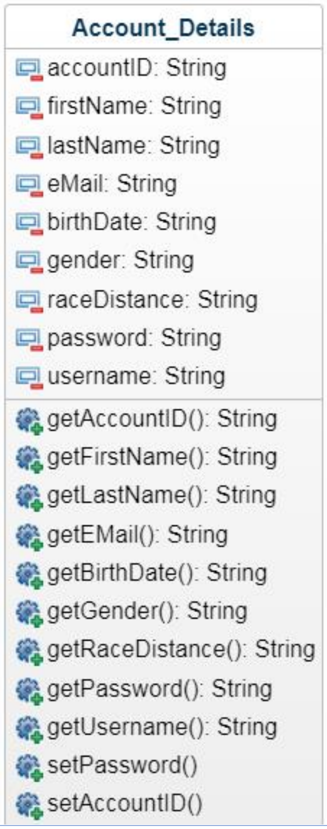
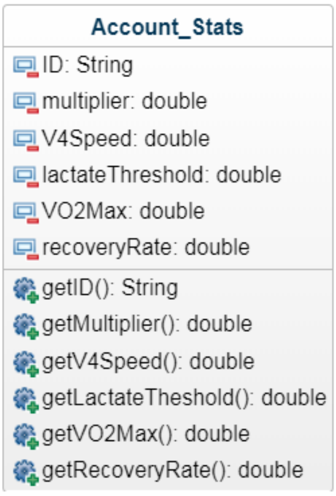
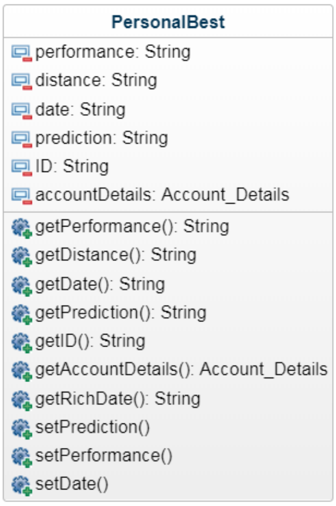
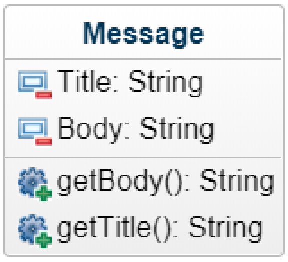
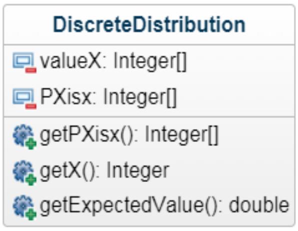
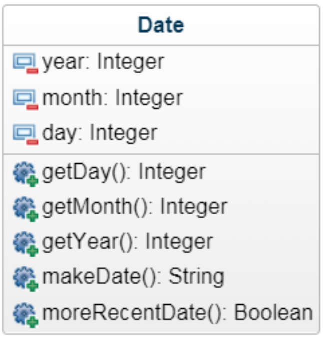
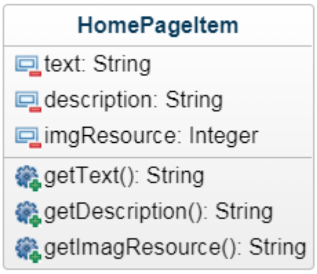
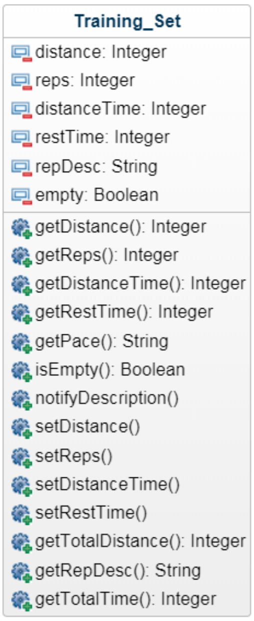
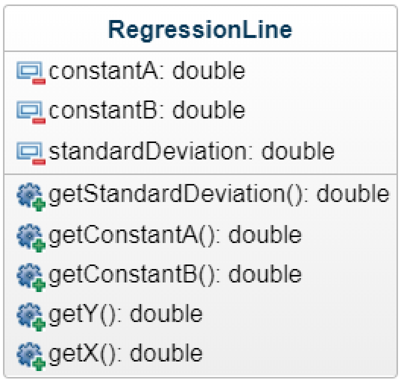
Structures

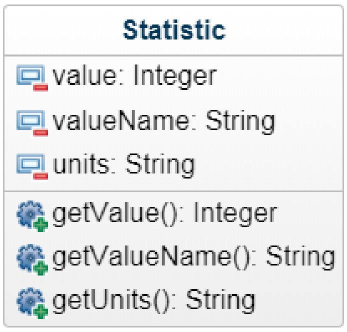
Class Structures

In android studio coding in Java is impossible without using class structures. I will use classes to structure data retrieved from the database and package data to be sent from activity to activity. Having a variety of class types allows me to use the *instanceof* operand in *IF* statements to perform different actions on different types of objects and parse the objects to their defined type. All the class structures will be fully encapsulated, with private members and corresponding public methods.

Class Diagram	Description
 Account_Details accountID: String firstName: String lastName: String eMail: String birthDate: String gender: String raceDistance: String password: String username: String getAccountID(): String getFirstName(): String getLastName(): String getEMail(): String getBirthDate(): String getGender(): String getRaceDistance(): String getPassword(): String getUsername(): String setPassword() setAccountID()	- Object that stores the details of an account. All members are private, with public functions to return their values. So fully encapsulated. - Object will be stored on the database in the *AccountDetails* table. - Class will implement the android *parcelable* interface, so the *Account_Details* objects can be passed from one activity to another. - Only 2 “Setter” methods because when the details are updated a new object will be instantiated and sent to the database and updated using the *AccountID*. - The *setPassword* method is used for editing the password and then sending the object to the database to be updated rather than creating a new instance.
 Account_Stats ID: String multiplier: double V4Speed: double lactateThreshold: double VO2Max: double recoveryRate: double getID(): String getMultiplier(): double getV4Speed(): double getLactateTheshold(): double getVO2Max(): double getRecoveryRate(): double	- Object that stores the statistics of an account which are used in the training generator. All members are private, with public functions to return their values. So fully encapsulated - Object will be stored in the *GeneratorTable* in the dynamic database, initial instances of this object will be stored in the *Stats* table of the static database. - Each *Account_Stats* object relates to exactly one *Account_Details* object. These are separate objects because *Account_Stats* objects are only used in the training generator activity, so it would be unnecessary to pass these values from activity to activity.

	- This object therefore will not implement the android *parcelable* interface.
 PersonalBest performance: String distance: String date: String prediction: String ID: String accountDetails: Account_Details getPerformance(): String getDistance(): String getDate(): String getPrediction(): String getID(): String getAccountDetails(): Account_Details getRichDate(): String setPrediction() setPerformance() setDate()	- Object that stores the details of a personal best. All members are private, with public functions to return their values. So fully encapsulated. - One account can have multiple *PersonalBest* objects so the *accountDetails* member is effectively used as a foreign key. - Will not extend (inherit) from *Race* because does not necessarily directly link to a *Race* in the database. The object also stores a prediction. - "Setter" methods are used to set members of the *PersonalBest* object, so rather than creating a new instance the object can be edited, and the layout refreshed.
 Message Title: String Body: String getBody(): String getTitle(): String	- Basic object that stores 2 Strings and public methods to return them. - Class is fully encapsulated. - Object used to display a simple message with a title and body of text in the recycler view layout.
 DiscreteDistribution valueX: Integer[] PXisx: Integer[] getPXisx(): Integer[] getX(): Integer getExpectedValue(): double	- Object that defines a discrete probability distribution. - Stores 2 arrays of equal length, *valueX* stores the values the variable x can take. Or values that can be returned from the *getX()* method. *PXisx* stores the probability that the corresponding value of x will be returned by the *getX()* function. - The *getExpectedValue* method returns the mean of the probability function.
 Date year: Integer month: Integer day: Integer getDay(): Integer getMonth(): Integer getYear(): Integer makeDate(): String moreRecentDate(): Boolean	- Simple object mainly for formatting strings into date format. - Will have a constructor that takes a string value and will instantiate a *Date* object if it is the valid *"dd/mm/yyyy"* format. Second constructor will take the values of year, month and day. - the *moreRecentDate()* method will compare itself to another instance of *Date* and return true if it is the more recent date.

	- The *makeDate()* method will construct and return a string in the format *“dd/mm/yyyy”*
 HomePageItem text: String description: String imgResource: Integer getText(): String getDescription(): String getImagResource(): String	- Simple object similar to a *Message* that stores 2 strings, and a Resource ID (Integer that corresponds to an item in the resources directory of the project) - Fully encapsulated with public methods to return its values. - Used to display information and an icon in the home page fragment’s recycler view layout.
 Training_Set distance: Integer reps: Integer distanceTime: Integer restTime: Integer repDesc: String empty: Boolean getDistance(): Integer getReps(): Integer getDistanceTime(): Integer getRestTime(): Integer getPace(): String isEmpty(): Boolean notifyDescription() setDistance() setReps() setDistanceTime() setRestTime() getTotalDistance(): Integer getRepDesc(): String getTotalTime(): Integer	- Object used for storing quantitative information about a training set within a training program. Quantitative information can be used in mathematical models as oppose to the string description which is stored on the database. - When manually recording, or inputting a completed or active training, instances of this object are created given the user inputs. The descriptions are then compiled into a single description for the training program to be stored on the database. - When editing a *Completed_Training* object, the string description from the database will be converted to instances of *Training_Set* objects, for editing and mathematical purposes. - Instances of this object will be created in the training generator activity so the quantitative information can be used for optimising the training program.
 RegressionLine constantA: double constantB: double standardDeviation: double getStandardDeviation(): double getConstantA(): double getConstantB(): double getY(): double getX(): double	- Object that defines a linear regression line in the form $y = bx + a$. - When the object is instantiated the value for a, b and the standard deviation are calculated from 2 arrays of real numbers (doubles). - the *getY()* and *getX()* methods return the corresponding value of y or x, given a value for x and y respectively using the regression line formula.



- Simple object that stores an integer value, and 2 strings.
- Integer value will be a statistic such as distance run calculated from a query in the database.
- Class is fully encapsulated.
- The 2 string members give the integer value context when it is displayed.
- This object will be displayed in the *AboutYou* fragment layout's recycler view.

Figure 1 describes the *AthleteLactateModel* class that will be used in the *Training Generator* activity for optimising trainings to the athlete's ability. It will also be used for optimising the athlete's stats when a *Completed_Training* object has been recorded or updated. The class has 3 private classes within it: *ExponentialModel*, *LinearModel* and *Point.* These classes are only used within the *AthleteLactateModel* class for calculating the average lactate of an athlete during a training program and for optimising their account stats accordingly. *lactateArray* is an array of doubles that correspond to the lactate level of the athlete for each second during the training. *lactateCurve* is an instance of the *ExponentialModel* class and is instantiated using the athlete's *V4Speed* and *VO2Max*. The other stats *RecoveryRate* and *LactateThreshold* are used to model the lactate levels throughout the training. *runDistance()* and *rest()* are private methods used by the *avgLactateForTraining()* method to model the athlete running and resting respectively. The average can then be calculated from the *lactateArray*.

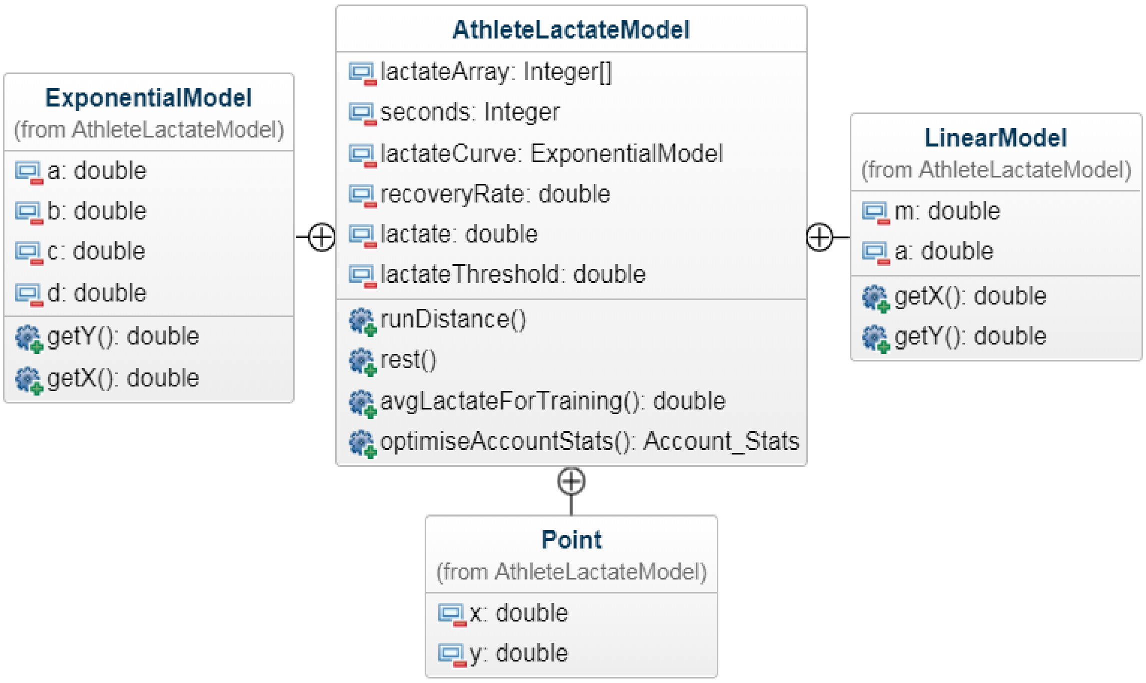


Figure 1 - AthleteLactateModel Class Diagram

Figure 2 describes the relationship between 3 separate objects: *Race, Active_Training* and *Completed_Training*. Firstly, *run_Activity* is the is the super class of these objects and it stores information about an activity: completed, description, name, date, ID and account details. All these members are protected and can be accessed by the subclasses but not outside the class. The object *Race* directly inherits from *run_Activity* and stores 2 more details: performance and distance. It also has a method that returns a *PersonalBest* object using its own values. *Training* is another subclass of *run_Activity*, but is not an object that is stored in the database. It defines the enumeration of *TrainingType* and stores a protected instance of this enumeration (*trainingType*). *Completed_Training* is an object that has all its values directly stored in the database. The class defines the enumeration of *Difficulty* and stores a private instance of this enumeration. *Active_Training* is also an object that is directly stored in the database and instantiated from the database. *totalTime* and *totalDistance* are not stored in the database and are calculated when the object is instantiated using the *makeTotalTime()* and *makeTotalDistance()* methods respectively. These classes will implement the android *parcelable* interface, so instances of these classes can be passed between activities. This allows the objects to be edited in a separate activity.

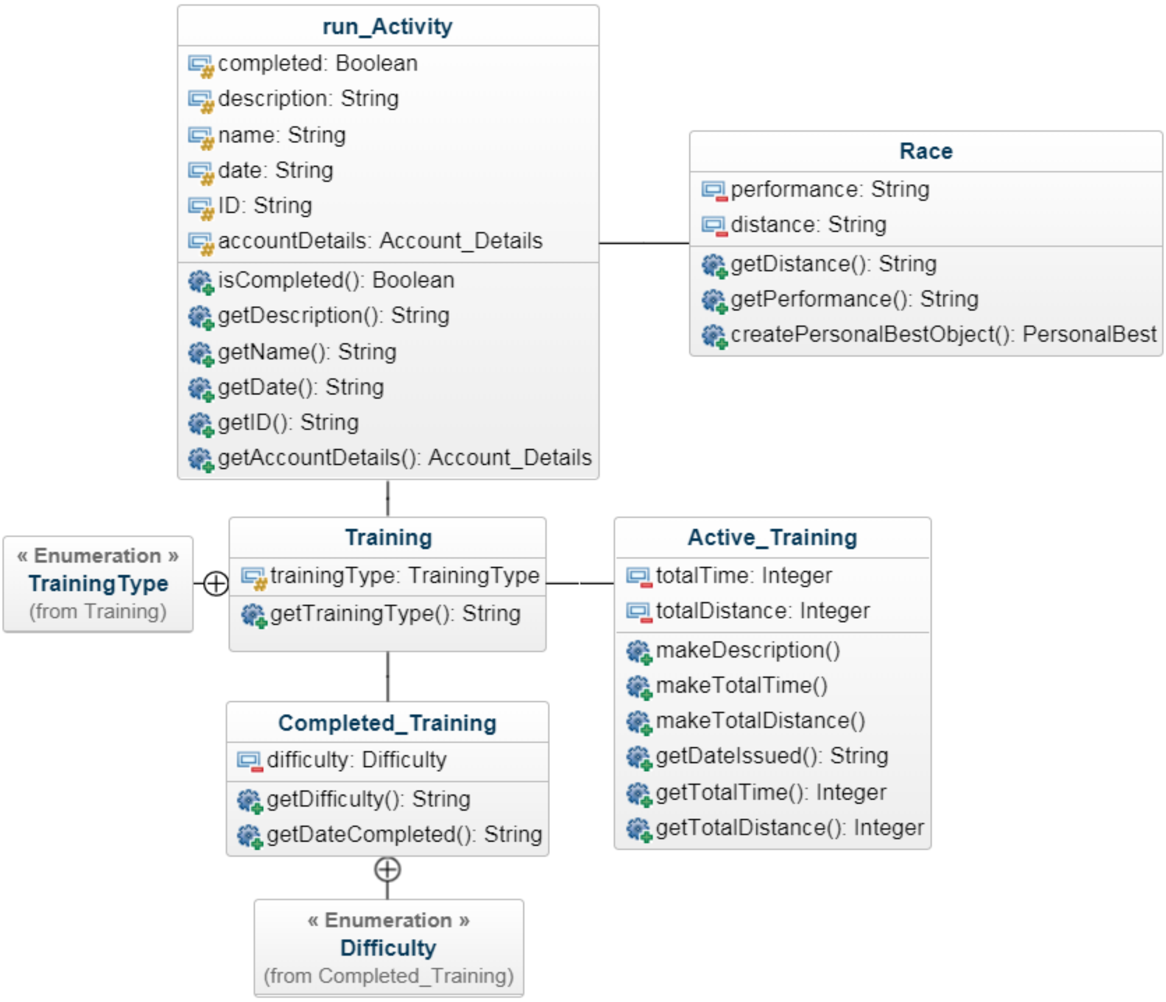


Figure 2 - run_Activity Inheritance Diagram

Activity Diagram

An android application makes use of activities to give the application functionality. In android studio an activity represents a single screen with a user interface similar to a window or frame. Java.Android activity is the subclass of ContextThemeWrapper class and each activity created in the application is the subclass of Java.Android activity. The layout of the user interface for the activity is described in a layout xml file. The interface elements can be accessed and given functionality in the *onCreate()* method in the activity's Java file.

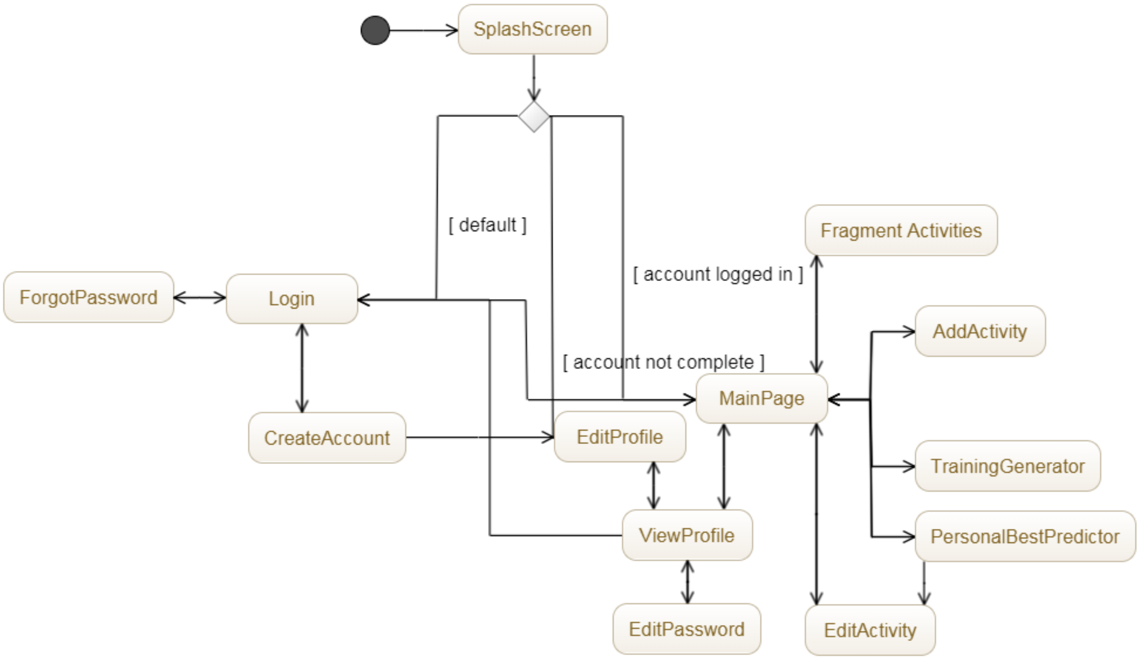


Figure 3 - System Navigation Diagram

Figure 3 describes the structure of the software and how all the activities will link to each other. The *SplashScreen* activity is the launcher activity (activity which is instantiated and loaded when starting the application). From the *SplashScreen* activity the system will check the database to see if there is an account logged in to the system or not. If there is an account logged it in, the account's ID is found from the database and the *MainPage* activity is loaded, otherwise the Login activity is loaded. In the rare occasion the account is not complete (just been created) the *EditProfile* activity is loaded. From the Login activity we can navigate to the *ForgotPassword* or *CreateAccount* activity by pressing the corresponding *TextView* widgets. We can login or create an account to navigate to the *MainPage.* If we create an account we are presented with the *EditProfile* activity, we can then navigate via the *ViewProfile* activity to reach the *MainPage* activity. In the *MainPage* activity one of 4 fragment activities or layout can be loaded. The user will be able to use a navigation drawer (slide-out menu) to change the fragment layout displayed in the *MainPage* activity. The user will also be able to navigate to the *ViewProfile, TrainingGenerator* and *PersonalBestPredictor* activities via the navigation drawer (slide-out menu). The *AddActivity* activity will be accessed from floating action button displayed in the *MainPage* activity. The *EditActivity* activity will be accessed from requesting to edit an object displayed in the recycler view layout within one of the fragment layouts displayed in the *MainPage* activity or in the *PersonalBestPredictor* activity. The *EditPassword* activity can be accessed from the *ViewProfile* activity with a button in the action bar of the activity, the user can also log out from this activity.

Database Structures

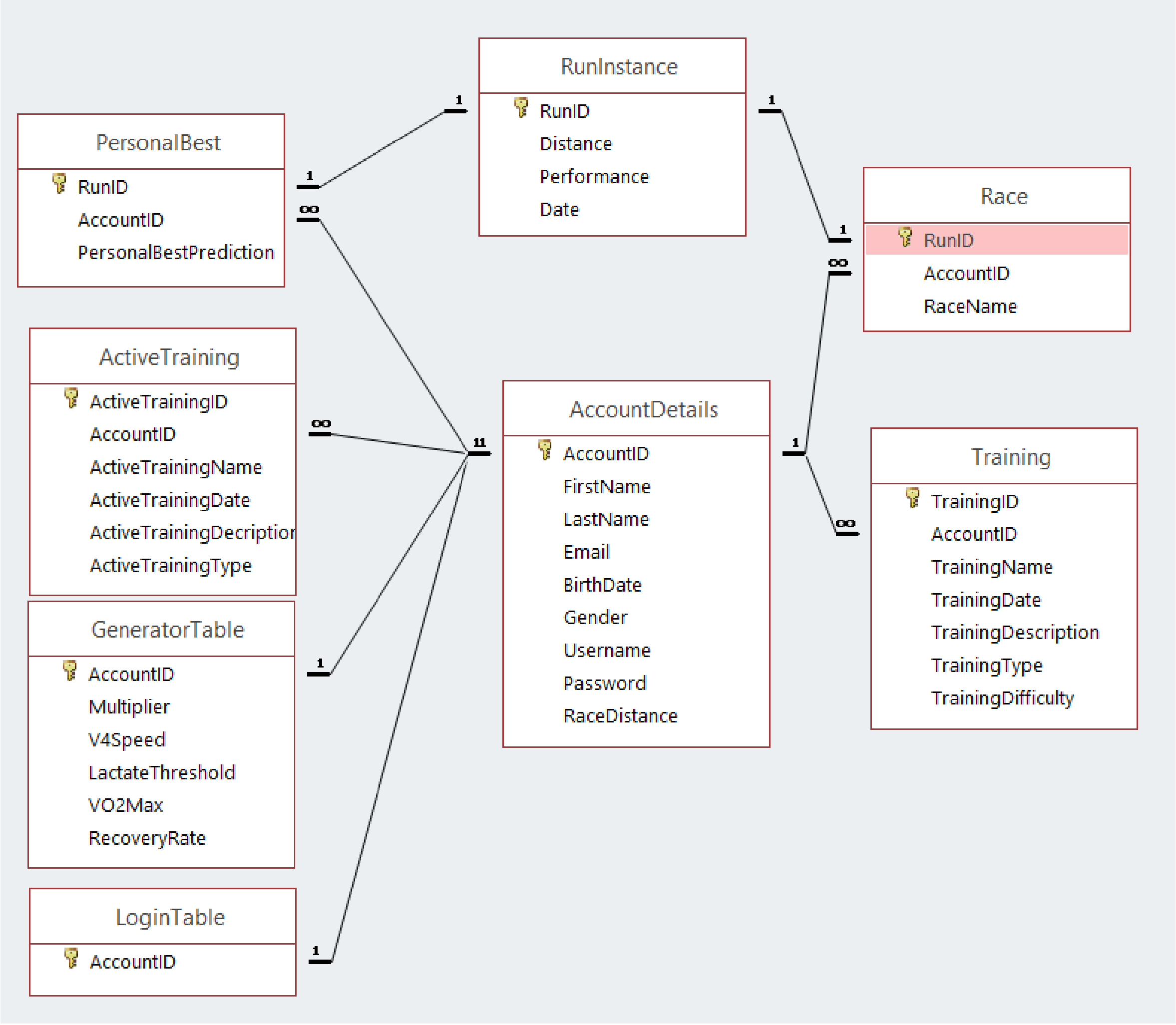


Figure 4 - ER Diagram of Dynamic Database

`AccountDetails (AccountID INTEGER, FirstName TEXT, LastName TEXT, Email TEXT UNIQUE, BirthDate TEXT, Gender TEXT, Username TEXT UNIQUE, Password TEXT, RaceDistance TEXT, PRIMARY KEY(AccountID))`
`ActiveTraining (ActiveTrainingID INTEGER, AccountID INTEGER, ActiveTrainingName TEXT, ActiveTrainingDate TEXT, ActiveTrainingDescription TEXT, ActiveTrainingType TEXT, PRIMARY KEY(ActiveTrainingID))`
``GeneratorTable (AccountID INTEGER, Multiplier REAL, V4Speed`REAL, LactateThreshold REAL, VO2Max REAL, RecoveryRate REAL, PRIMARY KEY(AccountID))``
`LoginTable (AccountID INTEGER, PRIMARY KEY(AccountID))`
`PersonalBest (RunID INTEGER, AccountID INTEGER, PersonalBestPrediction TEXT, PRIMARY KEY(RunID))`
`Race (RunID INTEGER, AccountID INTEGER, RaceName INTEGER, PRIMARY KEY(RunID))`

RunInstance (RunID INTEGER, Distance TEXT, Performance TEXT, Date TEXT, PRIMARY KEY(RunID))
Training (TrainingID INTEGER, AccountID INTEGER, TrainingName TEXT, TrainingDate TEXT, TrainingDescription TEXT, TrainingType TEXT, TrainingDifficulty TEXT, PRIMARY KEY(TrainingID))

Figure 5 - Dynamic Database Tables

Android studio supports the easy implementation of SQLite databases. SQLite databases are instances of non-volatile relational databases stored locally on the android device, that require no configuration and implement server-less transaction of SQL. SQLite databases only store 5 distinct datatypes: INTEGER, TEXT, REAL, BLOB, NUMERIC. I will use INTEGER for integer values, TEXT for string values and REAL for double values. I will use the free software ‘DB Browser for SQLite’ for creating a model of my SQLite database and testing queries. I will also use it for creating and inputting initial records into the static database.

Figure 4 describes the entity relationships of the dynamic database will be stored locally on the device so the application can be run on different android devices. This will be done by writing a class that extends (inherits) the android *SQLiteOpenHelper* class. 6 different entities will be stored in the local SQLite database: *Account_Details*, *Account_Stats*, *Active_Training*, *Completed_Training*, *Race* and *PersonalBest*. Multiple instances of these objects can be stored in the database in different records. The *AccountDetails* tables is the central table with one account relating to multiple *Race, PersonalBest, Active_Training* and *Completed_Training* objects. The *RunInstance* table is used to fully normalise the database, one *PersonalBest* object relates to one *RunInstance* and one *Race* object relates to one *RunInstance*. This means a *PersonalBest* object could be related to a *Race* object but not necessarily, and also one *Race* could relate to a *PersonalBest* but not necessarily and also less likely, as a *PersonalBest* is the fastest instance of a *RunInstance* for a particular distance. The user could have many *Race* objects of a particular distance but only the fastest one is related to a *PersonalBest*. This makes the data atomic as oppose to having 2 separate tables for *PersonalBest* and *Race* objects. The problem this poses is reading and writing *PersonalBest* and *Race* objects to the database as 2 tables must be accessed.

SQL	Function
CREATE TABLE Training (TrainingID INTEGER PRIMARY KEY, AccountID INTEGER, TrainingName TEXT, TrainingDate TEXT, TrainingDescription TEXT, TrainingType TEXT, TrainingDifficulty TEXT);	DDL *Create Table* statement will be used in the *onCreate* method which overrides from the android *SQLiteOPenHelper* Class. If the database does not exist on the device the *onCreate* method is called and all tables are created.

INSERT INTO GeneratorTable
VALUES (1, 1.0, 3.0, 4.0, 55, 0.05)
	Insert Into statement will be used in the *onCreate* method to transfer data from the static database to the dynamic database when it is created.

DROP TABLE IF EXISTS AccountDetails	DDL *Drop Table* statement will be used in the *onUpgrade* method overridden from *SQLiteOpenHelper* class to drop tables when the database is upgraded.

INSERT INTO RunInstance
(Distance, Performance, Date)
VALUES (‘800m’, ‘2:02.56’, ‘05/12/2017’)
SELECT SCOPE_IDENTITY()
 	
INSERT INTO RaceTable
VALUES (54, 4, ’Grand Prix’)
	2 tables will need to be accessed when inserting a *Race* object into the database. Firstly, details will be inserted into the *RunInstance* table and then the primary key of that record will be selected. The primary

	key will be used to insert the remaining details into the *Race* table to make sure the 2 records are related.

`SELECT *`
`FROM ActiveTrainingTable`
`WHERE AccountID = 4`
	This simple select statement returns all active training records using the account id as a parameter.

`DELETE FROM Training`
`WHERE TrainingID = 56`
	This simple delete statement deletes a record for the training table using the training id as a parameter.

`UPDATE LoginTable`
`SET AccountID = 4`
 	
`ContentValues contentValues =`
`new ContentValues();`
`contentValues.put(LoginTable.ACCOUNT_ID,`
`accountDetails.getAccountID());`
 	
`database.update(LoginTable.TABLE_NAME,`
`contentValues, LoginTable.ACCOUNT_ID + " =`
`?",new String[]{cusor.getString(0)});`
	The update statement updates a record in a table. This will be used for some objects. However, a simple alternative is provided by Java. The *ContentValues* object stores a value and an attribute identifier. After the *ContentValues* object has been filled with all the data we can use the *update* method to update the record using the id as a parameter. This method can also be used for inserting records and will only be used for completing a transaction on one table, otherwise raw SQL will be used.

`SELECT PersonalBest.RunID,`
`PersonalBest.AccountID,`
`RunInstance.Distance,`
`RunInstance.Performance, Runinstance.Date`
`FROM PersonalBest, RunInstance`
`WHERE RunInstance.Distance = '1500m'`
`AND RunInstance.RunID = PersonalBest.RunID`
`AND PersonalBest.AccountID = 7`
	This is a more complex select statement that return all the details of a *PersonalBest* object. It is required to traverse the *PersonalBest* and *RunInstance* table to fetch all the details. The statement takes the distance and account id as parameter. A cursor will then be used to instantiate the object.

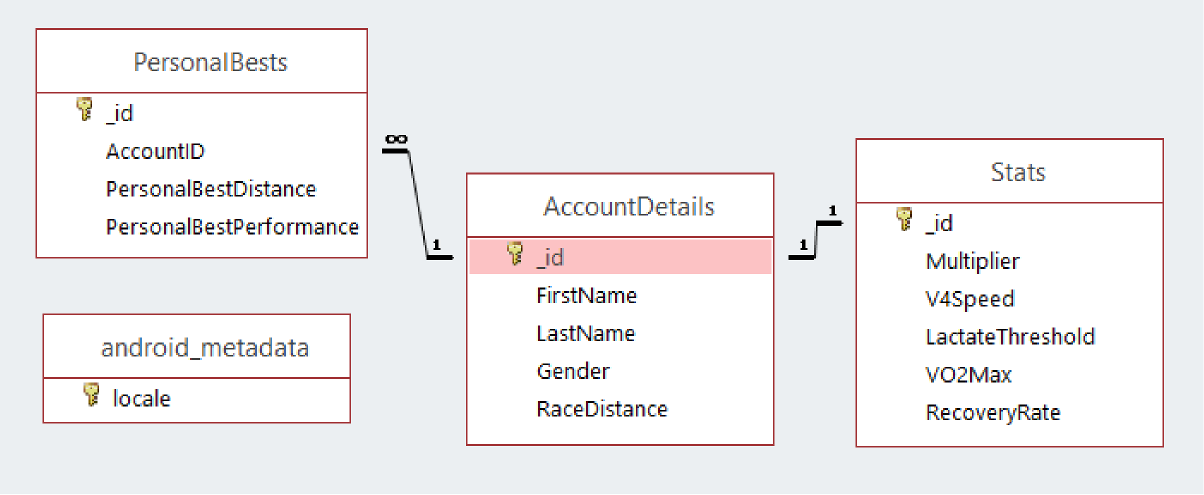


Figure 6 - ER Diagram of Static Database

AccountDetails (_id INTEGER, FirstName TEXT, LastName TEXT, Email TEXT, DateOfBirth TEXT, Gender TEXT, Username TEXT, Password TEXT, RaceDistance TEXT, PRIMARY KEY(_id))
PersonalBest (_id INTEGER, AccountID TEXT, PersonalBestDistance TEXT, PersonalBestPerformance TEXT, PersonalBestDate TEXT, PersonalBestPrediction TEXT, PRIMARY KEY(_id))
Stats (_id INTEGER, Multiplier REAL, V4Speed REAL, LactateThreshold REAL, VO2Max REAL, RecoveryRate REAL, PRIMARY KEY(_id))
android_metadata (locale TEXT DEFAULT 'en_US')

Figure 7 - Static Database tables

Figure 6 and *Figure 7* describe the entity relationships of the static database. Data from the static database will be loaded into the corresponding tables in the dynamic database when it is created. *Figure 8* describes the dataflow from the tables in the static database to the corresponding tables in the dynamic database. The android_metadata tables in the static database is simply used to configure the static database rather than store any records.

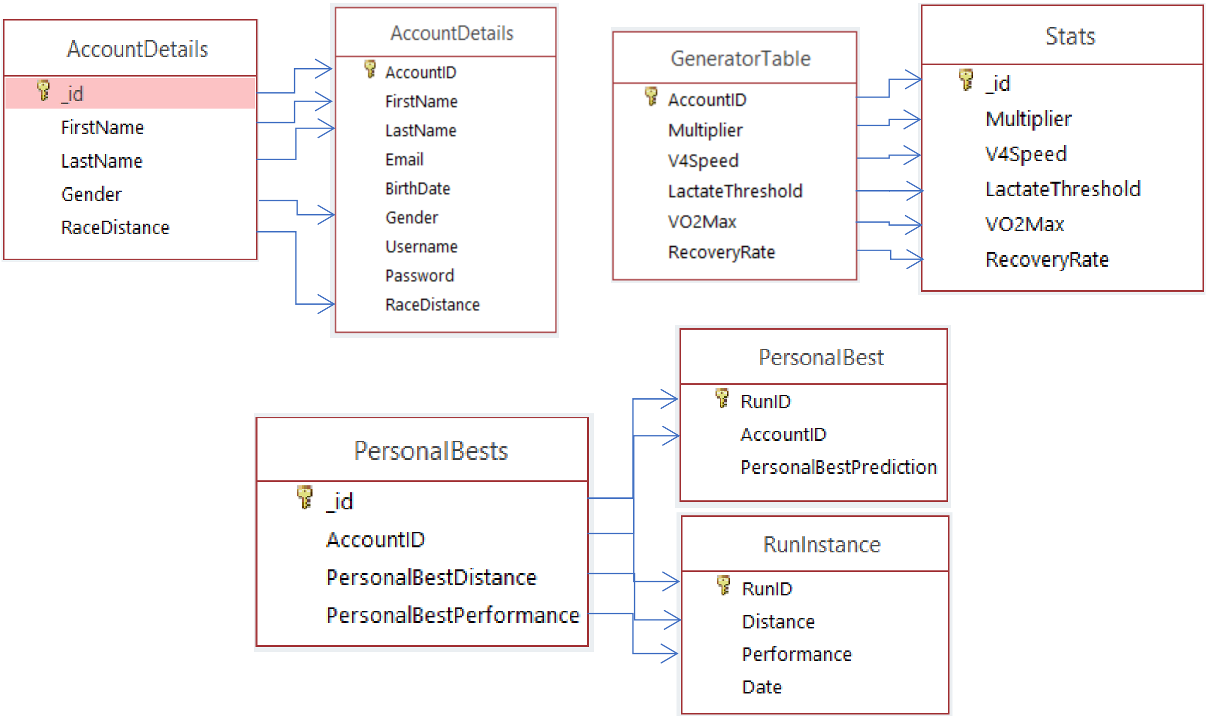


Figure 8 - Static Database Data Flow Diagram

GUI

Colour Scheme

Colour Name	Hexadecimal code (RGB)	xml	Colour
Primary	#7cb342	<color name="colorPrimary">#7cb342</color>	🟩
Primary Light	#aee571	<color name="colorPrimaryLight">#aee571</color>	🟩
Primary Dark	#4b830d	<color name="colorPrimaryDark">#4b830d</color>	🟩
Accent	#4CAF50	<color name="colorAccent">#4CAF50</color>	🟩
Secondary	#004d40	<color name="colorSecondary">#004d40</color>	🟩
Secondary Light	#39796b	<color name="colorSecondaryLight">#39796b</color>	🟩
Secondary Dark	#00251a	<color name="colorSecondaryDark">#00251a</color>	⬛
White	#ffffff	<color name="white">#ffffff</color>	⬜
Grey	#9e9e9e	<color name="grey">#9e9e9e</color>	▨
Transparent	#00000000	<color name="trans">#0000000</color>	⬜
Black Overlay	#66000000	<color name="black_overlay">#66000000</color>	⬛

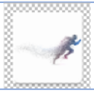
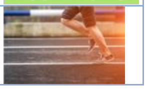
Note: for 8-digit hexadecimal codes the first 2 digits represent the transparency of the colour, 00 being fully transparent and FF being fully opaque.

Font and Sizes

Font style	Colour	Font Family	Size	Appearance	Text
Android Default	Black Overlay	Sans-serif	14sp	Small	TextView
Title	Black	Sans-serif-medium	18sp	Button/Bold	**TextView**
Sub-Title	Black	Sans-serif	18sp	Body	TextView
Activity Type	White	Sans-serif	18sp	Small, Bold	TextView
Borderless Button	Black Overlay	Sans-serif	14sp	Caption, All caps	TEXTVIEW
Sub-Heading	Black Overlay	Sans-serif	16sp	Body, Bold	TextView
Edit Text Default	Black Overlay	Sans-serif	18sp	Medium Inverse	TextView
Information Text	Grey	Sans-Serif	14sp	Small	TextView
Information Title	Grey	Sans-serif	18sp	Headline	TextView
Body Text	Black	Sans-serif	14sp	Body	TextView

Button	Black	Sans-serif-medium	14sp	Button, All caps	TEXTVIEW
Coloured Button	White	Sans-serif-medium	14sp	Button, All caps	TEXTVIEW
Message Header	Grey	Sans-serif	14sp	Small	TextView

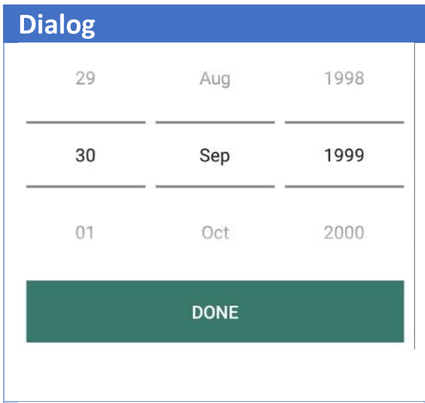
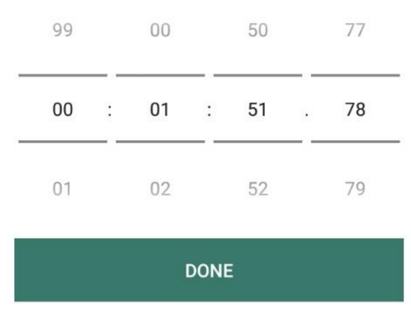
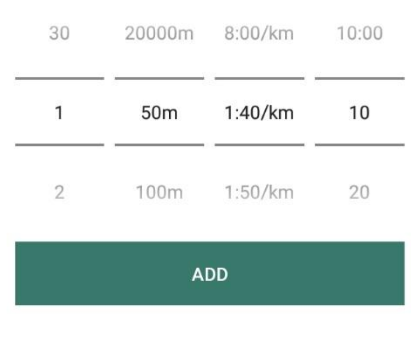
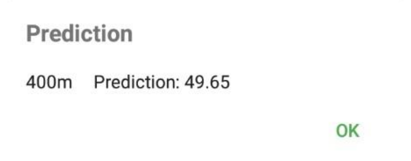
Icons and Images

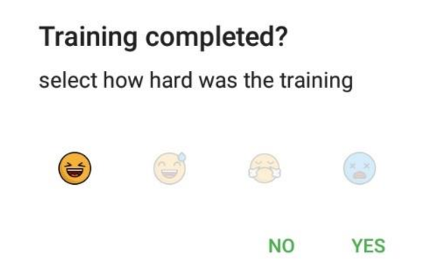
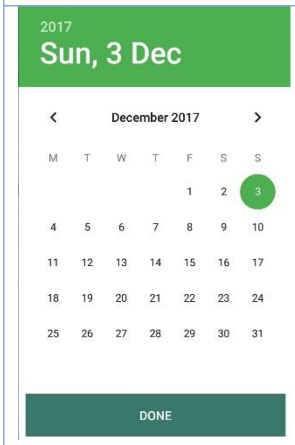
File Name	Size	Colour	Type	Icon
active_training_focused	android:width="24dp" android:height="24dp"	"@color/grey"	</vector>	A
active_training_icon	android:width="24dp" android:height="24dp"	"#FF000000"	</vector>	A
ic_account_box_black_24dp	android:width="24dp" android:height="24dp"	"#FF000000"	</vector>	👤
ic_add_white_48px	android:width="48dp" android:height="48dp"	"@color/white"	</vector>	+
ic_challenging	android:width="24dp" android:height="24dp”	-	</vector>	😣
ic_directions_run_black_24dp	android:width="24dp" android:height="24dp”	"#FF000000"	</vector>	🏃
ic_easy	android:width="24dp" android:height="24dp”	-	</vector>	😆
ic_edit_black_24dp	android:width="24dp" android:height="24dp”	"@color/white"	</vector>	✎
ic_event_note_black_24dp	android:width="24dp" android:height="24dp”	"#FF000000"	</vector>	📅
ic_good	android:width="24dp" android:height="24dp”	-	</vector>	😅
ic_hard	android:width="24dp" android:height="24dp”	-	</vector>	😵
ic_info_black_24dp	android:width="24dp" android:height="24dp”	"#FF000000"	</vector>	ℹ
ic_insert_chart_black_24dp	android:width="24dp" android:height="24dp”	"#FF000000"	</vector>	📊
personal_best_focused	android:width="24dp" android:height="24dp”	"@color/grey"	</vector>	📋
personal_best_unfocused	android:width="24dp" android:height="24dp”	"#FF000000"	</vector>	📋
training_focused	android:width="24dp" android:height="24dp”	"@color/grey"	</vector>	📅
icon_run	72x72	-	PNG	
logo_track_trainer	388x87	-	PNG	Track Trainer
myback	792x507	-	PNG	

ic_menu_home	48x48	"#FF000000"	PNG	⌂
ic_menu_action	48x48	"@color/grey"	PNG	⌂

Custom Dialogs

Android studio supports dialog interfaces. A dialog is a small window that prompts the user to make a decision or enter information. A dialog does not typically fill the screen and can show a title, up to 3 buttons, a list of selectable items, or a custom layout. I will use dialogs for providing the user with information, prompting them to make decisions and allowing them to easily input information.

Dialog	Function
	Date Selector Dialog - Will be used for selecting a date, when creating or editing an account. - Year goes from 1930-2016, so never in the future. - Day selector wheel changes values based on month. If month has 31 days, then day selector wheel takes values from 1-31. Else it takes values from 1-30. - If year is gap year and month is February then day selector takes values from 1- 29. - Output is a string format "dd/mm/yyyy".
	Time Selector Dialog - Will be used for selecting the performance of a race or personal best, when editing an account, updating a personal best, recording a race or personal best. - Standard chronometer form hh:mm:ss.cc. - Hour and centisecond (10th of a second) take values 0-99. - Minutes and seconds take values 0-59. - Output is a string format "hh:mm:ss.cc".
 30 20000m 8:00/km 10:00 1 50m 1:40/km 10 2 100m 1:50/km 20 ADD	Training Set Dialog - Will be used for adding or editing a *Training_Set*, when uploading a *Completed_Training* or *Active_Training* object. - Repetition wheel takes values from 1-30. - Distance wheel takes values from 50m-20000m. - Speed wheel takes paces in time/km from 1:40 to 8:00/km. - Rest wheel takes times from 10s – 10 minutes. - Output is saved in a string format (reps + "x" + distance + "Time: " + time + " Rest: " + rest). - Time is calculated from speed and distances selected.
	Prediction Dialog - Used for displaying the calculated prediction of a distance. - Will display distance and predicted personal best from prediction model.
	Saving Dialog - Will be displayed when a time-consuming process is happening on a different thread.

 Training completed? select how hard was the training 	Completed Dialog - Will be used to select the difficulty of an *Active_Training* object as it is being completed. - Difficulty will be saved to the database along with details of the training, as a *Completed_Training* object. - Difficulty can be selected by clicking on icon that corresponds with how the training felt. - When the icon is clicked it is highlighted.
 2017 Sun, 3 Dec December 2017 M T W T F S S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 	Day Selector Dialog - Will be used to select the date, when uploading a *Completed_Training* or *Race* object. - Will make use of the date selector widget provided by android. Default date is the current day. - When day is clicked on it is highlighted and date displayed is changed appropriately. - Month can be navigated with arrows, year can be navigated by pressing on the year displayed, and scrolling to the find desired year. - Output is a string format "dd/mm/yyyy". - Output can also not be a date in the future.

Animations

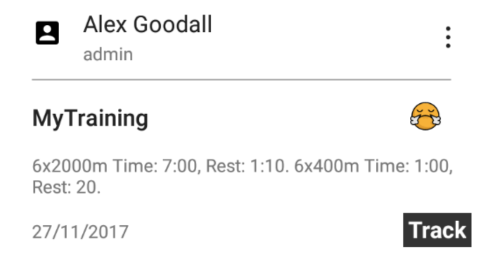
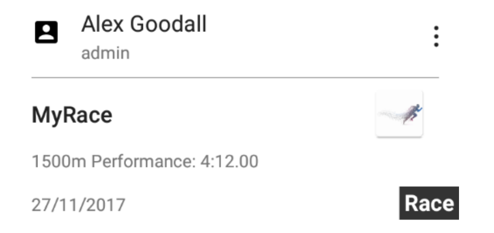
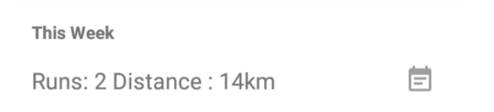
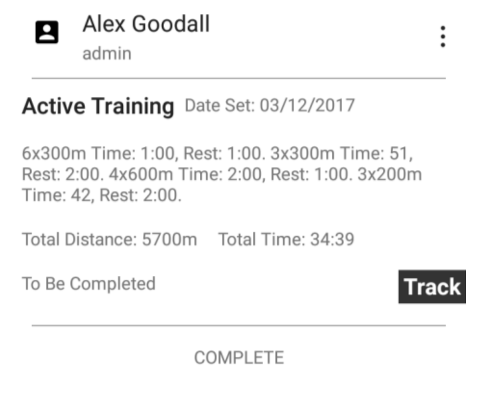
The animation package in android studio allows developers to define a sequence of animation instructions in an XML file or android code. I will use XML to define animations which will be used on the *FloatingActionButton* widget in the *MainPage* activity.

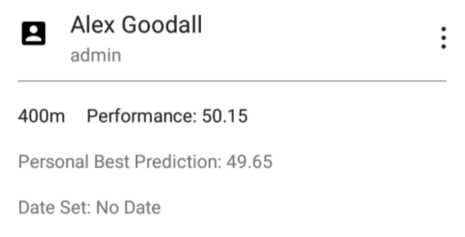
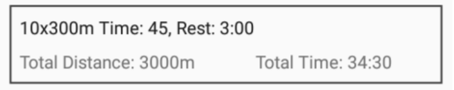
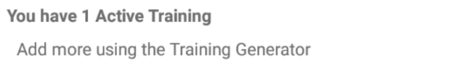
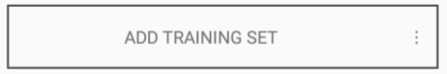
<table>
<tr><th>File Name</th><th>xml</th></tr>
<tr><td><b>fab_close</b></td><td><pre>
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:fillAfter="true">
    <scale
        android:duration="300"
        android:fromXScale="1"
        android:fromYScale="1"
        android:toXScale="0.0"
        android:toYScale="0.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:pivotX="50%"
        android:pivotY="50%"
        />
    <!-- this makes object decrease in size from 100% to 0% size
over a duration of 300ms -->
    <alpha
        android:duration="300"
        android:fromAlpha="1.0"
        android:toAlpha="0.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        />
    <!-- this makes object disappear from opaque to fully
transparent over a duration of 300ms -->
</set>
</pre></td></tr>
<tr><td><b>fab_open</b></td><td><pre>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">
    <scale
        android:duration="300"
        android:fromXScale="0.0"
        android:fromYScale="0.0"
        android:toXScale="1"
        android:toYScale="1"
</pre></td></tr>
</table>

```
            android:interpolator="@android:anim/linear_interpolator"
            android:pivotX="50%"
            android:pivotY="50%"
            />
        <!-- this makes object increase in size from 0 to 100% size over
a duration of 300ms -->
        <alpha
            android:duration="300"
            android:fromAlpha="0.0"
            android:toAlpha="1.0"
            android:interpolator="@android:anim/accelerate_interpolator"
            />
        <!-- this makes object appear from transparent to fully opaque
over a duration of 300ms -->
</set>
```

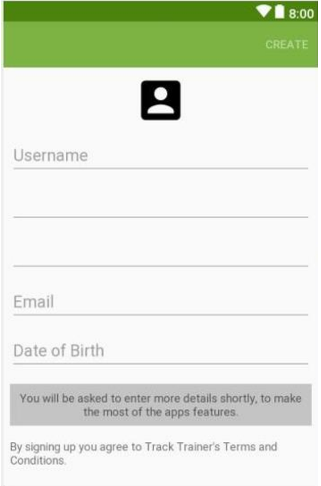
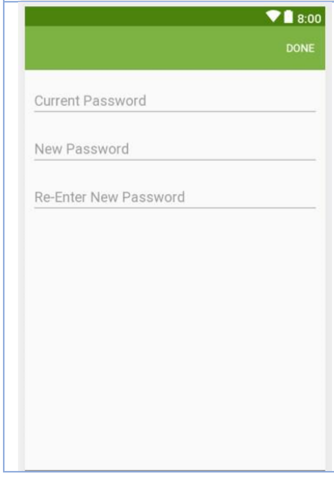
Recycler View Layouts

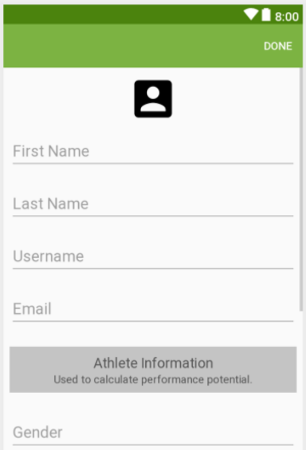
The *RecyclerView* is a more advanced and flexible version of the *ListView* widget. The *RecyclerView* is a view group that displays a list of scrollable items. The items are inserted using an adapter that pulls content from a source and converts it into a defined layout before placing and displaying it in a list. I will use SQL queries to fetch data from the database and instantiate the corresponding object. I will use *RecyclerView* widgets to inflate custom layouts that display all the information about the objects fetched from the database.

Layout	Object
 Alex Goodall admin MyTraining 6x2000m Time: 7:00, Rest: 1:10. 6x400m Time: 1:00, Rest: 20. 27/11/2017 Track	- Displays the details of a *Completed_Training* object: name, description, date, type and difficulty. - Displays the corresponding account details: first name, last name and username. - Menu button will display a pop-up menu with options to delete and edit the object.
 Alex Goodall admin MyRace 1500m Performance: 4:12.00 27/11/2017 Race	- Displays the details of a *Race* object: name, distance, performance and date. - Displays the corresponding account: first name, last name and username. - Menu button will display a pop-up menu with options to delete and edit the object.
 This Week Runs: 2 Distance : 14km	- Displays a *HomePageItem* object. Displays the text, description and image that corresponds to the image resource ID stored in the object.
 Alex Goodall admin Active Training Date Set: 03/12/2017 6x300m Time: 1:00, Rest: 1:00. 3x300m Time: 51, Rest: 2:00. 4x600m Time: 2:00, Rest: 1:00. 3x200m Time: 42, Rest: 2:00. Total Distance: 5700m Total Time: 34:39 To Be Completed Track COMPLETE	- Displays the details of an *Active_Training* object: name, date, description, type, total distance and total time. Total distance and time are calculated using the description from the database when the object is instantiated. - Displays the corresponding account details: first name, last name and username. - Menu button will display a pop-up menu with options to delete, complete and replace the *Active_Training.* - Complete button displays dialog to complete the *Active_Training.*

 Alex Goodall admin 400m Performance: 50.15 Personal Best Prediction: 49.65 Date Set: No Date	- Displays the details of a *PersonalBest* object: distance, performance, date (if exists), prediction (if exists). - Displays the corresponding account details: first name, last name and username. - Menu will button display a pop-up menu with options to delete or edit the object.
 Distance 35km	- Displays a *Statistic* object: name, value, units.
 10x300m Time: 45, Rest: 3:00 Total Distance: 3000m Total Time: 34:30	- Displays a *Training_Set* object: repetitions, distance, time, rest. - If object is long pressed then pop-up menu will be displayed with options to delete or edit the object. - Total distance and time calculated from the object's values.
 You have 1 Active Training Add more using the Training Generator	- Displays a *Message* object: title, body
 	- Empty object that will display a dialog to add a training set when pressed.

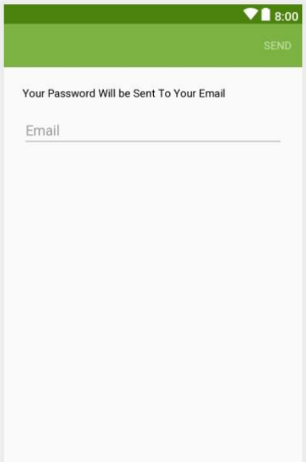
Activity Layouts

Activity Layout	Function
 You will be asked to enter more details shortly, to make the most of the apps features. By signing up you agree to Track Trainer's Terms and Conditions.	Create Account Activity - Allows the user to create an account by entering: username, email, password and date of birth. - Date of birth is entered with a custom view dialog. - Create button in the tool bar will light up when all the entries have been filled. - System will check that the email and username aren't already being used by another user. If they are, the fields will be emptied and a toast message pops-up notifying the user. - Will make the user enter their password twice and checks they are matching. - When account is created the user will be directed to the edit profile activity to complete their account details.
 8:00 DONE Current Password New Password Re-Enter New Password	Edit Password Activity - Allows the user to change their password by entering their current password and typing their new password twice. - Done button in the toolbar will light up when all the fields have been filled. - If the current password is incorrect or the new passwords do not match then the entries will be emptied, and the user will be notified with a toast message why they cannot change their password.



Edit Profile Activity

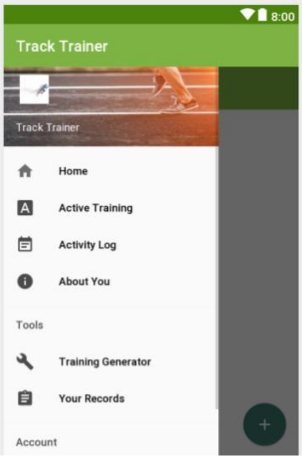
- Allows the user to edit their account details: first name, last name, username, email, gender, birth date, primary race distance and primary race distance personal best.
- Done button in the toolbar will light up when the entries are filled.
- If the email or username is already in use the entries will be changed to their original and the user is notified with a toast message why they cannot update their account.
- Gender, birth date, race distance and personal best will all be selected with dialogs.
- If the details have been changed and the user presses the back or home button a dialog will appear confirming navigation away from the activity.



Your Password Will be Sent To Your Email

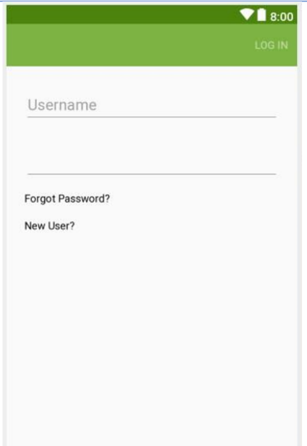
Forgot Password Activity

- Allows the user to enter their email if they have forgot their username or password.
- Sends an email over the internet with the username and password in the email body.
- If email is wrong or internet connection cannot be established then the user will be notified with a toast message.



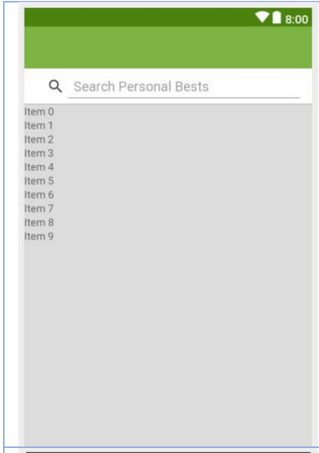
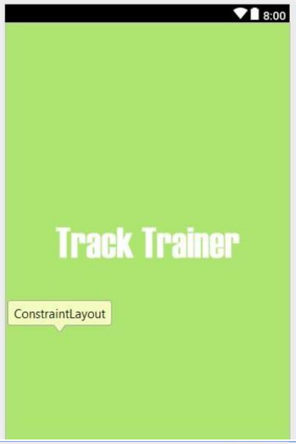
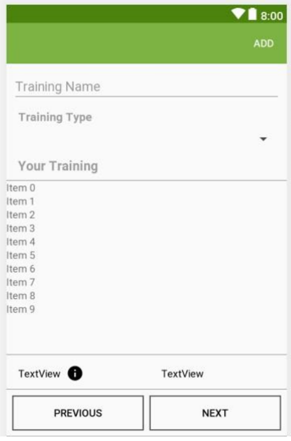
Main Page Activity

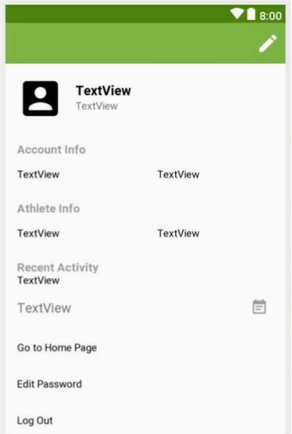
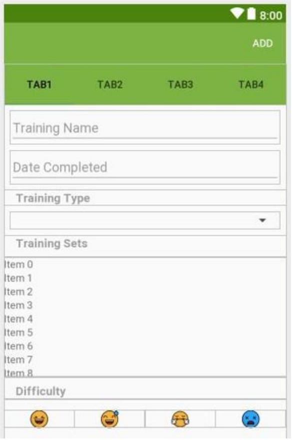
- Main Activity that allows the user to navigate with a navigation drawer (slide-out menu) to various other activities and fragment layouts.
- Activity will hold 4 different fragment layouts with recycler view widgets that hold different objects and data: Home, Active Training, Activity Log and About You.
- Floating action button will allow the user to add an activity manually.
- Menu button in the action bar will allow the user to refresh the data shown in the fragment layout's recycler view widget, or navigate to the add activity.



Login Activity

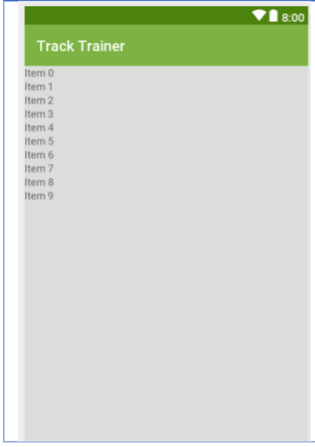
- Allows the user to enter their username, password and login into the system and navigate to the main page activity.
- Login button in the toolbar will light up when the entries have been filled.
- Small text views "Forgot Password?" and "New User?" can be clicked on to navigate to the forgot password and create account activities respectively.
- If the details are invalid then the user will be notified with a toast message that either their username or password is incorrect.

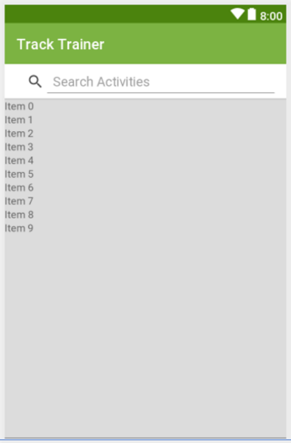
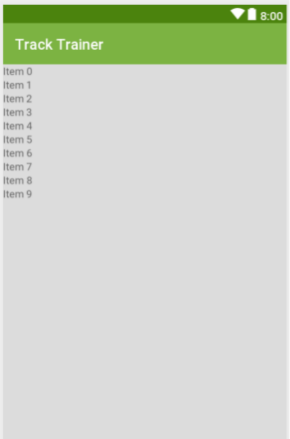
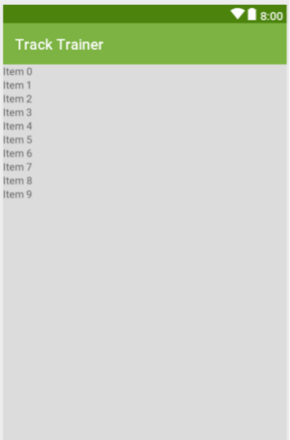
 8:00 Search Personal Bests Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Item 9	Personal Best Predictor Activity - Displays all personal best objects in the database related to the account logged in in a recycler view widget. - Recycler view can be filtered using the search function that will search the records by distance. - Menu button on the layout of the objects displayed can be used to edit, delete the object or generate a prediction for that distance. - Recycler view widget is the child of a swipe to refresh layout, so when the layout is swiped upwards the data within the recycler view will be fetched from the database and refreshed in the layout. - Objects will be ordered in ascending distance (shortest to longest distance).
	Splash Screen Activity - Launcher activity of the application, meaning it is the first activity that is instantiated when the application is run. - The activity will check if there is an account logged in locally to the device by querying the database. - If there is no account logged in then the login activity will be loaded, otherwise the main page is loaded and the account on the database is automatically logged in. - In the rare case when an account has just been created and the account details are not complete the splash screen will load the edit profile activity.
 8:00 ADD Training Name Training Type Your Training Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Item 9 TextView TextView PREVIOUS NEXT	Training Generator Activity - Allows the user to generate a training program by selecting their desired training type with a spinner widget that loads in a string array of values from the resources xml file. - The user will be able to title their training in the first entry. - The next button will generate the next training, and the previous will button load the previous training generated on the stack. - The 2 text views will display the total distance and total time of the training program. - The recycler view will display all the training sets generated for the training program. - The add button in the toolbar will light up when all the entries have been filled and a training program has been generated. If pressed a dialog pops-up to confirm to add that generated training program. - If the next button is pressed when a training type has not been selected a dialog will pop-up directing the user to do so. - If 5 *Active_Training* objects related to the account logged in exist then the user will be notified with a dialog they cannot have more than 5 *Active_Training* objects at any one time.

 8:00 TextView TextView Account Info TextView TextView Athlete Info TextView TextView Recent Activity TextView TextView Go to Home Page Edit Password Log Out	View Profile Activity - Allows the user to view all the details of their account and their recent activity in the past 7 days. - Edit button in the tool bar allows the user to navigate to the edit profile activity. - The activity will navigate to the main page with the home button in the tool bar or the button in the layout. - The user can also log out or navigate to the edit password activity using the buttons in the layout.
 8:00 ADD TAB1 TAB2 TAB3 TAB4 Training Name Date Completed Training Type Training Sets Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Difficulty	Add/Edit Activity - Allows the user to add a *Completed_Training*, *Active_Training*, *Race* or *PersonalBest* object to the database. - All attributes of each of these objects can be entered with an edit text widget, spinner widget, dialog or custom layout. - The user can choose which object to add to the database by clicking on the corresponding tab icon. All the entries will be emptied when the user navigates to another tab layout. - Add button will light up when all the entries are filled in the tab layout that the user is currently in. - Any dates selected must not be in the future and the user will be notified with a toast message if they pick a date in the future. - Edit activity extends (inherits) from the add activity, this means the layout is identical. The fundamental differences are: data from the object that is being edited will be loaded into the entries accordingly, and the user will not be able to change the tab layout displayed.

Fragment Layouts

A fragment represents a behaviour or user interface within an activity. A fragment can be re-used in multiple activities, so it is effectively a modular part of an activity, which receives its own inputs and acts like a sub-activity, which can be added, removed or replaced while the activity is running . A fragment contributes its own layout and user interface to the parent activity that is hosting the fragment. The fragment can be given functionality by implementing the *onCreateView()* Method. I will use fragments for providing different information and interfaces within the *MainPage* activity.

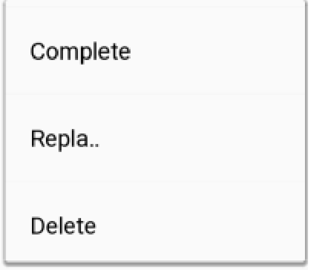
Layout	Function
 8:00 Track Trainer Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Item 9	Home Page Fragment - Default fragment layout displayed in the Main Activity every time it is loaded or instantiated. - Recycler view widget will display a *HomePageItem* object that will display the amount of activity (time and distance) the athlete has run in the past 7 days or week. - Displays a second HomePageItem object that will display the amount of active trainings the athlete has on the database. - Displays a simple message object saying 'Recent Activities'. - Displays the 5 most recent activities (*Race* or *Completed_Training*) objects that have been added to the database by that account.

	- Recycler view is the child of a swipe to load layout so swiping up on the layout will fetch the data from the database again and refresh the layout.
 8:00 Track Trainer Search Activities Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Item 9	Activity Log Fragment - Layout loaded via the navigation drawer (slide-out menu). - Displays a simple message saying 'Your Activities'. - Displays all *Completed_Trainings* and *Race* objects in the database related to the account logged in. Objects will be ordered by date, most recent to oldest. - Will have search functionality that filters the recycler view objects by name or description. - Recycler view is the child of a swipe to load layout so swiping up on the layout will fetch the data from the database again and refresh the layout.
 8:00 Track Trainer Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Item 9	Active Training Fragment - Layout loaded via the navigation drawer (slide-out menu). - Displays a message object indicating how many *Active_Training* objects the account has related to it in the database. - Displays all *Active_Training* objects related to the account in the database. - Recycler view is the child of a swipe to load layout so swiping up on the layout fetches the data from the database again and refreshes the layout.
 8:00 Track Trainer Item 0 Item 1 Item 2 Item 3 Item 4 Item 5 Item 6 Item 7 Item 8 Item 9	About You Fragment - Layout loaded via the navigation drawer (slide-out menu) - Displays *Statistic* objects about the account logged in, calculated from a database query. - Displays average weekly activity: number of trainings, distance and time. - Displays past month, and past year activity: number of trainings, distance and time.

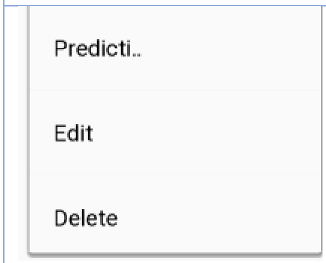
Menus

Menus are a common user interface component in android applications that provide the user with actions and other options. Android studio provides a standard XML format to define a menu of items. Menus can be used in a variety of contexts within an application. I will use XML menus for providing the user with actions in a custom layout of a *RecyclerView*, for providing the navigation drawer (slide-out menu) with a menu of available actions and activities to navigate to, and finally for providing the user with actions in the *MainPage* activity toolbar. Defining a menu with XML is far more useful than defining it in code, because the same menu can be inflated from a variety of different contexts and activities, rather than re-writing code across different classes and activities.

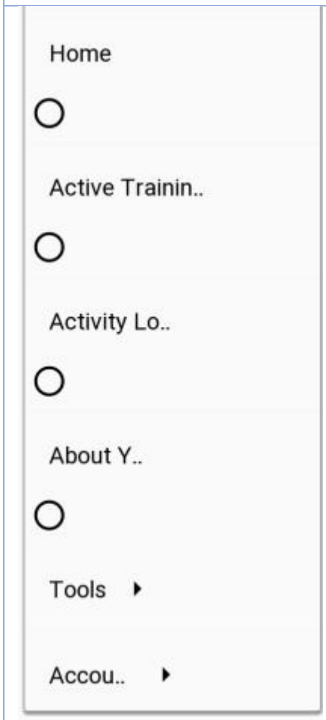
Menu | xml



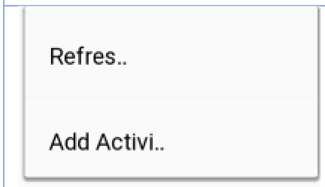
```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/active_training_complete"
        android:title="Complete" />
    <item
        android:id="@+id/active_training_replace"
        android:title="Replace" />
    <item
        android:id="@+id/active_training_delete"
        android:title="Delete" />
</menu>
```



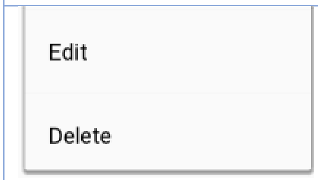
```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/personal_best_menu_predict"
        android:title="Prediction" />
    <item
        android:id="@+id/personal_best_menu_edit"
        android:title="Edit" />
    <item
        android:id="@+id/personal_best_menu_delete"
        android:title="Delete" />
</menu>
```



```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_home"
            android:icon="@drawable/ic_menu_home"
            android:title="Home" />
        <item
            android:id="@+id/nav_activetraining"
            android:icon="@drawable/active_training_icon"
            android:title="Active Training" />
        <item
            android:id="@+id/nav_traininglog"
            android:icon="@drawable/ic_event_note_black_24dp"
            android:title="Activity Log" />
        <item
            android:id="@+id/nav_aboutyou"
            android:icon="@drawable/ic_info_black_24dp"
            android:title="About You" />
    </group>
    <item android:title="Tools">
        <menu>
            <item
                android:id="@+id/nav_generatetraining"
                android:icon="@drawable/ic_menu_manage"
                android:title="Training Generator" />
            <item
                android:id="@+id/nav_personalbest"
                android:icon="@drawable/personal_best_unfocused"
                android:title="Your Records" />
        </menu>
    </item>
    <item android:title="Account">
        <menu>
            <item
                android:id="@+id/nav_viewprofile"
                android:icon="@drawable/ic_account_box_black_24dp"
                android:title="Your Profile" />
            </menu>
        </item>
</menu>
```



```
<menu xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_refresh"
        android:orderInCategory="100"
        android:title="Refresh"
        app:showAsAction="never" />
    <item
        android:id="@+id/actions_add"
        android:orderInCategory="102"
        android:title="Add Activity" />
</menu>
```



```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_edit"
        android:title="Edit" />
    <item
        android:id="@+id/menu_delete"
        android:title="Delete" />
</menu>
```

Algorithms

Quantifying the Difficulty of a Training Program (Modelling Lactate Changes)

```
SET k = 40
SET BaselineLactate = 1
SET LactateTime = 15
SET z = 20
SET f = 6
//Constants

SET AvgLactate = 0
SET Lactate = BaselineLactate
//Lactate = BaselineLactate before exercise has started
SET TotalTime = CALL Training.GetTotalTime

SET c = 1
SET b = k / VO2Max
SET a = V4Speed
//V4Speed determine the shift of the graph along the x axis
//VO2Max manipulates the gradient of the graph
SET d = BaselineLactate – e^(-b*a)
//y intercept must always be equal to the BaselineLactate

INSTANTIATE ExponentialModel with a, b, c and d
// where y = c*e^(b(x-a)) + d

FOR i = each Training Set in Training
   FOR j = each Rep in Training Set
      CALL RunDistance with Distance, Time and LactateThreshold
      IF j < each rep in training set - 1 THEN
         CALL Rest with Time and RecoveryRate
      END IF
   END FOR
   IF i <= each training set in Training – 1 THEN
      Call Rest with 5 minutes and RecoveryRate
      //5 minutes rest in-between each Training Set
   END IF
END FOR

SUB PROCEDURE RunDistance
   SET LactateLevel = CALL ExponentialModel.GetYValue with LactateLevel
   //LactateLevel is the expected level to be reached when exercising at a particular speed
   IF LactateLevel > LactateThreshold THEN
      //If LactateLevel is above the LactateTreshold, Lactate could increase above LactateLevel
      SET LTime = (BaselineLactate / Lactate) * LactateTime
      //LTime is the during of exercise which will result in the LactateLevel being reached
      SET m = (LactateLevel – Lactate) / LTime
      SET c = Lactate
      INSTANTIATE LinearModel1 with m and c
      //where y = mx + c
      IF LTime > Time THEN
         //If running for shorter than LTime LactateLevel is not reached
         FOR i = each second in Time
            Lactate = CALL LinearModel1.GetYValue with i
            //Calculate average cumulatively
            AvgLactate = AvgLactate + (Lactate / TotalTime)
         END FOR
      ELSE
         FOR i = each second in LTime
            Lactate = CALL LinearModel1.GetYValue with i
            AvgLactate = AvgLactate + (Lactate / TotalTime)
         END FOR
         FOR i = each second in (time – Ltime)
            //After LactateLevel is reached Lactate increases above LactateLevel
            //According to a linear model that is re-defined each iteration or second
            //In-effect the increase is not linear and closer to exponential
            SET m = (Lactate - LactateThreshold) / z
            SET c = LactateLevel
            INSTANTIATE LinearModel2 with m and c
            //where y = mx + c
            Lactate = CALL LinearModel2.GetYValue with i
            AvgLactate = AvgLactate + (Lactate / TotalTime)
         END FOR
      END IF
   ELSE
      SET LTime = (BaselineLactate / Lactate) * LactateTime
      INSTANTIATE LinearModel1 (y = mx + c) with m and c
      //where m = (LactateLevel – Lactate) / LTime, c = Lactate
      IF LTime > time THEN
         FOR i = each second in time
            //If running for shorter than LTime LactateLevel is not reached
            Lactate = CALL LinearModel1.GetYValue with i
            AvgLactate = AvgLactate + (Lactate / TotalTime)
         END FOR
      ELSE
         FOR i = each second in LTime
            Lactate = CALL LinearModel1.GetYValue with i
            AvgLactate = AvgLactate + (Lactate / TotalTime)
         END FOR
         FOR i = each second in (time – Ltime)
            //When LactateLevel is reached Lactate remains constant and equal to LactateLevel
            Lactate = LactateLevel
            AvgLactate = AvgLactate + (Lactate / TotalTime)
         END FOR
      END IF
   END IF
END
```

```
SUB PROCEDURE Rest
   SET Rise = (Lactate – LactateThreshold) / f
   //Rise defines the total rise over a period equal to LactateTime
   //f used as rate constant
   SET RiseRate = Rise / LactateTime
   //RiseRate defines the rate of rise each second
   IF Rise > 0 THEN
      //If rise is above 0 Lactate is above the LactateThreshold so a rise occurs
      FOR i = each second in Time
         IF i < LactateTime THEN
            //If resting for shorter than LactateTime change is defined as follows
            Lactate = Lactate – RecoveryRate + RiseRate
         ELSE
            //After LactateTime decrease is normal and equal to RecoveryRate
            Lactate = Lactate - RecoveryRate
         END IF
         IF Lactate < BaselineLactate
            Lactate = BaseLineLactate
         END IF
         AvgLactate = AvgLactate + (Lactate / TotalTime)
      END FOR
   ELSE
      //Else no rise occurs
      FOR i = each second in Time
         //Lactate decreases accordingly
         Lactate = Lactate – RecoveryRate
         IF Lactate < BaselineLactate
            Lactate = BaseLineLactate
         END IF
         AvgLactate = AvgLactate + (Lactate / TotalTime)
      END FOR
   END IF
END
```

Training Generator

```
SET LactateBracket = according to what Training Type equals
//LactateBracket defines the difficulty or average lactate the athlete should be running
at during the Training
SET DistanceDistribution = CALL GetDistanceDistribution with Training Type and Database
//Set discrete probability distribution generated from database query and initial
distribution
FOR i = number of Training Sets
   SET Rest = 0
   SET Distance = CALL DistanceDistribution.GetXValue
   //Get value of X from discrete probability distribution
   IF Training Type is track THEN
      SET RepDistribution = CALL GetRepsDistribution with Distance and Database
      SET RestDistribution = CALL GetRestDistribution with Distance and Database
      SET Reps = CALL RepDistribution.GetXValue
      SET Rest = CALL RestDistribution.GetXValue
   ELSE
      SET Reps = 1
      SET Rest = 0
   END IF
      SET Time = 0
   INSTANTIATE Training Set with Distance, Reps, Time, Rest
   ADD Training Set to Array of Training Sets
END FOR
INSTANTIATE Training with Array of Training Sets
//A Training consists of one or more Training Sets
CALL OptimiseTraining with Training

SUB PROCEDURE OptimiseTraining
   INSTANTIATE Array of Training Sets with Training
   FOR i = each Training Set in Training
      //Optimise each Training Set in the Training
      FOR j = 10 to 100
         //For loop varies speed from 10s per 100m to 100s per 100m
         SET Time = (j * Set Distance) / 100
         Training Set = CALL Training Set.SetDistanceTime using Time
         //Sets the time the athlete must run in the given distance
         ADD Training Set to Array of Training Sets
         INSTANTIATE Training with Array of Training Sets
         SET AvgLactate = CALL CalculateAvgLactate with Training
         IF AvgLactate <= LactateBracket THEN
            //If AvgLactate is less than LactateBracket a suitable speed for the athlete
to run has been found
            BREAK LOOP
         ELSE
            //Else Training Set is not suitable
            REMOVE Training Set from Array of Training Sets
         END IF
      END FOR
   END FOR
END
```

Personal Best Predictor

```
Set Distance1, Set Distance2 according to what Distance equals
//Distance1, Distance2 and Distance are all unique
//Distance is the distance that a prediction will be made for

Set PersonalBest1 = Database Transaction using account ID and Distance1
Set PersonalBest2 = Database Transaction using account ID and Distance2

IF PersonalBest1 doesn’t exists THEN
   YA1 = 0
ELSE
   YA1 = CALL PersonalBest1.GetPerformance
   //YA1 defines the actual value of performance for distance1 where it exists
END IF

IF PersonalBest2 doesn’t exists THEN
   YA2 = 0
ELSE
   YA2 = CALL PersonalBest2.GetPerformance
   //YA2 defines the actual value of performance for distance2 where it exists
END IF

SET RegressionLine1 = CALL GetRegressionLine with Distance1 and Distance
//RegressionLine comparing Distance1 and Distance
SET RegressionLine2 = CALL GetRegressionLine with Distance2 and Distance
//RegressionLine comparing Distance2 and Distance

IF YA1 AND YA2 = 0 Then
   XP = 0
   //Prediction cannot be made
ELSE
   IF YA1 = 0 THEN
      XP = CALL RegressionLine2.GetXValue with YA2
      //Prediction made solely off YA2
   ELSE IF YA2 = 0 THEN
      XP = CALL RegressionLine1.GetXValue with and YA1
      //Prediction made solely off YA1
   ELSE
      XP1 = CALL RegressionLine1.GetXValue with YA1
      XP2 = CALL RegressionLine2.GetXValue with YA2
      //Statistical check to see which prediction is most accurate
      //Comparing which prediction is closer to the actual value
      XDevs1 = AbsoluteValue((XA – XP1) / RegressionLine1.StandardDeviation)
      XDevs2 = AbsoluteValue((XA – XP2) / RegressionLine2.StandardDeviation)
      //Prediction is made, taking the more accurate prediction into account
      XP = (XDevs1 * XP1 / (XDevs1 + XDevs2)) + (XDevs2 * XP2 / (XDevs1 + XDevs2))
   END IF
END IF

IF XP = 0 THEN
   Prediction = nothing
ELSE
   IF XA < XP THEN
      //If prediction is worse than the actual value discard it
      Prediction = XA / ((Multiplier / 100) + 1)
      //Multiplier value used in the database modifies the prediction
   ELSE
      Prediction = XP / ((Multiplier / 100) + 1)
   END IF
END IF

SUB PROCEDURE GetRegressionLine
   SET count = 0
   SET PersonalBestArray1 = DataBase query using distance1
   SET PersonalBestArray2 = DataBase query using distance2

   //Loop used to make sure personal bests are corresponding to the same account
   //Some accounts may not have a personal best in Distance1 and Distance2
   FOR i = PersonalBestArray1 length
      FOR j = PersonalBestArray2 length
         IF PersonalBestArray1[i] is recorded by the same person as
PersonalBestArray2[j] THEN
         PerformanceArray1[count] = PersonalBestArray1[i].GetPerformance
         PerformanceArray2[count] = PersonalBestArray2[j].GetPerformance
         Count = count + 1
         BREAK LOOP
      END FOR
   END FOR
   RETURN INSTANTIATE RegressionLine with PerformanceArray1 and PerformanceArray2
   //RegressionLine can be instantiated using 2 arrays
END
```

Stats Updater

```
SET Multiplier = 0
Array of Trainings = CALL GetPastWeekTrainings //Database query using account ID
SET total = length of Array of Trainings

FOR i = total
   SET AccountStats = CALL GetAccountStats //Database query using account ID
   CALL PassTraining with Training
   //Each of the values average is calculated cumulatively
   V4Speed = V4Speed + (AccountStats get V4Speed / Total)
   LactateThreshold = LactateThreshold + (AccountStats get LactateThreshold / Total)
   VO2Max = VO2Max + (AccountStats get VO2Max / Total)
   RecoveryRate = RecoveryRate + (AccountStats get RecoveryRate / Total)

   IF Training is hard or challenging THEN
      Multiplier = Multiplier + 0.3
   ELSE
      Multiplier = Multiplier + 0.1
   END IF
END FOR

Set PastMonthActivity = CALL GetPastMonthActivityNumber //Database transaction using
account ID

//Used to check if training in the past month has been consistent
SET Check = ((PastMonthActivity * 7) / 30.42) * 0.3

IF Multiplier > Check THEN
   //If Multiplier > Check then training in the past month has not been consistent
   Multiplier = Check
```

```
END IF

INSTANTIATE AccountDetails with account ID, Multiplier, V4Speed, LactateThreshold and
RecoveryRate

SAVE TO DATABASE AccountDetails

SUB PROCEDURE PassTraining
   SET EstimatedLactate = according to Training Type and Difficulty of Training

   SET V4Speed = 2.85
   SET LactateThreshold = 3
   SET VO2Max = 20
   SET RecoveryRate = 0.05
   //Set values to their lower limit or smallest reasonable values they can take

   FOR i = 60
      V4Speed = V4Speed + 0.05
      //Vary V4Speed
      FOR j = 60
         LactateThreshold = LactateThreshold + 0.05
         //Vary LactateThreshold
         FOR k = 60
            //Vary VO2Max
            VO2Max = VO2Max + 1
            FOR l = 15
               //Vary RecoveryRate
               RecoveryRate = RecoveryRate + 0.01
               SET AvgLactate = CALL CalculateAvgLactate with Training
               IF 0 < (EstimatedLactate – AvgLactate) < 1 THEN
                  //If (EstimatedLactate – AvgLactate) is between 0 and 1 then this
variation of variables is a solution to the problem
                  Solution[4] = {V4Speed, LactateTheshold, VO2Max, RecoveryRate}
                  ADD Solution to Array of Solutions
               END IF
            END FOR
         END FOR
      END FOR
   END FOR

   SET SolutionIndex = 0
   SET SolutionValidity = 0

   RESET V4Speed, LactateThreshold, VO2Max, RecoveryRate to their original values using
Database query using account ID
   //Reset values for comparison

   //Check to see which of the solutions is most realistic for the athlete
   FOR i = length of Array of Solutions
      SET SolutionValidity = AbsoluteValue(Solution[0] – V4Speed / V4Speed) +
AbsoluteValue(Solution[1] – LactateThreshold / LactateThreshold)
AbsoluteValue(Solution[2] – VO2Max / VO2Max) + AbsoluteValue(Solution[3] – RecoveryRate
/ RecoveryRate)
      //Validity of solution calculated by percentage difference
      IF i = 0 OR ThisSolutionValidity < SolutionValidity THEN
         //Lowest solution validity gives the best solution
         SolutionIndex = i
         SolutionValidity = ThisSolutionValidity
      END IF
   END FOR

   SET Solution[4] = in Array of Solutions using SolutionIndex
   //Select best solution and set values

   SET V4Speed = Solution[0]
   SET LactateThreshold = Solution[1]
   SET VO2Max = Solution[2]
   SET RecoveryRate = Soluition[3]

   RETURN INSTANTIATE AccountStats with V4Speed, LactateThreshold, VO2Max, RecoveryRate
END
```