Report for Currency Trader news alert app

* 350 words per hour
* 6000 divided by 350 = 17 hours
* 4 hours per day = 4 days = 16 hours Wed / Thurs / Fri / Sat

Background:

Specification & Design:

* Proof of concept apps
* Localisation
* Planning – sprints proof of concept apps – separately & within the project
* naming conventions.
* 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. (uncle bob)
* XML to LINQ
* Database – how things are stored – ORM – Ignore Item (Ticks issues) - Ticks / DateTime object / Sqlite
* Workflow is more enjoyable when you have a definite plan
* that is laid out in a logical progression - step by step!
* Takes the 'creative worry' out of not knowing what you are going to work on next - helps to overcome procrastination!
* Date of xml update / download is now stored using Shared Preferences 17 / july
* Personal Alerts are left at whatever time the user set them for – they are NEVER altered.
* ~~Time zones & time differences~~

Implementation:

* LogCat Debug vs Toasts
* Passing data between activities – using intents – vs json – vs method arguments and parameters – vs setting a property on the receiving class
* Ahead of schedule !! Custom Adapter vs Recycler Adapter & Card view (notify adapter issue)
* Shared preferences for date of last data update
* AppCompat for themes / toolbars /

Testing & Results:

Conclusion & Future Work:

Learning experience - e.g. 2 different ways of implementing Toolbar using appcompat

- now have code available for use in future projects.

Preferences – added an option so the user can adjust all the market times plus or minus 59 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 – big job = FUTURE DEVELOPMENT!

**~~Development process:~~**

**~~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~~](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/~~](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’. These means that if application is run on devices of different sizes the appropriate sized graphic will be displayed, avoiding 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~~](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"~~

**~~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()~~

~~which 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.~~

**~~Data Access:~~**

~~Currently every time any data is required, a call to the database is carried out. There is no central repository cached in memory. This is because the amount of data is very small, 50 to 60 items of market data per week, discarded weekly, and because SQLite is essentially reading and writing to a text file, performance has not been issue.~~

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

~~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.~~

**~~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.~~

~~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.~~

**~~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, 1~~~~st~~ ~~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 it caused other issues downstream because 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 considerable change to the architecture of the application to avoid race conditions etc.~~

**~~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).~~

~~A method that used I used in ConvertString\_s\_ToDateTimeObject:~~

~~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"));~~

**~~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 wifi 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.~~