Glyndwr Timetabling App

Contents

1. Introduction

The Glyndwr Timetabling app is an android application which falls under the 'Tools' category and is utilitarian in nature. The app is aimed at students of the university as an extra method of attaining timetables related to; rooms, modules and courses. The apps necessity arises due to the website where the timetables can be found being inherently not mobile friendly [1].

The apps main function will take a user's input as a search term and web-scrape the universities timetable website to find the associated pdf file. It will then display the pdf file to the user and offer the user the option to discard or save the timetable to a local database. The app will have a menu system (also its home screen) which will allow the user to navigate between; searching, saved timetables, About section and the Help section.

2. Research –

2.1. App Monetisation –

App monetisation is the method used by developers and organisations that create and own applications on various platforms to create revenue streams in order to provide value to their business models from those assets. This could be implemented in many various forms such as directly purchasing the app to 'subscribing' to a service for greater functionality to name but a few [2].

There are many various methods to implement monetisation of apps and each have their associated merits and downsides. This may be because a direct purchase would stand as a larger barrier for a user to overcome with regards to making a purchase compared to offering a paid version of a free app which may entice that user much more as they have already been introduced to the app. The most obvious route most free products on mobile app stores take these days is in-app advertising [3]. Advertising through mobile apps has allowed a greater level of communication to consumers because of the large amount of time users are on their phones. This model is therefore an effective choice for advertisers and potentially lucrative option for developers as it is estimated to account for over 75% of all Digital Ad spend, especially so for in-app ads [4].

There are four general types of ad formats for a developer to choose from; banner ads, interstitial ads, native ads and rewarded video. Banner ads are the oldest and most common form of what people would think of when considering mobile ads. These are little sections of the screen filled with text and images. They are very simple, however the overuse and intrusiveness of their nature can be very annoying to users. This can lead to distraction which may impact upon the amount of time the user may be on the app. This in turn will affect revenue as less users are likely to click through [4].

Interstitial ads are the ads that run in full screen and take over the whole interface of the app that they are hosted in. Typically, they are found at transition points i.e. when there is a level change so that they have minimal distraction to the user immersion. The ad usually offers the user to close it after a brief period of time or to click through to the advert [5]. Interstitial ads can have major downsides such as the content requiring much more work since it takes over the entire screen, the ad can be considered highly intrusive if the transition points are not

carefully considered and have been penalised in some apps as they are considered against Googles guidelines [6]. A study found that ad viewers typically fixated 22% of the time looking for the 'X' to close the ad [4].

Native ads and rewarded video are considered the much more modern and effective method of in app advertising. The former is designed to blend in with its host app such that immersion for the user remains unaffected. This results in much higher engagement rates and therefore leads to much higher Cost Per thousand Impressions (CPM). The advertiser uses CPM to score how valuable an ad method may be so that they can better prioritise adverts. The latter method rewarded video is concerned with the user deliberately watching an ad for a period of time anywhere from 10 seconds up to 30+ seconds in order to unlock a 'reward'. This may be implemented in game apps by unlocking boosts, or has been very aptly used in the very popular streaming service 'Spotify'. Allowing users to listen free for 30 minutes after listening to an ad [4].

Outside of putting adverts directly in the app, there are other methods available to developers to monetise their work. The developer can find sponsors and partnerships so that multiple organisations can work together to advertise each other's apps within their products [3] [7].

If developers had made a successful app with a strong code base, other organisations may approach them to reuse that work or 'reskin' it and license it out to them [3]. Reskinning apps saves money and resources as developers no longer have to produce entire codebases. Rather, they can just code existing ones into a new design. Many of the most downloaded games fall under this existing method such as the 'King Sagas'. This initially started with 'Candy Crush Saga' and now has over 293 million people playing each month. This methods monetisation comes in when we look at the code purchasing aspect of it. Source codes can be found very cheap compared to the labour involved in producing one from scratch therefore being very appealing to 'Reskin' developers. The original developer still owns the code but a copy is provided either with a 'limited' or 'unlimited' license, the former allowing the developer to reskin a defined amount and no more while the latter has no such restriction [8].

The last couple of monetisation strategies are entirely dependent upon the user base and cut the advertisers out of the process. The first model is in-app purchases which is platform dependant, if the user has an Apple smartphone then in- app purchases are facilitated by use of Apple's 'App Store' [9]. These purchases can be for a variety of content whether it's one time purchases for features or in game items/currency, subscriptions or for services. These break down further into 'consumable and non-consumable' purchases which relate to one time purchases for features to unlock and purchases that can be repeated for things like items respectively. Subscriptions can be made to be auto-renewable or non-renewing. On Apple apps this can be implemented through Apple's 'Storekit Framework' [9]. Googles implementation covers all the same type of purchasing models but requires the developer to setup a 'Google Wallet Merchant Account', Define the 'durables' and 'consumables' (their variation of Apples consumable/non) and to integrate Google's 'In-app Billing API' into their app [10] [3].

2.2. User Experience (UX) and User Interface (UI) App Design –

It is imperative in the modern days approach to application development that the intended deliverables have a well-designed and thought out underlying plan with regards to both User

Experience (UX) and the User Interface (UI). Without such an approach, teams can fall short of producing the 'right' product and not be able to sustainably hold an effective development process – often having to go back to the initial design to make the appropriate alterations necessary to overcome limitations in development [11].

To make an effective design base, the developer(s) must have some knowledge of what the fundamentals of design are concerned with, namely the UX and UI design. UX design refers meeting the exact needs of a customer/end-user without incurred hassle. A hassle free experience usually implies the user has a 'joy to use/own' the product and therefore the product is deemed more likely to be successful in its field and be more marketable [12]. More aptly, UX design encompasses 'all aspects of the end-user's interaction with the company, its services and products' [13]. With such an encompassing nature, it is not surprising to see that UX Design has many core areas of research to explore and that all prove useful to overall approach of good design. Such areas covered include;

- UX Architecture (UxA)
- Information Architecture (IA)
- User Experience Design (UxD)
- Interaction Design (IxD)
- Navigation Design
- UX Strategy (UxS)

User Interface (UI) Design could be argued as one of the areas also encompassed within UX, however it in itself contains many areas of research too, these include; user research, usability engineering, usability validation and context strategy so UI is considered by some as the 'half-sister of UX' [13]. The three basic principles that are found throughout all areas are to 'focus on the user, measure and iterate'. [14]

User interface (UI) design is important as an intuitive interface removes a lot of the guesswork for users. An interface that is clunky and difficult to use would be a detriment to the user's experience and as mentioned previously, incurred hassle affects the products potential for success [15]. Design is influenced by a range of factors varying from user needs, culture and context on the end-user side to compliance, business needs and sponsor/market opportunities when the business side of design is considered. Good design also factors in cost and lifespan [14].

There are six common principles by which to design an effective user interface, these are; structure, simplicity, visibility, feedback, tolerance and reuse. The Structure principle describes an effective interface as having purposeful organization – meaning the model must be clear and consistent to the user such that related things are together and unrelated things are separated. Simplicity principle is concerned with making tasks easy to use. Visibility ties in here too as the tasks should have all relevant options and materials available i.e. a fitness tracker task may need the options to input weight, height and exercises. Information that goes above and beyond such as information about exercise goes above and beyond Visibility and Simplicities principle brief and may inundate the user [15]. Feedback principle should keep users aware of any changes that may occur while being used. Tolerance principle aligns with the design being flexible so that mistakes can be avoided by undoing/redoing and preventing errors. This can be done by tolerating various inputs i.e. handling exceptions when inputs are

non-sensical like inputting a word into a number entry or simply disallowing incorrect entry types altogether where possible. The final principle of reuse relates to reusing components where possible [15].

User interaction in mobile apps is considered very important in the realm of research as interaction mining is a very good source of gathering useful user-related information. This approach is carried out in a 'black box format' where the design and interaction generated data is captured during the App's usage [16]. In this, UI components are seen to be the low level 'semantics that describe structural roles' like images, content and colour. UX components are considered functional semantics to the likes of buttons, icons and other interactive tools the user can interact with [16].

Many successful developers have hinted that a good UX has played a large role in the success of their apps. This is because todays user expects a lot from apps, such as: fast load times, ease of use and a joy to use. In order to effectively design the UX components of a mobile app, a set of guidelines to follow will drastically improve an app's potential of success. These guidelines generally draw from successes of other App's as well as learn from mistakes of failed implementations by others. The guidelines are not guaranteed to work, nor do they all have to be followed but as a general rule of thumb these aspects can improve success [17].

- Minimizing Cognitive Load – Not overwhelming the user with too much information
- Decluttering – Keeping content and interface elements to a minimum for simplicity and ease of use to the user
- Offload tasks – Avoiding extra user effort where possible by reusing previously entered data or using smart defaults where possible to lower the barrier to entry for the user
- Breakdown of task into smaller chunks – Large tasks like signup, purchases and subscriptions can tax user's attention to the point where they may give up. This can also tie in with the first point of minimizing cognitive load. Break these large tasks into smaller parts and remember to offload where possible to.
- Anticipate User Needs – Helpful hints to describe complex areas
- Avoid Jargon
- Be consistent throughout design [17] [18]

It may be necessary that larger teams follow a more rigorous or structured approach. This is where methodologies could help, most follow a good structure that flows through the whole process. A good example of this is 'Task Centered System Design (TCSD)' whereby the users' problems are identified, and a task is created and described. The task is then analysed to decide whether it will be included or omitted from the design. The design is then created which identifies the process and data required (this may be done in simulation) and finally the walkthrough is evaluated to determine whether or not the design solution solves the initial outset of problems described [19].

On top of the interface design, the developer must also consider the delivery method. Mobile apps are more complex than they may seem. There are three types of apps that each have their own benefits and limitations that can play a role in the users experience. They are Native Apps, Hybrid Apps and Responsive Web Apps. A native app is designed to run on the platform its on – i.e. a native android app like Googles 'Gmail' will run really well on the

android operating system as it was designed for it. Native apps can usually access the full functionality of the device easily and are most like to run error free. The limitation being that Native apps can only run on their intended operating system and cannot work on others [20].

The second type of app is a 'Hybrid App'. These apps are designed to work on multiple platforms using a singular standard code like C# and then compiled to work on differing platforms. This is similar in concept to Java and the JVM where the code can be made once and compiled to run on many different operating systems. This makes development much easier. However, complex interactions and specific functionality may be dramatically reduced and a hybrid app may have larger maintenance costs than native variations [20].

Finally, the last type are responsive web apps. These are primarily developed in HTML5 and JavaScript and the more creative variations may be hardly noticeable to the user as a website. Developers who choose this option should choose a "mobile first" approach to development. There are large drawbacks to this type of app in that they can't be distributed through app stores – this can affect the apps monetisation. Secondly, as its inherently a website the app needs a constant web connection to function as intended [20].

3. Methodologies –

3.1. SDLC – Software Development Life-Cycle

Software Development Life Cycle (or SDLC for short) is a methodology with a set of processes aimed at creating high quality software. This methodology follows these phases; Requirement analysis, Planning, Software Design, Software Development, Testing and Deployment [21]. The methodology aims to produce high quality content at the lowest cost and shortest time feasible. This provides cost saving methods to the developers without harming the quality of deliverables [21].

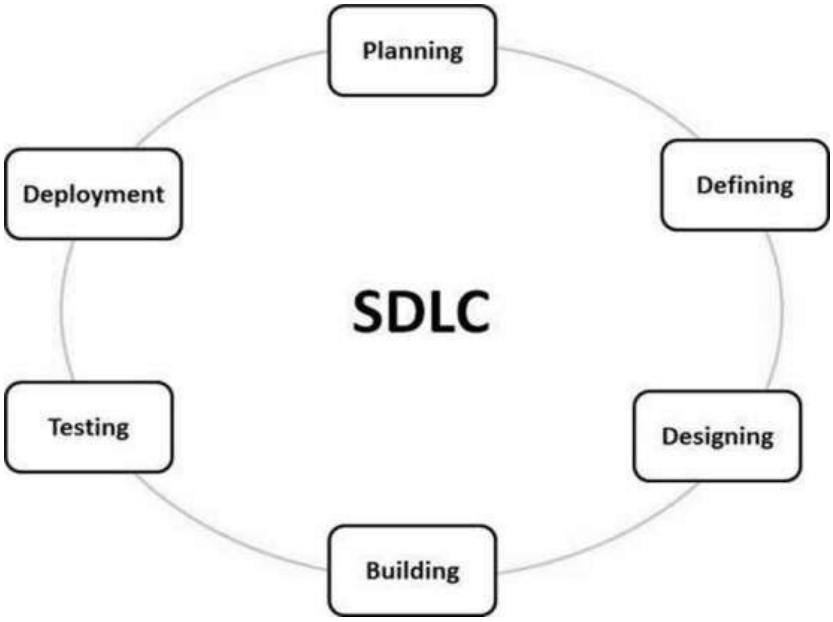

Figure 1 - SDLC Overview Diagram [22]

SDLC starts out by identifying the current problems, otherwise known as the requirements planning stage. In this stage, we gather input from all parties involved. Typically, this may involve stakeholders, end-users, experts, and members of the development team. This allows

a varied view of the strengths and weaknesses of current systems for improvement. In the case of new developments, it highlights development goals necessary for the software to be properly effective [21].

The second stage of planning is formulated from the requirements that have been gathered. Here the team determines the resources that are necessary for implementing said requirements. The team also identifies risk factors to the project and relevant solutions to alleviating them should the risks arise [21].

Once the requirements are finalised, the risks and resources managed then design can start. A good design approach highlights all the architectural aspects of the app along with data flow and communication to external components of the system. Internal system components should be described in high level detail [22].

Stages 4 and 5 of SDLC are concerned with the building and testing of the product. Here is where the development team take the largest amount of participation in. Their role being to develop the product in its entirety by following the designs and being aware of any risks. The testing verifies their work against a set of goals that are determined by earlier requirements gathering. Should any defects arise they will get 'reported, tracked, fixed and retested'. This is done to ensure the product reaches a good standard. After this, the product can finally be deployed to the marketplace [22].

3.2. RAD - Rapid Application Development

RAD is an approach used to help develop information systems. This development methodology belongs to the Agile family of methodologies. The key objectives of RAD are similar in nature to that of SDLC's; high quality product delivery and low resource costs with the caveat that RAD delivers on a much faster timescale in comparison to SDLC. This is done through streamlining some processes which take overly long in SDLC. 'Speed' is set as the significant core ideal as with any other Agile type development methodology [23].

Unlike SDLC, RAD stays to just four main phases to the approach. These four phases are essentially what is found in the former approach but more streamlined. RAD will be the approach I take in developing this timetabling app going forward. The four phases are;

Phase 1 – Requirements Planning

In this phase we determine the projects scope. This is similar to SDLC's; however, it is much more condensed to account for the critical time nature of typical RAD-based solutions. Like with the other phase the developers, users and stakeholders all communicate to determine project goals as well as issues that may arise and how they may be addressed [24].

Phase 2 – User Design

Once scoped out, user design can be built out iteratively through the use of prototyping. Each prototype furthers the design in accordance with the users needs. Each iteration can be tested to work out the bugs and kinks [24].

Phase 3 – Rapid Construction

This phase takes the latest prototype out of Phase 2 and converts it into the working model. This involves preparation, program development i.e. Program Logic and then the various forms of testing such as Unit, integration and System testing [24].

Phase 4 – Cutover

Product goes to launch at this point. All final changes are made, user training can now begin. In terms of apps, here is where the release to the various platform's app stores can occur [24].

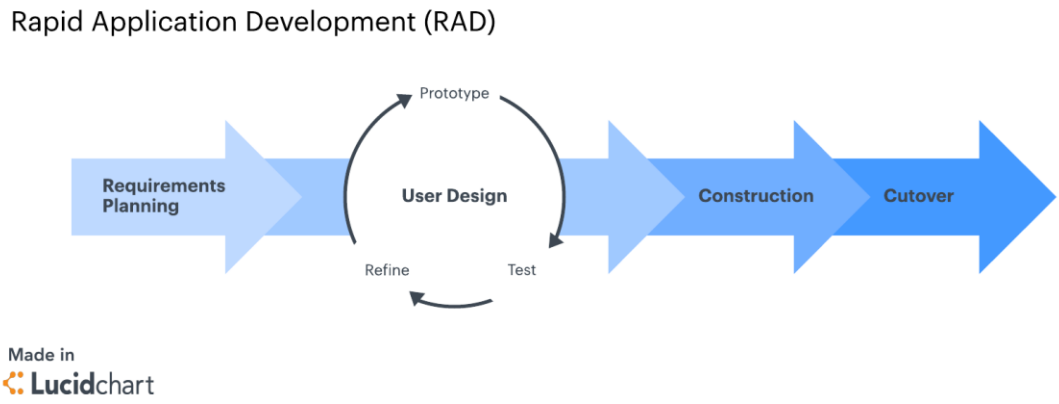

Figure 2 - Rapid Application Development (RAD) Methodology [24]

RAD provides a measurable form of progress. This is essential in keeping on track with development and allows for honest reports to stakeholders. It also leads to quick generation of productive code and compartmentalising components of the system, so that they can be better dealt with on their own. The iterating phase 3 allows for constant user feedback and also allows for early system integration of components into a cohesive whole [25].

RAD is not good with Large-Scale Projects as proper management can be very difficult with the flexibility and volatility within its nature. Because users play a role in each iteration, the demand for user interfacing can be very high and potentially cumbersome to the overall progression rate of development. It is worth noting that RAD is best suited to well skilled developers and quick adaptation and evolution to issues that may arise is essential [25].

4. Requirements Planning & Analysis –

4.1. Main Menu and Navigation

This is the main activity which will display upon launch. To make it more simplistic and intuitive to navigate. The home screen will be the main menu and it will contain four buttons to navigate to; Search, Saved, Help and About activities. Each activity listed will include a button at the bottom of their respective pages to navigate back to the main activity. As the main menu is the main activity and the entry point to the application, I shall put a logo of my own making in to improve stylistic qualities of the app and maintain a higher level of user engagement. In the theme of being an app that aids Glyndwr

Figure 2 - Glyndwr Timetables Logo

students specifically, all activities will also include a bottom bar image similar in likeness to Glyndwr's in keeping with their house style to improve the aesthetic quality therein.

Figure 3 - Glyndwr Timetables House Style Bottom Bar

4.2. Section Requirements

4.2.1. About & Help Sections

The About and Help sections of the app are requirements of the app specification and scope. These activities are near identical in all aspects except for general content. The About page will contain information relating to the app and creator. The Help page will have a 'How to use the app' section and the legend key that can be found on the Glyndwr timetables website that helps describe the various formatting of the actual PDF tables themselves. The help page will have some details for the user to contact should any issues arise that require fixing.

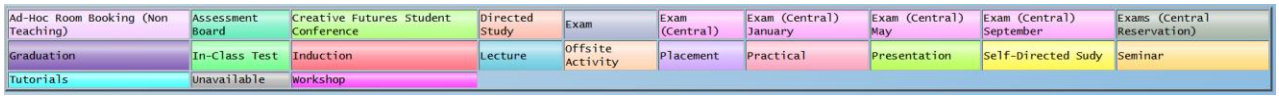

Figure 4 - Legend Key from Glyndwr Timetables site [1]

4.2.2. Search Timetables Section –

The search timetables section will form the main functionality of this app. This is the only activity which will carry out the required 'web-scraping' in order to gather the timetable information and links. The web-scraping will be done with the use of the 'Jsoup library' [26] and will occur during the activities 'onCreate'.

The timetable information will be stored as objects in an 'ArrayList' to populate a 'Scraped' table in a SQLite database. This will hold the fields; ID, Name, Dept, type and PDF URL. The layout will consist of an editText (with a hint to describe purpose) for the user to input search terms which will be searched against the table. This search will be initiated by a button and results will be loaded into a 'Recyclerview'. The user can then click on an item in the list to access further information about the timetable. This opens a dialog with the information and further options; Save, Open and Close. 'Save' will save the file using 'URLconnection and InputStream' to save the file to internal storage and populate a 'Saved' table with the record's information and 'PDF filepath' in lieu of 'PDF URL' as designated previously [27]. 'Open' will open the PDF file in a new layout using PDF Renderer which was brought in at API level 21, this will limit audience by a small but negligible margin as Google currently has cumulative distribution at this level at 94.1% [28].

4.2.3. Saved Timetables Section –

The saved timetables section is secondary in terms of importance for app functionality. Once the user has saved any timetables in the 'Search' timetables activity, the 'Recyclerview' in the 'Saved' activity will populate with any records from the 'Saved table' from the SQLite database just like it displays in the search activity. Once an item is clicked a near identical dialog to the search activities will open, except instead of the Save option there will be a delete option which will delete the file from internal storage and the record from the saved

table. The PDF functionality here will call the same functionality as found in search activity. For the layout, only the Recyclerview is necessary along with the back button.

4.3. – Database Requirements

4.3.1. Database

For this project, I will be using SQLite Version 3. SQLite is a small, fast and highly reliable SQL database engine that works really well with Android. From the requirements so far, it seems that only two tables are necessary to implement to allow full support for the apps functionality; a Search table for storing scraped timetable results and a Saved table for storing saved timetables for the user.

4.3.2. Search and Saved Tables

The search table will only be populated with records when the user is on the search activity. It will be populated by the Web-scraping process in the activities onCreate. This may seem a slower solution but it ensures that the content is always up to date with the latest timetable information. The table will empty on the activities close. The columns will be as follows; Primary key – ID, Name, Dept, Type and PDF URL.

The Saved tables will be populated by the user selecting the ‘Save’ option on the saved timetables dialog. For the most part, the columns will be the same in this table except for PDF URL. Instead of storing the URL, the table will save the file path the PDF file stored in internal storage. This will make loading the table much faster for the user. The Delete option in the Dialog screen will delete the record from this table and subsequently delete the PDF file using the file path from internal storage.

5. Design –

5.1 UI / UX Design

5.1.1 Wireframe –

Main Activity – Home / Menu

- Text will be in an editText at the top and four buttons below.
- All text will be in bold with the menu being about 32sp and buttons being 18sp. This helps accessibility somewhat as they are clearer to read.
- The centre will be taken up by a logo designed for the app by myself.
- To keep in line somewhat with the theme of being a university aid to Glyndwr students, there will be an image along the bottom of each standard activity with the bottom bar from Glyndwr's general house style.
- All images will have a content description to improve accessibility for users with screen readers.

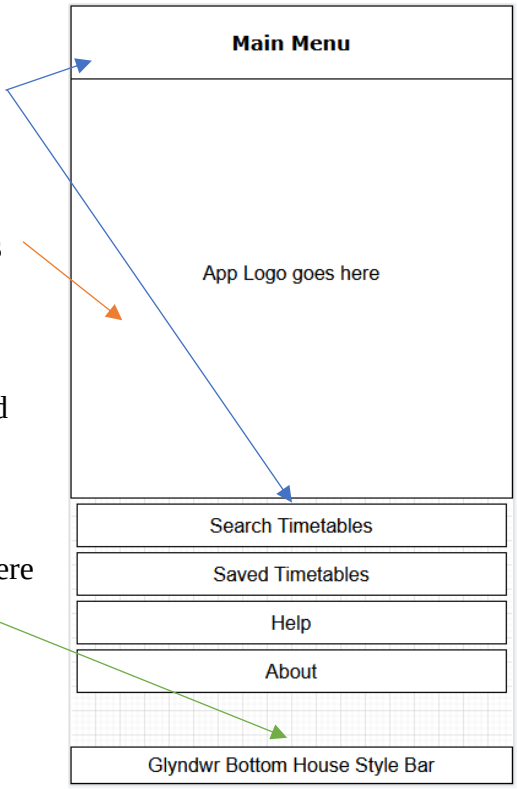

Figure 5 - Main Activity - Menu Wireframe

Search Activity -

- editText for user to enter Room / Course / Module they wish to search for. This has a hint on to indicate to the users what the purpose of this component is
- Button for user to confirm and search the timetables for what they are searching for
- List that will populate with all the search results that can be clicked to open a dialog to allow further options
- Back button for navigation back to main menu

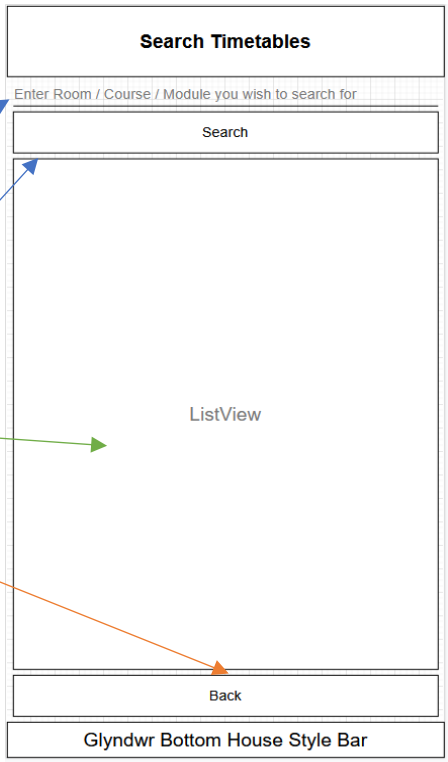

Figure 6 - Search Activity Wireframe

Saved Timetables -

- Like ‘Search Timetables’, this activity will also use a List to display records. However, it populates the list from the saved database. User can still click on the item to bring up a dialog for further options.

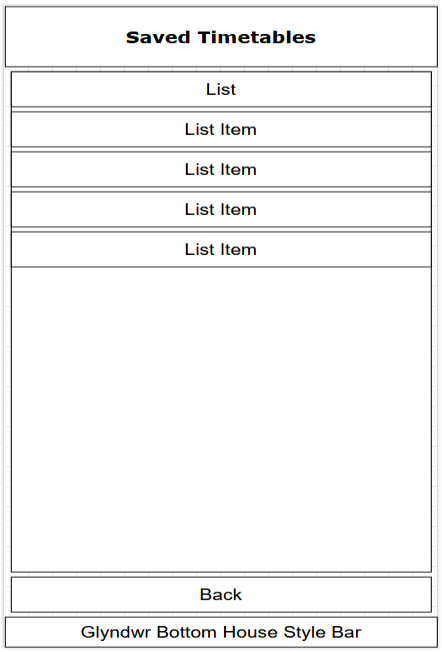

Figure 7 -Saved Timetables Wireframe

Dialogs -

- Saved Timetables Dialog – Contains options delete, open and close
- Search Timetables Dialog – Contains options save, open and close
- In both cases, ‘Open’ opens the PDF Renderer activity

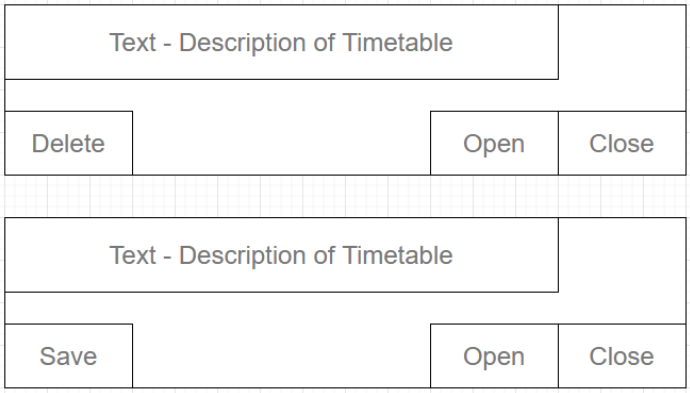

Figure 8 - Dialogs Wireframes

About and Help -

- As they are identical except for content, they are both listed here.
- Both have the same navigation functionality as all the other pages
- Content will differ but will primarily only be text and images to support the user.

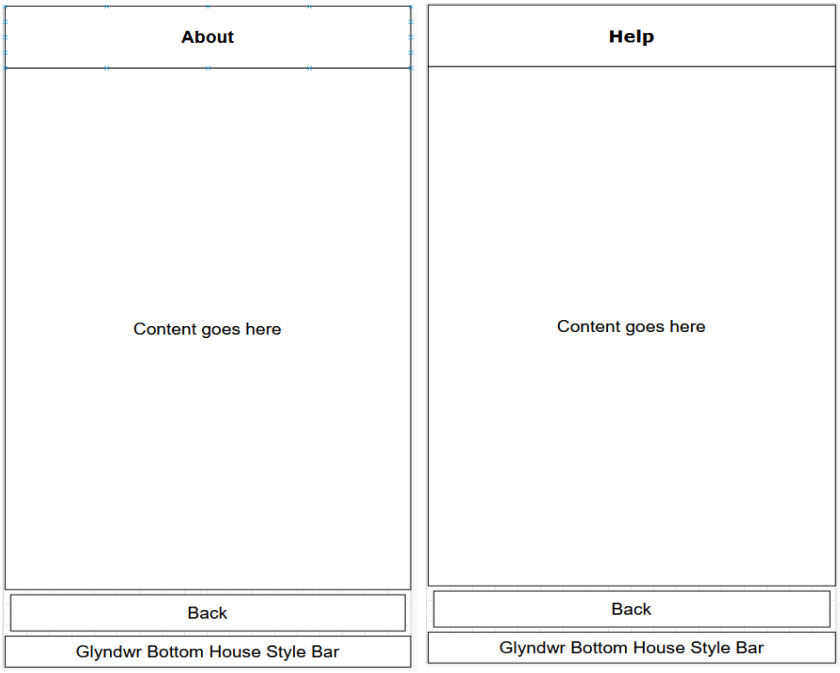

Figure 9 - About and Help Wireframes

5.1.2 Screenshots of in development interface

Main Activity – Home / Menu

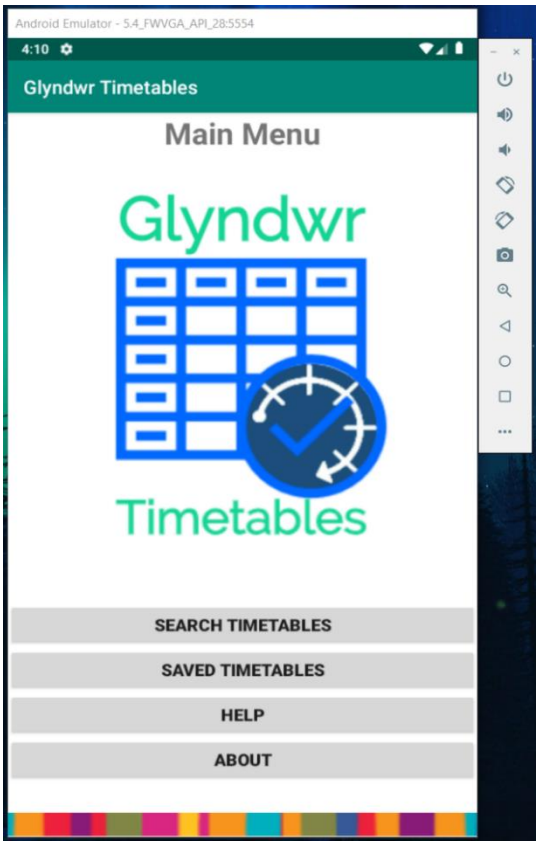

Figure 10 - Screenshot of Home Screen

Search Activity -

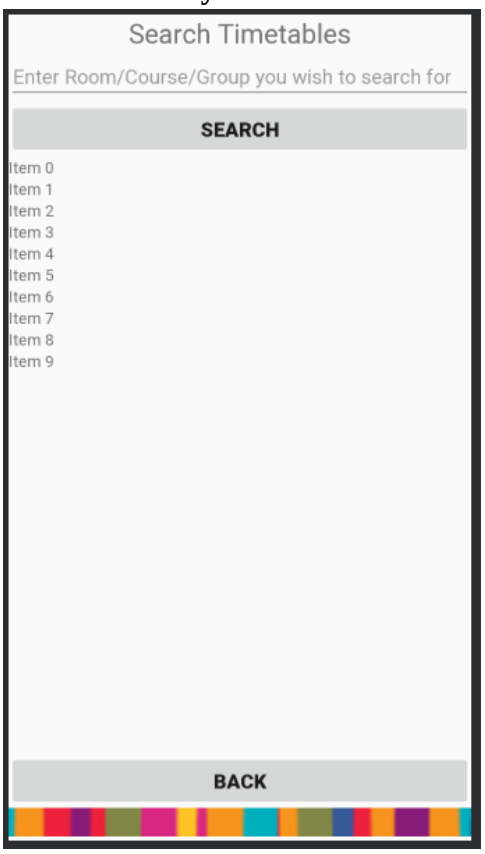

Figure 11 - Screenshot of Search Activity

Saved Activity -

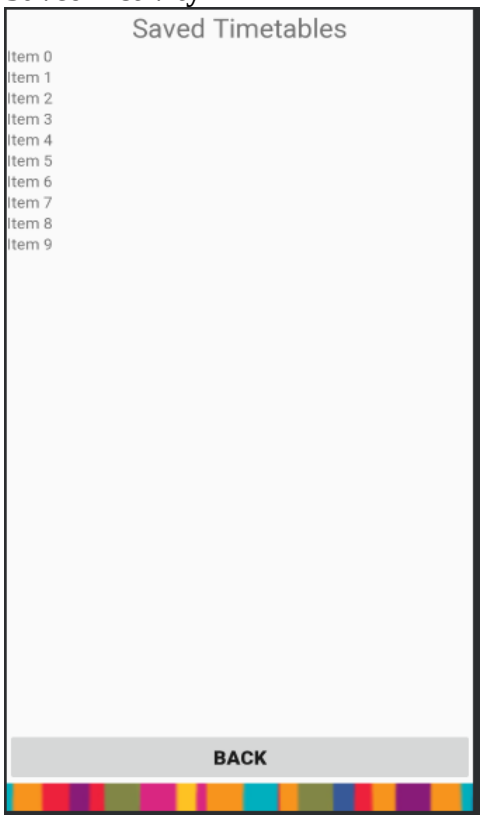

Figure 13 - Screenshot of Saved Activity

Help Activity –

Figure 12 - Screenshot of Help Activity

About Activity –

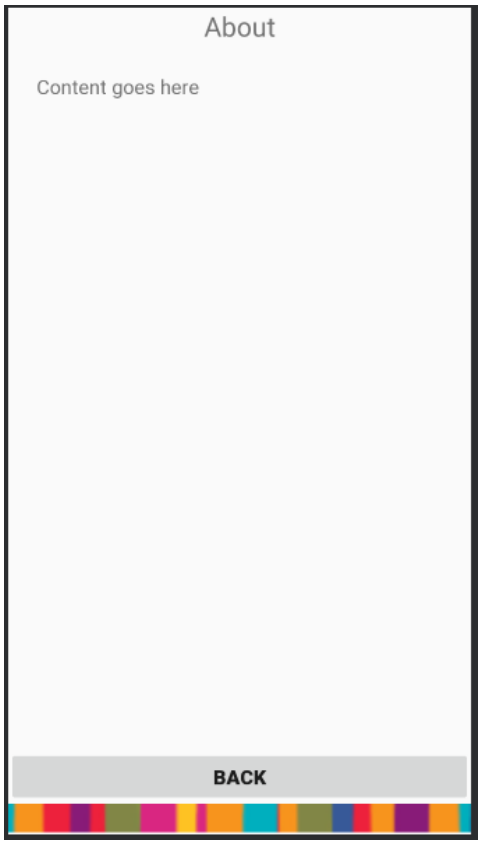

Figure 14 - Screenshot of About Activity

PDF Renderer Activity –

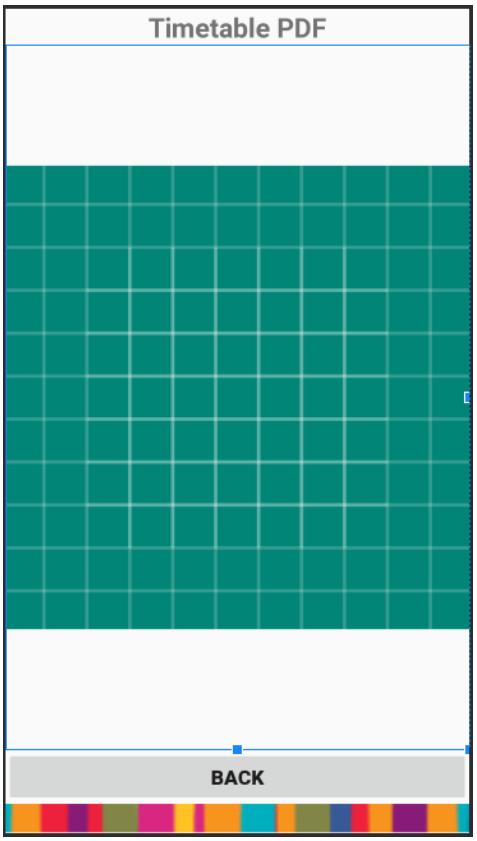

Figure 15 - Screenshot of the PDF Renderer Activity

Saved Timetables Dialog -

Figure 16 - Screenshot of the Saved Dialog

Search Timetables Dialog -

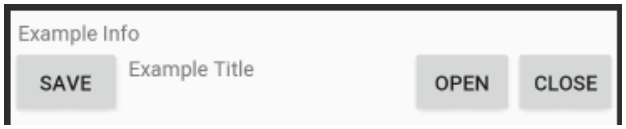

Figure 17 - Screenshot of the Search Dialog

5.2 – App logic and Mechanics

5.2.1 - Jackson Structured Programming

Main Activity – Home / Menu

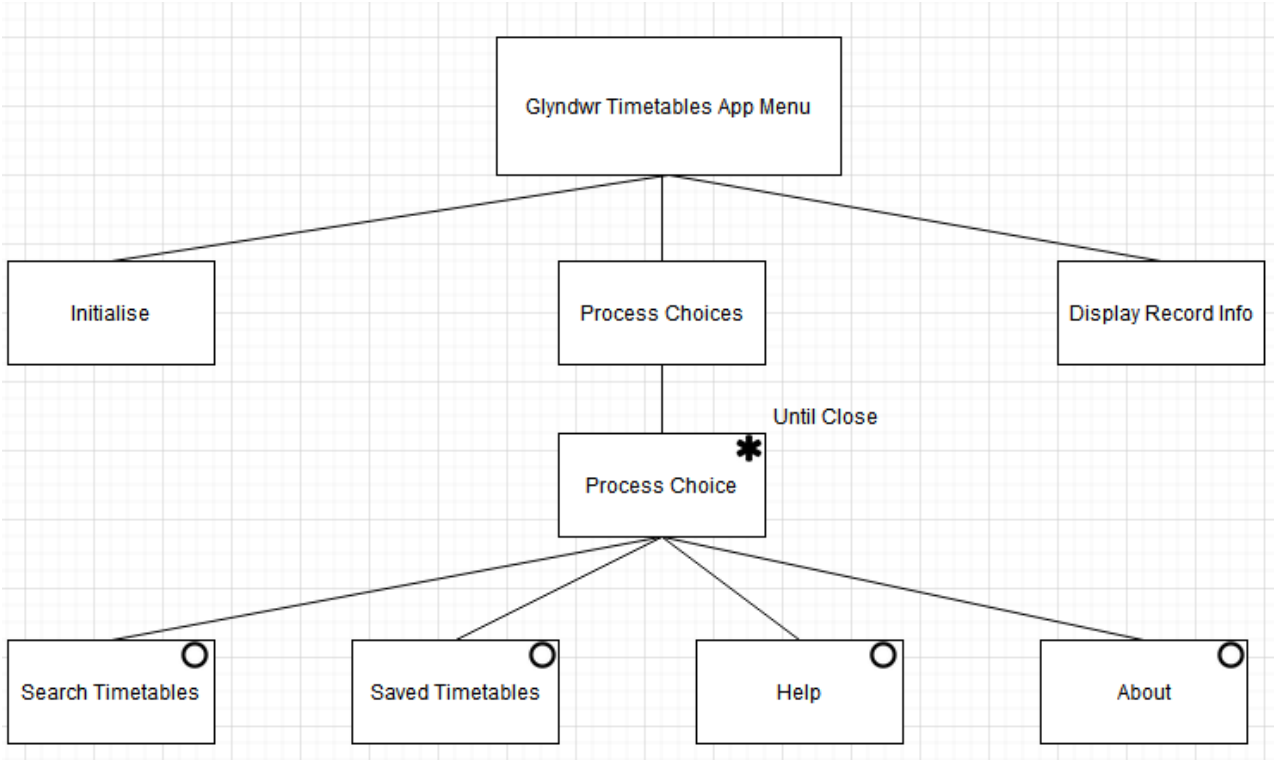

Figure 18 - Main Menu JSP Diagram

Search Activity –

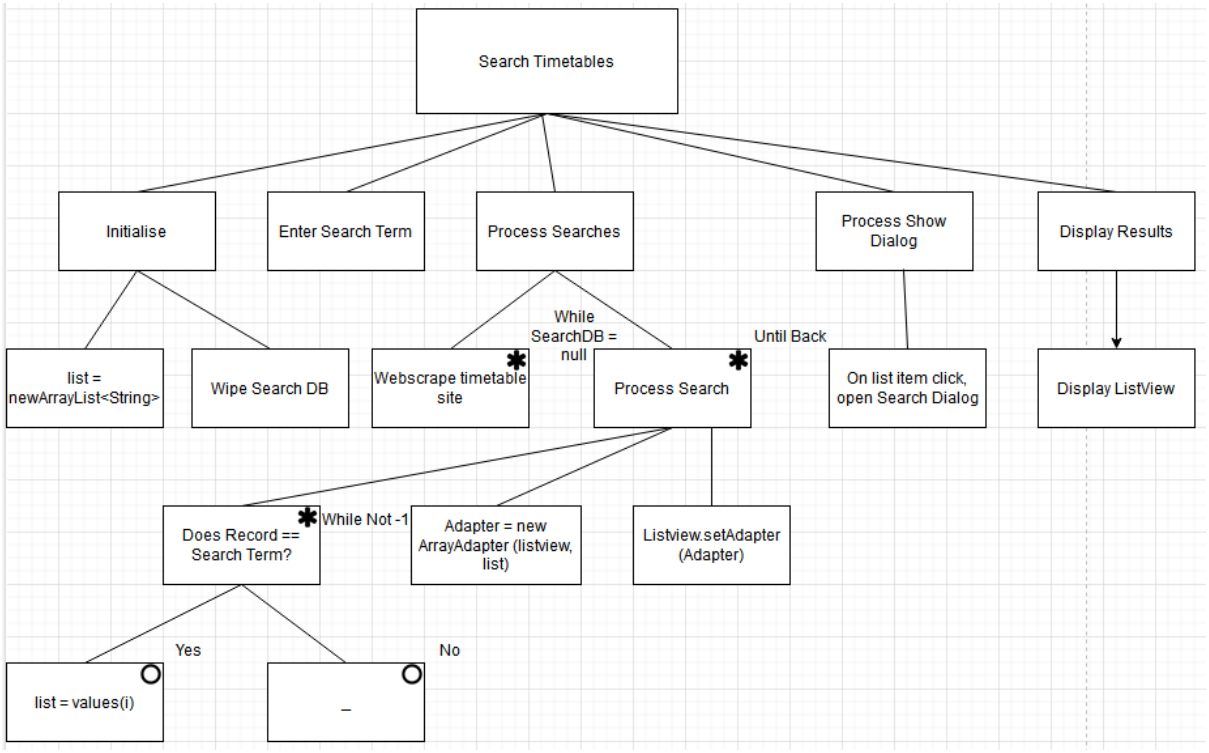

Figure 19 - Search Activity JSP Diagram

Saved Activity –

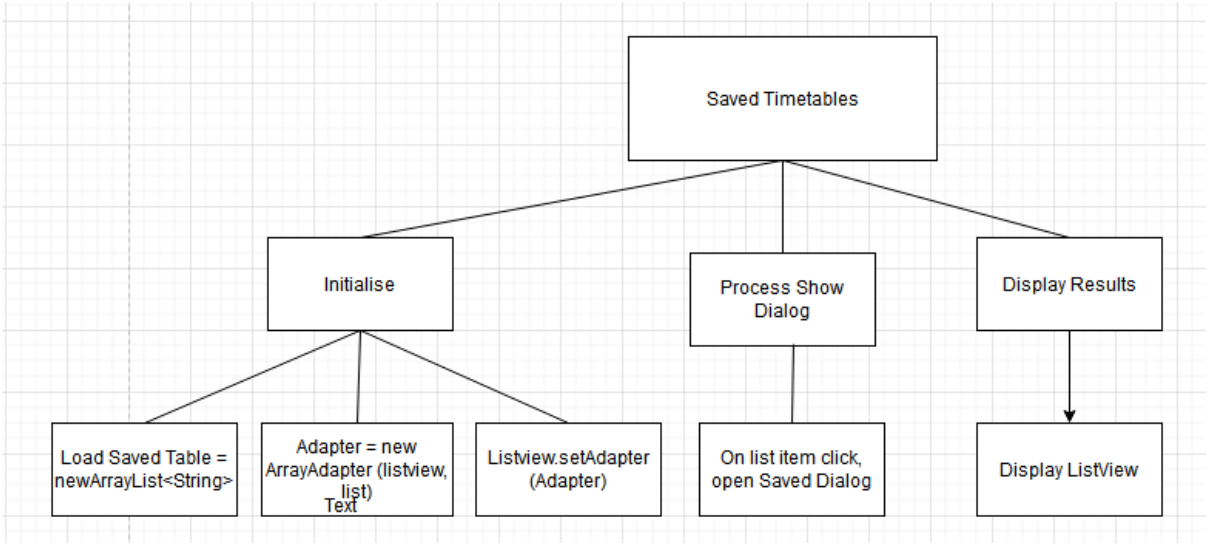

Figure 20 - Saved Activity JSP Diagram

Help/About Activities –

JSP not necessary as these activities have no app logic and only have one choice to navigate back to menu on both with the ‘Back’ button.

Search Dialog –

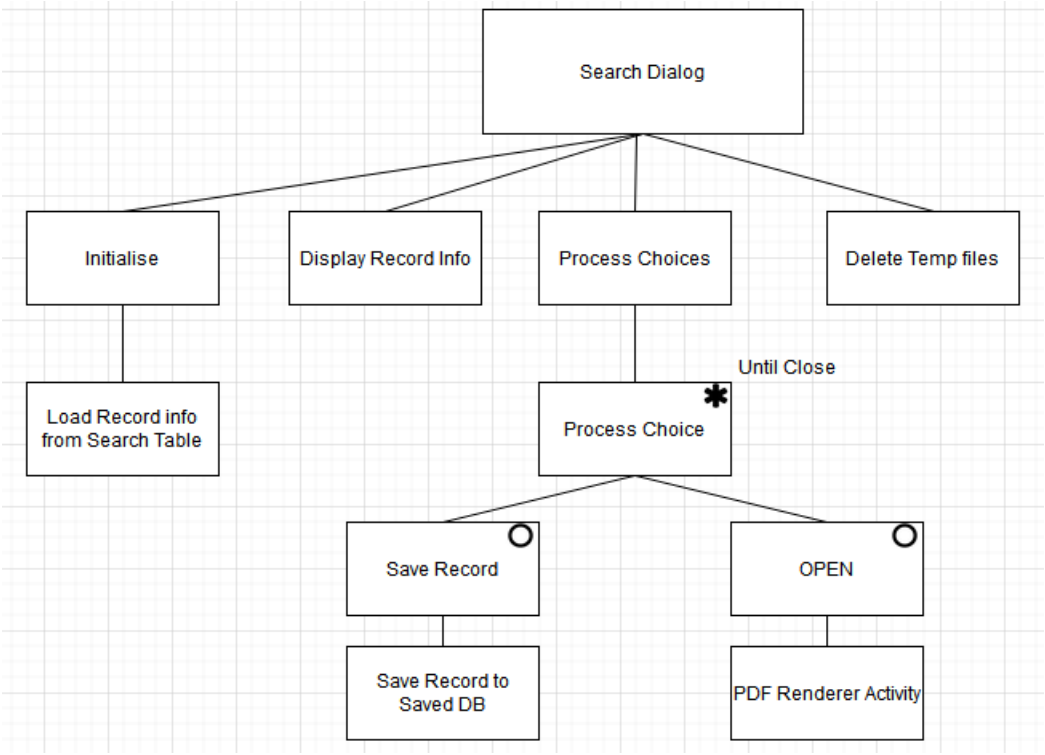

Figure 21 - Search Dialog JSP Diagram

Saved Dialog –

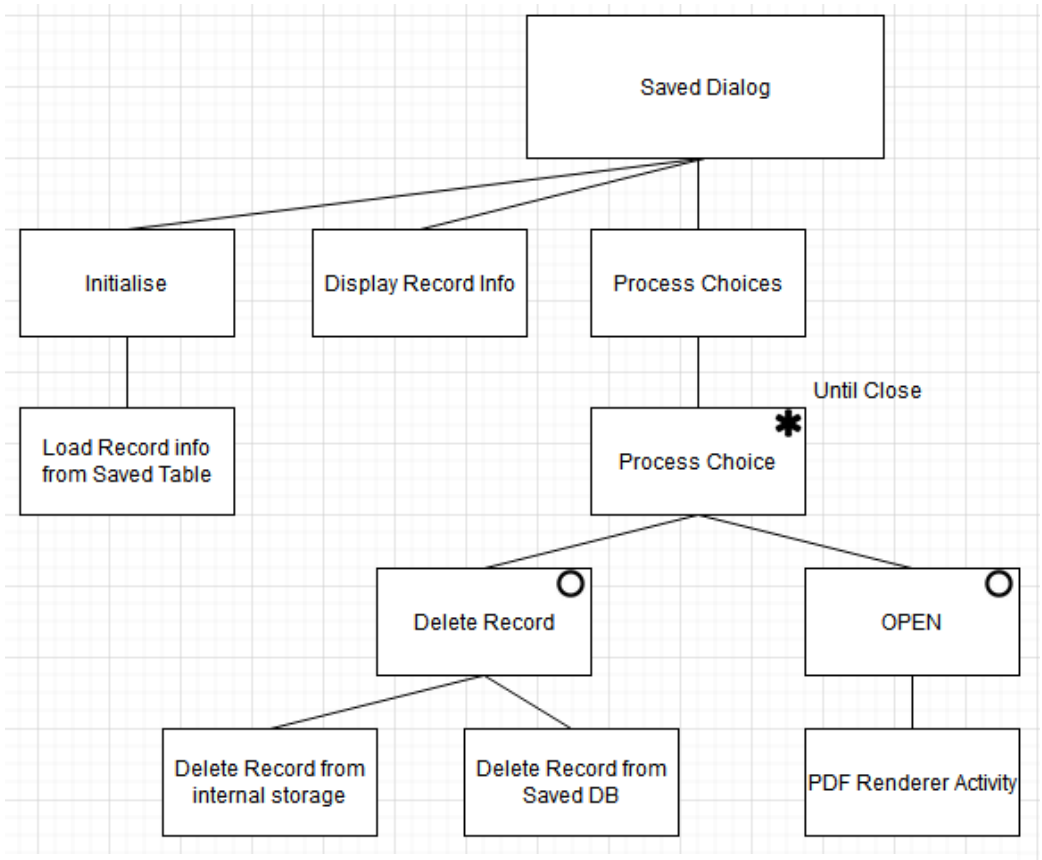

Figure 22 - Saved Dialog JSP Diagram

Web scrape Process –

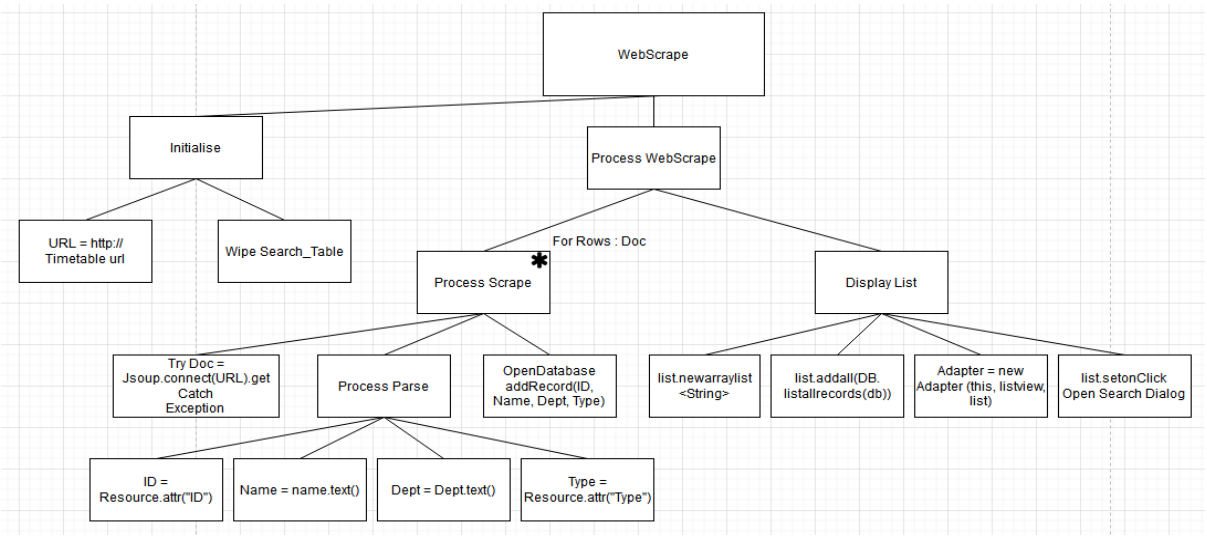

Figure 23 - WebScrape Process Diagram

PDF Rendering Process -

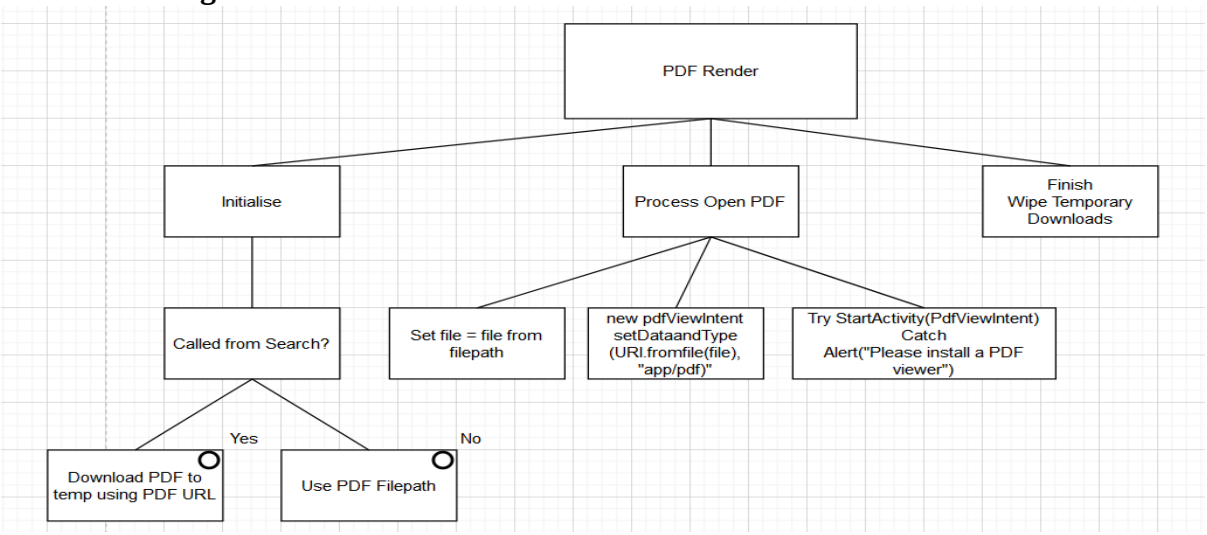

Figure 24 - PDF Render Process JSP Diagram

Open Database Process -

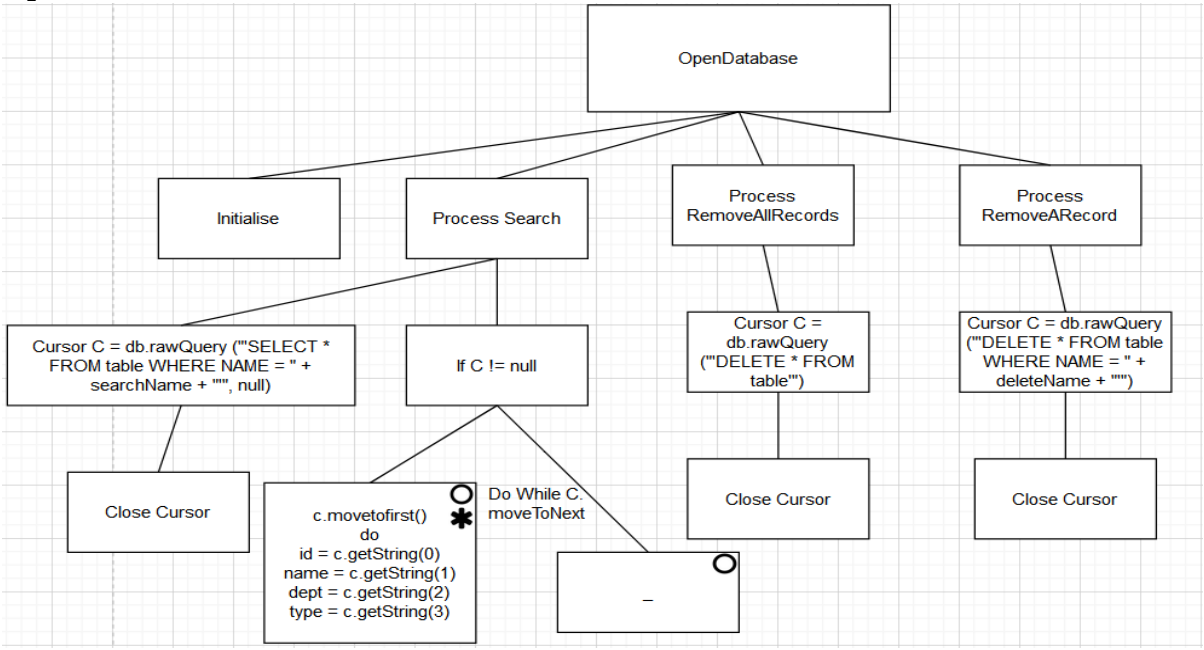

Figure 25 - Open Database JSP Diagram

5.3 – Database Design

5.3.1 - Data Flow Diagrams

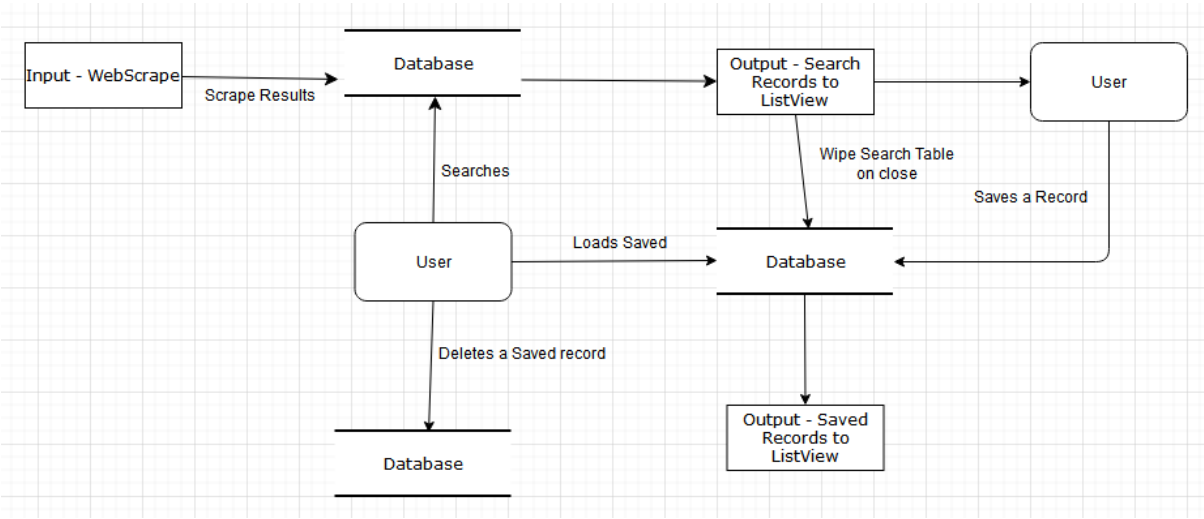

Figure 26 - Dataflow Diagram

5.3.2 – Table designs

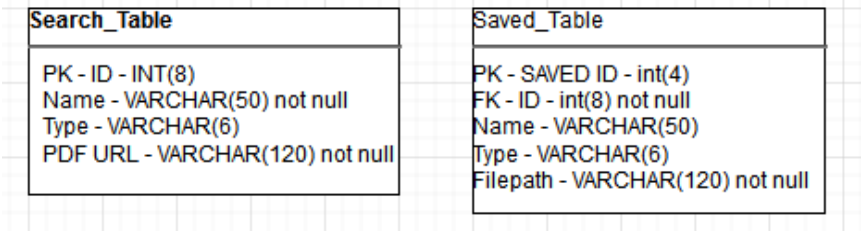

Figure 27 - Table Designs

6. Development & Rapid Construction –

6.1. Phase 1 Prototype

At this stage, thanks to the design process and RAD, the user interface features are already implemented and for the rest of this prototype only these features/App logic need implementing;

- database,
- web-scraping functionality,
- PDF Rendering Process,
- Search functionality,
- Search Activity list view,
- Saved Activity list view,
- Search dialog functionality,
- Saved dialog functionality.

Functionality mainly relies on the tables in the database which store web-scraped results and saved timetables so I shall start with the database implementation.

6.1.1. Database implementation

The database was created using SQLiteStudio (Ver 3.1.1). SQLite 3 works really well with android applications and this IDE was a good choice as it offers easy design tools to create databases, tables and views.

From here, I imported the database to the Assets folder of the application in Android Studio. This provides the app with access to the database. However, it does not provide READ or WRITE access to the tables. We can create a Writeable database by first copying the 'timetable_db' from our Assets folder using InputStream. So we don't overwrite anything, it is wise to first check for an existing database folder for the app, if there are none then a directory can be made for the copying to happen. We can then use OutputStream to write a file using the "Output Filepath" and "Output filename". Once the copy is finalised, we must flush and close both 'In' and 'Out' streams.

Now that we have a writeable copy of our original database, it is a good idea to create a new class we will call 'OpenDatabase' to handle our database functionality. This includes;

- Removing all records from the 'Search_table' and adding records to 'Search_table' during Web-scrape,
- Add records to 'Saved_table' by saving in search dialog,
- Search records in 'Search_table' for the search activity,
- Removing a record from the 'Saved_table' by deleting in the saved dialog,

6.1.2. Web-scraping Functionality

As mentioned previously, the web-scraping functionality will be implemented with the help of the Jsoup library. More specifically, Jsoup version 1.13.1. To do this, the Jsoup

implementation must be added as an entry to the ‘build.gradle (Module: app)’ as “**implementation 'org.jsoup:jsoup:1.13.1**” .

Once this is done, placement of the web-scraping logic must be considered. Although I had originally designed the placement to be in the Search Activities ‘OnCreate’, I found that placing it to run in the main activity on launch was much preferred. This gave the app chance to scrape all 1400+ records and add them to the ‘search_records’ table. This is preferable as the user would not need to then wait and refresh their list, but rather as they go to open search it runs in the background.

The scraping is carried out by a method called ‘scrapeForTimetables’ which is called after Database Initialisation and before Setup Controls. The method first wipes the ‘search_records’ table by calling the ‘removeAllRecords’ method of our OpenDatabase object, this avoids multiple records in the same table and allows a fresh scrape of new content.

The scraping is then done in a new runnable thread. Here we use ‘try catch’ to attempt to create a Jsoup document. Then initialising it to the website document (in this case XML) using **Jsoup.connect(URL).Get();.** Once the document has been retrieved, I have chosen to work through the document by looping through “resources”. As each entry in the table are under the class “resource”. To avoid null entries, I assign a String ‘check’ to **resource.attr(‘id’)** to check in an if statement. If true, it exits that iteration of the loop with **continue;**. If false, I gather four variables; Name, Dept, Type and PDF URL using **resource.select()** in the case of text from tags and **resource.attr()** when getting attributes from tags. Once gathered, it is easy to add to the ‘search_records’ table using the ‘addSearchToRecord’ method of our OpenDatabase object. If the Try/Catch fails, the exception will be printed to Logcat.

```
protected void scrapeForTimetables()
{
    // Wipe Records Table before Web-Scraping
    sqh.removeAllRecords(db);
    Log.w( tag: "SEARCH_RECORDS_WIPED", msg: "Search Records Table wiped for fresh web-scrape");

    new Thread((Runnable) () → {
            try
            {
                final Document doc = Jsoup.connect( url: "http://timetables.glyndwr.ac.uk/GenericTimetables/finder.xml").get();

                Elements resources = doc.select( cssQuery: "resource");
                Log.w( tag: "SCRAPE", msg: "Starting to scrape");
                for (Element resource : resources)
                {
                    String check = resource.attr( attributeKey: "id");
                    if (check == "")
                    {
                        continue;
                    }
                    else
                    {
                        String Name = resource.select( cssQuery: "name").text().replace( target: ",", replacement: "").replace( target: "'", replacement: "");
                        String Dept = resource.select( cssQuery: "dept").text().replace( target: ",", replacement: "");
                        String Type = resource.attr( attributeKey: "type").replace( target: ",", replacement: "");
                        String PDF_URL = resource.select( cssQuery: "link.pdf").attr( attributeKey: "href");
                        sqh.addSearchRecord(db, Name, Dept, Type, PDF_URL);
                    }
                }
                Log.w( tag: "SCRAPE SUCCESSFUL", msg: "Scrape is completed");
            }
            catch (IOException e)
            {
                Log.w( tag: "SCRAPING ERROR:", e.getMessage());
            }
    }).start();
}
```

Figure 28 - Jsoup Web-scraping Timetables Code

6.1.3. Search Activity Functionality, Dialog and list

The majority of the logic in the search activity is carried out in the 'Setup Controls' method, which is ran after Database Initialisation. Here the interface is setup and linked. The list is constructed and populated using the 'ListAllSearchRecords' method of the OpenDatabase Object. An adapter is created using the search context, simple list layout and the list. The adapter allows us to link our list to the listview of the interface.

The Dialogs are started with a list items onItemClick. Once the layout is created, the item at the position clicked is stored and the details are split into 4 variables using .split(','); Name, Dept, Type and PDF_URL. These are used to populate the title and info sections of the Dialog layout. Because of the size of the data can be particularly large, I have made another alteration to the initial design here too.

Figure 29 - Redesigned Search Dialog

The three buttons logic are as follows;

- Open – Creates 3 variables; Download_URL, PDF_FILEPATH and Condition (in this case it is "Temp"). The variables are passed to the 'DownloadTask' class which in itself is another addition to the app's logic design. This downloads the PDF to a temp directory and opens the PDF Render activity for Rendering.
- Save – Same as open except doesn't have PDF_Filepath and Condition = "Save" so that when the DownloadTask class is called, it saves the file to a permanent directory. Also the parameters for the dialog title/info are reused to add a Saved record to the 'Saved records' table using the 'addSavedRecord' Method of the OpenDatabase object.
- Close – Here the dialog is dismissed as well as all files in the temp directory being wiped. (i.e. if a user opened a pdf and not saved then the temp will slowly fill up unless it was wiped regularly)

The search feature is implemented by the use of the 'SearchAllSearchRecords' method of the OpenDatabase object. Here the users input from the EditText is passed through as a search term where the method searches the table for 'LIKE' on; Name, Dept or Type. The list is refreshed by wiping the list, re-populating it with the results and re-linking the adapter. I altered the search capabilities here to allow user to search for different things like department or type as it improves the search features usability.

6.1.4. Saved Activity, Dialog and list

Saved activity has the exact same logic for the list and dialogs except for the loss of search functionality, and only having delete capabilities in the dialog. Also, the open PDF does not call a DownloadTask and save to temp here but rather just sends the filepath to the PDF render activity. In this activity, I have come across a problem which no solution I have tried has worked. The problem stems from being in the dialog and making alterations to the dataset

that the list uses. Once a record is deleted and the dialog closed, the record is still listed in the listview as the dataset is not refreshed. As the data is still loaded, this allows the quirky fact that the user can still open the dialog for that record. The PDF will not open though as the file has been deleted.

6.1.5. Download Task Process

The download task process is an entirely new aspect to the design that I hadn't originally considered too much. This was because I initially figured I could just write a file using InputStream and leave it at that. However, this was short sighted as consideration was required for both the temporary storage of pdf files when viewing them from the search dialog, as well as for more permanent storage for when saving the pdf files.

On this basis, the new download task class will take in the context, PDF URL and the condition in which it is to be stored (temp or saved). This class will be called in the 'Open' and 'Save' button listeners of the search dialog.

Upon calling the class, it creates a download File name and opens a progress dialog for the user to show progress (i.e. 'Downloading…'). The 'doInBackground' holds the main logic and will check the initial condition previously mentioned. The logic for both 'Save' and 'Temp' is identical except for file-paths as to where the files will be stored, this will make it easier to wipe the temp directory upon completion.

The logic involves defining a URL to the 'downloadURL' we passed in, connecting using '**HttpURLConnection c = (HttpURLConnection) url.openConnection();'** and **'c.connect'**. Once the URL is connected, we check for the folder destination. If it doesn't exist we can create the folder for it to be stored in. Once the folder is ready, we create the file using the folder path and download filename. As before, we check if the file already exists. If it doesn't we use 'FileOutputStream' and 'InputStream' to write the file. All of this should be done in a Try/Catch to allow for errors with reading in the URL and writing the file/creating the directories.

On post execute, the progress dialog is updated to either show 'Downloaded successfully' or after a period of time show 'Download Failed.' Depending on whether or not the output file previously made is null or not.

6.1.6. PDF Rendering Process

The PDF Rendering process could have been implemented in two ways; Rendering the pdf file in-app and allowing the user to cycle through rendered pages or by opening the file in an installed PDF viewer that the user may have installed. Either way, as mentioned previously both have their limitations – the former which I have chosen being that only devices from Android 5.0 Lollipop and above can support this code. Being that distribution has 'Lollipop' as cumulatively ~96% of the android userbase though, this should not be a problem for nearly all users.

This process proved a little tricky and temperamental with opening temporary files. Eventually, I found that by implementing a UI handler on the search activity and having the UI wait for 1000ms.The DownloadTask could fetch the temporary file to its directory ready before entering the Rendering activity.

The PDF Rendering class uses googles ‘PdfRenderer’ functionality. Firstly, in order to even open the file we need to know where it is stored. To do this, I used ‘intent.putExtra()’ in the ‘Open’ buttons of both dialogs. This allows me to send the filepath between both activities to be used.

After getting the filepath, I initialise a ‘File’, ‘ParcelFileDescriptor’ and ‘PdfRenderer’ objects. Since all of the timetables PDF’s are similar on every page, I need only render the first page of each for the user. Because of this, I can initialise a ‘rendererPage’ to the first page of the pdf file. Once initialised, I need to get the height and width of this page in order to create a Bitmap to show the timetable as an image in the image view on the layout. Once rendered, I set the bitmap by calling ‘**pdfView.setImageBitmap(bitmap);**’. Remember to close the page renderer, pdf renderer and file descriptor after the bitmap has been linked as not closing them can cause serious issues.

6.2. Testing

To properly evaluate this app against its original goals and scope, it needs to be properly tested to ensure correct functioning as intended. There are numerous methods open to developers and testers that can be carried out to ensure proper functionality [29]. Most appropriate of the functional testing types to this small scale app are Unit tests and Integration tests. The former dealing with individual features on their own and ensuring they properly function and the latter dealing with the app as a whole and whether all the units work together as intended. Functional tests prove useful to ensure features work correctly but this does not always determine whether the user experience will be of a high quality or not. As is necessary in RAD that user feedback is constant throughout development, this small scale project did not require constant feedback. Rather, this feedback can be achieved in a non-functional test such as User testing.

6.2.1. Unit Testing

The first unit is the web-scraping functionality. This should occur on load, an effective scrape is indicated in the Logcat window with the Tag ‘SCRAPE’. This is verified by going to the search activity and seeing the populated table of records.

```
2020-04-23 00:32:26.690 7634-7729/com.example.glyndwrtimetables W/SCRAPE: Starting to scrape
2020-04-23 00:32:27.785 7634-7729/com.example.glyndwrtimetables W/SCRAPE SUCCESSFUL: Scrape is completed
```

Figure 30 - Web-Scrape Logcat

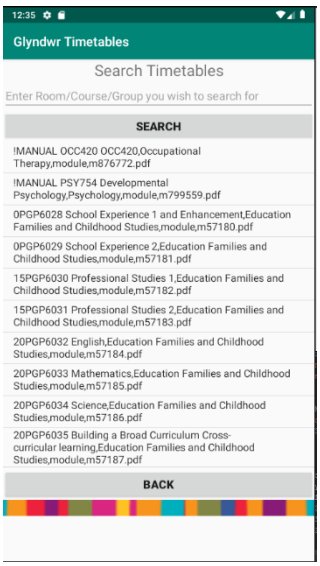

Figure 31 - Scraped Records

Since we are using the search records to verify the successful web-scrape, we can also test the search capability as a unit. In this, the search should allow users to search for Name, Type and Department. On loading the activity, the logcat displays the request for search table data.

```
2020-04-23 00:35:27.894 7634-7634/com.example.glyndwrtimetables W/SEARCH_RECORDS_REQUEST: Table successfully loaded
```

Figure 32 - Loading Search table data

Figure 35 - Name Search

Figure 34 - Type Search

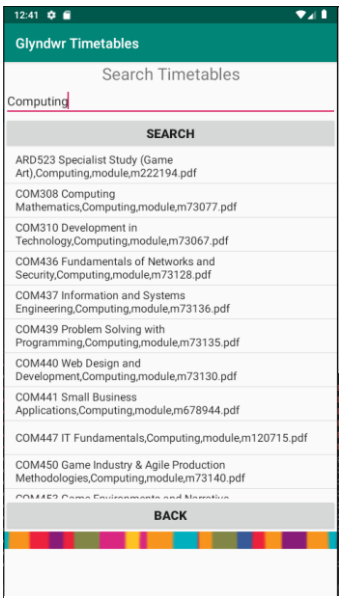

Figure 33 - Department Search

```
2020-04-23 00:40:26.239 7634-7634/com.example.glyndwrtimetables W/SEARCH RECORDS: searchTerm = Internet and Mobile
```

Figure 36 - Name Search Logcat

```
2020-04-23 00:41:24.618 7634-7634/com.example.glyndwrtimetables W/SEARCH RECORDS: searchTerm = group
```

Figure 37 - Type Search Logcat

```
2020-04-23 00:40:55.737 7634-7634/com.example.glyndwrtimetables W/SEARCH RECORDS: searchTerm = Computing
```

Figure 38 - Department Search Logcat

Another unit we can test that is found within the search activity is the Search Dialog. The dialog contains many different smaller processes, however it must be considered an integrated test type and as such we will test it later.

The other major unit to test is the saved activity. This is accessed by the main menu with the 'Saved Timetables' button. On loading this activity, the logcat will display an entry showing the request for saved records to display the saved records in a list.

```
2020-04-23 00:55:55.941 7634-7634/com.example.glyndwrtimetables W/SAVED_RECORDS_REQUEST: Table successfully loaded
```

Figure 39 - Loading saved records to populate list

Again, as with the Search Dialog, the Saved Dialog is considered an integrated type test as it has many processes tied together in its running.

6.2.2. Integration Testing

The first major integrated test is the navigation of the application. This involves testing each of the buttons on the main menu to navigate to each of the activities, using the item 'onClicks' to access both dialogs and the open/close buttons found therein to navigate to the PDF render activity and close the dialogs respectively. Finally, the back buttons found on all the activities should allow the user to navigate back to the original menu. All of which, after extensive testing here and later testing in user testing have proven to be successful.

Following the successful navigation testing, the next integration test is to open temporary PDF files. This is done by selecting a search item, and selecting 'Open'. During this, we can find evidence of the dialog properly displaying record information, creating a temp directory, temp file, the filepath of the temp file and the render of the pdf file.

Figure 40 - Dialog Title and Info

```
2020-04-23 01:09:08.223 3185-5471/com.example.glyndwrtimetables W/Download Task: URL Connected
2020-04-23 01:09:08.235 3185-5471/com.example.glyndwrtimetables W/Download Task: Temp Directory Created.
2020-04-23 01:09:08.236 3185-5471/com.example.glyndwrtimetables W/Download Task: Temp File Created
```

Figure 41 - Download Task Logcat

```
2020-04-23 01:15:30.464 3185-3185/com.example.glyndwrtimetables W/TEMP DIRECTORY: Contents wiped
```

Figure 42 - Wipe contents on Dialog Close

Figure 44 - PDF Render Activity

```
2020-04-23 01:09:09.403 3185-3185/com.example.glyndwrtimetables W/FILEPATH =: /data/data/com.example.glyndwrtimetables/temp/m876772.pdf
```

Figure 43 - Temp PDF Filepath

The last integration test is the saving and deleting of PDF files. In this, the logcat shows us the file being saved, file being deleted and the filepath of the saved file. The saved activity will also list saved records.

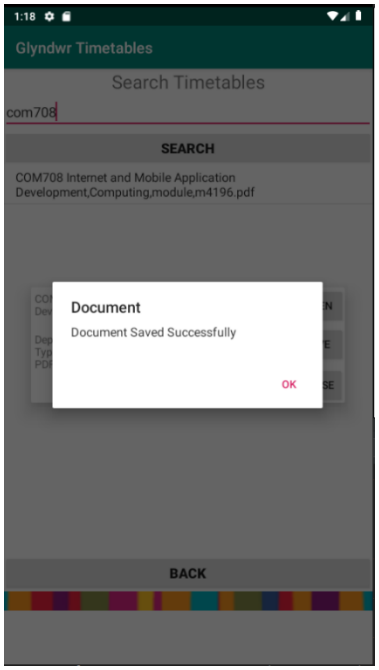

Figure 45 - Document Saved ProgressDialog

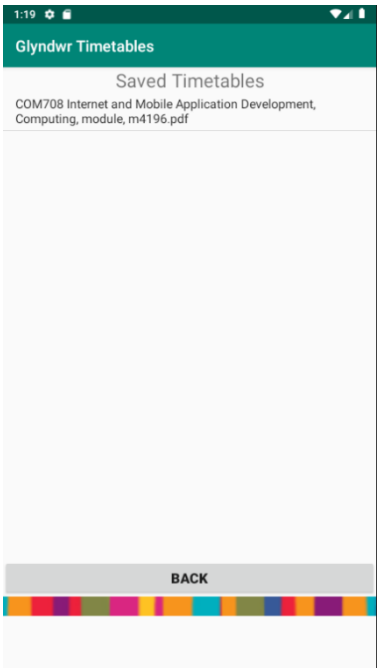

Figure 46 - Saved List

```
2020-04-23 01:18:09.484 3185-6093/com.example.glyndwrtimetables W/Download Task: URL Connected
2020-04-23 01:18:09.485 3185-6093/com.example.glyndwrtimetables W/Download Task: Save Directory Created.
2020-04-23 01:18:09.485 3185-6093/com.example.glyndwrtimetables W/Download Task: Save File Created
```

Figure 49 - Download Task Saved file

```
2020-04-23 01:19:34.196 3185-3185/com.example.glyndwrtimetables W/REMOVED FROM SAVED: Record = COM708 Internet and Mobile Application Development
2020-04-23 01:19:34.196 3185-3185/com.example.glyndwrtimetables W/RECORD REMOVED: Saved Record removed = COM708 Internet and Mobile Application Development
2020-04-23 01:19:34.196 3185-3185/com.example.glyndwrtimetables W/DELETED RECORD: Record = COM708 Internet and Mobile Application Development, PDF FILE DELETED
```

Figure 48 - Delete Record Logcat

```
2020-04-23 01:21:56.927 3185-3185/com.example.glyndwrtimetables W/FILEPATH =: /data/data/com.example.glyndwrtimetables/timetables/m4196.pdf
```

Figure 47 - Saved File Filepath

6.2.3. User Testing

User testing is considered a non-functional test. This is because rather than testing functions specifically, we aim to use the software as any end-user might. That may be erratically, not knowing of how to properly use the app as intended (i.e. bad inputs on search).

With my initial own user testing using the Android application package file (apk) on an android device. The apk is the closest method of user testing feasible as the user would be using the final release version of this file on their phones to use the app. I have come across a few quirky issues of my own with some of the apps functionality. Not enough in itself to stop the app reaching its intended goals but enough to be noticeable somewhat.

Search Function -

The first component I extensively tested was the search function on the search activity. Here, despite all the various entries I attempted such as; “Timetable, 123, None, != module, -1 or leaving it blank” all retrieved expected results. Though, I suspect that with more sophisticated knowledge of SQL injection attacks, this function may be susceptible to those types of attacks as they directly interface with the database using SQL Raw Queries.

Search Dialog -

The next feature I tried to test thoroughly was the search dialog. Here I used all three buttons as initially intended and they worked fine, either saving the record, opening the pdf render activity or closing the dialog. I found that I could select another record if I was quick, tapping in rapid succession allows the dialog to close and another record to open in time. This, in tandem with the open and save buttons either meant the wrong record was saved or the pdf was not rendered as it was not stored correctly. The app did not crash as both the downloads and renders still functioned correctly and was in try/catch blocks to catch exceptions. I suspect that the closing of the dialog by tapping anywhere else on screen or using the back button could be used to circumvent the temporary file wiping. This could mean that a user could inadvertently fill up temporary storage. Also, once the dialog is opened, if the user turned their phone to landscape the dialog will close.

Saved Dialog –

After initial testing with the search records dialog, I found that I had saved multiple copies of the same record in the database. This had somehow circumvented a check in the download task which checks if the file already exists. I now realise that there is no condition for what happens if it currently does and therefore the file will be made again. This results in the first quirk of the app, upon deleting a record, all copies of that record will be deleted in one. The list however, only refreshes once the user has backed out or once the user reopens the dialog rather than refreshing on closing the dialog. I initially knew about this problem but all of the solutions I attempted did not seem to fix the issue. One suggestion I had not implemented however was a UI handler. I suspect by having a UI handler listening for a particular thing like the record delete or the dialog close and then refreshing the listview may have worked.

Navigation and Appearance –

All navigation was intuitive, using the back button instead of the provided in app button did not seem to effect any app functions. The UI style throughout is not always consistent. On some pages, the lower bar and button sit at the bottom and in others it is slightly higher. Perhaps using a layout like constraint or relative layout may solve this problem as using linear layout didn't provide an easy method of locking elements to the bottom.

7. Evaluation –

When evaluating a product. It is important to consider how best to do so. Evaluating it from the viewpoint of an end user we can determine how well the app will perform based on theses measures; Findability, Utility, Searchability, Browsability, Buyability and Overall Design [30]. Findability does not apply to this project in particular as this app will not make its way to market. Findability is concerned with how easily the user can find the app based on search terms, appropriate app listings and its search ranking in app marketplace. The first aspect that applies to the evaluation from this is 'Utility', this is very important as after first time use a lot of apps may be uninstalled. Good utility means that the app proves a great tool to the user. We can tie this aspect to the functional goals of the app laid out in scope. From our testing, its determined that the functionality of the app has been achieved. This is because the user can search for timetables, save records, download them and display them with no issue occurring. As its intended audience is only Glyndwr University students, it provides great utility to them. However, since Celcat provide their services to many different companies it may prove useful for further development for the app to scrape multiple different sources and allow the user to pick. From the research, we determined that

a good source code could be licensed and re-skinned for other purposes. In this, the base could be used for many different services that use Celcat.

After utility, the next major aspect of evaluation is 'Searchability'. This is the amount of effort required on the users part to find what they are looking for. Either in how much interacting with the interface they need to do to get what they want. Or in terms of data input and whether they can use features like predictive text and autocomplete. The only aspect of the app that relates to this is the search timetable feature. This input, as seen from testing shows that multiple different entry types show acceptable results. Predictive text is allowed as an entry but autocomplete is not set up as it is not necessary. Maybe to show previous search results. Like I suggested in the testing phase, this search feature may be susceptible to SQL injection attacks.

The 'Browsability' functions perfectly as originally intended as testing proves navigation is a breeze in order to get where you're going. Like with findability, 'Buyability' is not necessary to evaluate as there are no in app purchases or payment functions that could be used. For overall design, the house style is consistent and layout makes sense. The interactivity is somewhat basic but as its functional and works, this hardly seems an important point as the app is utilitarian in nature.

The only feature worth mentioning that affects the design aspect is the PDF rendering. Although it works as intended without errors, rendering one landscape PDF page on a natively portrait platform makes it difficult to view the PDF. The render activity can be turned landscape to view it properly, but if the PDF function was implemented as opening a PDF reader installed on the phone then the user would be able to interact with it in a Fullscreen interface and pinch to zoom.

8. Critical Reflection –

As for the project itself, the app was made in good time well before its set deadline. This was in part due to the effective planning allowed by effective use of methodologies. By using RAD and parts of SDLC, the app was scoped out based on assignment requirements and my own end goals during the requirements gathering and analysis stage. Once this was complete, the design could be based around it to appropriately cover all the intended functions. Like RAD suggests, the design was based around User feedback but since it's a small scale project, I played the role of user in creating it. This could be a small issue either due to personal bias as the creator of the app or simply by overlooking what other users may deem important or not. Based on its evaluation and testing though, I believe I had achieved a good user interface despite that potential bias.

The designs were fully flushed out to include; wireframe diagrams, in-development screenshots, Jackson Structured Programming Diagrams, Data-flow diagrams and table designs. This covered all possible aspects one should consider when developing any software. The first two diagrams helped form a good user interface for the logic to be attached to. The JSP diagrams described the logic necessary for the different functions such as Web-scrape, Database, search and saved activities.

The next stage of RAD and SDLC is the development stage. Since it was a smaller scale project, Scrum and other more common agile methodologies seemed unnecessary and could even be more burdensome than the rapid approach I took. This is because RAD streamlines many of the processes, so smaller, more skilled teams can work better on narrow focussed projects like this one. Since I had already developed the interface as part of the design process and all of the logic was described in JSP. The design process was fluid and fast. By

compartmentalising the designs to specific areas, I could work on a section by section basis starting with the database functionality and writing code based of original specifications. The Open database method would change over time as the underlying table that was developed also had changes made to it.

Each section allowed me to iterate the previous section up to par and any changes made in development were able to be noted during the 'blog style' development section. After database functionality, the design called for web-scraping which went smoothly and highlighted the first database alterations I needed. To give the tables their own auto-increment ID's that don't get affected whatsoever. This extra column allowed for the code to function correctly as designed without throwing errors over using the Resource ID's from the online table data. Once the web-scraping worked, the search list, search function and dialog were the next to implement. This was able to be reused in terms of logic for the saved section too.

Finally, once the lists could populate with the correct table information and the search worked I started on the dialog logic. Here is one of the setbacks that with hindsight, I could see was not well thought out. Although the dialog was easy to implement and the data could be pulled from the item clicked, the design had not anticipated how to effectively download PDF files using the scraped URL and account for both saved files and temporary files. Since I knew it was related to 'Inputstream' however, I was able to do a bit of research and come up with a workable solution to downloading files to a directory of my own making in external storage. From here, I adapted my solution to use the internal directory like I do for my database and also pass through another parameter which indicates whether the file is temporary or to be saved. The code within 'DownloadTask' is duplicated to cover both parameter inputs with slight variations in save file path.

In a change from the initial app idea, I had thought to use PDF scraping and format data into a better format for the portrait style app. This proved out of my depth and completely unnecessary as many of the options available use built in PDF readers on phones to display PDF files. To keep some complexity and be different though, I decided for the slightly less complex form of rendering pdf files in-app. With hindsight, the rendering process was also somewhat unnecessary. Simply using a PDF reader already on the phone saves development time, offers the user a very good user experience as most readers on the market are well-made and the interactivity is far better with pinch to zoom and multiple page rendering in full screen.

As project management goes, this project was completed well on time and to spec with minimal issues occurring during development and minimal kinks that could be worked out provided I spent more time on the app.

9. Conclusion –

9.1. Problems that were solved –

There were a number of small problems that arose during the process of designing and implementing this application. I suspect, overall far fewer than would have arisen had I not planned appropriately and put as much effort in the design. Nonetheless, the design wasn't perfect in areas such as the downloading capabilities and PDF rendering. As mentioned

previously in the evaluation, my first real problem was that I wasn't able to download PDF files as the designs were insufficient. Once I found a solution however, I could overcome that but then had the issue of separation of files between saved and temporary files. Alongside this, an issue with inadvertently attempting to render the file before it was even downloaded when choosing the temporary route. The former issue was fixed by duplicating the code into an if statement, passing the parameter of save or temp through to the DownloadTask and by altering the file-paths for each condition. This solved the issue and allowed for me to wipe the temp file directory upon the closing of the search dialog.

The latter issue arose because the pdf file download and render happened simultaneously. This meant that the filepath that the render class is looking for in order to render does not yet exist as the DownloadTask class is working to download and create it. Once I had realised this, it took me a moment to realise I just had to implement some form of wait on the render class being called for temporary pdf files. Solutions online all suggested that a thread sleep is certainly not a good way of doing that, but rather a UI handler in the dialog which had a delay on it would be far more appropriate. As such, I set a delay on the PDF render class being called to allow time for the DownloadTask to finish. I suspect that with a poor or intermittent internet connection, the 1 second delay on there might not be enough in some exceptional circumstances. In this case, the exception is handled and the PDF render class will simply not render the file.

Another issue which arose earlier on in the development process was the tables and database class. Originally, I was going to use the Resource ID's that came with the web-scrape as primary keys for the 'search_records' table. This kept causing issues relating to the ID field, so I added a column 'ID' with autoincrement to be the primary keys of both tables which immediately fixed my issues with all the methods I had created for the Open Database class. From here, I also learned the importance of updating the assets folder version of the database and wiping the data of the virtual device to essentially start over.

Smaller issues to mention include pulling the wrong columns from a table for the dialogs so that I was displaying incorrect details. This was fixed by correcting which fields were selected and correcting which items were assigned to variables from the array of strings made. Also, upon web-scraping initially, I was just taking in the fields without any alterations. This meant that with module names that included apostrophes or commas, it could have potential issues with my table. To fix this, I simply used .replace(" ' ", ""); on the strings as they were scraped to reformat each. Any characters that caused issues I simply removed with this.

9.2. Problems that could not be solved due to limitations –

The only problem I seemed to have found that is a consistent issue is in the saved activity. As mentioned previously, once a record is deleted from the dialog, the list view is not updated immediately. This can give the illusion that the record has not been deleted. However, this is untrue as both the table entry and pdf file is deleted. This produces a 'quirky' side effect of the user still being able to click on the record again. This updates the list but still opens the dialog before doing so. From here the render can still open, but shows no PDF and the user can still press delete despite it doing nothing.

The solutions I tried involving '.NotifyDatasetChanged' on the adapter and reconnecting the adapter to the list after a fresh pull of the records on dialog close worked. However this did not update the listview display. Another solution suggested implementing a UI handler for the listview to listen for dialog 'close' button event of when the activity window is 'on focus' then implement a refresh. Due to limitations with development time that I had left, I decided that the issue wasn't serious enough to require further work on them.

To conclude, the app meets most of its target scope and I consider it to be of great use to its intended audience. The idea could be expanded upon to provide a 3rd party solution to a mobile app for celcat timetable services for more educational establishments and organisations. The app could also take on more advanced features such as: 'Timetable satnav' – a timetabling app that can direct you to where you need to be, 'QR and NFC' – Rooms, student cards and induction booklets could allow students and staff to scan using the app to access timetables and relevant information.

References

[1] Celcat, "Timetable Finder - 2019/2020 Timetable," 2019. [Online]. Available: http://timetables.glyndwr.ac.uk/GenericTimetables/finder.html. [Accessed 05 04 2020].

[2] M. Rouse, "What is app monetization? - Definition from WhatIs.com," 08 2016. [Online]. Available: https://whatis.techtarget.com/definition/app-monetization. [Accessed 02 04 2020].

[3] I. Blair, "10 Ways to Effectively Monetize Your Mobile App - BuildFire," 2016. [Online]. Available: https://buildfire.com/ways-monetize-mobile-app/. [Accessed 02 04 2020].

[4] N. Galland, "Why marketers should consider in-app advertising [stats] – Econsultancy," 26 07 2018. [Online]. Available: https://econsultancy.com/why-marketers-should-consider-in-app-advertising-stats/. [Accessed 02 04 2020].

[5] developers.google.com, "Interstitial Ads | Mobile Ads SDK for Android | Google Developers," Google, [Online]. Available: https://developers.google.com/ad-manager/mobile-ads-sdk/android/interstitial. [Accessed 02 04 2020].

[6] "Mobile Ad Formats Lab #4 — Interstitial Ads | Hacker Noon," 5 10 2018. [Online]. Available: https://hackernoon.com/mobile-ad-formats-lab-4-interstitial-ads-b683142ae4dc. [Accessed 02 04 2020].

[7] Crowd Compass, "Guide-to-Sponsorship-eBook.pdf," [Online]. Available: https://www.crowdcompass.com/lp/_assets/pdf/Guide-to-Sponsorship-eBook.pdf. [Accessed 02 04 2020].

[8] U. Khan, "How to Reskin Your App in 6 Steps | Clutch.co," 31 01 2018. [Online]. Available: https://clutch.co/app-developers/resources/how-to-reskin-your-app-six-steps. [Accessed 02 04 2020].

[9] Apple, "In-App Purchase - Apple Developer," [Online]. Available: https://developer.apple.com/in-app-purchase/. [Accessed 02 04 2020].

[10] Google, "Google Play | Android Developers," [Online]. Available: https://developer.android.com/distribute/best-practices/earn/in-app-purchases. [Accessed 02 04 2020].

[11] M. Seyam and S. McCrickard, "Collaborating on mobile app design through pair programming: A practice-oriented approach overview and expert review," in *International Conference on Collaboration Technologies and Systems (CTS)*, Atlanta, GA, 2015 .

[12] J. a. N. D. Nielsen, "The definition of user experience. Articles. Nielsen Norman Group," [Online]. Available: http://www.nngroup.com/articles/definition-user-experience. [Accessed 06 04 2020].

[13] C. Mullins, "Responsive, mobile app, mobile first | Proceedings of the 33rd Annual International Conference on the Design of Communication," 07 2015. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2775441.2775478. [Accessed 06 04 2020].

[14] M. Gaigg, "UX / UI Best Practices for Designing Map Apps - tw_2416-206.pdf," 2017. [Online]. Available: https://proceedings.esri.com/library/userconf/proc17/tech-workshops/tw_2416-206.pdf. [Accessed 08 04 2020].

[15] E. Beller, "Mobile App UI Design: How To Do It Like A Pro," 16 06 2019. [Online]. Available: https://careerfoundry.com/en/blog/ui-design/how-to-design-a-mobile-app-using-user-interface-design-principles/. [Accessed 08 04 2020].

[16] M. C. J. S. E. Y. R. M. a. R. K. Thomas F. Liu, "Learning Design Semantics for Mobile Apps. In Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)," in *Association for Computing Machinery*, New York, NY, USA, 2018.

[17] N. Babich, "A Comprehensive Guide To Mobile App Design — Smashing Magazine," 12 02 2018. [Online]. Available: https://www.smashingmagazine.com/2018/02/comprehensive-guide-to-mobile-app-design/. [Accessed 08 04 2020].

[18] S. De, "5 Mobile App Design Guidelines For Better UX | CustomerThink," 24 06 2019. [Online]. Available: http://customerthink.com/5-mobile-app-design-guidelines-for-better-ux/. [Accessed 08 04 2020].

[19] M. I. a. M. L. Adzani, "UI/UX analysis & design for mobile e-commerce application prototype on Gramedia.com," in *4th International Conference on New Media Studies (CONMEDIA)*, Yogyakarta, 2017.

[20] Interactive Design Foundation, "Native vs Hybrid vs Responsive: What app flavour is best for you? | Interaction Design Foundation," 09 2019. [Online]. Available: https://www.interaction-design.org/literature/article/native-vs-hybrid-vs-responsive-what-app-flavour-is-best-for-you. [Accessed 08 04 2020].

[21] Stackify, "What Is SDLC? Understand the Software Development Life Cycle," 2020. [Online]. Available: https://stackify.com/what-is-sdlc/. [Accessed 10 04 2020].

[22] TutorialsPoint, "SDLC - Overview - Tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm. [Accessed 10 04 2020].

[23] H. M. Paul Beynon-Davies, "Rapid application development (RAD): An empirical review," *European Journal of Information Systems ,* September 1999.

[24] Lucidchart, "4 Phases of Rapid Application Development Methodology | Lucidchart Blog," [Online]. Available: https://www.lucidchart.com/blog/rapid-application-development-methodology. [Accessed 11 04 2020].

[25] A. Powell-Morse, "Rapid Application Development (RAD): What Is It And How Do You Use It?," 23 11 2016. [Online]. Available: https://airbrake.io/blog/sdlc/rapid-application-development. [Accessed 11 04 2020].

[26] jsoup, "jsoup Java HTML Parser, with the best of HTML5 DOM methods and CSS selectors.," [Online]. Available: https://jsoup.org/. [Accessed 12 04 2020].

[27] James, "How to Download a PDF File and open it in Android using an installed PDF Reader? – MOBILE PROGRAMMING," 06 03 2013. [Online]. Available: http://www.coderzheaven.com/2013/03/06/download-pdf-file-open-android-installed-pdf-reader/. [Accessed 12 04 2020].

[28] developer.android.com, "PdfRenderer | Android Developers," [Online]. Available: https://developer.android.com/reference/android/graphics/pdf/PdfRenderer. [Accessed 12 04 2020].

[29] Software Testing Help , "Types of Software Testing: Different Testing Types with Details," 16 04 2020. [Online]. Available: https://www.softwaretestinghelp.com/types-of-software-testing/. [Accessed 22 04 2020].

[30] Apptuse Team, "6 Elements to Evaluate a Mobile App | Apptuse," 04 02 2015. [Online]. Available: https://apptuse.com/blog/6-elements-evaluate-mobile-app/. [Accessed 25 04 2020].

Figures