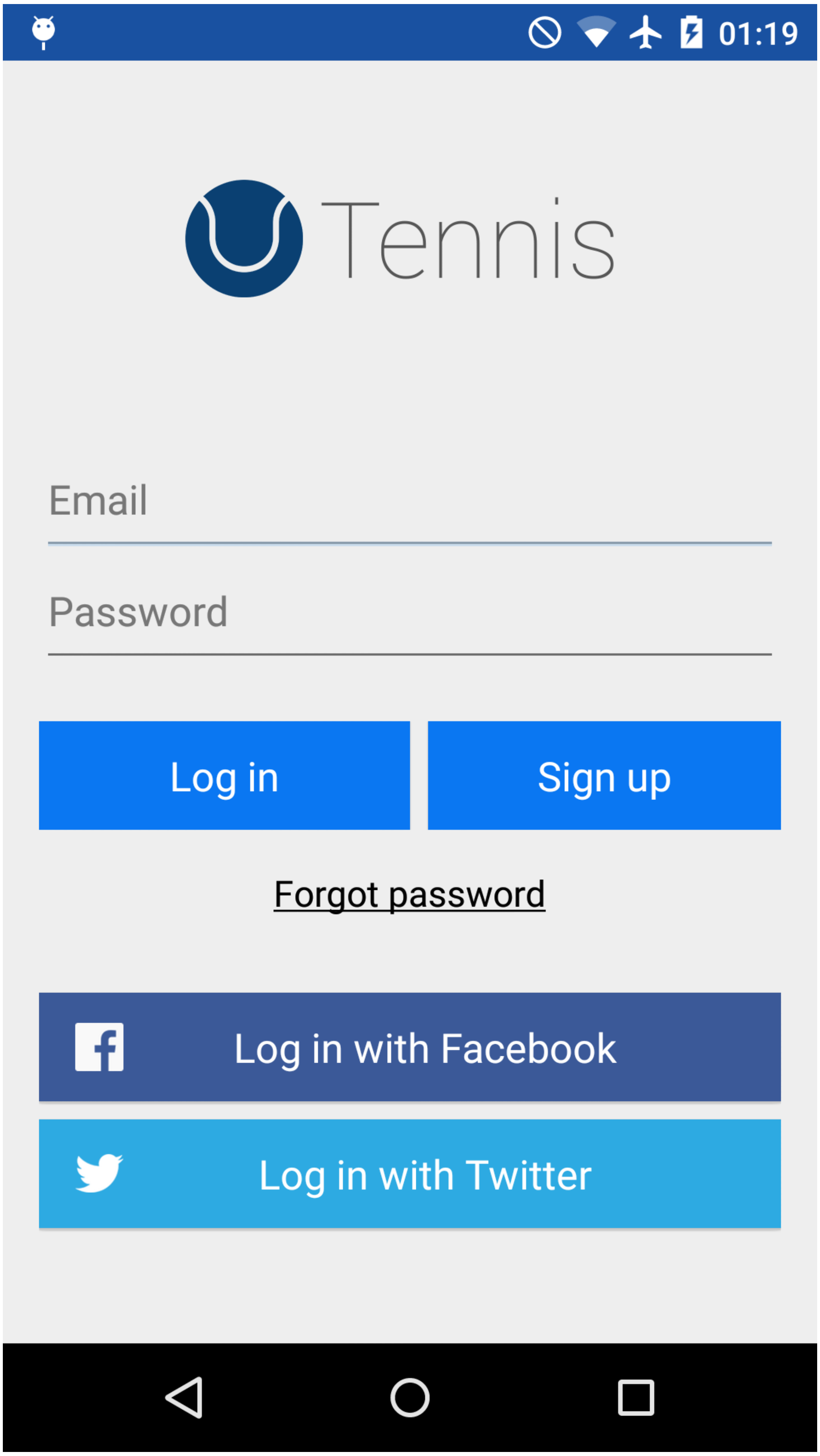


Figure A.15: Logging in

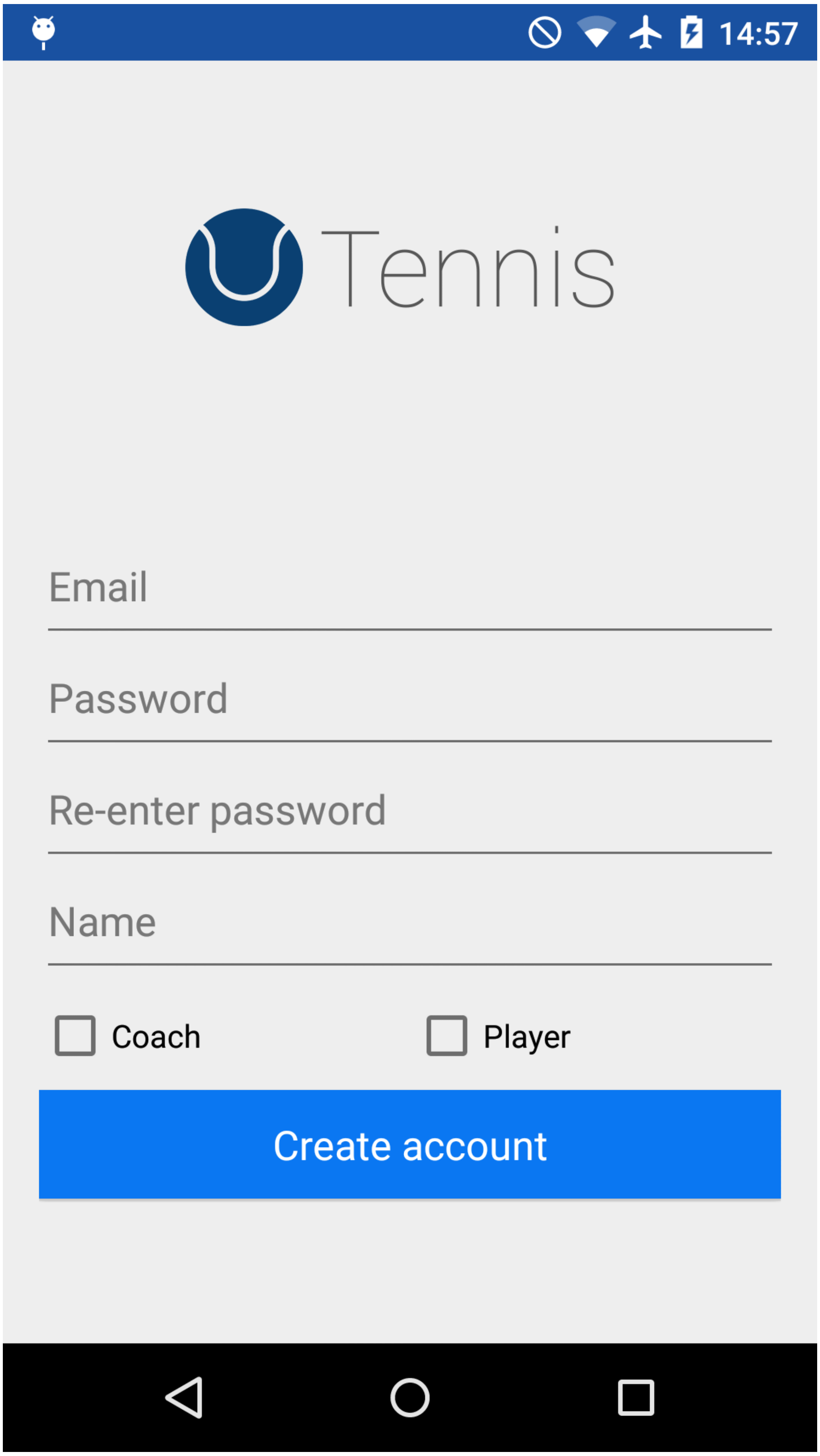


Figure A.16: Sign up – Mobile

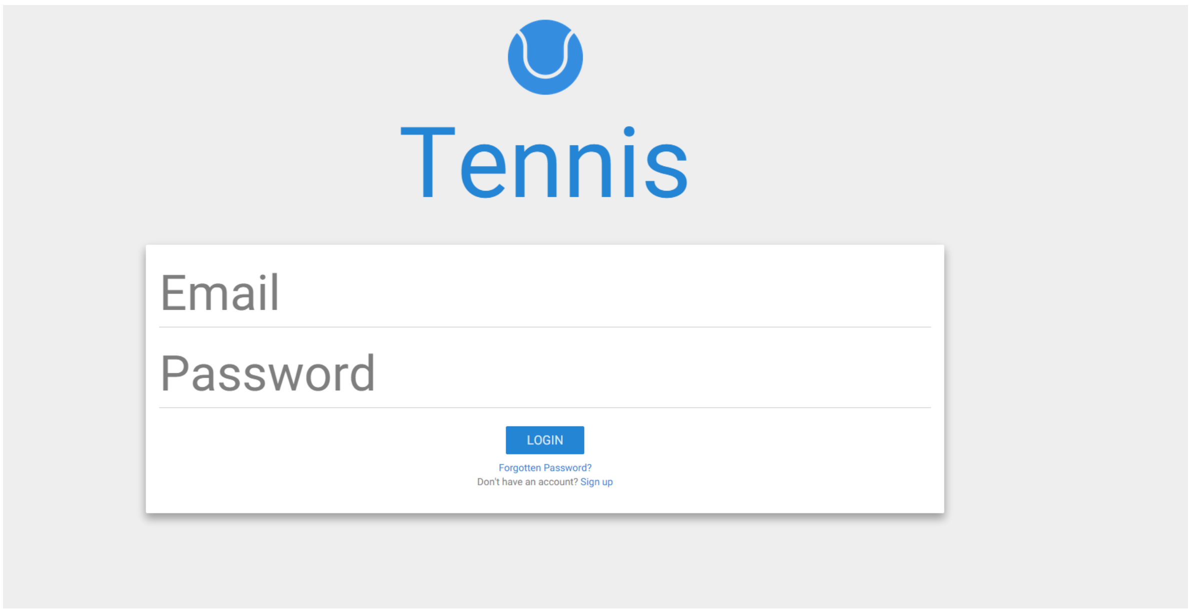


Figure A.17: Logging in on the web