

CURRENCY TRADING NEWS ALERT SYSTEM Final REPORT

Word Count: ()

Higher Diploma in Science in Computing

Gordon Leary (10352606)

10352606@mydbsi.ie

24/08/2018

# Contents

[Contents 1](#_Toc522441854)

[Abstract 3](#_Toc522441855)

[Acknowledgements 3](#_Toc522441856)

[1. Introduction 3](#_Toc522441857)

[Aim 3](#_Toc522441858)

[Approach 3](#_Toc522441859)

[2. Background 3](#_Toc522441860)

[3. Specification & Design 4](#_Toc522441861)

[Tools & technologies used 4](#_Toc522441862)

[Project Scope and Objectives 4](#_Toc522441863)

[Minimum Viable Product: 4](#_Toc522441864)

[Design Approach 4](#_Toc522441865)

[Design decisions 5](#_Toc522441866)

[Data Access 5](#_Toc522441867)

[Central data store 5](#_Toc522441868)

[Alarm codes from database 6](#_Toc522441869)

[URLs stored in database 6](#_Toc522441870)

[Shared Preferences 6](#_Toc522441871)

[Storage of date and time 6](#_Toc522441872)

[XML to LINQ 7](#_Toc522441873)

[Passing data between activities 7](#_Toc522441874)

[Backward compatibility 7](#_Toc522441875)

[Localisation 8](#_Toc522441876)

[UI Design 8](#_Toc522441877)

[Graphic Elements 8](#_Toc522441878)

[4. Implementation 9](#_Toc522441879)

[Development process: 9](#_Toc522441880)

[Refactoring 9](#_Toc522441881)

[Regularly re-installing the application. 10](#_Toc522441882)

[Geographic and location issues: 10](#_Toc522441883)

[Device language setting 11](#_Toc522441884)

[Recycle View and Card View 11](#_Toc522441885)

[5. Project Testing and Results 11](#_Toc522441886)

[Testing 11](#_Toc522441887)

[Manual Testing 12](#_Toc522441888)

[Testing on multiple devices 12](#_Toc522441889)

[Automated UI Acceptance Tests 12](#_Toc522441890)

[???????????? Critical evaluation ??????????????? 13](#_Toc522441891)

[Known issues 13](#_Toc522441892)

[6. Conclusions and Future Work 13](#_Toc522441893)

[Naming conventions 13](#_Toc522441894)

[Future Work 13](#_Toc522441895)

[???????? Run in multiple time zones ?????????? 14](#_Toc522441896)

[Testing future alarms 14](#_Toc522441897)

[Threading 14](#_Toc522441898)

[Preparing the application for commercial release 14](#_Toc522441899)

[Multiple data sources 15](#_Toc522441900)

[Data verification 15](#_Toc522441901)

[Conclusions 15](#_Toc522441902)

[Learning experiences 15](#_Toc522441903)

[Learning outcomes 16](#_Toc522441904)

[Milestones Achieved 16](#_Toc522441905)

[7. References / Bibliography 17](#_Toc522441906)

[8. Appendices 18](#_Toc522441907)

[Appendix 1 – Design specification 18](#_Toc522441908)

[Activities and Menus 19](#_Toc522441909)

[Schema for Currency Alert System Database 20](#_Toc522441910)

[Class Diagram 20](#_Toc522441911)

[Appendix 2 – Sample XML code 21](#_Toc522441912)

# Abstract

# 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

## 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.

## Approach

I am using an ‘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, time permitting, from a list of options in the future work section.

I am doing regular backups and commits of all code and documents to GitHub. I am also doing 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.

## 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

## 

## Project Scope and Objectives

### 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, select items of interest from the above and store them in a watch list.
* To set alarms and notifications on the user’s mobile device based on their watch list, to alert them of an imminent news announcement release.
* User can add personal alerts to the watch list 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.

## 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 project, so I would have access to the same methods, database access, that I would use in the final version, less duplication of effort in terms of creating ‘dummy’ or test data and methods.

## Design decisions

### Data Access

The application uses a persistent data layer using SQLite, schema outlined below.

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.

### 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.

### 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.

### URLs stored in database

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

### 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.

### 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.

### 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.

### 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.

### 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 RecyclerView 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.

## 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.

### 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, I wanted to interact with the app differently than anticipated, when the screen layouts were on my phone in front of me. 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 made adjustments accordingly.

### 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 other xml to change button colour, when a button was pressed, e.g.

android:state\_pressed="true"

# Implementation

## Development process:

### 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.

### 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 ReCycle 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.

### 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.

### 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"));

### Recycle View and Card View

In my interim report, implementing Recycle 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.

# Project Testing and Results

## 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.

## 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.

## 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’.

## 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.

# ???????????? Critical evaluation ???????????????

## Known issues

* 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.
* The ‘Update Market Data’ (Main Activity top menu, option 1) hangs if there is a problem with the host device’s Wi-Fi or internet connection. An option would be to have a timeout type function surrounding the XML download, and to use test data already in the Assets folder until downloading is possible again.

# Conclusions and Future Work

### 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 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.

## Future Work

Below are listed items for possible future development which I believe would be valuable additions to the application time permitting.

## ???????? Run in multiple time zones ??????????

### 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.

### Threading

I implemented threading in the following methods in UserAlertsActivity;

* SetAlarm
* DeleteAlarm

These 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 was 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.

### 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.

### Multiple data sources

The app relies on one data source, 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 call an API to enable downloading from multiple sources which could be selected from the choice of URLs stored in the URL table.

### 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 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.

* Export and export watch-list to csv.
* 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 & 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.

## Conclusions

????? Learnt different technologies through the development of proof of concept apps.

### 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. Initially I used methods that had no actual functionality to pass objects between activities, inefficient and not best practice. Finally, arriving at the understanding I could use properties, and ‘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.

### 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 Recycle 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 (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.

## Milestones Achieved

1. Small proof of concept app – to demonstrate a working example of calling an alarm or notification at a pre-scheduled time, e.g. setting mobile phone to ring, vibrate and display an on-screen message at a set time.
2. Console app to demonstrate retrieval and manipulation of the 3rd party XML news event list, e.g. convert XML data to list & sort by currency, date or impact.
3. A non-functional mock-up / prototype to;
   1. display design and flow of app from one screen to another,
   2. use of menus, graphics and theme templates,
   3. hard-coded sample data.
4. Prototype produced with basic functionality.
5. Wire-up Date-Picker & Time-Picker ‘proof of concept’ app into Personal Alerts.

# 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].

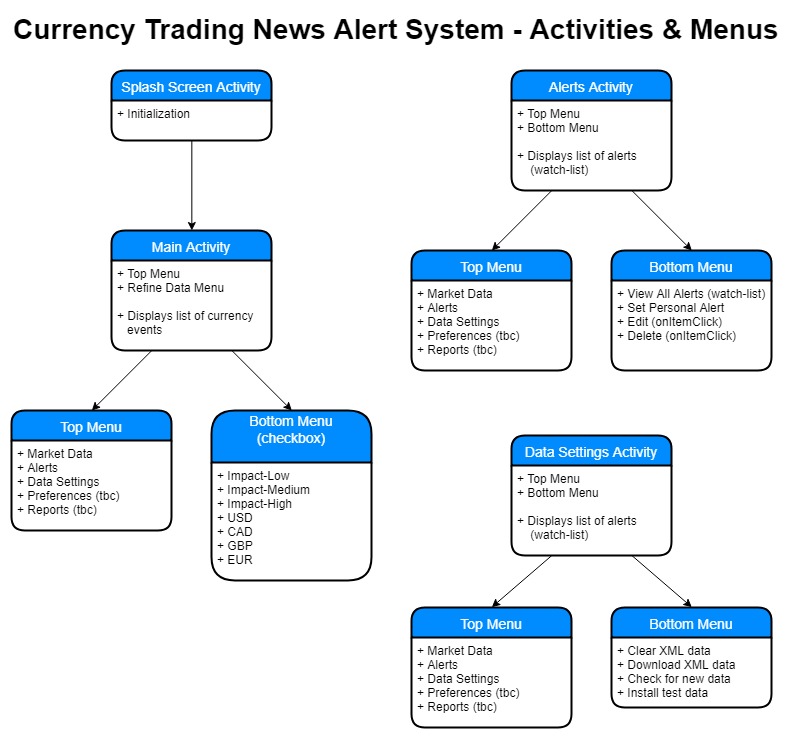
Xamarin Documentation: Available at: <https://docs.microsoft.com/en-us/xamarin/#pivot=platforms&panel=Android> [Accessed May 2018].

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]

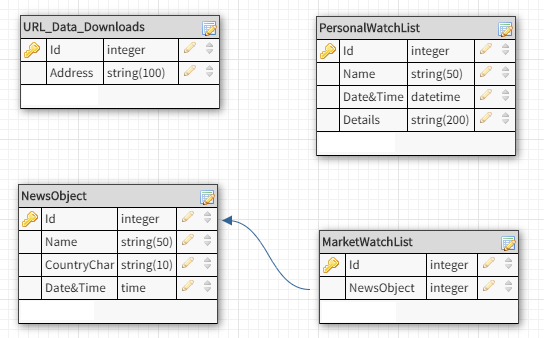
# Appendices

## Appendix 1 – Design specification

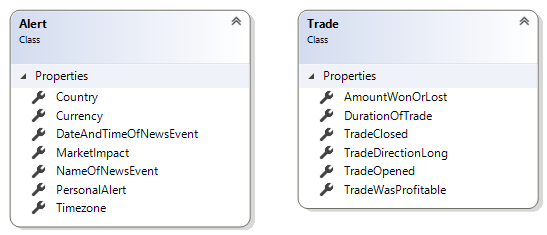
### Activities and Menus



### Schema for Currency Alert System Database



### Class Diagram



Nb. Trade class is only for use in future possible development.

## 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>