T-LEVELS

Exemplification Materials Technical Qualification in Digital Production, Design and Development

Occupational Specialism: Digital Production, Design and Development

Project 2

Contents

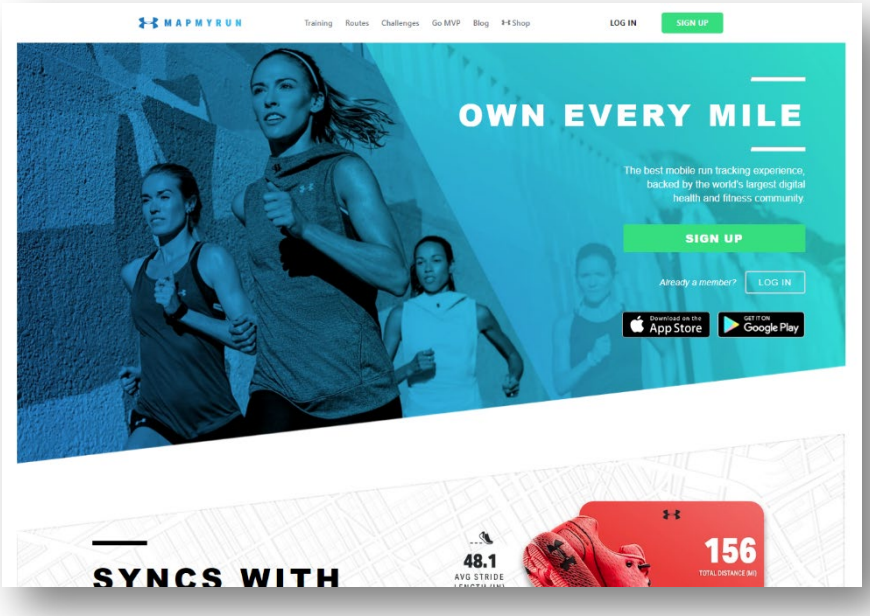
Task 1
Activity A(i)

Research

To show and emphasise our design, the following section is a comparison with an existing fitness website (in this case: https://www.mapmyrun.com/).

Homepage

From the above, you can see that the website has a minimalist style that refrains from data/information being crushed into a small screen space. The page is split up into sections where additional information to address the advertising and promotion of the website is placed. These still incorporate a minimalist design, in keeping with the theme.



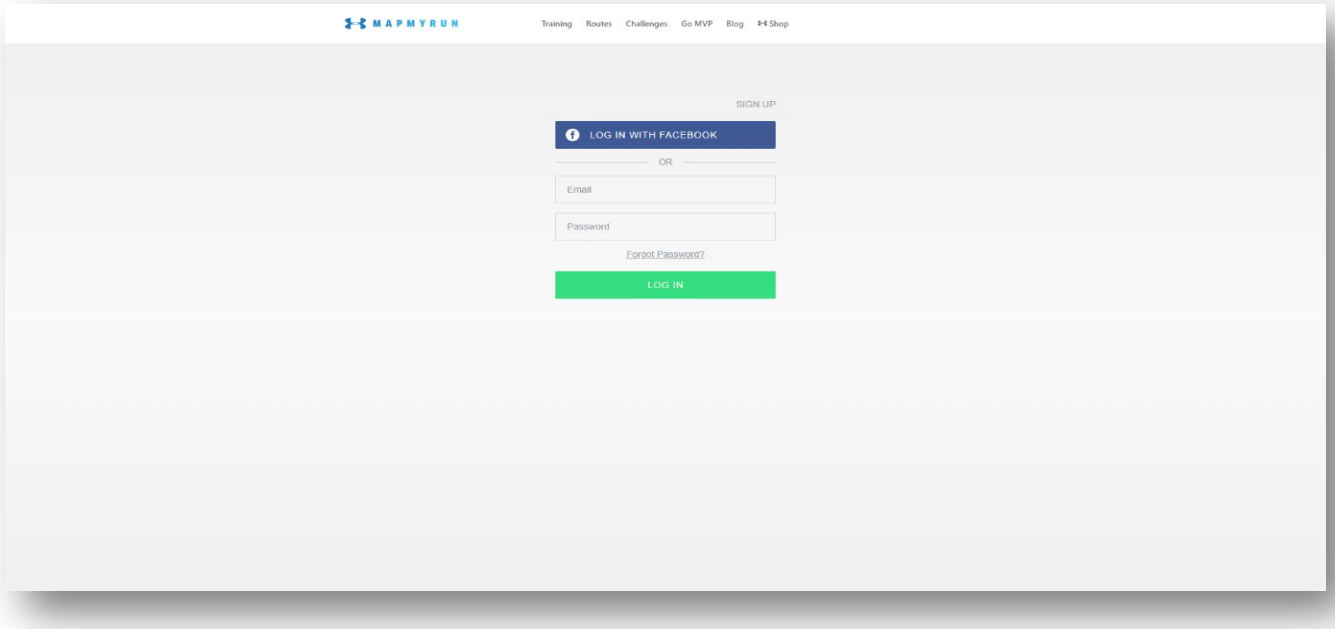
The differences are that the main colours that are used correspond to the colour of the logo. In our design we have used the same methodology and have used colours included in the newly-designed logo and in the original. Furthermore, MapMyRun (Under Armour) has used links to an app as well as logging information on a desktop. Their reasoning will be that it makes it easier for the user to access the data from their smartphone instead of signing into a desktop website, however choosing to include the desktop option allows for more customers who may not have a compatible smartphone to use the service.

Loading Times of Homepage

1.27s - MapMyRun
970ms – Our Mock Website

This shows that our website will give people a faster response and will allow users to access the website quicker than larger corporations such as Under Armour's MapMyRun.

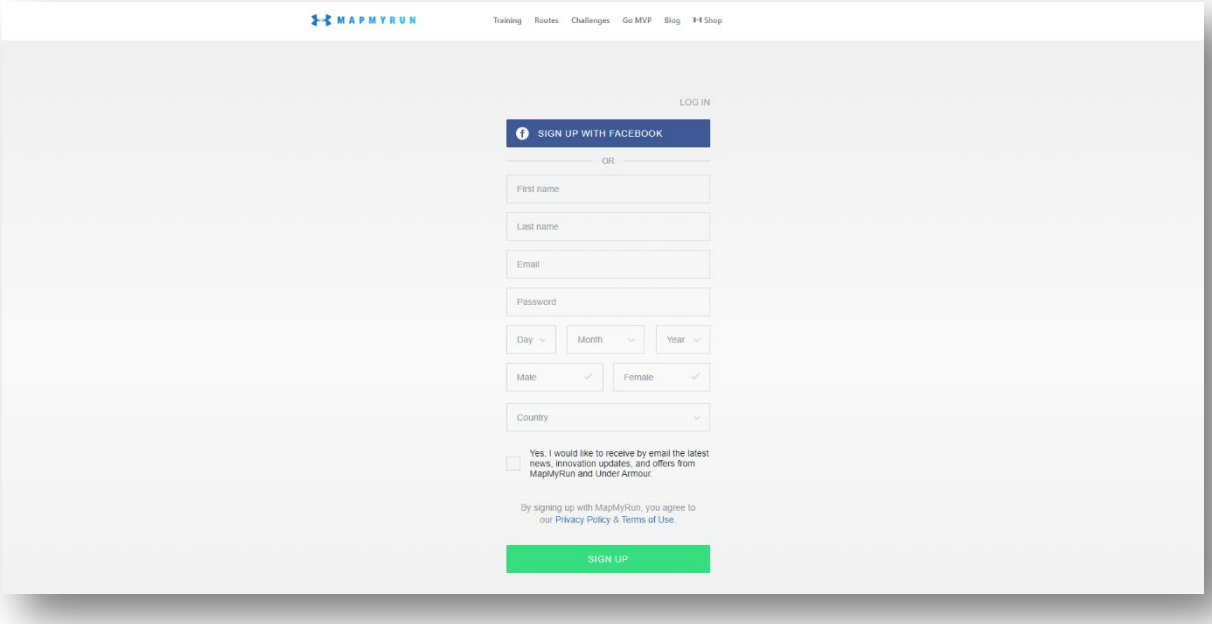
Login Page



Again, a minimalist design has been used in the login page that shows continuity within the website and provides a professional look for the user. However, MapMyRun has used a separate page for the user to login whereas, in our design, the login page is a pop-up that allows for faster access speeds as the element has already loaded when the user entered the URL. In addition, a “forgot password” feature has been implemented so that users can access their accounts even if they don’t remember their password. In our design we have not included this as many client will be able to have their password reset by users instead of remotely.

Sign in Page

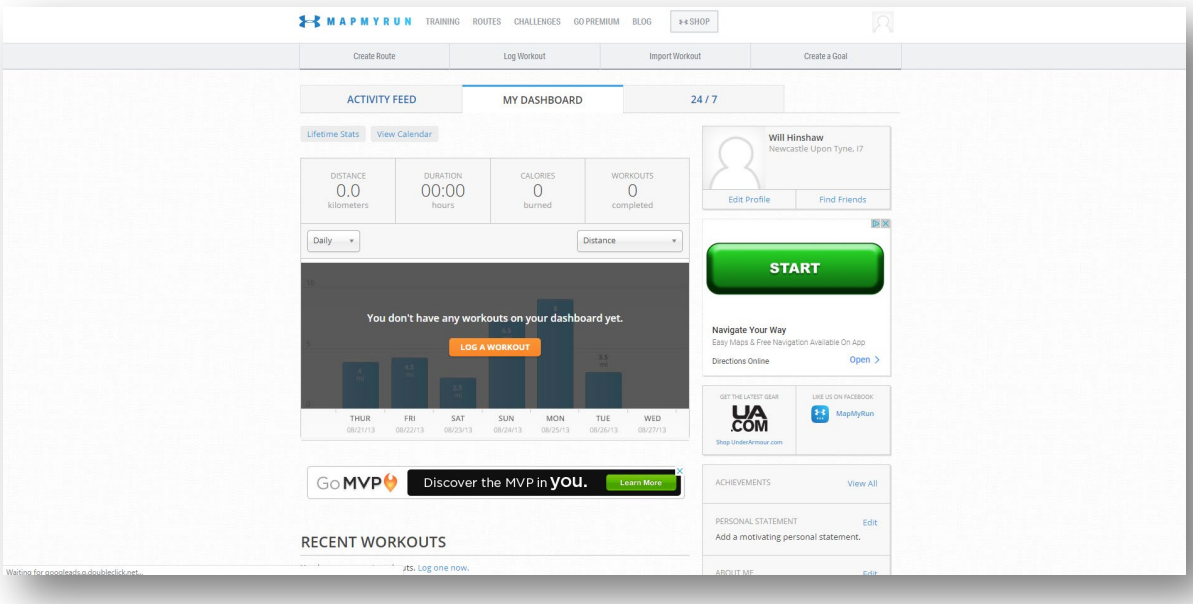
There are no differences within the overall layout of the MapMyRun design. However, the sign-up



criteria are different as the system is not based around a client and user platform. With this in mind, we

have included a “class code” requirement in the sign-up process so that the system can validate the user and the school they attend. This adds another layer of security to the system as the general public will be unable to access the features.

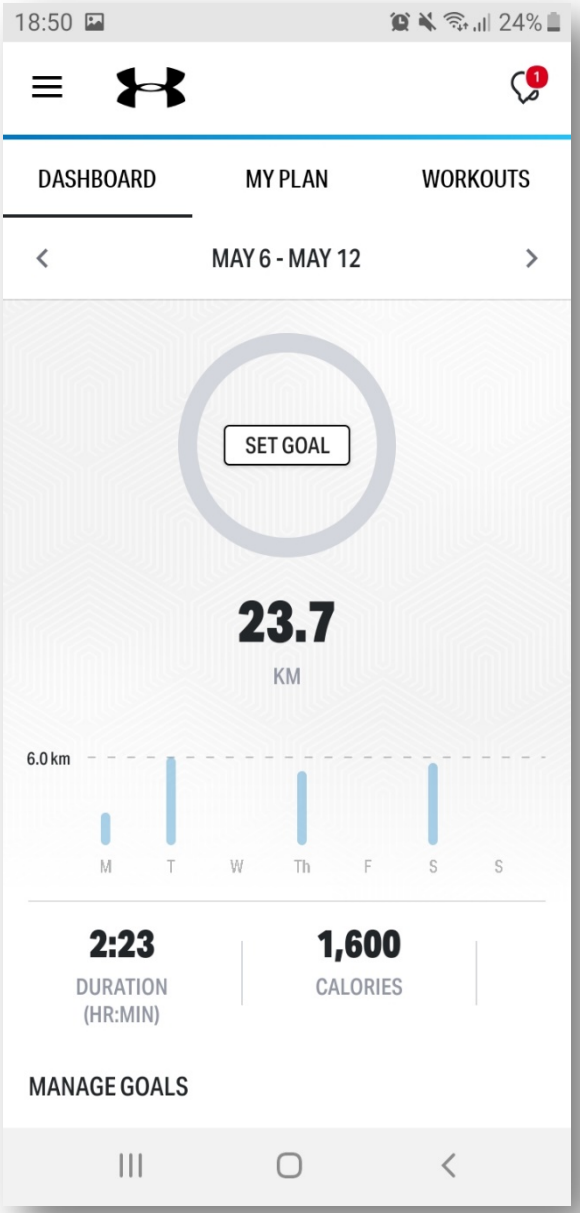
Dashboard



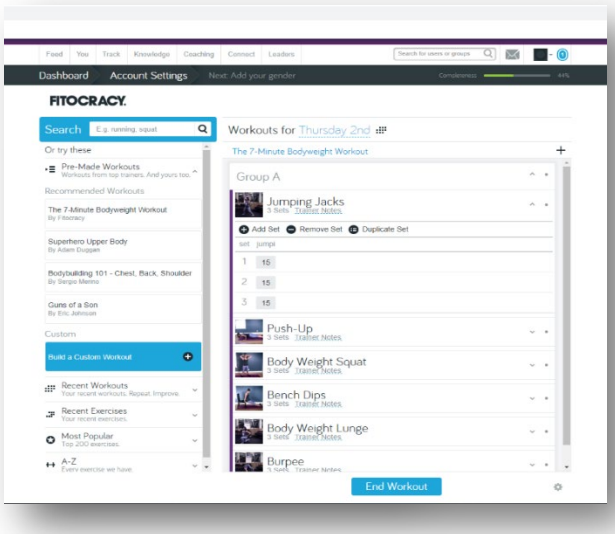
The overall dashboard of MapMyRun is entirely different as the user is able to access their profile settings without going to a drop-down menu. The user can view data directly as they login to the system without navigating to a menu and a second navigation bar has been added so that the user can directly access features such as “Create Route” or “Log Workout” without looking through the page to find these features. In our design the user has the ability to navigate all options through the navigation bar at the top of the page, which allows the user to remain orientated on the where they are in the website . This is a key design feature of a website as a user may become frustrated if they get lost, and have to navigate using the browsers “back” function. Within the navigation bar, the current page has been highlighted to further help the user locate themselves within the website.

In addition, the page includes adverts, which can clutter the page and make it feel unprofessional. On our design, the page doesn’t include any “bad” adverts that have no remote link to the website’s purpose. Furthermore, in the above diagram, there were no pop-up adverts that restricted the functionality of the website.

Other pre-existing fitness applications dedicated to fitness in general include – UA's MyFitnessPal, which allows users to log their food intake as well as a range of predetermined sports, and calculates their overall calorie needs, intake and expenditure, as well as providing guidance on macros and showing complex statistics on your eating habits. However, it has no facility for users to create workouts based on strength exercises or even just to track the exercise itself, as the application is focused on calorie usage. However, it is a very popular app with a very easy to use UI, depicting each day on one screen and allowing the user to flick between. The guidance from its interface found it to be the easiest to use. Most of these general fitness applications also have premium features only accessible to those who pay, usually the features that allow users to customise their experience more. MyFitnessPal also interfaces with many applications such as the preinstalled Samsung Health, other UA apps such as MapMyRun or MapMyRide, and wearable technology such as heart rate monitors and shoes with GPS or accelerometers inside.



FITOCRACY COACHING



This website was particularly interesting, you can sign up for free and it provides you with workout programmes and personal trainers you can select depending on what your goals are. Found it easy to navigate, you can also view other members' progress and what problems they have encountered on a live feed. Once have selected your workout session you can then log your progress as shown below. Found this to be very valuable.

Activity A (ii)

I have been asked by the owner of ToKa Fitness to develop a digital system that will:

- provide information on fitness training
- provide help on healthy living
- provide access to digital content
- provide to support customers with their training and healthy lifestyle
- encourage existing customers to use more of the services provided by ToKa Fitness
- allow customers individual accounts
- provide eating plans.

The Purpose of the App

In the context of our app, the purpose of the site is to store records of physical activities that are collected by wrist worn devices that can be accessed by a user or an instructor remotely or in class to view activity data at different levels (i.e. individual level or class level). In addition, the project team wanted features such as setting challenges and making the data available in different forms, especially for use in teaching of other subjects (e.g. maths, English etc.). Furthermore, the interface should show different level of details when used in class by the instructor/user than when used by them remotely.

Target Audience

Designing an app to make it "aesthetically pleasing/state of the art" is not the first port of call, if it doesn't appeal to the target audience then there is no point in having a app at all. In the context of this design brief, the app is targeted at clients and users who wish to store and keep track of physical activities. Developing an easy-to-understand and user-friendly app is a crucial part of any business as it defines the appropriate target audience – its needs, wants and expectations of the app.

Aims and Objectives (ToKa Fitness Brief)

The app is designed primarily to store and show physical activity data in an easy-to-read format both for instructors/users and clients who use the ToKa Fitness wrist worn device. It will allow for users to keep track of client progress and challenges with the ability to showcase class data to a class without revealing personal data or naming underworking clients.

A further possibility of the use of the app is with smaller personal training groups as the instructor can monitor progress and possibly motivate a certain group member to do more activities and set realistic challenges.

A further goal of the app is to educate clients of the importance of personal motivation and fitness in their life.

Functional Design Choices

The following are the main functionality design choices that should be considered when designing an app.

- Should be easy to navigate – Without a navigation system the user can become very lost within a complex app, in addition the navigation should allow the user to easily access part of the app without having to search through the page for the link.
- Have a clear indication of where the user is on the App – An indication of the location of the user can help to decide the user's next move , if, for instance, the user was in a shopping basket and no indication was present, then the user may not realise that clicking on "Buy Now" will process the transaction without their knowledge.
- App should be easy to find/search for – The app should be found on one of the first searches, research shows that the first search receives 95% of the traffic, leaving 5% for the other apps. This means that the app in 1st rank will gain more customers/users, attracting a larger audience.
- App & Login Forms should be secure – Protocol **Secure,** which is used for secure communication over a computer network and protects against "man-in-the-middle" attacks, where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other.
- No adverts.

Legal Considerations

- Domain names– Usually they are registered in the name of the designer or developer and not in the name of the client. It is the client who has paid for the domain name and therefore the client should be the owner or registrant.
- Ownership– The creator/designer of the app is by default the holder of the copyright and therefore the designer is the owner of the app, unless decided by the client.
- Content– Protection of the designer from the risks associated with clients giving them information or content that they want to include with the design when they are not entitled to do so. This may be illegal content or something that infringes the rights of the third parties.
- Data protection– It is under UK law that the app notifies the user of what you intend to use the personal data that you collect for.
- Cookies– Every mobile browser, just like desktop browsers, has different cookie settings and handles first party and third party cookies differently. Cookies also exist within apps when a browser is needed to view certain content or display an ad within an app. However, the cookies are completely "sandboxed" in apps. There is a requirement under the amended E-Privacy Directive and in the UK.

Regulations to:

- Inform users about cookies and what you are going to use their information for

- Obtain their consent to the use of the cookies.

Currently, gym users who want to track specific details at the ToKa gym are unable to do so for their workouts and exercises, often they have to do this on paper so they can track what they have completed during that workout session. However, this approach means that it's very difficult to quantifiably track their results – the client, ToKa Fitness therefore wants an application allowing users to track and log their activity at the gym. The application would allow users to enter data such as the exercise, weight, reps and sets the user completes for each workout, so that the user can track their progress long term.

This project lends itself to a computational approach, partly because of the convenience of tracking details by your phone or smartwatch (removing the need to carry pen and paper, and allowing users to track the details immediately, reducing the risk of errors), and partly because a computational solution will allow automatic progress tracking as it can automatically calculate and display your progress from the previous input data. This will make tracking users' gym progress much easier and provide them with details of their progress, which would be very difficult and time-consuming to recalculate themselves every week.

The key performance indicators (KPIs) and user acceptance criteria for the proposed ToKa Fitness app offers personal training sessions and advice on fitness training and healthy living to its customers and would like an app for their specific requirements. ToKa Fitness's specific requirements are that the proposed solution:

- has free access with some accessibility to services
- customer section for paid content to access full services
- accessibility features for users with sight loss
- link to 'social media' features
- ability to add new workouts and eating plans.

Other areas to consider are:

- the advice given to reduce health issues or injuries
- level of detail of instructions in a fitness training plan
- user experience to promote company image - visual assets and content
- complying with relevant legal regulations and guidelines
- compatibility across different devices, Android and iOS
- privacy and security of user data.

Accessibility issues to be considered:

- clear and readable interface and branding
- copyright
- access for people with disabilities
- error messages to also have an error icon.

My proposed solution should provide ToKaF itness with a professional and easy-to-use system that promotes their business and encourage customers to use it. All customers would have their own account. 'Free member' will have a basic level of accessibility and 'Full member' with a monthly fee for more complex features and functions.

Usability Features

The design of the above system features a lot that makes it accessible to many people. It has a simple layout with minimal options, meaning that the boxes/areas and text for everything in it can be large enough for those who have sight problems to read. The text displayed would also have to be actual text rather than images containing text so that it works with text to speech software. The fonts used should look professional yet easy to read. There is no audio that must be heard in order for the system to work as intended, as although the alert will make a sound when triggered, it will also pop up on screen and persist until dismissed.

There will be use of colours (uncertain about which colours at the current time) to break apart the sections and make it easier to navigate. Although the colours would be something to consult my client about, nothing too bright should be used, so greys, blacks or maybe muted blues and purples would be a good choice to make sure that everything is broken apart quite nicely but it isn't too much that it becomes hard to look at.

The program's basics will function the same as any other so the user has a sense of familiarity. The cross to close the window will be in the top right corner (although due to the nature of the program it will still run in the background) and familiar icons will be used to indicate features such as the drop-down boxes or an input text box. This will prevent the user from becoming confused.

Member
Private and secure customers area – password protected
Ability to customise some workouts and eating plans
Use of daily food planner with limited food analysis and reports
Use of daily fitness training planner with reports
Access to some fitness training/recipe videos
Links to social media/blog/forums
Use of daily food planner with food analysis and reports/graph analysis
Use of daily fitness training planner with reports/graph analysis
Access to all fitness training/recipe videos
Link to social media/fitness training videos/recipes/blog/forums
Detailed analysis of daily food plan and health plan

Customers would have access to a daily fitness training planner and videos that have been verified by sports professionals and doctors. It is important that the videos and planners are safe to use and provide the correct advice and guidance.

The setup of the customised eating plans will also have to be verified by a nutritionist as they must meet the correct and current nutrition guidance. If the nutrition guidance is incorrect, the member could get very ill as the result of following the customised eating plan, and ToKa Fitness could be sued and get a bad reputation and go out of business.

Members should be able to customise a workout and eating plans, depending on the customer's membership level. If the customer's experience of using the app is good, they will continue to use it and promote its use to their friends and on social media.

The app will have to be secure because of the member's privacy and the security of user data to comply with legal requirements such as data protection legislation and food and nutrition regulatory guidelines. Customer details must be secured by password protection and clear terms and conditions. The interface should be easy to use and navigate, be accessible across different devices and have accessibility features for users with sight loss.

User requirements

ToKaFitness has commissioned my software development company to develop a digital system. ToKaFitness offers personal training sessions and advice on fitness training and healthy living to its customers and would like an app for their specific requirements.

ToKaFitness's specific requirements are that the proposed solution:

- has free access with some accessibility to services
- customer section for paid content to access full services
- accessibility features for users with sight loss
- link to 'social media' features
- ability to customise workout and eating plans.

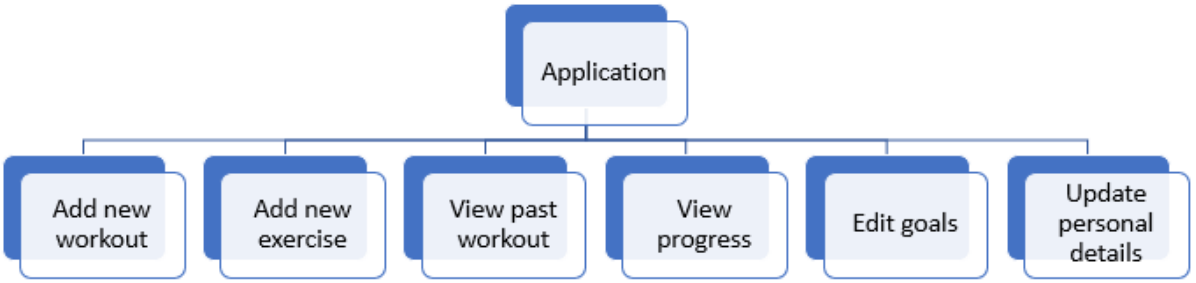
The target user group is adults, male and females. Due to legal and ethical nature of apps, it might be difficult for under 18s to follow the exercise and enter the correct details and follow the food and recipes correctly. To comply with age restrictions and guidelines, clear advice will be given to reduce health issues or injuries.

The customer area will be secure and accessed via a password and payment made in a secure platform. This will comply with legal requirements. The privacy and security of user data is important for each member to have access to their area via a secure login.

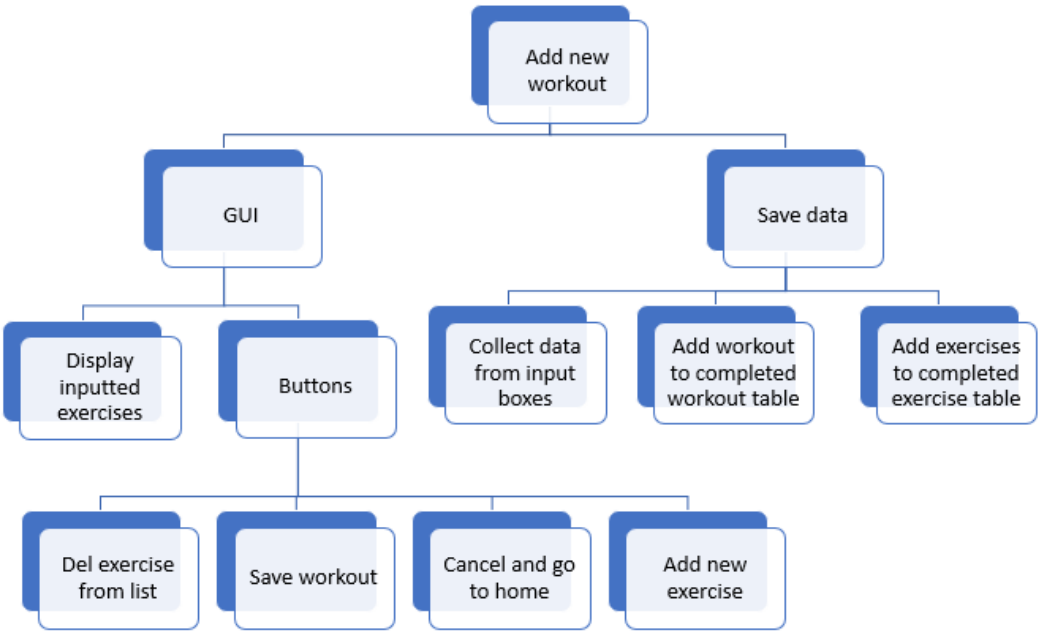
Each customer will be able to customise workout and eating plans and generate a report on their progress for specific periods, and the level of detail of instructions in a fitness training plan will vary to allow the customer to meet their identified goal.

All links to external sources will take consideration of copyright and intellectual property and licensing requirements.

Decomposition



The project consists of 6 main features - the ability to add a new workout and exercises , view previously entered data, view progress, edit goals and update personal details. These 6 comprise the main body of the application, as identified by my research as the most important features to include. Each will consist of its own activity and functions, being self-contained. I have not included optional features I will probably not have time to implement, like custom exercises or Facebook integration. Each of these 6 main features can be broken down into their component parts.



This is a breakdown of the component parts involved in adding a new workout. It's split between the GUI- containing display elements including a list of inputted exercises and buttons- and saving the data, which contains 3 main parts. The input data has to be collected from the input boxes and stored in local variables. Then I will use SQL to add a new record to the workout table, and any exercises to the

exercise table that are associated with this workout. The appropriate relational links will also have to be made. A series of buttons and associated tasks will be necessary, including the ability to cancel the input, delete an exercise and navigate to add new exercises to the workout page.

Validation

In terms of user input, there will be a handful of drop-down menus, and lots of buttons, which will require no validation as it will be restricted to correct input.

The input exercises will be limited to the exercises included in the database initially (although may expand on that and add custom exercises if it is feasible) and will have stored what type of data each exercise requires, e.g. sets, distance, weight etc., so that the only type of error possible is users entering the wrong values in their data. Add some validation into these inputs to ensure users don't enter unreasonable values like lifting 400kg .However, by restricting the user to pre-entered exercises can at least ensure consistency between workouts.

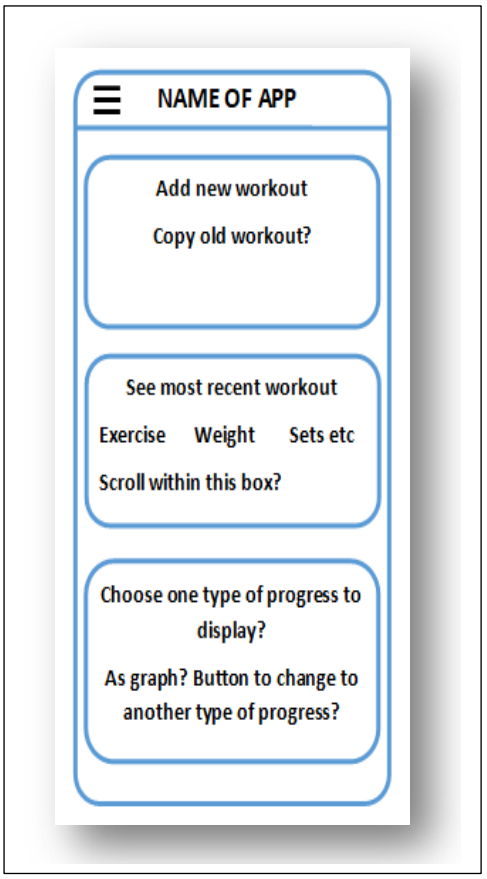
Most numerical input can be checked to see if the range is correct, but other than that it will be very difficult to validate- entering the wrong input will result in errors in the progress page but allowing users to edit past workouts should allow them to change any wrong input themselves when they notice wrong results in the progress page.

The best enforcements of validation that I can make to ensure incorrect data will not crash the application is to include a presence check on every input box and a type check on every database insert.

Activity B
Visual/Interface Designs

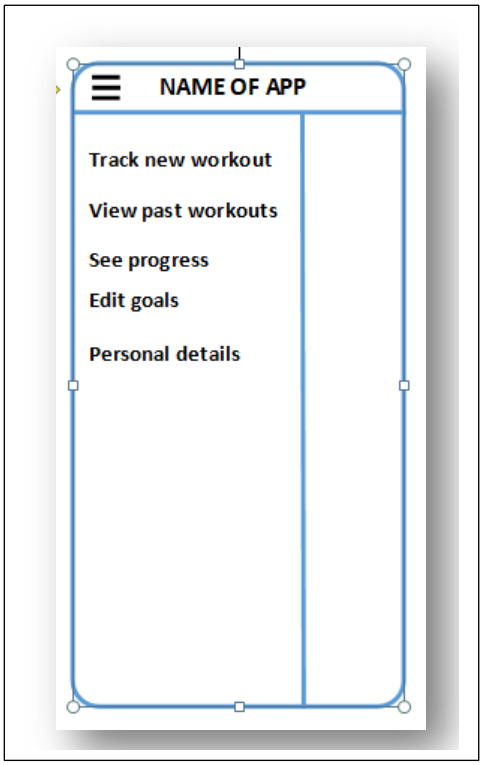
User Interface

HOME PAGE

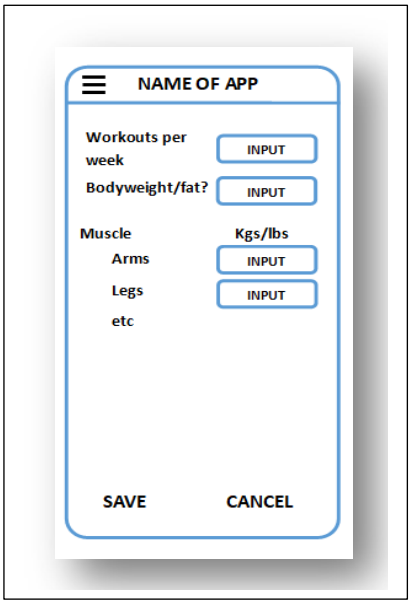


The home page is not required to do any major tasks, and is designed to have a simple display and allow users to easily access other parts of the application.

When the 3 black lines in the top left corner are pressed, the options menu will expand- this is a standard design used by most Android applications and so will be familiar and intuitive to the user. On this menu, the user will be able to switch between activities, such as to track new workouts, view past workouts, see progress, edit goals and any other activities subsequent development of the project require. All activities and features should be accessible through this menu.



Moving to the edit goals activity in the menu should bring up this page. In my market research, the most common goals potential users were working towards were orientated around how often they work out, how much they can lift, and body measurements such as size, weight and fat. Should also be an enable feature of these on the goals page, with varying input options? Next to each goal type will be an input box, allowing the user to enter numbers as appropriate, and save or cancel their changes to the relevant database

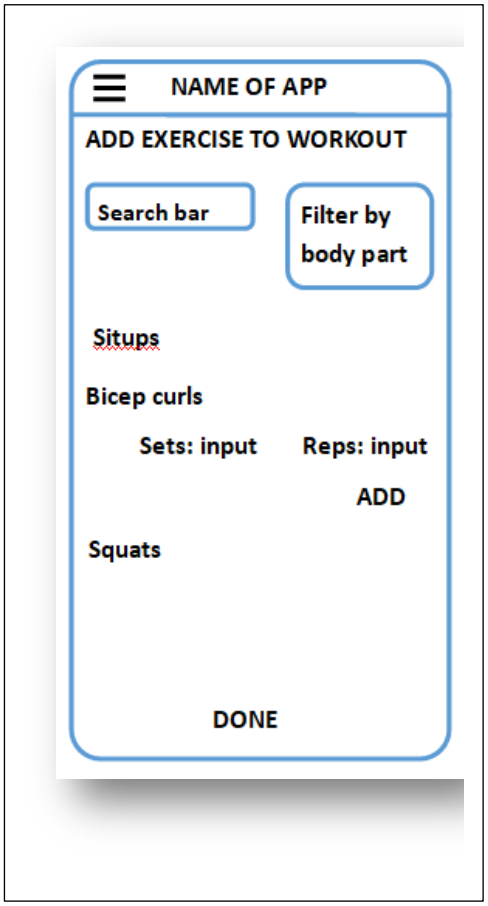


This is where users will log new workouts into their exercise log. This page is accessible both from the home page and from the menu.

Under the page title, an option for users to load previous workouts should be available. The day text next to it should allow the user to select a day from the past, and the workout associated with that day should be loaded automatically into the data below to reduce the difficulty in entering lots of repeat data into the workout.

Underneath, I have displayed some sample data of a workout. It displays the exercise name, and relevant data associated with that exercise, for example sets, reps, weight, time or distance.

The add button underneath allows users to add a new exercise to that workout (navigates to an entire new activity for this) and there is the option to save or cancel the user's input.

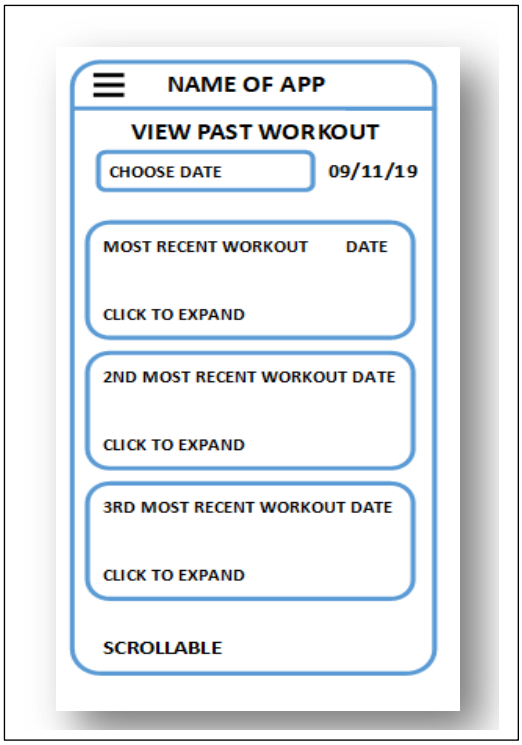


Here, the user can search by name, and filter by body part to select the exercise they want to add to a workout. This is scrollable to include dozens of exercises.

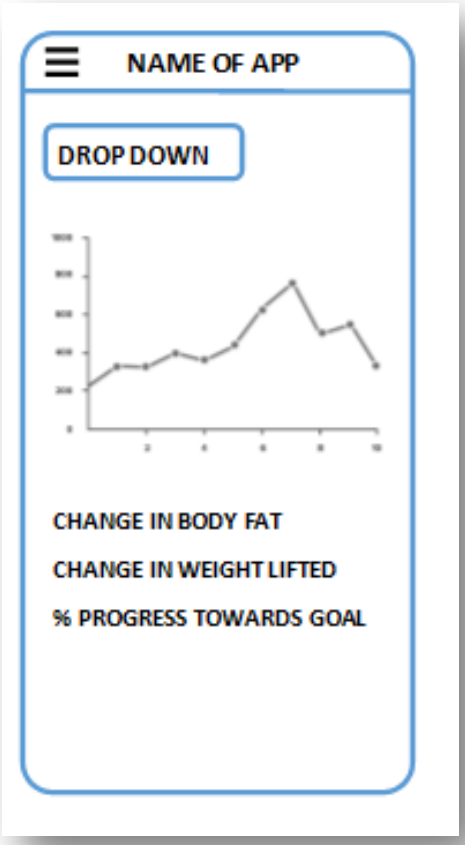
Once the user has selected an exercise, it expands to allow them to enter the relevant details. The add button will add this exercise to the workout but remain on this activity so the user can easily multi-add lots of exercises. The done button will then return the user to view the entire workout.

To allow users to access all their past data, it will display workouts in mini, potentially scrollable boxes within the activity. The activity can either be entirely scrollable or allow the user to move to the next set of 3 activities. If the users want to access a specific workout, they can select the date and it will come up below.

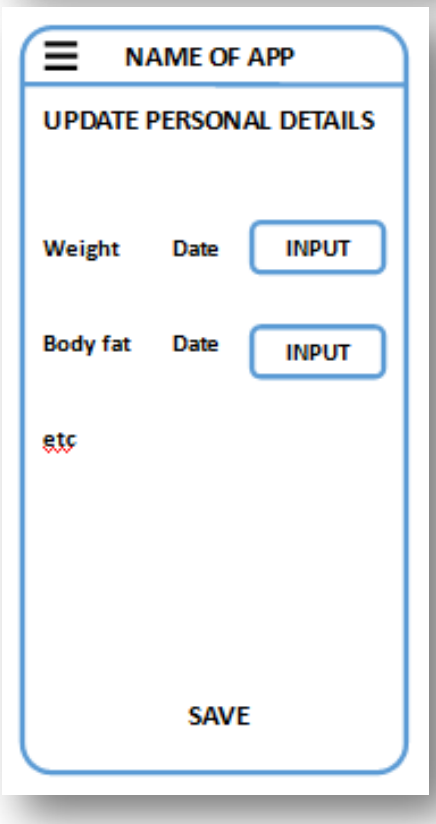
A preview of each workout should come up in each box, and if the user clicks on it should expand into a new activity with all the details about that workout.



SEE PROGRESS and PERSONAL DETAILS



Progress can be viewed in several formats, determined by the drop-down menu. The drop- down menu will include things like days worked out (in calendar form), % increase of weight lifted over time (by body part) and change in body weight.
Underneath I can include quantitative progress like change in body fat since initial value and progress made towards personal goals.



This page allows users to update data that is not part of a workout, e.g. body measurements like waist size, weight, body fat or any other appropriate measures. This data will be stored and used on the progress page.

Requirements	Requirements accessible in UI?
Allow users to input exercise, weight, reps, sets	Yes- add new exercise activity
Allow users to input personal details	Yes- personal details activity
Allow users to change personal details	Yes- personal details activity
Display past workouts	Yes- view past workouts and workout details activities
Display progress	Yes- see progress activity
Allow users to modify previous workouts	Yes- view workout details activity
Intuitive UI	Yes
Calculate some statistical analysis on input data, e.g. change in weight lifted over time, change in 1RM.	Yes- see progress activity
Allow users to see cardio and strength workouts in conjunction	Yes- view past workout activity
Provide features completely free of charge for all Android users.	Yes

Usability Features

My main criterion for usability of design is that each feature is clearly signposted and can be completed with a limited amount of clicks to ensure the system is efficient.

Using a standard design for the menu page means users will instinctively know where to go to find features they would like to use, and consistency of the design of each page (i.e. same style of inputs) will allow users to very quickly understand how each activity functions. Each activity has a clear heading and access to the menu page so users know they can return to any point in the application quickly and easily.

The design of the application has been done with the intention of having a designated activity for almost every task, so that each page can be kept as simply as possible and it is easy to signpost the user to each feature. This means that users can access each feature through the menu system and as a result reduces the clicks needed to access each feature and improves time efficiency for them to input their data after each workout – this satisfies my success criterion. The consistency and limited objects on each page mean the user will always know how to navigate the application and the application should have excellent visual clarity, if you limit the application to a handful of colours and use consistent fonts and font sizes. Having links to essential pages also on the home page is good practice in terms of usability. Keeping a menu button on the top bar of every activity will always allow users to get back to where they were if so needed.

Additionally, as the colours used in XML elements in Android Studio are defined in a values file, it's easy to define the few colours used in a colour scheme there and have the application use them automatically. This makes the application easier to use as each activity is consistent.

Algorithms and Pseudocode

As Android applications are event-driven, i.e. driven by the users actions rather than the results of previous code, the algorithms given below are used when an application is triggered to create, or when a button is pressed, for example. Not detailed below are the parts of the application that handle this, such as setting listeners for the user's actions- these trigger the algorithms given below.

As each activity is designed to solve a particular solution requirement(s), it is easy to develop the application iteratively- each iteration will involve the solution of one or more solution requirements, in the full development of one or more activities.

Add new workout

These are the algorithms associated with add new workout activity.

```
PROCEDURE build()
      CREATE textObject (text="ADD NEW WORKOUT")
      CREATE textObject (text="Load previous workout")
      CREATE buttonObject (text="Day", onclick=expand())
      CREATE buttonObject (text="SAVE", onclick=save())
      CREATE buttonObject (text="CANCEL", onclick=cancel())
      displayInput()
ENDPROCEDURE
```

```
PROCEDURE displayInput()
     IF activity launched from start THEN
          DO NOTHING
     ELSEIF day clicked in calendar THEN
          //DO STUFF
          //verify workout on that day exists
          //get all exercises associated with that days workout
          //input data to displayInput()
     ELSE
          ARRAY data = GET data from "add new exercise activity"
          data.append(previously saved data from this activity)
          FOR i between 0 and length(data)
               CREATE textObject (text = data[i][exerciseName])
               CREATE textObject (text = data[i][exercise stats])
               CREATE buttonObject (text="DEL", onclick
               =del(exerciseName))
          ENDFOR
     ENDIF
ENDPROCEDURE
```

This procedure is run on launch of the activity- it builds the GUI of the page and defines its behaviours. It will create each GUI object such as text or buttons and defines which procedures buttons call on click. It also calls the **displayInput** procedure, which displays any exercises and corresponding input that has been sent by add new exercise activity. This is a separate procedure as it may be called several times over the life of the activity.

```
PROCEDURE add()
      STORE input until activity relaunch
      CLOSE activity
      LAUNCH "add new exercise" activity
ENDPROCEDURE
```

The **displayInput** procedure displays any exercises that have been selected as part of the exercise. If the activity is launched from the menu, then none is displayed. If the activity is launched from the add new exercise activity, it will display data passed by that activity and any saved while the user is selecting this input by looping through the data and creating text objects and button objects for each exercise input.

When the user wants to add more exercises to their workout, they can press the add button, which calls this procedure on click. It stores any previously input data until the user navigates back to this page, closes the activity and launches add new exercise activity so the user can input more data.

```
PROCEDURE cancel()
      DELETE all input data
      CLOSE activity
      LAUNCH home page or last open activity
ENDPROCEDURE
```

If the user decides not to input this data, they can press the cancel button, calling this procedure. It deletes all saved input data, closes the activity and navigates to the home page.

```
PROCEDURE expand()
      DISPLAY calendar
      IF user clicks on date THEN
            displayInput(date)
ENDPROCEDURE
```

This procedure is also optional- if the user wants to auto-input data from their last workout, they can press on the “Day” text and it will call this procedure, which displays a calendar and calls the displayInput procedure if the user does select a day.

```
PROCEDURE save()

      CONNECT to db

      SQL1 = insert into completedWorkouts (Date, TotalTime) values
      (CurrentDate, TotalTime IF INPUT)

      DO SQL1

      SQL2 = SELECT ID from completedWorkouts where Date =
      CurrentDate

      WorkoutID = DO SQL2

      FOR i between 0 and length(data)

            SQL3 = SELECT * from ExerciseBank where ExerciseName =
            data[i][exerciseName]

            ARRAY ExerciseData = DO SQL3

            ExerciseID = ExerciseData[1]

            ARRAY stats = array of all statistics

            FOR i between 2 and length(number of possible
            statistics)+2

                  IF ExerciseData[i] = False THEN

                        stats[i].delete

                  ELSE

                        DO NOTHING

                  ENDIF

            ENDFOR

            SQL5 = insert into CompletedExercises (WorkoutID,
            ExerciseID, Relevant attributes) values (WorkoutID,
            ExerciseID, data[i][statistics])

            DO SQL5

            ENDFOR

      DELETE data

      CLOSE activity

      LAUNCH home

ENDPROCEDURE
```

```
PROCEDURE del(ID)

      DELETE all objects of exerciseName

      DELETE from data internal array of exerciseName

ENDPROCEDURE
```

Each displayed activity will have a delete button next to it to aid usability of the application. The delete button calls this function with its own ID, which then deletes all objects associated with it (i.e. the exerciseName text object, statistics) and deletes the data from the internal array of inputs. This allows users to delete one input exercise without having to delete and re-input everything else.

The save procedure comprises the main bulk of the code for this activity. When the user has finished inputting their data, they press the save button, which calls this procedure. The procedure must then create a new workout in the workouts table, then get the ID of the new record to be used later. Then it loops, so that it can check in the exercise bank which statistics are applicable to that exercise, stores the relevant attribute names, and gets the exercise ID, so that for every set of data in the array the procedure will add a new record to the exercises table, entering data into the relevant statistics and null values into the others. Then the procedure deletes any stored data, closes the activity and reopens the home page.

Add new exercise

These are the activities associated with add new exercise activity.

```
PROCEDURE build()

      CREATE textObject (text="Add exercise to workout")

      CREATE inputObject (text="Search bar",
      onsearch=displayExercises("search",input)

      CREATE dropDownObject (text="Filter by body part",
      onclick=displayExercises("filter",input)

      displayExercises(all, all)

      CREATE buttonObject (text="Done", onclick=done())

ENDPROCEDURE
```

The build procedure is called on launch of the activity, to build the UI. It creates the objects needed, then calls the **displayExercises** procedure to display all exercises. The **displayExercises** procedure is also called from the search bar or drop-down menu if the user inputs any preferences on the data they want displayed.

```
PROCEDURE done()
      CLOSE activity
      LAUNCH add new workout activity (data)
ENDPROCEDURE
```

This procedure is called on click of the done button and sends the user back to add new workout activity, passing all the input data to it.

```
PROCEDURE displayExercises(type, data)
//type corresponds to where input comes from
//if exercises are all displayed, or filtered by something
//what they're filtered by is in parameter data
     CONNECT to db
     IF type == "all" THEN
          SQL = "Select * from ExerciseBank"
     ELSEIF type == "search" THEN
          SQL = "Select * from ExerciseBank where ExerciseName =
          data"
     ELSEIF type == "filter" THEN
          SQL = "Select * from ExerciseBank where BodyPart = data"
     ENDIF
     exercises = DO SQL
     FOR i between 0 and length(exercises)
          CREATE textObject (text=exercises[i][1], onclick =
          expand(exercises[i][1])
     ENDFOR
ENDPROCEDURE
```

The **displayExercises** procedure is responsible for displaying the exercises users may want to add to their workout, depending on any previous input. If it's called on launch of the activity, all exercises are displayed. If it's called from the search bar or drop-down menu then the SQL is modified to query the exercises related to the user's search and display the relevant exercises. The exercises are stored in the exercise variable then iterated through to create all the relevant text objects.

```
PROCEDURE add()
      exerciseData = GET input from all fields
      data.append(exerciseData)
ENDPRCOEDURE
```

The add button gets all the input data from the fields of that exercise, and adds it as an array to the data variable, which will store all input data while this activity is running, and be passed back to the add new workout activity when the done button is pressed.

```
PROCEDURE expand(exerciseName)
     CONNECT to db
     SQL = SELECT * from ExerciseBank where ExerciseName =
     exerciseName
     ARRAY ExerciseData = DO SQL
     ARRAY stats = array of all statistics
     FOR i between 2 and length(number of possible statistics)+2
          IF ExerciseData[i] = False
               stats[i].delete
          ELSE
               DO NOTHING
          ENDIF
     FOR i between 0 and length of stats
          CREATE textObject (text = stats[i])
     ENDFOR
     CREATE buttonObject (text="ADD", onclick=add())
ENDPROCEDURE
```

Once all the exercises have been displayed, they will need to expand to show input boxes for the relevant statistics so that the user can input their workout stats. The procedure gets the data for that exercise from the exercise bank and displays an input box for each relevant stat. It also creates a button ADD for when the user wants to save this data.

Edit Goal

```
PROCEDURE build()
      CREATE textObject (text="Muscle")
      CREATE textObject (text = "Kgs/lbs")
      ARRAY goals = ["Workouts per week", "Bodyweight", "Bodyfat",
      "Arms", "Legs", "Torso"]
      CONNECT to db
      SQL = select goals from personal details
      ARRAY currentValues = DO SQL
      FOR i between 0 and length of goals
            CREATE textObject (text=goals[i])
            CREATE inputObject (text=currentValues[i])
      ENDFOR
      CREATE buttonObject (text="SAVE", onclick=save())
      CREATE buttonObject (text="CANCEL", onclick=cancel())
ENDPROCEDURE
```

This procedure is called on launch of the activity- it creates text objects for the titles, and creates an array of the names of the goals. Then it connects to the DBMS, gets the current goals set, and iterates through the goals to create the text and input objects. It also creates the buttons at the bottom of the page to save or cancel the input.

```
PROCEDURE save()
      ARRAY newValues = GET data from input boxes
      CONNECT to db
      SQL = insert into Personal Details (goals columns) values
      (newValues)
      DO SQL
      CLOSE activity
      LAUNCH home activity
ENDPROCEDURE
```

The save procedure is called on click of the save button and gets all the data from the input boxes, updates the personal details table to include this information and exits the activity.

```
PROCEDURE cancel()
      DELETE all input
      CLOSE activity
      LAUNCH home activity
ENDPROCEDURE
```

This procedure just exits the application- it exists mostly to aid usability.

Validation

In terms of user input, there will be a handful of drop-down menus, and lots of buttons, which will require no validation as it will be restricted to correct input.

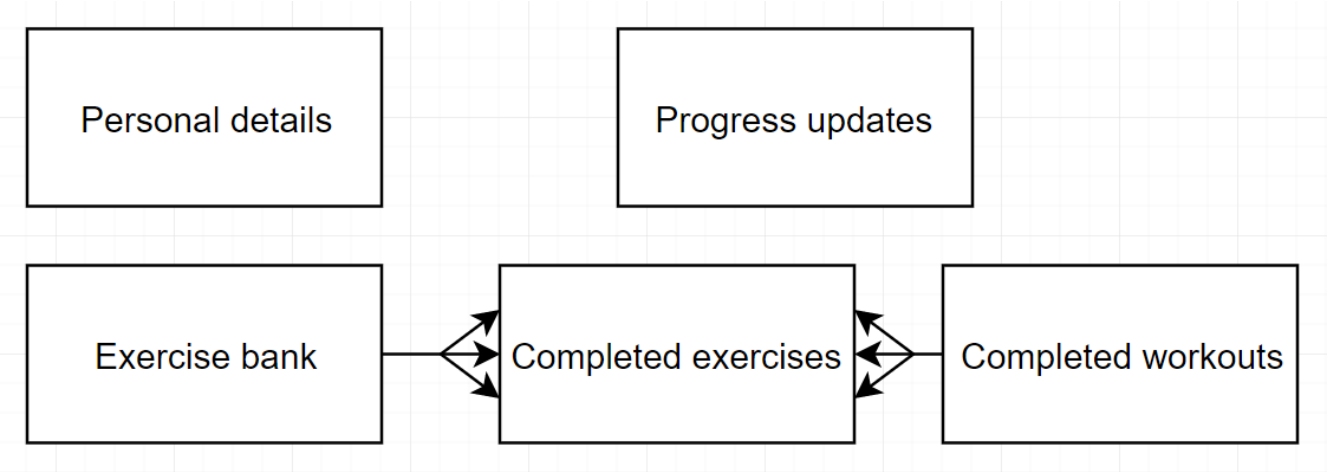
The input exercises will be limited to the exercises to include what type of data each exercise requires, e.g. sets, distance, weight etc., so that the only type of error possible is the user entering the wrong values in their data. However, by restricting the user can ensure consistency between workouts.

Most numerical input can be checked to see if the range is correct, but other than that it will be very difficult to validate- entering the wrong input will result in errors in the progress page, but allowing users to edit past workouts should allow them to change any wrong input themselves when they notice wrong results in the progress page.

The best enforcements of validation that I can make to ensure incorrect data will not crash the application is to include a presence check on every input box and a type check on every database insert.

Databases

Android Studio allows for the implementation of data storage using SQLite so this will be my primary method of storing all the users' data. As I have restricted the scope of the project to make it manageable, I will store all the users' data on their device and therefore do not need to store and query many users' worth of data. The attributes I need to store for each user are as follows.



These are the entities I will be including in my database. Two are simple stand-alone tables, while exercise bank and completed workouts both have a one-to-many relationship with completed exercises. This is because exercise bank and completed workouts would have a many-to-many relationship if this entity were omitted.

Personal Details

These are the basic details the user enters as part of their setup. They can be used to calculate stats such as calories burned and the user can update weight, body fat etc. as part of their progress to be measured. Weight and body fat are not stored in this table- in order to keep a record of previous measurements to be used for calculating progress, I will need to keep many values so the repeat data will be stored in a separate database exclusively for these values.

Attribute	Data type and restrictions
Name	String (32 char max)
Height (optional)	Integer/float. 0<height<250 cm
Gender (optional)	Char (M/F/N) or null
Date of birth (optional)	Date or null. Must be younger than 100, older than 10.
Goal weight (optional)	Float or null 30<weight<250.
Goal body fat (optional)	Float or null. Restrict to safe values,
Goal weight lifted (optional)	2d array of float and body part or null
Goal workouts per week	Integer

Exercise Bank

This will be set up by me so there will be no need for any validation of this database. Upload a sample set of exercises into it as part of my development process but can be expanded to include hundreds and potentially allow users to put their own custom exercises in it too (which would need validation).

Attribute	Data type and restrictions
ExerciseID	Integer (primary key)
Exercise name	String (32 char max)
Strength	Boolean (false = cardio)
Body area	String, e.g. arms, legs, abs
Sets	Boolean (if exercise is measured in sets) or null
Reps	Boolean (if exercise is measured in reps) or null

Weight	Boolean (if exercise is measured in weight) or null
Distance	Boolean (if exercise is measured in distance) or null

Time	Boolean (if exercise is measured in time) or null
Speed	Boolean (if exercise is measured in speed) or null

These are all the statistics that will be tracked, although the design of the database should easily allow me to add more if my clients request any others.

Each statistic is stored as a Boolean- whether that value is relevant to this exercise or not. On implementation, users will be allowed to enter values into the columns of the completed exercises table where there is a corresponding True in this table only.

Completed Workouts

There is a many-to-many relationship between workouts and exercises, normalised the database to include an extra table for completed exercises (which now restricts the relationships to one-to-many). The completed workouts table contains only the basic information about each workout. Exercises completed are stored in another table.

Attribute	Data type and restrictions
WorkoutID	Integer (primary key)
Day	Date
Total time	Time (could be same as exercise if workout consists of one cardio exercise) or null

Completed Exercises

There will be a one-to-many relationship between the completed workouts table and completed exercises table and between the exercise bank and completed exercises table. Each workout is made up of several exercises from this table. The presence of data in each attribute of this table is determined by the Boolean values in the relevant columns of the exercise bank table.

If clients decide that they want other data tracked than what has been designed, it will be easy to implement in the database. For example, if they want to track another type of stat for an exercise, it can be entered as an attribute in the exercise bank table with a Boolean value dictating when it is used and a corresponding attribute added to this table. This should not affect the normalisation of the database.

Attribute	Data type and restrictions
Primary key	Integer (primary key)
WorkoutID	Integer (foreign key)
ExerciseID	Integer (foreign key)
Sets	Integer
Reps	Array of length size(sets) of integers or null
Weight	Float. Restrict to reasonable weight

Distance	Float. Restrict to reasonable distance
Time	Time
Speed	Float. Do not restrict- different exercises will have different reasonable inputs

The data included with each exercise is dependent on what is relevant- therefore it is possible for these attributes (except for primary and foreign keys) to have a null value. Whether they contain a value is dependent on the Boolean data in the relevant attribute in the exercise bank (i.e. if the exercise bank signals a certain exercise is measurable in sets, the application will allow the user to enter the amount of sets completed. None of this data is mandatory).

UPDATES OF PERSONAL DETAILS/PROGRESS UPDATES

Attributes	Data type and restrictions
ProgressID	Integer (primary key)
Weight	Integer/float or null
Body fat	Percentage or null
Date changed	Date

EXPANSION OF PROJECT

If I need to expand the project and have all the users' data stored on an external server, I may have to modify these databases to allow for keeping each user's data separate. As this is not within the scope of this project, I have designed the database with one user per device in mind.

QUERYING DATABASE

Intend to query the database for several purposes- will need to get current details in order to display to the user their previously entered personal details and goals. Also need to get exercises from the exercises table to display to the user when adding their exercises to a workout and check if each attribute is relevant to that exercise.

Need to query the database to get data to calculate the user's progress. If, for example, by calculating their body fat change, will need to get the measurement and date of each measurement from the progress updates table, and either put each value into a graph or other visualisation, or calculate the total change and percentage change from the first and last pieces of data.

Testing

Approach to testing

To test my solution - white box testing (where I test the internal structure of the app) and black box testing (where I only test the inputs and outputs).

Algorithm	How to test it
Signing in	Use values not in the database Use some values in the database and others not in the database (e.g. correct username but wrong password) Use values in the database that are not related together (using a username but the password is for a different account) Use correct values in the database Use code to try and rewrite the checking program (e.g. use JavaScript code)
Signing up	Use correct values and datatypes Use correct datatypes but type different passwords for confirmation Use incorrect datatypes (e.g. use letters in telephone variable) Don't type anything in compulsory inputs Try to rewrite code using JavaScript or another programming language
Any validation	Use erroneous data (wrong datatype) Use correct data (right data type) Use code to try and change the validation algorithm (e.g. JavaScript)

Example of test to be carried out

Component to be tested	**Accessibility settings**			
Pre-requisites and dependencies	**Have access to the app** **Mobile device** **Computer**			
Description of test	**Test data to be used (if required)**	**Expected outcome**	**Actual outcome**	**Comments and intended actions**
App accessed on the mobile device		All pages of the app can be accessed on a mobile device		
App accessed on a computer		All pages of the app can be accessed on a computer		
Adjust settings for visually impaired users on the mobile device		Able to adjust settings to make the icons/links and text bigger and easier to use for visually impaired users		
Adjust settings for visually impaired users on the computer		Able to adjust settings to make the icons/links and text bigger and easier to use for visually impaired users		

Test Number	Test description	Expected result	Actual result	Successful ?
1	Open navigation menu from	Navigation menu	Navigation menu	Yes
2	Close navigation menu with back button	Navigation menu	Navigation menu	Yes
3	Open and close navigation menu by swiping	Navigation menu	Navigation menu	Yes
4	Navigate to home activity from home activity	Home activity launches	Home activity launches	Yes
5	As above, from track workout	Home activity launches	Home activity launches	Yes
6	As above, from view past workout activity	Home activity launches	Home activity launches	Yes

7	As above, from see progress	Home activity launches	Home activity launches	Yes
8	As above, from edit goals activity	Home activity launches	Home activity launches	Yes
9	As above, from personal details activity	Home activity launches	Home activity launches	Yes
10	Navigation menu closes when back button pressed	Navigation menu closes and activity remains the same	Navigation menu closes and activity remains the same	Yes
11	Navigate to track workout activity	Track workout activity launches	Track workout activity launches	Yes
12	Navigate to view past workout activity	View past workout activity launches	View past workout activity launches	Yes
13	Navigate to see progress activity	See progress activity launches	See progress activity launches	Yes
14	Navigate to edit goals activity	Edit goals activity	Edit goals activity	Yes
15	Navigate to personal details	Personal details activity launches	Personal details activity launches	Yes

Testing – Progress Updates

Test Number	Test description	Expected result	Actual result	Successful?
1	Add a more recent update	Most recent update is displayed on creation	Most recent update is displayed on creation	Yes
2	Add updates from several months and years	Most recent update is displayed on creation	Most recent update is displayed on creation	Yes
3	Set date 35th March	Display error message	Error message displayed	Yes
4	Set date 30th	Display error message	Error message displayed	Yes
5	Set date 2010	Display error message	Error message displayed	Yes
6	Set weight =5	Display error message	Error message displayed	Yes
7	Set weight =500	Display error message	Error message displayed	Yes
8	Set body fat = 0	Display error message	Error message displayed	Yes
9	Set body fat = 95	Display error message	Error message displayed	Yes
10	Press cancel	All values are removed	Input boxes are cleared	Yes
11	Press save	Data is added to entity	Data is added to entity	Yes
12-16	Leave each input box blank	Display error message	Error message displayed	Yes
17	Enter a letter	Not possible-keyboard	N/A	Yes

As this activity was rigorously tested during development, no issues in the post-iteration testing.

Testing- Initial Entry

Test Number	Test description	Expected result	Actual result	Successful?
1-11	Leave each input box blank	Displays error message	Displays error message then	No
12	Enter a number in name box	Displays error message	Displays error message then	No
13	Enter G in gender	Display error message	Displays error message then	No
14	Set weight = 5	Display error message	Displays error message then saves incorrect data	No
15	Set weight = 500	Display error message	Displays error message then	No
16	Set body fat = 0	Display error message	Displays error message then	No
17	Set body fat = 95	Display error message	Displays error message then	No
18	Press save with valid values	Data is added to entity	Data is added to entity	Yes
19	Enter goal workouts = 12	Displays error message	Displays error message then	No
20	Set date 42th March	Display error message	Displays error message then	No
21	Set date 30th February	Display error message	Displays error message then	No
22	Set date 200	Display error message	Displays error message then	No

As shown in the testing the logic of my validation, but the logic of the code is allowing the incorrect data to be saved. The same error has resulted in many failed tests and will need to be fixed before moving on to iteration 3. In addition, the error message on the phone is displaying on top of one of the input boxes due to the difference in screen size between that and the VM, so will need to readjust it.

Testing- Edit goals

Test Number	Test description	Expected result	Actual result	Successful?
1	Enter 55 workouts per	Displays error message	Displays error message	Yes
2	Enter weight = 12	Displays error	Displays error message	Yes
3	Enter weight = 500	Displays error	Displays error message	Yes
4	Enter bodyfat = 0	Displays error	Displays error message	Yes
5	Enter bodyfat = 98	Displays error	Displays error message	Yes
6	Press cancel button	Displays error	Displays error message	Yes
7-9	Leave each box blank	Displays error	Displays error message	Yes
10	Press save on correct data	Data is entered into entity	Data is entered into entity	Yes

Again, as this activity was rigorously tested during development, there are no more issues with the program.

Stakeholder Feedback

I asked a potential user a few questions after they had some time to use the application.

Q1) Do you feel the application is easy to navigate?

Yes- the swipe menu is very similar to most applications so you don't have to learn something different to use the app. And everything is the same sort of colour scheme so it's obvious where buttons and input boxes are.

Q2) Have you experienced any problems using the application?

The first version you gave me couldn't navigate away from the first screen, but after you fixed that, it's all worked fine. But lots of parts of the application are missing- entering workout data is the most important part and you haven't built that yet so there's loads of things I can't do.

Q3) Are there any other comments you have about the application at this stage?

I'm sure I've used apps before which can display a calendar to let you choose a date- that makes it much easier. Would it be possible to change that?

This survey is encouraging- all the completed parts of the application are working as intended and no unexpected errors have escaped testing. The issue of missing parts of the application isn't a problem as these will be developed in the next iteration

Task 2 Developing the System Prototype

Reviewing of the sources of data, information and code

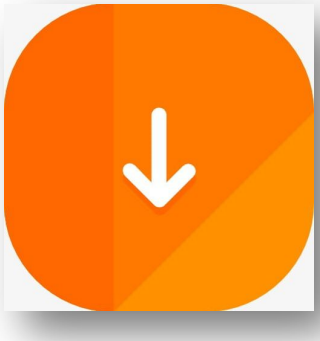
From the examination of the sources and assets gathered during Task 2, I have considered my options.

As the user requirements are clear and, due to legal and ethical implications of using the identified assets, all of the images are to be sourced from free imaging providers or providers that allow use for non-commercial purposes: "License: Non-commercial Use". Some of these providers require you to set up an account to download the images and some ask you to reference the source of the image.
Care was taken when selecting the images to use due to the legal and ethical implications of the assets. Great care was taken when selecting the external links to the health and fitness and food external links. All of the recipes were from a reputable source and promoted healthy eating and most identified the nutritional value of the meal. They were easy to follow with clear images of the prepared dishes.

The health and exercise video and information links were sponsored by medial and personal trainers and provide help and guidance as customers will be using these videos unsupervised by health and medical experts. All of the links hosted fitness videos of a very professional quality and level.

From the research of possible code snippets, the only code that I am going to modify is "chat room and blog example code".

Assets selected and rationale

Image	Source	Rationale
	https://www.freepik.com/free-icon/search_788138.htm	Users see this search image more than the other option and it is also uses fewer colours.
	https://icons8.com/icons/set/settings	Will use the traditional settings icon as all users know what it is.
	https://imgbin.com/png/wfLgfRPc/computer-icons-mobile-phones-png	This is my option for "no access to members' area". Users may find the other option confusing and it has too many colours.
	https://imgbin.com/png/Jm6mwH7T/a-logo-png	Decided to use this icon as it's free for non-commercial use as the other option needs an extended licence. It also stands out.
	https://www.flaticon.com/free-icon/facebook-logo-button_69407	Used this social media logo as users will be able to identify it quickly.

	https://www.stickpng.com/img/icons-logos-kemojis/tech-companies/whatsApp-logo	Used this social media logo as users will be able to identify it quickly.
	https://www.stickpng.com/img/icons-logos-emojis/tech-companies/twitter-logo	Used this social media logo as users will be able to identify it quickly.
	https://www.stickpng.com/fr/img/icones-logos-emojis/societes-de-technologie/logo-youtube-play	Used this social media logo as users will be able to identify it quickly.
	https://www.flaticon.com/free-icon/instagram-logo_69366	Used this social media logo as users will be able to identify it quickly.

Contact us example code	This is the only code that I am going to modify. I am going to develop my code for the app to ensure that it efficient and robust.
https://github.com/tmont/sunlight/commit/5ffc4edba355471434d2d75506dc3c3e52d9730a	
https://github.com	Has an extensive library of code that I can use if needed.
https://github.com/displylogo	Will display logo at location of the code.
<a href="https://www.freepik.com/free-photos-vectors/logo">	

Possible
Links to external sites

https://www.bbcgoodfood.com/recipes/collection/healthy-dinner	BBC good fool link	Decided to link to these external resources, they are all verified by the medical profession and have excellent reviews on social media.
https://lovefoodhatewaste.com/	Love food healthy recipes	
https://www.ketocustomplan.com/	Kenco diet plan	
https://www.foodsavvy.org.uk/	Foodsavvy healthy recipes	
https://www.delish.com/cooking/	Healthy recipes	

https://ifoodreal.com/clean-eating-recipes-dinners/	Healthy recipes	The recipes featured have an image of the food and clear recipe and methods. Most also have user reviews for each of the recipe. They are reliable sources of information to link to.
https://bitesofwellness.com/	Healthy recipes	
https://www.everydayhealth.com/	Health and fitness apps	
https://www.health.com/fitness	Health and fitness apps	
https://www.health.harvard.edu/topics/exercise-and-fitness	Health and fitness apps	
https://www.pinterest.com/reachyourpeak/health-fitness-topics/	Health and fitness apps	

Blogs:

https://www.quora.com/What-are-your-top-5-favorite-health-and-fitness-topics	Fitness blog	Will provide further options for customers to select their own topics.
https://www.wix.com/	Wix blog	These are the top 3 blogs in the sector and have been established for some time and have a good following and lots of posts.
https://bitesofwellness.com/blog/	Bites of Wellness	
https://blog.feedspot.com/uk_fitness_blogs/	Top 10 blogs	

Prototype for the proposed digital system

ToKaFitness system prototype has commissioned my software development company to develop a digital system. ToKaFitness offers personal training sessions and advice on fitness training and healthy living to its customers and would like an app for their specific requirements.

ToKa Fitness specific requirements are that the proposed solution:

- has free access with some accessibility to services
- customer section for paid content to access full services
- accessibility features for users with sight loss
- link to 'social media' features
- ability to customise workout and eating plans.

Test Strategies

Testing to inform development

During the development of the project, unit testing will ensure each activity works as designed once built. This prevents larger errors occurring later. This will involve testing all the buttons work as planned, that all input is processed correctly, and all output is correct in each activity. Some activities will involve rigorous testing including the use of correct, incorrect and extreme input. This will also include destructive testing.

For example, the test plan for the "Progress Updates" activity will be something like this.

Test Number	Test description	Expected result	Actual result	Successful?
1	Add a more recent update	Most recent update is displayed on creation		
2	Add updates from several months and years	Most recent update is displayed on creation		
3	Set date 35th March	Display error message		
4	Set date 30th	Display error message		
5	Set date 2010	Display error message		
6	Set weight =5	Display error message		
7	Set weight =500	Display error message		

The testing plan for adding a new workout should look as follows.

Test Number	Test Description	Expected Result	Actual Result	Successful?
1	Check setting of input text	Today's date loaded into input boxes in new workout activity		
2	Add button	Launches add new exercise activity		
3	Add button with input data	Launches add new exercise activity and previous data is kept when returned		
4	Exercise button in add exercise activity	Text is set to exercise name and appropriate input boxes appear		

5	Add button in add exercise activity	Input data is added to array and input boxes cleared		
6	Done button in add exercise activity	Exercise data is returned to new workout which displays it correctly		
7	Save button in new workout activity	All data is added to correct entities and linked. UI is cleared		

Development

Throughout the development of the application, at the end of the programming of each iteration, I will distribute to my client a prototype of the application, once I have completed my own testing. The client will be able to tell me if they have come across any other issues in the prototype, and if they feel the application so far is successful and meeting the solution requirements.

Iteration 1- Basic Infrastructure Code

Before development can properly begin, the basic infrastructure must be set up, including a working navigation menu and the database system.

In Android Studio, each activity is defined by a series of XML files and a corresponding Java class where the XML defines the elements (and their attributes) of the activity and the Java class defines its behaviour. Each activity has to be set up to include the same navigation menu, with the same behaviours, using a superclass that contains behaviours common to all activities relating to the navigation menu, called **BaseNavDrawer**.

```
public class BaseNavDrawer extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    public void onBackPressed() {
        //if navigation drawer is open and back button is pressed, close drawer
        //else execute usual method
        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        if (drawer.isDrawerOpen(GravityCompat.START)) {
            drawer.closeDrawer(GravityCompat.START);
        } else {
            super.onBackPressed();
        }
    }
```

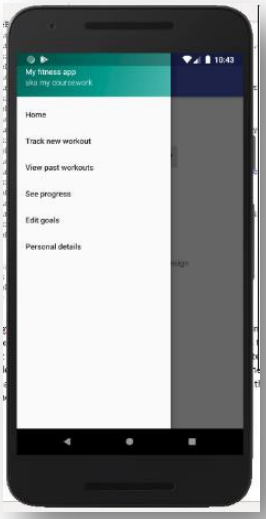
Other than the import statements, the rest of the class is shown. The method **onBackPressed()** overrides the standard method from the Android Studio libraries, to allow the user to use their back

button to close the navigation drawer if it's open. If it's not, then **super.onBackPressed()** calls the method from the superclass to undo the most recent action as expected.

```
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    //compare input to each activity
    //if it matches, change to that activity
    if (id == R.id.nav_home) {
        Intent intent = new Intent( packageContext: this, HomeActivity.class);
        startActivity(intent);
        overridePendingTransition( enterAnim: 0,  exitAnim: 0);
    } else if (id == R.id.nav_newWorkout) {
        Intent intent = new Intent( packageContext: this, NewWorkoutActivity.class);
        startActivity(intent);
        overridePendingTransition( enterAnim: 0,  exitAnim: 0);
    } else if (id == R.id.nav_pastWorkouts) {
        Intent intent = new Intent( packageContext: this, ViewWorkoutsActivity.class);
        startActivity(intent);
        overridePendingTransition( enterAnim: 0,  exitAnim: 0);
    } else if (id == R.id.nav_seeProgress) {
        Intent intent = new Intent( packageContext: this, SeeProgressActivity.class);
        startActivity(intent);
        overridePendingTransition( enterAnim: 0,  exitAnim: 0);
    } else if (id == R.id.nav_editGoals) {
        Intent intent = new Intent( packageContext: this, EditGoalsActivity.class);
        startActivity(intent);
        overridePendingTransition( enterAnim: 0,  exitAnim: 0);
    } else if (id == R.id.nav_personalDetails) {
        Intent intent = new Intent( packageContext: this, PersonalDetailsActivity.class);
        startActivity(intent);
        overridePendingTransition( enterAnim: 0,  exitAnim: 0);
    }

    //close navigation drawer
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

The method **onNavigationItemSelected (MenuItem, item)** defines what happens when the user clicks to navigate to a new page. It receives input from the activity class that calls the function, which is compared against the IDs of the activities to navigate to. A corresponding intent is created inside the "if" block of code, then called to change the activity the user sees and kill the previous activity. At the bottom, the navigation drawer is closed and the method returns true, so that the caller of the function knows the method has been executed successfully.

The navigation menu when drawn looks like this. It's called from the activity by a 3 lines symbol in the top left of any activity and swipes open and closed as required. This is all handled by the **NavigationView** class. On the menu, 6 activities have been created according to the design of the application, which are currently empty. When the user clicks on an option, the app navigates to their chosen activity. As these are only click buttons, there is no need to include any kind of input validation at this stage.




```
@Override
protected void onCreate(Bundle savedInstanceState) {
    //create variables to hold UI elements required, create activity
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    //set up the navigation drawer
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            activity: this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");
    drawer.addDrawerListener(toggle);
    toggle.syncState();
    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
```

In the Java class of each activity, an **onCreate()** function exists that is executed on launch of the activity. In here, the **onCreate()** function of the superclass is called, the XML file defining the layout is declared as activity home, and the toolbar is created (the bar at the top of the activity containing the navigation drawer button).

The next section of the method creates a navigation drawer, defines the creation of the button which opens and closes the drawer, and creates a listener that waits for the user to select an item from this drawer and calls my function **onNavigationItemSelected()** in the BaseNavDrawer class to implement its behaviour.

These lines of code are included in the **onCreate()** method in each activity's corresponding class so that each activity can include the same navigation drawer. This code can't be extracted as the **onCreate()** method must be included in each activity in order for Android Studio to compile the activities correctly.

The setup for the database system is more complicated. For the project I have chosen to use Android Studio's Room Persistence Library to interface between the application and the SQLite used with the OS. In the Room Library, a Java class and corresponding interface have to be created for each entity, and a class to define the overarching database structure. All these classes have been placed in a database package to make the file structure of my project easier to manage.

```
@Database(entities = {Users.class, ProgressUpdates.class, ExerciseBank.class,CompletedWorkouts.class,
        CompletedExercises.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {
    //when you need to do database migration? uninstall from emulator and reinstall instead.

    //create variables to manage DAO's
    private static AppDatabase INSTANCE;
    public abstract UserDao userDao();
    public abstract ProgressDao progressDao();
    public abstract BankDao bankDao();
    public abstract CompletedWorkoutsDao workoutsDao();
    public abstract CompletedExercisesDao exercisesDao();
```

The class that manages the database system is named AppDatabase. The Room Library makes extensive use of Java annotations, so this class is annotated with @Database, allowing me to declare the entities involved as attributes, the version number, and the default backup option. The rest of the code above creates the DAOs (Data Access Objects) to be used in accessing data from each entity.

```
//constructor called when AppDatabase is created
public static AppDatabase getAppDatabase(Context context) {
    //if database doesn't already exist, create it
    if (INSTANCE == null) {
        synchronized (AppDatabase.class) {
            if (INSTANCE == null) {
                //create database
                //callback inserts the data needed into the database on creation
                //this is run through an executor so that it doesn't execute on the main thread
                INSTANCE = Room.databaseBuilder(context.getApplicationContext(), AppDatabase.class,  name: "user-database")
                        .addCallback(new Callback() {
                            @Override
                            public void onCreate(@NonNull SupportSQLiteDatabase db) {
                                super.onCreate(db);
                                Executors.newSingleThreadScheduledExecutor().execute(new Runnable() {
                                    @Override
                                    public void run() {
                                        INSTANCE.bankDao().insertAll(ExerciseBank.populateData());
                                    }
                                }); }}).build();
            }}}
    return INSTANCE;
}

public static void destroyInstance() {
    INSTANCE = null;
}
```

This is the constructor for the AppDatabase class, i.e. is called when an AppDatabase object is created inside an activity. It checks first if the database already exists, then exits if it does. Otherwise, the database is created, and the callback created alongside it is executed. The callback calls the **run()** method defined inside it, which pre-populates the database with the data provided (at the moment this is only 5 sample exercises although I will expand it later on). The **run()** method exists inside a Runnable object as part of an Executor object – I will talk about this in detail later on.

```
package com.gamecodeschool.myfitnessapp.databases;
import androidx.room.*;

@Entity(tableName = "Completed_Exercises",
        foreignKeys = {@ForeignKey(entity = ExerciseBank.class,
                parentColumns = "exerciseId", childColumns = "exId"),
                        @ForeignKey(entity = CompletedWorkouts.class,
                parentColumns = "workoutId", childColumns = "workId") },
        indices = {@Index("exId"), @Index("workId") }   )
public class CompletedExercises {
    @PrimaryKey(autoGenerate = true)
    private int exerciseId;

    @ColumnInfo(name = "exId")
    private int exId;

    @ColumnInfo(name = "workId")
    private int workId;

    @ColumnInfo(name = "Sets")
    private int sets;

    @ColumnInfo(name = "Reps")
    private int reps;

    @ColumnInfo(name = "Weight")
    private float weight;

    @ColumnInfo(name = "Distance")
    private float distance;

    @ColumnInfo(name = "Time")
    private String time;

    @ColumnInfo(name = "Speed")
    private float speed;
```

```
public int getExerciseId() { return exerciseId; }

public void setExerciseId(int exerciseId) { this.exerciseId = exerciseId; }

public int getExId() { return exId; }

public void setExId(int exId) { this.exId = exId; }

public int getWorkId() { return workId; }

public void setWorkId(int workId) { this.workId = workId; }

public int getSets() { return sets; }

public void setSets(int sets) { this.sets = sets; }

public int getReps() { return reps; }

public void setReps(int reps) { this.reps = reps; }

public float getWeight() { return weight; }

public void setWeight(float weight) { this.weight = weight; }

public float getDistance() { return distance; }

public void setDistance(float distance) { this.distance = distance; }

public String getTime() { return time; }

public void setTime(String time) { this.time = time; }

public float getSpeed() { return speed; }

public void setSpeed(float speed) { this.speed = speed; }
```

On building the database, there were some problems with updating the schema when it changed the code in the AppDatabase class. As the database was already created when adding new entities, the correct way to update a database when sending application updates is to perform a database migration- this is a mapping of the old database onto the new database. However, it's complicated to implement correctly, and during development, it's significantly easier to delete the application and all database data and reinstall with the new and updated database structure. This would obviously NOT be done when updating the application on a real user's phone, as they would lose all their data.

Trying to use an updated database structure without erasing all the previous data returns several error messages, including "**Looks like you've changed schema but forgot to update the version number**". This error can be fixed by incrementing the version number, but that results in a new error. "**java.lang.IllegalStateException: A migration from 1 to 2 is necessary. Please provide a Migration in the builder**". Android Studio won't allow database schema to be updated without a migration path being specified so that data isn't accidentally lost or corrupted. The best way around this during development is simply to erase the database and reinstall it. The final part of the class creates a method to destroy the database.

Above is the content of the Java class that defines the structure of the completed exercises entity. Each entity I have created has the same attributes as in the design of the application. The class is declared with the entity annotation, which is also where the entity-entity relationships are declared. Below that, the primary key is defined, and set to auto-generate as a unique integer, then each column is defined with its name and variable type. Below the column declaration is all the getters and setters required.

```
@Dao
public interface CompletedExercisesDao {
    @Query("SELECT * FROM Completed_Exercises")
    List<CompletedExercises> getAll();

    @Insert
    void insertAll(CompletedExercises... exercises);

    @Delete
    void delete(CompletedExercises exercises);

}
```

This is the corresponding DAO, defined as a Java interface (a set of methods to be used by other classes). The DAO defines how Java code will access the database, where the relevant SQL code is, and insert and delete methods are defined.

The classes and corresponding interfaces for each entity have the same or similar format, with a few extra methods as shown below.

```
public ExerciseBank(String name, boolean strength, String body_area, boolean sets, boolean reps,
                    boolean weight, boolean time, boolean distance, boolean speed ) {
    this.name = name;
    this.strength = strength;
    this.body_area = body_area;
    this.sets = sets;
    this.reps = reps;
    this.weight = weight;
    this.time = time;
    this.distance = distance;
    this.speed = speed;
}

public static ExerciseBank[] populateData() {
    return new ExerciseBank[] {
            //Sample exercise data to pre-populate database with on create
            new ExerciseBank( name: "Pull-up", strength: true, body_area: "back,arms", sets: true,  reps: true,  weight: true,  time: false,  distance: false,  speed
            new ExerciseBank( name: "Bicep curl", strength: true, body_area: "bicep, arms", sets: true,  reps: true,  weight: true,  time: false,  distance: false,
            new ExerciseBank( name: "Sit-up",  strength: true,  body_area: "core", sets: true,  reps: true, weight: true, time: false, distance: false, speed: false),
            new ExerciseBank( name: "Squat", strength: true, body_area: "legs, glutes", sets: true, reps: true, weight: true, time: false, distance: false, speed: fal
            new ExerciseBank( name: "Run",  strength: false, body_area: "legs", sets: false, reps: false, weight: false, time: true, distance: true, speed: true)
```

Inside the declaration of the exercise bank entity is this extra code. I have created a constructor to set all the attributes of the object at once, just to reduce the lines of code necessary, and a method called **populateData(),** which returns as an array of objects of type ExerciseBank, 5 sample records for the database. The method is used in the AppDatabase class to pre-populate the database.

The other problem I have run into during development with the Room Persistence Library is to do with threading. As database operations can be computationally time-consuming and therefore make an application become laggy, the Room Persistence Library doesn't allow database operations to occur on the main thread. When I first tried to run **insertAll (),** I encountered an error- "**Caused by:**

java.lang.IllegalStateException: Cannot access database on the main thread since it may potentially lock the UI for a long periods of time." This require managing other threads for these operations to be run on.

```
.addCallback(new Callback() {
    @Override
    public void onCreate(@NonNull SupportSQLiteDatabase db) {
        super.onCreate(db);
        Executors.newSingleThreadScheduledExecutor().execute(new Runnable() {
            @Override
            public void run() {
                INSTANCE.bankDao().insertAll(ExerciseBank.populateData());
            }
```

This callback is created during the build of the database, as shown above in the AppDatabase class. A new callback is created with its mandatory **onCreate ()** method, which first executes the **onCreate ()** method of its superclass. The next line creates an Executor object, which creates a single thread executor to run code on, and then calls it's execute function, defined inside of the call. The execute function creates a new Runnable object, and its mandatory method **run ().**

This strategy of defining Runnable and Executors inside of their calls is useful as it restricts their domain to where they are used, but it can get messy and difficult to understand the code involved. This particular Runnable is only to be used once to pre-populate.

```
package com.gamecodeschool.myfitnessapp.databases;

public class DatabaseRunnable implements Runnable {

    private AppDatabase db;
    private Users user;
    private ProgressUpdates update;
    private ExerciseBank bank;
    private CompletedWorkouts workouts;
    private CompletedExercises exercises;

    public DatabaseRunnable(AppDatabase db, Users user) {
        this.db = db;
        this.user = user;
    }

    public void changeUser(Users newUser) { this.user = newUser; }

    @Override
    public void run() {
        db.userDao().insertAll(user);


    }
```

The alternative to defining an Executor within its call, is to define the Runnable in a separate class. This means that the same Runnable can be used many times from many points in the application without having to redefine and recompile the class. Here, the constructor defines the class variables as the data

to be added to the database, then the **run ()** method inserts the data from the Runnable declaration into the User database.

Testing

The project has been extensively tested during the first iteration including the navigation menu and the database. In this series of testing, I tested each activity that launches every other activity.

Test Number	Test description	Expected result	Actual result	Successful?
1	Open navigation menu from button	Navigation menu opens	Navigation menu opens	Yes
2	Close navigation menu with back button	Navigation menu closes	Navigation menu closes	Yes
3	Open and close navigation menu by swiping	Navigation menu opens and closes	Navigation menu opens and closes	Yes
4	Navigate to home activity from home activity	Home activity launches	Home activity launches	Yes
5	As above, from track workout activity	Home activity launches	Home activity launches	Yes
6	As above, from view past workout activity	Home activity launches	Home activity launches	Yes
7	As above, from see progress activity	Home activity launches	Home activity launches	Yes
8	As above, from edit goals activity	Home activity launches	Home activity launches	Yes
9	As above, from personal details activity	Home activity launches	Home activity launches	Yes
10	Navigation menu closes when back button pressed	Navigation menu closes and activity remains the same	Navigation menu closes and activity remains the same	Yes
11	Navigate to track workout activity	Track workout activity launches	Track workout activity launches	Yes
12	Navigate to view past workout activity	View past workouts activity launches	View past workouts activity launches	Yes
13	Navigate to see progress activity	See progress activity launches	See progress activity launches	Yes
14	Navigate to edit goals activity	Edit goals activity launches	Edit goals activity launches	Yes
15	Navigate to personal details activity	Personal details activity launches	Personal details activity launches	Yes

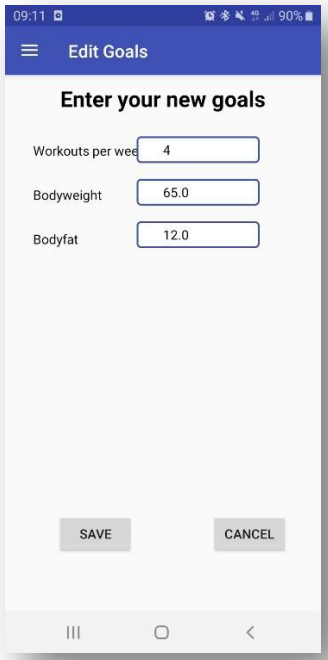
Iteration 2- Edit Goals and Personal Details Activities

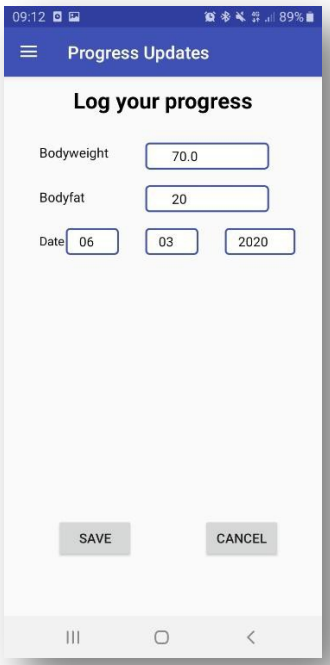
In the next iteration of development, I have decided to build the edit goals activity and the personal details activity, as they are both self-contained, and deal exclusively with the entry and maintaining of input data in database entities.

I have started with building the basic structure of the UI as per my design for both activities. Below is shown the UI as displayed on my phone, not the Android Studio virtual machine- it has slightly different

dimensions and so the input boxes will need to be readjusted based on the actual dimensions instead of the VM dimensions.

Test





In each input box, the current value is displayed- however, when this is removed, a hint is shown as to what to enter. All the activities built in this iteration include only text objects, buttons and input objects on their GUI.

```
for (int i = 0; i < name.length(); i++) {
    if (!Character.isLetter(name.charAt(i))) {
        return false;
    }
```

Added this clause in the validation methods- it checks every character input for the user's name that it is in fact a letter and not a number or symbol such as '!'.

Code – Personal Details

```
public class PersonalDetailsActivity extends BaseNavDrawer
        implements NavigationView.OnNavigationItemSelectedListener{

    private AppDatabase db;
    private Button saveButton;
    private Button cancelButton;
    private EditText bodyfat;
    private EditText bodyweight;
    private EditText day;
    private EditText month;
    private EditText year;
    private TextView errorMsg;

    //define what happens when save and cancel buttons are clicked
    private View.OnClickListener saveListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            {   addtoDb(db); }        }
    };

    private View.OnClickListener cancelListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            cancel();
        }
    };
```

The personal details activity begins by declaring the class variables to be used, and creating the on click listeners- this is the code that will be executed when the on screen buttons are clicked. When the save button is clicked, **addtoDb()** is called, and when the cancel button is clicked, **cancel()** is called.

The on create method for this activity is shown above. All the code presented (defining the menu objects and initialising the database)

```
    //get date and insert into input boxes
    Date c = Calendar.getInstance().getTime();
    SimpleDateFormat df = new SimpleDateFormat( pattern: "dd/MM/yyyy");
    String date = df.format(c);

    String currentDay = Character.toString(date.charAt(0)) + Character.toString(date.charAt(1));
    String currentMonth = Character.toString(date.charAt(3)) + Character.toString(date.charAt(4));
    String currentYear = Character.toString(date.charAt(6)) + Character.toString(date.charAt(7)) + Character.toString(date.charAt(8)) + Cha

    day.setText(currentDay);
    month.setText(currentMonth);
    year.setText(currentYear);

}
```

```
navigationView.setNavigationItemSelectedListener(this);

db = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class,  name: "database-name").build();

saveButton = findViewById(R.id.saveButton);
saveButton.setOnClickListener(saveListener);
cancelButton = findViewById(R.id.cancelButton);
cancelButton.setOnClickListener(cancelListener);

bodyfat = findViewById(R.id.inputBodyFat);
bodyweight = findViewById(R.id.inputBodyWeight);
day = findViewById(R.id.inputDay);
month = findViewById(R.id.inputMonth);
year = findViewById(R.id.inputYear);
errorMsg = findViewById(R.id.errorMsg);

//get latest data and insert into input boxes
ProgressUpdates recentUpdate = getDetails();
bodyfat.setText(String.valueOf(recentUpdate.getBodyfat()));
bodyweight.setText(String.valueOf(recentUpdate.getWeight()));
```

and then the other GUI objects are connected to the code by means of a unique ID declared within the XML. Each GUI object (text, buttons, input, etc.) has be connected to the code this way in order to change anything during the execution of the application. The next

section of code calls the **getDetails()** method. This method returns the most recent update stored in the database, so the on create method can use this information to display the most recent data in the input boxes. This means that the user can see what they last entered as an update, and this promotes usability of the application as users won't enter duplicate data.
The last part of the on create method gets the current date from a Calendar object, and sets the input boxes relevant to the current date. This should reduce clicks required by the user, as mostly updates should be entered on the day measured, and will at least be entered within a few days.

```
private ProgressUpdates getDetails() {
    //get most recent entry in table
    Future data = Executors.newSingleThreadExecutor().submit((Callable) () → {
                                return db.progressDao().getTop();
                            }
    );

    ProgressUpdates update;

    try { update = (ProgressUpdates) data.get();
    } catch (Exception ex){
        //if no value yet in database, return -1
        update = new ProgressUpdates();
        update.setBodyfat(-1);
        update.setWeight(-1);
        update.setDate("-1");
    }
    return update;
}
```

This is the **getDetails ()** method- it's called from the on create method as shown above and returns the user's most recent update in a ProgressUpdates object. In order to retrieve this data from the database, an Executor is created to execute the code in a non-GUI thread, but this time it's running a Callable instead of a Runnable. The only distinction is that a Callable returns a value, in this case a ProgressUpdates object. The returned value is stored in a future object, and retrieved in a try/except block, so that the program won't crash if the database execution takes all ongoing time and no data is as of yet stored in the future object. Once this code has executed, the most recent update is returned to the on create method.

```
private void addtoDb(AppDatabase db) {
    ProgressUpdates update = new ProgressUpdates();
    if (!presenceCheck(bodyweight) || !presenceCheck(bodyfat) || !presenceCheck(day) || !presenceCheck(month) || !presenceCheck(year)) {
        errorMsg.setVisibility(View.VISIBLE);
        return;
    }

    int newBodyfat = Integer.valueOf(bodyfat.getText().toString());
    float newWeight = Float.valueOf(bodyweight.getText().toString());
    String newDay = day.getText().toString();
    String newMonth = month.getText().toString();
    String newYear = year.getText().toString();
    String newDate =  newDay + "/" + newMonth + "/" + newYear;

    if (validate(newBodyfat, newWeight, newDay, newMonth, newYear)) {
        update.setBodyfat(newBodyfat);
        update.setWeight(newWeight);
        update.setDate(newDate);
        addUpdate(db, update);
        errorMsg.setVisibility(View.INVISIBLE);
    }
    else {
        errorMsg.setVisibility(View.VISIBLE);
    }
}
```

This is the **addtoDb()** method called when the save button is clicked. It creates a ProgressUpdates object to store the input data in, then validates the input data. If the code finds that the user has failed to enter a value in one of the boxes, the presence check will fail, an error message will be displayed and the method will escape. The values are retrieved from the input objects, and the validate function is called, which conducts range and other checks on the input data. If the data is valid, the error message is made invisible, and the **addUpdate()** method is called with the new ProgressUpdates as a parameter. Otherwise, an error message is displayed and the method escapes. All of this ensures that incomplete records aren't added to the database, which will cause errors further down the line, and ensures that the user doesn't accidentally enter incorrect data into the database.

```
private void addUpdate(final AppDatabase db, final ProgressUpdates update) {
    Executors.newSingleThreadScheduledExecutor().execute(new Runnable() {
                                                             @Override
                                                             public void run() {
                                                                 db.progressDao().insertAll(update);
                                                             }
                                                         });
    /*
    year.setText(null);
    month.setText(null);
    day.setText(null);
    bodyfat.setText(null);
    bodyweight.setText(null); */

    Intent intent = new Intent( packageContext: this, HomeActivity.class);
    startActivity(intent);
    overridePendingTransition( enterAnim: 0,  exitAnim: 0);
}
```

The addUpdate method again uses an Executor with a Runnable to insert the validated input data into the progress updates entity and returns the user to the home page.

```
private void cancel() {
    year.setText(null);
    month.setText(null);
    day.setText(null);
    bodyfat.setText(null);
    bodyweight.setText(null);
}
private boolean presenceCheck(EditText eText) {
    if(eText.getText().toString().isEmpty()) {
        return true;
    } else {
        return false;
    }
}
```

This is the cancel and presence check methods. The cancel method is called by the cancel buttons on click listener, and resets all the input values to null. This will cause the hint in the XML to be displayed in the input box instead.

The presence check uses the **isEmpty ()** method on an input box to determine if there is a value present and returns the validity of the data.

```
}
private boolean validate(int inputBodyfat, float inputBodyweight, String inputDay, String inputMonth, String inputYear){
    boolean valid = true;
    int day = Integer.valueOf(inputDay);
    int month = Integer.valueOf(inputMonth);
    int year = Integer.valueOf(inputYear);

    if (inputBodyfat < 3 || inputBodyfat > 45){
        valid = false;
    }
    else if (inputBodyweight < 40 || inputBodyweight > 140) {
        valid = false;
    }
    else if (day < 0 || day > 31 || month < 0 || month > 12) {
        valid = false;
    }
    else if ( day == 31 && (month == 2 || month == 4 || month == 6 || month == 9 || month == 11)) {
        valid = false;
    }
    else if ( day > 29 && month == 2) {
        valid = false;
    }
    else if (year < 2019 ) {
        valid = false;
    }
    return valid;
}
```

The last method of this activity is the validation. It receives the user's input data as parameters, and conducts range checks on body fat and bodyweight, and checks all dates are valid, e.g. not February 30th

or June 31st. This returns the validity of the data to the **addtoDb ()** method in order to decide whether the code should proceed in adding the data to the database.

Problems – Progress Updates

The only significant issues with the development of this activity, other than the compiler warning when requesting data from a future object, was with the pre-population of the database and the initial retrieval of non-existent data. Essentially, on install, the database has no data in it, so the launch of the ProgressUpdates activity crashed the application as it was trying to request the most recent no data. Could be logic error- the first set of data can't enter until the first set of data is retrieved. The first solution to this I attempted was to add an invalid collection of data as the first entry as part of the pre-population of the database on creation. However, I discovered that the pre-population of this data and the exercise bank wasn't reliable, i.e. wasn't being called on every install and often the application still crashed. Meaning had to find another way to reliably pre-populate the database with exercises.

The next solution to solve this was to have the home activity test for the existence of the database, and, again, pre-populate the ProgressUpdates entity with invalid data, and the exercise bank with some sample exercises. Appeared to work on my VM, but couldn't remove the invalid data successfully, and testing for the existence of the database didn't work on my phone. Maybe due to some distinction in the compilation of the application between different devices as my VM is a Google Nexus and my phone a Samsung.

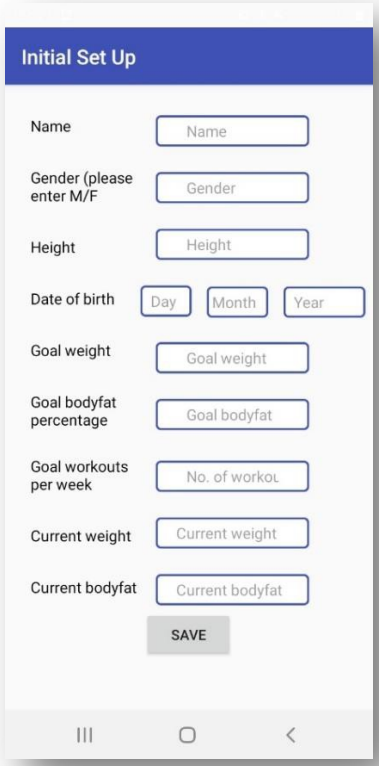
The eventual solution to all these issues was the creation of activity outside of my initial designs, which allows the user to pre-populate the database with their details on install. The code run on the home activity is shown below, the first code block being part of the on create method.

```
    //If app has just launched, launch initial entry activity
    SharedPreferences settings = getSharedPreferences( name: "prefs",  mode: 0);
    boolean firstRun = settings.getBoolean( s: "firstRun",  b: true);
    if ( firstRun )
    {             // to be ran on install
        goToEntry();
    }
}


private void goToEntry() {
    Intent intent = new Intent( packageContext: this, InitialEntry.class);
    startActivity(intent);
    overridePendingTransition( enterAnim: 0,  exitAnim: 0);
}
```

Here, the on create method creates a shared preference object, firstRun, set to true on creation of the variable. This variable is static across the whole application and can be changed in any activity. If firstRun is true, the goToEntry method is called, which takes the user to an activity called InitialEntry. This is where they can enter their personal details and first progress update. In the InitialEntry activity, once the user's personal details have been added, this is changed to false and never used again.

Code – Initial Entry

This activity was created as a solution to the problems that arose during the development of the ProgressUpdates activity detailed earlier, although it is a standard function of most mobile applications to have a data entry page before the application is used. The UI is of a similar design to the other activities below.




```
//If app has just launched, launch initial entry activity
SharedPreferences settings = getSharedPreferences( name: "prefs",  mode: 0);
boolean firstRun = settings.getBoolean( s: "firstRun",  b: true);
if ( firstRun )
{            // to be ran on install
    goToEntry();
}
```

This code has been inserted into the on create method of the home activity. The activity is triggered by this SharedPreference, which is static across all activities. It's set to true on creation of the preference, and then later set to false within InitialEntry on completion of all the input data. This variable remains as false across all activities from then, and so the InitialEntry activity is never retriggered.

```
public class InitialEntry extends AppCompatActivity {

    private Button saveButton;
    private AppDatabase db;
    private EditText inputName;
    private EditText inputGender;
    private EditText inputHeight;
    private EditText inputDay;
    private EditText inputMonth;
    private EditText inputYear;
    private EditText goalWeight;
    private EditText goalBodyfat;
    private EditText goalWorkouts;
    private EditText currentWeight;
    private EditText currentBodyfat;
    private TextView entryError;

    private View.OnClickListener saveListener = (v) → {
            { addtoDb(db);
                goHome();
                }
    };
```

The class begins by initialising all the class variables required to hold the input boxes, in EditText objects, and the buttons and database. Then, the listener for the save button is defined as calling the **addtoDb ()** method and then the **goHome ()** method, which returns the user to the home page.

The on create method is shown above for the InitialEntry activity. The first block of code initialises the toolbar and calls the superclass on create method, the same as every other activity. The save button variable is linked to the save button in XML through the findViewById function, and the on click listener defined as the one previously shown. Then, all the input boxes in the XML are connected to their corresponding variables in the code. Note that the code for the navigation bar isn't included in this class as it should not be possible to navigate away from the activity without entering the required data and shouldn't be possible to re-enter the activity on completion.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_initial_entry);
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    saveButton = findViewById(R.id.initialSaveButton);
    saveButton.setOnClickListener(saveListener);

    db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class,  name: "database-name").build();

    inputName = findViewById(R.id.inputName);
    inputGender = findViewById(R.id.inputGender);
    inputHeight = findViewById(R.id.inputHeight);
    inputDay = findViewById(R.id.inputDay);
    inputMonth = findViewById(R.id.inputMonth);
    inputYear = findViewById(R.id.inputYear);
    goalWeight = findViewById(R.id.inputGoalWeight);
    goalBodyfat = findViewById(R.id.inputGoalBodyfat);
    goalWorkouts = findViewById(R.id.inputGoalWorkouts);
    currentWeight = findViewById(R.id.inputCurrentWeight);
    currentBodyfat = findViewById(R.id.inputCurrentBodyfat);
    entryError = findViewById(R.id.entryError);
```

The **goHome**() method is called from the save button's listener, after all the data entry has been completed. It just directs the user away from this activity so they can now use the rest of the application as intended.

```
private void goHome(){
    Intent intent = new Intent( packageContext: this, HomeActivity.class);
    startActivity(intent);
    overridePendingTransition( enterAnim: 0,  exitAnim: 0);
}
```

```
private void addtoDb(AppDatabase db) {
    ProgressUpdates update = new ProgressUpdates();
    Users user = new Users();

    //check user has entered a value in every field
    if (!presenceCheck(inputName) || !presenceCheck(inputGender) || !presenceCheck(inputHeight) || !presenceCheck(input
        //if a field has been left blank
        entryError.setText("Please enter a value for every attribute");
        entryError.setVisibility(View.VISIBLE);
        return;
    }

    //collects all the data
    String name = inputName.getText().toString();
    String initialgender = inputGender.getText().toString();
    char gender = initialgender.charAt(0);
    int height = Integer.valueOf(inputHeight.getText().toString());
    int day = Integer.valueOf(inputDay.getText().toString());
    int month = Integer.valueOf(inputMonth.getText().toString());
    int year = Integer.valueOf(inputYear.getText().toString());
    float weight = Float.valueOf(goalWeight.getText().toString());
    int bodyfat = Integer.valueOf(goalBodyfat.getText().toString());
    int workouts = Integer.valueOf(goalWorkouts.getText().toString());
    String DOB = day + "/" + month + "/" + year;
    float currentWeightValue = Float.valueOf(currentWeight.getText().toString());
    int currentBodyfatValue = Integer.valueOf(currentBodyfat.getText().toString());
```

The first part of the **addtoDb()** method initialises a new object of types Progress Update and Users in preparation for the data. Then a presence check is conducted on all the input boxes- if the user has failed to enter a value in any field, the code will display an error message and stop processing the data. The next block of code collects the input data into variables.

```
Date c = Calendar.getInstance().getTime();
SimpleDateFormat df = new SimpleDateFormat( pattern: "dd/MM/yyyy");
String date = df.format(c);

//check numerical values are reasonable
if (!rangeCheck(height, weight, bodyfat, workouts, currentWeightValue, currentBodyfatValue)) {
    entryError.setText("One of your values seems wrong, please change it");
    entryError.setVisibility(View.VISIBLE);
    return;
}
if (!otherChecks(gender, day, month, year)) {
    entryError.setText("Please make sure your DOB is correct and gender is entered as m/f");
    entryError.setVisibility(View.VISIBLE);
    return;
}

//inserts data into object to be entered into db
user.setName(name);
user.setGender(gender);
user.setHeight(height);
user.setDOB(DOB);
user.setGoalWeight(weight);
user.setGoalBodyfat(bodyfat);
user.setGoalNumWorkouts(workouts);

update.setDate(date);
update.setWeight(currentWeightValue);
update.setBodyfat(currentBodyfatValue);

addUpdate(db, update, user);
```

The next part of the **addtoDb()** method gets the current date to be used in the update object. Then, it conducts a range check on all the input values, and if a value is out of a set range, the code displays

another error message and discontinues the method. A series of other checks are conducted on dates and gender, and then if the input data passes all the checks, the attributes of the objects created earlier are set to the input data, and **addUpdate()** is called.

```
private void addUpdate(final AppDatabase db, final ProgressUpdates update, final Users user) {
    Executors.newSingleThreadScheduledExecutor().execute(new Runnable() {
        @Override
        public void run() {
            db.progressDao().insertAll(update);
            db.userDao().insertAll(user);
            db.bankDao().insertAll(ExerciseBank.populateData());
        }        });

    //declare that it's no longer "firstrun" so this activity will never be called again
    SharedPreferences myPref = this.getSharedPreferences(
            name: "prefs",  mode: 0);
    myPref.edit().putBoolean( s: "firstRun",  b: false).commit();
}
```

The **addUpdate()** method is the final part of adding the data to the database- once it has been validated and set as attributes in entity objects in the previous method, **addUpdate** creates an Executor and inserts the input data into the ProgressUpdates entity and Users entity. Additionally, as a solution to the pre-population error I had earlier, the pre-population of possible exercises is completed here. Then, the **SharedPreference(),** which triggers the InitialEntry activity, is set to false, so that once complete, this activity is never called again.

```
private boolean presenceCheck(EditText eText) {
    if(eText.getText().toString().isEmpty()) {
        return true;
    } else {
        return false;
    }
}
```

The **presenceCheck ()** takes in the Edit Text object linked to the input boxes on the UI as a parameter, and uses the **isEmpty ()** method to conduct a presence check. It returns the result of the check.

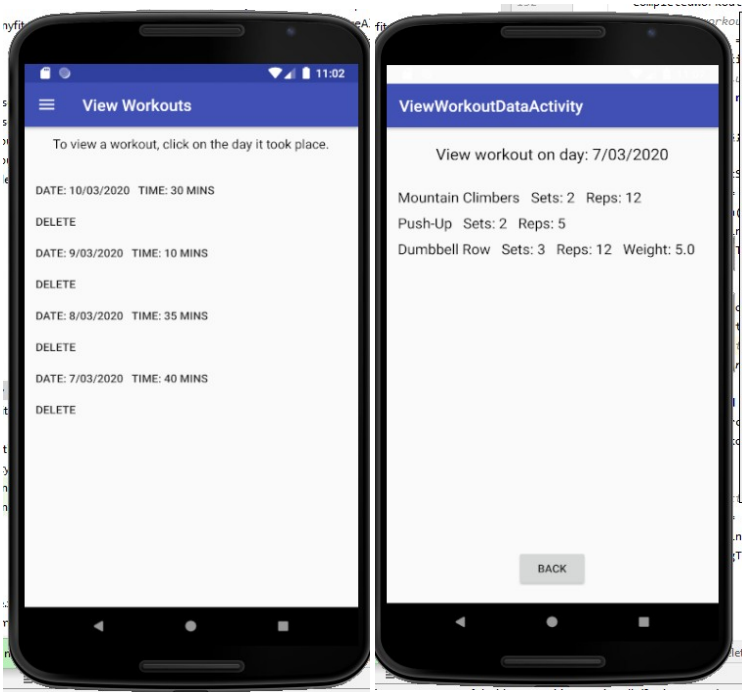
The final part of the InitialEntry activity is the method involved in validation- the **angeCheck()** method compares the parameters with values I have set as acceptable, and returns the result of the check- true if the data is valid and false if not. The **otherChecks()** method conducts the last few checks required- ensuring that only M or F is entered in the gender box and that invalid dates such as 31st June are not entered. Again, the result of the validation is returned to the method call. As demonstrated above, for each activity, I will need to test the successfulness of each button under different circumstances/input data, and I will also need to conduct presence and range/value checks on most input boxes that allow the user to type.

Testing

I have extensively tested the elements built in this first iteration – the navigation menu and the database. In this series of testing, I have tested that each activity launches each other activity, which is about 39 tests in total, so I have only detailed one of each type below.

Test Number	Test description	Expected result	Actual result	Successful?
1	Open navigation menu from button	Navigation menu opens	Navigation menu opens	Yes
2	Close navigation menu with back button	Navigation menu closes	Navigation menu closes	Yes
3	Open and close navigation menu by swiping	Navigation menu opens and closes	Navigation menu opens and closes	Yes
4	Navigate to home activity from home activity	Home activity Launches	Home activity launches	Yes
5	As above, from track workout activity	Home activity launches	Home activity launches	Yes
6	As above, from view past workout activity	Home activity launches	Home activity launches	Yes
7	As above, from see progress activity	Home activity launches	Home activity launches	Yes
8	As above, from edit goals activity	Home activity launches	Home activity launches	Yes
9	As above, from personal details activity	Home activity launches	Home activity launches	Yes
10	Navigation menu closes when back button pressed	Navigation menu closes and activity remains the same	Navigation menu closes and activity remains the same	Yes
11	Navigate to track workout activity	Track workout activity launches	Track workout activity launches	Yes
12	Navigate to view past workout activity	View past workouts activity launches	View past workouts activity launches	Yes
13	Navigate to see progress Activity	See progress activity launches	See progress activity launches	Yes
14	Navigate to edit goals activity	Edit goals activity launches	Edit goals activity launches	Yes
15	Navigate to personal details activity	Personal details activity launches	Personal details activity launches	Yes

Test

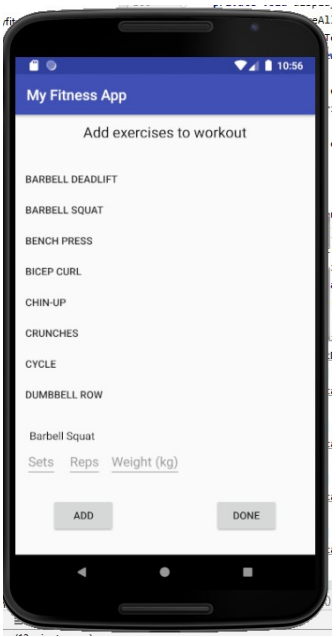



```
public class ViewWorkoutsActivity extends BaseNavDrawer
        implements NavigationView.OnNavigationItemSelectedListener {

    private AppDatabase db;
    private CompletedWorkouts[] workouts;
    private LinearLayout layout;
    private TextView topText;
    private Button[] buttons;
    private Button[] delButtons;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_view_workouts);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

One major problem encountered during the development of this activity, was issues with the delete button. Originally, tried to delete first the items in the completed workouts table, then in the completed exercises table, but due to the enforcement of referential integrity, this was preventing any items being deleted. This was fixed by reordering the statements as in the code shown above.

Code- Add Exercise Activity Test

Test to allow users to enter the details of their workouts, including the date, exercises and their respective sets, reps, weight, etc. The screen below shows the outcome.



The UI is shown above, as displayed when an exercise has been selected. The add new exercise activity is accessed through the add button on the New Workout activity, and not the navigation menu like the other activities.

```
public class AddExerciseActivity extends AppCompatActivity {

    private AppDatabase db;
    private Button done;
    private Button add;
    private Button[] nameButtons;
    private LinearLayout layout;
    private EditText[] inputTextBoxes = new EditText[6];
    private TextView nameText;
    private ExerciseBank[] exercises;
    private ConstraintLayout pageLayout;
    boolean[] inputs = {false,false,false,false,false};
    String[] types = {"Sets","Reps","Weight","Time","Distance"};
    List<CompletedExercises> completedEx = new ArrayList<~>();
    private TextView errorMsg;

    private View.OnClickListener doneListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            {   done();
            }
        }
    };

    private View.OnClickListener addListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            {   add();
            }
        }
```

The class begins with its class variables and the on click listeners for the done and add buttons, which call the done() and add() methods respectively.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_exercise);

    done = findViewById(R.id.doneButton);
    done.setOnClickListener(doneListener);
    add = findViewById(R.id.addExerciseButton);
    add.setOnClickListener(addListener);
    layout = findViewById(R.id.ExerciseButtonLayout);
    pageLayout = findViewById(R.id.pageLayout);
    nameText = findViewById(R.id.exName);
    errorMsg = findViewById(R.id.exerciseErrorMsg);

    db = AppDatabase.getAppDatabase(getApplicationContext());

    exercises = getExercises();
    String[] names = getNames(exercises);
    displayExercises(names);

    Intent intent = this.getIntent();
    Bundle bundle = intent.getExtras();
    CompletedExercises[] data = (CompletedExercises[]) bundle.getSerializable( key: "CompletedExercises");
    for (int i = 0; i < data.length; i++ ) {
        completedEx.add(data[i]);
    }
}
```

I have not yet built parts of the application to display data from my databases and not yet been able to test the data entry to see if it's working. However, the application now runs without crashing when attempting to complete database tasks such as its creation with the wrong version number, or attempting to run code on the main thread, so I have assumed that the use of Executors to manage threads has been correctly implemented. More testing will be done in later iterations on the database.

As t more testing was done in relation to corresponding errors during the development of the code (such as updating the schema of the database resulting in a version number error), the use of the whole application together has not resulted in new errors in this iteration.

The test data shows that the current account system is very robust and only has minor problems that do not affect its function and security (phone number can be changed to have incorrect datatype in profile page).

This problem cannot be addressed/met unless my client expands his business (by buying a dedicated file server or by paying for a cloud storage service).

The test data shows that the system functions correctly and is quite robust; this is shown by the lack of problems when inputting the erroneous data '555' (via 'inspect element' rather than selecting the available options).

The final product must also be compatible with various devices, such as smartphones or tablets. This criterion is at least partially met: Although the app functions perfectly and is easy to use on a typical smartphone or desktop, the app might not be as appealing due to the possible lack of testing on many different screen sizes, e.g. holding a phone or small tablets sideways will make the app functional but ugly (due to thick scrollbars). Also, not many browsers were used to test the app (only Chrome, Edge and Firefox), so the app may not work properly in some browsers.

To fully meet this criterion, I could create a subscription menu only available to regular users in the future. This was not done in this prototype because I did not have enough time and the scope/ambition of my client's business was misunderstood (couldn't host a file server, so the subscription system was not worth implementing).

According to some testers, the app was easy to use/navigate and aesthetically pleasing; this suggests that the app had a low amount of clutter. These 2 points show that the app fully meets another success criterion, where the app's layout should be simple and easy to navigate and is not confusing. However, I believe that the app has too much text compared to images and other media.

Overall, the app performed well, the navigation performed well and it was easy to use.

Services and functions to develop further are the content of the email confirmation, will include information about full membership and the terms and conditions. Some of the testers found that the type of exercise from the drop-down and quick list did not have many options. I will add further options to the exercise list. This will ensure that users have the ability to customise workout and eating plans to provide a daily progress report on fitness training /food and calorie daily intake/weight progress/sleep/water.

They also reported that linking to other technologies was not a stable option. Some technologies linked and the data was imported and calculated in the app but not all technologies. This needs to be investigated further and help files or links added to the app.

The search facilities for recipes were not as effective, the search function worked, but the search results need to be more specific in relation to the user's keyword search. One tester did not like the sound effects when entering data, the other users had no objection, so this facility will remain as it confirms when data has been input into the app. Testers were able to download social media apps. One tester suggested that a link to the ToKa Fitness Facebook page would also help.

Evaluation

Post-development Testing

Review of Previous Testing Results

So far, I conducted thorough testing of the application as it was developed and fixed any bugs and errors that arose as part of that. The following tests will be done on the application as a whole and then a final prototype sent out to clients to test. Clients will be using real data to see how it responds to real-life and evaluate its overall usefulness. The feedback received from clients will be in the form of rating the application's usability, functionality, robustness and any other overall comments they wish to make.

Alpha Testing the Product

So far, I have destructively tested each portion of the application individually. As part of the alpha testing, it is essential to ensure that regular use of the application as a whole doesn't bring up any more hidden bugs and that it all works as built. For this, I have used their real-life data, consisting of 3 runs and three strength training workouts a week.

My experience using the product in my real life has been enjoyable- found it easier to input my data in the product than any of the applications tried as part of my testing (MyFitnessPal, JeFit, etc.). Having not been able to input all the exercises done regularly in the sample test bank - anticipating feedback from clients informing me that they need the ability to create custom exercises to use the application to its full extent. No issues using

the interface, although did find it cumbersome having to scroll back through several workouts to view past data once I had added several exercises to the application.

Beta Testing

Client 1: Functionality: 8/10
Usability: 9/10
Robustness: 6/10

Other Comments: I found the application easy to use and because I'm not a significant fitness buff the fact that the interface is much simpler than other applications didn't put me off like I usually am. I would continue using it in my real life if I could add my exercises to it- the sample bank was tiny. All I need is somewhere to see what I've done before, and this application provides it well.

Client 2: Functionality: 4/10
Usability: 7/10
Robustness: 10/10

Other Comments: Although the application was easy to use, I found that it didn't fulfil my needs. I can write down what I'm doing on a piece of paper or any other app, but what I was hoping for was more of a measure of progress in terms of strength- I would like to see some means of how much more weight I'm lifting over time by body part and perhaps a graph of changes in body fat. This application makes the bare bones of the features easy to use but wouldn't persuade me to change to using it full time.

Client 3: Functionality: 8/10
Usability: 5/10
Robustness: 8/10

Other Comments: My main issue with the application is that over a long period, it would be impossible to see past data. I can't see myself scrolling back through hundreds of workouts to see the details of them- I need some searching feature for both exercises and workouts to make the application more natural to use for me on the whole. Other than that though I did find that it fulfils all the requirements I had for it- I can input all of my data pretty quickly, and it allows me to do everything that I need to do.

Client 4: Functionality: 5/10
Usability: 7/10
Robustness: 9/10

Other Comments: I did find that the application let me do everything it was built to do quickly and without errors. However, the missing optional functionality (such as searching for exercises, modifying workouts and graphs of progress) would make a substantial difference to how satisfied I would've been with the end-product.

Review of Final Testing

Average Functionality Score: 6.25
Average Usability Score: 7
Average Robustness Score: 8.25

On the whole, my client's results are encouraging- most of the problems they've found are due to previously identified issues or because some of the optional features haven't been developed yet. No major technical problems have been uncovered, so I have no more bugs or errors to report and fix, which I would say is some success. Additionally, the clients almost all tend to conclude that the application is, in fact, easy to use, and works robustly for the functionality it provides, and so could be a partial success at least.

Solution Requirements

Key features

Number	Requirements	Explanation	Achieved?
1	Allow users to input exercise, weight, reps, sets	This is needed to do any analysis on workouts completed	Yes- using a combination of New Workout and Add Exercise activities
2	Allow users to input personal details	This is needed to do any analysis on progress made	Yes- this is achieved with the InitialEntry activity
3	Allow users to change personal details	This allows users to change their goals as their weight and strength change	Yes- this is achieved with the Progress Updates and Edit Goals activity
4	Display past workouts	Allows users to see previously entered data	Yes- these are chosen in View Workouts activity and displayed in View Workout Data activity
5	Display progress	Allows users to see the progress calculated by the app	Yes- achieved in the Display Progress activity
6	Allow users to modify previous workouts	Means that users can rectify any mistakes made	Partly- users can delete previous workouts
7	Intuitive UI	Makes the application easy to use, unlike others on the market	Yes-the application requires minimal clicks to use and every function is labelled clearly.

8	Calculate some statistical analysis on input data, e.g. change in weight lifted over	This is part of what makes the application better than just writing past workouts on paper	Yes- a percentage change calculated of the user's bodyweight and fat data in Display Progress.
9	Allow users to see cardio and strength workouts in conjunction	Progress in cardio and strength workouts is inherently linked, so other applications not providing this is a mistake and something I intend to improve on	Yes- both strength and cardio workouts are displayed in the same list
10	Provide features completely free of charge for all Android users	Most decent fitness trackers have premium features or require users to buy hardware like wearable heart rate trackers – I intend to provide a free of charge service	Yes

Additional functionality

Number	Requirements	Explanation
1	Allow users to create custom exercises	There is an enormous variety of types of strength exercise and my database will never be comprehensive – allowing customisable ones prevents users from being limited to exercises chosen for my database.
2	Facilitate repeat entries.	Users tracking data on paper have to enter the entire set of repeat data twice – some kind of shortcut for this would make the application much quicker and easier to use.
3	Provide visualisations of the user's progress in graph form	A visualisation of progress is much easier for most people to understand than pure numbers – it would help them understand their progress.
4	Allowing custom/auto generated future workouts or	Allows users to further customise the application to their own personal goals and workout plans. Helps user plan their future workouts to progress.
5	Store data on an external server	This would prevent the data needing to be stored locally and help users backup their data/allow it to be transferred to another device.
6	Include social functionality/external software integration, e.g. Facebook or Twitter	Allow users to share their progress and compete with friends – many people find the social aspect of working out motivating.

As shown in the above tables, all of the key requirements for functionality originally received from my clients have been met. This means that the application can successfully and reliably do all of the things it MUST do, however as identified previously, there are places where the implementation of these requirements is perhaps not as easy as it could have been (e.g. the use of a DatePicker to facilitate date entry, or the ability to search for exercises when adding a workout) which detracts from the overall usability. None of the additional features have been attempted, and the clients have pointed out that 1 and 3 would make a big difference to the overall application had they been attempted.

Success Criteria

Number	**Success criteria**	**Justification**	**Met?**
1	Application is easy to install and set up	Else it will be much more convenient for users to just track on paper. Eventually placing the app on the Google Play Store would make this standard and easy.	Yes- will be available to download like any other application on the Play Store

2	Application is easy to navigate and features are clear and understandable	A complicated app will discourage users – this should be as simple and easy as possible to use.	Yes- application uses a common menu framework and all clients found it intuitive
3	Application stores the users data and allows them to modify it	Essential to the functionality of the data management.	Yes- done in the add new workout and view workout activity
4	Built features work reliably	If features don't work, will cause errors and be irritating to users.	Yes- all clients rated robustness highly
5	Users are able to see their workouts and progress	Can't track their workouts if data is inaccessible.	Yes- in the view workouts and progress activities
6	Data entry is as easy as possible	Users don't want to enter data if it will take hours to log each workout – to be feasible for usage, must be easy.	Partially- users would like custom exercises and a search bar
7	Provide all functionality free on Android phones	Most existing solutions have premium features or require paid hardware or an upfront cost – this makes the app desirable.	Yes

The above table contains the success criteria identified in the analysis of the project and declares whether each criterion is met and where. All of the success criteria have been met, with the exception of 6 which suffers from the same problems as in the clients' feedback earlier- some lacking additional functionality means users can't add custom exercises currently.

Further Development

As previously identified by beta testing and review of the success criteria and requirements, the key features wrong or missing from the application as it is are as follows:

- The ability to search for an exercise or filter by an exercise type
- The ability to create custom exercises
- The ability to modify a workout without deleting and re-entering it
- The calculation of progress based on strength
- The display of progress in a graphical form
- The use of a DatePicker object instead of integer inputs to input a date.

All of these are things that can be developed and integrated into the application in the future in order to improve it and ensure it meets all client requirements and makes it easier to use. Other features that could be developed in future include integrating a social feature- this would allow users to share their progress or workouts on media like Facebook or Twitter directly, or allow users to interact with each other within the app, seeing their friends' workouts perhaps for inspiration, motivation and accountability. Before social integration could be developed, the application would have to move towards a server-

based login system, as opposed to the use of local storage, so the database interactions would have to be redesigned (although I deliberately designed the database to support multiple users in the event that I did move towards a server-based option during development).

The display of progress as a graph instead of a numerical calculation would involve the learning of a graphical library but not much else, and could significantly improve the user's experience- this would be my first thing to complete in any further development, along with the creation of custom exercises.

Additionally, in the optional requirements, the functionality to have auto-generated workouts, or be able to add entire workouts in one go, could really enhance the user's experience if they often do the same or a very similar workout. The addition of this functionality would involve a method of grouping exercises into different workouts, and the ability for the user to select the workout and input only the sets/reps/date etc. It would make it much easier and faster to add details of a pre-created workout or previously input workout, perhaps by creating them in an additional page or choosing from workouts done on different dates. Auto-generated workouts and workout plans are features included in many published applications, although I'm unsure of how popular it is- my application intended not to teach clients how to work out but to facilitate them in their plans, and so although auto-generated workouts could give clients some inspiration, auto-generated workout plans perhaps are not the best approach.

Conclusion

On the whole, I would consider the final product to be at least a partial success. It implements every key requirement correctly and without any unforeseen bugs, and the clients' beta testing has resulted in excellent scores for usability and robustness. There is definitely room for improvements in terms of adding additional functionality, as some clients have realised that they need a few more features than originally thought to make the application viable for day-to-day use, and very few types of progress have been implemented so far. Both are things outlined to be completed in any further development. The application meets all its success criteria at least partially, but has some limitations in the usability and functionality changes required:

The app is suitable for the target customer group of adults, males and females, and complies with age restrictions and guidelines, and clear advice will be given to reduce health issues or injuries and to comply with the industry's legal and ethical requirements. The customer area is secured and accessed via password but no payment option, however this complies partly with legal requirements. The privacy and security of customer data is important and each member will have access to their area via secure login. This will ensure that ToKa Fitness is perceived as a reputable company and encourage existing customers who have access to digital devices to use more of the services provided by ToKa Fitness. Hopefully it will also encourage new customers to use the app and services.

The app can be accessed via mobile devices and a computer and has compatibility across different devices, Android and iOS, as requested, providing the customer with further options and flexibility of use especially accessibility for customers with sight loss. During the testing process, it was identified that the ‘My plan page ‘was where the summary of food and fitness training reports were displayed. The remaining calorie and fitness training targets were displayed in coloured text. Demonstrate that no input is required. This is a summary of daily calculations.

DRAFT

Exemplification Materials

Technical Qualification in Digital Production, Design and Development

Project 2 - commentary

Task 1 – commentary

Activity A (ii)

The student has selected an app to develop; they have identified some of the problems and effectively decomposed some of the issues. They have mentioned some potential risks and some regulatory and legal guidelines concerning software development.

The student has provided limited definitions of the functional and non-functional requirements and have written them into the user needs. The student has not mentioned potential risks and constraints concerning the development of the product.

Task 1 – commentary

Activity B

The proposed designs are fit for purpose. The student has demonstrated the aim of each object and that does not diminish other information which is relevant to the product.

The student has shown some design considerations, placing objects on the screen appropriately but with little information on layout and use of white space, elements or page specification. There is also no conventions or hierarchy navigation map, illustrating the direction the user would take or be able to navigate around the app.

Algorithm

In the algorithms section, the student is using pseudocode. The necessary process of decomposition has been used and explanations are clear. The algorithms demonstrate some understanding and the student has explained the inputs, process and outputs. Each step is defined, uses sensible names for variables, and indention has been provided although inconsistencies still exist.

Data requirements

The student has created a database with more than one table, then normalisation should be included. The student has provided variables with data types and explanations, which are appropriate. Data types are appropriate and normal conventions have been applied.

Test strategy

In the table, the student has mentioned the two main types of testing, Blackbox and Whitebox. They have indicated how they plan to check the primary and relevant data entry points.

The design consistently communicates the work, techniques, methods, and the formats are correct. The developer has shown excellent progression from the abstraction model to the algorithms.

Task 2 – commentary

The prototype has been planned and meets some of the criteria. Techniques and technical skills are appropriate and the use of API for the data table shows this. The structure is modular and code is well organised though it does lack comments. Indentation is used throughout the use of local variables and global variables. It is a basic interface, which is consistent, though it lacks clarity in relation to on-page elements such as fonts and colours.

Task 3 – commentary

The evaluation is good although it lacks detail on the iteration. The student has included feedback from testers but hasn't acted on it to show all changes. They have discussed KPIs and their success.

The student has provided some feedback from testers. The questionnaire and observations feedback and grading are appropriate. There have been some changes made based on user experience.

The assets used are appropriate for this product and the student has considered legal aspects.

The student has provided an evaluation that matches the needs of the client. They haven't clearly shown how they meet the functional and non-functional requirements.

Throughout the project, the quality of communication has been appropriate and somewhat technical in areas. The student has discussed techniques and methods used that are appropriate for the intended audience.