Android app to assist J-1 Students while travelling abroad

KARL DALY

B.A. (Mod.) Business and Computing

Final Year Project April 2014

Supervisor: Mary Sharp

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

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

_________________________  ___________________________
Name                        Date
Acknowledgements

I would like to thank my supervisor, Mary Sharp for her supervision, expert and valuable advice and keeping me on track throughout this project. I would also like to thank Anthony Lynch for his valuable insights and support.

Finally, I would like to express my gratitude to friends and most importantly my Parents for their unceasing encouragement and support.
ABSTRACT

The objective of this project was to create an Android application containing features which will be of benefit to students before, during and post carrying out a J-1 visa in the United States. The features chosen would result in a more organised and enjoyable experience, while minimising some of the avoidable disasters to befall first and regular J-1 visa travellers abroad.

The app has four main features; the first being a budget calculator which takes into account information given by the user such as money in the bank, rent etc. calculates the daily and weekly budget for the user.

The second is an information section which has all the information and contact details a student will require while abroad i.e. Forum links, Sponsors numbers and addresses as well as embassies and consults contact information.

The third section is a Location marker which allows the user to add a map marker via a “long click” on the map screen recording where the user has been. The user will then have a record of all the locations they have travelled to while abroad.

The final section of the app is the emergency money section, which when activated sends SMS and emails containing required banking information to selected saved contacts.
# Table of Contents

ACKNOWLEDGEMENTS........................................................................................................... 3

ABSTRACT.................................................................................................................................. 4

1. INTRODUCTION..................................................................................................................... 11
   1.1 Motivation.......................................................................................................................... 11
   1.2 Background....................................................................................................................... 13
   1.3 Objectives........................................................................................................................ 14

2. DESIGN.................................................................................................................................... 17
   2.1 Technology to be used......................................................................................................... 17
      2.1.1 Android platform over iOS....................................................................................... 17
      2.1.2 Android SDK............................................................................................................ 19
      2.1.3 Eclipse IDE.............................................................................................................. 19
      2.1.4 ADT Plugin............................................................................................................... 20
      2.1.5 Google Maps API.................................................................................................... 20
      2.1.6 Google Play Services............................................................................................... 21
      2.1.7 Parse vs SQLite......................................................................................................... 21
   2.2 User Interface.................................................................................................................... 23
      2.2.1 Low Fidelity Prototypes............................................................................................. 23
         2.2.1.1 Login/Register.................................................................................................... 23
         2.2.1.2 Main Menu........................................................................................................ 24
         2.2.1.3 Budget Calculator............................................................................................... 25
         2.2.1.4 Information Centre............................................................................................. 26
2.2.1.5 Location Marker (Google Maps).............................................................................27
2.2.1.6 Emergency Money.................................................................................................28

2.2.2 High Fidelity prototypes and design changes............................................................29
  2.2.2.1 App Background Theme.........................................................................................29
  2.2.2.2 Login/Register.......................................................................................................29
  2.2.2.3 Information Centre..............................................................................................30
  2.2.2.4 Location Marker..................................................................................................32
  2.2.2.5 Emergency Money..............................................................................................33
    2.2.2.5.1 Send information.........................................................................................33
    2.2.2.5.2 Settings.........................................................................................................34
  2.2.2.6 Design of app icon and splash screen.................................................................38

2.3 Storage of information and Database usage...............................................................39
  2.3.1 Database..................................................................................................................39
  2.3.2 Shared Preferences..................................................................................................40

3. IMPLEMENTATION.........................................................................................................42

3.1 Languages....................................................................................................................42
  3.1.1 XML.......................................................................................................................42
  3.1.2 Java.........................................................................................................................43
  3.1.3 SQL........................................................................................................................43

3.2 Technologies Deployed...............................................................................................44
  3.2.1 Google API V2.......................................................................................................44
  3.2.2 Google Play Services...............................................................................................44
3.3 Activity implementation........................................................................................................46
  3.3.1 Budget Calculator........................................................................................................46
  3.3.2 Information Centre.........................................................................................................47
  3.3.3 Emergency Money.........................................................................................................47
    3.3.3.1 Sending SMS........................................................................................................49
    3.3.3.2 Sending Email.........................................................................................................52
  3.3.4 Location Marker............................................................................................................54
  3.4 Database Architecture.......................................................................................................56
  3.5 Login/Register re-design..................................................................................................59
  3.6 Animation Implementation...............................................................................................60

4. EVALUATION........................................................................................................................62
  4.1 Testing................................................................................................................................62
    4.1.1 Testing Model.............................................................................................................62
  4.2. Successes........................................................................................................................64
    4.2.1 Ground-up application design.....................................................................................64
      4.2.1.1 Android Application.............................................................................................64
  4.3 Difficulties.........................................................................................................................65
    4.3.1 Approach....................................................................................................................65
    4.3.2 Budget calculator result to 2 decimal spots.................................................................65
    4.3.3 Google maps V2 API Key retrieval and importing Google play services.................65
    4.3.4 Null value being sent via SMS and email.................................................................66
  4.4 Future Work.......................................................................................................................67
4.4.1 Location marker improved functionality

4.5 Business Model

4.5.1 Key partners

4.5.2 Key activities

4.5.3 Key Resources

4.5.4 Costs

4.5.5 Value Proposition

4.5.6 Channels

4.5.7 Customer relationship

4.5.8 Customers

4.5.9 Revenue

5. CONCLUSION

6. REFERENCES
LIST OF FIGURES

Fig. 1 Low fidelity login screen prototype ................................................................. 24
Fig. 2 Low fidelity Main Menu prototype ................................................................. 24
Fig. 3 Low fidelity Budget Calculator prototype ....................................................... 25
Fig. 4 Low fidelity Information Centre prototype ..................................................... 26
Fig. 5 Low fidelity Location Marker prototype ......................................................... 27
Fig. 6 Emergency Money screen 1 prototype ........................................................... 28
Fig. 7 Emergency Money screen 2 prototype ........................................................... 29
Fig. 8 High fidelity Login screen prototype .............................................................. 30
Fig. 9 High fidelity Register screen prototype .......................................................... 30
Fig. 10 Information Centre Activity ........................................................................... 31
Fig. 11 Location Marker Activity ................................................................................ 32
Fig. 12 Emergency Money screen 1 ........................................................................... 34
Fig. 13 Emergency Money deactivate PIN entry ...................................................... 34
Fig. 14 Emergency Money screen 2 - Settings ......................................................... 35
Fig. 15 Emergency Money add contact 1 ................................................................. 36
Fig. 16 Emergency Money add contact 2 ................................................................. 36
Fig. 17 Emergency Money add contact 3 ................................................................. 36
Fig. 18 Application Icon ............................................................................................ 38
Fig. 19 Application Splash screen .............................................................................. 38
Fig. 20 Sample Database User table .......................................................................... 39
Fig. 21 Sample Database mobile contact table .......................................................... 40
Fig.22 Sample Database email contact table..............................................................................40
Fig.23 Banking information Pin Entry..........................................................................................41
Fig.24 Banking Information Screen..............................................................................................41
Fig.25 Verify Google play services code snippet........................................................................45
Fig.26 Error message – Google play services unavailable............................................................45
Fig.27 Custom deactivate button..................................................................................................48
Fig.28 Login re-design ..................................................................................................................57
Fig.29 Start-up Animation example 1...........................................................................................60
Fig.30 Start-up Animation example 2...........................................................................................60
Fig.31 Business model canvas.......................................................................................................68
Fig.32 Resource Cost breakdown....................................................................................................69
Fig.33 Google Play Store ..............................................................................................................70
1. INTRODUCTION

1.1 Motivation

The topic of a summer J-1 visa with Irish students is one which is consistently under discussion in colleges around the country. Over 40,000 people travel from all over the world on this type of visa to the US every year. It’s a great opportunity and a wonderful experience for many students where both participants and potential future employers alike discover the J-1 Visa’s enriching cross-cultural opportunity to gain mutual understanding of diverse backgrounds and customs.

Recent legal discussion had the traditional J-1 visa under threat which would have rendered many of the applications functionality obsolete. The new immigration reform bill in the US would have seen would-be-employers of Irish students intending to spend the summer in the US have to pay out a $500 fee as a sponsor of that student. A provision in the bill sought to ensure that the fee cannot be paid by the student themselves.

Luckily this bill was thrown out earlier this year and according to initial estimates from Irish sponsors; is set to surpass previous years with the number of students taking part in the J-1 Visa program topping last year’s number of over 7000. [1]

The market for a J-1 phone application clearly exists with no such application available today, but this being said; wasn't the only motivation behind designing this application and was based on personal and reported experiences of J-1 students last summer.

Prior to undertaking a J-1 student program to the U.S. considerable planning and organisation is required for J-1 travellers. Those more computer literate than most are able to locate information about a chosen location, activities to do while there, the best company/sponsor to go with and the type of jobs to expect etc. For others this task can be somewhat more complex. This information while accessible is scattered across numerous websites. This initially gave credence to the notion of combining all this information into a simple listed page where students could browse more efficiently the myriad of information sources with a more effective outcome as a result.
The motivation for the app becomes more evident upon arrival in America, where the everyday challenges of living and socialising take on a different life of their own and with no current app already on the market, there is really nothing that assists students other than the generosity of sponsors and friends, (whom you can’t be calling upon every day for simple problems.)

Tasks such as trying to manage an already tight and limited budget is a challenge when items such as monthly rent, daily food expenses and unexpected costs come into play (something which many are not prepared for, if for example still enjoying the comforts of home living). This along with the many horror stories relayed via student’s travel experiences underpinned the decision to design sections of an application that would help make the lives of students on a J-1 a little easier resulting in a more enjoyable experience. Some features could reassure worried parents in areas such as emergency funds and communication and contact with friends while abroad.

The J-1 student Visa program is a rite of passage for many Irish students. For this project, most of the activities reflected in the App are based on experiences gained during the J-1 summer period of 2013 and information gathered from other J-1 students. This provided a large amount of potential material to work on and support the design and development of an app required for a more relaxed enjoyable experience for future J-1 students.
1.2 Background of the J-1

“A J-1 visa is a non-immigrant visa issued by the United States to exchange visitors participating in programs that promote cultural exchange, especially to obtain medical or business training within the U.S. All applicants must meet eligibility criteria and be sponsored either by a private sector or government program”. [2] The three institutions which aid in sponsoring Irish students are Go4less, USIT and Sayit. The J-1 participants remain in the US for the duration of their exchange program normally around 3 months to coincide with the summer holidays of students.

J-1 visa sponsors are required to monitor the progress and welfare of their participants. The J-1 visa sponsors are there to ensure that the participants' activities are consistent with the program category identified on the participants' Form DS-2019. Sponsors are also required to have their participants provide current contact (address and telephone number) information and to maintain this information in their files. J-1 participants are required to check in every month to ensure their program is going well and details up-to-date.

Electronic records of J-1 visitors and their dependents are maintained in Student and Exchange Visitor Information System (SEVIS) of the Student and Exchange Visitor Program by their program sponsor. “J-1 visitors must report certain information, such as a change in legal name or a change of address, within 10 days. Failure is considered a violation of the J-1 visitor’s immigration status and may result in the termination of the visitor’s exchange program”. [2] This issue has resulted in many horror stories/experiences for a number of J-1 participants in the past, especially when leaving the US for trips to Mexico and other bordering countries; one of which was relayed via a close friend and was the motivation behind one of the key sections included in the information section of the application.
1.3 Objectives

The objective of this project was to create an application that will be of benefit to J-1 participants before, during and after taking part in their J-1 program. The app is intended primarily for use on android devices due to not owning an Apple Mac, and thus coding in objective C to produce an iOS compatible app is quite difficult, if not impossible. It should be noted however, that through research of other languages and developer tools, the app could be developed to accommodate the possibility of being run on other smart touch screen operating systems i.e. iOS and windows phones, if the time and hardware were available to do so.

Due to the app being designed for students who must clearly understand and be able to use the functionality (intended reason for my animation on initial start-up of the app which I will discuss in detail later). Thus, the app has to be user friendly and easy to use. All functions will be accessible through a clearly labelled main menu screen for easy navigation as to minimize user error. The applications main menu will present the user with five options corresponding to the applications activities and storage of information:

1. **Emergency Money**

   When this activity is pressed via the menu, it will open a screen with a large red button located on it, below this will be a settings button. This settings button when pressed will bring the user to an additional page where SMS and email contacts can be stored in a database, pin and user information can also be checked / altered here. On the emergency money page, if the user clicks the red button, banking information saved in the pin protected banking information section (discussed in section 3) will be sent to all saved SMS and email contacts.

2. **Budget Calculator**

   This section will have a simple UI and will clearly identify what and where information needs to entered by the user i.e. money saved, rent, food expenses, wage etc. Once the user adds this information, they will have the choice of two buttons at the bottom of the screen, one labelled “daily” the other “weekly”; these buttons when pressed will calculate and present the daily/weekly budget of the user.
3. **Information Centre** –

This section of the app may not present as the most exciting or have huge functionality, application but the list of information on this page will ensure all aspects of a student’s J-1 is covered ranging from sponsors numbers, address and websites to forum sites as well as embassy/consulate contact details and addresses. This section will allow students to have all information they need at their fingertips.

4. **Banking information (locked by pin)** –

This section of the app is linked with the emergency money section. This section will be where the user will store his/her banking information i.e. bank name and address, bank account number and sort code. To ensure no one other than the user has access to this information, when this section is clicked from the main menu, the user will be asked to enter a four digit pin code, which they set up when first the app is initially started and stored in the database.

5. **Location marker** -

This idea behind this section is applicable to the post J-1 period upon returning home, when trying to re-call locations visited while abroad. Currently on Apple IPhones, pictures are segmented by date, but also by the locations in which they were taken, allowing the user identify where they have travelled.

Unfortunately, this is not the case for android so in this section of the app, it initially opens a map which hones in on the users current position. From here the user will be able to set a marker on any location by a “long click” on the screen (holding their finger in a position for a moment). The user can store as many markers as they wish and on returning home can view where exactly they have travelled. If a user wishes to remove markers they can do so by pressing the clear marker button located at the bottom of the screen.
• **Database** –
  The application will make use of a database to store not only user information such as name, address and pin number (4 digits) which will be used in multiple sections in the application e.g. to access banking information, but also the contact details for the SMS and email recipients. This database will allow the application to query data and use it in general functionality of the app.

• **User Friendly** –
  It’s important for the application to be easily navigated and its functionality to be clearly understood by the user.
2. DESIGN

2.1 Technology to be used

In this section, an overview of the technologies that were planned to be used in the initial designs of the project will be discussed.

2.1.1 Android platform over IOS

Before carrying out any additional research into operating systems and their application development, a decision was made to develop the project in either android or iOS due mainly to their popularity and available information. The android operating system released in 2007 is based on the Linux kernel and is designed primarily for touch screen smart phones and tablets. Android provides a Java-based development platform for applications on the android operating system. Android applications are split into four components:

1. Activities:
   – An activity represents each screen (user interface) contained in an application.

2. Services:
   – Services do not provide a user interface, instead providing a component that runs in the background to perform long-running operations or work for remote processes [19].

3. Content Providers:
   – Content providers manage a shared set of application data through which other applications can query or modify application data.

4. Broadcast Receivers:
   A Broadcast receiver is the component that responds to system-wide broad-cast announcements such as low battery or when a picture has been taken [19].

Every Android application must contain a manifest file in its root directory. This file presents information that the system must have before it can run any of the application’s code [20]. As well as code, Android applications require resources that relate to the visual presentation.
of the application. The resources, such as the layout of ‘activity’ user interfaces and menus are declared in XML files [42]; this allows for modification to the applications appearance without modifying any code. Android is the most used platform among developers (71% use it) which underpins the decision to include along with the vast amount of information and tutorials on how to program with it.

From a business perspective, it was also a good choice. As of May 2012, Android became the most popular mobile OS, having the largest installed base, and is a market leader in most countries with a 53.6% market share in 2013 [18][3]

iOS also unveiled in 2007, is a mobile operating system developed by Apple Inc. for apple devices such as the iPhone, iPod etc. but unlike android, Apple does not license iOS for instillation on non-apple hardware. It has 21%+ share of the smartphone mobile operating system. [4]

Android apps are written in the Java programming language along with Extensible Mark-up Language (XML) used in the layout of the user interface. Time in college provided the opportunity to study both of these languages which gave android an initial advantage over iOS.

On the other hand, applications that run natively on the iPhone operating system are written in Apple's Objective-C, a dialect of the more common C language something which was unfamiliar, yet due to owning many apple products had great interest in learning.

As mentioned earlier, android is an open development platform, it allows developers leverage the device hardware, creating location-aware apps by accessing GPS and other sensory information on the device, set alarms to remind users of events, include notifications and other information on the status bar of the device, and more; some of which were planned to be used in the application. In contrast, iPhones have difficulty displaying multiple notifications, since applications are restricted to pop-up messages that are shown only one at a time.
Additionally, developers on Android can leverage various carrier features across the spectrum of Android devices, whereas iPhone devices are limited to the network features that their carrier allows. Android applications with 2.2+ SDK (software development kit) can be developed to be used on either touch screen or the device keyboard something which iOS cannot, meaning applications developed in Android can accommodate a larger set of devices.

The deciding factor on whether to use Android or iOS for the application was simply due to not having the equipment to code an iOS application. As briefly discussed earlier, in order to code in objective C to produce an iOS application, the developer must own a Mac. There are ways to code objective C on Windows, but it is incredibly difficult and requires the download and installation of a number of separate programs. [5]

2.1.2 Android SDK

The Android software development kit (SDK) is a kit which includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU (an emulator is hardware or software or both that duplicates the functions of one computer system (application) in another computer system (laptop)), documentation, sample code, and tutorials.

The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) Plug-in. Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing. [6]

2.1.3 Eclipse IDE

Eclipse is a free, open source integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse
can be used to develop applications by means of various plug-ins; Eclipse may also be used to develop applications in other programming languages. Eclipse was chosen for the application IDE because of previous experience in its use as well as a good understanding of its layout and tools. The Eclipse software development kit (SDK), which includes the Java development tools, are meant for Java developers but its abilities can be extended by installing plug-ins written for the eclipse platform, such as the ADT plug-in for android development.

2.1.4 ADT Plugin

Android development tools (ADT) is a plug-in for the eclipse IDE that’s designed to give developers a powerful, integrated environment where android applications can be built. It extends the capabilities of the Eclipse SDK allowing quick set up of new android projects, creation of application UI, and easy addition of android API packages; it debugs applications and can even export .apk files in order to distribute applications.

Developing in Eclipse with the ADT plug-in is highly recommended and is the fastest way to get started. With the guided project setup it provides, as well as tools integration, custom XML editors, and debug output pane, ADT gives you an incredible boost in developing Android applications. [7]

2.1.5 Google Maps API

The Google Maps API is a free service released in June 2005, which allows developers to integrate Google Maps into their applications. By using the Google Maps API, it makes it easier for developers to add powerful mapping capabilities to an application, the Google APIs add-on includes Maps external library, com.google.android.maps. The key class in the Maps library is MapView, a subclass of ViewGroup in the Android standard library. A MapView displays a map with data obtained from the Google Maps service. When the MapView has focus, it can capture key presses, touch gestures to pan and zoom the map automatically; including handling network requests for additional maps tiles. It also provides
all of the UI elements necessary for users to control the map. In general, the MapView class provides a wrapper around the Google Maps API that lets your application manipulate Google Maps data through class methods, and it lets you work with Maps data as you would other types of Views.

The Google APIs add-on provides the Maps library to you so that you can develop, build, and run maps-based applications in the Android SDK, with full access to Google Maps data. Over 1,000,000 web sites and far more applications use the Google Maps API, making it the most heavily used web application development API. [8]

2.1.6 Google Play Services

Google Play Services is a proprietary software development kit (SDK) and application programming interface (API) set for Android devices. First introduced in 2012, the layer provides APIs that allow apps for Android to provide functionality that directly integrates with Google services, such as account syncing, Google Maps, Location APIs, Android Device Manager, and others. Google Play Services is a dependency of all Google applications for Android.

2.1.7 Database

On initial design of the application, the choice as to which type of database to use was still under consideration. After researching in depth and coursing through numerous developer forums, two options were presented. The first was to implement parse (https://www.parse.com/) as the backend for the application and the other was to write a SQLite database from scratch.

Parse.com provides a backend/database service to developers. It provides mobile app developers with a way to link their applications to backend cloud storage, while also providing features such as user management, push notifications, and integration with social networking services. Parse provides these through their SDKs that let apps running on various devices connect up to the backend it provides. They provide SDKs for devices
running iOS, Android, Windows (Phone) 8, OS X and JavaScript. Parse offers three different payment plans at the moment. The first is free and only one to be concerned with this project, it allows an app to make 1 million requests per month with a burst limit of 20 requests per second and also provides 1 million pushes per month. It supplies everything a developer needs to build a full-featured Android app with a reliable backend. The Parse Android SDK allows developers to store data, manage users, and send push notifications, track analytics, and more in just a few lines of code. [9]

SQLite on the other hand, is a relational database management system contained in a C programming library. In contrast to other database management systems, SQLite is not a separate process that is accessed from the client application, but an integral part of it. It is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is completely free for use for any purpose, commercial or private. It is arguably the most widely deployed database engine, and is used today by several browsers, operating systems, and embedded systems, among others.

SQLite is ACID-compliant (Atomicity, Consistency, Isolation, Durability which are a set of properties that guarantee that database transactions are processed reliably) and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax. SQLite is a popular choice as an embedded database for local/client storage in application software such as applications. Unlike client–server database management systems, the SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program.

SQLite is a compact library. It can be run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as mobile phones, PDAs, and MP3 players. There is a trade-off between memory usage and speed so, SQLite generally runs faster the more memory it is given. Nevertheless, performance is usually quite good even in low-memory environments. [10]
2.2 User Interface

The design of the user interface for the application took some thought as the application needed a UI that was clear and easy to follow, lowering the chances of user error and confusion. Initially in this section, low fidelity prototypes of the application designed in the beginning will be discussed, after which high fidelity prototypes and any changes made to initial UI designs will be examined.

2.2.1 Low fidelity prototypes

Low fidelity or paper prototypes for the application involved creating rough, hand-sketched drawings of the interface. The expected result was to produce a rough picture of the structure of the application and thus giving a better structure to initially start coding the UI in XML. Prototypes for each activity were created along with the initial designed application start-up page:

2.2.1.1 Login/Register page

This design was set to be used as the initial start-up page for the application, the UI is like any other login/register page that the user would have seen before to reduce confusion, with two Edit Text boxes clearly labelled “username” and “password” along with two large “separated buttons” underneath (to reduce the chance of user pressing a button accidentally) “Login” and “Register”.
2.2.1.2 Main Menu

With any application that contains numerous activities, a clear main menu page is critical to provide easy navigation for the user, this application was no different and the main menu page was designed to be as clear and self-explanatory as possible. Each clickable option for the user is clearly labelled and with the use of large text size and buttons, user miss-clicks should be minimized.
2.2.1.3 Budget calculator

For this application activity as with all the activities, ideally it will be clearly laid out in order to let the user know exactly what needs to be filled in and where. The activity takes into account 6 pieces of information from the user, each located on separate lines with text informing the user exactly what they should enter i.e. earning p/w (per week); along with this inside the Edit Text boxes (where user inputs the information) symbols/hints are displayed e.g. $ / no. of weeks, again insuring the correct type (dollars instead of euro) and quantity (per week or per day) of information is added by the user. The budget calculator try’s to take into account as much information as possible to insure the most accurate resultant figure. The buttons “Daily” and “Weekly” located underneath when pressed will calculate and show the budget, daily or weekly in large clear text at the bottom of the screen to the nearest cent.

Fig.3 Low fidelity Budget Calculator prototype
2.2.1.4 Information Centre

The information centre activity, though being clear as to what the activity does itself, its layout and ease of information extraction is important. The page is laid out into sections and subsections starting with each sponsor users could have used, under which hyper-links and contact information are located.

Forum site links and importantly and for reasons listed earlier a list of Irish embassy and consulates with related contact information and locations. The use of different text sizes and styles will be well implemented in this activity in order to display information clearly to the user.

Fig. 4 Low fidelity Information Centre prototype
2.2.1.5 Location Marker (Using Google maps API)

As this section of the application uses the Google maps API, the design of the UI was already pretty much complete. The goal of this map activity was to design it, in order to be as similar to the actual Google maps application as it could be in order to make it look and have similar functionality insuring user’s competency while using this section.

With the use of additional Java, the UI will include the common Google map feature such as zoom, current location button etc. The only additional object added to the activities UI is a button located at the bottom of the screen, which is clearly labelled “Set Marker”, when pressed this places a marker (already incorporated in Google maps API) on the users current location.

![Fig. 5 Low fidelity Location Marker prototype](image-url)
2.2.1.6 Emergency Money

The initial UI design for this activity was completed previous to research of the android platform and its features, as well as how the application planned on storing user information. The section was separated into two pages, one for storage of user input information and the other for actually sending this information. The storage page was clearly laid out with titles located above EditText boxes, where the user would add information ranging from bank account number, sort code to the recipient’s phone and emails. The second page contained a single button in the centre of the screen labelled “Send Banking Information”, which when pressed sends the information to the recipients.

![Fig.6 Emergency Money screen 1 prototype](image1)  ![Fig.7 Emergency Money screen 2 prototype](image2)
2.2.2 High Fidelity prototypes and design changes

The low fidelity prototypes were very helpful in identifying the initial UI design; however, before starting to code the app, more realistic high fidelity designs were required for a number of the application sections, so using the software available at https://www.fluidui.com/editor/ [11] Some high fidelity prototypes were created which really helped in identifying what needed to be included, colour schemes to use as well as button position. Along with these prototypes, knowledge of android development as a whole far surpassed what it was like while designing the paper prototypes, so other design changes were made which will be discussed in this section.

2.2.2.1 App Background Theme

The theme of an app might mistakenly be perceived as a minor detail, but surprisingly if the wrong theme is chosen the app might appear cheap and badly designed. From the initial planning stages this application was designed to be user friendly requiring the theme to be easy on the eyes allowing objects such as buttons to be clearly visible and distinguishable to the user. Initially, a white themed background was chosen, but while designing the fluidUI prototypes it was observed that a white background made the app look cheap and the objects on screen not as clear as expected. Changing it to black was a good decision which gave the application a more aesthetic appearance as well as making the objects more distinguishable.

2.2.2.2 Login/Register

The original design for the login page does not significantly differ the initial paper prototypes; an additional text at the top of the page saying “Welcome” was added, and “hints” were inserted into each EditText box informing the user what to enter into each box, which in the authors opinion makes the app look more professional and aesthetic. Along with the opening page, a register page was designed which will open if the user clicks the
register button; it is a simple page with four EditText boxes clearly labelled along with a button labelled “register”.

![Fig.8 High fidelity Login screen prototype](image1)
![Fig.9 High fidelity Register screen prototype](image2)

Most applications initially start with a similar styled login screen where once logged in, the user remains logged in which is convenient, as requirement for regular logins can be quite tedious. However, due to the application storing sensitive banking information, allowing the user to be constantly logged in poses a security threat. Thus at this stage of the design it was thought necessary to require the user to login to the app on each use.

**2.2.2.3 Information Centre**

The high fidelity designs of this activity didn’t differ extensively from the original paper prototypes. As discussed earlier, the layout combined with ease of information extraction is important as is the quality of the information provided. Each main heading i.e. Sponsors, Embassy are given larger, “bold and italic” style text in order to clearly segment each section
of data. The hyper-links are highlighted in blue similar to the style used on other websites and application ensuring the user is aware of their functionality.

The addition of the clickable contact details feature will be more time efficient for the user; every contact number in this activity section is clickable. Once a number is clicked, it will open the user’s dialler keypad with the number already inserted allowing a single button click to call a related number. The quantity of information located on the page required the addition of a scroll bar, allowing the user scroll freely through the activity accessing all information.

Each sponsor website listed in the application has its own forum, which contains sufficient information. Also included is www.J1forum.com. [12] Which presents an array of information relating to all aspects of a J-1 from available jobs to what location students should visit; it will be very beneficial to any user of the application.

![Fig.10 Information Centre Activity](image-url)
2.2.2.4 Location Marker

The location marker activity required some minor modifications to the original design. In the initial paper prototype, a button located at the bottom of the map labelled “set marker” would allow the user, as the name implies, to set a marker on their current location. Unfortunately this presented as a design flaw. If for example, a user forgets to apply a location marker when they are at a specific location e.g. Las Vegas, they will be unable to add an additional marker at a later date due to the marker being set on their “current location”. To resolve this –as opposed to using a button- the application allows the user to “long click” the map at any location, this will place a marker on the pressed point, rather than the current location (which will still be visible via the normal google maps blue dot).

Another design flaw was the inability to delete a marker once it had been placed; the addition of a “delete marker” button to replace the “set marker” button resolved this issue. One final modification was the addition of a “My location button” (located in the top right hand corner of the map), when pressed it will move the map view to your current location rather then automatically doing so when the activity is opened, as per the original design. This will provide the user with the ability to revise all their map markers before placing an additional one.

![Location Marker Activity](image)

*Fig.11 Location Marker Activity*
2.2.2.5 Emergency Money

The design for this activity altered substantially based on cumulative learning, knowledge and experience at various stages of and during the development of the Android application. Similar to the original prototypes, this activity is segmented into two main separate pages, 1. A page used to initiate the sending of bank information and 2. A settings page, where the user can store and change information relating to themselves and the recipients.

2.2.2.5.1 Send information

The idea behind this page didn’t differ much from the original paper prototype; the screen is largely taken up by a custom button (rather than a small, un-coloured standard button) labelled “panic button”, the button is designed to “jump out” catching the user’s attention. The label and colour of the button is also meant to make it clear to the user that the button should only be used in an emergency.

In the original design, once the button is pressed, banking information would be sent directly to all recipients stored within the database. In the event of a user accidentally pressing the button (which resulted in the unnecessary transfer of monies) some fail safe mechanism was necessary. To resolve this issue a reverse timer count down from 10 is initiated and shown on screen. The user can then re-press the custom button, which will then present a dialog on screen asking the user for their PIN (already set up and stored in the database). When the user enters a password/PIN, it will be checked against the database and if successful, the button will be deactivated and no information will be sent. A notification will be presented on screen informing the user as to whether information was/wasn’t sent.

Along with the custom button, two additional buttons have been added to the design located at the bottom of the screen, one labelled “settings” (which is the link to the storage page discussed in the next section) the other is “Menu” (which brings the user back to the main menu.
2.2.2.5.2 Settings

The design for this page has altered substantially to the original paper prototype making it more aesthetically pleasing while offering increased functionality. This screen is the link between this activity and the database. As you can see from fig.14 below, the screen is a menu style page designated into three areas 1) Emergency money recipients, 2) User information and 3) Help section. Each area is clearly segmented and each line is clearly labelled with a custom icon for each option to be pressed e.g. image of a face for contacts.
1. The top two buttons on the menu are linked with adding SMS and email recipients contact information, when pressed the user is presented with a page (fig.15 below) allowing them to store contact information whether mobile number or email address to the database, the user can then delete contacts by “long clicking” the added contact (a dialog box will be presented asking the user “are they sure you wish to delete contact!” ensuring the deletion is intentional).
2. The two options located in the “User information” section are present just in case a user wishes to modify user information which has been stored to the database,
(User initially entered information on start-up of application) PIN information to deactivate the custom button and to access bank information (discussed further in section 2.3) can also be altered here.

3. The help option of this menu will be where a tutorial of the emergency money activity and its features will be displayed, ensuring the user understands how to correctly use this activity and what information needs to be added (explained in detail in implementation section).

Overall this activity is designed to be as user friendly as possible employing the use of strong bright colours, segmentation and custom buttons which ensures the user is fully aware and informed as to their use and application.
2.2.2.6 Design of app icon and splash screen

As the application is designed for J-1 students travelling to the US, the use of the American flag is utilised in the apps start-up icon as opposed to the standard android icon. This incorporates well into the apps overall function and stands out very well on the phone screen (eye-catching) which is ideal for any application.

A splash screen is an image that appears while a game or application is loading or simply used as an introduction page. Splash screens cover the entire screen or simply a rectangle near the centre of the screen. Splash screens are present in many applications and are more appealing then looking at a blank black screen while the application loads. Again, like with the app icon the American flag is present along with the application name “J-1 utility app”. Good eye-catching, memorable icons and splash screen are a very good way to draw user’s attention to an application.

![Fig.18 Application Icon](image1)

![Fig.19 Application Splash screen](image2)
2.3 Storage of information and Database Usage

Throughout this section the different types of storage to be used within the app will be discussed. The decision on which data storage model to implement is one of considerable importance for any application. The choice of storage method can have a substantial effect on both underlying performance and the potential scalability of the application and therefore must be considered carefully.

Two types of storage are used in the application in order to ensure the app is as secure and efficient as possible and to remain user friendly. The types of storage to be used in this application are database storage (Parse or SQLite) where the tables (where information is stored) will be similar regardless of the type of database chosen. The second type of storage will be shared preferences, this is a class that allows users to save and retrieve persistent key-value pairs of primitive data, the data will persist across users sessions i.e. saves user information even if app is closed. [13]

2.3.1 Database

Within the application database, three separate tables will be created where the fields in each table relate to the structure of the specific type of information stored. The three tables to be created are 1) User table 2) Contact table and 3) Email contact table.

1. The user table as the name describes, is a table where user information will be stored. Each user will have its own unique I.D which will be the table’s primary key (The primary key of a relational table is used to uniquely identify each record in the table). User information will include first name, second name, address, security pin. The application will retrieve this information on the initial start-up of the application; users will have the ability to view and change any information they wish in the settings section of the app.

Fig. 20 Sample Database User table
2. The contact table will be used to store the names and mobile numbers of the emergency money recipients. Again like the user table, each entry will have its own unique I.D (primary key). The information for this table will be gathered from the user via the “add contact” button in the settings screen. The user will have the ability to view each entry as well as delete a contact from the database.

<table>
<thead>
<tr>
<th>I.D</th>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Heather</td>
<td>872659854</td>
</tr>
<tr>
<td>1</td>
<td>Jack</td>
<td>836589456</td>
</tr>
<tr>
<td>2</td>
<td>Matthew</td>
<td>876432663</td>
</tr>
<tr>
<td>3</td>
<td>Eoin</td>
<td>864587523</td>
</tr>
</tbody>
</table>

*Fig.21 Sample Database Mobile contact table*

3. The email contact table will be very similar to the contact table above with the only noticeable difference being the user will be asked for an email address rather than the mobile number of a recipient. Again the user will be able to view each entry as well as delete them.

<table>
<thead>
<tr>
<th>I.D</th>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Shauna</td>
<td><a href="mailto:shaunamullen@gmail.com">shaunamullen@gmail.com</a></td>
</tr>
<tr>
<td>1</td>
<td>Mark</td>
<td><a href="mailto:mark_mccann@hotmail.com">mark_mccann@hotmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Anthony</td>
<td><a href="mailto:anto.lynch@yahoo.com">anto.lynch@yahoo.com</a></td>
</tr>
<tr>
<td>3</td>
<td>Mandi</td>
<td><a href="mailto:mandi.daly@gmail.com">mandi.daly@gmail.com</a></td>
</tr>
</tbody>
</table>

*Fig.22 Sample Database email contact table*

2.3.2 Shared Preferences

The application will use the shared preference class to store users secure banking information, three pieces of information will be stored 1) Bank name and address 2) Bank account number and 3) Sort Code. The choice to use shared preferences rather than just storing it in the database was one of time efficiency and user security.
The shared preferences class allows for fast retrieval of information, users can view and erase data without having to query, delete and re-write the database. In addition, use of shared preferences is simple to understand and program. As it is used in the most complex section of the application, the “emergency money” activity possessing a faster, shorter and simpler method of information retrieval and deletion presented as the best option.

Security; some users may not like their banking information being stored externally (even though parse is encrypted). The shared preferences class allows users banking information to be store locally on the phone and only be accessible by the user themselves. In addition, to access this information the user will have to enter their 4 digit PIN stored in the database giving additional security to the information stored. The shared preferences class will be located on the main menu of the application.
3. IMPLEMENTATION

3.1 Languages

In order to complete this application successfully, three languages were implemented. These were 1.) XML, 2) Java and 3) SQL.

3.1.1 XML

XML or Extensible Mark-up Language was used in numerous areas of the application.

Layout xml:

This is used to define the logical structure of the User interface (UI) of the application. The Android SDK provides a comprehensive XML vocabulary that corresponds to the View classes and subclasses, such as those for buttons and layouts. Layout holds all the elements and the tools that are used in the application.

Manifest xml:

This xml file is used to define all the components of the application. It includes the package name which serves as a unique identifier for the application, the activities, services, broadcast receivers and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities. These declarations let the android system know what the components are and under what conditions they can be launched, It also is the area where permissions are declared e.g. “<uses-permission android:name="android.permission.SEND_SMS"/>”, the application must have these permissions in order to access protected parts of the API and interact with other applications on the phone. [14]

Strings xml:

This xml file is used quite regularly throughout the application, it is used to replace the Hard coded strings with a single string, it reduces the amount of code and allows re-use of a String e.g. for a String such as “Help I need money, can you please send some” creating a reference in the String.xml class such as “@string/Help_money” allows for this string to be
called numerous times instead of having to constantly re-write the same Sting each time you wish to use it.

Draw-able xmls:

These are the xml files that are used to provide various graphics to the elements of application. Within the application a large number of custom pictures, buttons etc. are used and this is the area in which they are stored.

3.1.2 Java:

Java is a concurrent, class-based, object-oriented language that provided functionality to the UI and was the core language used to develop the application. The Android SDK again provides the API libraries and developer tools necessary to build, test, and debug applications for Android.

3.1.3 SQL:

The SQL language was used in order to query the chosen SQLite database. The SQLite database uses weakly typed SQL syntax and statements such as “SELECT” and “FROM” are used regularly.
3.2 Technologies Deployed

Within this section the technologies which were used in the development of the application will be discussed. All technologies discussed within the design section were used, but some had become deprecated and thus could not be implemented; how this was handled, failsafes of technology already discussed and choice of database which was utilised are discussed below.

3.2.1 Google API V2

In the initial design and research of the application, the Google maps V1 API seemed like the best option, there were many online tutorials showing its functionality etc. Unfortunately when actually trying to use the API, it had become deprecated. However, many developers still have google maps V1 implemented into their application; this is due to the level of work to upgrade to V2 as it is of a different design.

The main reason of the switch to the V2 API in the application was on attempting to retrieve an API key for the V1, where late last year this became impossible and the V2 was the only option. Implementing Google maps V2 initially seemed like a daunting task due to lack of knowledge and available information, but in reality the API is far more optimised than the previous one where some methods are now easier to implement eg finding the users current location which in V1 would take four lines of code now in V2 only takes one:

```
googleMap.setMyLocationEnabled(true);
```

Google maps API V2 may be more optimised then V1 and allow simpler coding, yet it still caused a number of difficulties which will be discussed later.

3.2.2 Google Play Services

The implementation of Google play services was crucial in order for success of the Location marker activity. Google Play delivers service updates for users on Android 2.3 and higher through the Google Play Store app. However, devices must have compatible Google play services APK before the Google play service features can be accessed; some devices do not
have it installed at all. To resolve any crashes or problems within the application, failsafe methods were implemented to verify the version and/or if Google play services are available on the device. The structure of the methods is as follows:

To verify the Google Play services version, “isGooglePlayServicesAvailable()” is called. If the resulting call is “SUCCESS”, then the Google Play services APK is up-to-date and the application can continue to make a connection. If, however, the result code is SERVICE_MISSING, SERVICE_VERSION_UPDATE_REQUIRED, or SERVICE_DISABLED, then the user needs to install an update. So, GooglePlayServicesUtil.getErrorDialog() is called. This returns a dialog which is presented to the user; it provides an appropriate message about the error and provides an action that takes the user to Google Play Store to install the update. [15]

```java
int resultCode = 
GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());

if (resultCode == ConnectionResult.SUCCESS){
    Toast.makeText(getApplicationContext(),
    "isGooglePlayServicesAvailable SUCCESS",
    Toast.LENGTH_LONG).show();
} else{
    GooglePlayServicesUtil.getErrorDialog(resultCode, this,
RQS_GooglePlayServices);
}
```

**Fig. 25 Verify Google play services code snippet**

**Fig. 26 Error message – Google play services unavailable**
3.3 Activity implementation

Within this section, each activity of the application will be examined, discussing in detail how each activity works and some of the main methods which have been implemented.

3.3.1 Budget Calculator

The implementation of the budget calculator activity was easier than expected, the UI design discussed previously (design section) was implemented requiring no modification. Only the java coding had two areas which required some additional thought. When users input all the information required in the Edit text boxes, two methods were created, where each is called depending on whether the user clicks the daily or weekly buttons, these methods were “DailyCalculation()” and WeeklyCalculation()”, the methods are very much the same, the only difference being an additional division in the arithmetic in DailyCalculation(). due to Edit Texts taking in the user input as a Sting, and as you cannot do arithmetic with strings, before each method is called by the application, it initially checks whether the inputted value isn’t null and then placing each value in a double variable, the application changes the String value for double value by using Double.parseDouble().

```java
if (weeks != null)
    weeksCal = Double.parseDouble(weeks.getText().toString());
if (bank != null)
    bankCal = Double.parseDouble(bank.getText().toString());
```

Once all the users’ inputted information has been changed to type double, a simple calculation was developed to take into account all this inputted information and set in to the totalSum variable which is the returned value of the method.

```java
totalSum = (((bankCal / weeksCal) + (hoursCal * wageCal)) - (((weeksCal / 4) * (rentCal / peopleCal)) / weeksCal) + (foodCal * 7) + otherXCal) / 7;
return totalSum;
```

The only difference between the daily and weekly budget calculation is that when the daily button is pressed the whole calculation is divided by 7 i.e. 7 days in a week (seen above in code).
Once the method is called, the application then setting the totalSum variable retuned by the method in a textview, presents the calculated information.

Depending on the user information inputted, the answer of the calculation can sometimes be to numerous decimal places, so in order to limit the resulting answer to two decimal spots, the application formats the answer using:

```java
DecimalFormat formatter = new DecimalFormat("0.##");
    total.setText("Your Budget is $ " + formatter.format(totalSum));
```

3.2.2 Information Centre

The information section did not require significant java coding as all the challenging work was completed in the info.xml file where all the information is stored and formatted into the UI. Using different text sizes and colour as well as using androids android:textStyle formatter, "android:textStyle="bold|italic". The information was made as clear and as easily readable as possible, the additional functionality of being able to click a hyper link was implemented using android:autoLink="web". And the final additional functionality of when a number is clicked the application brings the user to the phone application with the number already stored was implemented using android:autoLink="phone". This activity caused the least problems while being implemented as it had limited functionality and the only issue was designing the UI in order to make it as aesthetically pleasing as possible.

3.2.3 Emergency Money

This activity was by far the most complex and presented a number of issues. In order to implement the activity successfully, it makes use of 14 classes which is necessary to complete such activities as retrieving contact information, present dialogs to the user, send SMS / email etc. and thus is the main reason this activity makes up the bulk of the projects code.

As discussed in the design section, this activity is split into two main pages, one which is used to send banking information and another which is the link to the database (where contacts are store). Within the send money page, an additional UI design change was
implemented, where another custom button was created and used when the “panic button” button is pressed. Once pressed the button now changes into the custom deactivate button, which can be pressed to initializes the deactivate procedure of PIN entry etc. discussed below.

![Deactivate Button](image)

*Fig. 27 Custom deactivate button*

When the panic button is pressed a timer is initiated which counts down from 10 with a dialog on screen showing the user how much time they have left to deactivate the sending of the information.

**Method to start count down:**

```java
public void onTick(long milliseconds) {
    if (milliseconds / 1000 >= 0 && dialog.check() != true) {
        String val = getResources().getString(R.string.seconds_remaining);
        timer.setText(val + (milliseconds / 1000));
    }
}
```

if the user presses the deactivate button during the count down, they will be presented with a dialog asking them to enter their unique 4 digit PIN, if the PIN matches the value stored in the database, the activity is deactivated and no information is sent.
Code to verify Pin:

```java
public void onClick(DialogInterface dialog, int which) {
    text = textEdit.getText().toString();
    Database db = new Database(context);
    acceptedPin = db.checkPin(text, context);
    if (acceptedPin != true) {
        Toast.makeText(context, R.string.incorrect_pin, 1000).show();
        showDialog();
    }
}
```

If however, the counter reaches zero, the application calls the main method of the activity, onFinish(). This method was difficult to code as it uses method calls from numerous other classes as well as needing sufficient exception catches to prevent application crashes. This method retrieves contact information from the database of both email and mobile contacts using db.get calls such as db.getContacts(), it then implements this information in the prentices of the sendSMS() and GmailSender() methods which I will discuss below.

OnFinish() method:

```java
public void onFinish() {
    final String[] personalInfo = db.getPersonalDetails();
    final Cursor contacts = db.getContacts();

    if (match == false) {
        sendSms();
        if (db.hasGmail()) {
            Thread s = new Thread(new Runnable() {
                public void run() {
                    String args[] = db.getGmail();
                    GmailSender sender = new GmailSender(args[0], args[1], getApplicationContext());
                    Cursor c = db.getEmailContacts();
                    while (c.moveToNext()) {
                        try {
                            Log.e(args[0], args[1]);
                            sender.sendMail(args[0], c.getString(c.getColumnIndex("emailAddress")));
                        } catch (Exception e)
                    }
                }
            });
        }
    }
}
```

3.2.3.1 Sending SMS

For the application to be able to send SMS’s successfully, two pieces of information are required to be available and retrieved, 1) Banking information and 2) Contact details. In
addition, a methodology was employed to take this retrieved information and using broadcast receivers and the phones other applications, forward the information and ensure its delivery.

1. The banking information as discussed in the design section is stored locally on the phone in the shared preferences class and is called using the preference manager. In order to be able to be use these retrieved values in the SMS message, the application sets each preference value into a String variable with `prefs.getString()`.

   **Method to retrieve banking information:**

   ```java
   public void initilizePrefs() {
       prefs = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
       BankAccount = prefs.getString("BankAccount", null);
       BankNameAddress = prefs.getString("BankNameAddress", null);
       SortCode = prefs.getString("SortCode", null);
   }
   ```

2. Once the banking information has been retrieved and stored into variable, the string message to be sent to each recipient is created, a try and catch statement is used in this section in order to prevent crashes and to ensure correct information has been sent, the method initially verifies whether the value retrieved from the shared preferences class are not null, then using a variable it creates a suitable String message and sets it as the parameter of the sendSMS() method, where it will be sent to the recipients. Exceptions have been put in place in order to stop messaging with null valued information being sent i.e. if the user try’s to activate the emergency money activity and has not entered banking information into shared preferences, a dialog will appear asking them to do so.
Code snippet creating message and SendSMS() method call:

```java
String mes = "my account info is: " + BankNameAddress + " account number: " + BankAccount + " Sort Code is: " + SortCode + " Thank you so much!!";

try {
    if (BankNameAddress != null && BankAccount != null && SortCode != null) {
        sendSMS("Help!! I've completely run out of money and need you to send some via bank transfer please. " + mes);
    } else
        Toast.makeText(getBaseContext(), "Please ensure all sections of preferences are filled", Toast.LENGTH_SHORT).show();
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

The method required to actually send an SMS was extremely difficult to code and required a lot of additional research. Sending information via the phones SMS sender isn’t difficult, but doing it in the background of an application presents a number of challenges; however, after a lot of tweaking, the method was successfully implemented and the issues were resolved resulting in the application section functioning exactly as designed.

The method takes in a Sting as a parameter which as discussed earlier contains the secure banking information, from here it queries the database retrieving the mobile contacts details stored in it, it does this by going through the array of contacts and sends an SMS to each section of the array; exceptions have been implemented which are called if the contact information cannot be retrieved. Along with this, the method calls two additional broadcast receiver classes SMSReceiver which when called verifies if the message was sent correctly and the SMSDeliveredReceiver class which verifies that the message was delivered to the recipients, both classes present dialogs which inform the user the message has been sent and received.

**sendSMS() method:**

```java
private void sendSMS(String message) {
    Database db = new Database(this);
    Cursor cursor = db.getNumbers();
    db.onStop();
    if (cursor != null) {
        while (cursor.moveToNext()) {
```
String phoneNumber = cursor.getString(cursor.getColumnIndex("number"));

try {
    SmsManager sms = SmsManager.getDefault();
    ArrayList<String> mSMSMessage = sms.divideMessage(message);
    for (int i = 0; i < mSMSMessage.size(); i++) {
        sentPendingIntents.add(i, sentPI);
        deliveredPendingIntents.add(i, deliveredPI);
    }
    sms.sendMultipartTextMessage(phoneNumber, null, mSMSMessage,
        sentPendingIntents, deliveredPendingIntents);
} catch (Exception e) {
    e.printStackTrace();
    Toast.makeText(getBaseContext(), "SMS sending failed...", Toast.LENGTH_SHORT).show();
}

3.2.3.2 Sending Email

As the application sends emails in the background, a JSSE provider class had to be implemented. A JSSE or Java Secure Socket Extension provides a set of packages which enable secure Internet communications. It implements a Java technology version of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. It includes functionality for data encryption, server authentication, message integrity, and optional client-authentication. The application uses this class to access and use the users email account.

JSSEProvider() method:

public JSSEProvider() {
    super("HarmonyJSSE", 1.0, "Harmony JSSE Provider");
    AccessController.doPrivileged(new java.security.PrивilegedAction<Void>() {
        public Void run() {
            put("SSLContext.TLS",
                "org.apache.harmony.xnet.provider.jsse.SSLContextImpl");
            put("Alg.Alias.SSLContext.TLSv1", "TLS");
            put("KeyManagerFactory.X509",
                "org.apache.harmony.xnet.provider.jsse.KeyManagerFactoryImpl");
            put("TrustManagerFactory.X509",
                "org.apache.harmony.xnet.provider.jsse.TrustManagerFactoryImpl");
            return null;
        }
    });
}
Along with the JSSE class, the gmailSender class was implemented in which its function was to send emails containing banking information to recipients via the users email account. The main method of this class is the sendMail() method which takes in the user email account information and the recipients information as parameters.

This method similar to sendSMS() method retrieves the banking information using the preference manager from shared preferences using “

```java
prefs = PreferenceManager.getDefaultSharedPreferences(context);
```

where each value is stored into a variable where it can be implemented into a String message in order to be sent.

As highlighted earlier, to send an email in the background the application must make use of the user email account using the JSSE provider, the email account credentials must be verified and this is done using the getPasswordAuthentication() method:

```java
protected PasswordAuthentication getPasswordAuthentication() {
    return new PasswordAuthentication(user, password);
}
```

Again similarly to the sendSMS() method the sendMail method retrieves its recipients via the database where it goes through the array of contacts stored and sends a string email with banking information to each of them, an exception is put in place which prevents the application from crashing if the email can’t be sent for whatever reason e.g. no email contacts have been stored in the database.

**Email sender code snippet:**

```java
public synchronized void sendMail(String sender, String recipients) throws Exception {
    try{
        String mes = "Help!! I've completely run out of money and need you to send some via bank transfer please. " + "my account info is: " + BankNameAddress + " account number: " + BankAccount + " Sort Code is: " + SortCode + " 
```
+ "Thank you so much!!";
DataHandler handler = new DataHandler(new
ByteArrayDataSource(mes.getBytes(), "text/plain"));
message.setSender(new InternetAddress(sender));
message.setSubject("Help Money Emergency!");
message.setDataHandler(handler);
addAttachment(mes);
message.setDataHandler(_multipart);
if (recipients.indexOf(',') > 0)
message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(recipients));
else
message.setRecipient(Message.RecipientType.TO, new
InternetAddress(recipients));
Transport.send(message);
}

catch(Exception e){

Overall this activity was extremely difficult to implement, it required a significant amount of
code and the thought process of designing its implementation was lengthy. Fortunately
through persistence and research, its implementation was a success.

3.2.4 Location Marker

Once the problem of retrieving the google maps API V2 key and instillation of Google play
services etc. was overcome the location marker activity wasn’t too difficult to code and only
three additional methods were implemented. One of these was the initializeMap() method,
which initially checks whether the googleMap variable is null, if it is null a map fragment
can’t be created and the user is presented with an appropriate message informing them of
this. If however the variable is not null, the googleMap variable is initialized as a map
fragment thus presenting the user with a map on screen. This map already has most of the
functionality associated with google maps, but does not hone in or move easily to the users
current position so additional code was added.

googleMap.setMyLocationEnabled(true);
googleMap.getUiSettings().setMyLocationButtonEnabled(true);

These lines of code when implemented create a button located in the top right hand corner
of the screen which when pressed hones in on the user’s current location.
The idea behind this activity was to allow the user set markers on areas in which they had travelled to. The activity is able to do this by calling the `onMapLongClick()` method which taking a LatLng as its parameter sets a marker down where the user long clicks the map.

**onMapLongClick() method:**

```java
public void onMapLongClick(LatLng point) {
    marker = googleMap.addMarker(new MarkerOptions()
        .position(point)
        .title("Marker")
        .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED)));
}
```

One more important feature that needed to be implemented was the deletion of markers. This was achieved by calling the `removeMarker()` method which is supplied by the Google API; this method is called in the `onclick()` method of the “clear marker” button. If the user was to press the clear marker button when no markers where present, it could cause an application crash, so a try and catch loop was implemented which lets the application try and delete a marker but if it can't do so i.e. no markers present, it is caught by an exception which stops the app from crashing.

```java
clearMarker.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        try {
            RemoveMarker(googleMap);
        }
    } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```
3.4 Database architecture

After much thought and deliberation, the database implemented into the application was a SQLite database. There were a number of reasons for this,

1. Excessive Functionality - Parse provides a lot of functionality in which the application would not make use of, such as push notification etc. and the functionality in which was needed was quite difficult to implement

2. User satisfaction – many users would not like having their banking information stored externally, even though parse encrypts its databases, external storage was not needed so to reduce the chance of users being unhappy, a local database seemed like the best option.

3. Query limit - Initially, the application would not break the database query limit of 1 million requests per month set by parse, however, if the customer base of the application were to expand, there would be a need to upgrade to a payment option for parse which would result in users having to pay for the application, something which had to be prevented.

4. Need for internet connection – Parse being an external database requires users to have access to the internet something which a SQLite database does not.

5. The lack of information – There is a very limited amount of information and tutorials on how to implement a parse database, which is not the case for SQLite.

Taking all these negatives into account and knowing a SQLite database had the required functionality that the application needed, the choice was simple.
As discussed in the design section, the SQLite database is used to store 3 tables of information:

1. User information
2. Mobile contacts
3. Email contacts

1. This set of information is retrieved from the user on the initial start-up of the application, the user is presented with a screen (Fig.--->), once the information has been input, it is sent to the database to be stored in the user table which is created using:

```java
private String createTable = "CREATE TABLE user(ID INTEGER, firstName TEXT, secondName TEXT," + " address TEXT, securityPin TEXT, location TEXT, emailAddress TEXT, password TEXT);";
```

2/3. The mobile and email contacts are retrieved from the user via the settings menu which was discussed in the design section and then stored in the database by creating tables:

```java
private String createTableContacts = "CREATE TABLE contacts(ID INTEGER, name TEXT," + " number TEXT);";
private String createTableEmail = "CREATE TABLE emailContacts(ID INTEGER, name TEXT, emailAddress TEXT);";
```

The database class also implements numerous methods which gives the application the ability to query, write and re-write all of the information which is stored within it. The method below retrieves all contacts located in the mobile contact table; it uses both java and SQL languages.
Database query method example:

```java
public Cursor getEmailContacts(){
    myDatabase = SQLiteDatabase.openDatabase(myPath, null,
    SQLiteDatabase.CREATE_IF_NECESSARY);
    Cursor args = myDatabase.rawQuery("SELECT * FROM emailContacts;",
    null);
    args.getCount();
    myDatabase.close();
    return args;
}
```
3.5 Login/register re-design

The original designed login and register screen discussed in the design section was not implemented. Initially, due to the application storing banking information, it was thought necessary to continually ask the user to login. This design would prove tedious for users and would affect its uptake and utilisation, so instead of a login page, the user is initially requested to give information to be stored in the User table of the database (fig 28, seen above).

After the initial start-up (after users have entered their information), they will no longer be required to login again. A method is called on each start-up which checks if the user has used the application before; if they have, they are sent directly to the main menu screen where they can proceed to use the app.

However, if it’s the first start-up of the app, they are asked for the user information as identified above, proceeding this the user is then presented with a tutorial animation, which will be reviewed below, ensuring complete understanding of the applications functionality. In regards to the security of the users banking information, in order to access the information the user must provide a 4 digit PIN, meaning the information is still secure with far less work for the user.
3.6 Animation implementation

Due to the number of activities located within the application, it was necessary to include a tutorial which would demonstrate to users, the functionality of some of the harder to understand activities i.e. emergency money activity. The animation activity written in XML uses images located in the drawable.xml files, it links them together giving each a set time to be presented, which to the user looks like an animation on screen, along with pictures, text is presented explaining what each section of the activity does and how to use it.

Animation creation Code snippet:

```
<item android:drawable="@drawable/phone" android:duration="200" />
<item android:drawable="@drawable/phone1" android:duration="200" />
<item android:drawable="@drawable/phone2" android:duration="200" />
<item android:drawable="@drawable/phone3" android:duration="200" />
<item android:drawable="@drawable/phone4" android:duration="200" />
```

When you want to send emergency bank information, press the panic button.

If panic button is pressed acidentally, you can deactivate the app by pressing the deactivate button. You then enter your 4 pin code and press ok to deactivate it. You only have 10 seconds to do so.

Fig. 29 Start-up Animation example 1  
Fig. 30 Start-up Animation example 2
The animation is implemented in such a way that, it is only shown to the user on initial start-up of the application after user information has been collected. The user does not have to re-watch the animation on each start-up, however, if the user wishes to re-view the tutorial they can do so by clicking the help button located in the settings menu.
4. Evaluation

The objective of this section is to review the project as a whole. First, testing of the application will be discussed, followed by a section outlining the successes of the project, and furthermore, the challenges encountered during development. Finally, a section will conclude with a discussion of the opportunities for future development work related to the project and a business model.

4.1 Testing

Without a comprehensive user-base, testing the application's overall effectiveness was inherently left to the developer. Regardless, various usage tests were carried out to evaluate the performance of each individual activity of the application. The logic assumed was that if all the individual features worked as intended, the application itself would be suitable for its intended use as a tool to assist J-1 students while abroad.

4.1.1 Test Model:

Due to not requiring any additional users for testing and time constraints, testing was mainly based on the author’s general observations and experiences of using the application during its development. During this time, various usage tests were conducted and for the emergency money and location marker activities, two different conditions were tested based on internet connectivity strength:

1. Strong Connectivity – Wi-Fi, 3G available

2. Weak Connectivity – 2G and Edge (GPRS) Connections

Aspects of the application tested were location sensor accuracy, speed in which map loaded, budget calculator resulting in a correct figure and all other functionality of the activities were tested in order to ensure they were working effectively.

1. Weak-Connectivity Usage Tests

Location Sensor Accuracy: When GPS was unavailable, the observed readings of the location sensor were less reliable due to the need for internet-based network updates. In some cases, the sensor failed to detect location changes for up to a minute. In addition, readings
were often inaccurate by anywhere up to an approximation of few meters i.e. a road or two off.

Sending of information: When 2G/ EDGE was enabled during testing, it was recorded that there was a delay in the sending of information via email, SMS as expected saw no change. The delay was quite short and thus is not considered to be a significant issue as all the major mobile networks now offer 3G internet connectivity as standard and furthermore, over the next few years, 4G mobile broadband is expected to be implemented across the country.

Based on the results of the testing process, it can be deduced that application Location marker performance is only affected by lack of strong connectivity, all other activities worked effectively without any crashes or unexpected results. These tests were conducted more to assess the worst-case scenario and the related problems could potentially occur in such situations.

2. Strong connectivity Usage tests

Results of strong connectivity usage tests demonstrated positive and reliable performance in all internet related activities.
4.2. Successes

The aim of this project from the outset was to create an application which contained activities which would assist J-1 students while, during and after carrying out their programs in the US. The implementation of a budget calculator, location marker, emergency money and an extensive information centre will benefit users. To this extent, the project is considered a success. Additionally, to the author’s knowledge, no application with similar functionality has been implemented to date.

4.2.1 Ground-up application design

This project was implemented from scratch and was developed in stages, allowing most of the components to be developed modularly; many of the components can be reused in future projects. An invaluable amount of knowledge was accumulated throughout the development process:

4.2.1.1 Android Application

The development of this project was embarked upon without any previous software development experience other than an introduction to programming module and programming techniques 1 and 2 modules taken in 1st and 2nd year respectively, of University. Hence, all technologies other than basic Java and XML used throughout the project were learned from scratch.
4.3 Difficulties

As is the case with any software project, a number of difficulties were encountered throughout the development process:

4.3.1 Approach

One of the biggest challenges was establishing what and how to implement each activity. With no real app development and limited programming experience, the choice of what activity to implement and more importantly how, was certainly a challenge, but through the use of online tutorials and research, the choice of an android application paid off.

4.3.2 Budget calculator result to 2 decimal spots

After designing and implementing an activity to calculate a user’s budget, the challenge of presenting weekly and daily budgets to 2 decimal places proved more challenging than anticipated. Even with the help of online developer forums, inquiries on numerous sites resulted in the same answer being given which was to use: “Text.setText(String.format("Value of a: %.2f", a));”

This solution however, did not work. Fortunately with the support system of University well established, the programming support centre offered assistance and within a few minutes a solution was found and successfully implemented.

4.3.3 Google maps V2 API Key retrieval and importing Google play services

The release of the API V2 and the deprecation of the V1 didn’t help during the development of the application, as nearly all online Google Maps API tutorials were now redundant and the limited numbers of tutorials available for the V2 API quite difficult to follow. Along with limited information two additional difficulties arose when trying to develop the Location marker activity 1) Retrieval of an API key and 2) Import of google play service into the project.

1. The retrieval of an API key is crucial to any application wanting to implement google maps; there are several steps involved before these APIs can be made use of by an application. In the case of an Android application, this key is obtained by registering the package name of the Android project on the ‘Google API Console’.
Unfortunately, this assumed simple set of tasks was in actuality quite difficult as it required the use of command prompt. Many of the online tutorials didn’t go into detail on how to retrieve a key, but after much research and a few failed attempts, a key was acquired. The obtained key must then be included in the project manifest.

The idea behind this task was easy to comprehend; it consisted on downloading the google play services .jar file into the applications workspace and from there import it into the project. Unfortunately, the task itself was not as easy as expected and required some tweaking in order to be successful.

4.3.4 Null value being sent via SMS and email

A key success during the development of the application was the ability to send SMS and emails to a large number of stored contacts in the background. However, the message in both SMS and email were sending perfectly, but upon arrival the banking information stored in shared preferences was null. After much time reading the application code, the problem was located. The shared preferences variables were not being populated correctly and the values were being automatically set as null. The solution was the correct implementation of the preference manager class.

```java
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
String bankAccount = prefs.getString("BankAccount", "null");
```
4.4 Future Work

As with any smartphone application, there is room for future work / updating. The following is one such area:

4.4.1 Location marker improved functionality

The main focus of future work will be on incorporating real-time location tracking of user’s location markers via the introduction of a server. Currently the user can set markers on areas where they have travelled to. A further development might include where instead of the user setting markers via a long click, they will take a photo via the application. When a photo is selected, it will be placed on the location as the marker in place of the normal red marker supplied by the google API.

Additionally, linking the application to a server will allow for real time location tracking thus allowing users of the application view their friend’s markers and also be able to track their position. This supports users within the same cities to know exactly where their friends are (users will be able to become invisible if they wish via the settings menu for privacy reasons). This type of service can already be seen in the Apple iPhone “Find my friends” application, but no similar application is available on the android market.
4.5 Business Model

Building apps for Apple and Android app stores can be highly lucrative venture for developers, but understanding the apps market, competitors and most importantly how the application will generate revenue is a big decision, one which can be clearly laid out with the use of a business model canvas which will be discussed throughout this section. A business model canvas is divided into nine segments.

![Business model canvas](image)

**Fig. 31 Business model canvas**

4.5.1 Key partners

As this application is the first to be developed by the author, no partnerships exist at present. Future partnerships with one or all of the J-1 sponsors would be of great benefit to this application. A partnership with the likes of Go4less will allow the developer bring new capabilities to users. It will provide a good marketing base and opportunity to advertise not only on the google play store, but on the sponsors company
websites, forum etc. With millions of applications present on the android market, any additional advertising will be of clear benefit to the application.

4.5.2 Key activities

These are the key activities that must be completed to support the business model. With the design and implementation of the application already completed, all that is required is to upload the application onto the google play store and try to obtain as many downloads as possible; advertising the app via forums may also been undertaken.

4.5.3 Key Resources

The application makes use of a number of different resources, on the development side of the application the eclipse IDE, android SDK, google maps API and many other tools are used. In order to make the application available to customers, the google play store will also be a vital resource.

4.5.4 Costs

As the application makes use of a number of resources, it would be expected that the costs of development would be high; in fact 99% of the resources used on this application are free. Obviously during the development stages the developers time coding can be seen as a time cost, but no monetary value is assumed except for an android market registration fee, which is a one-time payment of approx. $25. This entitles the developer to upload as many applications to the market as they wish. [16]

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Operating System</td>
<td>0€</td>
</tr>
<tr>
<td>Eclipse IDE</td>
<td>0€</td>
</tr>
<tr>
<td>Android SDK and Tools</td>
<td>0€</td>
</tr>
<tr>
<td>Version Control, BackUp, Server</td>
<td>0€ - ~</td>
</tr>
<tr>
<td>Other Tools</td>
<td>0€</td>
</tr>
<tr>
<td>Android Market Registration</td>
<td>25$ once and that's it</td>
</tr>
</tbody>
</table>

*Fig.32 Resource Cost breakdown*
4.5.5 Value Proposition

For the 7000+ people carrying out a J-1 every year from Ireland, no application is in place to assist them while they are abroad. This application provides clear value to these people, designed as a free application with good functionality; the simplistic design and overall performance of the application should satisfy and create value for all J-1 participants.

4.5.6 Channels

As the application is only available on the android platform, the distribution channels are very limited. Only one channel is available for distributing this application and it is the google play store. The google play store is a digital distribution platform for applications developed for the android operating platform. It allows users browse and download applications developed with the android SDK; as of July last year the store reached over 1 million published apps and over 50 billion downloads which shows how large a customer base it has.

Fig. 33 Google Play Store

4.5.7 Customer relationship

The development of Customer relationships is a three step process acquiring, keeping and growing our customer base. Initially, acquiring customers is down to advertising
good app icon and description. Once downloaded the functionality aesthetic user interface will lead to a high level of Customer satisfaction and retention. Though the application was designed around J-1 participants, many of its activities can be utilised in other countries. Once a substantial customer base has been established, more functionality can be applied to the application, an example of which was discussed in the future work section above.

4.5.8 Customers

Today, the J-1 visa supplements over 40,000 people to travel to the US for study and employment, of which over 7,000 are from Ireland, these are the main target for this application. Along with the significant number of participants, the android platform which the app is designed for is now the most widely used, thus the market for the application has been established already and with good advertising via forums etc. the application could do very well.

4.5.9 Revenue

This application will be free to download and make use of in-app advertising to generate revenue, which research has indicated is better than direct payment. The future of app monetization clearly lies in ad-supported model. A study carried out by Cambridge University computer scientists found that 73% of apps in the android marketplace were free, and of those, 80% relied on advertising as their main business model.

Free apps are also far more popular in terms of downloads. Just 20% of paid apps are downloaded more than 100 times and only 0.2% of paid apps are downloaded more than 10,000 times. On the other hand, 20% of free apps get 10,000 or more downloads. The advantage of more downloads, higher probability of repetition and higher net profit makes releasing the app for free to be the best option.
However, with this type of revenue generator model there are two disadvantages where revenue is not guaranteed and can take sufficiently more time to accumulate profits versus direct payment. [17]
5. Conclusion

This report has outlined the successful design and implementation of an application with the purpose of creating activities which will be of benefit to J-1 students while embarking on their programs in the US. This application implements four very useful activities:

1. Budget calculator - which calculates a user’s weekly/ daily budget
2. Emergency money - which when activated sends banking information to stored recipients
3. Information centre – an activity which holds all contact and other information a participant should require while abroad and finally
4. Location marker – allowing the user to mark all locations travelled to while abroad.

Based on this, the project has been quite successful. An application such as this with no real competitors has excellent market potential and therefore is an appealing candidate for future work.

Having had limited previous experience with Android Application Development and back-end databases, there was a considerable learning curve involved which made it difficult to implement everything that would have been ideal e.g. introduction of a server. However, in undertaking this project, a more rounded knowledge of not just computer science but app development as a whole has been attained.
6. REFERENCES


