Evaluation

Objectives

All the objectives were met to a very reasonable standard and all that was set out to be done was achieved. From the initial problem, I have created an android application that allows the user to record their trainings, race and personal bests in detail. I have also implemented a page that provides the user with information about themselves, so they can track their progress. I have also met the initial requirement of implementing a training generator that provides the user with relevant and useful trainings. However, the initial reliability of this feature could be improved in my opinion. I also believe I could have done more in terms of statistical analysis of the data the user enters; providing the user with a better idea of how they are progressing and other achievements. I kept a lot of variables associated to the athlete hidden from the user which I could have made available to the user, so they could make the most of the training generator feature. This would have also provided the user with a better idea of how the training generator works and how they are progressing. Other than that, I believe the application almost completely fulfils the initial problem to a professional standard allowing the user to easily upload data to the database, keeping it all stored in one place.

Objective	Explanation
Creating an account by inputting: username, password, email and date of birth, details saved to database.	I implemented a *CreateAccount* activity, with *EditText* views for inputting a username, password and date of birth. The date of birth is inputted with a custom dialog that can only select a valid date from 1930-2008. The details are saved in the database with an INSERT INTO SQL command.
Forgotten password – sending an email (containing password and username) to a user after they have entered their email correctly (internet required for this).	I implemented a *ForgotPassword* activity with an *EditText* view for inputting the email of your account. I used a JSSE provider class that provides the device with safe and encrypted access to the internet from an android application. I also wrote a *GMailSender* class that makes use of the SMTP using the Gmail mail server as a host to send an email (containing the username and password of the account) to the desired email. The class uses try-catch statements to make sure any errors are handled (no internet connection, email doesn't exist).
Secure system by logging into the application by entering your username and password correctly as private data will be stored.	I implemented a *Login* activity with 2 *EditText* views, one for inputting a username and the other for inputting a password. An SQL query is used on the database to check if the username and password match a specific account, access to the rest if the application is then given if the correct details are inputted.
Display some information regarding the athlete and recent activities on the home page.	On the *MainPage* activity I incorporated 4 fragment layouts, one of which is the default fragment – the *HomePage.* The *HomePage* uses a *RecyclerView* layout to display 2 *HomePageItem* objects. One object displays information on how many runs and how far the athlete has run in the current week (calculated from an SQL query on the database). The other object displays the number of active trainings the account has (found using an SQL query). The *RecyclerView* also displays the most recent activities recorded by the account.

Navigating from activity to activity using a *Navigation Drawer Layout* (slide-out menu).	In the *MainPage* activity I incorporated a *Navigation Drawer Layout* which can be accessed by a button in the top left corner of the action bar. This allows the user to change the fragment layout or navigate to another activity.
Viewing active trainings given by training generator or added manually. Functionality to replace, delete or complete an active training.	The *ActiveTraining* fragment is also incorporated into the *MainPage* account. It uses a *RecyclerView* to display details of *Active_Training* objects (found using a parameterised SQL query on the database). In the *ViewGroup* that is used to display the *Active_Training* objects I incorporated a button with a drop-down menu to replace, delete or complete the Active_Training object. These commands are executed with the appropriate SQL command.
Dialog to complete and select the difficulty of an active training.	When complete is selected on the drop-down menu or the complete button is pressed on the *ViewGroup* that displays the *Active_Training* object a pop-up dialog is displayed. On the pop-up dialog *ImageViews* are used to select the difficulty of a training.
Activity log displaying all activities (trainings and races) in the database with search functionality to search by name and description. Functionality to edit or delete any of the displayed activities.	The *TrainingLog* fragment is incorporated into the *MainPage* activity. I used a *RecyclerView* to display the details of *Race* and *Completed_Training* objects found using a parameterised SQL query on the database. The *ViewGroup* that is used to display both the Race and *Completed_Training* objects incorporates a button with a drop-down menu to delete or edit any of the objects. The delete functionality is provided by a DELETE SQL command. The edit functionality sends the object to the *EditActivity* activity. A *SearchView* is used to filter the objects displayed in the *RecyclerView* by a string query, filtering *Completed_Training* objects by name and description, and filtering *Race* objects by name.
Statistics activity which displays information about the athlete's training history and average weekly activity calculated from the database.	The *AboutYou* fragment is incorporated into the *MainPage* activity. I used a *RecyclerView* to display information and statistics about the athlete; average weekly activity (time run, distance run, training completed), past year activity and all-time activity (distance run, trainings completed) is displayed using a *Statistics* object and *ViewGroup*. All the statistics are calculated from SQL queries on the database.
A training generator activity which allows the athlete to name their training, select the type of training and generate an appropriate training to be saved as an active training to the database. Only 5 active trainings maximum allowed.	The *TrainingGenerator* activity can be accessed from the Navigation Drawer. I used an *EditText* view to allow the user to name their training, a *Spinner* with a drop-down menu is used to allow the user to select the *TrainingType* and a *RecyclerView* is used to display the *Training_Set* objects generated. If 5 *Active_Training* objects are found on the database with an SQL query, then the system warns the user with a dialog and does not allow another *Active_Training* object to be added by the training generator. Otherwise, once a training is generated the 'add' button can be pressed, and the details of the

	generated training are added to the database as an *Active_Training* object using the INSERT INTO SQL command.
'training generator' function must adapt to the athlete's habits, needs, abilities. By consistently generating distances and programmes the athlete wants to run at the appropriate speed.	I decided to use the idea of discrete probability distributions to shape the training generator algorithm. Typically, only one discrete probability distribution is used, and this is for the distance. A discrete probability distribution is generated using an initial distribution, the *TrainingType* and a database query on distances run by the athlete of a particular *TrainingType*. This distribution is used to generate a distance for the user for run. If the *TrainingType* 'Track' is selected then at least 4 discrete probability distributions are used: one for the distance(s), one for the rest in between each repetition of a particular distance (for each distance), one for repetitions of a particular distance (for each distance), one for the amount of sets (different distances to run). Each discrete probability distribution is created from an initial distribution, database queries, and any other parameters needed. I used discrete probability distributions to provide variety in the trainings generated, and as more data on the athlete is gathered the distributions become more representative of the athletes needs and habits with respect to the distances they tend to run, the amount of reps they tend to do for a specific distance, the amount of rest they tend to take for a specific distance and the amount of sets they tend to do. Secondly, the training generator algorithm uses a lactate model to decide what speed the athlete should run the distance(s) generated for him/her. I designed the lactate model to make use of 4 biological variables hidden at the user end that represent the athletes ability: *V4Speed, LactateThreshold, VO2Max, RecoveryRate*. Each biological variable represents a different aspect of the athlete's ability and are initially calculated using regression line models created when the athlete creates an account and inputs his/her personal best for his/her main distance. The regression line is modelled off real data stored initially in the static database. The algorithm decides the lactate level the athlete should be operating under for each *TrainingType*. The algorithm varies the speed at which the athlete should be running and calculates the average lactate for the training using the lactate model, which in turn uses the athlete's biological variables. When an optimum solution is found (a speed where the average lactate calculated is just below the lactate level the athlete should be operating under) the training has been generated and is displayed. Finally, when an athlete completes an *Active_Training* or adds a *Completed_Training* object, the difficulty of the training calculated by the lactate model in conjunction with the

	difficulty inputted by the user are used to modify the biological variables associated to the athlete accordingly. This means next time a training is generated it is at a more appropriate speed/difficulty.
A 'Next' and 'Previous' button to choose between the training programmes generated for the athlete. Using a stack object to store previously generated trainings.	In the *TrainingGenerator* activity I used 2 buttons: 'Next' and 'Previous'. When the 'Next' button is pressed the details of the current training being displayed are pushed onto a java stack object, then a new training is generated and displayed. When the 'Previous' button is pressed the details of the previously generated training are popped off the java stack object and displayed. I made sure the *TrainingType* does not change with each button press and must be manually changed if a different type of training is required.
A Personal Bests activity which displays all the records and predictions the athlete has recorded on the database. Functionality to edit and delete any records displayed.	The *PersonalBestPredictor* activity can be accessed from the Navigation Drawer. I used a *RecyclerView* to display the details of every *PersonalBest* object found using a parameterised SQL query on the database. The *ViewGroup* that displays the details of the *PersonalBest* object incorporates a button with a drop-down menu to edit or delete any of the objects. The delete functionality is provided by a DELETE SQL command. And the edit functionality sends the object to the *EditActivity* activity. Any predictions generated are stored in the database in the same record as the details of the *PersonalBest* object and displayed in the same *ViewGroup.*
Search functionality to search through the personal bests by distance.	I used a *SearchView* in the *PersonalBestPredictor* to filter the *PersonalBest* objects displayed in the *RecyclerView* by a string query which provides search functionality. The *PersonalBest* objects are filtered by distance.
Prediction functionality, to make predictions for race distances based on records of other similar distances.	In the drop-down menu provided by the button in the *ViewGroup* that displays *PersonalBest* objects, I added a 'prediction' option that can be selected. This option generates a prediction for the given distance which is then saved to the database using an UPDATE SQL command. The prediction algorithm uses the *PersonalBest* records of 2 distances similar to the given distance, where they exist. The algorithm models 2 regression lines of the given distance against the 2 similar distances, using real data stored initially in the static database. If a record for one of the similar distances can't be found, then only one regression line is modelled. If both similar distances do not have a record, then a prediction can't be made. The 2 regression lines are used to give a prediction for the given distance. Then then difference in standard deviations of the 2 predicted values from the actual value are used as a measure of reliability of the 2 predictions. This is considered when putting the 2 separate predictions together. The final prediction is generated using the multiplier variable associated to the athlete. This is another hidden variable

	that represents the extent of consistency of training completed by the athlete in the past month. The prediction is then displayed in a pop-up dialog.
Profile activity that allows the user to view all the details of their account, along with an option to log out.	The *ViewProfile* activity can be accessed using the *Navigation Drawer*. I used *Text* views to display the details of the account along with recent activity in the past week (distance run). I added 3 buttons provide the user the ability to: return home, edit their password, or log out.
Edit password activity that allows the user to change their password.	The *EditPassword* activity can be accessed from the *ViewProfile* activity. I used 3 *EditText* views to allow the user to enter their current password and enter their new password twice. The 'done' button can then be pressed: if the current password is correct (authenticated with a database query) and the 2 new passwords match the password of the account is updated using an UPDATE SQL command.
Add activity, allows the user to add a training (select: date, difficulty, distance and speeds), race, active training and personal best to the database.	The *AddActivity* activity can be accessed from the *MainPage* by pressing the *FloatingActionButton* or using the drop-down menu displayed when the action bar menu button is pressed. I used a *TabWidget* to provide the activity with 4 tabs and 4 separate layouts. The 1st tab uses 2 *EditText* views that allow the user to enter the name and date (with a custom dialog) of a *Completed_Training* object, I used a *Spinner* to select the *TrainingType* with a drop-down menu and I used a custom dialog to add a *Training_Set* object. I used a *RecyclerView* to display the *Training_Set* objects that make up the *Completed_Training* object and *ImageViews* are used to select the difficulty. The *Training_Set* objects can be edited (with a custom dialog) or removed from the *RecyclerView* with a drop-down menu displayed when the object is long-pressed. The 2nd uses a similar format however I did not include the *ImageViews* for selecting a difficulty or an *EditText* view for selecting the date completed, because the second tab is used to add *Active_Training* objects. The 3rd tab is used to add *Race* objects, I used 3 *EditText* views to allow the user to enter the name, date (with a custom dialog) and performance (with a custom dialog) of their *Race* object. A *Spinner* is also used to select the distance of the *Race* object. The 4th tab is used to add a *PersonalBest* object, I used 3 *EditText* views to allow the user to enter the distance (using a pop-up dialog), performance (using a custom dialog) and the date (using a custom dialog) of their *PersonalBest* object. When the entries on a tab are filled the add button can be pressed which adds the corresponding object into the database using an INSERT INTO SQL command. Switching between tabs also causes the entries to be emptied.

Splash screen with a logo that automatically logs in a user who has been using the app on the device.	I made the *SplashScreen* activity the launching activity meaning it will be displayed first when the application is launched. It checks the local database to see if an account is logged onto the application using an SQL query. If it finds details of an account logged on then the *SplashScreen* launches the *MainPage* or *EditProfile* activity (depending on the status of the account), otherwise the *Login* activity is loaded.
Items from database loaded into a 'Recycler View' with a swipe up to load widget.	I used *SwipeRefreshLayout* to wrap the *RecyclerView* widgets in the *HomePage* fragment, *TrainingLog* fragment, *ActiveTraining* fragment and *PersonalBestPredictor* activity. When the layout is swiped upwards the event is handled and objects from the database are reloaded into the *RecyclerView* using the corresponding SQL query.
Attractive dialogs for selecting dates, times, distance, and training sets.	I used custom dialogs to select dates of objects, date of birth, adding or editing *Training_Set* objects and entering performances (times). Distances are either selected by a *Spinner* or a *SingleChoiceItem* pop-up dialog. I made sure these dialogs and methods of data insertion are standardised throughout the application.
Edit profile activity, that allows the user to edit their account details.	The *EditProfile* activity can be accessed from the *ViewProfile* activity. I used 8 *EditText* views to allow the user to enter or edit their first name, second name, username, email, gender (using a pop-up dialog), date of birth, primary race distance (using a pop-up dialog) and primary race distance personal best (using a custom dialog). When the entries are filled the details can be updated by an UPDATE SQL command.
Appropriate dialogs asking the user to continue when navigating the application and saving data to the database.	In the *ActiveTraining* fragment I use pop-up dialogs to confirm the user's decision to delete or replace the *Active_Training* objects. In the *AddActivity* and *EditActivity* activities I use a pop-up dialog to confirm the user's decision to navigate back when data in the entries has been input or changed. In the *EditProfile* activity I use a pop-up to dialog confirms the user's decision to navigate back and ask if any changes are to be saved (if entries have been changed). In the *HomePage* and *TrainingLog* fragments I use pop-up dialogs to confirm the user's decision to delete any *Race* or *Completed_Training* objects displayed in their *ViewGroup*. In the *PersonalBestPredictor* activity I use pop-up dialogs to confirm the user's choice to delete any *PersonalBest* objects. In the *TrainingGenerator* I use a pop-up dialog to confirm the user's choice to use the training generated as an *Active_Training* object. Also, I use a pop-up dialog to confirm the user's choice to navigate back after a training is generated. Finally, in the *ViewPofile* activity I use a pop-up dialog to confirm the user's choice to log out of the application.

Appropriate validation and completion checks when entering data into the database.	In the *AddActivity* and *EditActivity* activities I use validation checks on the *EditText* views that are used to select the date to make sure only a valid date in the past is entered. I use a completion check on each of the tabs to make sure an object cannot be added or edited unless all the entries are filled in the current tab. In the *CreateAccount* activity I use validation checks to make sure a valid username, password and email are entered (username and email cannot be used by anther account). I also use a completion check so that an account cannot be created until all the entries are filled. In the *EditPassword* activity I use validation checks to make sure the new password is a valid password and that the 2 new passwords match. I also use a completion check to make sure the password can't be changed until all the entries are filled. In the *ForgotPassword* activity I use a validation check to make sure the email entered matches the email of an account on the database. I also use a completion check to make sure an email is entered before the database query happens. In the *Login* activity I use a completion check to make sure a username and password is entered before the authentication happens. In the *TrainingGenerator* activity I use a completion check to make sure a *TrainingType* is selected and to make sure a training is generated and given a name. This must happen before the training can be added to the database as an *Active_Training* object. In most cases validation checks do not need to be done, because the dialogs used to enter data can only return valid data (When selecting gender, the pop-up dialog only allows male or female to be selected). I also made sure the completion checks are standardised throughout the application; I use a button in the action bar that lights up when all the entries within the activity or tab are filled, otherwise the button cannot be pressed. When invalid data is input then either a toast message or pop-up dialog is used to notify the user and prompt them to change something.
Information buttons displaying a dialog that reads "Track Trainer will assume you take a 5-minute rest in-between each training set, this allows your heart-rate to return to normal".	In the *TrainingGenerator* activity the total time of the training is calculated and displayed. To avoid any confusion, I added an information button which displays a pop-up dialog explaining why the total time calculated may be more than expected. I used a similar button in the active training tab of the *AddActivity* activity where the total time is also calculated and displayed.
Easy and logical navigation throughout the application	I made sure all activities and fragment layouts can be accessed in a logical manner with the majority being accessed from the *Navigation Drawer*, I also made sure all activities have a home up button which loads the parent activity of the current activity, if the back button on an android device is pressed the parent activity can also be

	loaded. I specified the parent activity of each activity in the *AndroidManifest.xml* file.
Logical entity relationship between Race and Personal Best objects. Whereby a personal best represents the fastest instance of a Race object of a given distance. A personal best can also be defined uniquely without a Race object associated to it, if it is the faster than all instances of Races of a given distance.	I used the idea that *PersonalBest* objects can be defined as one of 2 things: a standalone entity on the database representing a distance with an associated performance attached to it (and possibly date), or the fastest instance of a *Race* object. The way the relational database is designed allows this to be the case. The *RunInstance* table in the database has 3 main fields: distance, performance and date (also Id). A *Race* object stored on the database is directly linked to exactly one *RunInstance* record. A *PersonalBest* object stored on the database is also directly linked to exactly one *RunInstance* record. The fastest instance of a *RunInstance* record of a particular distance is always linked to a *PersonalBest* object so it can be displayed in the *PersonalBestPredictor* activity. It is possible the same *RunInstance* record is linked to a *Race* object in the database, hence this *PersonalBest* object is defined as the fastest instance of a *Race* object. If this *Race* object were to be deleted or edited the *PersonalBest* object would no longer exist and another *PersonalBest* object would be created for the new fastest *RunInstance* record where it exists. However, if the *PersonalBest* object and *RunInsance* record are not linked to a *Race* object then it becomes a standalone entity. If the *PersonalBest* object was directly edited or deleted it would not longer exist and a *PersonalBest* object would be created for the next fastest *RunInstance* record where it exists. If a *RunInstance* record is created or edited (by creating a *Race* or *PersonalBest* object) such that it is quicker than the *RunInstance* record linked to the *PersonalBest* object of a given distance, then the *PersonalBest* object becomes linked to the new *RunInstance* record which is now faster. I made sure that database operations are called frequently when operating the application to keep the relationship between *Race* and *PersonalBest* objects consistent.

Feedback

Email from Barry Allen a 1500m middle distance runner.

> Alex,
>
> I have been using and testing your application now for the past week - using it to record my trainings and input my personal bests. I am very pleased to say I have found very few flaws and using the application has been a pleasure. Firstly, I found it very easy to use your application right away from creating account to uploading my first training, I was guided quite well by the design of the app, however more prompts could have been used to direct me where to go and tell me what to do when I first started using the app - perhaps a mini tutorial or video would have helped. Otherwise, I found it quite easy to start using all the features of the app. I will admit a lot of the features I see you included I did not use, such as the forgot password, change password and edit profile features. These features are obviously necessary, so I am happy to see you have been very thorough as to include them - making sure the app is as professional as possible is key. Another standout for me was the pop-ups that are used to enter information, I found them very cool, and a very suitable colour scheme and style were used for them which also gives the app a professional touch. However, I feel you could have been more creative with some of the other user interfaces, the style of the app and user interfaces were standardized across the app which is important, but I feel in places the app looked a bit dull or empty, especially in the more basic pages. Perhaps another approach or design could have been used for certain basic features of the app. Moving on, the training generator and personal best predictor features were 2 very unique features that I was very excited to use. The personal best predictor was spot on, it made very accurate and believable predictions - nothing crazy or impossible. I was very excited to find out what a computer thought I should be running, I played around with this feature a lot and I could see the amount of thought process that went into this feature - definitely a must-keep. The training generator on the other hand was interesting. I felt when I first started using the app the training programmes that were being generated for me were very difficult. Over the week I used the app the training generator seemed to be learning somewhat and started giving me more realistic training programmes. I also noticed that the training programmes generated for longer distances such as 10km were always at the same pace as training programmes for 5km. I found this odd because it is very difficult to run the same speed at 5km and 10km. I think a bit more consideration for longer distance training programmes could be done. However, overall the training programmes generated for track sessions seemed to be more realistic which was nice. Bearing in mind I only used the app for one week, so I'm sure over perhaps months the training generator can become very reliable and become quite a useful feature to have. Other than that, everything was there, I couldn't ask for much more from a running app - all the features a runner would need were there. I could add trainings, edit trainings, record my personal bests, add races, edit races, search through lists and even view interesting statistics on my progress and trainings in the past week. All these features were well implemented and designed and gave the app a very professional refined feel - certainly making it very useful.
>
> Thanks, Barry.

One thing I had not considered until the feedback from Barry was including more prompts for the user. Once the user understands how to use the application I'm sure my design is very adequate, but I can understand how a new user could easily get lost or confused. I could add more prompts for inexperienced users in the form of pop-up dialogs or use more *Text* views to make more information available to the user initially in the *CreateAccount, EditProfile* or *ViewProfile* activities as these activities are rarely visited unless the user is new to the application. I could also create a simple video tutorial which is displayed in a borderless activity using a *VideoView* widget once the user has created their account. The second point made by Barry was that the creativity of the UI was lacking in certain places. I also noticed this – in some more basic activities such as the *ForgotPassword* and *EditPassword* activities there is a lot of blank space and even in some of the fragment layouts there is a lot of blank space when the *RecyclerView* is empty. To deal with this I could research into

different and more creative ways to implement these features, for example I could use a custom dialog to retrieve the users forgotten password rather than creating a whole activity with a lot of blank space in it. Many other basic features could be re-visited and implemented in a more creative way. Another option is to fill the blank spaces in the activities with more navigation buttons or even pictures (icons or logos) and displays. For example, I could add a picture of a running track in the background of the *ForgotPassword* activity, I could also add a display in the *EditPassword* activity providing information on the last time the password was changed. Another idea is adding a 'login with google plus' button in the *Login* activity to fill the empty space; a google API would need to be used and implemented for this feature. All these ideas would make the app design look very professional and consistent. Furthermore, although the personal best predictor feature was very highly praised by Barry I would consider revisiting the algorithm. I could improve the algorithm to base predictions off all the distances recorded by the athlete, making sure the least similar distances have the least significant effect on the prediction and the most similar distances to the given distance have the most significance on the prediction. This would improve the accuracy of the prediction as it would consider all the personal best data on the athlete. I could also consider using the biological values of the athlete stored on the database or even compare training programmes from one athlete to the next and use these comparisons to base predictions off. This would make my prediction algorithm very complicated, however if designed correctly could produce the most statistically accurate prediction possible as all the data on the athlete is being considered. Additionally, it was also made apparent by Barry that the training generator tended to be inconsistent initially when the account had just been created. There could be 3 possible reasons for this: the data in the static database is biased, the algorithm used to determine the biological values of the athlete is inconsistent, the training generator algorithm itself is inconsistent. Barry also made it apparent that the training generator seemed to improve as more data on the athlete was gathered. This suggests the data in the static database is biased or unrealistic, this means the algorithm used to determine the biological values is affected. If I were to revisit this I would consider using an online database that stores more real data about athletes personal best and their biological values. The online data could gather a lot more data if I had a large user base. This would make my data less biased as the data used and stored on the static database at the moment is simply from a small sample of athletes. The biological values on the static database are also not real and have been estimated. To deal with this I could find real biological values that have been physically tested on athletes or do some of my own testing on athletes with a sports scientist to determine the biological values of athletes in my database. Another alternative is allowing the user to input their own biological values, so they themselves can optimise the training generator function to their standard, or even better if they have had these biological values of themselves determined by a sports scientist. Finally, I would revisit the training generator function. Barry found that for longer distances the training generator was inconsistent as it made him run 10km at the same speed as 5km. Thinking about the way the algorithm models lactate this would be a problem. Perhaps in real life there is another biological factor other than blood lactate that limits an athlete's ability to run at a consistent pace, perhaps energy; as the athlete runs further he/she runs out of energy so their lactate threshold lower and more anaerobic respiration is occurring. This is a theory of mine and I would need to research into other factors that affect running performance and implement these factors as variables into the algorithm. I would need to consider how these variables affect the athlete from a logical and biological point of view to make sure the training generator can recognise the difference in difficulty between running 5km and 10km at the same speed.

Improvements

If I were to revisit this problem or build upon what I have already created I would focus on 2 main aspects, allowing the user to record their trainings in even more detail, and providing the user with

better statistical analysis for tracking their progress. Firstly, most training programmes start with a warm up and a warm down, especially track sessions. Currently there is no way to attach a warm up or warm down distance to a *Completed_Training* object. I could implement this and allow the user to add a warm up distance and warm down distance and time to their *Completed_Training* object. I would need to change the *Completed_Training* entity in the database to deal with this, I would also need to re-design the *ViewGroup* that displays the details of a *Completed_Training* object in a *RecyclerView* layout so that the warm up and warm down could be displayed. The warm up and warm down distance would be taken into account when statistics are calculated from the database, so more reliable information is provided to the user. A warm up and warm down could be added in the *AddActivity* activity or added with a custom dialog accessed from the pop-up menu that is displayed when the menu button in the *ViewGroup* is pressed. Another idea that allows the user to add more details of their *Completed_Training* and *Race* objects is to allow them to add a commentary on the object. This would allow the user to enter their own objective feedback or information on the *Completed_Training* or *Race* object so they can properly reflect and track their progress. The *Completed_Training* and *Race* entities would need to be edited in the database to allow the user to attach a commentary to the objects and the *ViewGroups* that display both objects would need to be modified so that the commentary could be displayed. Furthermore, I could be more flexible with *Training_Set* objects; currently there can only be a finite amount of different *Training_Set* objects that can be input (as they are input with a custom dialog), I would consider being more flexible and allowing the user to input their *Training_Set* objects in even more detail so that they could be very specific, allowing the user to input their trainings in as much detail as possible. This would require a lot of extra formatting if I were to support all the different *Training_Set* formats. Secondly, I would like to provide the user with more statistics enabling them to track their progress more thoroughly. The biological variables and multiplier variable that are hidden from the user could be made available to the user. I could consider expanding the database to track each time the biological variables are changed. This data could then be displayed on a graph to show the user how their biological variables are changing over time, this would allow the user to see if they are improving or not. The multiplier variable (which represents the amount of training in the past month) could also be recorded over time and displayed on a graph, so the user can see how intense they are training all year round.