Real-time Television Companion Application

James Carter

6th May 2016

Cardiff School of Computer Science & Informatics

CM2303 Final Year Project 40 Credits

Supervisor	Dr Xianfang Sun
Moderator	Dr Richard Booth

Abstract

The objective of this project was to develop a TV companion application for Android capable of scraping data online and displaying them in a human-readable form. This information includes title, description, images and airing times. This report contains identification of the problem, research, specification, design, implementation and evaluation of the application.

Acknowledgements

Firstly, I would like to thank my supervisor, Dr Xianfang Sun for his consistent support and understanding throughout the project. He provided me with confidence from the start by taking an interest in my project proposal. His feedback proved to be invaluable to the project's success.

I'd also like to thank my mother and father in which I could confide in when times proved difficult; without them, I would be lost without.

Finally, I would like to thank my partner and my closest friends. You all gave me the passion to achieve and were there for me when I needed you guys the most.

Table of Contents

Table of Figures

Introduction

Identifying Problem

As mentioned in my initial plan, the use of smartphones and/or tablets while watching TV is has become an increasing trend. Just under a third of smartphone users in the UK use a smartphone while watching TV (Laura Zain, 2015). With 22.1% conducting activities directly related to the program, it is apparent that people use their smartphones to enhance their TV experience.

From my own experience, I have found myself trying to find out information about a TV show that I am watching. When brought up in conversation it appears that many people do the same. The majority of the time, it can be achieved by simply using a search engine to search for the show. However, I find that this can take a long time as one would have to filter through the results offered by the search engine. Not only that, but the information returned can be of different qualities and formats. Information that was once previously available may also be deleted in the future due to websites changing content.

Project Aims

My project aims to improve the usability of smartphones while watching TV by creating a companion application that provides information about TV shows. This can be information about shows that are currently on, or by allowing the user to search for shows. During this project, I have produced a fully-functioning Android application that is capable of allowing users to browse popular British TV channel schedules, search for TV shows, bookmark shows for future reference and access information for TV shows and episodes.

I believe this project is important as I aim to pursue a career in mobile software development and as of yet, I have not had an opportunity to fully develop an Android application during my time at Cardiff University.

This is also the first time I have been able to fully complete a software development project by myself from start to finish, an experience I feel is imperative to my own development and overall professional skills.

I am also extremely interested in web scraping and how it can be used to display information in a more human-friendly manner, another opportunity that this project has provided me that would have otherwise been missed.

My project aims to improve the usability of smartphones while watching TV by creating a companion application for Android.

Target Audience

As covered in the initial report, two thirds of people in the UK own a smartphone (Media.ofcom.org.uk, 2015) and we are ever spending more time using them. Smartphones are a convenient tool for accessing the internet, especially when watching TV. I have aimed to develop this application to appeal to anyone who has an Android smartphone and has a basic knowledge of how to use it.

Scope

Having some initial idea of what was required to develop an Android application, I was aware that I would be programming in Java for Android and also working with XML files. I was also aware that APIs normally return XML or JSON files. I have had little experience in all of these fields so knew that the scope of the project would be limited by the fact that this project would also form part of a learning experience in Android development.

Nevertheless, I had a rough idea of how I would implement the system using Java, a language I have considerable experience with; All that was needed was research into applying this knowledge to Android and how to work with HTTP API requests, already a keen interest of mine and something I felt would be easy to learn given the multitude of resources available on the internet.

Project Objectives

In order to best prioritise the aims of this project, I have categorised them as 'Must Have', 'Should Have' and 'Could Have'.

Must Have: These must be implemented to deem the project successful

- A method to access the following information of TV programs:
  - Episode Name
  - Date aired
  - Description
  - Series Name
- A method to process this information into a human-readable form
- A GUI (Graphical User Interface) that follows basic Human Computer Interaction goals
  - A way for users to *select* a TV program
  - A method to display information of selected show
- A way for users to *browse* TV shows:
  - This should mimic a traditional EPG
- The system must be deployed on Android for smartphones

Should Have: These will assist in the project's overall success

- A way for users to *search* for a TV show:
  - The user should be able to type in a show and select from a list of results
- A way for users to select which sources to fetch the information from
- A built-in user guide to enhance usability

Could Have: These are additional aims that add value to the successful project

- A way for users to bookmark favourite shows
- A way for users to set reminders for when a favourite show is airing

Research

Existing Solutions

There are already existing TV companion applications available to download for Android, the majority of these applications are free to download and are supported by adverts to cover costs.

The Freeview TV Guide is one example of such an application (Play.google.com 2016[1]). This application displays what shows are being aired on each Freeview channel, much like a traditional EPG. It also displays upcoming shows, a feature to favourite a show, a recommended show section based on favourite shows and Twitter integration. Although this may appear to be an ideal solution, it should be noted that the application has not been updated since July 2015 and has received negative reviews regarding stability, random crashes and intrusive adverts.

Another existing solution is the TV Listings UK application (Play.google.com 2016[2]). This application also displays shows being aired and upcoming shows. The only additional feature this application has the ability to set a reminder for a show. Like Freeview TV Guide, this application has adverts, something my solution will avoid in order to improve usability.

Feersum UK TV Guide (Play.google.com 2016[3]) is yet another potential existing solution. It features an EPG complete much like other solutions. This application allows the user to highlight genres as well as search channels. Feersum UK TV Guide also features a colour-coded system where films are shown in orange text and finished shows are grey. This application has received more favourable reviews on the Google Play Store, however, it has not been updated since 3rd of April 2014.

UK's TV Guide Free (Play.google.com, 2016[4]) is another example of an existing solution. It features a basic EPG style list that allows users to select a channel to see what is scheduled to be aired. The application allows users to see a list of every show that is currently being aired. Users can also search for content. This application hasn't been updated since March 2015 and the only available review states that the application doesn't load any data.

These applications all have similar functionality of my planned solution. However, they are generally outdated and rely on adverts to cover their costs. Main broadcasters such as the BBC, Channel 4 and ITV offer their own applications for viewing schedules on their channels. However, this would mean the user has to have many applications to view schedule information which is far from ideal.

	EPG	Search	Reminders	Bookmark/Favourite	Adverts
Freeview TV Guide					
TV Listings UK					
Feersum UK TV Guide					
UK's Television Guide Free					

Figure 1: Comparison of existing solutions

Data Sources

In order to develop a solution, it is necessary to provide information about television shows to the user. In theory, it would be possible to hard code in this information, however, it would be extremely time consuming and memory intensive. Updates would need to be pushed very frequently to ensure future shows are displayed.

Alternatively, data can be accessed and saved in a central database. As the information aims to cover many different TV shows, the database necessary would be huge and would require frequent updates as and when new TV shows are aired. Not only that but representing airing times for channels would require the system to be constantly updated as channels list their schedules. My solution is non-profitable; it would be economically ineffective to purchase a subscription for a server to be able to host this information.

Another method for dynamically accessing TV data would be through the use of existing APIs. This method avoids costly servers and takes advantage of data that is already available.

There are many websites that offer TV data through use of an API in which the user sends an API request. This includes a unique API Key and what data is required, for example, 'GET /4/schedules/{X}', using Metabroadcast's API (Atlas.metabroadcast.com 2016) would return all shows scheduled for channel with the ID: X. The general format for TV data is through XML (Extensible Mark-up Language), a common format used to store and transport data. It provides users with both human-readable and machine-readable content, useful for displaying data and supported in Android. There is also a lesser common format called XMLTV, an extension of XML that is used for describing TV listings, however it is not widely supported.

The BBC have their own API for shows aired on the BBC network (Developer.bbc.co.uk 2016). Documentation for their API is currently in a draft status with limited guidance on how to use it and as it only includes data for BBC broadcasts, it would only provide a partial solution.

Another API that could be used is the Digiguide.tv API (Digiguide.tv 2016), a free and open platform that aims to support TV listings for its users. The API has well-structured documentation to aid development and supports JSON responses, a simpler data-exchange format than XML that is also supported in Android. The API shows information about Terrestrial, Sky Digital and Virgin Media broadcasts in the UK and Ireland, ideal scope for my solution. Information about shows such as seasons, images, airing time and actors can be retrieved using Digiguide.tv's API.

TheTVDB (Thetvdb.com 2016) also features an API similar to that of Digiguide.tv, however, it is an open database that can be modified by anybody, and it also supports the XML format. This allows for extensive information to be produced for TV programs in virtually any country from regular users, this may include additional fields such as fan art, director and even DVD disk number. The API also features useful documentation and support for developers. Considerations must be taken regarding the openness of this data, it may be inaccurate despite having a large user base of honest people. Also, it should be noted that airing times aren't always supported.

The above APIs all have their advantages and disadvantages; it would be desirable to use all of them to create tailor made information in my solution. The final data source I have discovered is Metabroadcast's API, it accesses data from multiple sources and stores it in a convenient, flexible and highly accessible format which allows for developers to take advantage of otherwise hard to access data. Sources include the BBC, Press Association, UK Channel Lineups, Wikipedia and TheTVDB.com

There are many levels of customisation using Metabroadcast's API including retrieving TV schedules for many UK channels with optional time constraints, i.e. shows between a given set of time points. The API also allows for retrieving other information about shows including description, episode number, release date, synopses, encoding, episode image, certificate rating and much more. The only limitation this API has is the lack of real support for actors/actresses. The system requirements will be adjusted to accommodate.

Metabroadcast's API appears to be an ideal core data source to aid development of my solution as it will provide both basic EPG functionality as well as extended information. In order to gain access to the API, I must sign up for an API Key. Metabroadcast allows users to sign up using Twitter, Github or Google. I have selected Google as I use Google Drive to store project documents.

Available IDEs

There are many IDEs that help developers to develop software quickly and effectively. Android requires the use of Java to develop applications, because of this, generic Java IDEs can be used for Android development. However, Android extends Java and includes XML files for GUI design.

The most common IDE for Android development was Eclipse with the Android Development Tools plugin. This set up was directly supported and encouraged by Google until 2013 when they released their own IDE called Android Studio (Android Studio 2016). Android Studio is the official IDE for Android development, it includes coding templates to aid development as well as a layout editor that supports drag and drop editing of the GUI, a very useful feature to help provide optimal HCI goals. As Android Studio is Google's official IDE, it uses Java for native development, a familiar programing language to myself with support for XML/JSON parsing that is required to process TV data. It also provides a facility to produce multiple versions of an application, useful for updating and debugging. Android Studio features device emulation, this means that the software can be tested on an emulated device straight from the computer without the need for a physical device.

Another IDE to aid Android development is Xamarin.Android (Xamarin.com 2016). This IDE allows developers to produce packages in C# by bringing Android's Java API to the C# language. It also allows developers to call existing Java code without the need to convert it into C#. This IDE would be very useful if C# was a language I was familiar with, however, I am more familiar with Java.

Kivy is a possible library to use for developing my solution (Kivy.org 2016). It is an open source cross platform library that allows the production of Android applications using the Python programming language. It carries the obvious advantage of Python's simplicity and ease of use, however, it doesn't feature a GUI editor like Android Studio does. I also feel that this IDE would not encourage my project aims of learning to develop traditional Android applications.

Yet another IDE that can aid in Android development is IntelliJ IDEA (JetBrains 2016). This IDE aims to provide intelligent and automatic coding assistance; this includes code completion based only on what is expected in the current scope, version control and even an Android GUI editor (similar to Android Studio). IntelliJ IDEA also supports JavaScript, HTML and CSS which may provide useful in Android development.

Overall, I believe Android Studio will provide me with the best IDE to use as it is Google's official IDE and comes with a lot of support. Android Studio is also commonly used in examples and tutorials so will be ideal as an IDE for me to learn Android development with.

	GUI Editor	Language	Version Control
Android Studio		Java	
Kivy		Python	
Xamarin.Android		C#	
IntelliJ IDEA		Java	

Figure 2: Comparison of IDE options

Android Version

As technology has progressed, there have been many versions of Android released to support new features and devices. Google decided to give each version of Android a name and an API level. Android studio summarises each version from version 2.3 (API 10) up to the latest 6.0 (API 23) to help the developer select what minimum version to support.

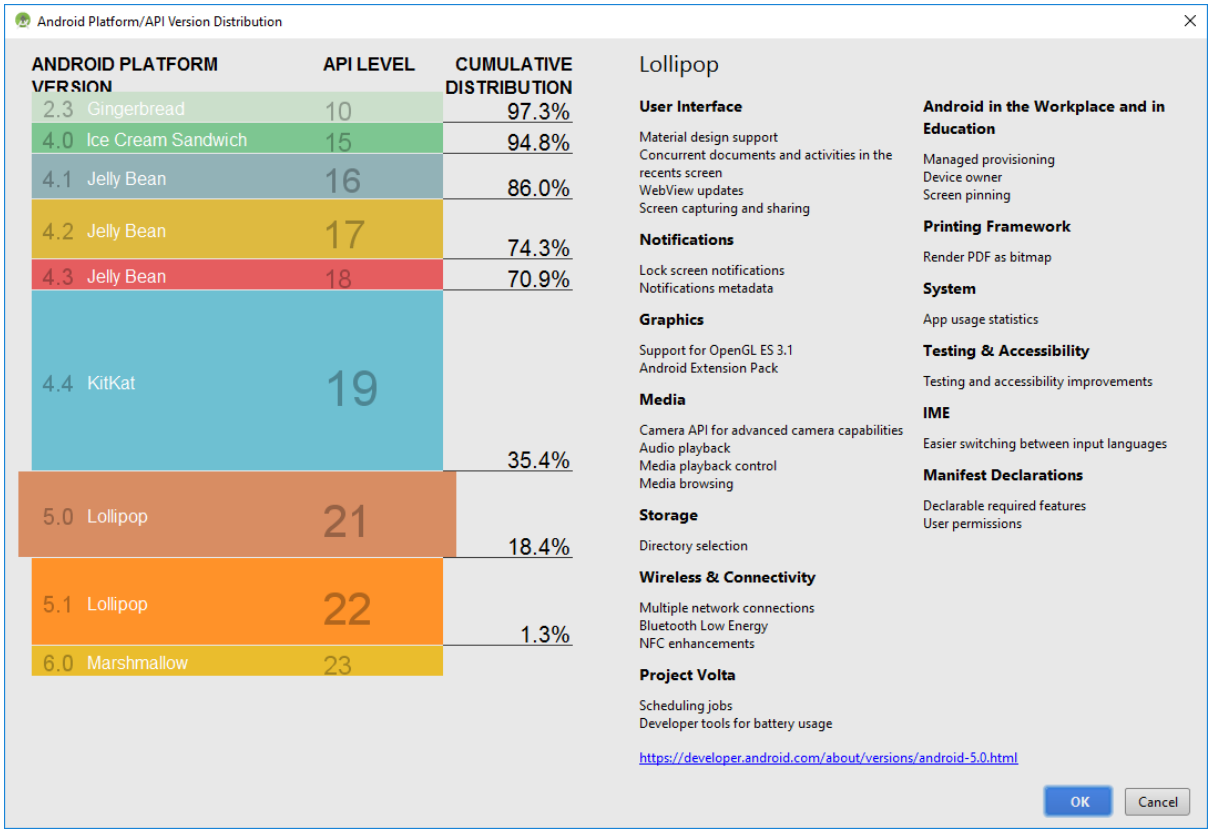

Figure 3: Android Summary from Android Studio

As this a non-profit project, appealing to a large market share is not a high priority. However, following basic Human Computer Interaction goals is an essential requirement, it is important to ensure good interface design is achievable. Android version 5.0 (API 21) saw the introduction of material design, a lightweight and minimalist visual language that enables developers to create consistent designs across multiple devices and screen sizes. This 'flat' design is now a commonly accepted design concept that many users are used to.

As Android Studio has a GUI editor, it would be preferable to support API 21 and up to enable the use of material design. Selecting API 21 still yields an ever increasing device target of 35.4% (February 2016) and with manufacturers constantly aiming to support the latest version of Android, I feel this is a good level to support the application if I were to release it on Google's Play Store.

Specification

Approach

I have decided to follow the classic waterfall model given it is what I feel most comfortable with, and as this project already contains a lot of personal learning goals, I feel it is wise to stick to a development model that wouldn't interfere with the learning process. Another reason why I felt this model was most appropriate was the fact that this project requires an initial plan, outlining the process and setting out some initial project aims and objectives that would later come to form system requirements.

Other development models such as Agile are iterative, meaning that requirements are liable to be changed, including the implementation, design and all other key areas of the system. As my project has clear system requirements set by myself that do not depend on an external client who may want to change them, there is no requirement to need to change development once requirements have been set.

The waterfall model also clearly outlines traditional software development into five main categories:

1. Requirements
2. Design
3. Implementation
4. Testing
5. Maintenance

This meant that I could create a detailed Gantt chart to aid development, helping me to keep on track and outline each stage of development complete with sub-goals at each stage. Having a structured work plan is essential to maintaining pace and producing quality software.

System Requirements

Following the project aims and objectives I have outlined the requirements of the system in order to be deemed a success. Each requirement has a priority with 'Must Have' requirements having the highest and 'Could Have' having the lowest. Categorising system requirements enables the developer(s) to identify sub-goals which helps with planning the implementation. Understanding what is required for the system to be a success is essential if a quality solution is to be implemented.

Should the requirements change later in the project, having categorised requirements helps minimise the impact changes have, a critical factor when developing a system for a real-life client.

Must Have

Req ID	Requirement	Description
101	Method to access information of TV programs	The system must be able to access information for TV programs. This must include: ▪ Episode Name ▪ Description ▪ Date Aired ▪ Series Name
102	Method to process information into human-readable form	Information from Req 101 must be parsed and processed into a human-readable format
103	A Graphical User Interface	The GUI must follow basic Human Computer Interaction goals to ensure ease of use
104	A way for users to *select* a TV program	The user must be able to *select* TV programs from a suitable list/box
105	A method to display information on selected TV program	Using human-readable information from Req 102, the system must display this information to the user
106	A way for users to *browse* TV programs	The user must be able to *browse* TV shows much like a traditional Electronic Program Guide
107	Android support	The system must be deployed on Android

Figure 4: Table of Must Have requirements

Should Have

Req ID	Requirement	Description
201	A way for users to *search* for a TV program	The user should be able to type in a search query and select a program from the results.
202	A way for users to select data sources to fetch information from	The user should be able to enable and disable the various data sources used to fetch the information, i.e. TheTVDB API
203	A built-in user guide	The system should have a comprehensive guide to aid both basic and expert users

Figure 5: Table of Should Have requirements

Could Have

Req ID	Requirement	Description
301	A way for users to bookmark a program	The user could 'favourite' a particular program in order to bookmark it, making it easier for the user to access the program again
302	A way for users to set reminders	The user could set a reminder for a particular show to remind them when it starts

Figure 6: Table of Could Have requirements

Non-functional Requirements

It is important that the system adheres to basic non-functional requirements as system requirements do not cover these principles that are followed by nearly all professional software developers. Following the table below will ensure the system will be of high quality throughout.

Requirement	Description	Measurement Criteria
Usability	The system must be easy to use	Users should be able to use the system with minimal guidance from external sources such as other users
		Intuitive GUI
Reliability	The system must run with zero errors or bugs	The system should pass all test cases
		The system should run on multiple Android devices
Maintainability	The system must be easy to change and maintain	The system should be easy to modify to maintain its functionality throughout its lifetime
		Comment all code
		Code easily accessible
Performance	The system must perform efficiently	System should work in real-time with little to no loading times
Scalability	The system must continue to function as its scale increases	Adding more channels to the EPG should have no ill-effects on the system
Robustness	The system must be able to recover from errors	System design incorporates error handling throughout
Security	The system must be able to store user information (if any) securely	System should store sensitive data with password protection
Flexibility	The system must allow for the change of functionality without damaging the system	System should be modular allowing updates to have no effect on other areas of the system

Figure 7: Table of Non-functional requirements

Risk Assessment

As this is a large project, it is useful to assess any possible risks that may have a negative impact on progress. It is important to consider different forms of risks and how best to reduce the impact and likelihood of these risks occurring. Figure 8 shows various risks I have assessed and how I plan to reduce them.

Category	Potential Risk	Risk-management Technique to Use
Timing	Underestimating time needed for the project or parts of the project	I will follow the work plan and Gantt chart proposed in my Initial plan
Human Resources	Illness may delay progress	Ensure work plan is followed at all times to minimise impact of illness.
		Liaise with supervisor to ensure illness is known
Human Resources	Technical ability may be inadequate to fulfil a requirement	Scope of project is confirmed as adequate for course by Supervisor
		Make use of resources available for Android development
Requirements	During the course of the project the requirements may change	Ensure final Aims & Objectives are completed and confirmed with supervisor before implementation
		Any subsequent changes can be absorbed in the 1-week buffer (16/03-25/03)
Productivity	Procrastination due to length of project	Follow Gantt Chart in order to maintain pace. Many sub-goals each week make the workload evident from the start and ensures I am aware of what is needed to be done each week
Technical	System may not be thoroughly tested resulting in errors and requirements not being fulfilled	Create an in depth test plan for each requirement
		Carry out tests when each requirement has been implemented, fix any errors where necessary

Figure 8: Table of Risk Assessment

Design

During the design stage of any software development, it is important to undertake various methods to fully design and understand the way in which the system will work both systematically and functionally. I have designed a Use Case diagram to help visualise how each feature works and how the user interacts with them in order to achieve their induvial system requirements, this also includes how different features interact with each other.

A Use Case does not describe how the system will actually achieve their given requirements so pseudo code will be used to describe exactly how the system is planned to function from a code perspective. Pseudo code provides an invaluable way to demonstrate the logic behind a given feature. The original designer can use it to quickly develop the underpinning algorithms without having to directly implement it using a specific language; ultimately this saves a lot of time and does not require any coding knowledge as such. However, time does have to be taken to then adapt this pseudo code to fit in with the boundaries of a coding language. Pseudo code benefits from the fact it can be used as a way to demonstrate how an idea or feature will work to someone else. For example, if working in a development team, pseudo code can be used to quickly describe how a system requirement will be implemented; alternatively, it can be used to describe complex algorithms and ideas to stakeholders and clients who may not be technically-minded.

To be able to fully evaluate the success of the finished project, test measures need to be put into place to test against each system requirement. I have decided to create test cases to test each functionality to ensure they are successful. Testing is also very important to ensure any bugs are discovered that may have gone unnoticed under normal use.

As my system will have a GUI, it is as equally important to design the way in which it looks and navigates. I have decided to create hand drawn designs initially to get an idea of how the system will look. These will then be converted to a computerised wireframe design and then finally produced into a final version using Android Studio. The benefit of creating hand drawn designs is that they are quicker to produce. This allows for rapid prototyping of the GUI without the need to conform to set standards from Android Studio's IDE. However, this also means that it may not be accurate as it does not conform to these standards. Creating a wireframe design aims to bridge that gulf by producing more accurate designs that conform to these standards. Wireframe designs take longer to produce but are of better quality, to the extent that they could be used to demonstrate a design to any potential stakeholders or clients. The final Android design can then be implemented, fully demonstrating how the system will look. I will include the final design within the implementation section to save repeating the same images.

Use Case

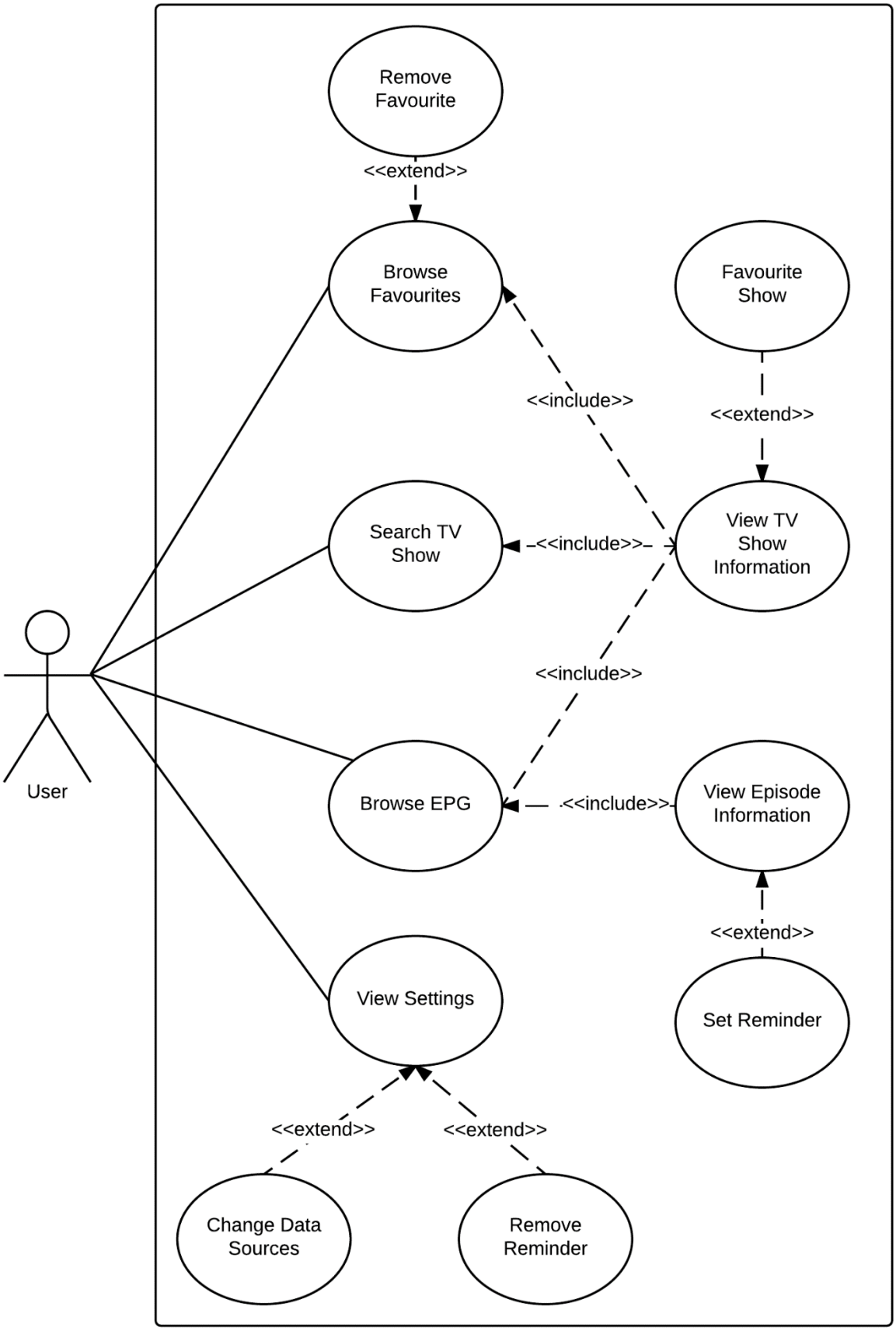

Figure 9: Use Case

Primary Use Cases

Use Case	Search TV Show
Actors	User (initiator)
Type	Primary
Description	The user clicks the Search TV Show button on the main screen of the system. The user then types in a search query and hits submit. This returns the results as a list.
Use Case	Browse Electronic Program Guide
Actors	User (initiator)
Type	Primary
Description	The user clicks the Electronic Program Guide button on the main screen of the system. The class returns the schedules for all channels as a 2D list.
Use Case	Browse Favourites
Actors	User (initiator)
Type	Primary
Description	The user clicks the Browse Favourites button on the main screen of the system. Any favourite shows will be displayed here.
Use Case	View Settings
Actors	User (initiator)
Type	Primary
Description	The user clicks the Settings button on the main screen of the system. All settings are visible here.

Figure 10: Primary Use Cases

Secondary Use Cases

Use Case	View TV Show Information
Actors	User
Type	Secondary
Included in	**Browse Favourites**, **Search TV Show**, **Browse EPG**
Description	The user can view information on a selected TV show, this includes the title, description and image.
Use Case	Favourite Show
Actors	User
Type	Secondary
Extends	**View TV Show Information**
Description	The user can favourite a selected TV show.
Use Case	View Episode Information
Actors	User
Type	Secondary
Included in	**Browse EPG**
Description	The user can view information on a selected TV episode, this includes the title, description and image.
Use Case	Set Reminder
Actors	User
Type	Secondary
Extends	**View Episode Information**
Description	The user can set a reminder for a selected TV episode.
Use Case	Change Data Sources
Actors	User
Type	Secondary
Extends	**View Settings**
Description	The user can change the data sources for the API.
Use Case	Remove Reminder
Actors	User
Type	Secondary
Extends	**View Settings**
Description	The user can remove a reminder for a selected TV episode.

Figure 11: Secondary Use Cases

Test Cases

Testing is critical to ensure the system runs as it should. Although we can debug while coding, not all bugs will be found by using this method alone. It is important that the system is tested thoroughly to discover any underlying bugs that may go unnoticed during normal use. This system will also have a GUI which involves human input, therefore, this must also be tested thoroughly. The tables below outline all of the tests I plan to carry out on the system. If the test fails, adequate adjustments will be made and the system will be retested until all tests pass.

Test ID	Req ID	Scenario	Test Steps	Expected Result
001	101	Data can be retrieved	Temporarily print result of JSON requests	JSON object will be printed for each request made
002	101	Failed JSON request	Temporarily edit request to return an error	JSON exception should be caught without crashing system
003	102	Parse JSON object	Temporarily print result of parsing JSON requests	Information such as Episode Name, Series Name will be printed
004	102	Parse incompatible JSON object	Pass a random JSON object into parser	JSON exception should be caught without crashing system
005	103	Test GUI	Click 'Search TV shows'	Search TV shows activity starts
006			Click 'Electronic Program Guide'	EPG activity starts
007			Click 'Settings'	Settings activity starts
008	101 102 103 104 105 201	Test TV show search	Type in 'Dr' in search box	List of results will appear
			Click 'Doctor Who' from results	Display Series activity starts with Dr Who as the series
			Wait for Display Series to load	Information for Dr Who will load and be displayed

Figure 12: Table of Test Cases

Test ID	Req ID	Scenario	Test Steps	Expected Result
009	101 102 103 104 105 106	Test Electronic Program Guide functionality	Click 'Electronic Program Guide'	EPG activity starts
			Select channel (BBC 1) from drop down list	Loading animation starts
				Data is loaded and displayed
				Loading animation stops
			Click 'Episode Info' button	Display Show activity starts with selected show
				Episode information is displayed
			Click 'Series Info' button	Display Series activity starts with selected series
				Series information is displayed
			Click 'Stream' button	External link for the selected channel (BBC 1) will open using the device's default browser
			Select '24' from the time limit drop down list	A toast notification will alert the user it has been changed
			Pull down on the list view, invoking a refresh	Loading animation starts
				Data is loaded and displayed for next 24 hours
				Loading animation stops
010	107	Test application runs on Android	Use emulator to run application (Version 5.0)	Application will compile and run
			Use my LG G3 (Version 6.1.1)	

Figure 13: Table of Test Cases

Test ID	Req ID	Scenario	Test Steps	Expected Result
011	202	Test data sources can be changed	Click 'Settings' from main menu	Settings activity will open
			Select only Press Association from checkboxes	Toast will appear listing the sources currently selected
				Global API link will change to only include data from Press Association
012	203 202	Test built-in user guide works Test data sources can be changed	Start application	Main menu will appear
				Welcome popup will appear with descriptions for Search TV and EPG
			Click 'Search TV Shows'	Search TV shows activity starts
				Search bar will have a tooltip describing what it does
			Click 'Electronic Program Guide' Click 'Settings' from main menu	EPG activity starts
				Channel and Time drop down lists labeled with self-explanatory labels will appear
				Self-explanatory icons appear for Episode & Series information Icon appears for Stream
				Settings activity will open

Figure 14: Table of Test Cases

Test ID	Req ID	Scenario	Test Steps	Expected Result
013	301	Test bookmark functionality	Access Display Series activity either via EPG or Search TV Shows	Display Series activity starts
			Click 'Bookmark' button	Series ID is saved to the device
				Toast alerts user that the series has been added to favourites
			Click 'Favourites' button from main menu	List of favourites will load showing series just added
014	302	Test reminder functionality	Click 'Electronic Program Guide'	EPG activity starts
			Click 'Reminder' for the first show in the list	Toast alerts user that the show is already airing
			Click 'Reminder' for the second show	Toast alerts user that a reminder has been set for the time that the show will air
				A notification is displayed when the show is about to air

Figure 15: Table of Test Cases

Pseudo Code

This is the first time I have attempted to create a fully functional Android application. A lot of my efforts have gone into learning and understanding how Android works and how it extends the standard Java language. Therefore, it is important to fully understand what is logically required for each system requirement before attempting to implement it for Android.

API Access

Firstly, it is essential that the system can send and receive an API call to Metabroadcast. This method should return a custom class derived from the results from the API call.

```
APICall (String URL)                    //This method sends an HTTP Request with the supplied URL
        Create HTTP Client with URL
        URL = checkPrefs(URL)           //This method is linked to the settings section
        Send HTTP Request
        Receive JSONObject              //The result from the API is a JSON Object
        For each Show in JSONObject     //Iterate each Show in the results
                New Show = Show(JSONObject[i])   //Parse JSON into a custom class
        Return Shows                    //Return the list of Show classes
```

JSON Parsing

Once the API request has been received successfully, the results need to be parsed into a data structure that allows for easy use of information. For this, a custom Show class will be used. This enables the system to have a consistent method to represent a TV show throughout the system; especially useful when displaying information onscreen.

```
Show (JSONObject Results)               //This class parses the resulting JSON Object into a class
        String title, description, ID,  //Each Show object contains these variables and more
        episode_number, series_number
        String start_time, end_time
        For each object in Results      //Add each individual result from the JSON Object
                title = object.title
                description = object.description
                ID = object.ID
                episode_number = object.episode_number
                series_number = object.series_number
                …
                …
```

Search TV Shows

Now that there are methods to make an API call and return a Show class, it is necessary to receive user input to create the query the API call contains. For this, a searchShows method will be used to take the user's input and return a list of results to select from.

```
searchShows ()                          //This method lets users search shows using a query
        String query = user input       //fetch query from a text box user types in
        String URL = www.metabroadcast.co.uk/query   //Build the URL with the query
        Results = APICall(URL)          //Run the query and return the list of Shows found

        For each Show in Results        //Iterate through each show
                Add Results[i] to ListView      //Add the Show to the on screen list
```

Series Information

Now that the user has a list of shows to select, it is necessary upon selecting a show to fetch and display information it to the user. Selecting a show will be handled by Android's onClickListener:

```
onClickListener ()
        Get Show.ID from List           //Get the ID for the selected show
        displaySeries(Show.ID)          //and pass it to displaySeries
```

Once the show's ID has been fetched, it needs to be used to make a new API call to receive information about it. This will be achieved by using a method called displaySeries which takes the show's ID and outputs the relevant information.

```
displaySeries (Show.ID)                         //This class displays information about a specific show
        String URL = www.metabroadcast.co.uk/show=Show.ID //Build a query using the show's ID
        Result = APICall(URL)   //Pass it through APICall and return the one result
        For each item in Result  //For each piece of info in the Show Class
                Display Result.description      //Display it on the screen (Title, Description, Image etc.)
                Display Result.title
```

Episode Information

Once the user selects an episode to view information for, it's necessary to make a new API call to receive information about it. This will be achieved using a method called displayEpisode which takes the show's Episode ID and outputs the relevant information.

```
displayEpisode (Show.EpisodeID)          //This class displays information about a specific episode
        String URL = Episodewww.metabroadcast.co.uk/episode=Show.ID
        Result = APICall(URL)     //Pass it through APICall and return the one result
        For each item in Result  //For each piece of info in the Show Class
                Display Result.description          //Display it on the screen (Title, Description, Image etc.)
                Display Result.title
```

Electronic Program Guide

Another requirement of the system is to provide the user with an EPG similar to that of a TV.

```
mainEPG ()                              //This method fetches and displays shows that are on
        For each Channel in Channels    //Do this for each channel there is
          String URL = www.metabroadcast.co.uk/schedule=channel     //Build a schedule query
          Results = APICall(URL)                                    //Pass it through APICall and return the shows
          For each item in Results                                  //For each Show in the schedule
            Add Result[i] to list of Shows                          //Add it to the on screen list
```

Now that there is a list of shows for each channel it's necessary for the user to be able to interact with the results. The user must be able to view information about that series and individual episode. Therefore, the activity that shows the EPG should also have an onClickListener for series information, identical to display series information and one for episode information that takes the ID of the Episode:

```
onClickListener ()
        Get Show.EpisodeID from List                    //Get the ID for the selected episode
        displayEpisode(Show.EpisodeID)                  //and pass it to displaySeries
```

Favourites

Users should be able to favourite a TV show when using the system. This option will be available on the displaySeries screen by using a start icon. This can be seen in the design here. Favouriting a show will be achieved using an onClickListener:

```
onClickListener ()
        Get Show.ID & Name from List        //Get the ID for the selected show
        addFavourite(Show.ID, Name)         //and pass it to addFavourite
```

Once the ID has been fetched, it is passed to addFavourite. I plan to save a txt file with the ID and Name of shows that the user adds as favourites:

```
addFavourite (Show.ID, Name)
        Load favourite.txt                  //Load the favourite txt file
        add Show.ID & Name to favourite.txt //Add the ID and name to the list
        Write to file favourite.txt         //Save file
```

When the user wishes to view their favourites from the home menu, a method will load the favourite.txt file from the device and display the name of each show in the list.

```
viewFavourites ()
        Load favourite.txt                  //Load the favourite txt file
        For each Show in file               //Iterate the file for each show
                Add Show.Name to the list   //Add the name of the show to the on screen list
```

Once the favourites have been displayed, the user should be able to select the show to view information about it; another onClickListener identical to display series information will be used. Also, the user should be able to remove a show, this will be achieved with a deleteFavourite method accessed with a button as in the design:

```
deleteFavourite (Name)
        Load favourite.txt                  //Load the favourite txt file
        If file has Name                    //If the show exists in the file
                remove Show.ID & Name from file //Remove the ID and name from the list
        Write to file favourite.txt         //Save file
```

Settings

In order to enable users to select which data sources to access data from, a simple setting page will allow them to select and deselect individual data sources. This can be implemented using Android's native preferences handler. When the APICall method is called, it could open this preference file and adjust the URL accordingly to add or remove data sources:

```
checkPrefs (URL)
        Load preferences                    //Load the preferences file
        For each selected data source       //For every data source selected
                URL + data source           //Include it in the URL
        Return URL                          //Return the URL
```

Reminder

To allow users to set a reminder for a given show. An alarm feature needs to be implemented. Android has its own native AlarmManager object (Google[2] 2016) that can be used. The device can then be woken from a sleeping state to alert the user of an alarm. My system can take advantage of this to alert the user when a TV show is on.

```
setReminder (Show ID, Show Time)
        Set alarm at Show Time with Show ID          //Set the alarm and save the ID to the alarm
```

```
alarm ()                                             //Called when alarm goes off
        Show notification with Show ID               //Show the notification and show name
```

```
cancelAlarm ()                                       //Called to cancel alarm
        Select reminder from list in settings
```

Hand-drawn Designs

In order to ensure usability, it is important to make the system as user-friendly as possible. Therefore, I have started to design the GUI from hand-drawn designs. This method is ideal for rapid prototyping as they are quick to create. However, hand-drawn designs do little to represent limitations in Android's native GUI elements such as buttons, checkboxes and textboxes so it is important to further develop these designs into computerised versions.

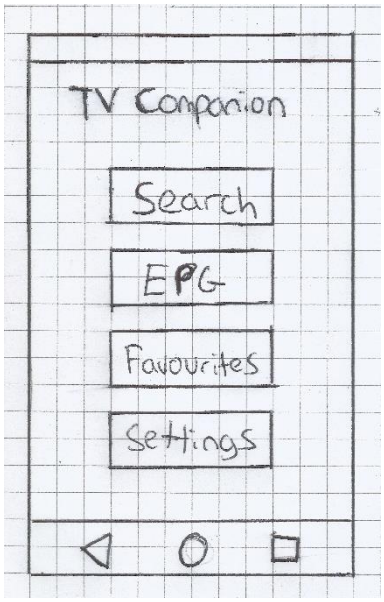

Figure 19: Hand-drawn Main Screen

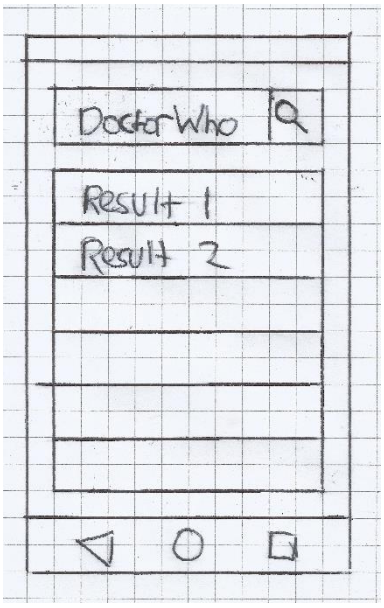

Figure 18: Hand-drawn Search TV Shows

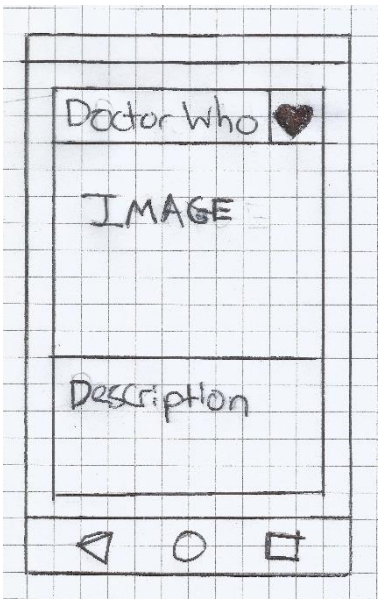

Figure 17: Hand-drawn Show TV Information

Above are the initial designs for the main screen, search TV shows and display TV show information. I have tried to keep the design as clean and simple as possible following Nielsen's Ten Usability Heuristics (Nielsen 1995); mainly of aesthetic and minimalist design, meaning the system should not contain any information that is irrelevant or unnecessary and consistency and standards, meaning the system should have a consistent design and a consistent use of language.

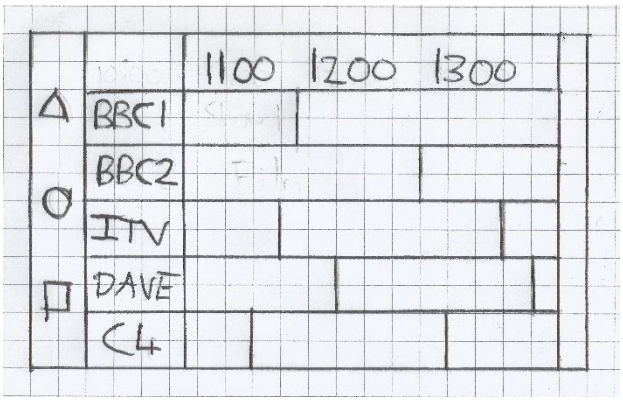

Figure 16: Hand-drawn EPG

This is the design for the EPG. Again, following a minimalist approach I have included everything that a regular EPG includes. Each channel has its own row with shows in each column relative to the table header containing the time that the shows are on for. Clicking on a TV show would then open the show episode information page.

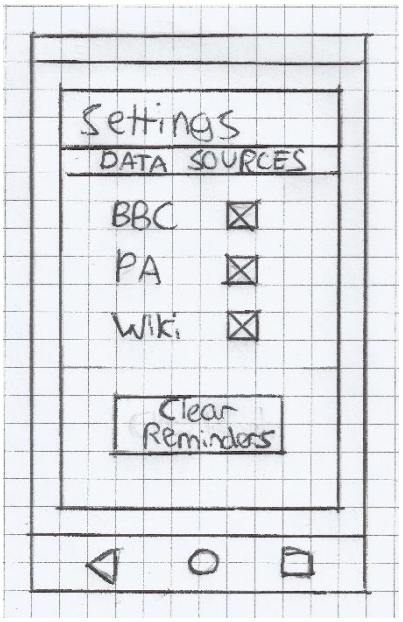

Figure 20: Hand-drawn Settings

Figure 21: Hand-drawn Episode Information

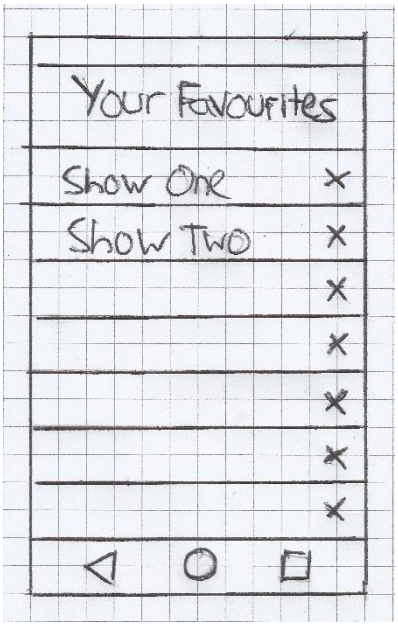

Figure 22: Hand-drawn Favourites

Above are the initial designs for episode information, favourites and settings. Following a similar layout to the rest of the system, I aim to make it as easy to use as possible by using self-explanatory labels and buttons and using titles to describe what that section does.

Wireframe Design

After producing initial designs, I used an online tool called proto.io (proto.io 2016) to create wireframe designs that follow android's UI more closely. As I am a keen user of Android, the designs did not have to be altered much being familiar with the layout of Android applications in general.

Figure 28: Wireframe Main Screen

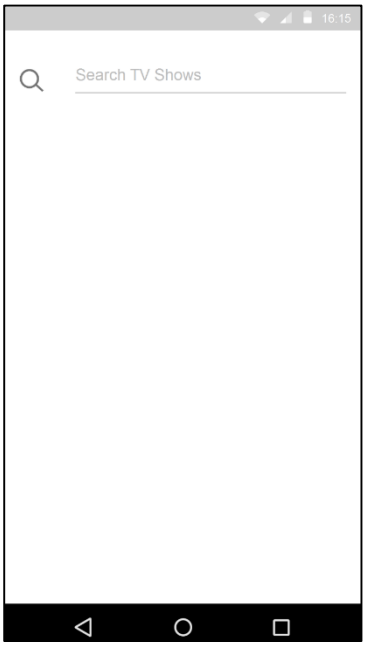

Figure 25: Wireframe Search TV Shows

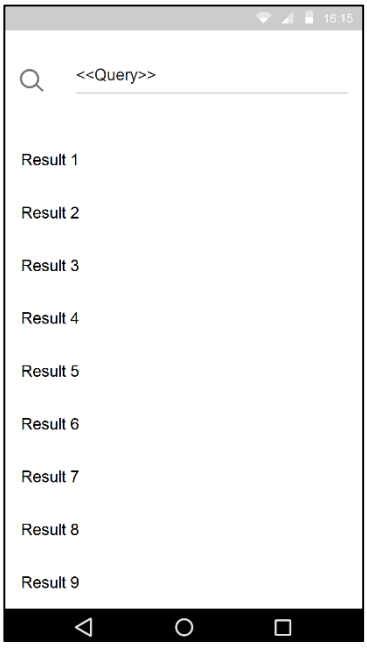

Figure 26: Wireframe Search TV Shows

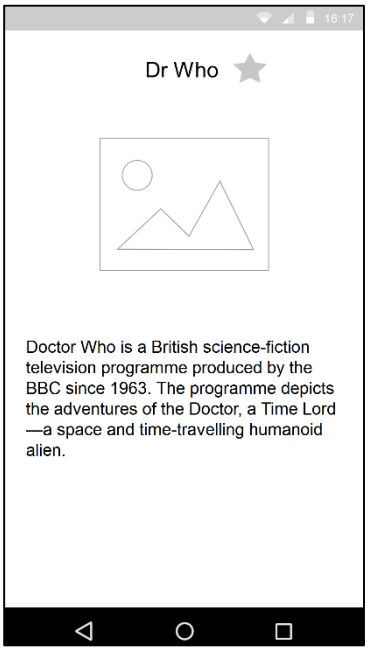

Figure 24: Wireframe Series Information

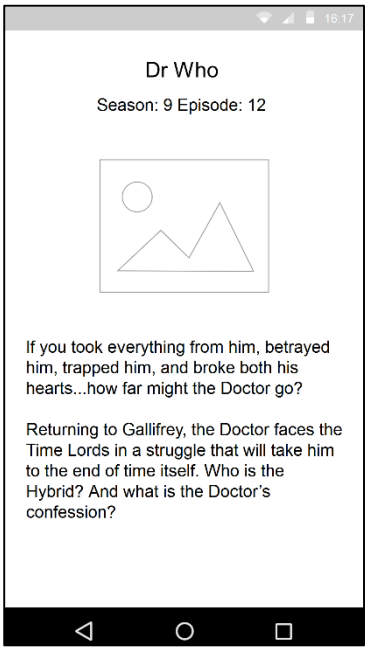

Figure 23: Wireframe Episode Information

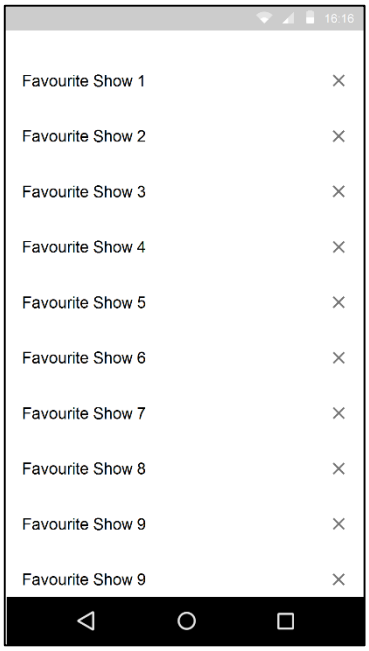

Figure 27: Wireframe Favourites

The only two major changes to the design have been to the settings and EPG. Realising that users may want to have multiple reminders set, only being able to clear all of them using one button was a bad design flaw. Instead, I have decided that there will be a list, similar to that of the favourites feature. The list will have a cross icon to allow users to individually delete reminders.

The EPG has had a drastic change in design. The main reason why I decided to do this was the fact that implementing a classic EPG design would require implementing a complex dynamically rendered nested table in which each TV show took up a custom width based on its duration. As I felt the time needed to invest in learning how to code that feature I decided it would be best to focus on ensuring the data could be retrieved in the first place. The EPG now has a row for each TV show that displayed the time, title and buttons to relevant tasks that extend it

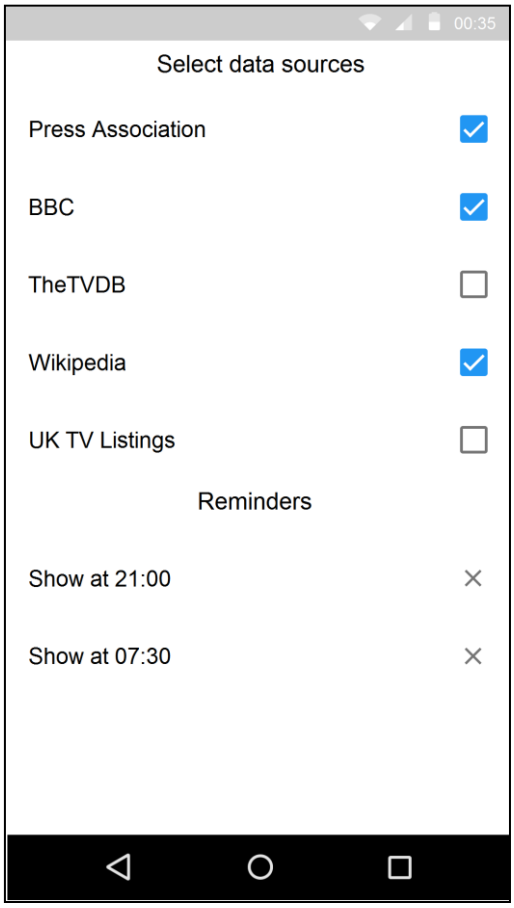

Figure 29: Wireframe Settings

Figure 30: Wireframe EPG

Branding

Icon

In most smartphone applications, users expect a high-quality design as much as solid functionality. Therefore, it is important to consider the brand the system will convey to its users. A consistent approach to the theme will help users build a familiarity to the system and its brand. Another method to give the system a unique brand is to have an icon. This icon will be the image the user sees on their phone along with the description. In Figure 31, it is clear, without needing to read the label which application serves as an email client. These simple and self-explanatory icons subliminally define a standard amongst new and experienced users alike. By default, most Android smartphones do not come with any pre-installed TV related applications and as the system is geared towards TV it makes sense to create a logo that resembles a TV. A simple TV logo would help users instantly understand the purpose of that application in their app drawer, perfect if they are watching TV at the same time of wanting to use the system. Android currently uses its own material theme, based on simplicity and flat icons.

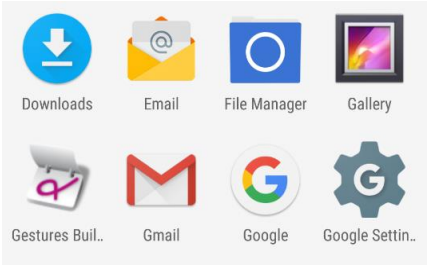

Figure 31: Sample icons from Android Studio's Emulator App Drawer

After browsing the Google Play Store, it is apparent that there is a huge variance in applications related to TV; there is also a noticeable difference in their icons. Even in Figure 32 all of the icons for the ten applications listed are all unique enough to tell apart. Despite their differences, most of the icons do a good enough job of representing their respective app's purpose.

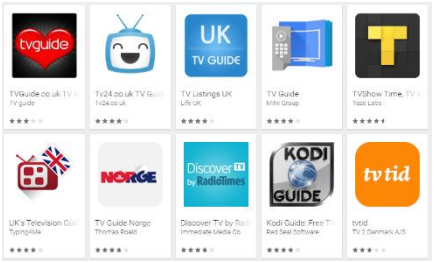

Figure 32: Sample of TV related apps (Play.google.com[5], 2016)

Design

Figure 33: Icon design development

The design for the system's icon aims to follow the material theme of Android to help it integrate with the look of the device's user interface. A simple TV with a phone inside suggests the paralleled use of the two devices. The icon is also unique such that if a user has other TV applications installed that their icons do not clash.

I created the intial design by hand and used it to produce an initial digital design using software called Inkscape (Inkscape 2016). However, as Android supports screens of varying resolutions and densities it is important to create a set of icons to accommodate them all. I utilised an open source program called Android Asset Studio (Nurik 2016) which allowed me to upload my icon and generate a compressed folder with icons ready to import into the Android Studio project. The tool also offered customisation to help the icon follow Google's Material theme.

Name

Another important method for promoting the system's brand is the name. The name of the system should be self-explanatory with the aim that a potential user should be able to understand the system's purpose by its name alone. For that reason, I have decided to call the system 'TV Companion'.

Figure 34 demonstrates TV Companion's appearance on the home screen of a generic Android device. The minimalist style helps it blend in with the rest of the screen and promotes an elegant professional brand.

Figure 34: Home screen from Android Studio's Emulator App Drawer

Implementation

Android Application

When implementing the solution using Android Studio, adaptations had to be made to the original pseudo code design. As Android extends the Java library, a large amount of work has been made to extend my knowledge from Java to Android.

API

The Metabroadcast API provides a Java based client to directly query the API server. However, this client has no documentation, has infrequent updates and has changed considerably since its release meaning there is no relevant information available. Instead of spending time trying to understand a bespoke Java client for a single API, I felt it would be more beneficial to my learning to learn a more generic, HTTP API approach where the skillset gained can be transferred to other projects calling for the use of an API.

The API works by sending a simple URL request in a specific format. If the request is successful, a JSON object will always be returned. This is useful as the system can be developed to solely deal with JSON objects rather than a mixture of objects and arrays.

The structure for a URL request is as follows:

Base URL: https://atalas.metabroadcast.com

- For schedules: /Schedules/**CHANNEL ID**.json?from=**START TIME**&to=**END TIME**
- For content: /content/**ID**.json?type=item
- For queries /content.json?limit=20&annotations=description&q=**QUERY**

End URL: &source=SOURCES&key=**UNIQUE API KEY**

An example of the URL used to fetch schedules for a channel with an ID of ID:

```
https://atalas.metabroadcast.com/4/schedules/"+ID+".json?from=now&to=now.plus."+until+"h&source=pressassociation.com&annotations=channel&key=
```

The resulting response contains all the information from the request. Each JSON Object is formatted in the same way for each type of request making parsing very simple to achieve:

Queries return a JSON Object containing an array of results which contains a JSON Object each where the ID and title can be retrieved and displayed as a list of results. When a user selects a result, the ID can simply be used in a content API request.

Content returns a JSON Object containing JSON Objects for each piece of information. This information is consistent for each TV show or film making parsing simple.

Schedule returns a JSON Object containing an array of JSON Objects for each TV show that is being aired. This information also includes the start and end time for its scheduled airtime on that given channel.

It has also been noted that changing the source section in the URL has no effect on the results returned. Therefore, I feel that requirement 202, a way for users to select data sources, should be

categorised as a 'Could Have' requirement as the scope of the project does not cover the time required to research into changing data sources.

Asynchronous HTTP Client

In order for my system to make an HTTP request, Android enforces all networking tasks to be performed in a separate thread to the main UI thread. This ensures the GUI doesn't freeze while a network task is running and also prevents timeouts from stopping the user from interacting with their device.

Originally I had researched and discovered Google's Volley (Google 2016), an HTTP library that allows for Android apps to make HTTP requests in the background. However, I was having difficulty understanding how to extract the results from the request thread into the main UI thread in order to display the results on screen. The documentation for Volley proved to be difficult to understand and lacked examples relevant to my situation. This became a major limitation to the development of my solution as I was unable to use the API call results in the main UI thread.

To get around this, I utilized James Smith's Android Asynchronous HTTP Client (Smith 2016). This library allows HTTP requests to be made outside of the system's main UI thread without the need to manually program an HTTP client. I adapted this library to suit the needs of the Metabroadcast API by including methods for each type of API request.

Method	**Parameters**	
getContentSearch	String Query	RequestListener listener

```
    String ApiURL =
"content.json?limit=20&annotations=description&q="+Query+"&key=";
    System.out.println(ApiURL);

    JSONRequest.get(ApiURL, null, new JsonHttpResponseHandler() {
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject
response) {
            listener.onSuccess(statusCode, headers, response);
        }
    });
```

getContentSearch is responsible for building a query to fetch a JSON object containing all of the results from the query inputted by the user. In order to make the request, JSONRequest is used, a method from the Android Asynchronous HTTP Client library. JSONRequest's onSuccess method is then overridden to be able to take control of the returned results.

Method	Parameters	
getContent	String ID	RequestListener listener

```
String ApiURL =
"content/"+ID+".json?type=item&source=pressassociation.com&limit=2&offset=0&annot
ations=brand_summary,broadcasts,channel,description,images,people&key=";
    System.out.println(ApiURL);
    JSONRequest.get(ApiURL, null, new JsonHttpResponseHandler() {
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject
response) {

            obj = response;
            listener.onSuccess(statusCode, headers, response);

        }

    });

}
```

getContent is responsible for building a query to fetch a JSON object containing the information for a given unique ID. As per Metabroadcast's API, the ID used in this query can be from either a series or an individual episode from a series, this saves any duplicate code being used as this method works for both.

Show Class

As every show follows a similar format, I decided to use a custom Show object to best represent the data instead of working from the JSON Object directly within the EPG and displayEpisode sections of the system. By using a custom object, the data can be accessed and modified in any way I choose to without having a direct impact on the rest of the system. For example, if I wanted to format the series title to include the episode name I could do so from within the Show class. Attempting to do this sort of operation within the JSON Object is not best practice and ultimately creates limitations to the system should anything need to change.

As the airing times are in GMT and it is currently GMT+1, the times were an hour behind. To remedy this problem, I implemented a formatTime method in the Show class to convert the airing time String into the correct format and time. The input String is formatted "2014-09-26T14:45:00.000Z" so the time section is extracted and a SimpleDateFormat is applied and cast to a Date datatype. To increment the time by an hour, a Calendar is created and the time added to it; Calendar has a method to add an hour. The Calendar returns the new time and is returned.

I decided not to use this class to represent data with displaySeries as the data used in this method is slightly different from an 'Episode' object returned from the API. As the data is not accessed anywhere else, it is a simple case of retrieving a single JSON Object and displaying the data.

Please see the source for Show.java here.

Home Screen

The home screen follows the same design set out throughout the design stage. The buttons are all self-explanatory, aiming to achieve learnability and usability. They are also of a uniform size to comply with a consistent theme throughout.

The code works by simply creating a new intent upon the onClick method of the button. Favourites extends that by featuring a check to see if the user has any saved favourites. If they do not, the button does not do anything and displays a toast alerting them of the lack of favourites:

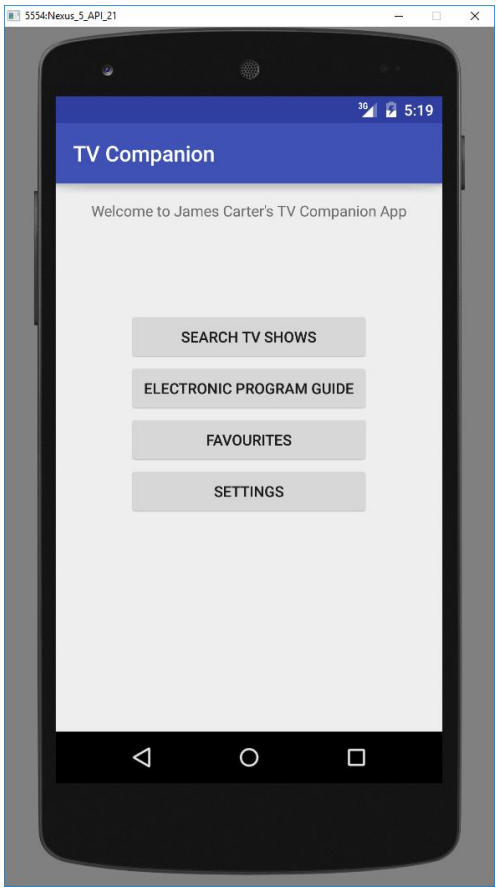

Figure 35: Main Screen

```
public void favourites(View view){

    SharedPreferences preferences =
getSharedPreferences("com.example.c1319936.tv
companion", 0);
    if(preferences.contains("Favs")){
        Intent i = new Intent(this,
Favourites.class);
        startActivity(i);
    }
    else {
        Toast.makeText(this, "You have no
favourites", Toast.LENGTH_SHORT).show();
    }
```

Search TV Shows

The search for TV shows section incorporates a self-explanatory search bar which the user can type a query. Upon completing the query, an API call is made with the query text and results are returned on the same screen using a custom List View. Clicking one of these shows will open display series.

Populating the ListView with results is achieved by using an Array Adapter; this works by mapping an ArrayList of results to ListView elements. An onClickListener is then responsible for retrieving the ID of the selected show that the user clicks on. This ID is then sent over to displaySeries within an intent extra:

```
ListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> adapter, View v, int position, long
id) {

        //fetch the show name and ID at the current position
        String selectedFromList = (String)
(ListView.getItemAtPosition(position));
        String ID = (String)(resultID.get(position));

        //new intent to displaySeries
        Intent i = new Intent(searchShows.this, displaySeries.class);
        //add the show ID to the intent
        i.putExtra("show", ID);
        //send the intent
        startActivity(i);
```

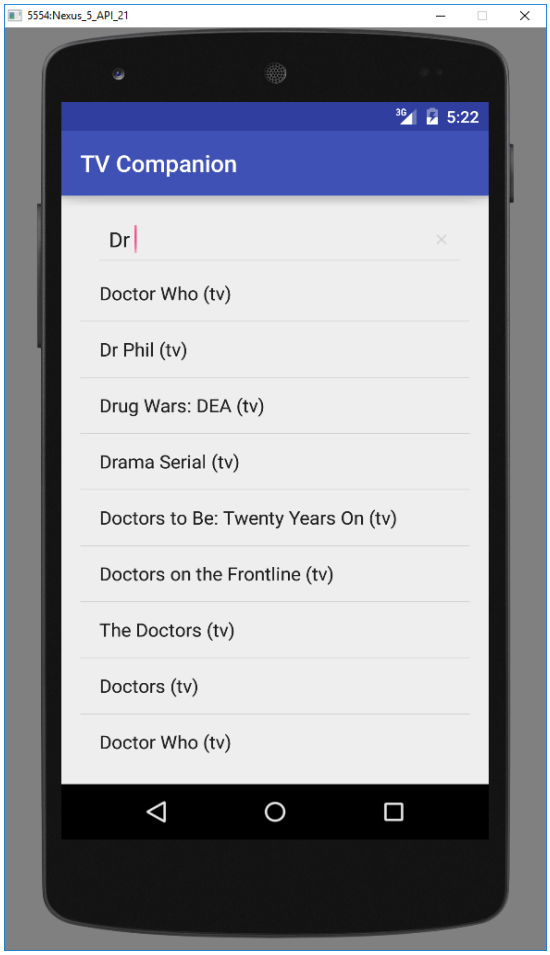

Figure 36: Search TV Shows

Displaying Series Information

In order to work with the results, a listener is implemented in the class that made the original HTTP request. This means that when the request has been successfully returned, the main UI thread can use the data to populate onscreen objects.

An example of how this is used in the main thread can be demonstrated in the displaySeries class.

Method

displaySeries

```
Content.getContent(ID, new RequestListener() {
    @Override
    public void onSuccess(int statusCode, Header[] headers, JSONObject response)
{

        try {
          //populate variables with series information
          title = response.getJSONObject("brand").getString("title");
          type = response.getJSONObject("brand").getString("specialization");
          description = response.getJSONObject("brand").getString("description");

            try {
                imageURL =
response.getJSONObject("brand").getJSONArray("images").getJSONObject(0).getString
("uri");
            } catch (JSONException e) {
                imageURL = "null";
            }

            //if there is no value retrieved for specialization
            if (type.equals("null")) {
                //then set type to empty
                type = "";
            }

            //if there is a URL for an image for the episode
            if (!imageURL.equals("null")) {
                //then use loadImageAsync task to add image to an Image View
                ImageView imgView = (ImageView) findViewById(R.id.imageView);
                new loadImageAsync(imageURL, imgView).execute();
            }

            //if there is a description
            if (!description.equals("null")) {
                //then add the description
                TextView txtDesc = (TextView) findViewById(R.id.textDescription);
                txtDesc.setText(description);
            }

            //if there is a title
            if (!title.equals("null")) {
                //set a title TextView to hold the title of the series
                TextView txtTitle = (TextView) findViewById(R.id.textTitle);
                txtTitle.setText(title);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }

    }

});
```

The onSuccess method is overridden in the main thread, allowing me to populate TextViews and an ImageView with the relevant information. I have ensured that appropriate error checking is in place to prevent any results that do not contain certain information from causing JSONObject null pointer errors resulting in the system crashing.

The display series section can be opened from either search shows or by clicking the series information button from the EPG. This page follows a simple layout to show the title, image and description of the show itself. The quality of the description is dependent on the returned results from the API. Clicking the favourite icon will add the show to the user's favourites list.

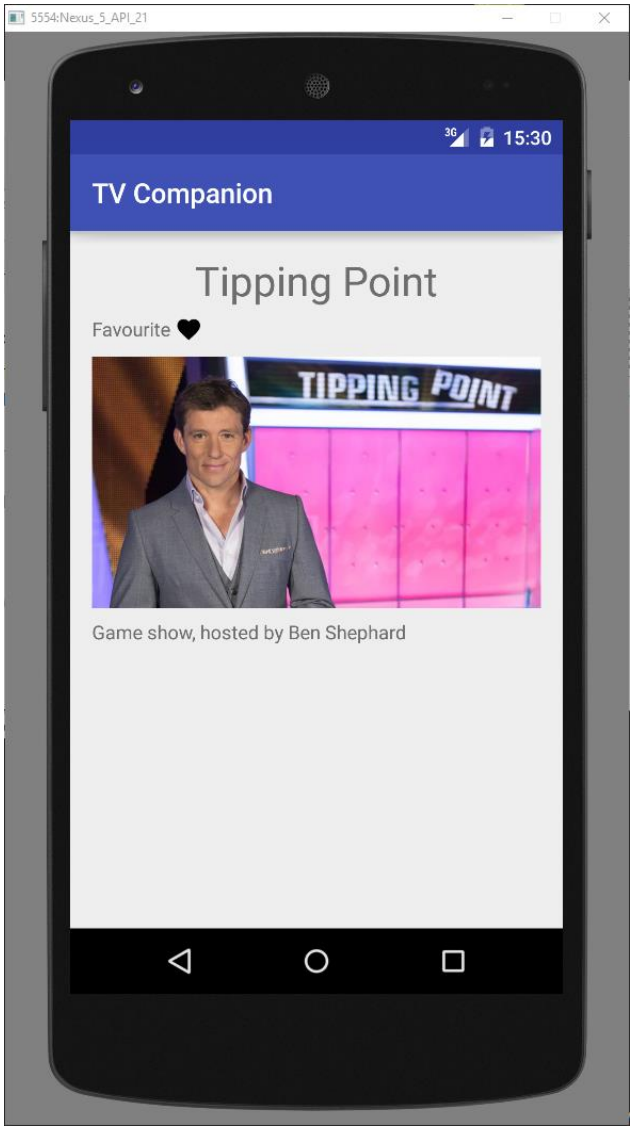

Figure 37: Series Information

Displaying Episode Information

Displaying information for individual episodes follows much the same approach using an episode ID instead of a series ID. As the data follows the same format as the EPG, we can take advantage of the Show object to represent the data. In order to display the correct

The display episode is similar to display series. It includes the title, image, description and adds the series/episode numbers and time due to be broadcast. The minimalist design aims to make it easier for the user to read the information displayed to them. The image is used to pad out the screen and break up the text.

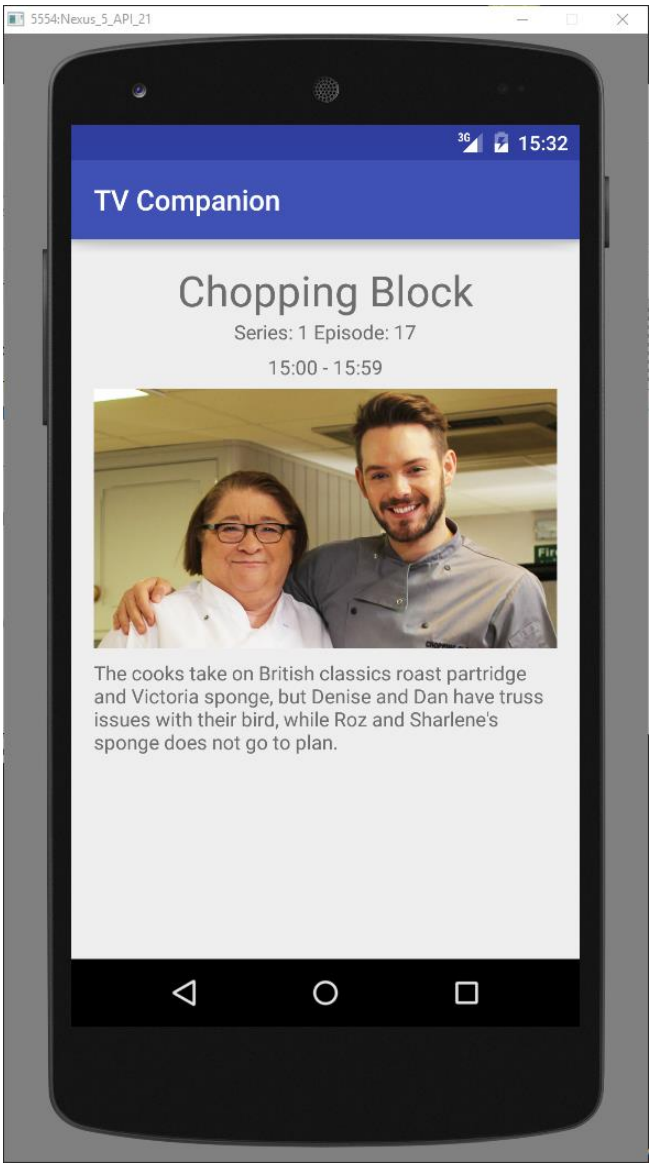

Figure 38: Episode Information

Electronic Program Guide

The EPG formed the most difficult section to implement as it requires multiple queries to made that depend on the results of an initial query. I attempted to use the same Android Asynchronous HTTP Client used previously in getContent and getContentSearch; however, as the data has to be used within onSuccess, it is impossible to create a For loop within the method as onSuccess has already been overridden. Results from the first query would not be returned in time for them to be used in queries that depend on them. This was a limitation of the library and the way I had set it up.

To solve this problem, I researched Google's documentation on Android and implemented an AsyncTask (Google[2] 2016). An AsyncTask enables the system to perform background operations and return the results to the UI thread, similar to overriding onSuccess. The AsyncTask wraps a Google Volley JSON Object Request which is then called from within a thread separate to the UI thread. Once the initial schedule results have been returned, an array of threads equal to the number of results is created and iterated through within the initial thread, returning the results for each scheduled show. The results are added to an array of Show objects which are then passed to displayEPG which runs in the UI thread to display the results. See the full code here.

Another unexpected change had to be made regarding the airing times for shows. The initial schedule query contains the airing times for each show. When running each separate query, the airing times can be different causing incorrect airing times to be displayed. To remedy this, I fetched the correct times from the initial query and append them to the query results for each show.

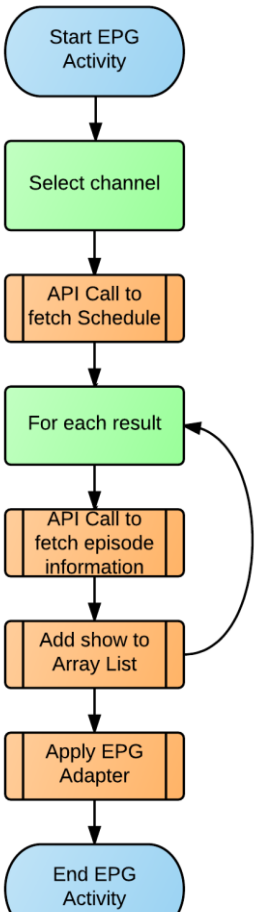

Figure 39: Flow Diagram for EPG

The EPG is by far the busiest screen for users. Aiming to adhere to a minimalist design I have ensured each show is laid out in the same manner, again, with self-explanatory buttons. I incorporated the use of icons for the information sections to save room. The channel and time selection drop down lists are labeled accordingly to allow users to instantly understand what each do.

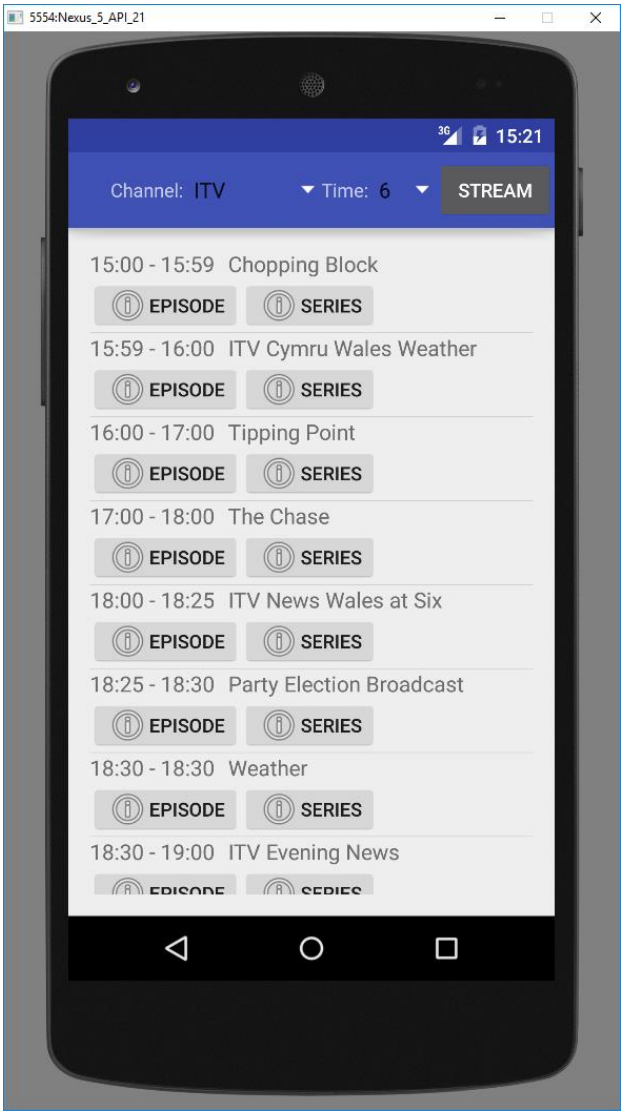

Figure 40: EPG

Feedback is given in the form of onscreen toasts to tell the user if the time limit has changed as well as a refresh icon when data is being loaded.

The drop down list is dynamically generated by using my custom APIDict object. This object works by using a LinkedHashMap that links a key-value pair. The key is the name of the channel i.e. ITV and the value is the unique API ID for the channel. When the user clicks on the channel, mainEPG uses APIDict's getChannel method to return the ID. Adding channels to this object is extremely simple, all one needs to do is add a line of code adding the ID and name of the channel.

I have also implemented a new 'Stream' feature which when the user clicks, opens up the appropriated streaming website for the channel i.e. ITV player. As this was an unplanned feature, I simply appended the link to the unique ID within APIDict and modified the return methods to only return the first four characters as the unique ID is of that size. Although this is not the most elegant way, it saved time by avoiding a redesign of representing shows and their unique ID.

Favourites

Favouriting a show has been adapted to take advantage of Android's native Shared Preferences feature. Every application on a given device has its own Shared Preferences file which can be opened, edited and saved within the code. This feature is extremely useful as there is no need to implement a bespoke file reader and writer.

When the user runs the displaySeries task, if there is a shared preferences file it is opened, if not then a new one is made. When the user clicks the favourite button, a method loads the preferences and extracts the set of data that holds the list and casts it to an ArrayList. From there, the new show is added to the ArrayList, cast back to a set and saved to the preferences file.

When the user runs the favourites task, the preferences files is loaded, cast to an ArrayList and a custom Adapter is used to add the title of each show to a ListView. An example of a favourites list can be found in Figure 41. A check is made in the main menu to see if a preferences file exists. If it isn't, the favourites button is disabled until the user adds a show to their favourites.

The favourites section follows the same layout set in the design stage. I have added a text box with instructions to help users understand how to interact with the data displayed on screen. The user can click on the show and display series will load, showing information about the show; they can also click on the cross icon to remove the show from their list

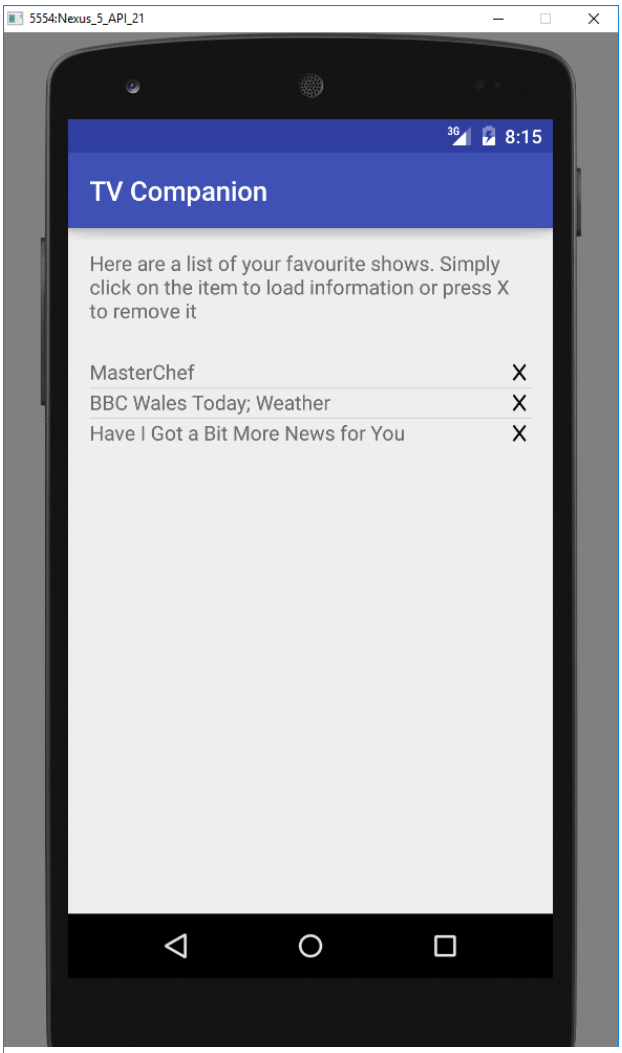

Figure 41: Favourites

Settings

Although settings haven't been implemented, the GUI for it has been created ready for its implementation. A text box briefly explains what the check would boxes do. I removed the reminders ListView as it hasn't been implemented and leaving the object empty caused the system to crash on compile.

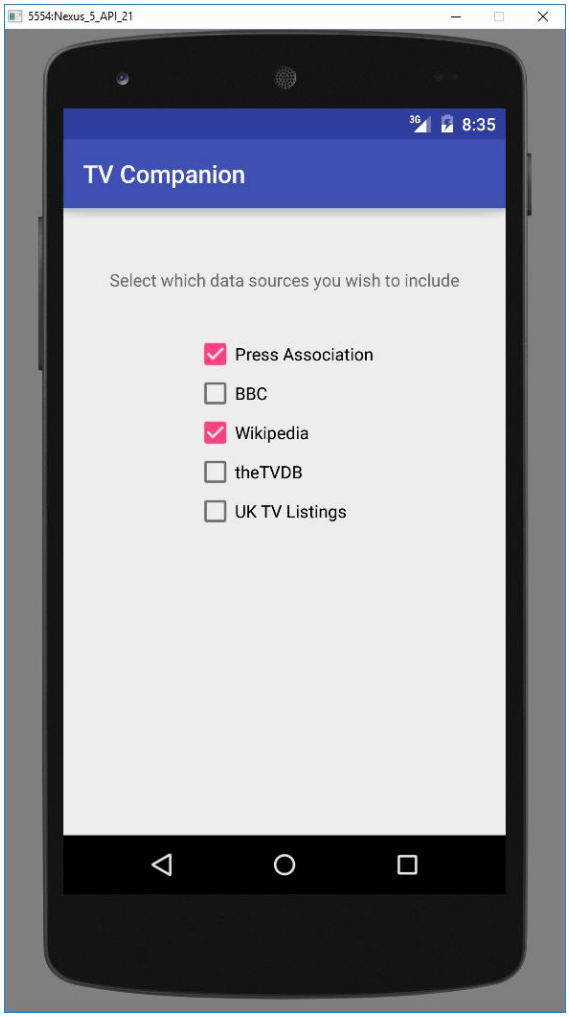

Figure 42: Settings

Class Diagram

This is the UML Class Diagram of the final system. This diagram gives an overall view of the system and how each class interacts with each other. From this view it is clear mainEPG uses a ScheduleThread in order to achieve nested threaded retrieval of the schedule, a novel feature of the system. It also helps to visualise how the RequestListener is used to interface with HTTP requests in order to override the onSuccess method.

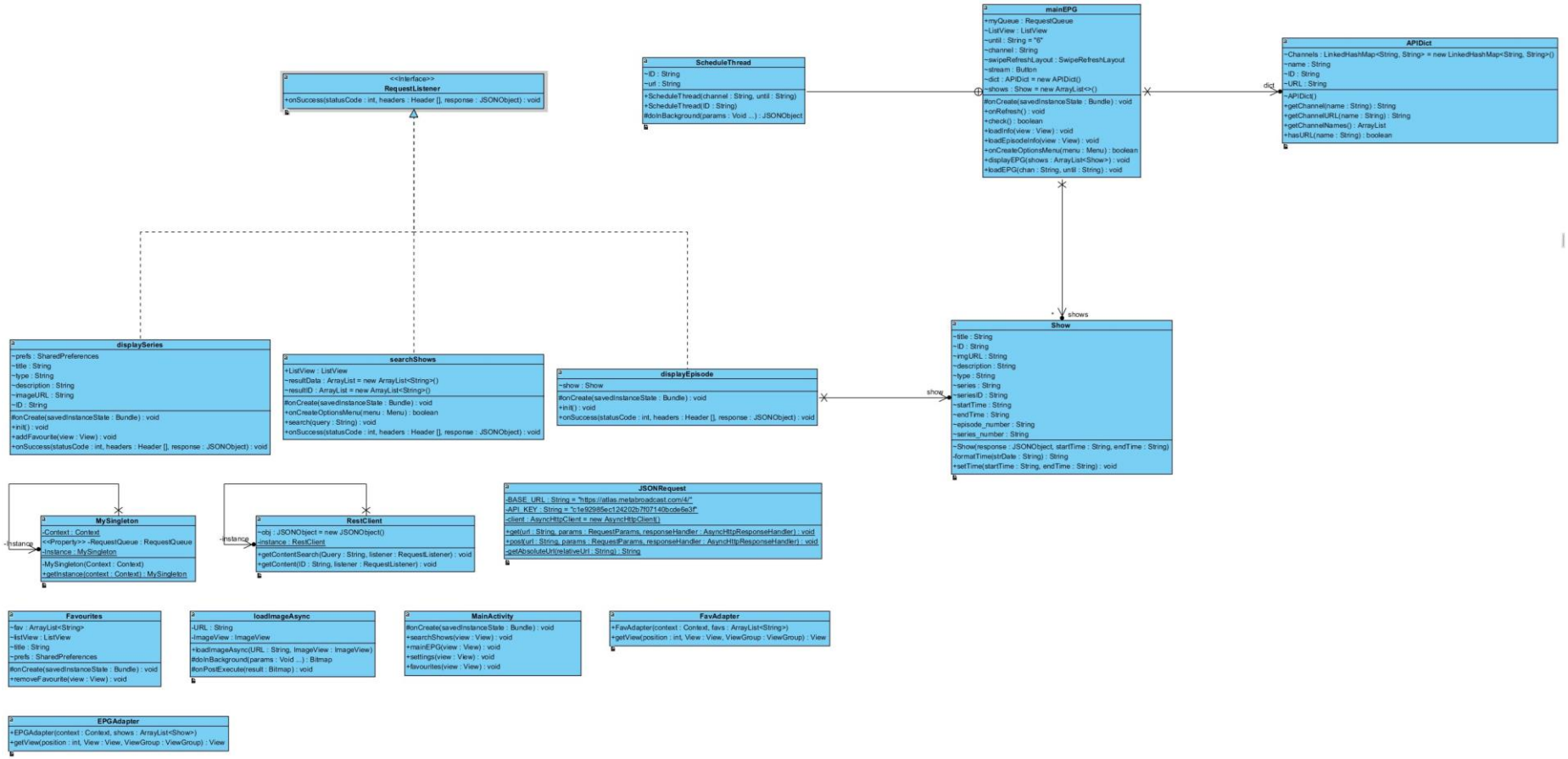

Figure 43: UML Class Diagram of Final System

Testing

Test Cases

After implementation, I proceeded to test the system against the test cases set out in the design stage.

Test ID	Req ID	Scenario	Test Steps	Expected Result	Actual Result	Pass/ Fail
001	101	Data can be retrieved	Temporarily print result of JSON requests	JSON object will be printed for each request made	System successfully printed JSON Object (See Image 1)	Pass
002	101	Failed JSON request	Temporarily edit request to return an error	JSON exception should be caught without crashing system	System crashes (See Image 2)	Fail
003	102	Parse JSON object	Temporarily print result of parsing JSON requests	Information such as Episode Name, Series Name will be printed	System successfully printed Name, ID and Series ID during parsing in Show Class (See Image 3)	Pass
004	102	Parse incompatible JSON object	Pass a random JSON object into parser	JSON exception should be caught without crashing system	System crashes with a similar output as Test 002	Fail
005	103	Test GUI	Click 'Search TV shows'	Search TV shows activity starts	Successfully navigates to searchShows	Pass
006			Click 'Electronic Program Guide'	EPG activity starts	Successfully navigates to mainEPG	Pass
007			Click 'Settings'	Settings activity starts	Successfully navigates to settings	Pass
008	101 102 103 104 105 201	Test TV show search	Type in 'Dr' in search box	List of results will appear	System displays results (See Image 4)	Pass
			Click 'Doctor Who' from results	Display Series activity starts with Dr Who as the series	System successfully navigates to displaySeries, loads and displays information for 'Doctor Who' (See image 5)	Pass
			Wait for Display Series to load	Information for Dr Who will load and be displayed		Pass

Figure 44: Table of Test Cases

Test ID	Req ID	Scenario	Test Steps	Expected Result	Actual Result	Pass/Fail
009	101 102 103 104 105 106	Test Electronic Program Guide functionality	Click 'Electronic Program Guide'	EPG activity starts	EPG activity starts	Pass
			Select channel (BBC 1) from drop down list	Loading animation starts	Loading animation starts	Pass
				Data is loaded and displayed	Data is loaded and displayed (See Image 6)	
				Loading animation stops	Loading animation stops	
			Click 'Episode Info' button	Display Show activity starts with selected show	Display Show activity starts with selected show	Pass
				Episode information is displayed	(See image 7)	Pass
			Click 'Series Info' button	Display Series activity starts with selected series	Display Series activity starts with selected series	Pass
				Series information is displayed	(See image 8)	Pass
			Click 'Stream' button	External link for the selected channel (BBC 1) will open using the device's default browser	Link opens successfully using device's default browser	Pass
			Select '24' from the time limit drop down list	A toast notification will alert the user it has been changed	A toast notification will alert the user it has been changed	Pass
			Pull down on the list view, invoking a refresh	Loading animation starts	Loading animation starts	Pass
				Data is loaded and displayed for next 24 hours	Data is loaded and displayed for next 24 hours	Pass
				Loading animation stops	Loading animation stops	Pass
010	107	Test application runs on Android	Use emulator to run application (Version 5.0)	Application will compile and run	Application compiles and runs	Pass
			Use my LG G3 (Version 6.1.1)		Application compiles and runs	Pass

Figure 45: Table of Test Cases

Test ID	Req ID	Scenario	Test Steps	Expected Result	Actual Result	Pass/Fail
011	202	Test data sources can be changed	Click 'Settings' from main menu	Settings activity will open	Settings activity opens (See here)	Pass
			Select only Press Association from checkboxes	Toast will appear listing the sources currently selected	Settings have not been implemented	Fail
				Global API link will change to only include data from Press Association		Fail
012	203	Test built-in user guide works	Start application	Main menu will appear	Main menu appears	Pass
				Welcome popup will appear with descriptions for Search TV and EPG	The menu shows self-explanatory buttons and a welcome notice at the top	Pass
			Click 'Search TV Shows'	Search TV shows activity starts	Search TV shows activity starts	Pass
				Search bar will have a tooltip describing what it does	'Search for a TV show' appears as a tooltip	Pass
			Click 'Electronic Program Guide'	EPG activity starts	EPG activity starts	Pass
				Channel and Time drop down lists labeled with self-explanatory labels will appear	The two lists appear	Pass
				Self-explanatory icons appear for Episode & Series information Icon appears for Stream	Icons appear with labels	Pass

Figure 46: Table of Test Cases

Test ID	Req ID	Scenario	Test Steps	Expected Result	Actual Result	Pass/Fail
013	301	Test bookmark functionality	Access Display Series activity either via EPG or Search TV Shows	Display Series activity starts	Display Series activity starts	Pass
			Click 'Bookmark' button	Series ID is saved to the device	The series ID and name is saved to the device	Pass
				Toast alerts user that the series has been added to favourites	A toast alerts the user of the show being saved to favourites	Pass
			Click 'Favourites' button from main menu	List of favourites will load showing series just added	This list loads correctly, clicking it will load Display Series	Pass
014	302	Test reminder functionality	Click 'Electronic Program Guide'	EPG activity starts	EPG activity starts	Pass
			Click 'Reminder' for the first show in the list	Toast alerts user that the show is already airing	This has not been implemented	Fail
			Click 'Reminder' for the second show	Toast alerts user that a reminder has been set for the time that the show will air		Fail
				A notification is displayed when the show is about to air		Fail

Figure 47: Table of Test Cases

Questionnaires

Understanding how users feel about the application is very important in evaluating the success of the project. I made a simple survey using https://www.surveymonkey.com, a convenient tool to create and anaylse questionnaires. The questions I chose to ask were solely based around the usage of the application as I wanted to understand how different users rate the ease of us.

Question	Purpose
Have you ever used an application to view TV information?	This question is used to ascertain whether or not the user has ever tried a similar application. If they have previous experience they may find it easier to use
Describe your level of experience using a smartphone	This question is used to understand the experience level of the user. If the user is inexperienced in general, they may find the system harder to use
Were you able to search for a TV show?	These questions aim to understand if the user can use the features of the system. If they can use the features, this can be used to suggest the project has been a success.
Were you able to favourite a TV show?	
Were you able to browse the Electronic Program Guide?	
Were you able to remove a favourite show?	
Were you able to view information for a selected show?	
How easy was the application to use? (0 Impossible - 5 Extremely easy)	This questions aims to understand how the user felt the system was to use. The higher the better.

Figure 48: Table to summarise choice of questions

Results

Six people managed to carry out the questionnaire. Although a small sample size, the people included my Mother (IT & Basic Skills Teacher), my Father (Bought his first smartphone this year), my partner (An average smartphone user), my friend (An average smartphone user), a peer in university (Considered advanced smartphone users) and a colleague in work (An average smartphone user).

The variance in people helps give me an understanding of how different people use and feel about the application I have produced. The results are as follows:

Have you ever used an application to view TV information?	Yes	1
	No	5

Unsurprisingly, not many people have actively used a similar application to view TV information.

Describe your level of experience using a smartphone	Advanced	1
	Intermediate	3
	Beginner	2

The majority of people stated they were intermediate users. This was my target audience set out from the start. Having two beginners helps me understand whether at their level that my application is still easy to use

Were you able to search for a TV show?	Yes	6

	No	0

All users were able to search for a TV show, suggesting that the feature is very easy to use.

Were you able to favourite a TV show?	Yes	6
	No	0

All users were able to favourite a TV show, suggesting that the feature is very easy to use.

Were you able to browse the Electronic Program Guide?	Yes	6
	No	0

All users were able to browse the Electronic Program Guide, suggesting that the feature is very easy to use.

Were you able to remove a favourite show?	Yes	5
	No	1

Only one user was unable to remove a favourite show. The user, surprisingly, claimed to be intermediate. Upon further questioning, the user didn't read the small instructions on the favourite screen so didn't understand that the cross was responsible for removing a show from their favourites.

Were you able to view information for a selected show?	Yes	6
	No	0

All users were able to view information for a TV show, suggesting that the feature is very easy to use.

How easy was the application to use? (0 Impossible - 5 Extremely easy)	0	0
	1	0
	2	0
	3	2
	4	1
	5	3

50% of users agreed that the application was extremely easy to use with the others ranging from 3/5 and 4/5. Overall, it is safe to say that users did not find the system difficult to use.

Test Report

Overall, the system runs very well. There are no bugs related to the flow of the GUI, each button correctly goes to the expected activity. The application runs on all devices tested with no compilation problems at all. The questionnaires have helped prove that the system is easy to navigate and use with 50% of users asked stating it was extremely easy to use. The GUI proved to be mostly self-explanatory.

Searching for TV shows does not cause any errors, nor does browsing the EPG. The main system requirements have no errors.

Favouriting works flawlessly using Android's native Shared Preferences. Testing shows that the show is saved and can be accessed even after closing the application. Favourite shows can also be deleted with no errors. Toasts are used to alert the user when they have added a favourite, removed a favourite and if they try to access favourites without any, a toast to tell them they have no favourites.

However, there are some errors within the system; Parsing JSON Objects that are formatted incorrectly it will crash the system. As this is extremely unlikely to happen, given that the API returns all results in the same format, it does not have a detrimental effect on the system. Similarly, if the API URL is incorrect, the exception is caught but crashes the system as it still tries to parse an empty JSON Object. This bug does not have a severe impact on the system as the API calls are hard coded into the system. There may be a vulnerability in the system if the API call times out due to lack of internet connectivity. This would render the system useless without internet access.

The settings feature has not been implemented, this means that clicking on the data sources through the GUI has no effect on the system. Although this does not have a severe impact on the system, it still failed the test case set out for it. Similarly, the reminder feature has not been implemented resulting in failed test cases for that feature too.

Results and Evaluation

In order to evaluate whether the project has been a success, it is necessary to compare the finished solution against the system requirements. Of course, the more system requirements achieved results in the project being more successful. As system requirements were categorised into three levels of priority, in order for the project to be successful all of the 'Must Have' requirements need to be implemented. 'Should Have' requirements are necessary to gauge how successful the project is and 'Could Have' requirements can be used to assess how the project has exceeded the requirements to be considered a success.

Requirement 101, a method to access information of TV programs has been achieved; The system can access the Metabroadcast API and retrieve information. Requirement 102, to be able to process this information has been achieved too. The system has a completely functional GUI that aims to follow basic Human Computer Interaction goals that were set out right at the beginning during the hand-drawn design stage, fulfilling requirement 103. Users are able to select a TV program from either the EPG or from the list produced when searching for a TV show, fulfilling requirements 104, 106 and 201. Selecting a TV show then results in the user being able to view information for that specific show, fulfilling requirement 105. Finally, the system compiles and runs on Android, as proven during testing.

The system allows users to search for TV shows, fulfilling requirement 201. The system also aims to provide a built-in user guide by using self-explanatory icons, buttons and titles; some more advanced features of the system include a text box to describe how to use them such as the favourites section. As I could not conclude as to why changing data sources in the API URL made no difference to results, I focussed my efforts on implementing requirement 302, allowing users to favourite a TV show. It should also be noted that I introduced a new feature, allowing users to click-through to an external link to stream the selected TV channel.

Considering non-functional requirements is important in evaluating the system's overall success too. The questionnaires and testing suggest that usability has been achieved as users find it easy to use with minimal guidance. Reliability is partially achieved as the system runs error free in normal usage and runs on Android but can still crash due the lack of error handling; This means that robustness has not been fully achieved. The system, I feel, is easy to change if necessary, the code is commented thoroughly throughout with every line described to allow myself or someone else to understand and change, fulfilling maintainability. Performance has been achieved too, with the only loading necessary being when the EPG fetches information using nested threads; this loading cannot be improved and is dependent on the internet connection and speed of the device. Security is a negligible requirement as the system stores no personal information at all. Similar to maintainability, flexibility has been achieved by means of well-documented code and a modular design has been incorporated. For example, changing the EPG section of code would have zero effect on the other functionality.

Future Work

In this section, I will outline future plans to enhance the application.

Firstly, I would ensure an appropriate error handler is implemented to handle all API requests. This error handler would catch any exceptions and save them as a log to the device without crashing the system. The method that called the API call would then receive a callback notifying it that the request had failed and would simply terminate the operation, having no effect on the UI thread. I would also aim to dynamically provide error messages to the user should they require them, including a mandatory error message if there is no internet connectivity. These additions would ensure the system does not crash under any circumstances.

Secondly, I would implement the reminder feature. This would work by using Android's native Alarm feature. The user would simply click on a button and the start time would be added to the alarm. When the alarm is activated, a notification would appear on the user's status bar, this can be set to permanent until the user clicks on it, opening the application. Depending on the timescale given, I would aim to add extra functionality to this feature by allowing the user to select exactly when the reminder would be set to go off.

I would implement the settings feature too. However, from my testing, I have noticed that changing data sources using Metabroadcast's API has no effect on the results returned as they default the results to the Press Association unless the API key is upgraded or enrolled on other third-party data providers. During my implementation I noticed that changing the data sources in the API URL made no difference in the results returned so did not improve the system in any way. In future works, I would attempt to research into the effects of this feature and to see if it improves the quality of the system.

I would make tweaks to the GUI to allow for easier navigation between screens. For example, now that I understand Android Layout Resources, I would like to be more ambitious with the GUI using custom icons and buttons. I would also like to like to adjust the way in which search TV shows looks as I feel it looks a little empty when the user first opens the screen before typing in a query.

I would also like to explore other APIs as my priority at the start of this project was to find a single suitable API for the scope of the project. Now that I have gained a wealth of experience working with Android Studio, Google's Volley and JSON Objects, I would be very keen to apply this newly found knowledge to other APIs to extract even more information.

Conclusions

The main objective of this project was to produce a TV companion application for Android and this has been achieved. TV Companion is a success; The project has fulfilled all of the 'Must Have' and 'Should Have' requirements and has extended partially into 'Could Have' requirements with the bookmark and streaming feature. The application enables users to select, browse and search for TV shows and to view information about them. This has been achieved by successfully using a third-party API, something that I have personally wanted to achieve while in university.

I feel that the overall process has been successful too, I followed my work plan to the best of my abilities and although I did not meet face-to-face with my supervisor weekly, we liaised frequently via email; I continuously reported my progress with him and delivered the second milestone of demonstrating a functioning application. I feel that my illness, although unsubstantial, pushed progress back by around ten days towards the start of the project. Thankfully, I managed the impact of that risk by planning a buffer period in the centre of the project to allow myself to catch up if needed and it paid off greatly.

The level of success could have been increased by implementing all system requirements and ensuring robustness through implementing better error-handling. However, I have identified these factors in future works and given more time, I would be able to successfully implemented these improvements. Given the maintainability of my project, I confidently feel anyone with similar coding experience as myself would be successfully able to continue working on this project. Scalability is achievable too, adding channels requires adding a line of code per channel.

Reflection on Learning

Having completed this project, I have learned to understand the level of work necessary to undertake in order to produce a piece of quality software. During the course of the project I have learned how to program using Java for Android with Android Studio IDE, something I set out to do from the start of my degree. I have really enjoyed learning how to develop a mobile application for the first time and it has helped direct me towards my career goal of becoming a mobile software developer.

I feel that working on such a large project alone has improved my self-motivation more so than any other module during my time at Cardiff University. I have had to work with my own work plan and Gantt chart which has really helped me understand how time management plays a huge part in the success of a project. Feedback for my initial plan suggested that I was too meticulous in planning my Gantt chart, with the benefit of hindsight, I can understand this comment; I fell behind due to illness and felt that due to my meticulous planning, I was falling rapidly behind as I hadn't finished work in the time I set. In reality, I was still on course and really became my own worst enemy.

The project has not been without its failures; I have spent countless days trying to implement certain aspects of my project, namely the EPG. Trying to learn a new language on the fly while implementing complex code proved to be a huge personal challenge, one that I am proud to say I have achieved. I feel that the whole project has enabled me to dramatically improve my ability to self-learn, to be able to research without the need of a structured lecture or module. At first, the thought of working alone on such a large project was both exciting and daunting. I had a vague idea of how the system would work, that an API would be used and a Java based language to display the data. But I had no idea how different Android is to normal Java. The challenge to learn a completely new way of programming with Java after only covering Java for two years has been hugely rewarding. My understanding of Java has improved greatly and I feel that it will definitely help me in the future, both professionally and as a hobby.

On reflection, I feel I attempted to implement the code solution too early, before fully understanding what was required to fulfil the system requirements. Because of that, I wasted time having to go back on my work to adapt it to the final needs of the system. Although this didn't cause me any great delay, I feel If I were to code another application I would definitely aim to identify my personal limitations and opportunities for development before attempting to implement any code solution. I feel that more time should have been spent producing simple Android examples just to show proof of concept. For example, the EPG required using a ListView, an object that I was familiar with, but not to the extent where I would have been able to dynamically populate it with any data, let alone a complex custom Show object. Undoubtedly, I achieved this goal but I feel my methods were not best practise and if I were to do it again, I would take a more academic approach by going through examples provided by Google's Java documentation so that I could work with the objects better.

Table of Abbreviations

EPG	Electronic Program Guide
IDE	Integrated Development Environment
API	Application Program Interface
JSON	JavaScript Object Notation
XML	Extensible Markup Language
XMLTV	Extensible Markup Language Television
GUI	Graphical User Interface
HCI	Human Computer Interaction

Appendices

Images

Image 1: System output of JSON Results

Image 2: System output of JSON crash

```
4585-4699/com.example.c1319936.tvcompanion I/System.out: Name: BBC Wales Today; Weather ID: d48cpb Series ID: ds67cz
4585-4701/com.example.c1319936.tvcompanion I/System.out: Name: Party Election Broadcast ID: d5fqdm Series ID: cyn6
4585-4707/com.example.c1319936.tvcompanion I/System.out: Name: The One Show ID: d5bfp6 Series ID: dw9
4585-4713/com.example.c1319936.tvcompanion I/System.out: Name: BBC News; Regional News ID: d5bfp8 Series ID: hbs
4585-4715/com.example.c1319936.tvcompanion I/System.out: Name: Panorama ID: d487gr Series ID: 5w4
4585-4716/com.example.c1319936.tvcompanion I/System.out: Name: EastEnders ID: d48cnc Series ID: cf2
4585-4717/com.example.c1319936.tvcompanion I/System.out: Name: MasterChef ID: d48cnb Series ID: g6m
4585-4718/com.example.c1319936.tvcompanion I/System.out: Name: Peter Kay's Comedy Shuffle ID: d487gs Series ID: d4yjy4
4585-4727/com.example.c1319936.tvcompanion I/System.out: Name: I Want My Wife Back ID: d48cm8 Series ID: d4yjyz
4585-4729/com.example.c1319936.tvcompanion I/System.out: Name: BBC News at Ten ID: d48cm7 Series ID: ds67bf
4585-4730/com.example.c1319936.tvcompanion I/System.out: Name: BBC Wales Today; Weather ID: d48cn9 Series ID: ds67cv
4585-4731/com.example.c1319936.tvcompanion I/System.out: Name: Becoming Mr Nice ID: h99hj Series ID: h99hj
4585-4732/com.example.c1319936.tvcompanion I/System.out: Name: A Very Welsh Undertaking ID: dth8kx Series ID: dsyvz2
4585-4737/com.example.c1319936.tvcompanion I/System.out: Name: PFA Awards ID: d48cm4 Series ID: d48cm4
```

Image 3: System output of parsed JSON Object

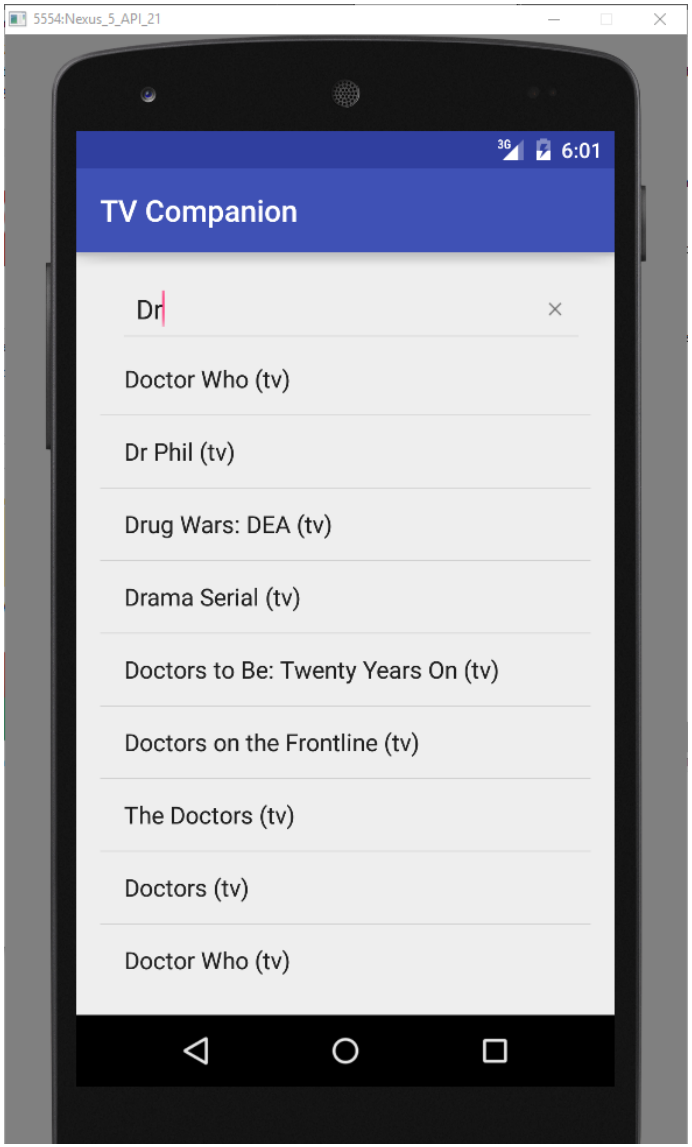

Image 4: Results from entering 'Dr'

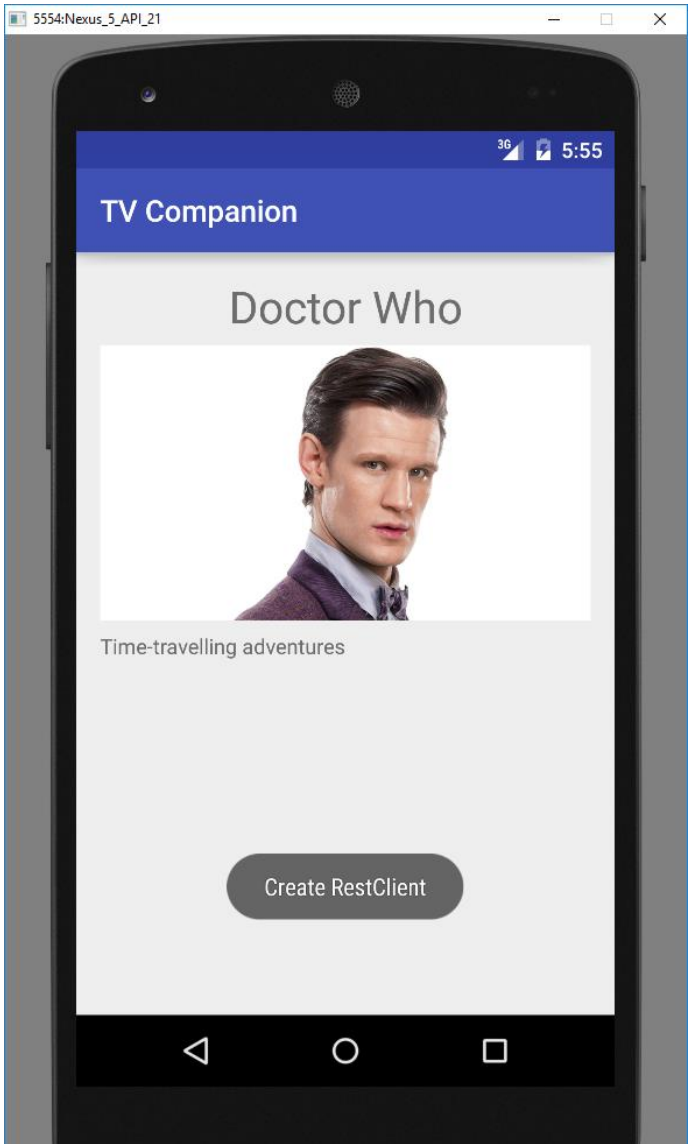

Image 5: Information displayed after clicking 'Doctor Who'

Image 7: EPG data for BBC 1

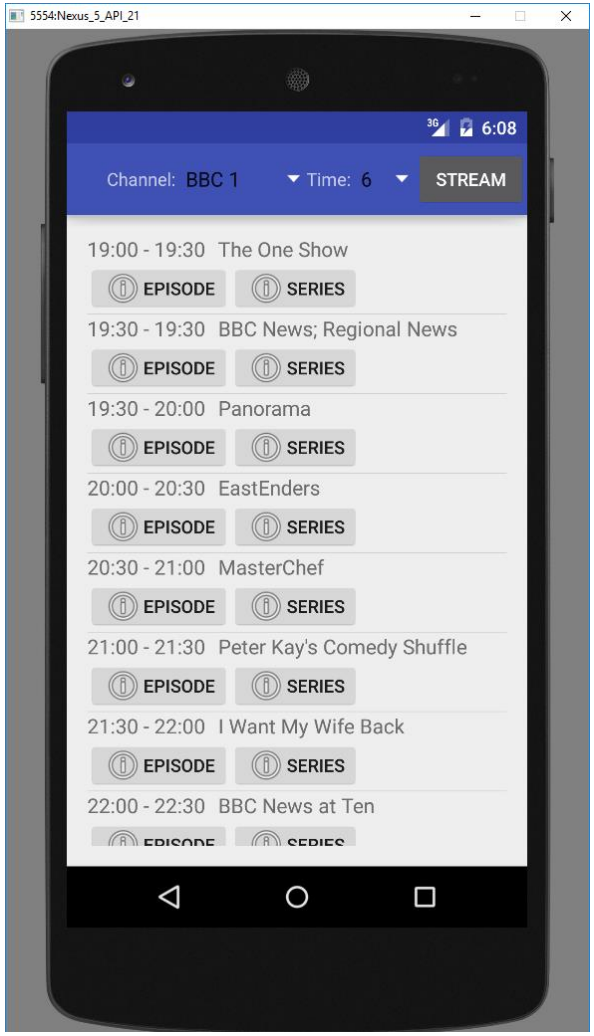

Image 6: Results after clicking Episode Info

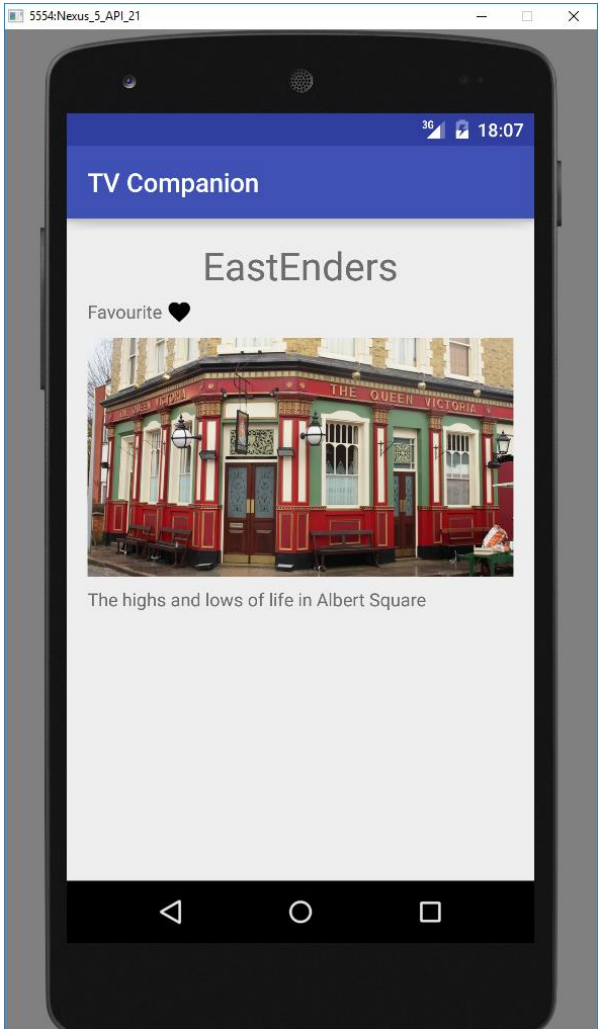

Image 8: Results after clicking Information Info

Code

Show.java

```
package com.example.c1319936.tvcompanion;

import org.json.JSONException;
import org.json.JSONObject;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

/**
 * Created by James on 27/03/2016.
 * This class is used to hold related information for a given Episode/Show
 */
class Show {
    String title;
    String ID;
    String imgURL;
    String description;
    String type;
    String series;
    String seriesID;
    String startTime;
    String endTime;
    String episode_number;
    String series_number;

    Show(JSONObject response, String startTime, String endTime){
        try{

            this.episode_number = "null";
            this.series_number = "null";

            if(response.has("film")){

                this.type = "film";
            }
            else if (
```

```
            response.has("episode")){type= "episode";
            this.episode_number = response.getJSONObject(type).getString("episode_number");
            this.series_number = response.getJSONObject(type).getString("series_number");

        }
        else{type ="item";}

        JSONObject data = response.getJSONObject(type);


        this.ID = data.getString("id");

        try{
        this.seriesID = data.getJSONObject("container").getString("id");}
        catch(JSONException e){
            this.seriesID = ID;
        }

        if(!startTime.equals("")) {
            this.startTime = formatTime(startTime);
            this.endTime = formatTime(endTime);
        }

        this.title = data.getJSONObject("display_title").getString("title");


        this.description = data.getString("description");


        try {
            this.series = data.getJSONObject("series").getString("id");
        } catch (JSONException e) {
            this.series = "null";
        }
        try {
            this.imgURL = data.getJSONArray("images").getJSONObject(0).getString("uri");
        } catch (JSONException e) {
            this.imgURL = "null";
        }

    }
    catch(JSONException e){
        e.printStackTrace();
```

```
        }

        System.out.println("Name: "+title+" ID: "+ID+ " Series ID: "+ seriesID);
    }

    /*
    * Metabraodcast uses a String for their transmission time (Very annoying)
    * it is formatted as "2014-09-26T14:45:00.000Z"
    * in order to use this date, it would be useful to convert it into a Java friendly entity
    * setTime does this by taking the raw String and outputting a simple String formatted as "HH:mm" (22:00)
    */
    private String formatTime(String strDate){

        //Set the output
        String output = null;
        //First let's cut the useless information (everything bar the time portion)
        String input = strDate.substring(11, 16);
        //An empty date variable is used (explained below)
        Date date;
        //Now let's setup a simple date format
        SimpleDateFormat df = new SimpleDateFormat("HH:mm");


        try {
            //Parse the time, creating a date variable
            //(This has a side effect of creating a fully formatted date including Day, Month etc)
            date = df.parse(input);
            //Format this date to only show the time
            output =  df.format(date);

            //As we are currently in GMT+1, we need to add an hour to our time
            //TODO convert this section to dynamically change time according to timezone
            //Create a calendar to achieve this
            Calendar cal = Calendar.getInstance();
            //Add our date variable
            cal.setTime(date);
            //Increment the calendar by an hour
            cal.add(Calendar.HOUR, 1);
            //Return our date
            date = cal.getTime();
            cal.clear();
            //Reapply our date format
            output = df.format(date);
        } catch (ParseException e) {
```

```
            e.printStackTrace();
        }

        //Return our new time!
        return output;
    }


    public void setTime (String startTime, String endTime){

        this.startTime = startTime;
        this.endTime = endTime;

    }

}
```

mainEPG.java

```
public void loadEPG( final String chan, final String until) {

    //empty the Shows Array List
    shows.clear();

    //create a new thread as we want this process to happen seperate to the UI thread
    Thread t = new Thread(new Runnable() {
        @Override
        public void run() {

            //create an empty response to hold our results
            JSONObject response = null;
            //create a new instance of my custom ScheduleThread that extends AsyncTask
            ScheduleThread scheduleThread = new ScheduleThread(chan, until);

            //size of results set to zero
            int size = 0;
            //variables to get around incorrect times between queries of the API
            String startTime = null;
            String endTime = null;

            try {
                //execute the schedule thread
                response = scheduleThread.execute().get();
                try {
                    //try to fetch the size of the results
                    size = response.getJSONObject("schedule").getJSONArray("entries").length();

                } catch (JSONException e) {
                    e.printStackTrace();
                }

            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }

            //now make an Array of threads equal to the size of the results
            Thread[] threads = new Thread[size];

            //for every entry returned in the schedule
```

```
            for (int i = 0; i<size;i++) {

                //set an ID for the episode
                String ID = null;
                try {
                    //fetch the ID
                    ID =
response.getJSONObject("schedule").getJSONArray("entries").getJSONObject(i).getJSONObject("item").getString("id");
                    //fetch the start and end times from the schedule JSON object
                    startTime =
response.getJSONObject("schedule").getJSONArray("entries").getJSONObject(i).getJSONObject("broadcast").getString("transmission_ti
me");
                    endTime =
response.getJSONObject("schedule").getJSONArray("entries").getJSONObject(i).getJSONObject("broadcast").getString("transmission_en
d_time");
                } catch (JSONException e) {
                    e.printStackTrace();
                }

                //use final variables to enable use of variables within inner class
                final String finalID = ID;
                final String finalStartTime = startTime;
                final String finalEndTime = endTime;

                //create a new Thread at index i of the list
                threads[i] = new Thread(new Runnable() {
                    @Override
                    public void run() {

                        //create an empty response to hold our results
                        JSONObject JSONObject = null;
                        //create a new Schedule Thread passing the ID of the episode
                        ScheduleThread threadA = new ScheduleThread(finalID);

                        try {

                            //try to execute the thread
                            JSONObject = threadA.execute().get();

                            //System.out.println(JSONObject); USE FOR TEST 001

                            //add the JSON Object result to the list of Shows with the original
                            //times found in the initial query (API returns broadcast times in a peculiar order otherwise)
                            shows.add(new Show(JSONObject, finalStartTime, finalEndTime));
```

```
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        } catch (ExecutionException e) {
                            e.printStackTrace();
                        }
                    }
                });

                //start the thread
                threads[i].start();
                try {
                    //join the thread, meaning nothing else will run until that thread is done
                    threads[i].join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            //now that the Shows Array List is full of results we need to show them
            //this code has to be run on the UI thread
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    //display the shows
                    displayEPG(shows);
                }
            });

        }
    });

    //start the original thread that encompasses this
    t.start();

}


//my custom ScheduleThread class
protected class ScheduleThread extends AsyncTask<Void, Void, JSONObject>{

    //ID needed to add to query
    final String ID;
    //URL to hold the query url
    String url;
```

```
    //constructor that takes channel and until, this is used for the first query to fetch episode ID's that are scheduled on that
channel
    public ScheduleThread(String channel, String until){
        //convert the channel name to its unique ID
        this.ID = dict.getChannel(channel);
        //add the channel ID and until variables to the query url
        this.url
="https://atlas.metabroadcast.com/4/schedules/"+ID+".json?from=now&to=now.plus."+until+"h&source=pressassociation.com&annotations
=channel&key=c1e92985ec124202b7f07140bcde6e3f";
    }
    //constructor that takes just the eiposde ID, this is used for the queries to fetch actuall information for each episode
    public ScheduleThread(String ID){
        //set the ID
        this.ID = ID;
        //add the episode ID to the query
        this.url
="https://atlas.metabroadcast.com/4/content/"+ID+".json?type=item&source=pressassociation.com&limit=2&offset=0&annotations=brand_
summary,broadcasts,channel,description,images,people&key=c1e92985ec124202b7f07140bcde6e3f";
    }

    @Override
    protected JSONObject doInBackground(Void... params) {
        //in the background thread, make a new Volley Future Request
        RequestFuture<JSONObject> future = RequestFuture.newFuture();
        //add the url to the request
        JsonObjectRequest request = new JsonObjectRequest(url, null, future, future);
        //add request to the 'queue'
        myQueue.add(request);

        try {
            //request the query to run, giving 10 seconds to timeout
            return future.get(10, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {

            e.printStackTrace();
        } catch (TimeoutException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Questionnaires

Q1: Have you ever used an application to view TV information?

No

Q2: Describe your level of experience using a smartphone

Beginner

Q3: Were you able to search for a TV show?

Yes

Q4: Were you able to favourite a TV show?

Yes

Q5: Were you able to browse the Electronic Program Guide?

Yes

Q6: Were you able to remove a favourite show?

Yes

Q7: Were you able to view information for a selected show?

Yes

Q8: How easy was the application to use? (0 Impossible - 5 Extremely easy)

5

Q1: Have you ever used an application to view TV information?

No

Q2: Describe your level of experience using a smartphone

Intermediate

Q3: Were you able to search for a TV show?

Yes

Q4: Were you able to favourite a TV show?

Yes

Q5: Were you able to browse the Electronic Program Guide?

Yes

Q6: Were you able to remove a favourite show?

Yes

Q7: Were you able to view information for a selected show?

Yes

Q8: How easy was the application to use? (0 Impossible - 5 Extremely easy)

5

Q1: Have you ever used an application to view TV information?

Yes

Q2: Describe your level of experience using a smartphone

Intermediate

Q3: Were you able to search for a TV show?

Yes

Q4: Were you able to favourite a TV show?

Yes

Q5: Were you able to browse the Electronic Program Guide?

Yes

Q6: Were you able to remove a favourite show?

No

Q7: Were you able to view information for a selected show?

Yes

Q8: How easy was the application to use? (0 Impossible - 5 Extremely easy)

3

Q1: Have you ever used an application to view TV information?

No

Q2: Describe your level of experience using a smartphone

Advanced

Q3: Were you able to search for a TV show?

Yes

Q4: Were you able to favourite a TV show?

Yes

Q5: Were you able to browse the Electronic Program Guide?

Yes

Q6: Were you able to remove a favourite show?

Yes

Q7: Were you able to view information for a selected show?

Yes

Q8: How easy was the application to use? (0 Impossible - 5 Extremely easy)

5

Q1: Have you ever used an application to view TV information?

No

Q2: Describe your level of experience using a smartphone

Intermediate

Q3: Were you able to search for a TV show?

Yes

Q4: Were you able to favourite a TV show?

Yes

Q5: Were you able to browse the Electronic Program Guide?

Yes

Q6: Were you able to remove a favourite show?

Yes

Q7: Were you able to view information for a selected show?

Yes

Q8: How easy was the application to use? (0 Impossible - 5 Extremely easy)

4

Q1: Have you ever used an application to view TV information?

No

Q2: Describe your level of experience using a smartphone

Beginner

Q3: Were you able to search for a TV show?

Yes

Q4: Were you able to favourite a TV show?

Yes

Q5: Were you able to browse the Electronic Program Guide?

Yes

Q6: Were you able to remove a favourite show?

Yes

Q7: Were you able to view information for a selected show?

Yes

Q8: How easy was the application to use? (0 Impossible - 5 Extremely easy)

3

References

Media.ofcom.org.uk, 2015. The UK is now a smartphone society [Online]. Available at: http://media.ofcom.org.uk/news/2015/cmr-uk-2015/

[Accessed: 29 January 2016].

Laura Zain, A. 2015. Over a quarter of UK smartphone users use their phones while watching TV [Online]. Available at: https://www.comscore.com/Insights/Data-Mine/Over-a-quarter-of-UK-smartphone-users-use-their-phones-while-watching-TV

[Accessed: 29 January 2016].

Atlas.metabroadcast.com, 2016. api documentation - atlas [Online]. Available at: http://atlas.metabroadcast.com/api-docs/

[Accessed: 1 February 2016].

Developer.bbc.co.uk, 2016. BBC Developer Portal | Programmes API [Online]. Available at: https://developer.bbc.co.uk/content/programmes-api

[Accessed: 1 February 2016].

Digiguide.tv, 2016. Digiguide.tv API Documentation [Online]. Available at: https://digiguide.tv/api/documentation/default.asp

[Accessed: 1 February 2016].

Play.google.com[1], 2016. Available at: https://play.google.com/store/apps/details?id=uk.co.freeview.android

[Accessed: 1 February 2016].

Play.google.com[2], 2016. Available at: https://play.google.com/store/apps/details?id=com.lifefoc.uktvlistings

[Accessed: 1 February 2016].

Play.google.com[3], 2016. Available at: https://play.google.com/store/apps/details?id=com.feertech.android.tvguide

[Accessed: 1 February 2016].

Play.google.com[4]. (2016). [online] Available at: https://play.google.com/store/apps/details?id=com.typing4me.tvguide.UK&hl=en

[Accessed 1 February 2016].

Play.google.com[5] (2016). [online] Available at:

https://play.google.com/store/search?q=TV%20Guide&c=apps&hl=en

[Accessed 1 Feb. 2016].

Android Studio, 2016. Android Studio Overview | Android Developers [Online]. Available at:

http://developer.android.com/tools/studio/index.html

[Accessed: 2 February 2016].

Thetvdb.com, 2016. Programmers API - TvDBwiki [Online]. Available at:

http://thetvdb.com/wiki/index.php?title=Programmers_API

[Accessed: 2 February 2016].

Nielsen, J. 2016. 10 Heuristics for User Interface Design: Article by J Nielsen [Online]. Available at:

https://www.nngroup.com/articles/ten-usability-heuristics/

[Accessed: 9 February].

Inkscape 2016. Home | Inkscape [Online]. Available at:

https://inkscape.org/en/

[Accessed: 9 February 2016].

Nurik, R. 2016. Android Asset Studio - Icon Generator - Launcher icons [Online]. Available at:

http://romannurik.github.io/AndroidAssetStudio/icons-launcher.html

[Accessed: 9 February 2016].

Proto.io 2016. Proto.io - Prototypes that feel real [Online]. Available at:

https://proto.io/

[Accessed 11 February 2016].

Google 2016. Sending a Simple Request | Android Developers [Online]. Available at:

http://developer.android.com/training/volley/simple.html

[Accessed: 27 February 2016].

Smith, J. 2016. Android Asynchronous Http Client [Online]. Available at:

http://loopj.com/android-async-http/

[Accessed: 27 February 2016].

Google[2] 2016. AlarmManager | Android Developers [Online]. Available at:

http://developer.android.com/reference/android/app/AlarmManager.html

[Accessed: 28 February 2016].

Google[3] 2016. AsyncTask | Android Developers [Online]. Available at:

http://developer.android.com/reference/android/os/AsyncTask.html

[Accessed: 11 March 2016].