Progressive Overload Mobile Application

Mobile Application For Weight Lifting

[Pick the date]

Office Black Edition - tum0r

User

Acknowledgements:

Abstract:

Contents

[Introduction: 4](#_Toc416201531)

[Aim: 4](#_Toc416201532)

[Originality: 5](#_Toc416201533)

[Choosing a platform: 6](#_Toc416201534)

[Research: 8](#_Toc416201535)

[Methodology: 9](#_Toc416201536)

[Development and results: 12](#_Toc416201537)

[Design: 12](#_Toc416201538)

[Database Design: 13](#_Toc416201539)

[Table Normalisation: 14](#_Toc416201540)

[Final Entities: 16](#_Toc416201541)

[RI Diagram: 17](#_Toc416201542)

[EAR Diagram: 17](#_Toc416201543)

[Gui Design: 18](#_Toc416201544)

[Android Fragments: 18](#_Toc416201545)

[Gui design and function: 21](#_Toc416201546)

[Title Activity: 21](#_Toc416201547)

[Edit Workouts: 22](#_Toc416201548)

[Edit Exercises: 23](#_Toc416201549)

[Select Workout: 25](#_Toc416201550)

[Do Workout: 27](#_Toc416201551)

[View Statistics: 28](#_Toc416201552)

[Class structure design: 29](#_Toc416201553)

[Implementation: 31](#_Toc416201554)

[Using Eclipse ADT: 31](#_Toc416201555)

[Android logcat: 31](#_Toc416201556)

[Activity Intercommunication: 32](#_Toc416201557)

[Explicit intent: 32](#_Toc416201558)

[Implicit intent: 32](#_Toc416201559)

[Creating, accessing and displaying the database: 34](#_Toc416201560)

[Creating the database: 34](#_Toc416201561)

[Accessing and displaying the database: 35](#_Toc416201562)

[Displaying data in a ListView: 35](#_Toc416201563)

[Displaying a single piece of data: 36](#_Toc416201564)

[Dialogs: 37](#_Toc416201565)

[Image Acquisition and Conditioning: 38](#_Toc416201566)

[Custom Graph View: 39](#_Toc416201567)

[Problems Encountered: 40](#_Toc416201568)

[Testing: 41](#_Toc416201569)

[Functional Test: 41](#_Toc416201570)

[Real World Test: 42](#_Toc416201571)

[Future Work: 43](#_Toc416201572)

[Evaluation: 44](#_Toc416201573)

[Conclusion: 45](#_Toc416201574)

[Recommendations and future work: 46](#_Toc416201575)

[References and bibliography: 47](#_Toc416201576)

[Appendices: 48](#_Toc416201577)

# 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, one must strive to increase the load placed on one’s body in order to stimulate muscle growth. In practice this means having a set workout and increasing each exercise in some way, be it number of reps, weight or time between sets. 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 and 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.

## 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.

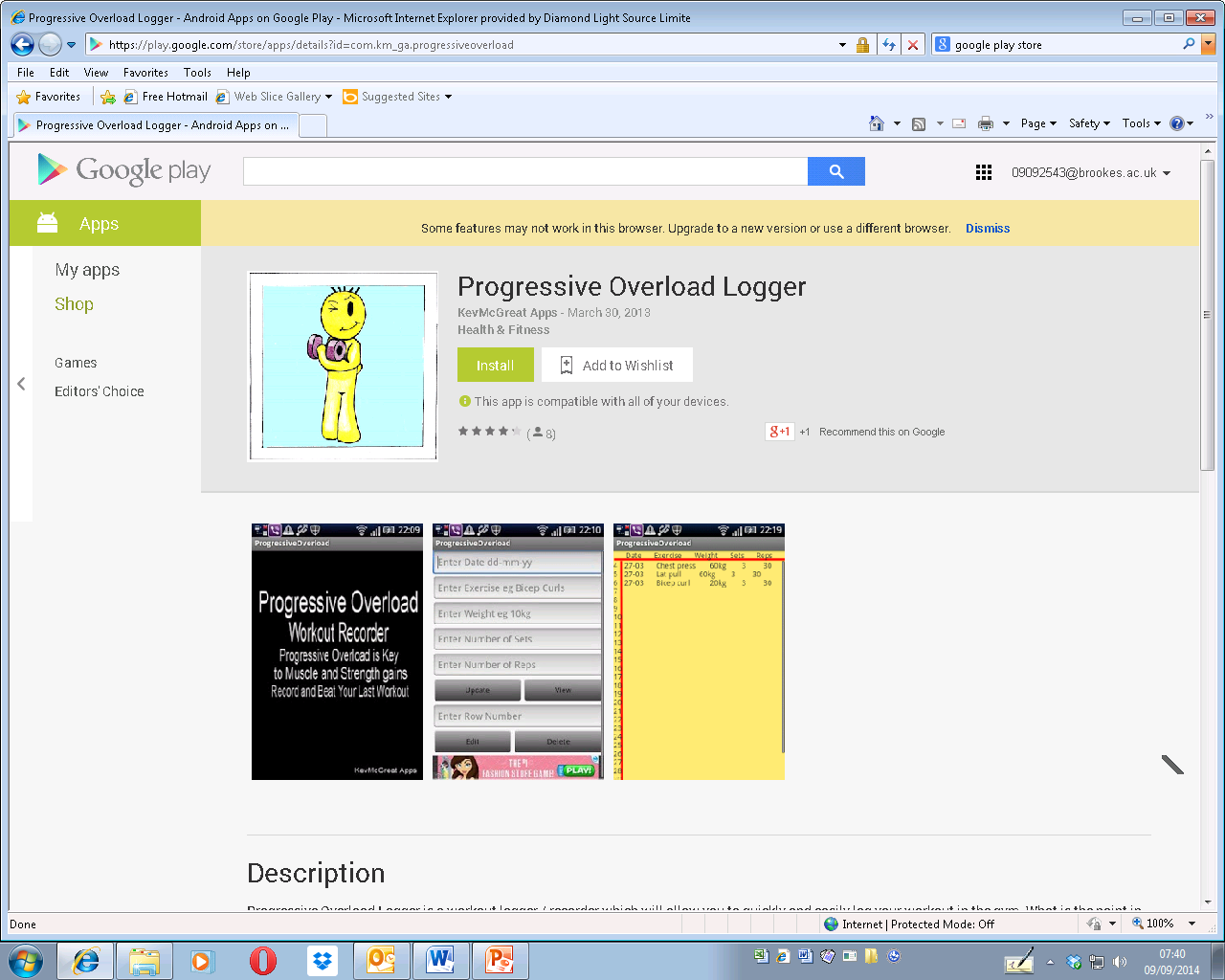
In terms of implementing the above on a mobile device there are the following key issues to be considered:

1. What platform should this be developed on?
   1. The main choices are Android, IOS and windows mobile.
2. Battery usage
   1. 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
   1. 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
   1. The data entered into the application must be retained on power off. The data need to either 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
   1. Most mobile devices allow both portrait and landscape views, this has to be considered.
6. Graphical representation of performance
   1. 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
   1. The application must be easy to use and surprise the user with features such as customisation and nice graphics.

## Originality:

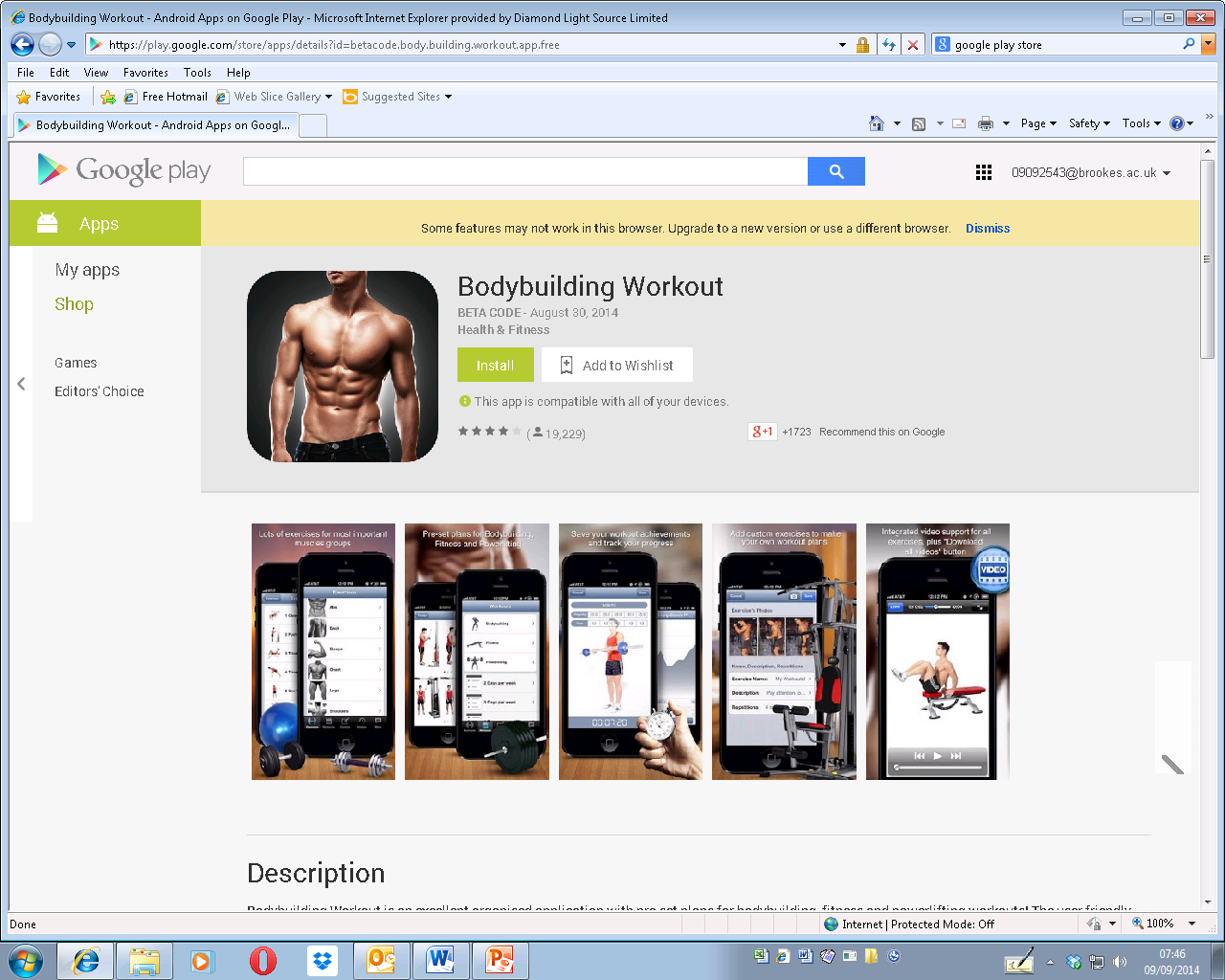
Currently there are a few applications available for workouts but none that address the issues of progressive overload effectively. The following applications are available for android:

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 it 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
   1. The more popular it is amoungst its users the more exposure the application will get.
2. Ease of developing
   1. The easier it is to code the application the less the application costs to produce.
3. Ease of publication
   1. How easy does the platform make it to get the application on the market?
4. Longevity of the operating system
   1. 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 my iOS having 11.7% and then Windows Phone having 2.9%. From a popularity point of view this is the obvious choice. From now on Windows Phone won’t be mentioned as it’s 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 if 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:

# 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:
   1. Complete the requirement specification
2. Design
   1. Decide on the classes to be used
   2. Decide what these classes will do and how they will interact
   3. Compose a UML diagram of the system
   4. Compose and normalise the database
   5. Design the user interface
3. Implementation
   1. Create android activities
   2. Implement GUIs for each activity
   3. Implement communication between activities
   4. Implement database
   5. Implement custom graph view
   6. Implement timer option
   7. Implement any audio
4. Verification
   1. Create a test schedule
   2. Test the mechanics of the system
   3. 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.
   1. 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.
   1. 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.
   1. The database will be normalised to BCNF.
4. The application must work in both landscape and portrait orientations.
   1. This it catered for in the android framework. Separate GUI layout files are used for portrait and landscape.
   2. 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.
   1. 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
   1. The GUI programming was initialled seen as a possible source for unforeseen complexity. As further work has been done and GUIs are now understood well this is only a small risk.
   2. 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.
   1. 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
   1. 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
   1. 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.
   2. 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
   1. 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:

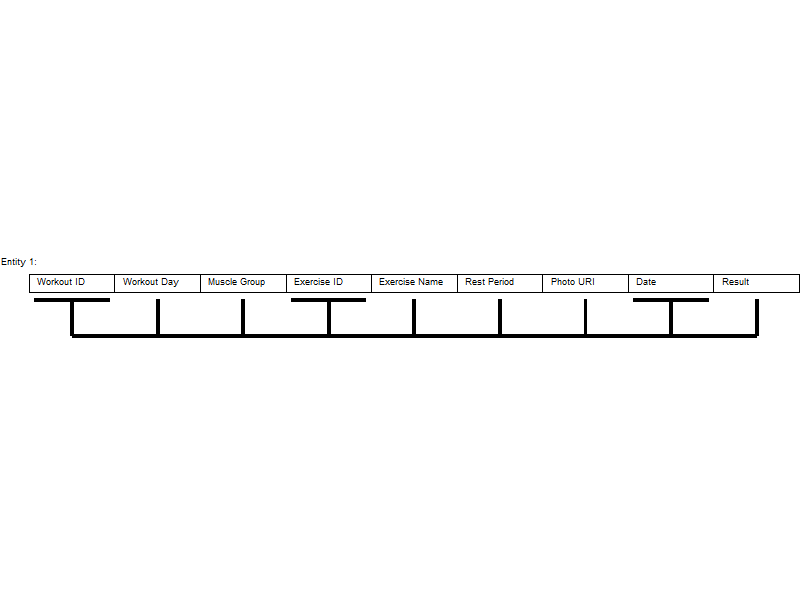
1. Workout
   1. Day
   2. Muscle Group
   3. Exercises in that workout
2. Exercise
   1. Name
   2. Rest period
   3. Photo URI
3. Exercise Instance
   1. Exercise name
   2. Date
   3. Result

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.

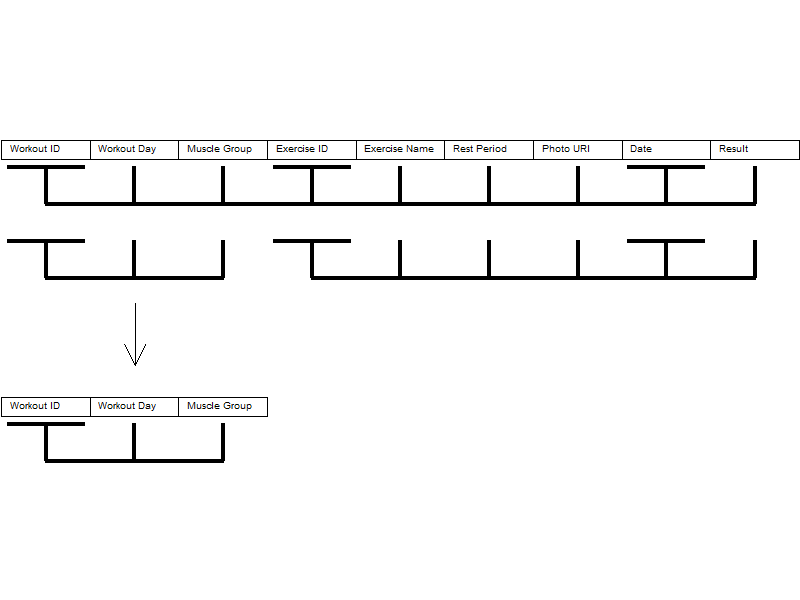
#### Table Normalisation:

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



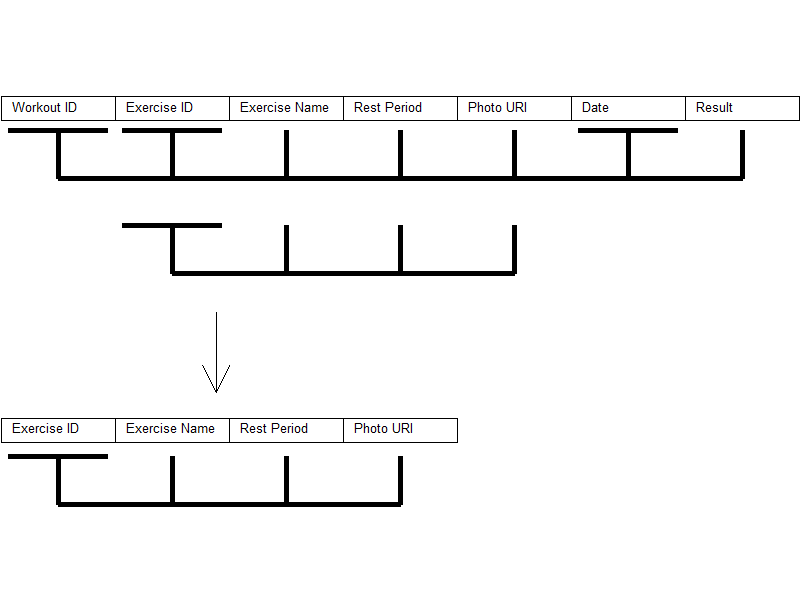
Entity 1. From the workout ID, exercise ID and the date it is possible to determine all other entries.

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



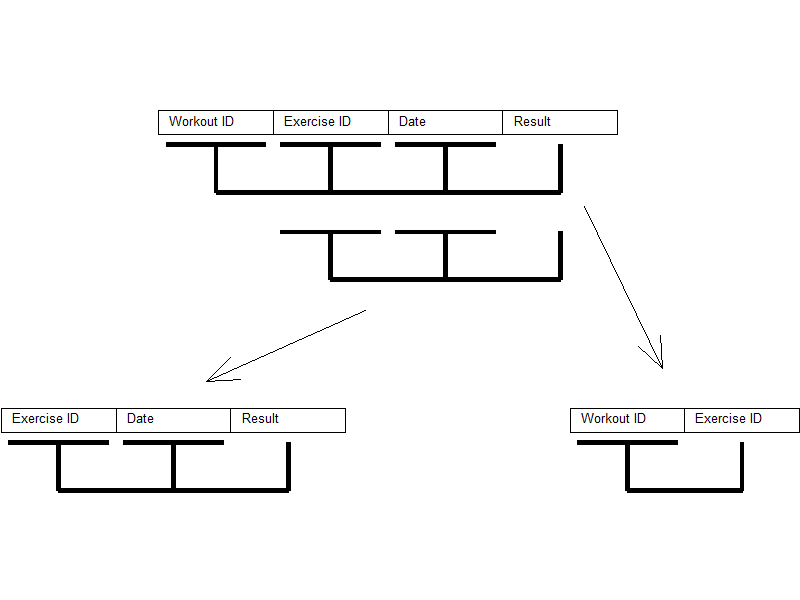
From this initial normalisation we now have an entity in BCNF. This will be the “Workout” entity.

The remaining table still needs further normalisation:

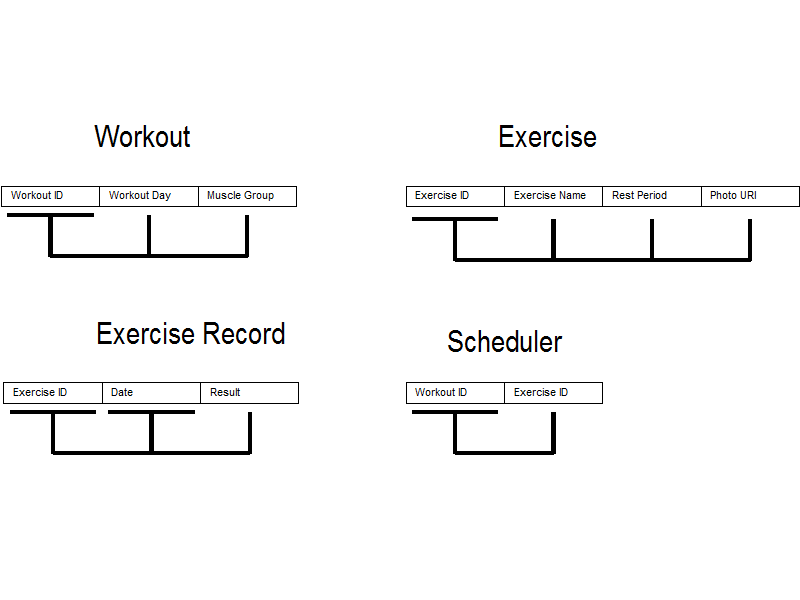


We now have another entity normalised to BCNF, this will be called the “Exercise” entity.

The remaining table still needs further normalisation:

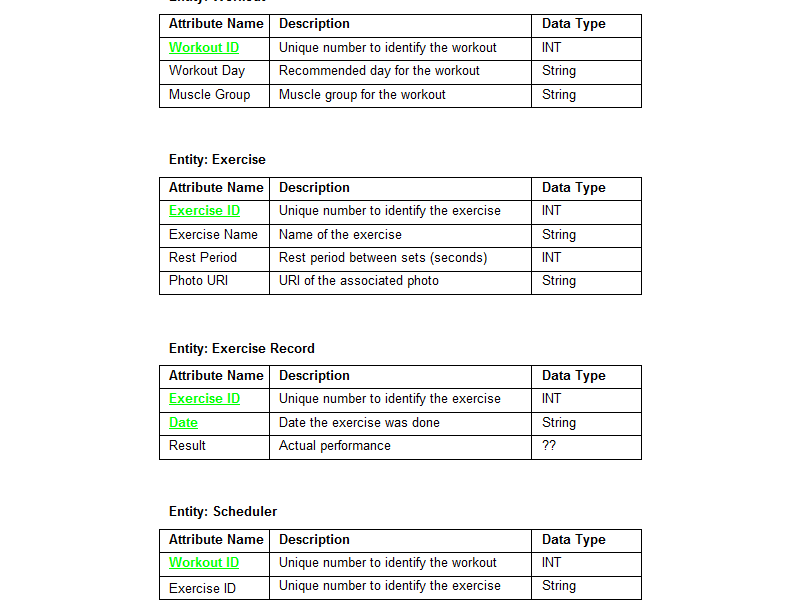


With the above process of normalisation we can now describe all entities in the database:



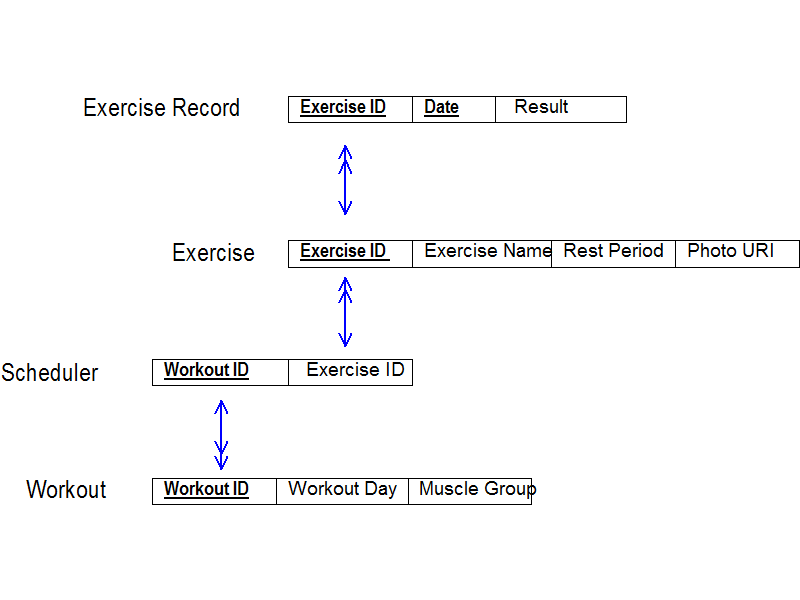
With the above entities we are able to avoid all update, insert and delete anomalies. All entities are in BCNF. The “Scheduler” entity will act as a linker between the “Workout” and “Exercise” entities and will have a many to one relationship with each to avoid them having a many to many relationship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

#### Final Entities:

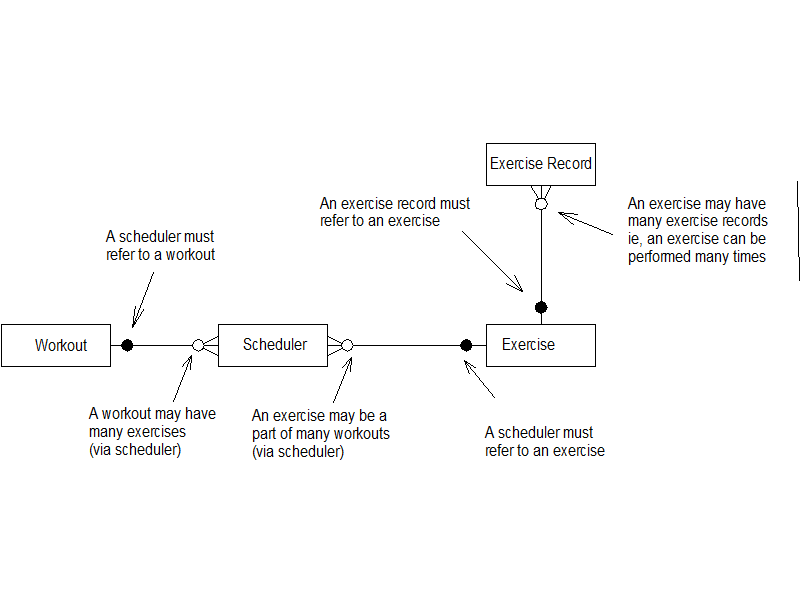


The above entities are a preliminary specification of the database. There are still a few unknown factors as the operation of the android content provider is still not fully understood. In the “Exercise Record” entity it can be seen that the “Result” element has not been defined yet. This element must represent performance in terms of the number of sets completed, the number of reps on each set and the rest period after each set. This could be realised with multiple string elements but there may be a better way of doing this, hence further research and development is needed before this can be finalised.

#### RI Diagram:



#### EAR Diagram:



### Gui Design:

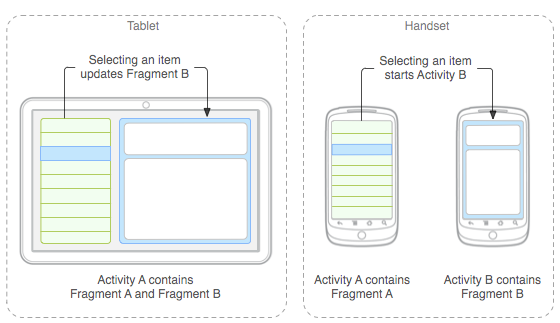
Using a CAD package the GUIs can be designed as a starting point. This will help to clarify the behaviour of the application. As a starting point we can define the base activities and the flow between the:

INSERT VISO DRAWING HERE!!

#### 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:



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 file 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 “editWorkoutDetailFrag” will be used if the device is in landscape.

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



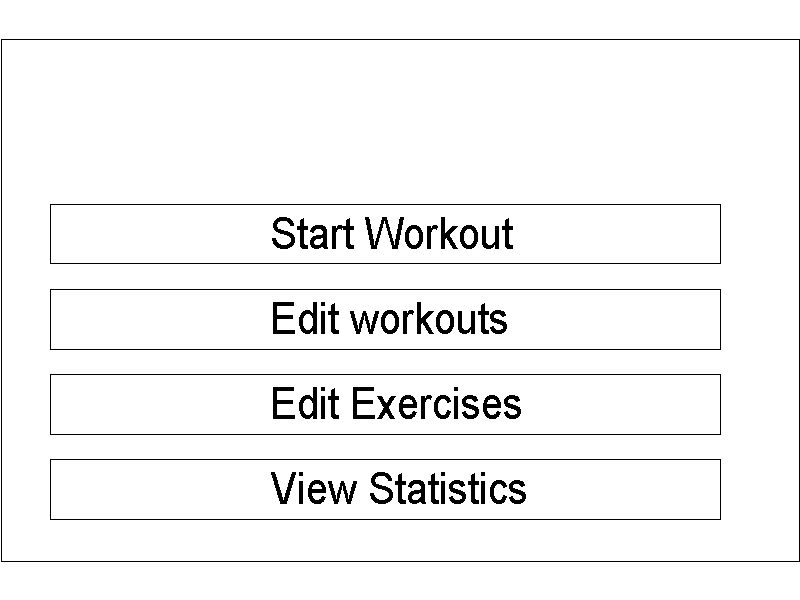
Whilst in landscape mode a single activity takes care of both fragments, however when the device is in portrait there needs to be an activity for each GUI. When an action causes the second GUI to be needed a second activity is started:



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:

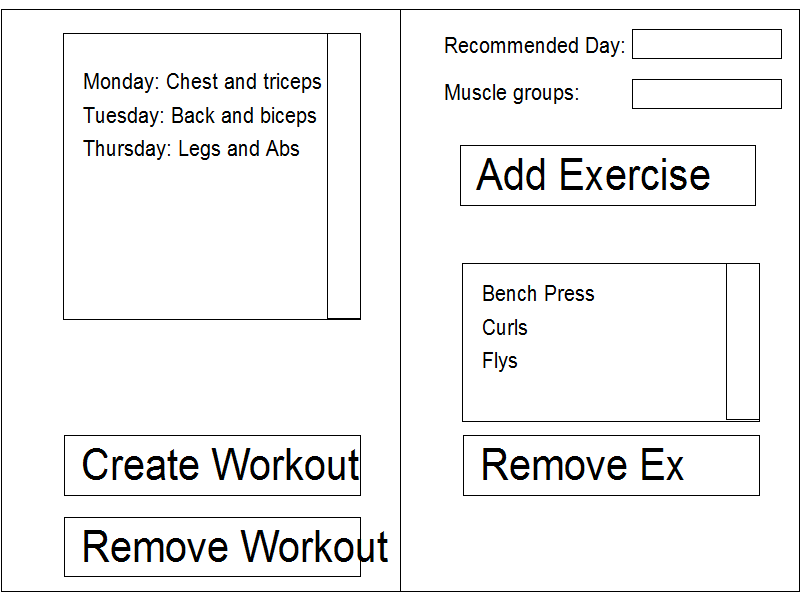
##### Title Activity:

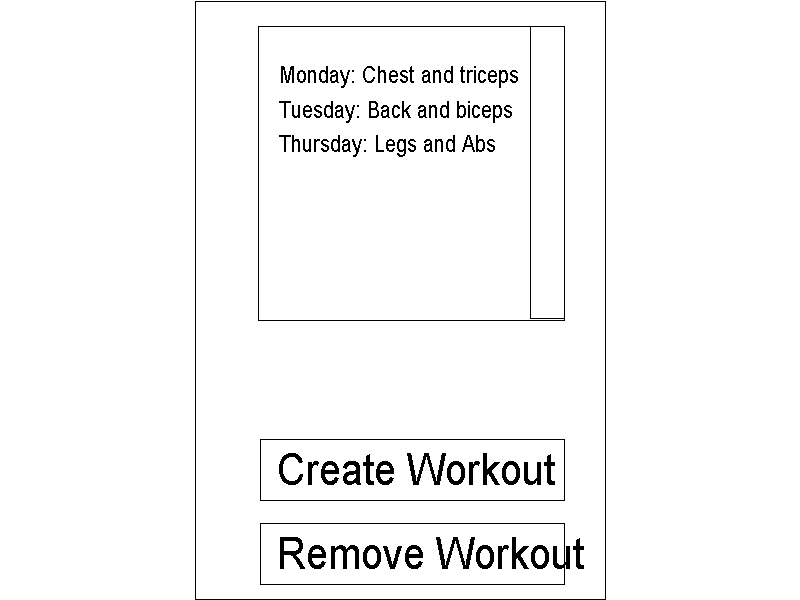


The title activity will be the first activity the user will come into contact with. It has a fairly simple GUI that will allow the user to access the rest of the application. Due to its simplicity the GUI can remain the same for both portrait and landscape. Each button on this GUI just launches the described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

##### Edit Workouts:

EditWorkoutsListFragment | EditWorkoutsDetailFragment





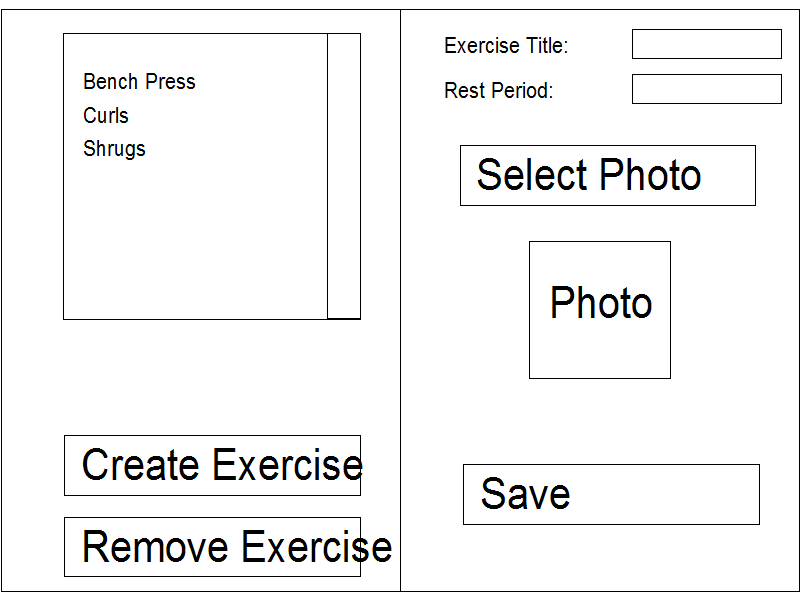
The edit workouts activity will use 2 fragments. These are the “EditWorkoutsListFragment” and the “EditWorkoutsDetailFragment”. Depending on the device used they will be displayed as above.

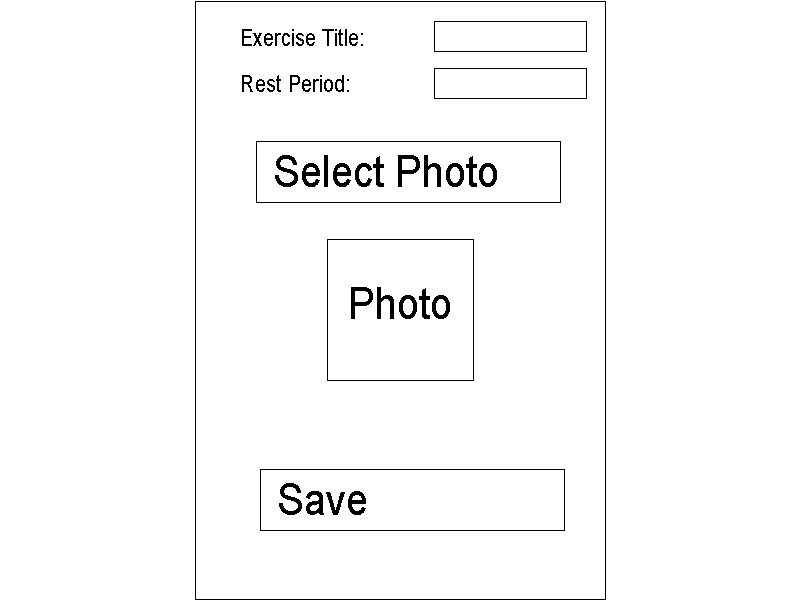
The edit workouts activity allows the user to do the following:

1. View all workouts
   1. 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 editied.
2. Edit any workout
   1. 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.
      1. 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
   1. 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
   1. Pressing the “Create Workout” button will invoke a dialog to aid creating a new exercise.

##### Edit Exercises:

EditExercisesListFragment | EditExerciseDetailFragment





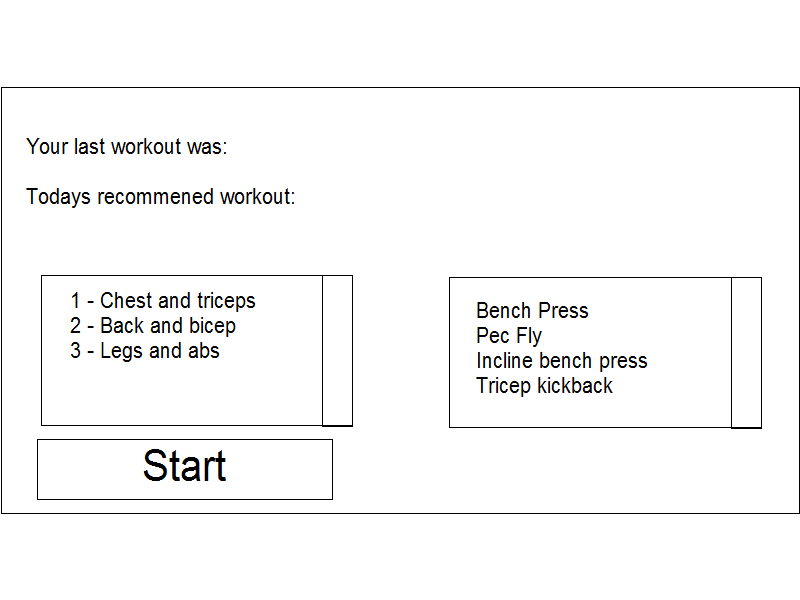
The edit exercise activity will use 2 fragments. These are the “EditExerciseListFragment” and the “EditExerciseDetailFragment”. Depending on the device used they will be displayed as above.

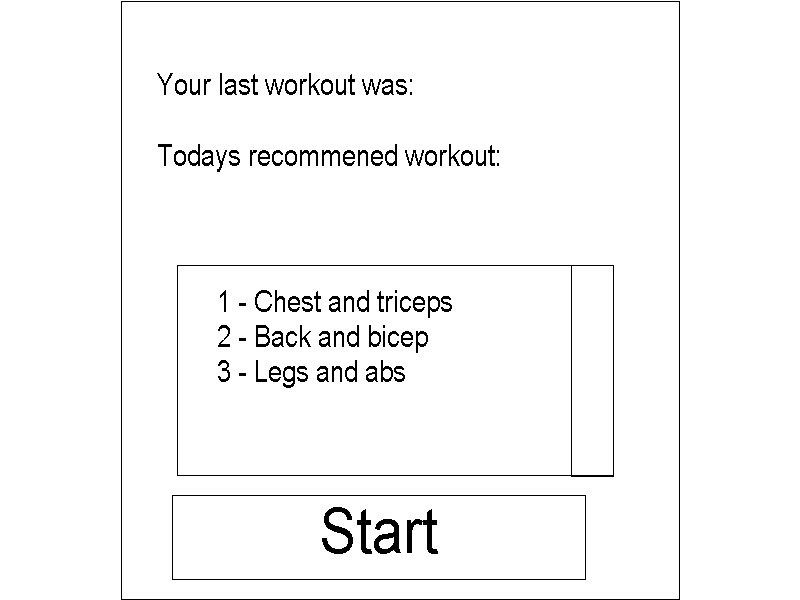
The edit exercise activity allows the user to do the following:

1. View all Exercises
   1. Exercises will be displayed in a list view. The list view will display all exercises entities in the database. This list view will also respond to clicks on any of its items to allow the exercise to be editied.
2. Edit any exercise
   1. By clicking on a 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.
      1. 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
   1. 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
   1. By pressing the “Remove exercise button” the user can remove a exercise. A list dialog with all exercises is displayed and once an exercise is clicked on it is removed.

##### Select Workout:

SelectWorkoutListFragment | SelectWorkoutDetailFragment



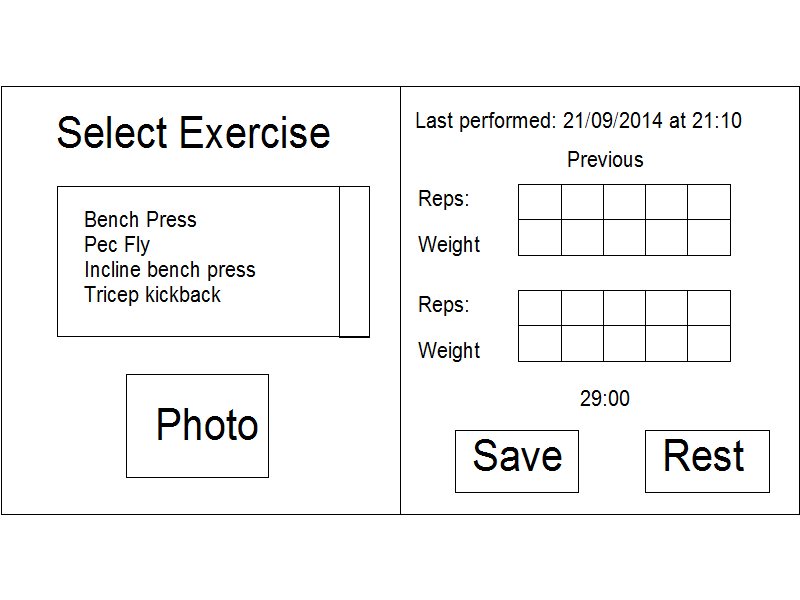


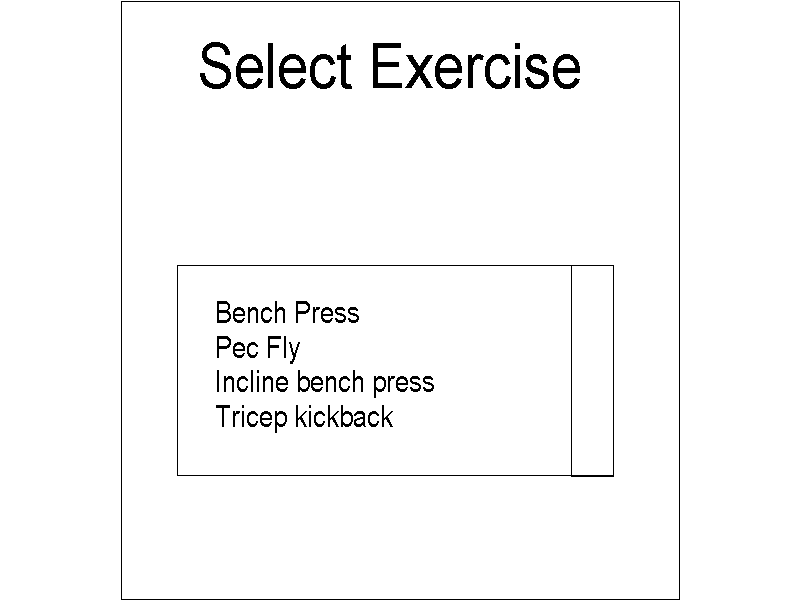
The select workout activity will use two fragments but unlike previous activities it will only use one when in portrait mode. The tablet user will benefit from being able to view more information.

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.

##### Do Workout:

DoWorkoutListFragment | DoWorkoutDetailFragment





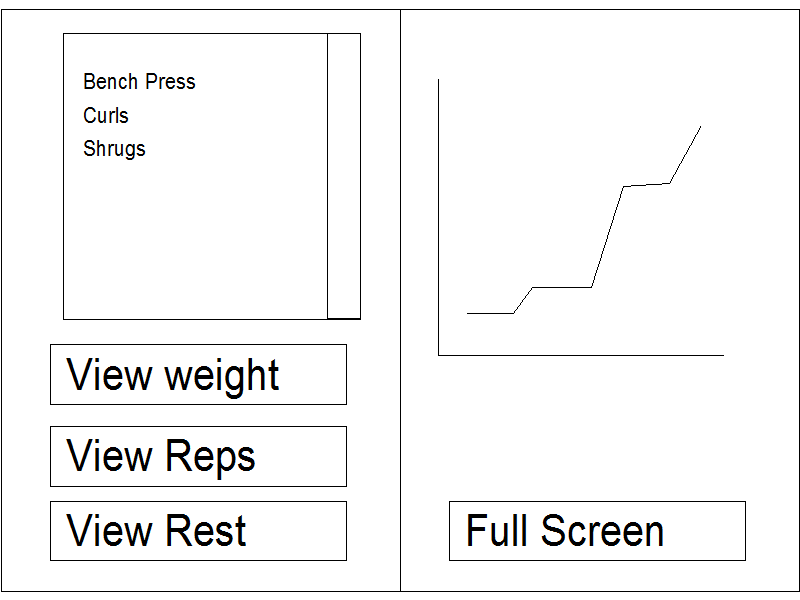
The do workout activity will use 2 fragments. These are the “DoWorkoutListFragment” and the “DoWorkoutDetailFragment”. Depending on the device used they will be displayed as above.

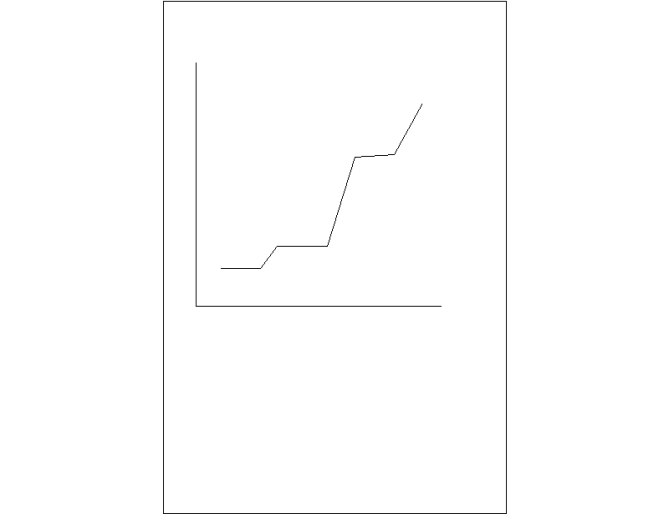
This purpose of this activity is to allow the user to record their performance for a given workout. It will allow the user to do the following:

1. Select an exercise
   1. 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
   1. When an exercise has been selected the detail fragment will display when the exercise was last performed and the 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
   1. 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
   1. Clicking the “Rest” button will start a timer that will make a sound when the timer is complete.

##### View Statistics:

ViewStatisticsListFragment | ViewStatisticsDetailFragment





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.



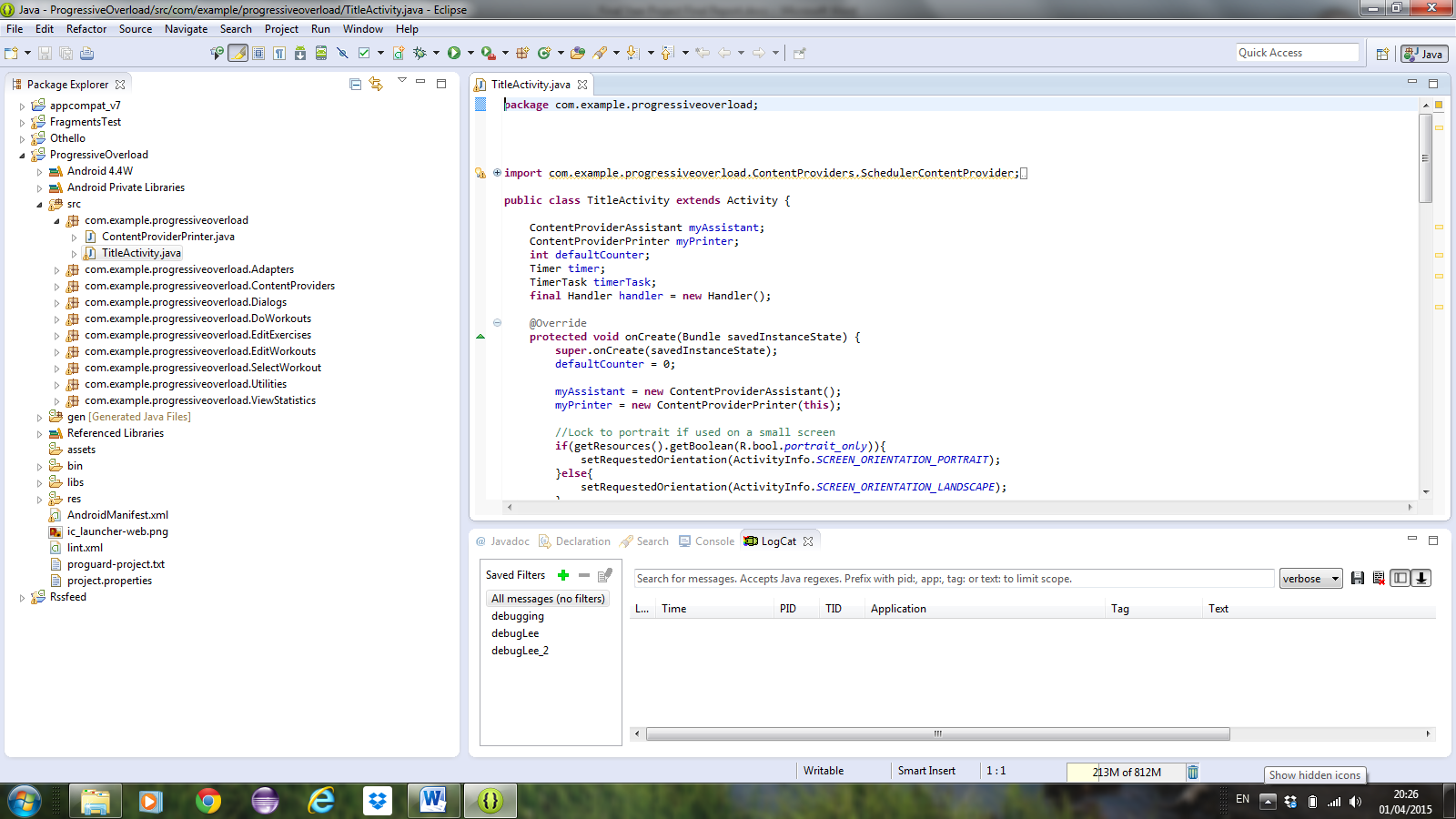
The purpose of the main activity (EditWorkouts) and the portrait activity have been described on page 20. In addition to these we have the following classes which serve the following purposes:

1. DisplayPopulator (EditWorkoutsListPopulator)
   1. 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 case 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)
   1. 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)
   1. This class takes care of the dialog ivocation when for example creating a new exercise. This is detailed later on in the “Dialogs” section.
4. ContentProviderAssistant
   1. This is a basic class used to perform basic operations on the content provider. Common operation too be performed on content providers are done here to avoid code duplication.
5. CursorSelector
   1. 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 and APK file but also allows online debugging.



3

2

1

1. Project View
   1. This contains all the source code and resources.
2. Code window
   1. This displays the source of the selected file
3. Console
   1. 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 elipse, it’s all quite familiar to anyone who has used it.

#### Android logcat:

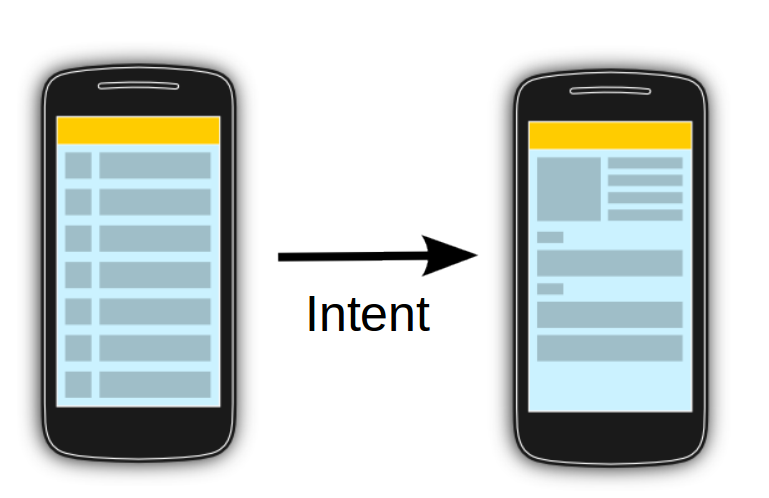
Android has a class called “logcat” that can be very useful with debugging and works extremely well with eclipse. Just like a print statement one can print to the logcat providing a message and a tag. Eclipse can then filter logcat messaged depending on the tag name given:



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 should start, 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”.

vogella

#### Explicit intent:

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



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, explicit 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:



I discovered whilst attempting to access images on an android device that they 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:



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 out 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 stand alone SQL database. The main reason for this was future proofing the application. The main difference between using a content provider and a stand alone 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”:



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



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:



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 page 20). 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:



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



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 exerciseListAdapter. This is a relatively simple class to map the data over to a list view:



The newView() 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.

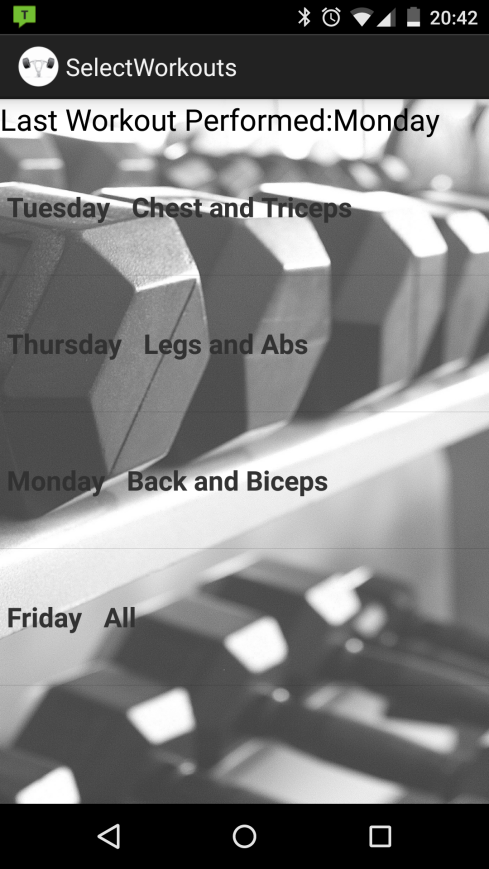
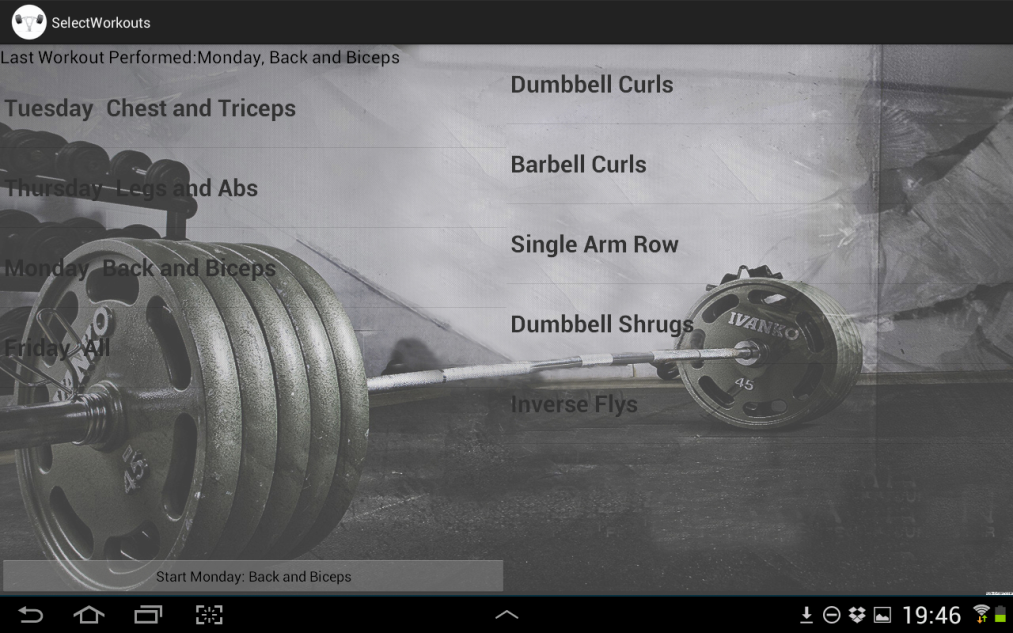
The bindView() method is the method that takes the data from the Cursor and writes it to the TextView in the layout file. This method is called for every entry in the Cursor.

##### 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:



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 colomn numbers for the workout day and workout muscle group colomns. 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:

### Image Acquisition and Conditioning:

### Custom Graph View:

### Problems Encountered:

## Testing:

### Functional Test:

### Real World Test:

## Future Work:

# Evaluation:

# Conclusion:

# Recommendations and future work:

# References and bibliography:

http://www.idc.com/prodserv/smartphone-os-market-share.jsp

# Appendices: