

Progressive Overload Mobile Application

Mobile Application For Weight Lifting

4/23/2015

Oxford Brookes University

Lee Hudson 09092543

Acknowledgements:

David Lightfoot - Project Supervisor
Chris Cox - Ached emic Advisor

Abstract:

Technology can be used to solve many problems in day to day life. The aim of this project is to apply mobile technologies to solve the problems inherent in a weight lifting method known as progressive overload. In creating this application particular attention to making the app work well on both tablet and mobile phone will be paid. With the added screen size of a tablet more information can be displayed at any one time.

Progressive overload is a weight lifting technique that is commonly used to improve performance and weight gain. The principle of progressive overload is simple, you must strive to increase the load placed on your body in order to stimulate muscle growth. In practice this means having a set workout and each time that workout is performed the previous performance acquired for each exercise must be improved. Improvements can be increasing the weight used, increasing the amount of reps per exercise or decreasing the rest period between reps. The aforementioned increases are quantative and easy to represent numerically but technique is also important; this will be hard to record.

The above technique of progressive overload can be hard to implement effectively in a gym environment, it requires a lot of note taking which can be time consuming in itself. It can sometimes be hard to remember where you left off and it can be hard to determine if you really are improving. This is where the advantages of mobile computing can assist.

This project led to the successful creation of such an application and details how android deals with databases and multi pane GUIs called "Fragments".

Contents

INTRODUCTION:	3
ETHICAL AND LEGAL CONSIDERATIONS:	3
AIM:	4
ORIGINALITY:	5
CHOOSING A PLATFORM:	6
RESEARCH:	7
METHODOLOGY:	9
DEVELOPMENT AND RESULTS:	12
DESIGN:	12
Database Design:	13
Table Normalisation:	14
Final Entities:	16
RI Diagram:	17
EAR Diagram:	17
Gui Design:	18
Android Fragments:	18
Gui design and function:	21
Title Activity:	21
Edit Workouts:	22
Edit Exercises:	23
Select Workout:	25
Do Workout:	26
View Statistics:	27
Class structure design:	28
IMPLEMENTATION:	30
Using Eclipse ADT:	30
Android logcat:	30
Activity Intercommunication:	31
Explicit intent:	31
Implicit intent:	31
Creating, accessing and displaying the database:	33
Creating the database:	33
Accessing and displaying the database:	34
Displaying data in a ListView:	34
Displaying a single piece of data:	35
Dialogs:	37
Showing a dialog:	38
Image Acquisition and Conditioning:	40
Custom Graph View:	41
Problems Encountered:	42
TESTING:	43
Functional Test:	43
Real World Test:	43
FUTURE WORK:	44
Work to address issues found in real world test:	44
Other future work:	44
CONCLUSION AND EVALUATION:	45
REFERENCES AND BIBLIOGRAPHY:	52
APPENDICES:	53

Introduction:

Technology can be used to solve many problems in day to day life. The aim of this project is to apply mobile technologies to solve the problems inherent in a weight lifting method known as progressive overload.

Progressive overload is a weight lifting technique that is commonly used to improve performance and weight gain. The principle of progressive overload is simple, you must strive to increase the load placed on your body in order to stimulate muscle growth. In practice this means having a set workout and each time that workout is performed the previous performance acquired for each exercise must be improved. Improvements can be increasing the weight used, increasing the amount of reps per exercise or decreasing the rest period between reps. The aforementioned increases are quantitative and easy to represent numerically but technique is also important; this will be hard to record.

The above technique of progressive overload can be hard to implement effectively in a gym environment, it requires a lot of note taking which can be time consuming in itself. It can sometimes be hard to remember where you left off and it can be hard to determine if you really are improving. This is where the advantages of mobile computing can assist.

The above activity is essentially record keeping and data analysis. A mobile application can drastically improve the practicality of progressive overload.

Ethical and legal considerations:

It is important to consider any ethical and legal implications when undertaking this project:

Ethical:

1. Testing the application on human subjects
 - a. During the testing phase of the project the application was given to 3 members of the public to try the application in the real world. It is important that the privacy of these members of the public is maintained. As such these test subjects will remain anonymous.
 - b. It is important to ensure the application testing on human subjects has no adverse effects on their mental health. As this application is just recording data this point has been considered and disregarded.
2. Data protection
 - a. As stated above, the test subject will remain anonymous.
 - b. The application will store data about the users' performance. Whilst this data is not in the public domain it does remain on the device. I don't think this is a real consideration as the data store is not sensitive.

Legal:

1. Use of copyright material
 - a. Some images obtained from the internet were used in the making of this application. To avoid any copyright infringement this application will not be published and will not be available to the public. If more time was allowed the images would have been either purchased or created.

Aim:

The aim of this project is to create a mobile application to address the issues of progressive overload. The finished application will allow users to enter in their performance for a given work out and to compare their performance as a function of time. It will tell the user what benchmark they reached on their previous workout so they can better it. The exact requirements are detailed in the requirements specification.

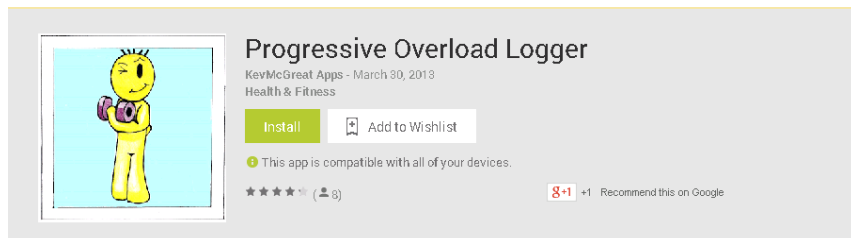
In terms of implementing the above on a mobile device there are the following key issues to be considered:

1. What platform should this be developed on?
 - a. The main choices are Android, IOS and windows mobile.
2. Battery usage
 - a. This is not normally a consideration when programming for a static device but for a mobile device this needs to be considered. Sometimes good programming practices have to be re-considered in favour of efficiency.
3. Portability
 - a. If this application is developed on IOS or android it must be able to be used on a variety of screen sizes and resolutions. Some applications on the market do not consider this and it is evident when you run the application. Android and IOS have the facility to display certain GUIs depending on the size of the screen. In Android this is called a fragment. The application will appear different from device to device.
4. Non-volatile data storage
 - a. The data entered into the application must be retained on power off. The data needs to either be written to a file or stored in a database. Both android and IOS have SQL interfaces, a database such as SQL would be favoured over a text file as data consistency is important. When implementing a database it is important to ensure the database has been normalised to at least Boyce-Codd normal form.
5. Gui orientation
 - a. Most mobile devices allow both portrait and landscape views, this has to be considered.
6. Graphical representation of performance
 - a. It would be useful to represent the users' performance in the form of a graph, this will most likely need to be programmed from scratch and will have features such as auto scaling.
7. Innovative
 - a. The application must be easy to use and surprise the user with features such as customisation and nice graphics.

Originality:

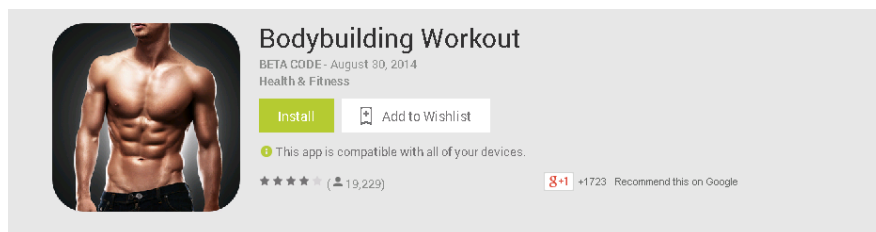
Currently there are a few applications available for workouts but none that address the issues of progressive overload effectively. The following applications are available for android:

Progressive overload logger - KevMcGreat Apps



This application addresses the needs of progressive overload to a certain extent by allowing the user to log their performance, however, it is hard to use. It requires too many user interactions for a single operation which is unsuitable for use in a gym.

Bodybuilding workout - Beta Code



This application does nearly everything needed for progressive overload but doesn't do it in a robust way. This app is full of bugs and is not intuitive. It also doesn't allow you to view your progress in a graphical way.

The above applications are the only ones to be found that directly target progressive overload. Neither of these applications fully address the needs of a progressive overload so the market would benefit from an application that does specifically target progressive overload.

The project outlined above is a suitable project for a software engineering degree as it contains many aspects of software engineering. It will be very code intensive, make use of a database and graphical user interfaces. Good software engineering practices will be observed where possible.

Choosing a platform:

Before the project can be fully planned and designed it must be decided which platform to develop for. There are only really 3 feasible mobile operating systems:

1. Apple IOS
2. Android
3. Windows Phone

In order to decide which platform to develop for one must consider the following qualities required:

1. Popularity of the operating system
 - a. The more popular it is amongst its users the more exposure the application will get.
2. Ease of developing
 - a. The easier it is to code the application the less the application costs to produce.
3. Ease of publication
 - a. How easy does the platform make it to get the application on the market?
4. Longevity of the operating system
 - a. Is it likely that the platform will become obsolete in the near future?

With the above in mind the most obvious choice is Android. The Android platform is by far the most popular platform having 84.4% of market share in the 3rd quarter of 2014, followed by iOS having 11.7% and then Windows Phone having 2.9% (www.idc.com). From a popularity point of view this is the obvious choice. From now on Windows Phone won't be mentioned as its market share rules it out completely.

Android is also probably the easiest to develop for, it is also free to develop for. Android applications can be programmed in java which given my personal experience is predominantly in java this would be preferable. Apple iOS apps are developed in objective C which is a language I have no experience of.

There are less stringent requirements for publishing an application to the android market than the iOS app store. You may find that if you develop an app for the iOS that apple decides it doesn't want to publish it, this would be a huge waste of resource.

Both Android and iOS are likely to be around for a long time yet however due to the market share that android has it would be logical to assume Android probably has a greater life span.

Taking into account the points made above it has been decided that the platform this application will be developed on is android. The reason for this is due to the extra costs associated with developing for apple devices. Although the one of the most popular app categories on the apple market is fitness (Rahul Varshneya,2013) the ease of developing and publishing on the android market makes android the best choice for this application. Also the diverse hardware associated with android adds an interesting challenge. The variety of screen sizes and resolutions will pose some technical challenges.

Research:

There are two main subjects to research, the technical aspect and the topic area. From a technical point of view some research is needed into mobile applications and programming. From a topical point of view research is needed into the method of progressive overload. For this literature search the IEEE library was used along with a generic internet search engine.

Article 1: **How to choose the best platform for your app**

Sherman,E.(2012) *How to choose the best platform for your app*. [Online] 08.10.12. Available from - <http://www.inc.com/erik-sherman/avoiding-the-single-mobile-platform-trap.html> [Accessed: 03.10.14]

The above article is a personal blog posted at the referenced URL. The article details things that must be considered when choosing a platform such as the possible risks associated with developing with apple, specifically the app store and the control apple asserts over it. It also looks at the risks with android, specifically the challenge of developing for many devices at once. It doesn't appear to have a bias to any particular platform and does highlight things that must be considered. The article is very brief but it does make the reader aware of some interesting considerations. It touches on the technical limitations of platforms and limitations from an infrastructure point of view, specifically the apple market place.

The author Erik Sherman is a journalist and also has experience as a communications consultant. He is also the author or co-author of 10 books, some of which are technical.

This article is still relevant as it's only 2 years old. Technology moves fast but these key issues are still present so this article is still applicable. This article will be useful when it comes to deciding which platform to develop on.

Article 2: **The ten rules of progressive overload**

Bret(2013) *The ten rules of progressive overload*. [Online] 26.02.13. Available from - <http://bretcontreras.com/progressive-overload> [Accessed: 03.10.14]

This article is also a blog but it is focused on the subject material rather than the technical aspect of an application. It does a good job of explaining the principle of progressive overload but as with allot of weight training articles it is subjective. It uses examples of how well the technique works and is useful in that it underlines all the basic principles of progressive overload. This article will be very useful when designing this application.

The Author is just a personal blogger who has a passionate interest into the subject. There are so many opinions on this subject it is hard to verify the credibility of this article or the author but it must be said that this article is methodical and logical.

This article is a year old so it is definitely relevant. The subject of weight training has been around for a very long time with very few developments in technique so this will probably remain relevant for some time to come. This article will be useful to ensure the mobile application meets the requirements of progressive overload and to make sure the concept of progressive overload is fully understood.

In addition to the above articles many other articles have been seen to be relevant and so have been used. These have been detailed in the references.

Article 3: Multi-pane development in android with fragments - Tutorial

Vogel,L.(2014) *Multi-pane development in android with fragments - Tutorial* [Online] 29.04.14. Available from - <http://www.vogella.com/tutorials/AndroidFragments/article.html> [Accessed: 03.10.14]

The above article is a tutorial detailing showing how to develop android applications for a variety of screen sizes and in particular catering for both tablets and phones by displaying applications differently depending on both screen size and orientation. This tutorial has a specific example which is extremely applicable to the progressive overload application. In the RSSFeed example it shows the class structure and interactions required to display multiple GUIs on the screen at once for larger landscape orientated displays and single GUIs for smaller or portrait displays.

The author Lars Vogel is a prolific tutorial write for a variety of computer science based tutorials and is the founder of the professional training company Vogella. This company not only write tutorials online but assist in training other companies in many areas of computing. From personal experience the tutorials online are fairly clear and easy to follow, they also seem to work.

This article is kept up to date with the last update on 13/11/2014. Therefore this article can be considered current and up to date with the current state of technology, in particular android.

Article 4: Fragments

Unknown(unknown) *Fragments*. [Online] 20.11.14. Available from- <http://developer.android.com/guide/components/fragments.html> [Accessed: 20.11.14]

This article is a tutorial / explanation of how to use android fragments. As with article 1 it details the class structure required to create flexible user interfaces in android using fragments. This article comes across as more of a technical reference than a step by step tutorial but is still extremely helpful in assisting ones understanding of how fragments work.

This article is on the android developers' website. Whilst the actual author is not mentioned it can be assumed that if this article is on the Android website that it will be correct.

As this article is published on the Android website it will probably be up to date. This website is the main point of reference for android developers.

In addition to the above articles many other articles have been seen to be relevant and so have been used. These have been detailed in the references.

Methodology:

The software development method used for this project was the waterfall method. It was very hard to truly follow any development method when the work was undertaken by a single person but the waterfall method seemed the most applicable. Using the waterfall method the project was split up into different progressive phases to be completed in order.

Using the waterfall method did however pose some problems. This method would be fine for developing an application or piece of software if the developer fully understands how to implement it. Clearly this is not the case for an educational project as learning is still part of the process. Initially the waterfall method was used but once the implementation phase began it became apparent that there were better ways of doing things in practice, this involved going back and tweaking the design during the implementation stage.

With the above in mind the following stages of the waterfall method were specified but not used as rigidly as originally intended:

1. Requirements:
 - a. Complete the requirement specification
2. Design
 - a. Decide on the classes to be used
 - b. Decide what these classes will do and how they will interact
 - c. Compose a UML diagram of the system
 - d. Compose and normalise the database
 - e. Design the user interface
3. Implementation
 - a. Create android activities
 - b. Implement GUIs for each activity
 - c. Implement communication between activities
 - d. Implement database
 - e. Implement custom graph view
 - f. Implement timer option
 - g. Implement any audio
4. Verification
 - a. Create a test schedule
 - b. Test the mechanics of the system
 - c. Perform beta testing on application by deploying to a few test subjects

This application will be developed using Eclipse with the android plugin. I am familiar with both Eclipse and the Android plugin. For testing purposes both a large screen and a small screen device will be needed. These are available and will be a Samsung 10.1 tablet and a Nexus 5 phone. Basic testing has proven that I can develop with this software on both devices.

For debugging the eclipse package has a useful logging tool called “Logcat” which allows live readable outputs whilst the android device is connected. This is much like printing to the terminal in other languages.

The main language used to develop android is Dalvik; this is basically java with some additional API. Therefore I am already familiar with this language and this doesn’t really pose any additional barriers for me. So far the hardware and software used hasn’t posed any limitations.

After playing out different use cases the below requirement specification has been finalised:

Functional requirements:

1. Allow the user to create a workout.
2. Allow the user to create an exercise.
3. Allow the user to add exercises to a work out.
4. Allow the user to remove a workout.
5. Allow the user to remove an exercise.
6. Allow the user to record the performance each time an exercise is performed.
7. Allow the user to see the performance from their last workout to allow them to clearly see what they have to better.
8. Give the user an audible alarm to signify the end of a rest period.
9. Allow the user to see how their performance has increased in a graphical way.

Non-functional requirements:

1. The application should respond to an input within 500ms.
2. The application must retain data on power off.
3. The data stored by the application must be consistent.
4. The application must work in both landscape and portrait orientations.
5. The applications must work on screen sizes from 640x480 to 1920x1080.
6. The application should be locked in portrait for screens less than 6" and locked in landscape for screens greater than 6".

Using the above functional requirements it was possible to design the GUIs and verify that once the functional code has been implemented all the functional requirements will be met.

Looking at the non-functional requirements I can now comment on how these will be met:

Non-functional requirements:

1. The application should respond to an input within 500ms.
 - a. Any time consuming functions such as the rest timer or audio playing will be done on a separate thread.
2. The application must retain data on power off.
 - a. The data will be stored in an SQL database, in android this is implemented using what is called a content provider.
3. The data stored by the application must be consistent.
 - a. The database will be normalised to BCNF.
4. The application must work in both landscape and portrait orientations.
 - a. This is catered for in the android framework. Separate GUI layout files are used for portrait and landscape.
 - b. Android fragments improve on this further by allowing multiple user interfaces to be displayed at one time.
5. The applications must work on screen sizes from 640x480 to 1920x1080.
 - a. The screen size is a variable that can be queried directly in the code and so can be catered for.

With the above project there are inherent risks:

1. Project taking too long to due lack unforeseen complexity.
 - a. The GUI programming was initially seen as a possible source of unforeseen complexity. As further work has been done and GUIs are now understood well this is only a small risk.
 - b. Implementing the content provider on an android application is always quite complex, the addition of having multiple SQL tables adds to this complexity. If this proves too complex it may be easier to use multiple content providers, one for each table.
2. Some aspects not possible on android.
 - a. This risk has been mitigated as all areas have been researched to the extent that I now know all functional and non-functional requirements can be met using android.

Development and results:

Design:

Following the waterfall method the first stage of the software development process is the design stage. The application was designed in the following order:

1. Database
 - a. It is known that the application will required to store data in a non-volatile way. In order to do this an SQL database will need to be designed. In order to design an efficient database it will be normalised to BCNF. The database will be the core of the application.
2. GUIs
 - a. In order to realise how the application will be used it would be useful to visualise how the user will interact with it. In android each GUI usually has a class associated with it; once the GUIs have been designed a rough class list will be derived.
 - b. As the application needs to work well on a variety of screen sizes; this is where the visual distinction will be defined between tablets and phones.
3. Class structure
 - a. Once the GUIs have been designed the mechanism for the overall functionality and intelligence must be designed. This will encompass the interaction between the android activities and classes and also the mechanism of how the application changes behaviour depending on the screen size.

Database Design:

In order to maintain the application data after exit a relational database must be embedded in the application. Android allows the creation and use of an SQL database through what is called a content provider. As with all relational databases the database entities must be normalized to maintain data consistency. The following database entities have been deduced from normalisation:

1. Workout
2. Exercise
3. Exercise Record
4. Scheduler

The application must be able to access the following data from the database:

1. Workout
 - a) Day
 - b) Muscle Group
 - c) Exercises in that workout
2. Exercise
 - a) Name
 - b) Rest period
 - c) Photo URI
3. Exercise Instance
 - a) Exercise name
 - b) Date
 - c) Result

The application must be able to perform the following functions on the database:

1. Add / Remove a workout.
2. Add / Remove an exercise.
3. Record the results of an exercise / workout.
4. Access all of the above data.

Table Normalisation:

From the initial description the following structure with the following functional dependencies was deduced:

Entity 1:

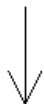
Workout ID	Workout Day	Muscle Group	Exercise ID	Exercise Name	Rest Period	Photo URI	Date	Result
┌	┌	┌	┌	┌	┌	┌	┌	┌
└	└	└	└	└	└	└	└	└

Entity 1. From the workout ID, exercise ID and the date it is possible to determine all other entries.

It can be seen above that Entity 1 is in need of normalisation as it has multiple functional dependencies.

Workout ID	Workout Day	Muscle Group	Exercise ID	Exercise Name	Rest Period	Photo URI	Date	Result
┌	┌	┌	┌	┌	┌	┌	┌	┌
└	└	└	└	└	└	└	└	└

┌	┌	┌	┌	┌	┌	┌	┌	┌
└	└	└	└	└	└	└	└	└



Workout ID	Workout Day	Muscle Group
┌	┌	┌
└	└	└

From this initial normalisation we now have an entity in BCNF. This will be the “Workout” entity.

The remaining table still needs further normalisation:

Workout ID	Exercise ID	Exercise Name	Rest Period	Photo URI	Date	Result
┌	┌	┌	┌	┌	┌	┌
└	└	└	└	└	└	└

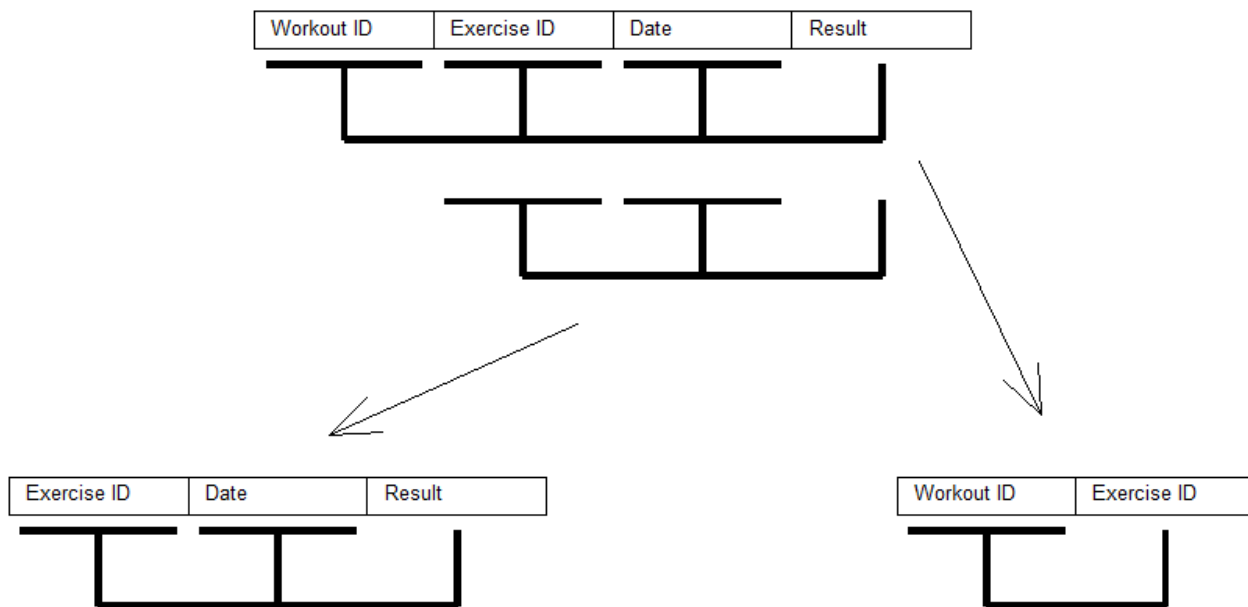
┌	┌	┌	┌	┌	┌	┌
└	└	└	└	└	└	└



Exercise ID	Exercise Name	Rest Period	Photo URI
┌	┌	┌	┌
└	└	└	└

We now have another entity normalised to BCNF, this will be called the “Exercise” entity.

The remaining table still needs further normalisation:



With the above process of normalisation we can now describe all entities in the database:

Workout

Workout ID	Workout Day	Muscle Group

Exercise

Exercise ID	Exercise Name	Rest Period	Photo URI

Exercise Record

Exercise ID	Date	Result

Scheduler

Workout ID	Exercise ID

With the above entities we are able to avoid all update, insert and delete anomalies. All entities are in BCNF. The “Scheduler” entity will act as a linker between the “Workout” and “Exercise” entities and will have a many to one relationship with each to avoid them having a many to many relationship with each other (see the EAR diagram later).

Final Entities:

Attribute Name	Description	Data Type
<u>Workout ID</u>	Unique number to identify the workout	INT
Workout Day	Recommended day for the workout	String
Muscle Group	Muscle group for the workout	String

Entity: Exercise

Attribute Name	Description	Data Type
<u>Exercise ID</u>	Unique number to identify the exercise	INT
Exercise Name	Name of the exercise	String
Rest Period	Rest period between sets (seconds)	INT
Photo URI	URI of the associated photo	String

Entity: Exercise Record

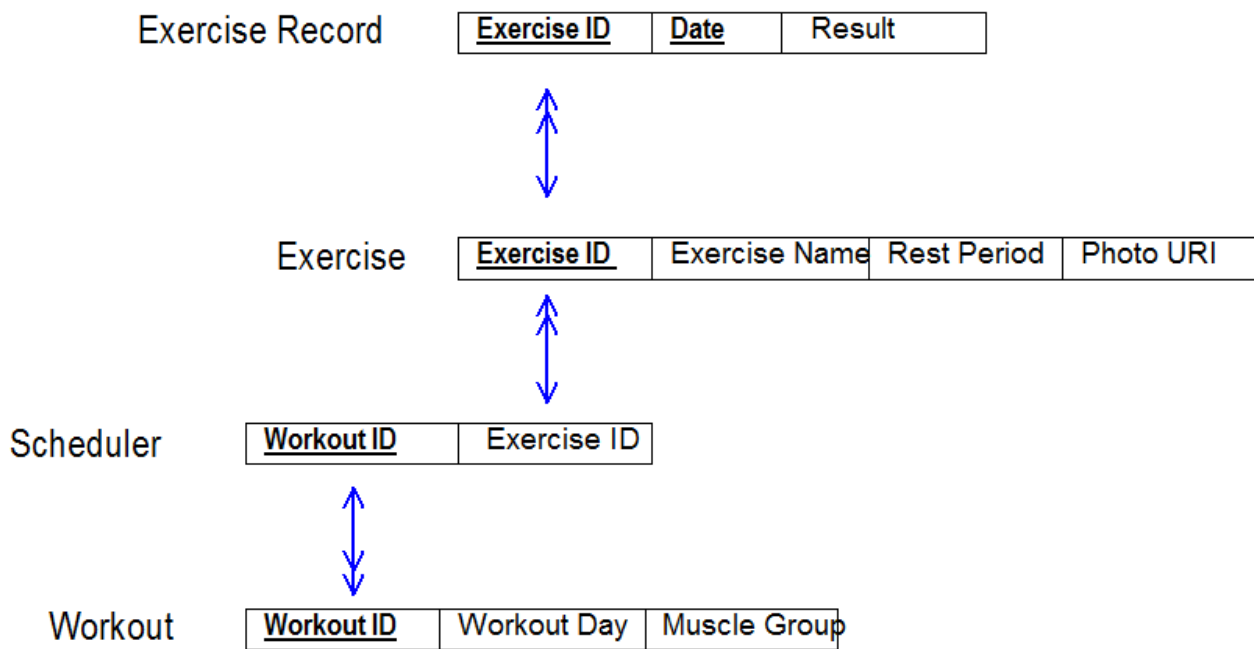
Attribute Name	Description	Data Type
<u>Exercise ID</u>	Unique number to identify the exercise	INT
<u>Date</u>	Date the exercise was done	String
Result	Actual performance	??

Entity: Scheduler

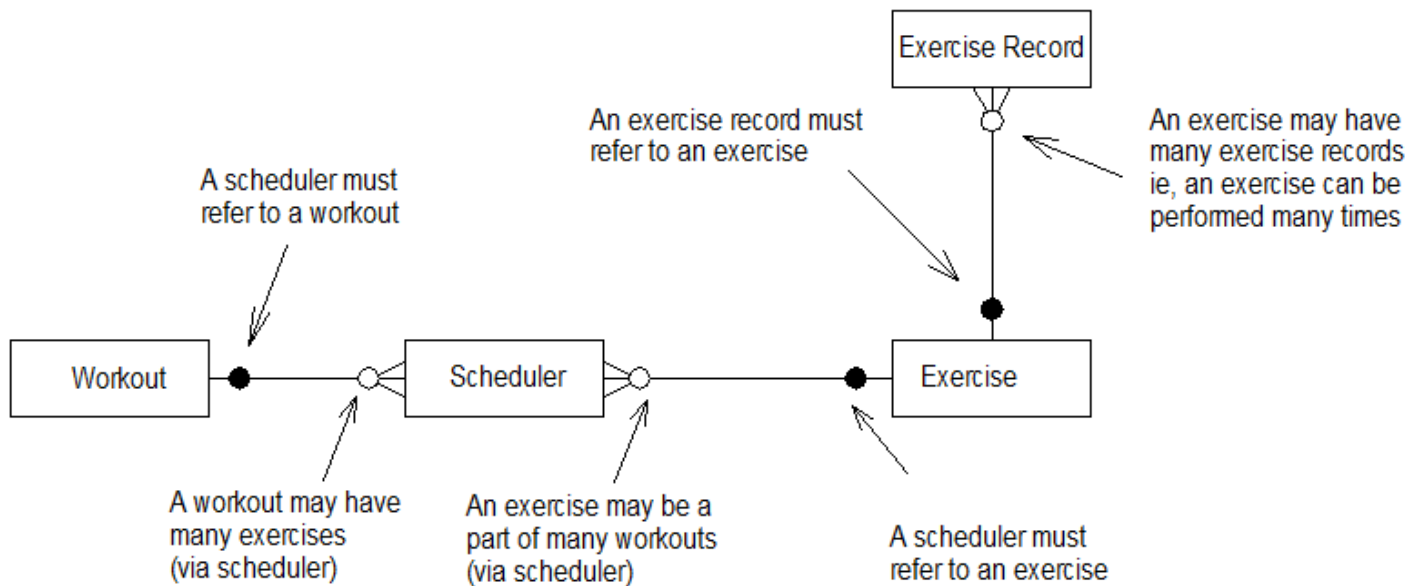
Attribute Name	Description	Data Type
<u>Workout ID</u>	Unique number to identify the workout	INT
Exercise ID	Unique number to identify the exercise	String

The above entities are a preliminary specification of the database. There are still a few unknown factors as the operation of the android content provider is still not fully understood.

RI Diagram:

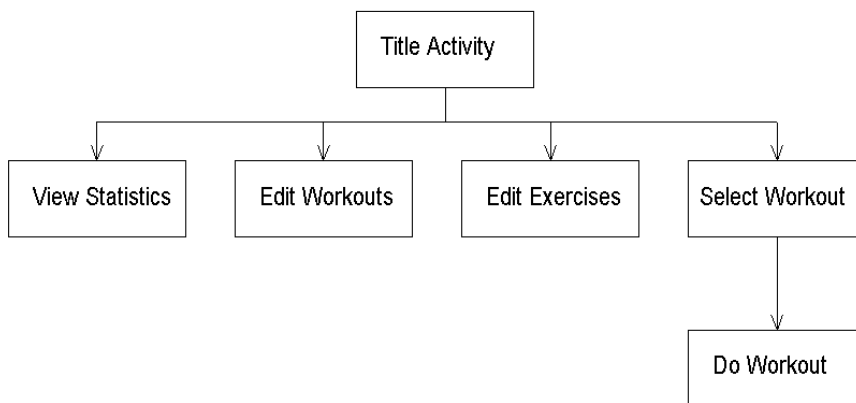


EAR Diagram:



Gui Design:

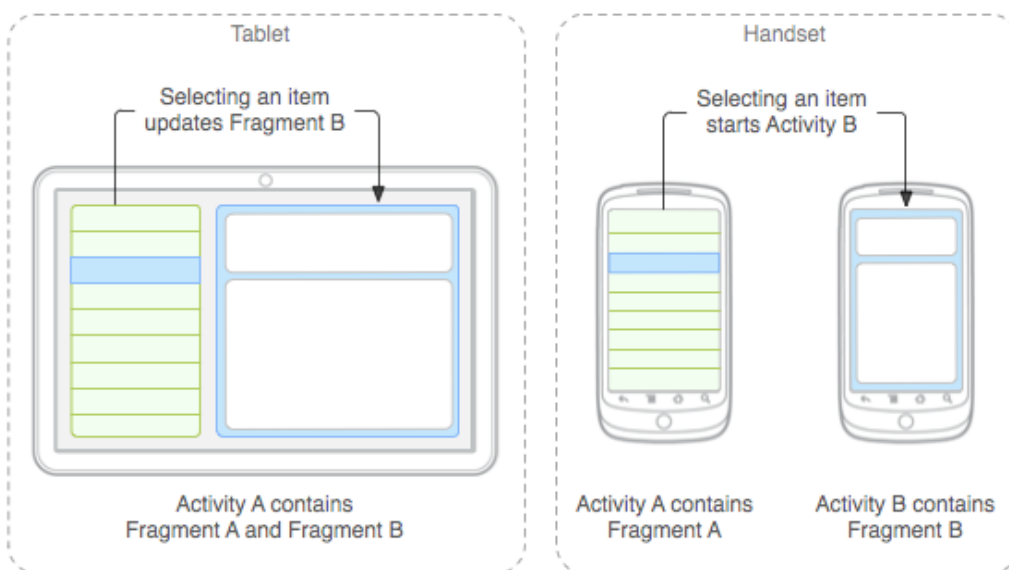
Using a CAD package the GUIs can be designed as a starting point. This will help to clarify the behaviour of the application. As a starting point we can define the base activities and the flow between the:



Android Fragments:

With the above activities defined we can design how the user interface will appear. We will need to design how the interface will look in both landscape and portrait.

In order to make the application suitable for both phones and tablets we will use a component in android called a “fragment”. When the application is being run on a tablet in landscape mode there is much more display real estate that can be taken advantage of. Using fragments essentially allows us to display more than one GUI at a time:



(www.vogella.com)

In the above example we can see how the larger display on tablets can be used to great effect. The mechanism for invoking fragments depending on screen orientations is quite simple; GUI layouts are store in XML files that are located in the “layout” folder. Rather than specifying GUI components in the XML “fragment” elements are specified. By default the XML file in the “layout” folder will be used, however if a “layout-port” folder is specified the Android OS will get the GUI XML file from this folder if the device is in portrait orientation. For example:

layout/activity_edit_workouts.xml

```
<LinearLayout
    android:orientation="horizontal" >

    <fragment
        android:id="@+id/editWorkoutsListFrag"
        android:name="com.example.progressiveoverload.EditWorkouts.EditWorkoutsListFragment"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="fill_parent">
    </fragment>
    <fragment
        android:id="@+id/editWorkoutsDetailFrag"
        android:name="com.example.progressiveoverload.EditWorkouts.EditWorkoutsDetailFragment"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="fill_parent">
    </fragment>
</LinearLayout>
```

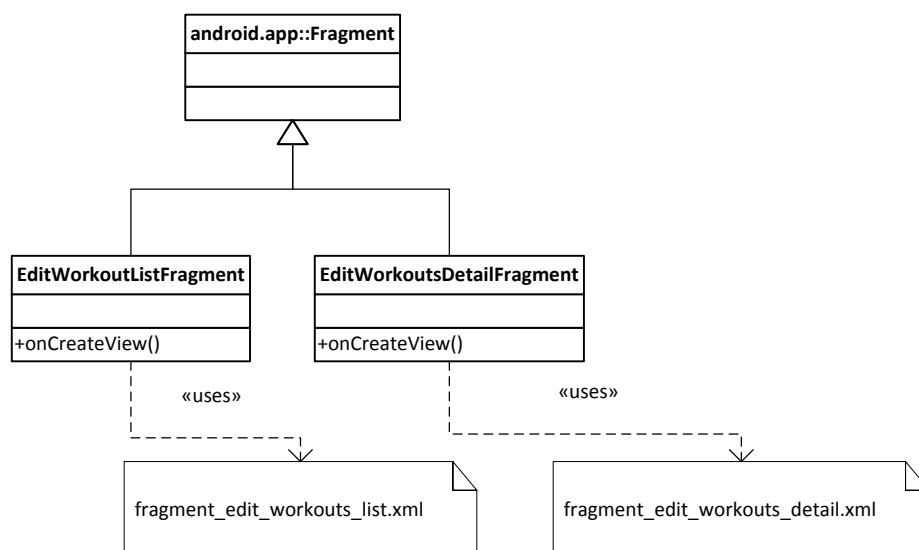
layout-port/activity_edit_workouts.xml

```
<LinearLayout
    android:orientation="vertical" >

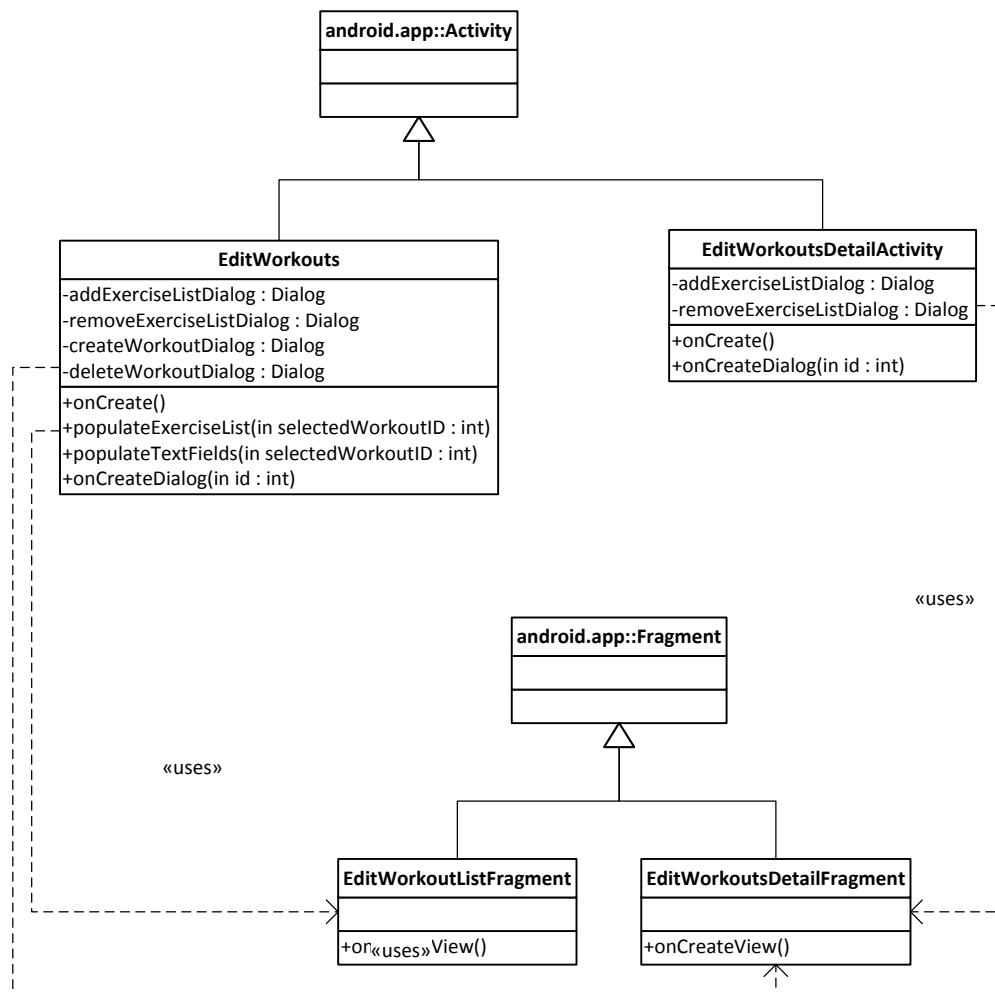
    <fragment
        android:id="@+id/editWorkoutsListFrag"
        android:name="com.example.progressiveoverload.EditWorkouts.EditWorkoutsListFragment"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="fill_parent">
    </fragment>
</LinearLayout>
```

Here we can see that if the device is in portrait only the “editWorkoutsListFrag” will be used whereas both the “editWorkoutsListFrag” and “editWorkoutsDetailFrag” will be used if the device is in landscape.

These fragment elements in the XML file refer to a fragment class that extends Fragment. These fragments classes then inflate another XML for the GUI:



Whilst in landscape mode a single activity takes care of both fragments, however when the device is in portrait there needs to be an activity for each GUI. When an action causes the second GUI to be needed a second activity is started:



With this in mind it is apparent that both the main activity (**EditWorkouts**) and the secondary portrait activity (**EditWorkoutsDetailActivity**) will both need to interact with the same fragment (**EditWorkoutsDetailFragment**). Any interaction that is duplicated will be encapsulated in a separate class.

Gui design and function:

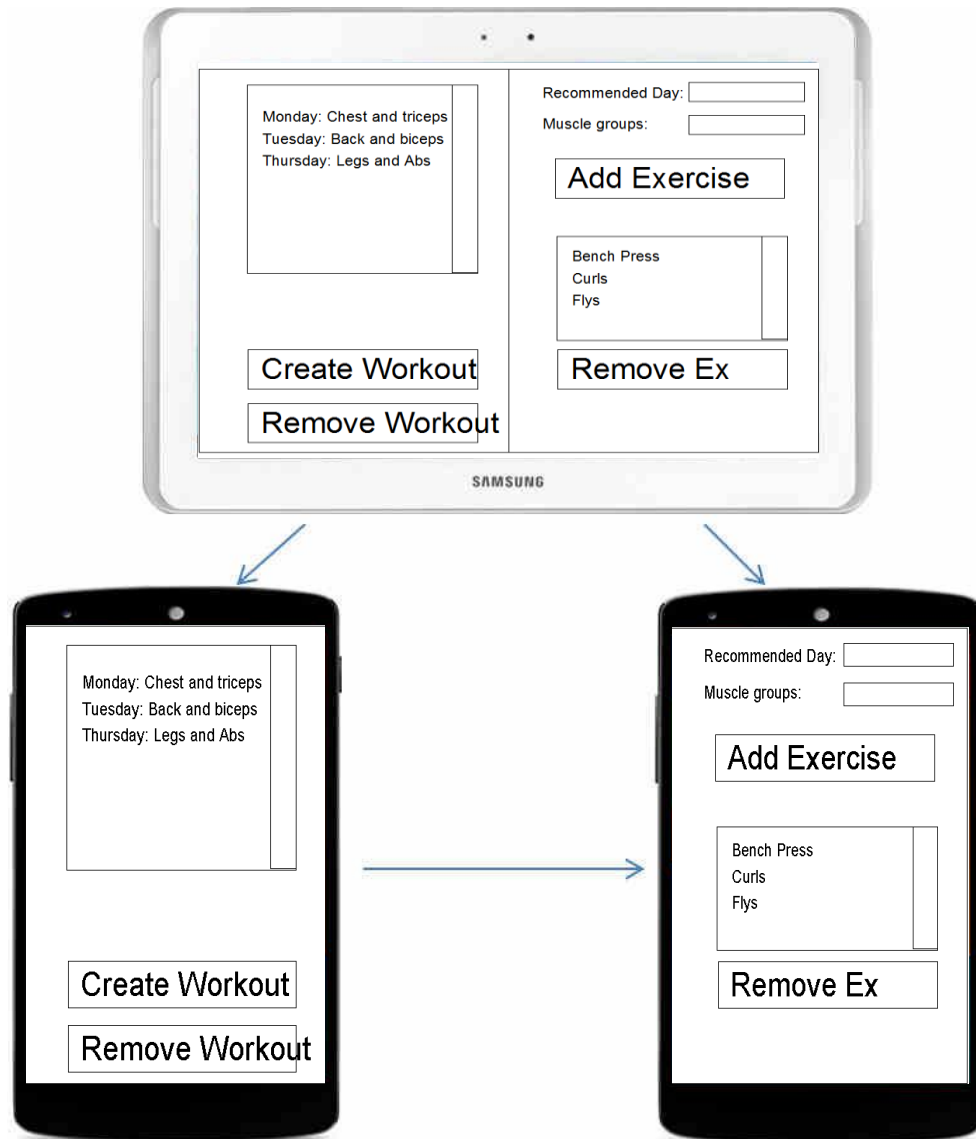
Title Activity:



The title activity will be the first activity the user will come into contact with. It has a fairly simple GUI that will allow the user to access the rest of the application. Due to its simplicity the GUI can remain the same for both portrait and landscape. Each button on this GUI just launches the described activity.

Edit Workouts:

EditWorkoutsListFragment | EditWorkoutsDetailFragment



The edit workouts activity will use 2 fragments. These are the “EditWorkoutsListFragment” and the “EditWorkoutsDetailFragment”. Depending on the device used they will be displayed as above.

The edit workouts activity allows the user to do the following:

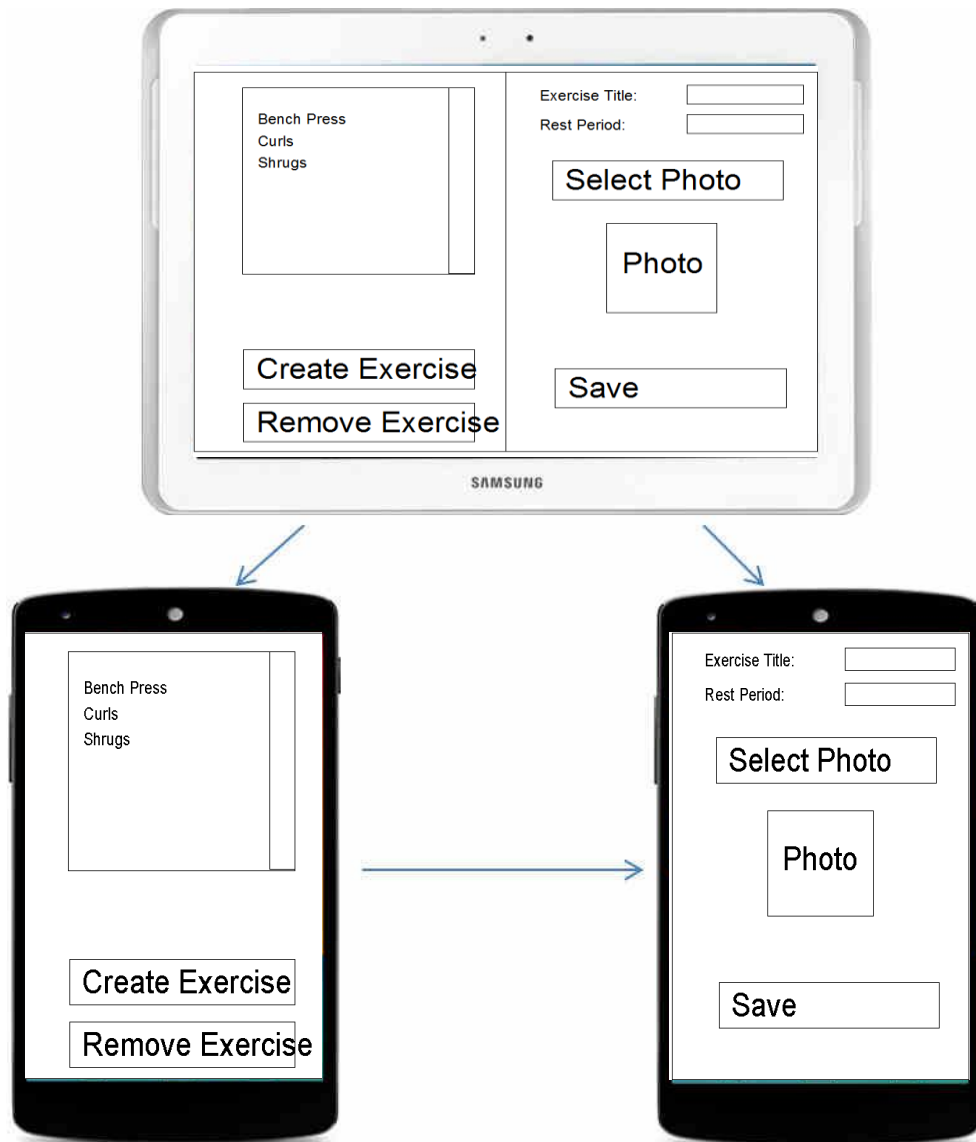
1. View all workouts
 - a. Workouts will be displayed in a list view. The list view will display all workout entities in the database. This list view will also respond to clicks on any of its items to allow the workout to be edited.
2. Edit any workout
 - a. By clicking on a workout in the list view the user can edit the details of a workout. When the application is in landscape mode the fragment to the right side of the GUI will allow the user to edit the workout, if the application is in portrait mode another activity will be started for this and hence another GUI.
 - i. The edit workout detail fragment GUI will also contain a list view that displays all exercises assigned to the selected workout. Pressing the “Add Exercise” button will bring up a list dialog of all exercises, clicking on any of these exercises will add that exercise to the

workout. Pressing the “remove exercise” will bring up a list dialog of all exercises assigned to the selected workout, clicking on one will remove it from the workout.

3. Remove any workout
 - a. By pressing the “Remove workout button” the user can remove a workout. A list dialog with all workouts is displayed and once a workout is clicked on it is removed.
4. Create workout
 - a. Pressing the “Create Workout” button will invoke a dialog to aid creating a new exercise.

Edit Exercises:

EditExercisesListFragment | EditExerciseDetailFragment



The edit exercise activity will use 2 fragments. These are the “EditExerciseListFragment” and the “EditExerciseDetailFragment”. Depending on the device used they will be displayed as above.

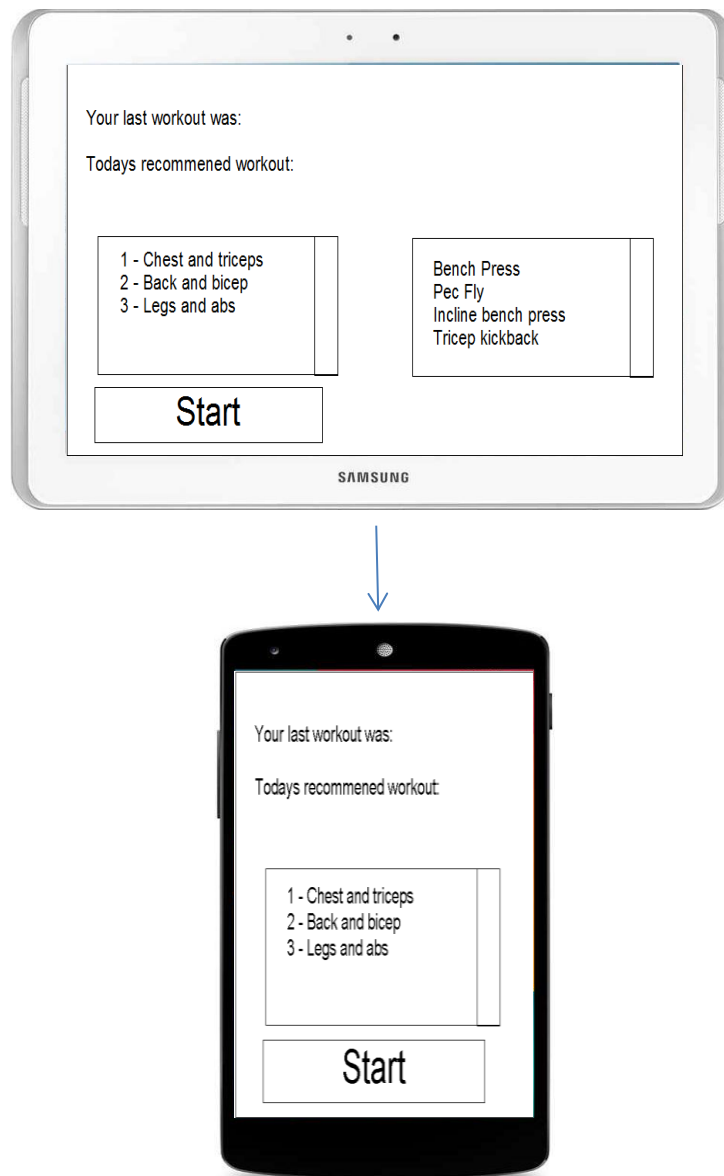
The edit exercise activity allows the user to do the following:

1. View all Exercises
 - a. Exercises will be displayed in a list view. The list view will display all exercise entities in the database. This list view will also respond to clicks on any of its items to allow the exercise to be edited.

2. Edit any exercise
 - a. By clicking on an exercise in the list view the user can edit the details of an exercise. When the application is in landscape mode the fragment to the right side of the GUI will allow the user to edit the exercise, if the application is in portrait mode another activity will be started for this and hence another GUI.
 - i. The edit exercise detail fragment GUI contains the details of the selected exercise. This includes the name of the exercise, the rest period between sets and a photo / image illustrating that exercise. The user can edit any of these details and click the save button to update that exercise. Clicking on the "Select Photo" button will allow the user to select an image from the inherent android gallery.
3. Create an exercise
 - a. By clicking on the "Create Exercise" button the user can create a new exercise. If the device is in landscape mode this will simply clear the details fragment. If however the device is in portrait mode the details GUI will be invoked and with all the fields as blank. Pressing the save will create a new exercise.
4. Remove any exercise
 - a. By pressing the "Remove exercise button" the user can remove an exercise. A list dialog with all exercises is displayed and once an exercise is clicked on it is removed.

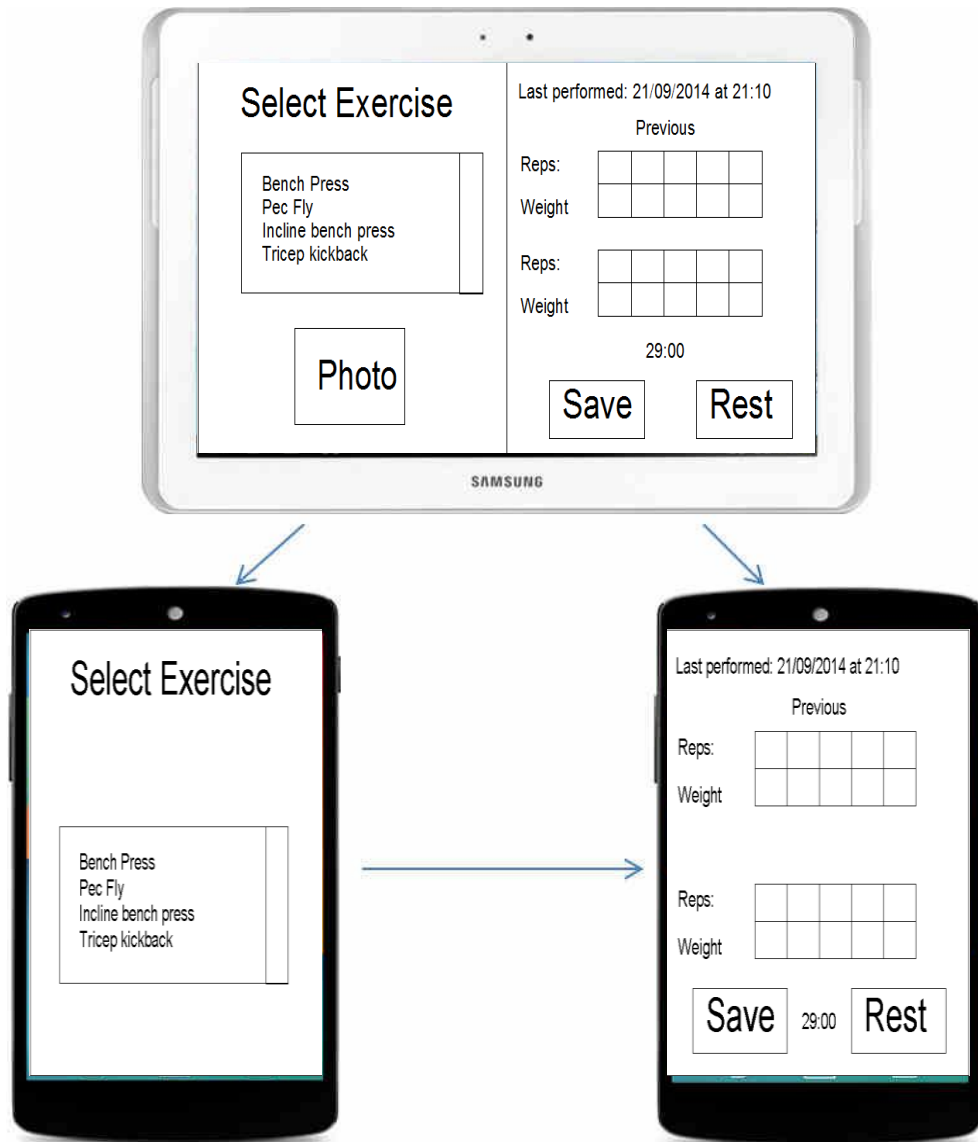
Select Workout:

SelectWorkoutListFragment | SelectWorkoutDetailFragment



The select workout activity will use two fragments but unlike previous activities it will only use one when in portrait mode. The tablet user will benefit from being able to view more information.

This activity will allow the user to select which workout to begin. Clicking on an item in the workout list view will select the clicked workout. In landscape mode the exercises associated with the selected workout will be given, in portrait mode they will not. Clicking on the "Start" button will then progress the user to the do workouts activity for the selected workout.



The do workout activity will use 2 fragments. These are the “DoWorkoutListFragment” and the “DoWorkoutDetailFragment”. Depending on the device used they will be displayed as above.

This purpose of this activity is to allow the user to record their performance for a given workout. It will allow the user to do the following:

1. Select an exercise
 - a. Clicking on an exercise in the list view will select an exercise. In landscape mode the attached photo of the exercise is displayed and the fields that detail the performance that was previously achieved will be populated with data. If the device is a portrait device clicking on an exercise will bring up the detail fragment where again, the details of the previous performance will be displayed. Note, in portrait mode there will be no image of the exercise displayed as clicking on an item will immediately bring up the next GUI.
2. View performance information about the last time the selected exercise was performed
 - a. When an exercise has been selected the detail fragment will display when the exercise was last performed and the performance last obtained. This will be in the form of a row of reps and a corresponding row of the weight for each set.
3. Allow the user to save their new performance

- a. Armed with the information of their previous performance to hand the user can now try to better their previous performance. Once they have performed the exercise they can select the reps and weight used for each set using an android Spinner. A spinner is a drop down menu with pre-determined values. Pressing the “Save” button will create a new entry in the Exercise Record database entity.
4. Allow the user to time their rest period
 - a. Clicking the “Rest” button will start a timer that will make a sound when the timer is complete.

View Statistics:

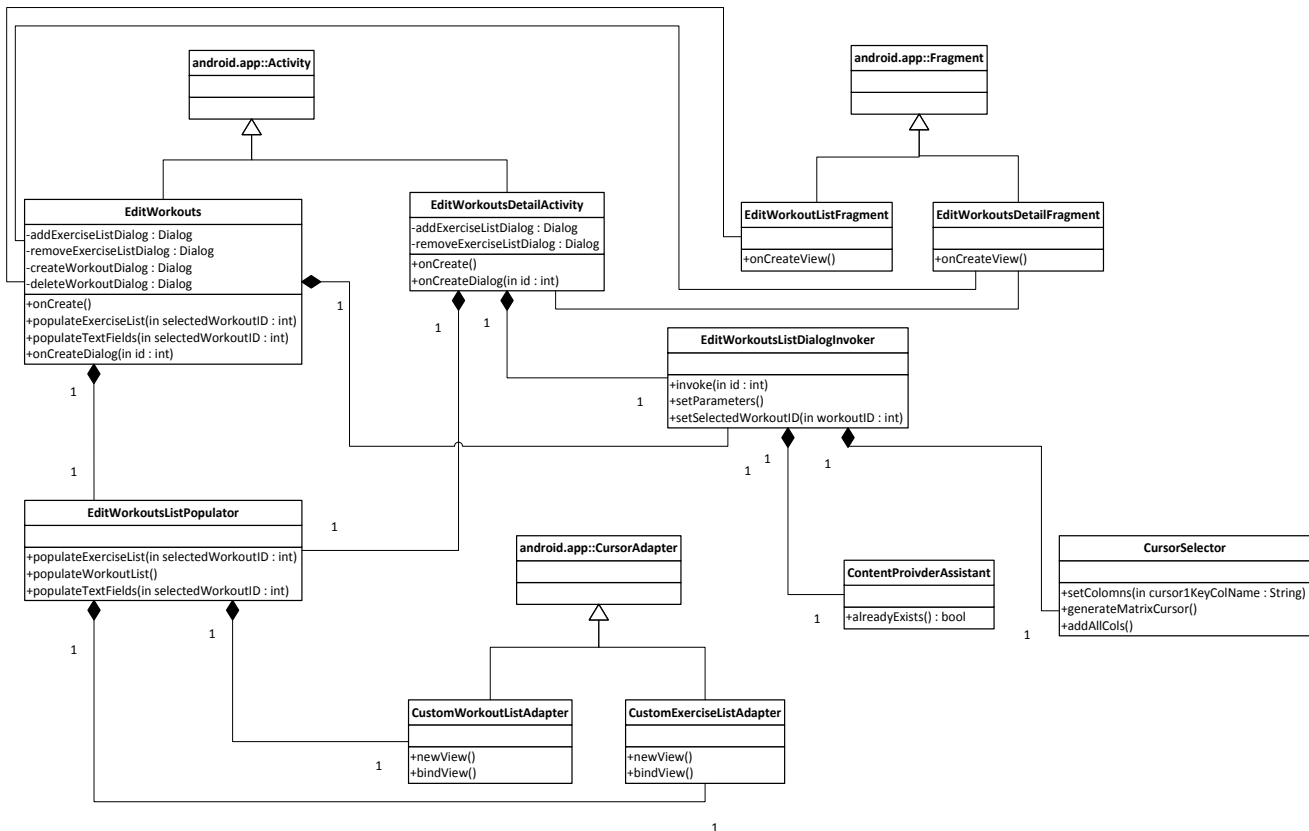


The view statistics activity will use 2 fragments. These are the “ViewStatisticsListFragment” and the “ViewStatisticsDetailFragment”. Depending on the device used they will be displayed as above.

The purpose of this activity is to allow the user to see their previous performance in a graphical way. This way they can see if their general performance is increase as it should be. The statistics will be viewable in terms of average weight, average reps and rest time.

Class structure design:

With the GUI and application flow defined we can define the classes and android activities to be used. The below UML diagram is only for one section of the application as this design can be copied to the other sections. Most sections of the application consist of 2 fragments, listviews and dialogs so are very similar.



The purpose of the main activity (EditWorkouts) and the portrait activity have been described on page 20. In addition to these we have the following classes which serve the following purposes:

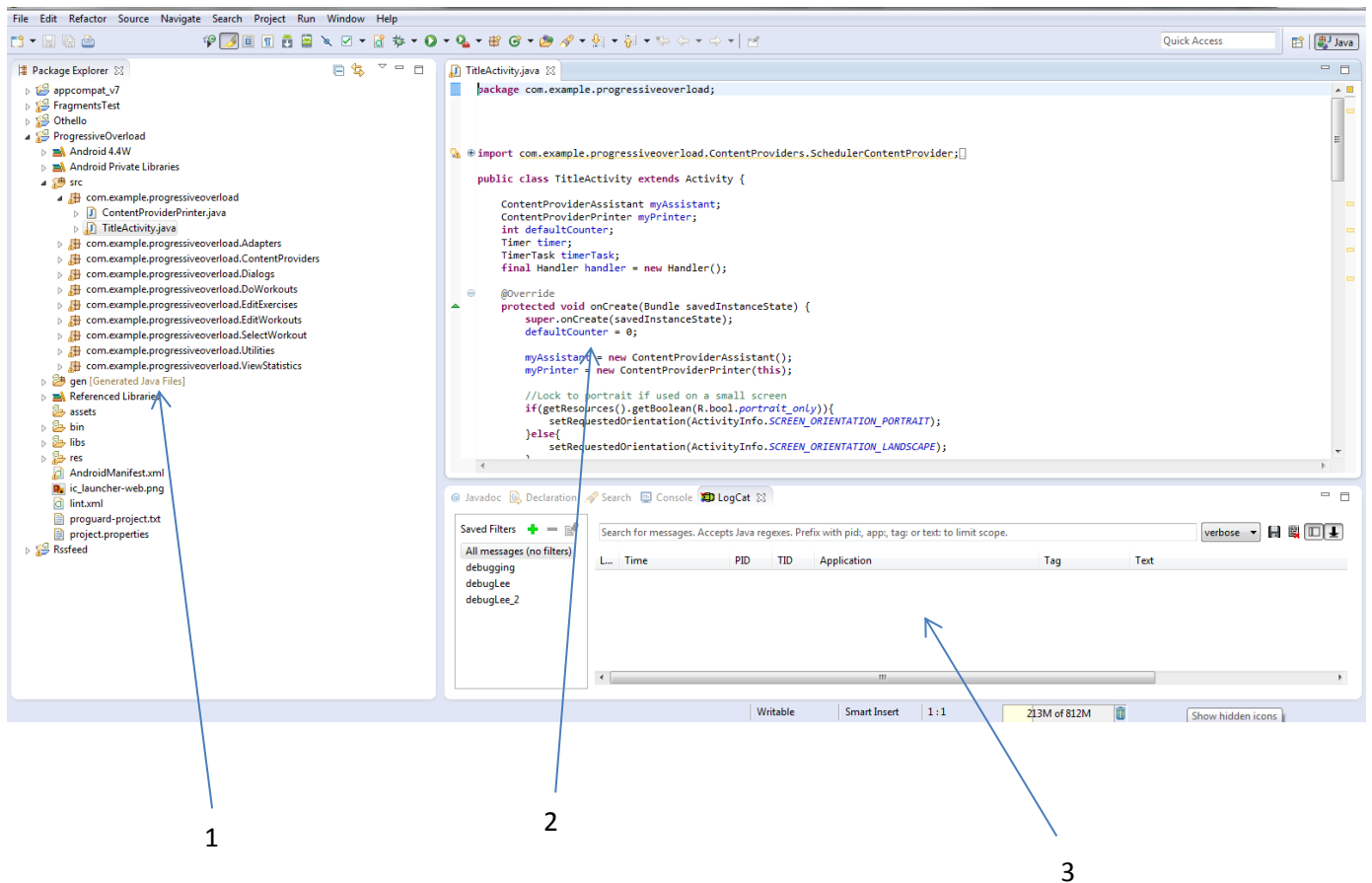
1. DisplayPopulator (EditWorkoutsListPopulator)
 - a. This class is a subclass of a custom class called the “DisplayPopulator” (inheritance omitted for clarity). As both the main activity and the portrait activity may be displaying the same information I have created this class to take care of the display population. Things like lists and text views will be populated using this class and hence only defined once. This serves to avoid duplicating code.
2. CursorAdapter (CustomWorkoutListAdapter and CustomExerciseListAdapter)
 - a. These classes serve to interface the SQL database / Content provider to a ListView. This is detailed later on in the “Accessing and displaying the database” section.
3. DialogInvoker (EditWorkoutsDialogInvoker)
 - a. This class takes care of the dialog invocation when for example creating a new exercise. This is detailed later on in the “Dialogs” section.
4. ContentProviderAssistant
 - a. This is a basic class used to perform basic operations on the content provider. Common operations to be performed on content providers are implemented here to avoid code duplication.
5. CursorSelector

- a. This is a specialised class that was only used in the “EditWorkoutActivity” in the end. It performs a selection between 2 content providers. Specifically it was used to display all exercises that belonged to a particular workout.

Implementation:

Using Eclipse ADT:

This application will be created using the Eclipse Android development tools (ADT). This is a plugin for Eclipse and not only allows you to create an android project and create an APK file but also allows online debugging.



1. Project View

- a. This contains all the source code and resources.

2. Code window

- a. This displays the source of the selected file

3. Console

- a. This contains messages from the Dalvik virtual machine on the device. It can also be used for debugging.

As seen above the android plugin doesn't alter the appearance of eclipse, it's all quite familiar to anyone who has used it.

Android logcat:

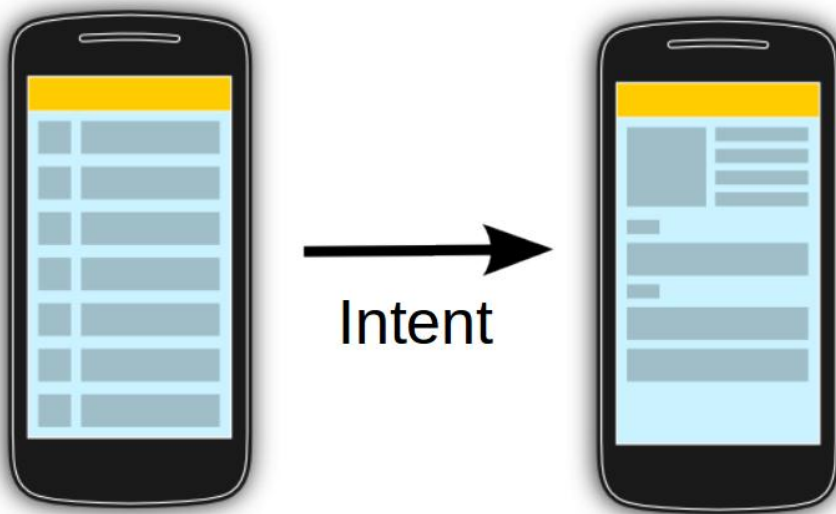
Android has a class called "logcat" that can be very useful with debugging and works extremely well with eclipse. Just like a print statement one can print to the logcat providing a message and a tag. Eclipse can then filter logcat messages depending on the tag name given:

```
Log.d("debug", "Count Before = " + cursor.getCount());
```

This example was used for counting database entries with the tag "debug". Now the printed information can be displayed on the console and distinguished from other messages using a filter looking for the tag debug.

Activity Intercommunication:

In android the different activities are considered separate components; there are no shared components or variables between them; they are isolated. In order to communicate between the activities one must use what is called an intent. An intent is an object that can start a specific activity and pass data to that activity. An intent can also be used to start an activity for a particular function without knowing what activity it is required, for example to make a phone call. An intent that knows the specific activity to start is called an “Explicit Intent” and an intent that is to start an unknown activity is called an “Implicit Intent”.



(www.vogella.com)

Explicit intent:

The vast majority of intents used in this application will be explicit intents, i.e., the activity we want to start is specific and known. For example, in the title screen we want to start the edit exercise activity:

```
public void editExercisesButtonPressed(View view){  
    Intent intent = new Intent(this,EditExercises.class);  
    startActivity(intent);  
}
```

The above example is just starting an activity; it's not passing any data to that activity. In this application very little data will ever be passed between activities directly; activities will get data from the SQL database much like a “Blackboard” architecture.

Implicit intent:

As stated before, implicit intents are used to start an unknown activity to perform a known function. The progressive overload application does make use of implicit intents when it comes to selecting an image to assign to an exercise.

The application doesn't know how to get an image; it just knows that it wants an image. This is performed in the following way:

```
public void onClick(View v) {
    Intent intent;
    if(android.os.Build.VERSION.SDK_INT < 19){
        intent = new Intent();
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
    }else{
        intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("image/*");
    }

    startActivityForResult(Intent.createChooser(intent,
        "Select Picture"), SELECT_PICTURE);
}
```

I discovered whilst attempting to access images on an android device that the way the permissions work had been changed between Ice cream sandwich and KitKat, hence there is a conditional statement that decides exactly how to implement the implicit intent.

As with an explicit intent an Intent object is created. The method `intent.setType()` is called to define the type of data to be returned. The method `intent.setAction()` is then called to define the fact that we expect data back. We then start the unknown activity but this time we call the `startActivityForResult()` as we expect a result back. In this case the user is presented with the option so select an image from any application that can return an image; this includes the android gallery app.

By overriding the `onActivityResult()` we can receive data from that intent once the called activity has finished:

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK) {
        if (requestCode == SELECT_PICTURE) {
            selectedImageUri = data.getData();

            Bitmap bitmap = bitmapHandler.generateBitmap(this,
                selectedImageUri, R.integer.edit_exercise_photo_size, R.integer.
                edit_exercise_photo_size);
            ImageView photoIcon = (ImageView)
                this.findViewById(R.id.exercisePhoto);
            if(photoIcon != null){
                photoIcon.setImageBitmap(bitmap);
            }
        }
    }
}
```

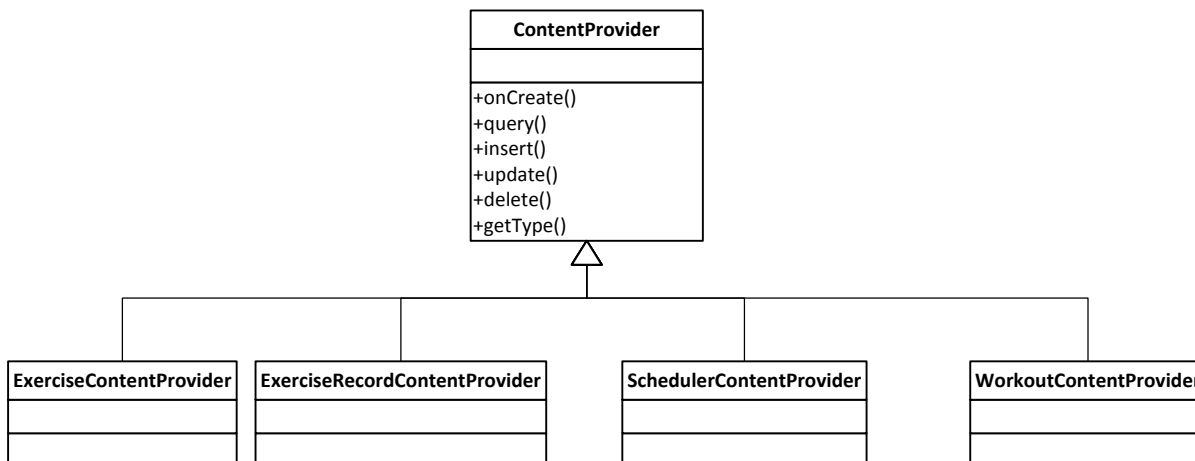
The called activity will call the `onActivityResult()` and provide the parameter "requestCode" so we can determine which activity is returning data (in this application we're only using one implicit intent but you can see how this could be extended for many). We set the constant `SELECT_PICTURE` when we created the intent so this is the value we expect on its return. In order to get the data from the intent (we know we're expecting an image Uri) we call `Intent.getData()`. Now we have the data from the implicit intent and can use it how we wish. In this case we're using our custom class "BitmapHandler" to generate a bitmap and setting the `ImageView R.id.exercisePhoto` to display that image.

Creating, accessing and displaying the database:

Creating the database:

A large part of this application was storing the data and displaying the data in each activity. I chose to store data in an android content provider as opposed to a standalone SQL database. The main reason for this was future proofing the application. The main difference between using a content provider and a standalone database is that data stored in a content provider can be accessed from other applications. A content provider is essentially a wrapper for an SQL database; it provides it with a global location.

With the database defined I then had to implement the content provider. Practically this means writing a class that extends `ContentProvider`. I chose to create a content provider for each database table:



Creating a custom content provider involves overriding the super class methods and defining an SQL table. In order to make the data accessible in other applications we must define what is called an “Authority”:

```
public static final String AUTHORITY =  
"com.example.progressiveoverload.ContentProviders.workoutprovider";
```

We must then define the database, the content provider name and the column names in the table:

```
private SQLiteDatabase workoutDB;  
private static final String WORKOUT_PROVIDER = "workoutprovider";  
public static final String KEY_WORKOUT_ID = "_id";  
public static final String KEY_WORKOUT_DAY = "Day";  
public static final String KEY_MUSCLE_GROUP = "Muscle_group";  
public static final String KEY_DATE = "Last_performed";
```

Within the content provider we define a static custom class that extends the `SQLiteOpenHelper` class. This class will assist in the actual creation of the SQL database. Then by overriding the `onCreate()` method to actually create an instance of this class we can call an overridden `onCreate()` in the `SQLiteOpenHelper`:

```

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + WORKOUT_PROVIDER + " (" + KEY_WORKOUT_ID
        + " INTEGER PRIMARY KEY AUTOINCREMENT, " + KEY_WORKOUT_DAY
        + " TEXT," + KEY_MUSCLE_GROUP + " TEXT," + KEY_DATE + " DATETIME);");
}

```

This uses the strings to create the SQLite database.

Accessing and displaying the database:

The vast majority of database information will be displayed as lists but on occasion it will be required to display a single piece of information. Either way this information will need to be displayed by both the main activity and the portrait activity(see the “Gui Design” section). As detailed in the “Class structure design” section these operations were encapsulated in a “DisplayPopulator” class.

Displaying data in a ListView:

As mentioned above, the majority of information from the database will be displayed using ListViews. I have done this by extending the android “CursorAdapter” class. These custom adapters take what is called a “Cursor” as a parameter in their constructor. A Cursor is essentially the start of a result of a database query. What the adapter has to display must first be produced as a Cursor. Here is an example of that kind of query to get all entries in the ExerciseContentProvider and display them in a ListView:

```

public ListView populateExerciseList(){

    Cursor selectionCursor =
myContentResolver.query(ExerciseContentProvider.CONTENT_URI,null, null,
null, ExerciseContentProvider.KEY_EXERCISE_NAME+" ASC");

    exerciselListAdapter = new CustomExerciseListAdapter(hostActivity,
        selectionCursor);
    ListView exerciselListView = (ListView)
hostActivity.findViewById(R.id.exerciselList);
    if(exerciselListView != null){
        exerciselListView.setAdapter(exerciselListAdapter);
        return exerciselListView;
    }
    return null;
}

```

In the above example a Cursor is created using a “ContentResolver” class query method. The query method has the following paramters:

```
ContentResolver.query(uri, projection, selection, selectionArgs, sortOrder);
```

In the example we are passing the content provider URI as the uri parameter. For the projection, selection and selectionArgs parameters we are using null, this will project and select all elements in the content provider. We are then sorting the results by exercise name in ascending order.

Now we have a Cursor that contains all exercise content provider entries we can create an instance of the exerciselListAdapter. This is a relatively simple class to map the data over to a list view:

```

public class CustomExerciseListAdapter extends CursorAdapter {
    private LayoutInflater mLayoutInflater;
    private Context mContext;
    public CustomExerciseListAdapter(Context context, Cursor c) {
        super(context, c);
        mContext = context;
        mLayoutInflater = LayoutInflater.from(context);
    }

    @Override
    public View getView(Context context, Cursor cursor, ViewGroup parent) {
        View v = mLayoutInflater.inflate(R.layout.exercise_listview_contents,
            parent, false);
        return v;
    }

    @Override
    public void bindView(View v, Context context, Cursor c) {
        String exerciseName = c.getString(c.getColumnIndexOrThrow(
            ExerciseContentProvider.KEY_EXERCISE_NAME));

        TextView nameTextView =
            (TextView)v.findViewById(R.id.exercise_name_textview);
        if (nameTextView != null) {
            nameTextView.setText(exerciseName);
        }
    }
}

```

The `getView()` method inflates a layout file which describes a single line in the list. In this example we are only displaying the name of the exercises in the list, therefore the layout defined in the XML file `exercise_listview_contents` just specifies a single `TextView` with the ID `exercise_name_textview`. We could display whatever we like in a single line, it just has to be described in this layout file.

The `bindView()` method is the method that takes the data from the `Cursor` and writes it to the `TextView` in the layout file. This method is called for every entry in the `Cursor`.

Displaying a single piece of data:

It was occasionally a requirement to display a single piece of data in a `textView`. An example of this would be in the `SelectWorkoutsActivity` when it was a requirement to display the last exercise that was performed. In this case I did a single `Cursor` query:

```

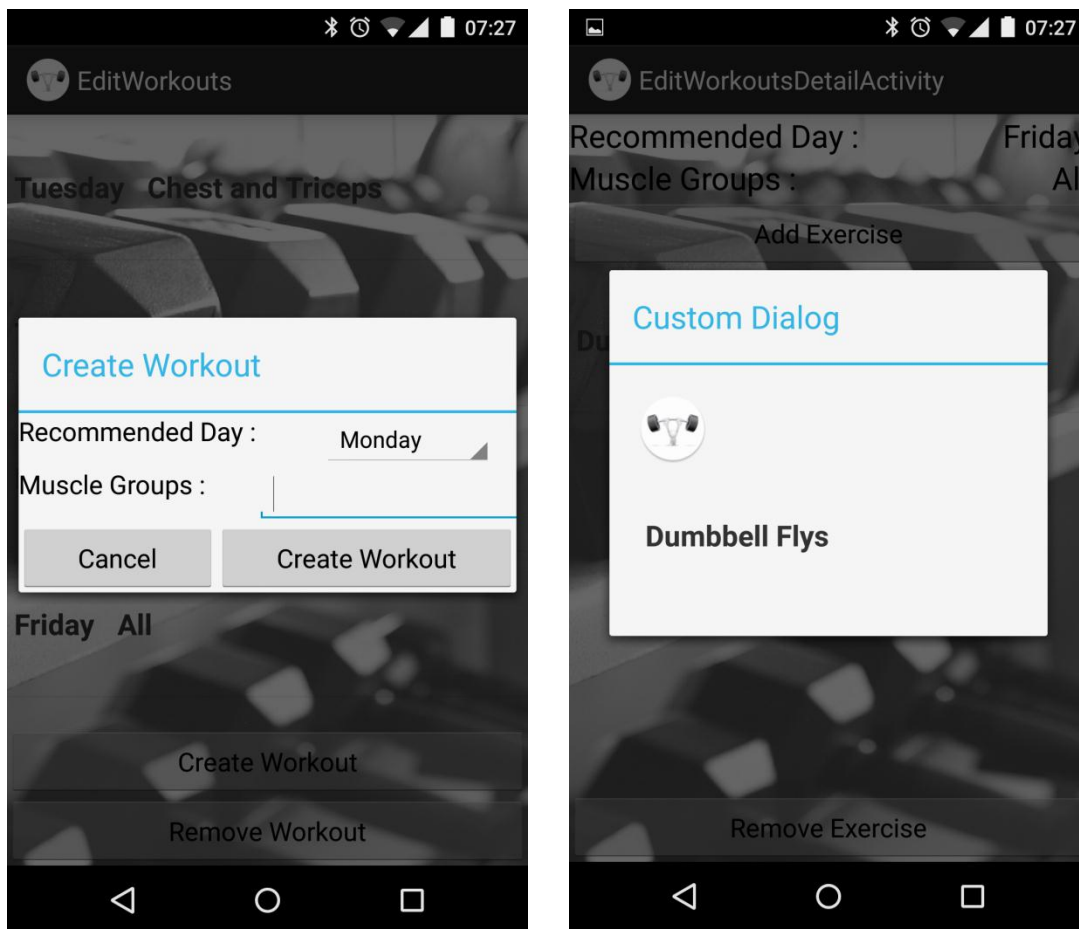
Cursor selectionCursor =
myContentResolver.query(WorkoutContentProvider.CONTENT_URI,null, null, null,
WorkoutContentProvider.KEY_DATE + " DESC");
int dayCol = selectionCursor.getColumnIndexOrThrow(
    WorkoutContentProvider.KEY_WORKOUT_DAY);
int groupCol = selectionCursor.getColumnIndexOrThrow(
    WorkoutContentProvider.KEY_MUSCLE_GROUP);
if(selectionCursor.moveToFirst())
    if(getResources().getBoolean(R.bool.portrait_only)){
        lastPerformedDay = selectionCursor.getString(dayCol);
    }else{
        lastPerformedDay = selectionCursor.getString(dayCol) + ",
"+selectionCursor.getString(groupCol);
    }
lastPerformed.setText(lastPerformedDay);

```

To do a query in this manner I did a query on the content provider that produced a Cursor with all workouts sorted by date. This way I know the first item in the Cursor is also the most recent. The variables dayCol and groupCol hold the column numbers for the workout day and workout muscle group columns. The next conditional statement checks there is actually data in the first entry of the cursor. If there is it then decides if the application is in portrait or landscape and then decides how to format that data accordingly.

Dialogs:

Dialogs are an important part of the application. I have used them for both creating a new workout and removing an exercise from a workout, adding an exercise to a workout and removing a workout. Wherever a dialog is used I have used a custom dialog invoker class (detailed later). This is to both simplify the activities and to encapsulate dialogs that may be used in more than one activity. All dialogs are listview dialogs with the exception of the create workout dialog. Both types of dialog can be seen below:



To invoke a dialog as above (I will use the create workout dialog as an example as this is the most complex) I first created an instance of the dialog class:

```
createWorkoutDialog = new Dialog(this);
```

I then created an instance of the custom dialog invoker class. The constructor for this takes the following parameters:

```
EditWorkoutsListDialogInvoker(Activity hostActivity, int dialogLayout, int  
listViewID, String dialogDesc, Dialog dialog)
```

All dialogs require an activity; this is usually just “this” as the dialog invoker is created in an activity. The dialog layout is the address of the XML layout file for the dialog. The parameter listViewID is the ID of the list view to be used if there is a list in the dialog. As in the case of the create workout dialog it wasn’t so this was just left as 0. The parameter dialogDesc is the text that will appear at the top of the dialog and the parameter dialog is the instance of the previously created dialog. For the example of the create workouts dialog the following constructor was used to instantiate a custom dialog invoker class:

```
EditWorkoutsListDialogInvoker(this,R.layout.create_workout_dialog,0,"Create Workout",createWorkoutDialog);
```

Showing a dialog:

Now the dialog and the dialog invoker has been created we can show the dialog when a particular event happens. To do this we call the showDialog(int) method in the activity class where the int is a constant used to determine the exact dialog to call. This eventually calls the onCreate(int) method, again in the activity class. We can override this method to decide what dialog to call:

```
@Override
protected Dialog onCreateDialog(int id){
    if(id == EXERCISE_LIST_DIALOG)
        return addExerciseDialogInvoker.invoke(id);
    if(id == CREATE_WORKOUT_DIALOG)
        return createWorkoutDialogInvoker.invoke(id);
    if(id == REMOVE_WORKOUT_DIALOG)
        return deleteWorkoutDialogInvoker.invoke(id);
    if(id == REMOVE_EXERCISE_DIALOG)
        return removeExerciseDialogInvoker.invoke(id);
    return null;
}
```

By calling showDialog(int) with an identifier we can then call our custom dialog invokers invoke(id) method. As stated before the reason I have done it this way is because multiple activities will want to call this dialog, both the main activity and the portrait activity. Creating a class that takes care of the actual invocation serves to reduce code duplication.

Inside the EditWorkoutDialogInvoker.invoke(id) method there is a case statement that decides what to do depending on the integer passed to it. For example we will pass an integer to it in this case that indicates the create workout dialog is to be invoked.

In the invoke() method the logic is pretty simple. First of all the elements in the dialog are identified. This involves creating buttons and spinners, etc., from the layout file:

```
Button createWorkoutButton = (Button)dialog.findViewById(R.id.createWorkoutCreateButton);
Button cancelCreateWorkoutButton = (Button)dialog.findViewById(R.id.createWorkoutCancelButton);
final Spinner createWorkoutDayIn = (Spinner)dialog.findViewById(R.id.createWorkoutDayField);
final EditText createWorkoutGroupIn =
(EditText)dialog.findViewById(R.id.createWorkoutMuscleGroupField);
```

Once we have the objects created we can decide what to do when the user clicks the “cancel” and the “Create workout” buttons. When the user clicks “cancel” we want to exit the dialog. This is done by creating a listener that listens for the button being actuated:

```
cancelCreateWorkoutButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        hostActivity.dismissDialog(CREATE_WORKOUT_DIALOG);
    }
});
```

This dismissDialog(int) not only dismisses the dialog but also triggers a listener that has been defined in the parent activity. This way we can stipulate what is to be done when a dialog is dismissed, in this case we do nothing:

```
createWorkoutDialog.setOnDismissListener(new OnDismissListener(){
    @Override
    public void onDismiss(DialogInterface dialog) {

    }});
```

The “Create Workout” button is doing slightly more. It takes the values found in the “Recommended Day” spinner and the “Muscle Groups” EditText and creates an entry in the database for the workout, assuming it is an allowed entry:

```
createWorkoutButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        String day = "";
        String muscle = "";

        if(createWorkoutDayIn!=null)
            day =createWorkoutDayIn.getSelectedItem().toString();
        if(createWorkoutGroupIn!=null)
            muscle = createWorkoutGroupIn.getText().toString();

        if((day.length()>0)&&(muscle.length()>0)){

            if(contentProviderAssistant.alreadyExists
                (hostActivity,WorkoutContentProvider.CONTENT_URI,new String[] {
                    WorkoutContentProvider.KEY_WORKOUT_DAY},new String[]{day}))){
                Toast.makeText(hostActivity.getApplicationContext(), "Workout already
                    exists for that day", Toast.LENGTH_SHORT).show();}
            else{
                ContentResolver myContentResolver =
                    hostActivity.getContentResolver();
                ContentValues values = new ContentValues();

                values.put(WorkoutContentProvider.KEY_WORKOUT_DAY, day);

                values.put(WorkoutContentProvider.KEY_MUSCLE_GROUP, muscle);

                myContentResolver.insert(WorkoutContentProvider.CONTENT_URI,
                    values);
            }
        }else{
            Toast.makeText(hostActivity.getApplicationContext(), "Invalid Day or Muscle
                Group", Toast.LENGTH_SHORT).show();
        }
        hostActivity.dismissDialog(CREATE_WORKOUT_DIALOG);
    }
});
```

This was the most complex dialog in the application. It looked like this:

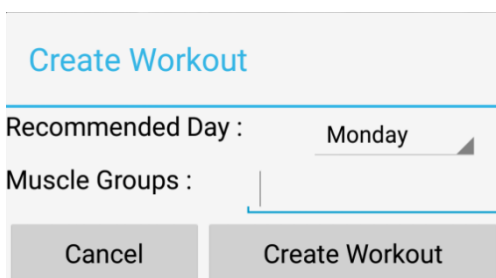


Image Acquisition and Conditioning:

As previously described the user is able to select a photo to be displayed for each exercise. In the “Implicit Intent” section I detail how the user is able to get an image using the native gallery app or any other app that is capable of returning an image. Whilst this makes the user experience quite polished and professional it does raise an issue. As the user can select any image they may select a very large image, selecting a large image not only takes up processing time but can also look odd if its forced to display on a very small area of the screen. In order to prevent this I needed to find out the size of the selected image before displaying it and take action accordingly.

To handle this I decided to make a custom class called “BitmapHandler”. The BitmapHandler class has a method that returns a bitmap from a provided Uri that conforms to a provided width and height. It does this using the android BitmapFactory class and an input stream. The BitmapFactory class takes an object “Options” which stipulates the operating parameters for the BitmapFactory. By setting a Boolean variable within the options object called “inJustDecodeBounds” to true we can access the details of the image without actually creating it:

```
//Decode image without returning a bitmap, just used to get projected size.
InputStream = mContentResolver.openInputStream(imageUri);
options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeStream(inputStream,null,options);
inputStream.close();
```

Now we have decoded the input stream with the given Options object we can access the height and width of the image and calculate a scale factor to apply in order to convert the image to the correct size. The Options.inSampleSize allows a scale factor to be set:

```
options.inSampleSize = calculateInSampleSize(options, width, height);

public static int calculateInSampleSize(BitmapFactory.Options options, int reqWidth, int reqHeight) {
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if (height > reqHeight || width > reqWidth) {

        final int halfHeight = height / 2;
        final int halfWidth = width / 2;

        // Calculate the largest inSampleSize value that is a power of 2 and keeps both
        // height and width larger than the requested height and width.
        while ((halfHeight / inSampleSize) > reqHeight && (halfWidth / inSampleSize) > reqWidth) {
            inSampleSize *= 2;
        }
    }
    return inSampleSize;
}
```

(Developer.android.com)

Now the Options.inSampleSize variable has been set with a scale factor we set the Options.inJustDecodeBounds to false and call BitmapFactory.decodeStream again, this now returns the scaled bitmap.

Custom Graph View:

A requirement of the application was also to display previously achieved performance. Displaying this as a graph seemed like a good way to instantly tell if user performance was improving. Originally I intended to create my own custom view in android but due to the lack of time I decided to use someone else's library. I found an open source library called "Graph View" (Gehring,2015) create by Jonas Gehring that seemed a good fit.

This library was extremely easy to use. Firstly an element in the XML layout file was created to stipulate where the graphview would appear and how it would look:

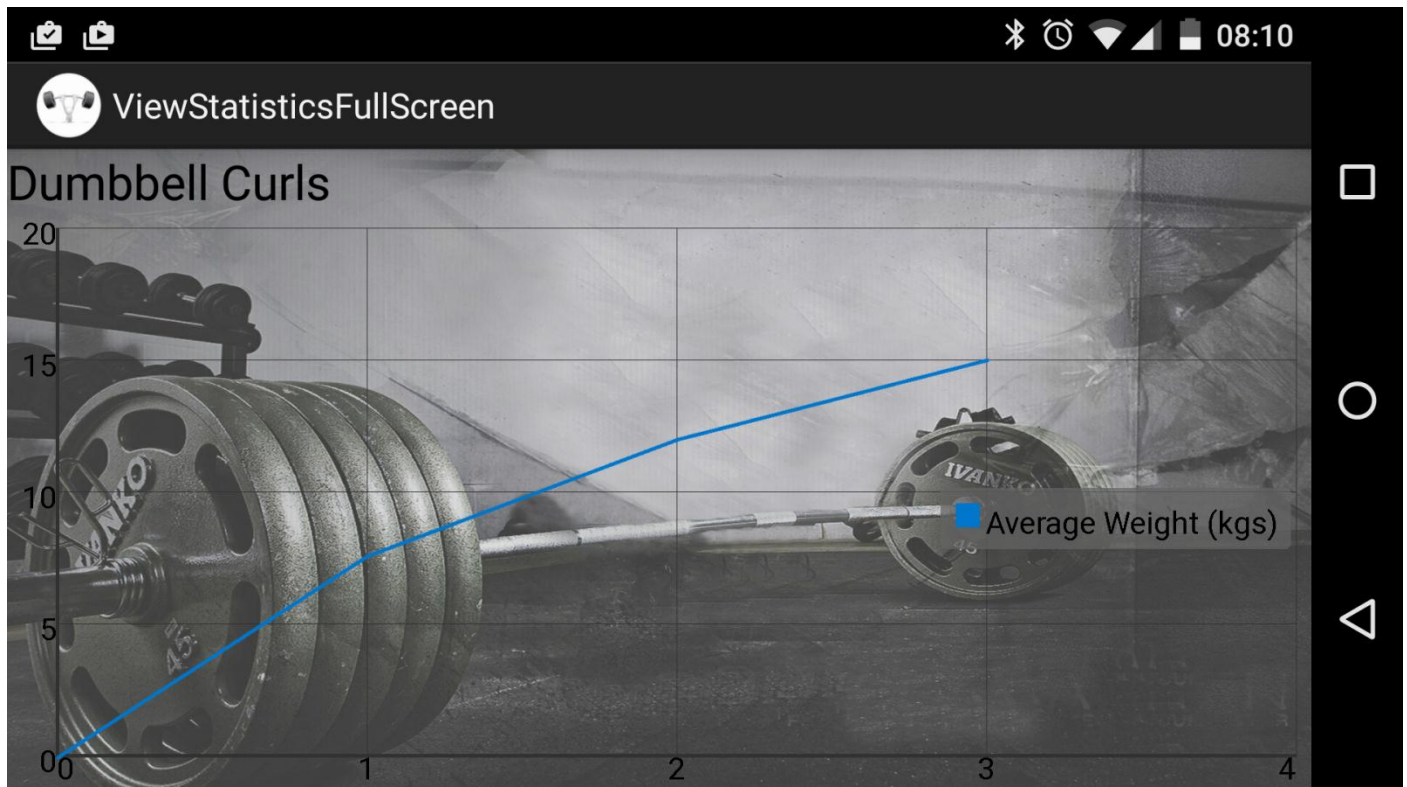
```
<com.jjoe64.graphview.GraphView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:id="@+id/graph" />
```

Note in the android:id field I have called this "graph". This is so it can be identified in the code. This is how all GUI components are identified.

To get the graph to display data a List of datapoint objects called a "LineGraphSeries" must be created. A datapoint is an object included in the library and is just an object that contains 2 floating points, the X and the Y coordinates for a single entry. In the below code I have created an array of DataPoints called "dataPoints" and put these in a LineGraphSeries<> called series. The graph.addSeries(LineGraphSeries<DataPoint>) is then called with the datapoints. The series.setTitle(String) method can be used to set the title of the series as shown below.

```
graph.removeAllSeries();
LineGraphSeries<DataPoint> series = new LineGraphSeries<DataPoint>(dataPoints);
series.setTitle(legend);
graph.getLegendRenderer().setVisible(true);
graph.addSeries(series);
```

The resulting graph appears like this:



Problems Encountered:

Most of the problems encountered making this application have been detail in the previous sections. Below is a summary of the main issues encountered:

1. Implementing a selection and projection query across more than one content provider.
 - a. When trying to display the exercises for a given workout I needed to only display the exercises that appeared in the scheduler entity along with the workout. This involved doing a selection and projection query across multiple content providers. If I had just used a single content provider with multiple tables this would have been easy, however too much time had been spent when I realised this was a problem. To overcome this issue I created custom class “CursorSelector”. This essentially did the selection and projection query and returned what is called a “MatrixCursor”. This is used in the same way as a conventional cursor but it is one the CursorSelector class created by traversing the database and making selections.
2. Differences between different versions of android.
 - a. The only issue I encountered relating to this is how the permissions work. There was an update in build version 19 that made it hard to display an image in activities subsequent to its selection. The exact details of this problem and its solution can be found in the “Implicit Intent’s” section.
3. Avoiding the application crashing due to the user selecting a large image.
 - a. As detailed in the “Image acquisition and conditioning” section I initially had problems with the application crashing when a large image was selected in the EditExerciseActivity. How I got around this problem is detailed in the aforementioned section.
4. Problems with Eclipse.
 - a. Whilst on the whole I found Eclipse was easy to use I did have an annoying issue. Quite often eclipse would not perform a clean rebuild after a code modification. This resulted in the application being run without the modifications being applied. There was no indication when this was happening apart from the build time was less. To rectify this I had to restart eclipse. This was not a common occurrence but did serve to confuse me at times. I didn’t get to the bottom of it.
5. Deciding if the device being used is a tablet or a phone.
 - a. This was not so much of a problem but is worth mentioning. In order for the application to decide what kind of device is being used I created a Boolean variable “portrait_only” in the “values” folder of the project and set it to true. I then created a folder “values-sw600dp” with the same Boolean variable set to false. The operating system will use the “values” folder by default and only use the “values-sw600dp” if the screen being used has a width greater than 600dp. In the code I then checked this variable and locked the device to portrait or landscape accordingly.

Testing:

Functional Test:

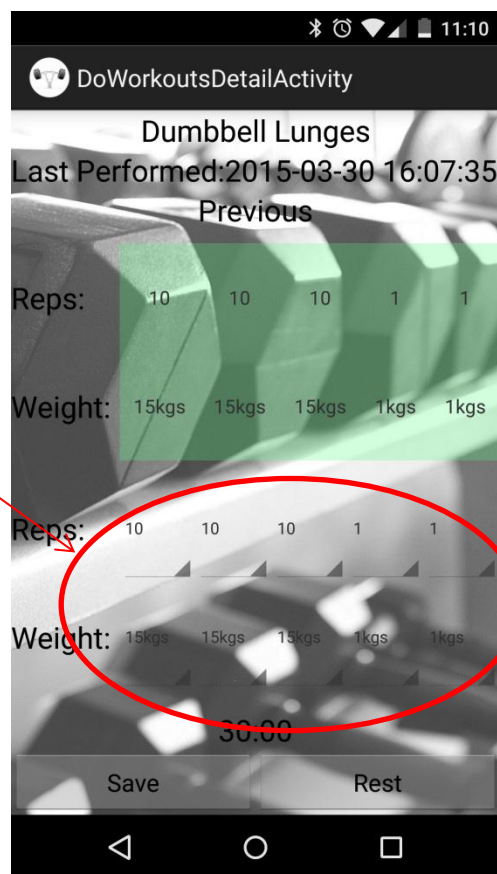
The first part of the testing phase was the functional testing, this involved going through the application and testing every user interaction. A full test plan was created and completed. See Appendix A.

Real World Test:

In order to see how the application would perform in the real world the application was given to 3 test subjects for a week. Each test subject would do at least 3 workouts in that week and report back with any bugs or improvements. Ideally a longer duration test would have been done but this test did prove useful. The following bugs / improvements were highlighted:

1. In the area where the actual workout is performed it is hard to keep track of how many of a given exercise you have done. If you have 5 sets of an exercise to perform how do you know which one you're on? It would be a good improvement to have the set appear red if it has not been completed:

Highlight these in red until the user interacts with them.



2. If you back out from the DoWorkoutsDetailActivity before pressing save your progress is lost. It would be good if this data was retained.
3. You can't change the number of sets to do. Setting the reps to 1 is good enough but this could be better.

Future Work:

Work to address issues found in real world test:

Obviously the improvements detailed in the “Real World Test” section could be addressed. The amount of work to implement these improvements varies. The points below relate to the points in the “Real World Test” section:

1. This should be trivial to implement. When a user interacts with either spinner the background colour could be changed.
2. This is less trivial. A new entry in the database could be created as a temporary store. This would require a fair amount of work but would be useful.
3. This is really not trivial. The Exercise Result entity on the database has a fixed number of entries for the results. Perhaps if there was some way to store the location of an arraylist rather than the raw data this could be allocated dynamically. Extensive work would be required here.

Other future work:

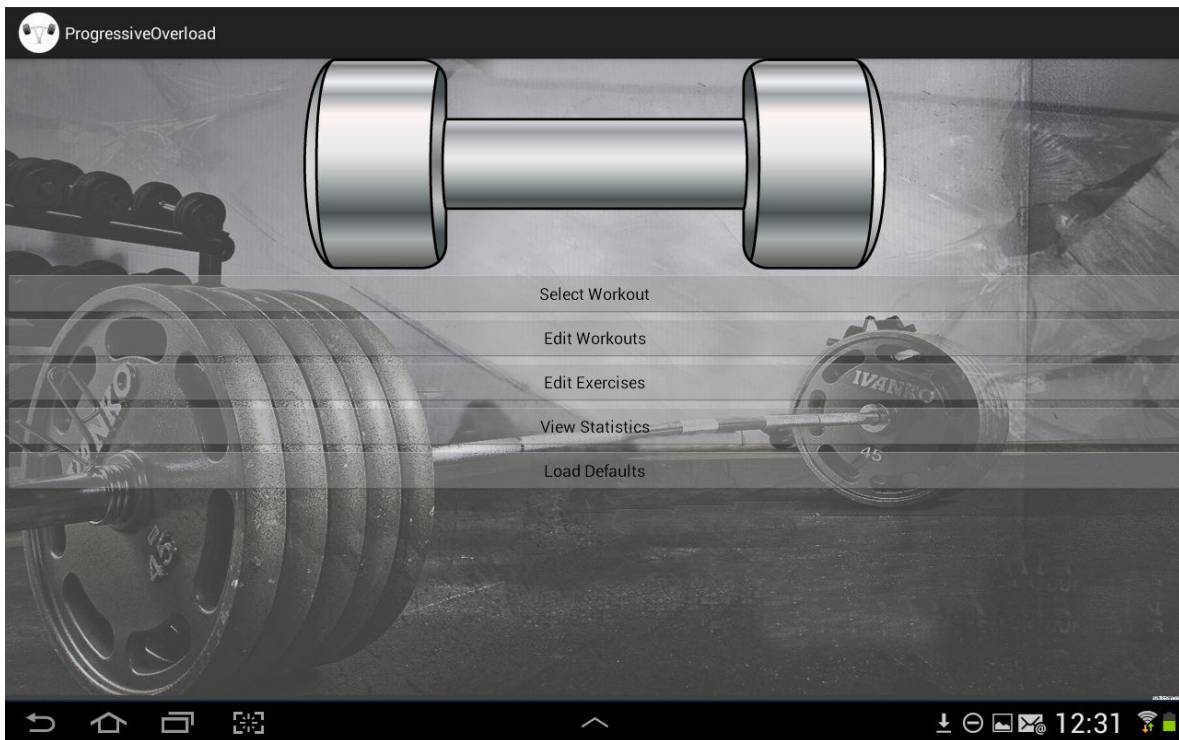
Other future work to be done would include:

1. Importation and exportation of data stored in the database.
 - a. If a user changes the device they use they would currently lose their data. On a very basic level this could just export a CSV of the database values then import them on the new device. A more elegant way to do this would be to store the database on the cloud. I’m not sure how to do this yet so more work would be required.
2. Use of custom artwork.
 - a. To avoid any copyright infringement it would be good to create the artwork for the application from scratch.
3. Have another section that monitors the users’ weight.
 - a. In a very similar way to the way other data is entered the user could keep track of their weight. This could then be displayed in the statistics page as a graph.
4. Compare performance to others.
 - a. It would be good if perhaps over Bluetooth the user could compare their performance to another user. Even being able to save the graph as an image would be useful for this.
5. Publish the application on the google play store.

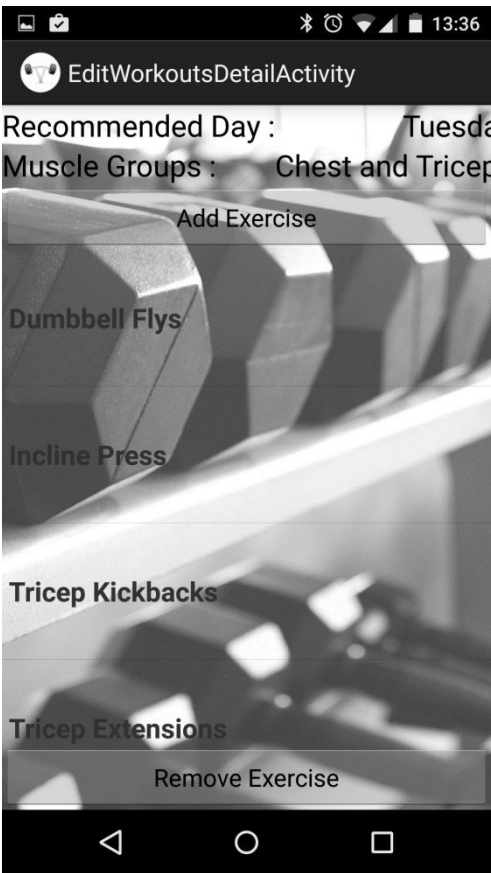
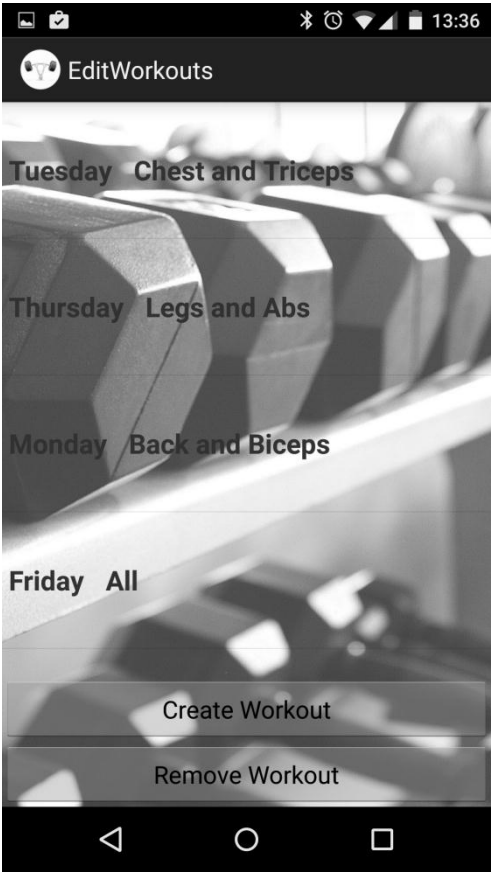
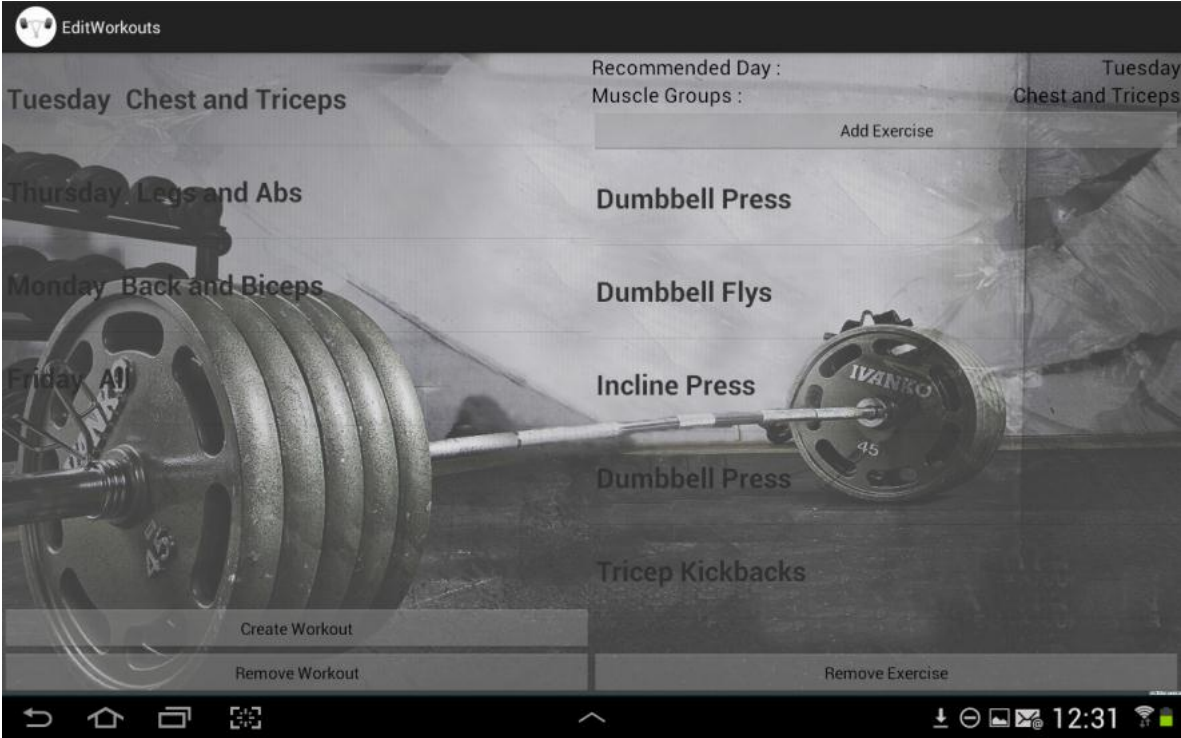
Conclusion and Evaluation:

Overall I have created a fully functioning android application that works well on both tablets and mobile phones:

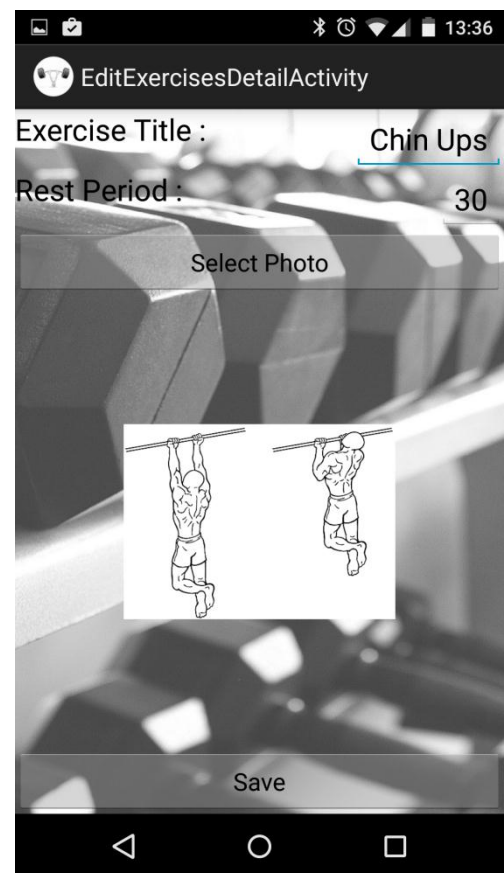
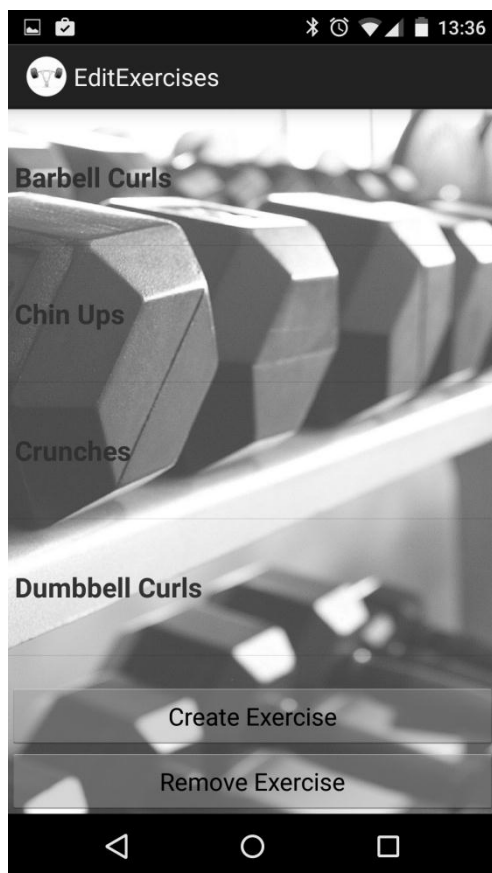
Title Page:



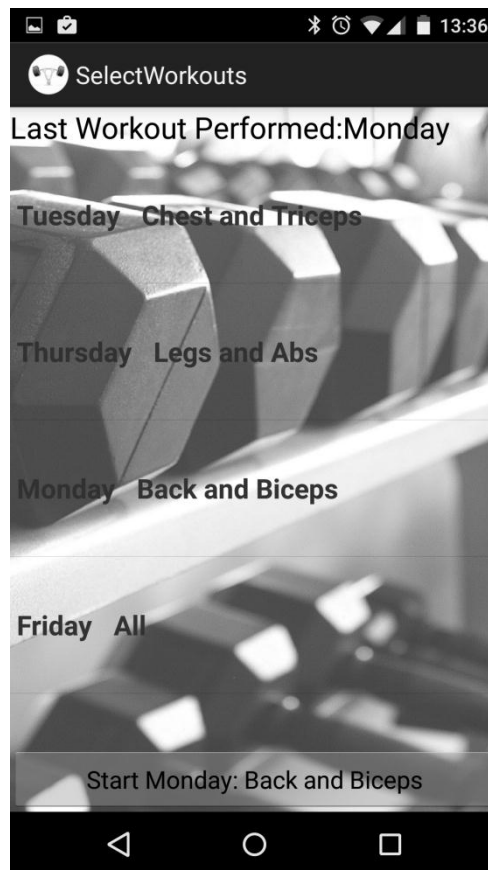
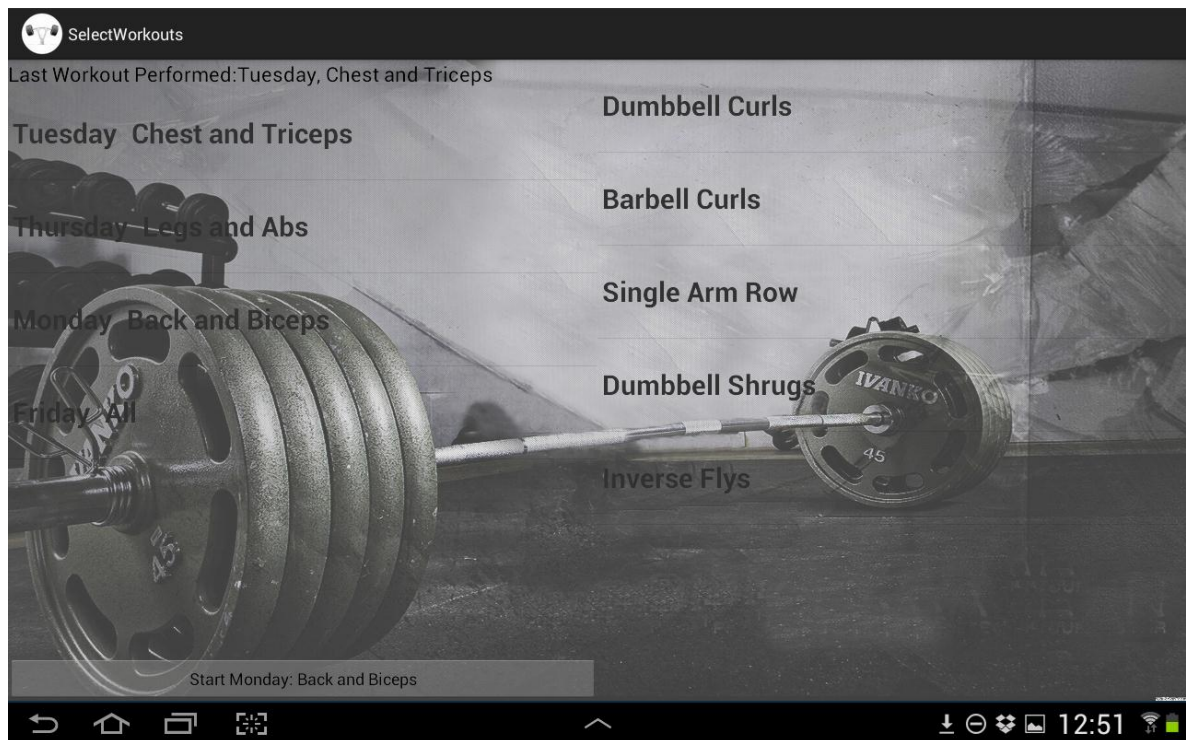
Edit workouts:



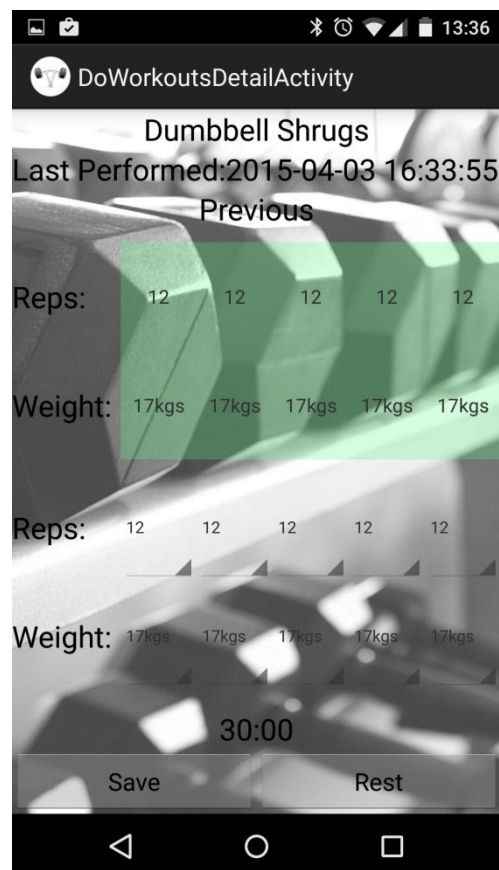
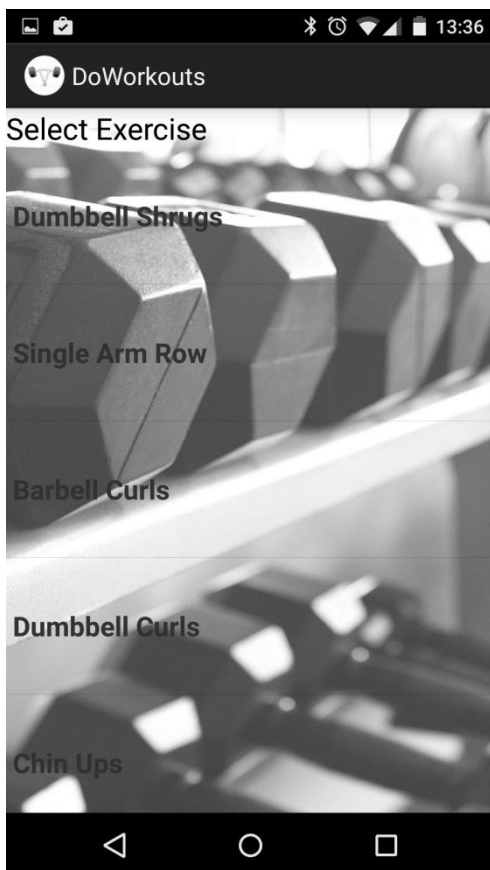
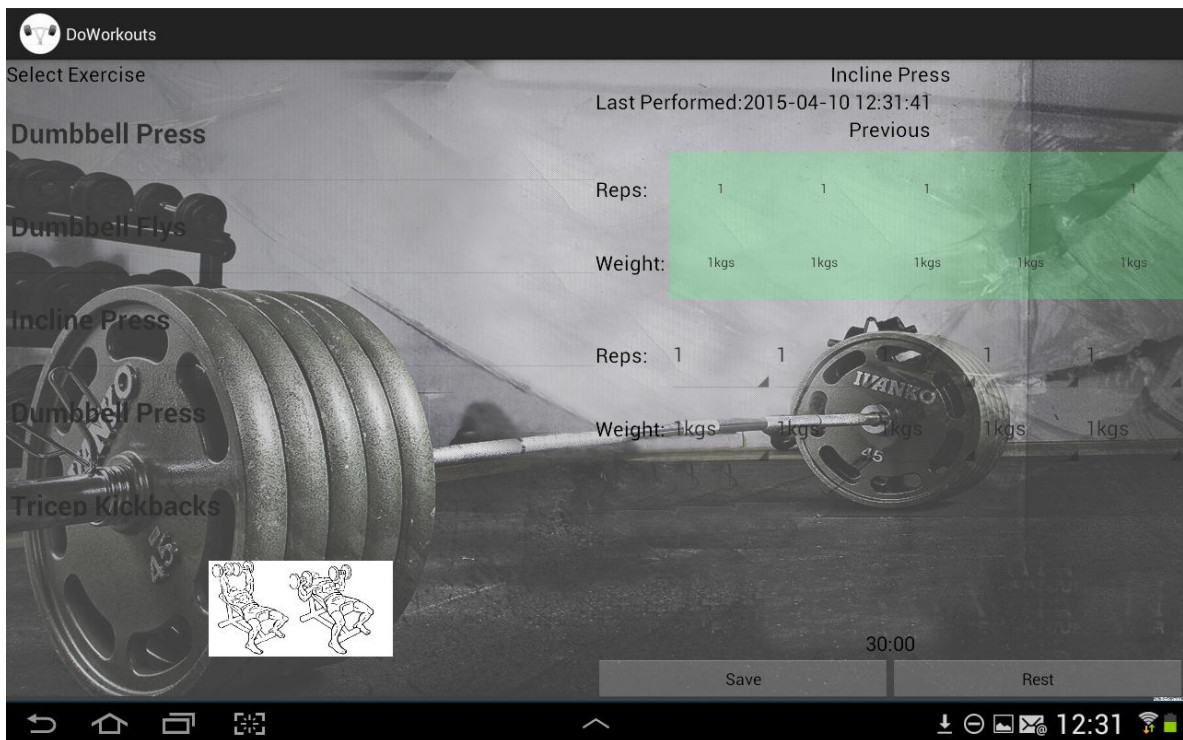
Edit Exercises:



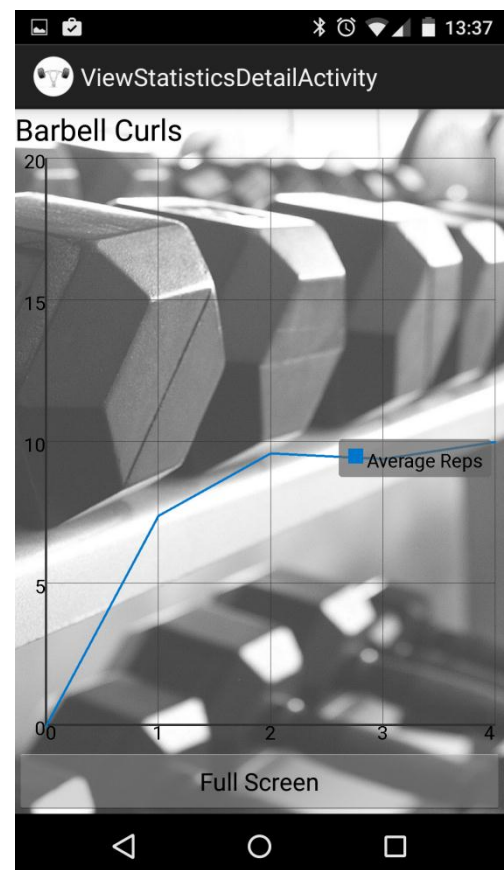
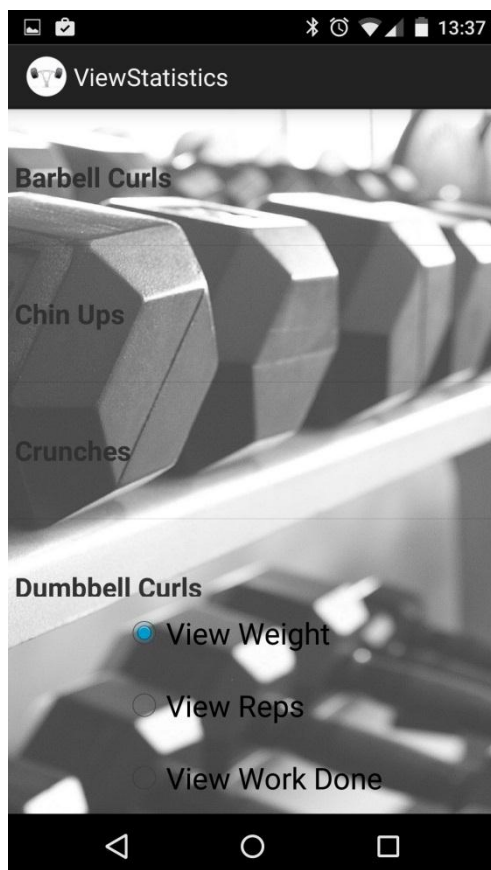
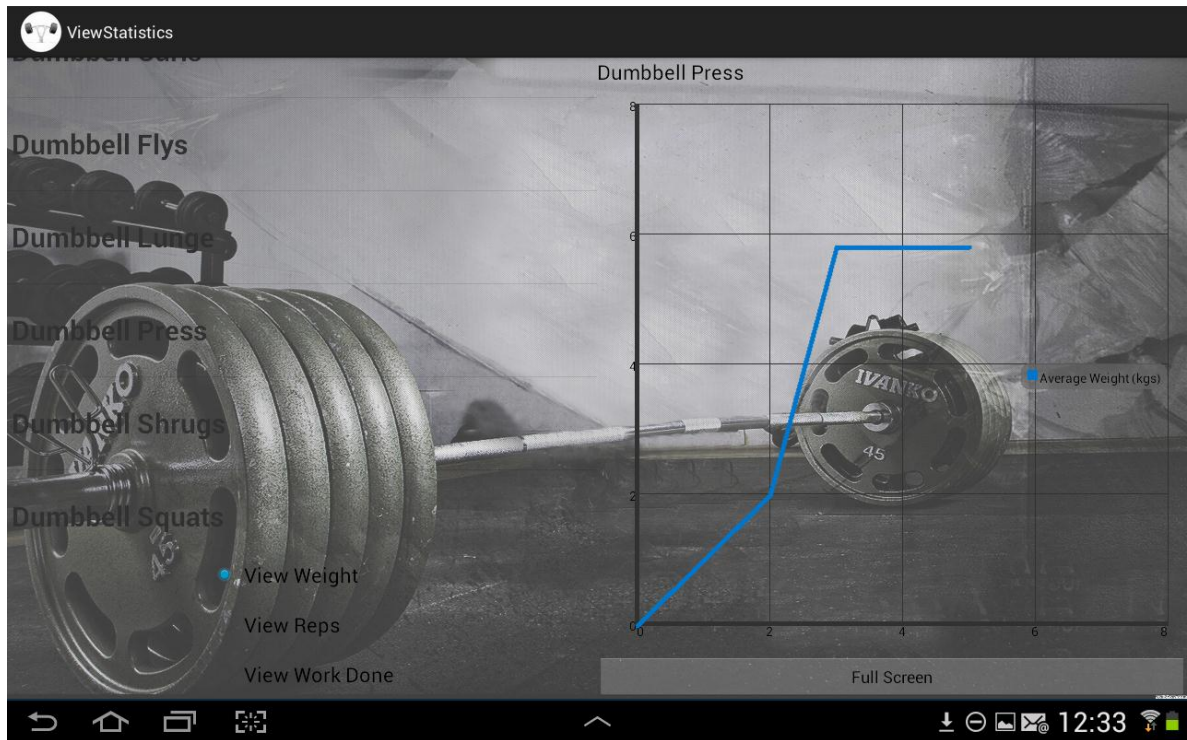
Select Workout:



Do Workout:



View Statistics:



As you can see from the above screenshots the application looks very different depending on the device it is being run on. It does have some areas that need improving as detailed in the “Future Work” section but overall it is a very functional application.

In undertaking this project I have developed my skill set. I now have a better knowledge of programming for the android platform and SQL databases. If I were to undertake this project again I could do a better job. It has been a learning process hence the usual software development methodologies don't really work. I initially picked the waterfall method but as I was learning as I went along I didn't rigidly stick to this methodology.

References and bibliography:

- I. Sherman,E.(2012) *How to choose the best platform for your app*. [Online] 08.10.12. Available from - <http://www.inc.com/erik-sherman/avoiding-the-single-mobile-platform-trap.html> [Accessed: 03.10.14]
- II. Bret(2013) *The ten rules of progressive overload*. [Online] 26.02.13. Available from - <http://bretcontreras.com/progressive-overload> [Accessed: 03.10.14]
- III. Vogel,L.(2014) *Multi-pane development in android with fragments - Tutorial* [Online] 29.04.14. Available from - <http://www.vogella.com/tutorials/AndroidFragments/article.html> [Accessed: 03.10.14]
- IV. Varshneya,R.(2013) *iOS or Android? Choosing the best platform for your application*. [Online] 17.07.13. Available from - <http://www.entrepreneur.com/article/227426> [Accessed: 03.10.14]
- V. Goulet,C.(2004) *Progressive overload: The concept you must know to grow!* [Online] 19.10.04. Available from - <http://www.bodybuilding.com/fun/goulet11.htm> [Accessed: 03.10.14]
- VI. Vogel,L.(2014) *Android SQLite database and content provider - Tutorial* . [Online] 19.08.14. Available from - <http://www.vogella.com/tutorials/AndroidSQLite/article.html> [Accessed: 03.10.14]
- VII. Gehring,J.(2015) *GraphView – open source graph plotting library for android - Library* . [Online] 19.08.14. Available from - <http://www.android-graphview.org/> [Accessed: 03.10.14]
- VIII. Unknown.(2014) *Smartphone OS Market Share-*. [Online] 03.10.14. Available from - <http://www.idc.com/prodserv/smartphone-os-market-share.jsp> [Accessed: 03.10.14]

Appendices:

Appendix A

Edit Exercise Activity Test Plan

Test number	Description	Expected result	Actual result	Pass or Fail
1	In the EditExercisesActivity, click create exercise and fill in the form for a new exercise (One that doesn't exist). Click save.	New exercise created and viewable.	As expected	PASS
2	In the EditExercisesActivity, click create exercise and fill in the form for an exercise that already exists. Click save.	"Exercise already exists" message appears, no exercise is created.	As expected	PASS
3	In the EditExercisesActivity, click on an exercise in the list.	Application jumps to the details page for the exercise	As expected	PASS
4	In the EditExercisesActivity, click on an exercise in the list. Edit each component of that exercise. Click save.	The new version of the exercise is stored. No duplication is made	As expected	PASS
5	In the EditExercisesActivity, click on an the remove exercise button. Select an exercise from the list.	The selected exercise is removed and the exercise list is updated	As expected	PASS

Edit Workouts Activity Test Plan

Test number	Description	Expected result	Actual result	Pass or Fail
6	In the EditWorkoutsActivity, click create workout and fill in the form for a new workout (On a day that currently doesn't have a workout). Click create workout.	A new workout is created with no assigned exercises and is viewable in the list.	As expected	PASS
7	In the EditWorkoutsActivity, click create workout and fill in the form for a new workout on a day that already has a workout on it. Click create workout.	"Workout already exists on that day" message appears, no workout is created.	As expected	PASS
8	In the EditWorkoutsActivity, click create workout and fill in the form for a new workout on a day that already has a workout on it. Click cancel.	Dialog is dismissed with no workout created.	As expected	PASS
9	In the EditWorkoutsActivity, click on a workout in the list.	Application jumps to the details of that workout.	As expected	PASS
10	In the EditWorkoutsActivity, click on a workout in the list. Click the Add Exercise button and select an exercise from the list.	The selected exercise is added to the workout and displayed in the list.	As expected	PASS
11	In the EditWorkoutsActivity, click on a workout in the list. Click the Remove Exercise button and select an exercise from the list.	The selected exercise is removed from the workout and no longer displayed in the list.	As expected	PASS
12	In the EditWorkoutsActivity, click on a the Remove Workout button. Select a workout from the list dialog.	The selected workout is removed and no longer displayed in the list.	As expected	PASS

Select Workouts Activity Test Plan

Test number	Description	Expected result	Actual result	Pass or Fail
13	In the SelectWorkoutsActivity, select a workout.	A button appears at the bottom of the screen with the name of the workout	As expected	PASS
14	In the SelectWorkoutsActivity, select a workout. Click the button "Start..." on the bottom of the screen.	A list of exercises should appear for that workout.	As expected	PASS

Do Workouts Activity Test Plan

Test number	Description	Expected result	Actual result	Pass or Fail
15	In the DoWorkoutsActivity, select an exercise from the list that has not been performed before.	The application displays the exercise name selected with no information in the "Previous" fields. It also has no date in the "Last performed" field.	As expected	PASS
16	In the DoWorkoutsActivity, after completing step 1 enter some performance data and press "Save".	The data entered now appears in the previous fields. The "Last Performed" field displays the date and time "Save" was clicked.	As expected	PASS
17	After completing Step 2 press the rest button.	At time starts a count down for the rest period and the beeps	As expected	PASS

View Statistics Activity Test Plan

Test number	Description	Expected result	Actual result	Pass or Fail
18	After entering some workout data for a given exercise click that given exercise. Select the "View weight" radio button.	An average of the weight lifted for that exercise is displayed in a graph format	As expected	PASS
19	After entering some workout data for a given exercise click that given exercise. Select the "View reps" radio button.	An average of the reps performed for that exercise is displayed in a graph format	As expected	PASS
20	After entering some workout data for a given exercise click that given exercise. Select the "View work done" radio button.	An average of the work done (reps*weight) for that exercise is displayed in a graph format	As expected	PASS