

CURRENCY TRADING NEWS ALERT SYSTEM Final REPORT

Word Count: (6131)

Higher Diploma in Science in Computing

Gordon Leary (10352606)

10352606@mydbsi.ie

24/08/2018

# Contents

[Contents 1](#_Toc522804519)

[Abstract 3](#_Toc522804520)

[Acknowledgements 3](#_Toc522804521)

[1. Introduction 3](#_Toc522804522)

[1.1 Aim 3](#_Toc522804523)

[1.2 Approach 3](#_Toc522804524)

[2. Background 4](#_Toc522804525)

[3. Specification & Design 4](#_Toc522804526)

[3.1 Tools & technologies used 4](#_Toc522804527)

[3.2 Project Scope and Objectives 4](#_Toc522804528)

[3.2.1 Minimum Viable Product: 4](#_Toc522804529)

[3.3 Design Approach 5](#_Toc522804530)

[3.4 Design decisions 5](#_Toc522804531)

[3.4.1 Data Access 5](#_Toc522804532)

[3.4.2 Central data store 5](#_Toc522804533)

[3.4.3 Alarm codes from database 6](#_Toc522804534)

[3.4.4 URLs stored in database 6](#_Toc522804535)

[3.4.5 Shared Preferences 6](#_Toc522804536)

[3.4.6 Storage of date and time 6](#_Toc522804537)

[3.5 Preferences – time offset 6](#_Toc522804538)

[3.6 XML to LINQ 7](#_Toc522804539)

[3.7 Passing data between activities 7](#_Toc522804540)

[3.8 Backward compatibility 7](#_Toc522804541)

[3.9 Localisation 8](#_Toc522804542)

[3.10 UI Design 8](#_Toc522804543)

[3.10.1 Graphic Elements 8](#_Toc522804544)

[4. Implementation 9](#_Toc522804545)

[4.1 Development process: 9](#_Toc522804546)

[4.1.1 Refactoring 9](#_Toc522804547)

[4.1.2 Debugging with Logcat 9](#_Toc522804548)

[4.1.3 Regularly re-installing the application. 10](#_Toc522804549)

[4.1.4 Geographic and location issues: 10](#_Toc522804550)

[4.1.5 Device language setting 10](#_Toc522804551)

[4.1.6 Recycler View and Card View 11](#_Toc522804552)

[5. Project Testing and Results 11](#_Toc522804553)

[5.1 Testing 11](#_Toc522804554)

[5.1.1 Manual Testing 12](#_Toc522804555)

[5.1.2 Testing on multiple devices 12](#_Toc522804556)

[5.1.3 Automated UI Acceptance Tests 12](#_Toc522804557)

[5.2 Critical evaluation 12](#_Toc522804558)

[5.2.1 Known issues 13](#_Toc522804559)

[6. Conclusions and Future Work 13](#_Toc522804560)

[6.1 Conclusions 13](#_Toc522804561)

[6.1.1 Learning experiences 13](#_Toc522804562)

[6.1.2 Learning outcomes 14](#_Toc522804563)

[6.1.3 Importance of naming conventions 14](#_Toc522804564)

[6.2 Future Work 15](#_Toc522804565)

[6.2.1 Testing future alarms 15](#_Toc522804566)

[6.2.2 Threading 15](#_Toc522804567)

[6.2.3 Preparing the application for commercial release 16](#_Toc522804568)

[6.2.4 Multiple data sources 16](#_Toc522804569)

[6.2.5 Data verification 16](#_Toc522804570)

[6.2.6 Multiple time zones 16](#_Toc522804571)

[6.2.7 Internet connectivity issues 17](#_Toc522804572)

[6.2.8 Miscellaneous 17](#_Toc522804573)

[7. References / Bibliography 17](#_Toc522804574)

[8. Appendices 19](#_Toc522804575)

[Appendix 1 – Design specification 19](#_Toc522804576)

[8.1 Use case diagram 19](#_Toc522804577)

[8.2 Activities and menus 20](#_Toc522804578)

[8.3 Database schema 21](#_Toc522804579)

[8.4 Classes & activities – part 1 22](#_Toc522804580)

[8.5 Classes & activities – part 2 22](#_Toc522804581)

[8.6 Classes & activities – part 3 23](#_Toc522804582)

[8.7 Flow chart – Create alert on market data item 24](#_Toc522804583)

[8.8 Flow chart – Create personal alert 25](#_Toc522804584)

[Appendix 2 – Sample XML code 26](#_Toc522804585)

[Appendix 3 – Development milestones 27](#_Toc522804586)

[Appendix 4 – Testing 28](#_Toc522804587)

[8.4.1 Sample tests 28](#_Toc522804588)

[8.4.2 Sample test execution 29](#_Toc522804589)

# Abstract

The purpose of this project is to develop an artefact that is the practical application of the knowledge gained during my Science in Computing diploma. I proposed to build a mobile application that would allow a currency trader to filter all weekly scheduled news events in the market place, and to be alerted shortly prior to their announcement. In this study I discuss how the design of the system was approached and how the development work was implemented. To conclude I discuss possible options for future development, a critical evaluation of the application, and the knowledge achieved as a result and its benefits to me as a future IT professional.

# Acknowledgements

I would like to take this opportunity to thank my supervisor Harnik Dhoot for his ongoing support during the development of this project. He has been very helpful, insightful and generous with his time and I have learnt a great deal from working with him.

# Introduction

## 1.1 Aim

To develop a mobile phone application that will alert a trader in the currency market of upcoming fundamental news announcements. The user will be able to download an xml file of all these news release times from a 3rd party broker website and filter these by currency and market impact, save selected events to a watch list, and be alerted by alarm and notification on their android device when such an announcement is imminent. The user will also have the facility to set ‘personal’ alerts.

(See use case diagram appendix 8.1)

## 1.2 Approach

I used a ‘MoSCoW’, Agile type approach to development due to its fixed deadline, aiming to keep the scope of the project small and tight, working towards my goals and objectives, expanding on these, as time permitted, from a list of options in the future work section.

I produced prototypes on which to base further development, which helped to clarify my approach, especially the layout and design of the application.

I did regular backups and commits of all code and documents to GitHub. I also did a lot of research on Xamarin development, utilising various tutorials including those available on Microsoft’s MSDN channel.

# Background

Currency traders take positions in the market with a view to making a profit by buying a currency pair at a given price and then later selling it at a higher price, the reverse, ‘shorting’, is also possible.

Each week economic data is announced at specific, scheduled times. Most dealing brokerages publicise the times of these news events at the beginning of each week, and then the results as they are released.

These news announcements have the potential to move the market significantly against the position held by traders, causing larger than expected losses. Many shorter-term traders stay out of the market completely at these times, deliberately avoiding any possible negative volatility.

Although there are many tools to automate the trading process, many short-term traders who open and close multiples trades in the same day, often do so manually, keeping note of any news announcements and exiting their trades beforehand.

# Specification & Design

All specification & design diagram are in appendix 1.

## 3.1 Tools & technologies used

* Xamarin.Android
* Visual Studio 2017 (for development & unit testing)
* C#
* Android Support v7 AppCompat library
* LINQ to XML
* SQLite
* GitHub
* www.Draw.IO & www.dbdesigner.net(for diagrams)
* Paint 3D (windows 10) for editing flag graphics

## 

## 3.2 Project Scope and Objectives

### 3.2.1 Minimum Viable Product:

* To download a list of all available scheduled news events for the current week from a third-party website (sample XML file included in appendix 2).
* To be able to sort this list by both currency and expected impact on the market.
* To let the user set alarms and notifications on select items of interest from the above, to alert them of an imminent news announcement release.
* User can add personal alerts, to be alerted at times not in the XML schedule.
* User can select amount of time to be notified before the news announcement, e.g. 10 or 15 minutes before the news release.

## 3.3 Design Approach

I developed a basic prototype which has helped to clarify design decisions and gain insight into how a potential user might use the application.

I have also developed several small ‘proof of concept’ apps, which I have then integrated into the main project, including;

* + Phone ringing and vibrating.
  + Alarm manager – to wake up an activity at a future point in time.
  + XML to LINQ – to consume and filter data from XML file.
  + Toolbars – to use multiple toolbars on same activity using ‘appcompat’ for backward compatibility.

Initially all the ‘proof of concept’ applications were developed separately from the main app, in separate Visual Studio solutions. Over time I found it was easier just to develop within the same proj3.3ect, so I would have access to the same methods and database access, that I would use in the final version, less duplication of effort in terms of creating ‘dummy’ or test data and methods.

## 3.4 Design decisions

### 3.4.1 Data Access

The application uses a persistent data layer using SQLite and C#’s object relational mapping (ORM). Schema is outlined in appendix 8.3.

The class DataAccessHelpers contains static methods that can be used throughout the entire application to perform the required data manipulation.

The app was designed to make a call to the database every time data is required. While there are temporary lists of items that have been retrieved from the database, there is no central repository cached in memory. The reasoning behind this being that, because the amount of data that is to be consumed is very small, 50 to 60 items of market data which are only relevant for the current week and because SQLite is essentially reading and writing to a text file, performance would not be an issue. To date this has proved to be the case.

The user can update the market information at any time by selecting the ‘Updata Market Data (XML) option, which will download the current XML file from forexfactory.com, overwriting all previous data. New data is usually published on Sunday night for the week ahead.

All alerts created by the user, both personal alerts, and alerts set on market events are immediately written out to the database, so in the event of an unexpected shut down, e.g. host device runs out of battery power, the user’s data will already be saved.

### 3.4.2 Central data store

In future development, if performance did indeed become an issue, a ‘Singleton’ type pattern could be implemented. This would create a central repository, e.g. ‘DataStore’ which would retrieve data from the database and cache it in memory, where it could be manipulated by all classes as required.

Using the ‘Singleton’ pattern would ensure that only one instance of the repository would be created, avoiding any duplication issues. The data would then be written back to the database upon the exit of the application.

### 3.4.3 Alarm codes from database

When setting the alarm on an event, the android alarm manager requires a unique code or ID as part of the alarm construction. For this I used the autoincremented primary key from the UserAlert table, as SQLite automatically assigns this to each object as it inserts it into a table.

### 3.4.4 URLs stored in database

The URL for downloading xml data file is also be stored in the database, thus providing a location for multiple URLs if required in future development.

### 3.4.5 Shared Preferences

The data in this application is only valid and useful for a week. However, I did want to keep a persistent record of the last date the market data XML was downloaded, to be displayed to the user. I used the Shared Preferences to store this, writing methods to pass in, and retrieve a string of the required date.

### 3.4.6 Storage of date and time

SQLite was chosen for data storage. This involves using C#’s ORM, object relational mapping, whereby tables are created from classes. Mostly this worked perfectly but two issues arose;

* SQLite can’t store C# date-time objects.
* Methods for the display and manipulation of date or time use the C# date-time class.

The class to be used for ORM ***can*** have properties that aren’t allowed by SQLite, they simply have an [ignored] declaration. This allowed the use of a corresponding date-time with each object, but it couldn’t be written to the database.

I added a long ‘ticks’ to store the date and time in an SQLite allowable type. But this did entail a lot of extra converting from long ‘ticks’ to C# date-time object and back, sometimes multiple times per process, not ideal, and certainly not anticipated during the design stage.

#### An example of this conversion flow

XML file download (date & time are stored in 2 separate strings)

🡪 strings are converted to a date-time object for manipulation and display

🡪 date-time object is converted to ticks (long) for database storage.

When objects needed to be read from the database and then manipulated and / or displayed the process would happen in reverse.

### 3.5 Preferences – time offset

One of the primary objectives of the application is to be able;

*To let the user set alarms and notifications on select items of interest from the above, [list of market events] to alert them of an imminent news announcement release.*

All market data that is downloaded has an associated date and time in the XML. But I needed to have some facility for the user to be able to set an amount of time, an offset, so that she can be alerted before the scheduled announcement. This is an important feature, as a currency trader would need adequate time to close out a trade position if required to do so.

The preferences activity lets the user set an offset time of up to minus 60 minutes.

(During testing I found that there was usually a delay of 23 seconds before an alarm would fire).

### 3.6 XML to LINQ

This app is based around utilising a 3rd party XML file of currency market events that is located on an external website, parsing that data, storing it in a database, and then letting the user set alerts on the dates and times that it contains.

I choose to C#’s XML to LINQ for this. It allowed me to download the 3rd party XML, iterate through its contents storing each item in a list, and then writing this list out to the database.

I used C#’s LINQ to provide the user with the facility to filter the market items by currency and by its market impact. For example, she could choose to view only United States Dollar (USD) events, that are expected to have a ‘High’ impact on the market. After which an alarm can be set on any item in the filtered list.

I found LINQ to be very useful and easy to use, especially after studying SQL in our DBS Database module.

### 3.7 Passing data between activities

This initially presented me with quite a challenge. I knew where and when I needed data, but it was unclear how to access and share it. Android does have the facility to pass primitive types with intents as they start a new activity, but I needed to be able to pass objects between activities, e.g. news-alert objects.

The simplest option in the end was to declare a public, static property in the class or activity that the object or data needed to be passed to, and then simply set that property from the class or activity that the data was being passed from.

E.g. in Main Activity the user can select a market event, and after selecting to set a user alert on it, the news-alert object is passed to User-Alert-Activity by assigning the news-alert object to a User-Alert-Alert property.

An obvious solution in hindsight, but one that was a ‘lightbulb’ moment for me personally.

### 3.8 Backward compatibility

The application was designed to be backwardly compatible, to be able to run on as many older devices as possible. To achieve this goal, I decided to use the Android Support v7 AppCompat library.

This allowed me to use the newer Toolbar feature and Material Design in the application, while still being able to target older devices as far back as API 7 (Android 2.1). I used the support toolbar feature extensively to provide menu options at the top, and sometimes the bottom of each activity layout.

I used the Android.Support.V7.App.AlertDialog for user confirmation; ‘OK’, ‘Cancel’ etc.

I also used checkboxes within two alert dialogs, for the user to select their required currencies and to select market impact levels. This was surprisingly difficult to implement as there was very little online reference material available on the subject!

To facilitate the use of Recycler View and CardView, again back as far as API 7, I used the following support libraries:

* Xamarin.Android.Support.v7.CardView
* Xamarin.Android.Support.v7.RecyclerView

I had no issue with the Alarm Manger that is used to set alarms on the user’s selected events, as it has been supported since API level 1.

## 3.9 Localisation

Any application that deals with currency trading is a possible candidate to have a multinational user base and to be used in many different languages, as a result, it was important to design with localisation in mind.

Localisation would also ensure better design and separation of concerns, making it easier to translate the application to other languages and possibly gaining more potential users in other territories as a result.

I stored all UI text in a strings.XML file in the values folder, and then used the following C# code to access it as it was required;

GetString(Resource.String.mainActivity\_top\_toolbar\_title);

In the interim report I had put translating the app into French on the future development list. Once, proper localisation was already in place it was a quick process to convert the strings.xml file to French using Google Translate, to demonstrate the power of localisation and what the app might look like in another language. I would through, recommend using a qualified translator for guaranteed production quality.

## 3.10 UI Design

The UI design has developed in a natural organic manner as development on the application progressed. In the interim report I outlined the menu and activity structure that I was planning to implement. However, I found that as a potential user, once the screen layouts were on my phone in front of me, I wanted to interact with the app differently than anticipated, e.g. I would intuitively go to select a menu option only to find that it was not positioned where I’d expect it to be, so, I have adjusted accordingly.

### 3.10.1 Graphic Elements

An important aspect of any mobile application is its visual aesthetic. If the UI elements are visually pleasing, this will make it easier for the user to interact effectively with the app and enhances the overall user experience.

For the splash screen I created my own design using the windows 10 program ‘Paint 3D’.

For the currency flags that are used in the scrollable views I used icons from:

<https://www.flaticon.com/packs/international-flags>

While appropriate, these were in the wrong size and had issues with transparency that caused noticeable, irregular border lines. Android Asset Studio was a great help in fixing all the above.

<https://romannurik.github.io/AndroidAssetStudio/>

It also provided different screen density versions of each image, these are stored in the Resources folders: ‘mipmap-hdpi’ to ‘mipmap-xxxhdpi’. This means that if application is run on devices with different screens sizes or densities, the appropriate sized graphic will be displayed, avoiding quality degradation issues such as pixilation, when a small graphic is enlarged too much.

I used a colour tool to find complementary colours, and different shades of the same colour:

<https://www.colorhexa.com/0070bf>

And used the android code in the button\_state.xml file to set corners, colour gradients and to call another xml to change button colour, when a button was pressed, e.g.

android:state\_pressed="true"

# Implementation

## 4.1 Development process:

### 4.1.1 Refactoring

Refactoring became an important part of my development process. Each day I would spent some time reading through all my code to see where it might be improved. Most often it would be in terms of trying to make my method names more accurately descriptive. I would also try, but not always succeed, to subdivide any larger methods, so as far as possible, each method only performs one function. Reading ‘Clean Code’ by Robert Martin was an inspiration here.

### 4.1.2 Debugging with Logcat

Previously, when developing smaller applications during my DBS course I would use Toasts to display debugging type information in addition to setting breakpoints etc. This very quickly became unworkable in this larger scale development. During my own research I came across Android Debug Log and used this extensively throughout my work. I still used breakpoints as well, but it was sometimes invaluable to have a record that I could search or scroll through to ascertain was the control flow or output performing as expected.

### 4.1.3 Regularly re-installing the application.

Usually when I would run the application, using Visual Studio, I would either run it in debug mode, or run without debugging. Regardless of whether I was deploying to the emulator or to a real phone the app would avail of whatever data was still available from a previous deploy. Eventually there was no virtual memory left on the emulator which forced the deleting of unwanted apps and a clean install of the app. This exposed several hidden issues, including a crash that was caused by a call to the database before the appropriate tables were ready to be accessed.

E.g. method:

GetAllNewsObjectDataFromDatabase()

This is called to get all the required data from the database, and then passed to Recycler View Adapter to be displayed on the screen.

Going forward part of my development process was to;

* Uninstall the app regularly, to be able to test in a clean, blank environment.
* Ensure the appropriate checks are in place, e.g. checking that a table exists before calling a method that tries to read or write to it.

### 4.1.4 Geographic and location issues:

Currency trading is a global activity and the website that provides the XML file of weekly market news events, ForexFactory.com is an American site. To simplify zone type issues, they release their XML in GMT, Greenwich Mean Time.

During DST, daylight savings time, we are one hour ahead of GMT time.

In method:

ConvertXmlAndStoreInDatabase of DataStore.cs

I check to see if the application is currently in DST using:

DateTime.IsDaylightSavingTime Method ()

If it is, then an hour is added to the XML data as it is stored in the database, to bring it in line with DST. When not in DST nothing is added, so the time simply remains the same. This only applies to market events and alerts and does not apply to personal alerts set by the user.

Currently the application would only be suitable for countries that are in the same time zone as Ireland and the UK, and that follow DST. In future development a facility could be added to update the market alert times to be correct for whatever time zone the device running the application is in.

### 4.1.5 Device language setting

Another issue that presented itself very unexpectedly was that of which version of English the host device is set to. During development I have been testing the app on both my own phone, Samsung S7 (Oreo) and the emulator (Nougat). Unknowingly the Samsung had its language set to English (Ireland) while the emulator was set to English (American).

In my method: Convert\_Strings\_DateAndTime\_To\_SingleDateTimeObject

I used the C# method: DateTime.Parse(dateAndTimeString)

to convert a string to a date-time object, caused the app to app to fail on the Samsung phone, a difficult problem that took some time to locate and solve, and was eventually fixed by creating a CultureInfo object:

DateTime.Parse(dateAndTimeString, new CultureInfo("en-US"));

### 4.1.6 Recycler View and Card View

In my interim report, implementing Recycler view and Card view were listed as items for possible future development. Whilst carrying out some additional, preliminary research, I was pleasantly surprised to discover that implementing these would not involve a large learning curve as they are very similar to the Custom Adapter approach that we covered in our Mobile Application Development module. So, for my own personal learning as well as Microsoft’s recommendation to use these instead of the previously mentioned option and others such as List view, I decided to press ahead with their implementation. I found that they have provided enhanced performance, making for the smoother scrolling through market data events and their accompanying flag graphics.

(Please see appendix 3 – for approximate chronological order of development milestones and accompanying worklog.)

# Project Testing and Results

## 5.1 Testing

The importance of testing in the development of any system or application cannot be overstated. With a variety of testing methodologies to choose from I decided to implement a manual testing process for several reasons.

In our advanced programming module, we had exposure to TTD, test driven development and the work of ‘Uncle Bob’, Robert Martin. In time, and with a more experienced skill set, TTD is something I would aspire to, writing your tests before your code seems to be a very progressive way to approach development.

I would prefer to be using test driven development, TDD, but would need to implement the system using an MVC or MVVC type pattern to achieve this, beyond the current scope of this project, but this has been included as a possible consideration for future work.

We also had exposure to Unit Testing, a methodology which should be employed by all developers, regardless of whatever other testing methods are being utilised at higher levels within their team or organisation. This requires that your code be developed in such a way as to facilitate such testing.

In my application I have endeavoured to write my code to enable unit testing, where possible writing my methods and functions so that they return a value such as a true or false bool, or an int containing the number of rows updated in the database etc. I have also ‘wired up’ a unit test project within Visual Studio, with some dummy tests.

A large percentage of my application uses technologies that I’ve researched myself, e.g. utilising Android’s date and time pickers, phone notifications etc. As result, it has been impractical to write tests, or adapt my code ahead of time for such features. I would though, regard this as a priority for any future development.

### 5.1.1 Manual Testing

During development, anytime I would introduce a new feature, e.g. adding an extra item to a menu, I would go through a process of not only trying out the new feature, but also verifying that all the previously working features still worked as expected. Choosing manual testing, in the end, was a process of documenting my natural, personal development style.

I researched a lot of tutorials on YouTube, where different tutors presented how they laid out their manual tests in an Excel format, and took that as a basis for my own Excel based, manual test layout.

(See appendix 4 for sample screen shots from accompanying excel test file).

### 5.1.2 Testing on multiple devices

For future development of an application that is aiming to be released commercially it would be desirable to test the app on as many different devices as possible, to access both functional performance and visual aesthetics. While it is possible to run many different emulators in Visual Studio, a time-consuming task, a commercial option such as Microsoft’s own ‘Visual Studio App Centre’ which includes Xamarin Test Cloud, (appcenter.ms) would be helpful. This lets the developer test their app ‘in a hosted device lab with 1000s of real iOS and Android devices. You’ll receive test results, full-resolution screenshots of every step, along with performance metrics’ (MSDN).

### 5.1.3 Automated UI Acceptance Tests

Another invaluable option for testing in future development would be the use of Xamarin.UITest. This allows the automation of UI acceptance tests and can be used within Visual Studio. Its most impressive feature is that in can simulate a user interacting with all the user interface components such as button presses, swipes, gestures etc., providing a very thorough and robust testing environment.

(Please note, manual test cases are included in an accompanying Excel file.)

## 5.2 Critical evaluation

To provide a critical evaluation of the project and to place it in the context of related work requires I believe, personal objectivity without being overly self-critical. In terms of software development, it is useful to use commercially produced work as a benchmark to measure against.

Yes, I believe I have achieved the outcome I set in my proposal, of producing a mobile application in the time allocated with the previously discussed objectives and functionality, but is it comparable to a commercial product?

In a way that is an unfair question and comparison, but it is still a useful one. Obviously, an in-experienced development team of one, can’t compete with the resources, budget and experience of a multi person team or department. But the question can still point the way forward to the next steps in future development, which can be strategically selected to play to one’s strengths.

The application would benefit from increased functionality, like the facility to automatically convert all data to whatever time zone the user’s device is currently in, wherever that may be globally. But rather than pursuing such a grand goal, a more feasible approach would be to simply market the app at users in the appropriate time zone.

A well-funded team could also purchase a cloud-based device testing service such as Microsoft’s own ‘Visual Studio App Centre’ to test their application on all types and ages of devices, whereas a self-produced app might have to rely on the user acceptance testing of friends and colleagues.

One area where a personal application might be able to compare favourably against commercial development would be code quality. This would all depend on the experience of the developer. For example, I have developed more of an understanding and appreciation of ‘threading’ during the development of this project, and with more time and practice this could greatly enhance the performance and stability of the application.

I have discussed threading and supporting multiple time zones further in the ‘future development’ section.

### 5.2.1 Known issues

There are several known issues which I have discovered during testing and which I have been unable to resolve in the time remaining:

* If the app is not connected to the internet or Wi-Fi it will hang if the following options are selected:
  + Update XML (Market Data). (Main Activity top menu, option 1).
  + Preferences - Set (number of minutes required before alert). This is listed as an item for future work, as it is not necessary to make a call to the data provider to set this, it could be set from data already stored in the database.
* If a user alert is set on market data whose date-time has already passed, then the alert fires instantly, a check needs to be added to only allow the user to set alerts on future market events.
* Multiple alerts are allowed for the same date-time.
* Alarm won't sound if Spotify is playing – notification will display.
* Alarm – sounds 23 seconds after due scheduled time.
* The Preferences – option, that allows the user to adjust all the market times zero to minus 60 minutes – this updates all data shown in Main Activity, but it does require a new download of the xml data – I would prefer to just adjust the already downloaded data that is stored in the database.

# Conclusions and Future Work

## 6.1 Conclusions

### 6.1.1 Learning experiences

I have found the process of developing my application during this project to be both a challenging and an immensely beneficial learning experience. In my journey to date, I have become very aware of the fact that software development is as much an art and skill set that requires regular practice to gain proficiency.

During the daily programming work that I undertook during the development of the project, I found that many of the concepts that I had learnt in my course over the last two years, and how they are related to each other became much clearer, especially OOP and passing data between activities in Android.

Simply working on the development of a larger scale application requires a different thought process. It is easy to pay lip service to such concepts as code reuse, naming conventions or ensuring that every method has one primary function or objective, when working on smaller, assignment size applications, but they become essential allies as the scale of your project increases.

As an example, an insight I gained was where previously I had a challenge passing anything other than primitive types between Android activities using Intents. Initially I used methods that had no actual functionality to pass objects between activities, inefficient and not best practice. Finally, arriving at the understanding that I could use properties, ‘getters & setters’ to achieve my goal.

In terms of becoming a future practicing IT professional and on-going progress as a learner, the greatest benefits have been increased confidence in my ability to research and solve larger programming challenges and the ability to be able to work with a problem without getting overwhelmed, even if there is currently no solution in sight.

### 6.1.2 Learning outcomes

I believe that I have achieved the outcomes that I put forward in my proposal at the outset of the project;

* To develop a full mobile application from concept to implementation, with an intuitive, GUI user interface.
* To be able to retrieve and manipulate XML data from a third party, using SQLite & LINQ to XML.
* To develop an alert, notification system using the android alarm manager to be able to call services and wake up applications at a scheduled time.
* To implement the Xamarin Recycler View, displaying larger lists of data to the user and providing sorting mechanisms to select what is displayed.
* To develop insight into my personal production velocity, refine my time and project management skills.
  + To this end I kept a detailed work log where I recorded my daily activities and rough work (please see attached Word document). This has proved to be a great reference and very useful in terms of working out my personal velocity, and how long it might take me to complete similar tasks in the future.

### 6.1.3 Importance of naming conventions

Not having ever attempted a project of this scale I would admit that I hadn’t truly grasped the importance of having a firmly established naming convention at the outset, before any coding starts.

The Visual Studio IDE is very efficient for re-naming elements consistently throughout the application. But often it didn’t pick up name changes in the XML files, making it very easy to add a difficult to find bug if you renamed something that was referenced in several XML files.

According to Uncle Bob (Robert Martin) in his ‘Clean Code’ book, when we code

‘the ratio of time spent reading vs. writing is well over 10:1. We are constantly reading old code as part of the effort to write new code.’

The more descriptive and relevant my naming became the easier it was to follow the flow of my code and to locate items. I refactored regularly.

## 6.2 Future Work

Below are listed items for possible future development which I believe would be valuable additions to the application time permitting and a possible point of reference for another developer to continue the project.

### 6.2.1 Testing future alarms

I manually tested that alarms would fire at the expected time that they were set for. However, I would be more comfortable with some other way to verify this. I tried setting an alarm for a time in the future, and then resetting the actual time on the device, emulator and real phone, but this didn’t work.

Also, while I could set an android alarm, and record that in the database myself, I couldn’t get android to programmatically return what alarms were currently stored as pending in the android alarm manager ecosystem.

### 6.2.2 Threading

I implemented threading in the following methods;

* SetAlarm (UserAlertsActivity)
* DeleteAlarm (UserAlertsActivity)
* BtnSetOffset\_Click (Preferences Activity)

The UserAlertsActivity methods both use the AlarmManager class to set and delete alarms. It was possible for me to implement these on separate threads as the application didn’t need to wait for any returned response before continuing. If they remained on the UI thread, the UI would freeze until they had completed their work. Also, the debug console window was warning that there were too many processes running on the UI thread.

I also tried to implement threading in the ‘Update XML’ option (Main screen, top menu, 1st option). When the user selects this option, the application downloads an XML file from the ForexFactory.com brokerage site. During the download, the UI temporarily freezes, preventing any other user interaction with the application. I experimented with putting this process onto a separate thread, which did prevent the UI from freezing during the download, but had to abandon the idea because it caused other issues downstream as I hadn’t designed the application with threading in mind from the outset, e.g. methods were getting called before the required data that was returned by the download thread had become available, causing timing issues in terms of updating the screen display and database access

In any future development I would like to implement threading throughout the application. While threading would bring performance benefits it would require some changes to the architecture of the application to avoid race conditions.

### 6.2.3 Preparing the application for commercial release

An obvious step, but one that requires some planning and work, several steps are involved;

* + Protecting application content, by adding Google Play’s licensing key to the app.
  + Obfuscating the code to prevent de-compilation, also reducing the size of the app.
  + Remove debugging methods such as ‘Log.Debug’ to save host CPU resources.
  + Reduce the size of the app package, to only include the resources and parts of the .Net libraries that are being used, using;
    - Xamarin.Android linker on the managed code.
    - ProGuard on the Java bytecode. (MSDN)
  + Ensure that all app metadata is formatted to conform to Google Play’s standards.

### 6.2.4 Multiple data sources

The app relies on one data provider, ForexFactory.com for its XML data (news announcement schedule). While this is a reliable and well-established resource some redundancy would be desirable.

Already included in the database is the URL table which stores the URL from which to download the XML data. As new sources become available they too can be stored here.

A web service could be developed and deployed to Azure. The app could then call an API to enable downloading from multiple sources which could be selected from the choice of URLs stored in the URL table.

### 6.2.5 Data verification

Ideally when XML data is consumed, an XML parser would first be used to verify that that the data is correctly formed. Unfortunately, ForexFactory.com does not provide such a parser or schema for their proprietary format XML. This might be less of a concern if the application was being developed by the business inhouse, but in this third-party type context the development of such a parser would provide an additional safe guard to the stability and performance of the application.

### 6.2.6 Multiple time zones

As discussed previously, any application related to currency trading by its very nature has the potential to have a global userbase. For this app to work globally it would need the facility for all data to automatically be converted to the local time zone of the user’s device. I had to programmatically adjust for time differences just to work with Greenwich Mean Time and Daylight Savings Time. The ideal solution would be to interact with a 3rd party API that could calculate the global time differences automatically.

### 6.2.7 Internet connectivity issues

* Implement a timeout type function surrounding the XML download to prevent issue if there is a problem with the host device’s Wi-Fi or internet connection, possibly using test data already in the Assets folder until downloading is possible again.
* Implement a new method for Preferences - Set (number of minutes required before alert) so that it does not make a call to the data provider to offset the time data. This could be set using data already stored in the database.

### 6.2.8 Miscellaneous

* Import and export watch-list to csv file, possibly sharing via email.
* Landscape and portrait modes for different devices.
* Integrate application with Google calendar API – flag user time scheduling conflicts.
* Tracker facility for the user to be able to keep a record of their market trades and profitability, and to forward reports to a selected email.
* Convert application to a Model, View, Controller (MVC), or Model View, View Model (MVVP) type structure, to have greater separation of concerns and to enable more comprehensive testing of the application.

# References / Bibliography

Deitel, Deital, 2014. *Visual C# 2012 How to Program* 5th ed.  H. Marcia et al., eds., Harlow: Pearson Education.

Gamma, Helm, Johnson, Vlissides, 1995. *Design Patterns: elements of reusable object-oriented software* 13th ed. Erich Gamma et al., eds., Massachusetts: Addison-Wesley Publishing Company.

Leibowitz, M., 2015. *Xamarin Mobile Development for Android Cookbook* 1st ed. A. Banerjee et al, eds. Birmingham: Packt Publishing Ltd.

Albahari, Albahari, 2016. *C# 6.0 Pocket Reference* 1st ed. B MacDonald et al., eds., California: O’Reilly Media Inc.

Sommerville, I., 2015. *Software Engineering* 10th ed. H. Marcia et al., eds., Harlow: Pearson Education.

Murphy, Mark L., 2018. *The Busy Coder's Guide to Android Development* 8.10th ed. Commons Ware

Xamarin University. Available at: <https://university.xamarin.com> [Accessed May 2018].

Xamarin Android Tutorials by Joe Rock. Available at: <https://www.youtube.com/channel/UCFH-Tn6ggt9yLWJ4V9NizZA> [Accessed May 2018].

C# Training Tutorials. Available at :<https://www.youtube.com/user/IAmTimCorey> [Accessed May 2018].

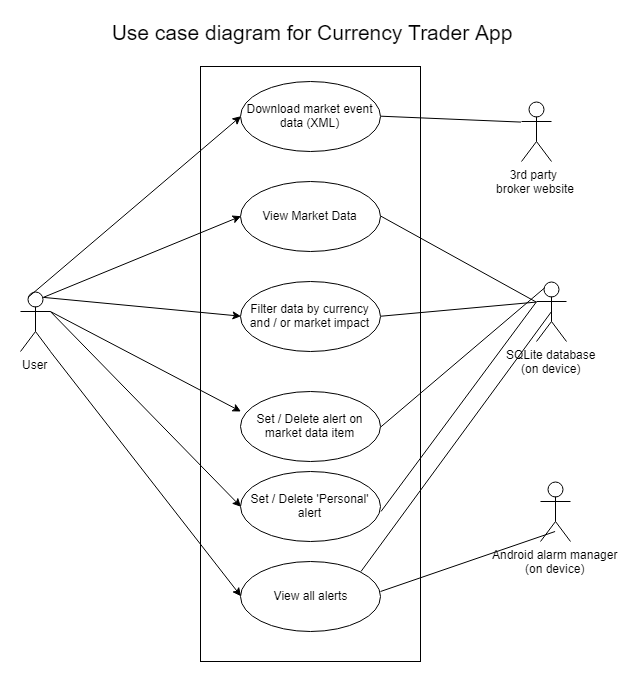
Xamarin Documentation: Available at: <https://docs.microsoft.com/en-us/xamarin/#pivot=platforms&panel=Android> [Accessed May 2018].

Xamarin Android Udemy Course by Grant Klimartys. Available at: <https://www.udemy.com/xamarin-android-a-master-guide-to-mobile-app-development/learn/v4/overview> [Accessed May-June 2018]

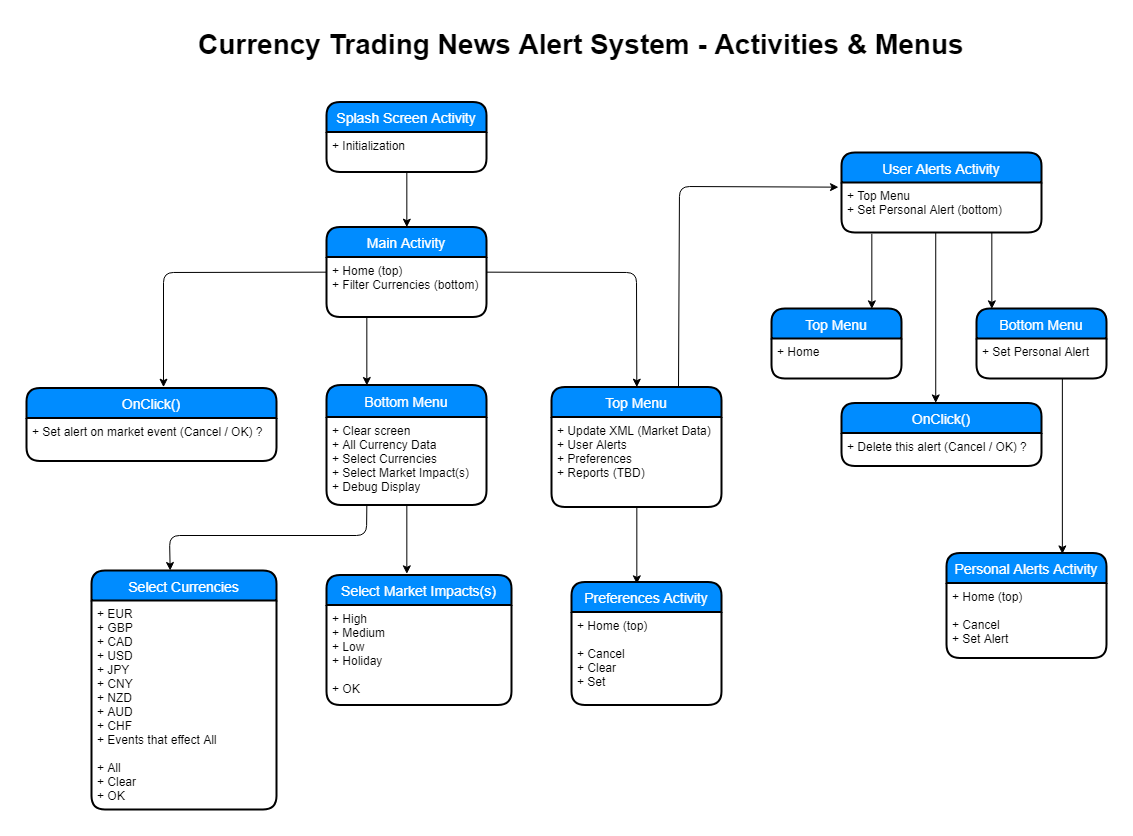
# Appendices

## Appendix 1 – Design specification

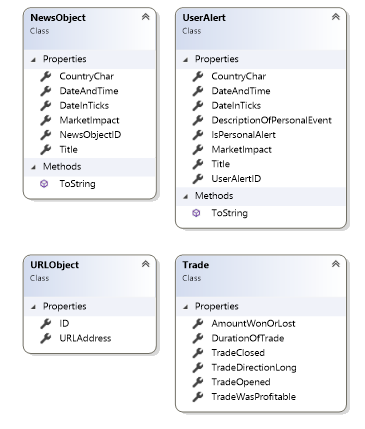
### 8.1 Use case diagram



### 8.2 Activities and menus

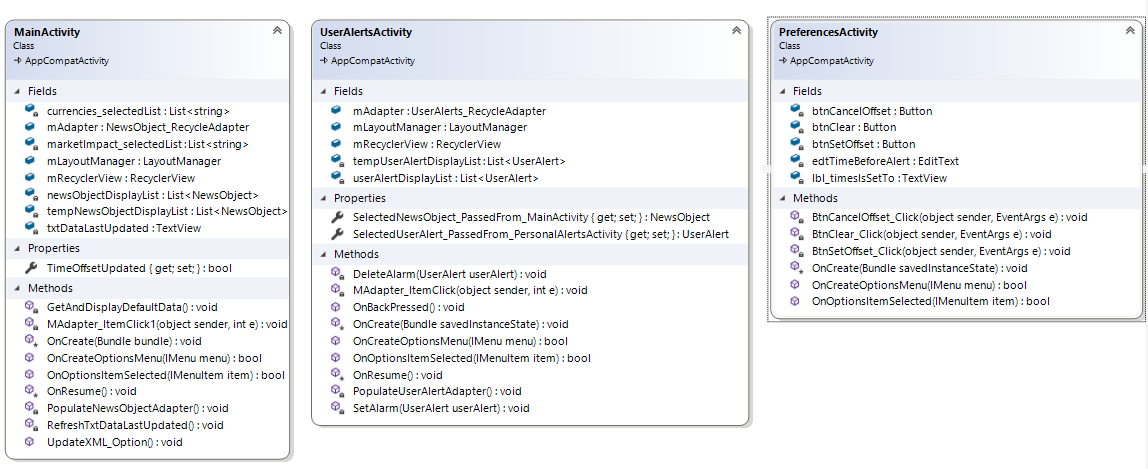


### 8.3 Database schema

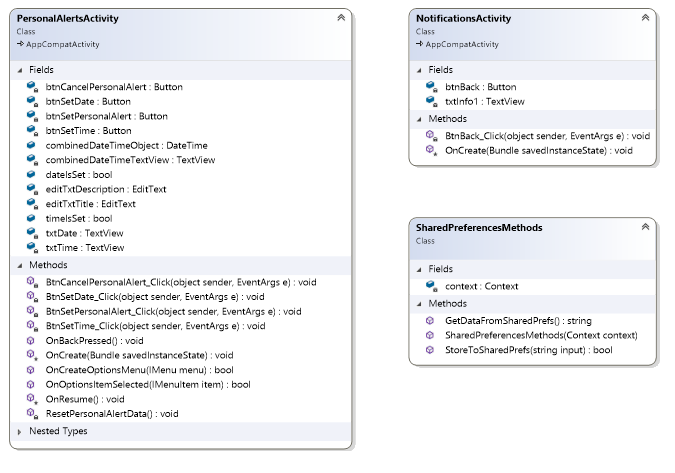


* Using C#’s object relational mapping (ORM).
* UserAlertID in UserAlert table is declared as the primary key and used as the unique code required by android to set an alarm using alarm manager.
* Nb. Trade class is only for use in future possible development.

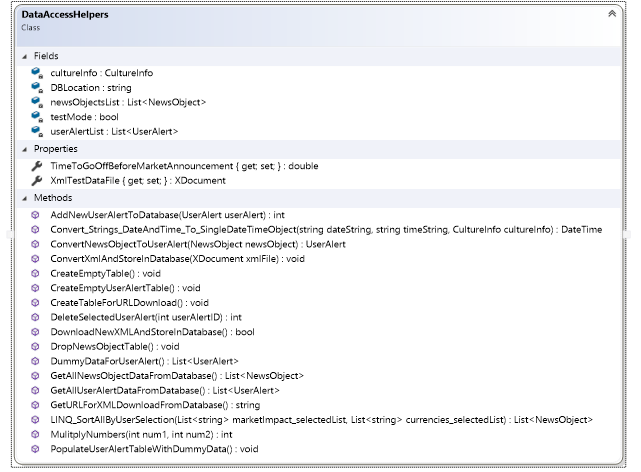
### 8.4 Classes & activities – part 1



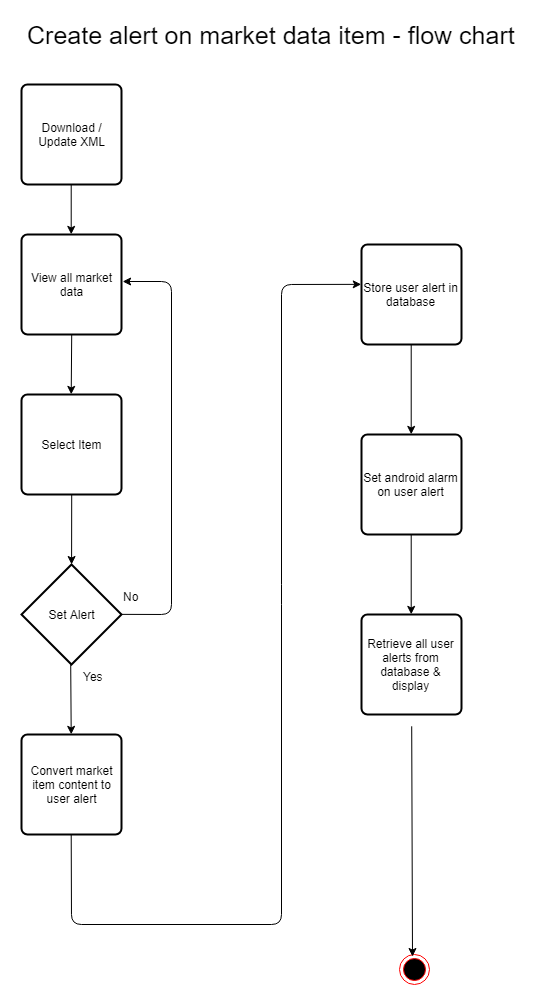
### 8.5 Classes & activities – part 2



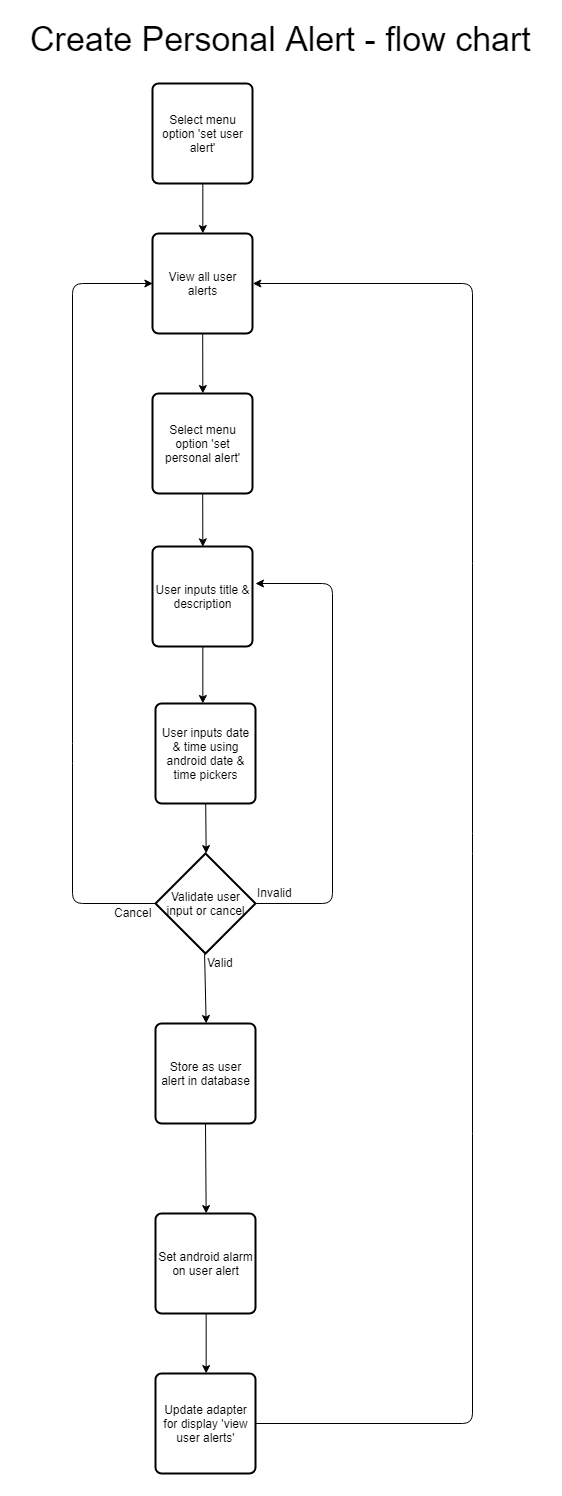
### 8.6 Classes & activities – part 3



### 8.7 Flow chart – Create alert on market data item



### 8.8 Flow chart – Create personal alert



## Appendix 2 – Sample XML code

Sample XML of fundamental news releases published weekly. Available at: https://www.forexfactory.com/ffcal\_week\_this.xml

<?xml version="1.0" encoding="windows-1252"?>

<weeklyevents>

<event>

<title>FPI m/m</title>

<country>NZD</country>

<date><![CDATA[04-15-2018]]></date>

<time><![CDATA[10:45pm]]></time>

<impact><![CDATA[Low]]></impact>

<forecast />

<previous><![CDATA[-0.5%]]></previous>

</event>

<event>

<title>Rightmove HPI m/m</title>

<country>GBP</country>

<date><![CDATA[04-15-2018]]></date>

<time><![CDATA[11:01pm]]></time>

<impact><![CDATA[Low]]></impact>

<forecast />

<previous><![CDATA[1.5%]]></previous>

</event>

<event>

<title>WPI m/m</title>

<country>EUR</country>

<date><![CDATA[04-16-2018]]></date>

<time><![CDATA[5:52am]]></time>

<impact><![CDATA[Low]]></impact>

<forecast><![CDATA[0.4%]]></forecast>

<previous><![CDATA[-0.3%]]></previous>

</event>

</weeklyevents>

## Appendix 3 – Development milestones

Approximate chronological order of events during development (taken from accompanying worklog).

* Proof of concept app - vibrate, alarm/ringtones.
* Proof of concept app - local-notifications (NotificationsCOMPAT), Broadcast, Receivers, Alarm Manager.
* Proof of concept app (console) - XML & LINQ
* Splash screen using appCompat,
* 2 toolbars in one activity / appCompat Toolbar
* Material Theme / appCompat themes
* Button state xml
* DataAccessHelpers Class
* Menu options wired up
* NotifyDataSetChanged
* Proof of concept app - DatePicker & TimePicker
* Localisation
* SQLite implementation
* Alert Dialogs (including problematic check boxes)
* Shared Preferences
* LINQ sorting
* Custom Adapter implementation
* Language setting issues, English(Ireland) vs English(American).
  + Causing DateTime.Parse(dateAndTimeString) to fail.
  + Solution:
    - * DateTime.Parse(dateAndTimeString, new ultureInfo("en-US"));
* Recycler-View & Card-View implementation
* Graphics for currency icons.
  + Download individually & then converted to different mimap sizes (mimap-hdpi/ mdpi / xhdpi / xxhdpi / xxxhdpi)
  + using Android Asset Studio: https://romannurik.github.io/AndroidAssetStudio/
* Graphic for splash screen – used Paint3D
* Alarm manager ticks and milliseconds issues.
* Database has stored datetime using ticks (recommended best practice).
  + - need C# object DateTime for display & for AlarmManager - requires milliseconds (long)
* Update XML to be in line with daylight savings hours & if it is not then adding an hour to the time stored.
* Preferences – added an option so the user can adjust all the market times zero to minus 60 minutes.
* Testing, refactoring, issues with visual studio updates and emulator.

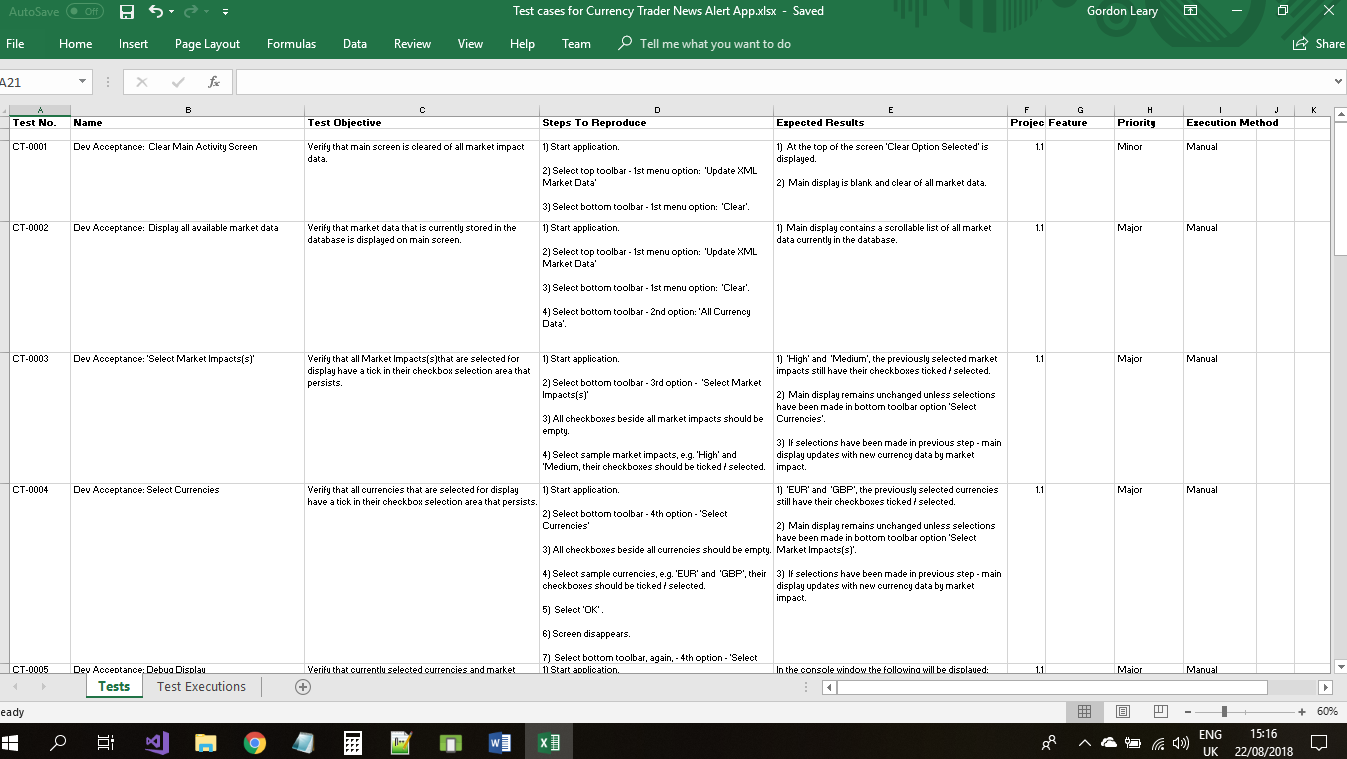
## Appendix 4 – Testing

Tests were performed on android emulator within Visual Studio:

x86 Nougat (Android 7.1 API 25 American English).

(Full tests are in accompanying excel file.)

### 8.4.1 Sample tests



### 8.4.2 Sample test execution

