

How to Use this Template

1. Make a copy [File → Make a copy...]
2. Rename this file: “**Capstone_Stage1**”
3. Replace the text in green

Submission Instructions

1. After you’ve completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it “**Capstone Project**”
3. Add this document to your repo. Make sure it’s named “**Capstone_Stage1.pdf**”

[Description](#)

[Intended User](#)

[Features](#)

[User Interface Mocks](#)

[Screen 1](#)

[Screen 2](#)

[Key Considerations](#)

[How will your app handle data persistence?](#)

[Describe any corner cases in the UX.](#)

[Describe any libraries you’ll be using and share your reasoning for including them.](#)

[Describe how you will implement Google Play Services.](#)

[Next Steps: Required Tasks](#)

[Task 1: Project Setup](#)

[Task 2: Implement UI for Each Activity and Fragment](#)

[Task 3: Your Next Task](#)

[Task 4: Your Next Task](#)

[Task 5: Your Next Task](#)

GitHub Username: basu901

UFit Beta

Description

Stay Fit by keeping a track of the calories burnt and consumed and also monitor your body mass index. Equipped with a stopwatch and counter, this app also helps you to catalogue a routine to stay healthy. Users who incorporate running in their fitness programme can also view the distance covered and calories burnt, along-with the details of the route taken by them. The

total number of calories burnt in a month can also be compared through a graph on a day-to-day basis.

Intended User

Can be used by anyone who intends to stay fit and healthy.

Features

- Saves your height and weight to calculate your body mass index(BMI).
- You can log the amount of calories consumed daily, based on the food items in every meal.
- Records the intake of calories on a daily basis alongwith the amount of calories burnt.
- Displays the recorded data pertaining to calories in a day-to-day format for you to monitor and build your fitness routine.
- Allows you to make a workout routine for yourself which you can follow.
- Shows the route taken by you while running, along with the distance travelled, speed and calories burnt during the run.
- Has a supplementary stopwatch.

User Interface Mocks

Screen 1

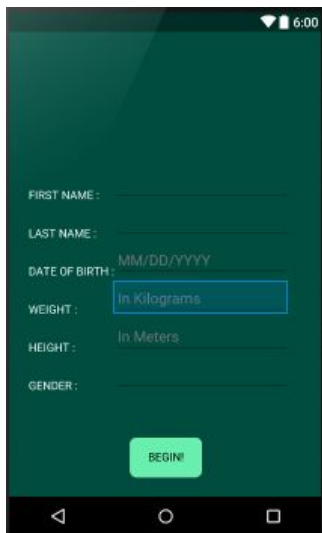


Fig. 1:Login

The login screen which prompts the user for personal information required for performing app functionalities.

Screen 2

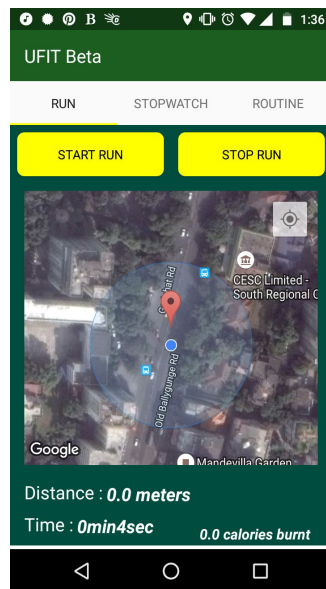


Fig.2: The Main Display

The main screen which contains a tab for the user to navigate between a stopwatch, a routine builder and a screen displaying the details of a user's run. Fig 1 displays the screen which provides information during the user's run.

Screen 3

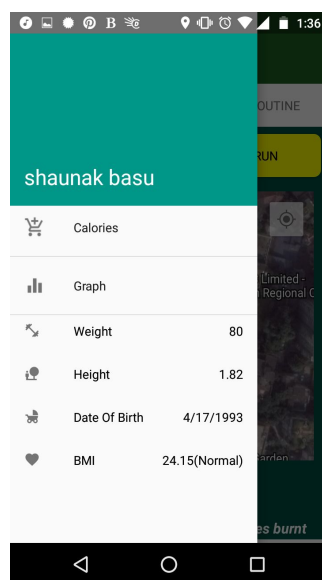


Fig. 3: The Menu

A view displaying the user information required for calculating the BMI(Body Mass index). This view also contains the fields to navigate to other functionalities of the application.

Screen 4

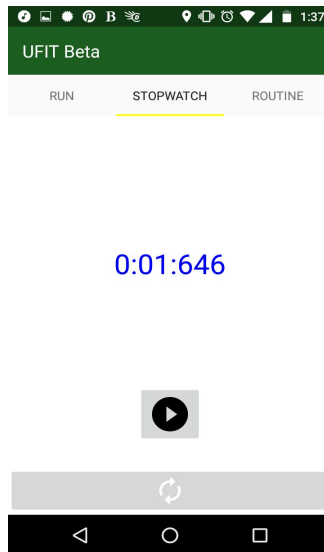


Fig. 4:Stopwatch Screen

The screen displaying the stopwatch of the application.

Screen 5

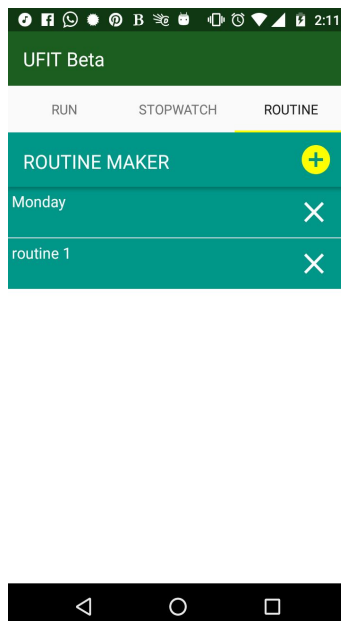


Fig. 5: Routine Screen

The routine screen which allows the user to make a new routine. This screen also shows the headers of different routines.

Screen 6

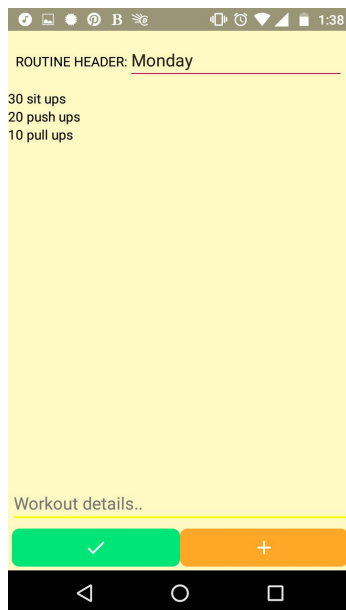


Fig. 6: Create Routine

The screen which is displayed to user when he/she wants to create a new routine. The user navigates to this screen from the Routine Screen shown above.

Screen 7

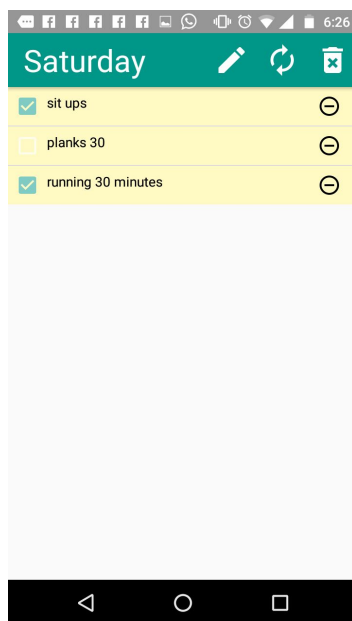


Fig. 7:Routine Displayer

This screen shows the full routine made by the user with the ability to keep track of the exercises completed in the routine.

Screen 8

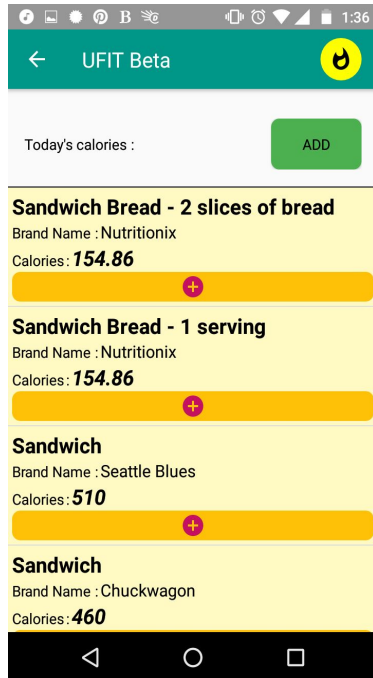


Fig. 8:Calorie Page

This page allows the user to log the calories consumed in the day by adding the calorie content in each food item for every meal.

Screen 9

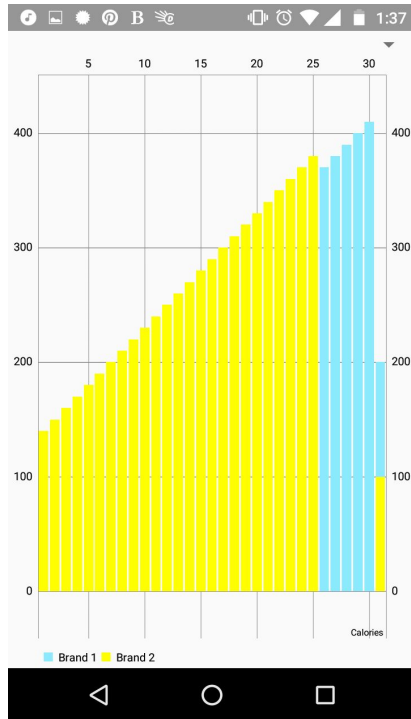


Fig. 9:Calorie Graph

The calorie graph screen shows gives graphical comparison of the number of calories burnt and consumed on each day of the month.

Screen 10



Fig.10: App Widget

The App Widget which contains information pertaining to how much calories was consumed or burnt by the user on the present day.

Key Considerations

How will your app handle data persistence?

The app will build three content providers for data persistence. The functionality of each is:

- CalorieIntakeProvider : Contains information regarding the amount of calories consumed by the user per day.
- CalorieBurntProvider : Contains information regarding the amount of calories burnt by the user per day.
- RoutineDetailsProvider : Contains all the routines made by the user.

The app will also use shared preferences to store the personal information of the user and selectively display the login screen or the main display depending on the value of a particular shared preference.

Describe any corner cases in the UX.

Corner case include the following:

- When the user presses the back button from the graph activity or the activity displaying the calorie of each food item, the user return to the main activity containing the tab layout.
- The user also returns to the main screen from the activity which creates the routine, when the back button is pressed.
- Similarly from the activity showing the details of a particular routine, the user returns to the main screen on pressing the back button.

Describe any libraries you'll be using and share your reasoning for including them.

The application will be using the following libraries:

- **net.simonvt.schematic:schematic-compiler** and **net.simonvt.schematic:schematic** for building the content providers for this app
- **MPAndroidChart,com.github.PhilJay:MPAndroidChart** for making the graph used for displaying the calorie values
- **SmoothProgressBar,com.github.castorflex.smoothprogressbar:library** for displaying a progress bar as the data is loaded in the background.

Describe how you will implement Google Play Services.

The following google play services will be used:

- **Google Maps**, for displaying the position of the user on a map as he/she goes for a run. The map will show the user the path he/she is taking and also the start and stop positions.
- **Location Services**, will be used to monitor the current position of the user while he/she is running. From the information received, the distance travelled will be calculated.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Project Setup

We start off by creating a new project in Android Studio and making it applicable for only Phone and Tablet modules. The minSdk version is set to 17. The libraries need to be configured next:

- We have to include google play services,
com.google.android.gms:play-services:9.4.0 in the build.gradle file of the app module.(9.4.0 was the latest version at the time of implementing this project)
- Include libraries for support and design:**com.android.support:design:24.2.1**(24.2.1 was the latest version at the time of implementing this project)
- Include libraries for implementing recycler view
com.android.support:recyclerview-v7:24.2.1(24.2.1 was the latest version at the time of implementing this project)
- Include the third party libraries
- Modify the manifest file to include the gms version.This involves including the following tag in the manifest,**android:name="com.google.android.gms.version"**
android:value="@integer/google_play_services_version" in a meta tag.
- **com.android.support:appcompat-v7:24.2.1** is automatically included on project creation.

With these attributes set we can begin writing code.

Task 2: Implement UI for Each Activity and Fragment

Use an App theme which has a parent: Theme.AppCompat.Light.NoActionBar

1)Building the UI for the login screen(activity_login.xml):

- This screen contains an ImageView containing the application name.
- TextViews are then inserted which displays the labels of the information required from the user.
- EditText views are then laid out for the user to enter the required information.
- A Button is added for the user to navigate onto the main screen.

2)Building the UI for the main screen(activity_main.xml):

- The main screen includes a DrawerLayout for implementing a NavigationView which will contain menu items.
- A NavigationView for populating menus.
- A TabLayout which will hold the different functionalities of the app and a ViewPager to enable swiping between these functionalities.
- An AppBarLayout containing a Toolbar is implemented for displaying header information.(main_activity_toolbar.xml)

3)Building the Run Page UI(fragment_map.xml):

- Buttons are included for users to notify when the users start running and when they are finished with their run.
- TextViews to display the labels as well as the corresponding information to the user when he/she completes his/her run.
- A fragment which will contain the GoogleMap
- A scroll view to allow the user to scroll through the information.

4)Building the Stopwatch UI(stopwatch.xml):

- A TextView which will display the interval between start and pause.
- An ImageButton to start and stop the stopwatch.
- An ImageButton to reset the stopwatch.

5)Building the Routine UI:

- Building the Routine Maker Page(routine_maker.xml):
 - An AppBar which contains the Toolbar for displaying header information.
 - The Toolbar contains an ImageButton which allows the user to add a routine on clicking, a TextView to convey the functionality of the ImageButton
 - A ListView which will contain all the routines made(routine_item.xml).
 - A TextView displaying the routine_header.
 - An ImageButton allowing the user to delete the routine.
- Building the Routine Builder(routine_detail.xml)
 - A TextView showing the label of "Routine Header"
 - An EditText view to allow the user to enter the name of the routine.
 - An EditText view to allow the user to enter each item of the routine.
 - Two ImageButtons ,one to enter the routine items and the other to confirm that the entire routine has been built and should be saved.
 - TextViews are dynamically inserted into a LinearLayout contained in a ScrollView. These TextViews hold the text of each routine item which is inserted by user to give convey which routine items have been added.
- Building the Routine Display(routine_display.xml)
 - An AppBar and its associated Toolbar

- The Toolbar elements include a TextView to display the name of the routine,three image buttons,one for editing the routine,one for unchecking all the items in the routine and the other for deleting the routine completely.
- A ListView which will populate all the items in the routine(routine_detail_item.xml).
 - A checkbox for checking or unchecking an item.
 - A TextView to display the item name.
 - An ImageButton to delete the item from the routine

6) Building the elements of the NavigationView:

- Building the custom menu items (nav_text_fields.xml):
 - An ImageView containing a relevant image.
 - A TextView displaying the menu name.
 - A TextView displaying the menu information.
- Building the navigation header(nav_header.xml)
 - Contains a TextView inside a linear layout to display the user's name.

7) Building the Calorie adder UI(activity_calorie.xml)

- Contains an AppBar with its Toolbar.
- The Toolbar contains an ImageButton which allows the user to add calories burnt manually
- A Smooth Progress Bar from a third party library which conveys that data is being fetched and loaded.
- TextViews to display to the user the amount of calories consumed for the present day.
- A Button to allow the user to add calories from a meal
- TextViews conveying the API information from where the results are being retrieved.
- A RecyclerView to populate the results of the search(calorie_list_item.xml)
 - TextView indicating the labels and the corresponding values for the data retrieved.
 - An ImageButton to allow the user to add the calorie values.

8)Building the Graph UI(activity_graph):

- Contains an AppBar and its associated Toolbar.
- A bar chart view from a third party graph library.

9)Building the confirmation dialogue(confirmation_dialogue.xml):

- Contains a TextView asking the user whether he/she is certain that the action should be carried out.

10)Building the Calorie Burn Dialogue(prompt_calorie_burn):

- TextViews displaying the labels for the relevant information required from the user.
- EditText Views to allow the user to enter the information.

11)Building the Attribute Setter UI(prompt_dialogue.xml):

- A textview asking for the relevant user attribute information.
- An EditText view to input the data.

12)Building the widget UI(widget_layout_info.xml):

- Contains TextView to display the label and the information regarding the calorie metrics for the present day.

Task 3: Handle Error Cases

This app requires a lot of input from the user and the following error cases have been handled.

- Only alphabets error case:
 - Applied during login when the user inputs his/her first name,last name and gender .Relevant snackbar message is displayed conveying when any other format is inserted conveying that the input is invalid .
 - Same method applied during querying for food items.Relevant snackbar message is displayed.
- Only whole numbers with no decimal point:
 - Applied when the user inputs his/her date of birth.The input type of the edit text does not allow a decimal point.The input type also prevents alphabets on being inserted.
 - Same method applied while calculating number of calories by taking input from the user on the workout time field.
- To check that the date of birth is valid the input is parsed to a valid date and according a snackbar message is displayed when the date is invalid.

Task 4: Implementing Accessibility and Layout Mirroring and Sstrings

- Accessibility is implemented by setting **contentDescription** for all ImageButtons in the layout files which have been described above.
- Layout mirroring is implemented by using **End** and **Start** attributes and avoiding the use of **Right** and **Left** attributes.
- All strings will be stored under res/values/strings.xml and accessed from there by using getResources().getString(string id).

Task 5: Implementing the App Widget

The widget displays the amount of calories consumed and burnt for the present day.

- Specify the widget characteristics in a file called `widget_info_layout.xml`. Make a new directory under the **res** folder and place the file there. Contains information regarding the update time,height,width and category
- Build the UI as specified above.
- Assign required views to reference variables.
- Make a new class which extends the `AppWidgetProvider` class and override `onUpdate()`. Here, the content providers are queried to get the correct information and all instances of the widget is then updated.
- Add the widget in the manifest file.

Task 6: Implementing the Main Activity Display

- Build the UI as specified above.
- Assign required views to reference variables.
- Add the activity to manifest.
- Add the fragments to the viewpager through a custom adapter.
- Set the number of tabs and the title for each.
- Connect the viewpager and tablayout through listeners to display the correct fragment on swiping.
- Populate the custom menu items and write corresponding functions which will be triggered on clicking each menu item.

Task 7: Implementing the Google Play Services

The Run Fragment: This fragment uses google play services for google maps and location.

- After the initial project configuration,implement the callbacks of the `GoogleApiClient` for location.
- Build the UI as specified above.
- Assign required views to reference variables.
- Add the permissions in the manifest.
- Ask the user for permission for accessing fine location.Handle the results from `PackageManager`.If the user grants permission ask the user if location services can be enabled.
- Use a `Location Request` to handle location privileges.Check the status of location through `Location Request` and handle the status in `onResult`. If location services is not

on prompt a dialogue asking for the services to be on and handle the result in onActivityResult.

- When the GoogleApiClient is connected update the UI from onConnected.
- Implement OnMapReady callback and set the map type in onMapReady.
- Use the start button to note the time of SystemClock.elapsedRealtime.Add a marker option on the map to indicate the starting position.Retrieve the latitude and longitude of through the location object received in onLocationChanged.
- Calculate the distance between each latitude and longitude received through a helper function.
- Use the stop button to note the time of SystemClock.elapsedRealtime.Retrieve the last location and stop further location updates.
- Calculate the time elapsed between the two noted times and find the speed by dividing the total distance by the time.
- Calculate the calories burnt through a helper function and insert in the database.
- Update the textviews in the UI.

Task 8: Implementing the Stopwatch

- Build the UI as specified above.
- Assign required views to reference variables.
- Set a listener on the play/pause button.On clicking the button the SystemClock.uptimeMillis is noted and handler is started which runs a thread which controls the main UI.
- The runnable thread also notes the SystemClock.uptimeMillis and the difference between the two noted values is converted to the minutes,seconds,milliseconds format and displayed to the user.
- The listener contains a control button which starts the handler immediately or removes its callback depending on whether the button is in start or pause mode respectively.
- The reset image button resets all the values and also removes any callbacks on the handler.

Task 8: Implementing the Routine Maker

- Available Routines:
 - Build the UI as specified above.
 - Assign required views to reference variables.
 - Initialize a loader and start loader through loader manager
 - Use a custom adapter which extends a cursor adapter and load all the routines through a cursor loader after querying the database.
 - Load the data in onLoadFinished.

- Add a listener to the add routine image button which will trigger a new activity to create a new routine.
- Add a listener on the delete button which will remove the routine permanently from the database.
- New Routine:
 - Build the UI as specified above.
 - Assign required views to reference variables.
 - Add the activity to the manifest
 - Register a listener on the add item button which will add the input to the database by querying the content provider.
 - If the header of the routine is not specified display a snackbar message asking the user to do so.
 - On clicking the done button check whether the user navigated here from a previously built routine.
 - If yes,check whether the routine header has been changed.If the header has been changed,**update** all the rows which contained the previous header otherwise navigate to the display activity.
 - If no navigate to the display activity.
- Routine Display
 - Build the UI as specified above.
 - Assign required views to reference variables.
 - Add the activity to the manifest
 - Initialize a loader and start loader through loader manager
 - Use a custom adapter which extends a cursor adapter and load all the routine items through a cursor loader after querying the database.
 - Load the data in onLoadFinished.
 - Register a listener for on the checkbox and accordingly update the status of the checkbox in the database,whenever clicked.
 - Register a listener for the delete image button which will delete all the items from the routine and in effect delete the entire routine and navigate back to the main display screen.
 - Register a listener for the delete button on each individual routine item which will delete the corresponding routine item and call notifyDataSetChanged on the custom adapter.
 - Register a listener for the uncheck button which will uncheck all the items and update their status in the database.Update the UI by calling notifyDataSetChanged on the custom adapter.
 - Register a listener on the edit button which will start the new routine activity through an intent and put an intent extra containing the routine name which will specify that this routine is for editing.

Task 8: Implementing the Calorie Adder

- Build the UI as specified above.
- Assign required views to reference variables.
- Add the activity to the manifest
- Query the database for the amount of calories consumed on the present day and update the UI.
- Add the link to the textview which helps the user to navigate to the API.
- Add a listener on the add calorie button which will prompt a dialogue for supplying the food item name.
- Launch an intent service if the input is valid, to start an HttpURLConnection for querying the API in the background.
- Display the progress bar.
- Register a broadcast receiver which will receive the response.
- Retrieve the desired values from the JSON response.
- Use a recyclerview adapter to load the data into the view.
 - Add a listener on the calorie adder image button which will update the number of calorie for the day in the database and also update the textview in the UI
- Disable the progress bar.
- Add a listener on the image button in the toolbar which notes the amount of calories burnt by the user by taking in relevant inputs. If the input is valid, use a helper function to calculate the number of calories burnt.
- Update the database and show the user the number of calories he/she has burnt in the fitness routine.

Task 9: Implementing the Graph

- Build the UI as specified above.
- Assign required views to reference variables.
- Add the activity to the manifest
- Query the respective content providers for the amount of calories burnt and consumed for each day of the entire month.
- Store the data in Bar Entry objects and then add these objects to a Bar Data Set object.
- Update the graph.

Task 10: Implementing the Login Screen

- Build the UI as specified above.
- Assign required views to reference variables.
- Add the activity to the manifest
- Handle the error cases mentioned through helper functions
- Add a listener on the button to start the Main display screen through an intent with a simple transition.

Task 11: Gradle Tasks

- Generate the signed APK
- Generate keystore and store in project root folder.
- Access the passwords through relative path by adding signingConfigs to app's build.gradle
- Put the signingconfig in release scope.

Add as many tasks as you need to complete your app.

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"