# Implementation

This chapter will include details on the implementation stage of the project. It will discus the major sections of implementation, the tools I used to carry out the implementation and any problems I encountered whilst implementing the application.

## Overview

The implementation stage of the project went very well, a few errors did arise but these were only ever minor errors and were quickly resolved. Using an IDE helped keep syntax errors to a minimum, which helped decrease the development time. The initial designs that were created were followed and only 1 major change was made from the original designs. The majority of the time during this stage was spent implementing the data download.

## IDEs that were used

In this section I will discuss the two IDEs that were used whilst developing and the reasons why I switched from Eclipse to Android studio.

**Eclipse with Android developer tools**

When following the setup tutorials on the Android developer guide, the tutorial taught me how to install the Android developer tools (ADT) plugin for Eclipses. I had already used Eclipse for other academic assignments so were already familiar with the basic user interface.

The ADT plugin added the ability to create Android application’s to Eclipse. The plugin also provided a GUI interface for downloading Android SDK’s, including all documentation. The ADT plugin also allowed Eclipse to launch Emulators for a variety of devices.

As Eclipse is an IDE, when writing Android code with the ADT plugin installed code suggestions and syntax errors are shown in real time. Using an IDE greatly improved the speed at which I wrote code. As I was new to Android development the documentation suggestions provided whilst coding were extremely useful.

Eclipse is occasionally buggy when running on my computer, for example occasionally every line of code within the project will be highlighted as an error and I will therefore have to restart Eclipse for the error to disappear. Due to this error and a few other minor problems with Eclipse I searched the net for an alternative IDE half way through implementation.

**Android Studio**

Android Studio is an IDE created by the Android developers at Google. Android Studio, which is currently still in beta is based off the IntelliJ IDEA IDE. Although Android studio is still in beta, the application works flawlessly, for everything that I needed it for.

As Android Studio was built solely for Android development it does not contain a lot of the extra bloat added to Eclipse. This makes it easier to find what you want in the menus as they contain fewer options. Android studio also has more intelligent code suggestions than Eclipses, making development easier. Android studio build configurations are configured using Gradle, which meant that adding library’s and custom build configurations was easy within Android Studio.

After running into issues with Eclipse I found the early access preview of Android studio on Android developer website. Installing Android Studio was easy and the SDK’s and emulators I had downloaded within ADT were automatically transferred. Android Studio also provides an automatic migration manager to convert projects from Eclipse into Android Studio. I migrated the main application into Android studio, but never migrated the projects prototypes, as they were already complete at this stage in development.

## Use of Gradle

Grade is an automatic build configuration tool, similar to Apache’s Maven. Gradle allows you create custom build configurations for a variety of setups. Gradle also has support for handling application dependencies, meaning importing and managing external libraries is easier and more efficient than manual methods.

I setup two build configurations using Gradle. The first configuration was used for building the application so that it was ready to be debugged. This configuration automatically enabled the debugable option within the Android manifest configuration file and also disabled Android’s ProGaurd. Enabling the debugable variable allows the application to be ran in the Android Studio debugger and disabling ProGaurd prevented the built APK from being obfuscated and shrunk, which improved the build time.

The second build configuration that was setup was the configuration for building the release version of the application. This build setup disabled the debuggable options and enabled Android’s ProGaurd. Enabling ProGaurd prevents other developers from easily stealing your code as the generated code is obfuscated. Using ProGaurd also shrinks the file size and optimizes the final APK.

On both of the above-mentioned build configurations I used Gradle to automatically check for the latest version of Robospice and download it if needed, including all of the needed dependencies. I found this feature extremely useful as Robospice had a list needed dependencies, that’d have to have been managed manually.

## Test driven-development

I had originally planned to follow the test-driven development pattern taken from the eXetreme programming methodology throughout the project. As this was my first ever large project that I had used test-development I found using that it slowed down my development. As this project had a limit time scale, I decided to change the original plan of using test-driven development for the entire project to only using test driven development for critical classes such as the calculator and view drug activity. I then followed the waterfall model’s method of testing for the remaining classes, which is to test the application thoroughly once implementation was complete.

## Implementing the models

The first package that I implemented for the project was the models package. The model’s had been first implemented within the prototypes, but the models had been slightly updated during the design stage, so these changes were made.

Implementing the models first allowed me to have classes to contain the data used for the system, meaning that classes that use the models will be able to from the start.

## Authenticating the user

The next part of the system to be implemented was the login activity and the authentication class. Implementing the authentication would allow the user’s username and password to be stored for the downloading of data, so this was logically the next step to implement.

When starting this project the only method of authenticating a user was to request a piece of data from the provided API and checking if an error was thrown, therefore I used the drug indexes API URL as this was the smallest XML file to download. Although I used the smallest file possible whenever the login was successful the request would take several seconds as the drug index was downloaded. As the drug indexes were not used when logging in, this was a waste of data. To improve this I emailed the NHS representative and asked them to create an API URL.

The NHS implemented the newly requested API URL. This URL takes two parameters (the username and password of the user) and returns true or false if the credentials are correct. This new API URL greatly sped up the login process and improved efficiency.

## Downloading the data and populating the local database

The next logical step was to implement the classes for downloading the database. Implementing the database and populating it early on ensured that all classes that used the data could be implemented afterwards.

This is the section of the implementation stage that changed majorly from the initial design. In the original design I planned to carry out the task of downloading the data using AsyncTasks. As mentioned in the design chapter of this report, AsyncTasks were not appropriate for long running tasks as they’re attached to the activity. This meant that if the user minimised the application or changed the devices orientation the download would be cancelled. As the tasks of downloading all the data was not a short running task and that I wanted the user to be able to run the task in the background I could not use AsyncTasks.

As my original design would not work how I had expected I had to redesign the download classes. I then learnt about Robospice services, which would allow me to carry out the download in the background. Robospice services run in their own thread and therefore the download’s can be ran simultaneously through multi-threading.

Whilst using Robospice services I still encountered some problems. Robospice was not built for downloading data and storing it within the database, Robospice was built for long running HTTP requests such as downloading large images from the web. Due to the intended nature of the Robospice services the caching abilities of Robospice did not suit my application. This is because Robospice only caches the return value of the service, as my application adds the information to the database within the service, there is no return value. After researching into the best practices I learnt that Robospice has a class made specifically for tasks that are un-cacheable services, thus this class was extended in all services the applications services.

Another issue I encountered when implementing the download service was determining when all the services had finished downloading. If the user had the application open whilst the download was in progress the download task worked as expected as the on success and on failure methods of the activity, but if the user minimised the application and a service completed the task, then when the service attempted to call the on success or on failure method nothing would occur as the activity would not exist at that point in time.

To solve this issue, extra methods had to be added to the DataProgress singleton, these extra methods keep track of the amount of started services and the number of completed services. To keep a track of the number of completed service I had to read into the Robospice service source code and plug into the method that notifies the activity when a service is complete, I then override this method and increased the finished count within the DataProgress singleton. When the number of started services is equal to the number of completed services then the all services have finished. The applications then checks that all the required API URLS have been downloaded, if they haven’t the user is notified of the failure and given the option to retry to download the parts that failed.

The finished implementation of the download activity and service works excellently, both in the background and in the foreground. The user is notified of any errors, even if the errors occur in the background. If errors occur whilst in the background, when the user reopens the applications they are presented with the retry option. Whilst the download service is in progress a notification is placed into the notification centre, which allows the user to know the download is still occurring.