

**(Flutter)**

Lab Manual

**Lab 1 – Session 1**

**Total Time:**

**3 hours**

**Pre-Lab Activities:**

* No Pre-Lab Activity

**Learning Outcomes:**

* Perform the execution, debugging, testing, and profiling of mobile apps in modern IDEs.

**Lab Tasks:**

* + To install and configure Android Studio.
  + To install and configure the Emulators.
  + To explore the Android Studio functionalities.

**Student Activities:**

* + To explore Android Studio installation.
  + To explore the Emulators.
  + Android Studio functionalities.

**Lab Solutions:**

# Lab Tasks

# 1 – Install Android Studio.

**System Requirements**

* Microsoft Windows 7/8/10 (32-bit or 64-bit)
* 4 GB RAM minimum, 8 GB RAM recommended (plus 1 GB for the Android Emulator)
* 2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE plus 1.5 GB for Android SDK and emulator system image)
* 1280 x 800 minimum screen resolution

**Installation:** Perform the following steps to install the Android Studio.

**Step 1:**Head over to [**this link**](https://developer.android.com/studio/#downloads) to get the Android Studio executable or zip file.

**Step 2:**Click on the **Download Android Studio** Button.

Graphical user interface, text, application

Description automatically generated

Click on the “I have read and agree with the above terms and conditions” checkbox followed by the download button.

Graphical user interface, text, application, email

Description automatically generated

Click on the Save file button in the appeared prompt box and the file will start downloading.

**Step 3:**After the downloading has finished, open the file from downloads and run it. It will prompt the following dialog box.

Graphical user interface, text, application, chat or text message

Description automatically generated

Click on next. In the next prompt, it’ll ask for a path for installation. Choose a path and hit next.

**Step 4:**It will start the installation, and once it is completed, it will be like the image shown below.

Graphical user interface, text, application

Description automatically generated

Click on next.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Step 5:**Once “**Finish**” is clicked, it will ask whether the previous settings need to be imported [if the android studio had been installed earlier], or not. It is better to choose the ‘Don’t import Settings option’.

Graphical user interface, text, application

Description automatically generated

Click the **OK**button.

**Step 6:**This will start the Android Studio.

Graphical user interface, application

Description automatically generated

Meanwhile, it will be finding the available SDK components.

Graphical user interface, text, application

Description automatically generated

**Step 7:**After it has found the SDK components, it will redirect to the Welcome dialog box.

Graphical user interface, application

Description automatically generated

Click on **Next**.

Graphical user interface, text, application

Description automatically generated

Choose Standard and click on Next. Now choose the theme, whether the **Light**theme or the **Dark**one. The light one is called the **IntelliJ**theme whereas the dark theme is called **Darcula**. Choose as required.

Graphical user interface, text, application

Description automatically generated

Click on the **Next**button.

**Step 8:**Now it is time to download the SDK components.

Table

Description automatically generated

Click on Finish. Components begin to download let it complete.

Text

Description automatically generated

The Android Studio has been successfully configured. Now it’s time to launch and build apps. Click on the Finish button to launch it.

**Step 9:**Click on **Start a new Android Studio project** to build a new app.

Graphical user interface, application

Description automatically generated

# 2 – Create and manage virtual devices

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the [Android Emulator](https://developer.android.com/studio/run/emulator). The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.

To open the AVD Manager, do one of the following:

* Select **Tools > AVD Manager**.
* Click **AVD Manager** AVD Manager icon in the toolbar.

A picture containing text

Description automatically generated

## About AVDs

An AVD contains a hardware profile, system image, storage area, skin, and other properties.

We recommend that you create an AVD for each system image that your app could potentially support based on the [<uses-sdk>](https://developer.android.com/guide/topics/manifest/uses-sdk-element) setting in your manifest.

### Hardware profile

The hardware profile defines the characteristics of a device as shipped from the factory. The AVD Manager comes preloaded with certain hardware profiles, such as Pixel devices, and you can define or customize the hardware profiles as needed.

Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully [CTS](https://source.android.com/compatibility/cts/) compliant and may use system images that include the Play Store app.

### System images

A system image labeled with **Google APIs** includes access to [Google Play services](https://developers.google.com/android/guides/overview). A system image labeled with the Google Play logo in the **Play Store** column includes the Google Play Store app and access to Google Play services, including a **Google Play** tab in the **Extended controls** dialog that provides a convenient button for updating Google Play services on the device.

To ensure app security and a consistent experience with physical devices, system images with the Google Play Store included are signed with a release key, which means that you cannot get elevated privileges (root) with these images. If you require elevated privileges (root) to aid with your app troubleshooting, you can use the Android Open Source Project (AOSP) system images that do not include Google apps or services.

### Storage area

The AVD has a dedicated storage area on your development machine. It stores the device user data, such as installed apps and settings, as well as an emulated SD card. If needed, you can use the AVD Manager to wipe user data, so the device has the same data as if it were new.

### Skin

An emulator skin specifies the appearance of a device. The AVD Manager provides some predefined skins. You can also define your own, or use skins provided by third parties.

### AVD and app features

Be sure your AVD definition includes the device features your app depends on. See [Hardware Profile Properties](https://developer.android.com/studio/run/managing-avds#hpproperties) and [AVD Properties](https://developer.android.com/studio/run/managing-avds#avdproperties) for lists of features you can define in your AVDs.

## Create an AVD

**Tip:** If you want to launch your app into an emulator, instead [run your app from Android Studio](https://developer.android.com/studio/run#RunningApp) and then in the **Select Deployment Target** dialog that appears, click **Create New Virtual Device**.

**To create a new AVD:**

1. Open the AVD Manager by clicking **Tools > AVD Manager**. A picture containing text

   Description automatically generated
2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog.

The **Select Hardware** page appears.

Graphical user interface

Description automatically generated

Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully [CTS](https://source.android.com/compatibility/cts/) compliant and may use system images that include the Play Store app.

1. Select a hardware profile, and then click **Next**.

If you don't see the hardware profile you want, you can [create](https://developer.android.com/studio/run/managing-avds#createhp) or [import](https://developer.android.com/studio/run/managing-avds#importexporthp) a hardware profile.

The **System Image** page appears.

Graphical user interface, application

Description automatically generated

1. Select the system image for a particular API level, and then click **Next**.

The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the [minSdkVersion](https://developer.android.com/guide/topics/manifest/uses-sdk-element) attribute of the app manifest file. For more information about the relationship between system API level and minSdkVersion, see [Versioning Your Apps](https://developer.android.com/studio/publish/versioning).

If your app declares a [<uses-library>](https://developer.android.com/guide/topics/manifest/uses-library-element) element in the manifest file, the app requires a system image in which that external library is present. If you want to run your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The **Verify Configuration** page appears.

Graphical user interface

Description automatically generated

1. Change [AVD properties](https://developer.android.com/studio/run/managing-avds#avdproperties) as needed, and then click **Finish**.

Click **Show Advanced Settings** to show more settings, such as the skin.

The new AVD appears in the **Your Virtual Devices** page or the **Select Deployment Target** dialog.

To create an AVD starting with a copy:

1. From the [**Your Virtual Devices**](https://developer.android.com/studio/run/managing-avds#viewing) page of the AVD Manager, right-click an AVD and select **Duplicate**.

Or click Menu  and select **Duplicate**.

The [**Verify Configuration**](https://developer.android.com/studio/run/managing-avds#verifyconfigpage) page appears.

1. Click **Change** or **Previous** if you need to make changes on the [**System Image**](https://developer.android.com/studio/run/managing-avds#systemimagepage) and [**Select Hardware**](https://developer.android.com/studio/run/managing-avds#selecthardwarepage) pages.
2. Make your changes, and then click **Finish**.

The AVD appears in the **Your Virtual Devices** page.

## Create a hardware profile

The AVD Manager provides predefined hardware profiles for common devices so you can easily add them to your AVD definitions. If you need to define a different device, you can create a new hardware profile. You can define a new hardware profile from the beginning, or [copy a hardware profile](https://developer.android.com/studio/run/managing-avds#copyavd) as a start. The preloaded hardware profiles aren't editable.

To create a new hardware profile from the beginning:

1. In the [**Select Hardware**](https://developer.android.com/studio/run/managing-avds#selecthardwarepage) page, click **New Hardware Profile**.
2. In the **Configure Hardware Profile** page, change the [hardware profile properties](https://developer.android.com/studio/run/managing-avds#hpproperties) as needed.
3. Click **Finish**.

Your new hardware profile appears in the **Select Hardware** page. You can optionally [create an AVD](https://developer.android.com/studio/run/managing-avds#createavd) that uses the hardware profile by clicking **Next**. Or, click **Cancel** to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

To create a hardware profile starting with a copy:

1. In the [**Select Hardware**](https://developer.android.com/studio/run/managing-avds#selecthardwarepage) page, select a hardware profile and click **Clone Device**.

Or right-click a hardware profile and select **Clone**.

1. In the **Configure Hardware Profile** page, change the [hardware profile properties](https://developer.android.com/studio/run/managing-avds#hpproperties) as needed.
2. Click **Finish**.

Your new hardware profile appears in the **Select Hardware** page. You can optionally [create an AVD](https://developer.android.com/studio/run/managing-avds#createavd) that uses the hardware profile by clicking **Next**. Or, click **Cancel** to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

## Edit existing AVDs

From the [**Your Virtual Devices**](https://developer.android.com/studio/run/managing-avds#viewing) page, you can perform the following operations on an existing AVD:

* To edit an AVD, click **Edit this AVD**  and [make your changes](https://developer.android.com/studio/run/managing-avds#copyavd).
* To delete an AVD, right-click an AVD and select **Delete**. Or click Menu  and select **Delete**.
* To show the associated AVD .ini and .img files on disk, right-click an AVD and select **Show on Disk**. Or click Menu  and select **Show on Disk**.
* To view AVD configuration details that you can include in any bug reports to the Android Studio team, right-click an AVD and select **View Details**. Or click Menu  and select **View Details**.

## Edit existing hardware profiles

From the [**Select Hardware**](https://developer.android.com/studio/run/managing-avds#selecthardwarepage) page, you can perform the following operations on an existing hardware profile:

* To edit a hardware profile, select it and click **Edit Device**. Or right-click a hardware profile and select **Edit**. Next, [make your changes](https://developer.android.com/studio/run/managing-avds#copyhp).
* To delete a hardware profile, right-click it and select **Delete**.

You can't edit or delete the predefined hardware profiles.

## Run and stop an emulator, and clear data

From the [**Your Virtual Devices**](https://developer.android.com/studio/run/managing-avds#viewing) page, you can perform the following operations on an emulator:

* To run an emulator that uses an AVD, double-click the AVD. Or click **Launch** .
* To stop a running emulator, right-click an AVD and select **Stop**. Or click Menu  and select **Stop**.
* To clear the data for an emulator, and return it to the same state as when it was first defined, right-click an AVD and select **Wipe Data**. Or click Menu  and select **Wipe Data**.

## Import and export hardware profiles

From the [**Select Hardware**](https://developer.android.com/studio/run/managing-avds#selecthardwarepage) page, you can import and export hardware profiles:

* To import a hardware profile, click **Import Hardware Profiles** and select the XML file containing the definition on your computer.
* To export a hardware profile, right-click it and select **Export**. Specify the location where you want to store the XML file containing the definition.

## Hardware profile properties

You can specify the following properties of hardware profiles in the [**Configure Hardware Profile**](https://developer.android.com/studio/run/managing-avds#createhp) page. AVD configuration properties override hardware profile properties, and emulator properties that you set while the emulator is running override them both.

The predefined hardware profiles included with the AVD Manager aren't editable. However, you can copy them and edit the copies.

|  |  |
| --- | --- |
| **Hardware Profile Property** | **Description** |
| Device Name | Name of the hardware profile. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (\_), parentheses ( () ), and spaces. The name of the file storing the hardware profile is derived from the hardware profile name. |
| Device Type | Select one of the following:   * Phone/Tablet * Wear OS * Android TV * Chrome OS Device * Android Automotive |
| Screen Size | The physical size of the screen, in inches, measured at the diagonal. If the size is larger than your computer screen, it’s reduced in size at launch. |
| Screen Resolution | Type a width and height in pixels to specify the total number of pixels on the simulated screen. |
| Round | Select this option if the device has a round screen, such as some Wear OS devices. |
| Memory: RAM | Type a RAM size for the device and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). |
| Input: Has Hardware Buttons (Back/Home/Menu) | Select this option if your device has hardware navigation buttons. Deselect it if these buttons are implemented in software only. If you select this option, the buttons won’t appear on the screen. You can use the emulator side panel to "press" the buttons, in either case. |
| Input: Has Hardware Keyboard | Select this option if your device has a hardware keyboard. Deselect it if it doesn’t. If you select this option, a keyboard won’t appear on the screen. You can use your computer keyboard to send keystrokes to the emulator, in either case. |
| Navigation Style | Select one of the following:   * None - No hardware controls. Navigation is through the software. * D-pad - Directional Pad support. * Trackball * Wheel   These options are for actual hardware controls on the device itself. However, the events sent to the device by an external controller are the same. |
| Supported Device States | Select one or both options:   * Portrait - Oriented taller than wide. * Landscape - Oriented wider than tall.   If you select both, you can switch between orientations in the emulator. You must select at least one option to continue. |
| Cameras | To enable the camera, select one or both options:   * Back-Facing Camera - The lens faces away from the user. * Front-Facing Camera - The lens faces toward the user.   Later, you can use a webcam or a photo provided by the emulator to simulate taking a photo with the camera. |
| Sensors: Accelerometer | Select if the device has hardware that helps the device determine its orientation. |
| Sensors: Gyroscope | Select if the device has hardware that detects rotation or twist. In combination with an accelerometer, it can provide smoother orientation detection and support a six-axis orientation system. |
| Sensors: GPS | Select if the device has hardware that supports the Global Positioning System (GPS) satellite-based navigation system. |
| Sensors: Proximity Sensor | Select if the device has hardware that detects if the device is close to your face during a phone call to disable input from the screen. |
| Default Skin | Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the resolution can mean that the screen is cut off, so you can't see the whole screen. See [Create an emulator skin](https://developer.android.com/tools/devices/managing-avds#skins) for more information. |

## AVD properties

You can specify the following properties for AVD configurations in the [**Verify Configuration**](https://developer.android.com/studio/run/managing-avds#verifyconfigpage) page. The AVD configuration specifies the interaction between the development computer and the emulator, as well as properties you want to override in the hardware profile.

AVD configuration properties override hardware profile properties. Emulator properties that you set while the emulator is running override them both.

|  |  |
| --- | --- |
| **AVD Property** | **Description** |
| AVD Name | Name of the AVD. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (\_), parentheses ( () ), dashes (-), and spaces. The name of the file storing the AVD configuration is derived from the AVD name. |
| AVD ID (Advanced) | The AVD filename is derived from the ID, and you can use the ID to refer to the AVD from the command line. |
| Hardware Profile | Click **Change** to select a different hardware profile in the [Select Hardware](https://developer.android.com/studio/run/managing-avds#selecthardwarepage) page. |
| System Image | Click **Change** to select a different system image in the [System Image](https://developer.android.com/studio/run/managing-avds#systemimagepage) page. An active internet connection is required to download a new image. |
| Startup Orientation | Select one option for the initial emulator orientation:   * Portrait - Oriented taller than wide. * Landscape - Oriented wider than tall.   An option is enabled only if it’s selected in the hardware profile. When running the AVD in the emulator, you can change the orientation if both portrait and landscape are supported in the hardware profile. |
| Camera (Advanced) | To enable a camera, select one or both options:   * Front - The lens faces away from the user. * Back - The lens faces toward the user.   The **Emulated** setting produces a software-generated image, while the **Webcam** setting uses your development computer webcam to take a picture.  This option is available only if it's selected in the hardware profile; it's not available for Wear OS and Android TV. |
| Network: Speed (Advanced) | Select a network protocol to determine the speed of data transfer:   * GSM - Global System for Mobile Communications * HSCSD - High-Speed Circuit-Switched Data * GPRS - Generic Packet Radio Service * EDGE - Enhanced Data rates for GSM Evolution * UMTS - Universal Mobile Telecommunications System * HSDPA - High-Speed Downlink Packet Access * LTE - Long-Term Evolution * Full (default) - Transfer data as quickly as your computer allows. |
| Network: Latency (Advanced) | Select a network protocol to set how much time (delay) it takes for the protocol to transfer a data packet from one point to another point. |
| Emulated Performance: Graphics | Select how graphics are rendered in the emulator:   * Hardware - Use your computer graphics card for faster rendering. * Software - Emulate the graphics in software, which is useful if you're having a problem with rendering in your graphics card. * Automatic - Let the emulator decide the best option based on your graphics card. |
| Emulated Performance: Boot option (Advanced) | * Cold boot - Start the device each time by powering up from the device-off state. * Quick boot - Start the device by loading the device state from a saved snapshot. For details, see [Run the emulator with Quick Boot](https://developer.android.com/studio/run/emulator#quickboot). |
| Emulated Performance: Multi-Core CPU (Advanced) | Select the number of processor cores on your computer that you’d like to use for the emulator. Using more processor cores speeds up the emulator. |
| Memory and Storage: RAM | The amount of RAM on the device. This value is set by the hardware manufacturer, but you can override it, if needed, such as for faster emulator operation. Increasing the size uses more resources on your computer. Type a RAM size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). |
| Memory and Storage: VM Heap | The VM heap size. This value is set by the hardware manufacturer, but you can override it, if needed. Type a heap size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). For more information on Android VMs, see [Memory Management for Different Virtual Machines](https://developer.android.com/tools/help/am-memory#vm). |
| Memory and Storage: Internal Storage | The amount of nonremovable memory space available on the device. This value is set by the hardware manufacturer, but you can override it, if needed. Type a size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). |
| Memory and Storage: SD Card | The amount of removable memory space available to store data on the device. To use a virtual SD card managed by Android Studio, select **Studio-managed**, type a size, and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). A minimum of 100 MB is recommended to use the camera. To manage the space in a file, select **External file** and click **...** to specify the file and location. For more information, see [mksdcard](https://developer.android.com/tools/help/mksdcard) and [AVD data directory](https://developer.android.com/studio/run/emulator-commandline#data-filedir). |
| Device Frame: Enable Device Frame | Select to enable a frame around the emulator window that mimics the look of a real device. |
| Custom Skin Definition (Advanced) | Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the skin can mean that the screen is cut off, so you can't see the whole screen. See [Create an emulator skin](https://developer.android.com/tools/devices/managing-avds#skins) for more information. |
| Keyboard: Enable Keyboard Input (Advanced) | Select this option if you want to use your hardware keyboard to interact with the emulator. It's disabled for Wear OS and Android TV. |

# 3 - Run apps on a hardware device

When building an Android app, it's important that you always test your app on a real device before releasing it to users. This page describes how to set up your development environment and Android device for testing and debugging over an Android Debug Bridge (ADB) connection.

## Set up a device for development

Before you can start debugging on your device, decide if you want to connect the device to using a USB cable or Wi-Fi. Then do the following:

1. On the device, open the **Settings** app, select **Developer options**, and then enable **USB debugging** (if applicable).

**Note:** If you do not see **Developer options**, follow the instructions to [enable developer options](https://developer.android.com/studio/debug/dev-options).

## Connect to your device using USB

## When you are set up and plugged in over USB, you can click **Run**  in Android Studio to [build and run your app on the device](https://developer.android.com/studio/run).

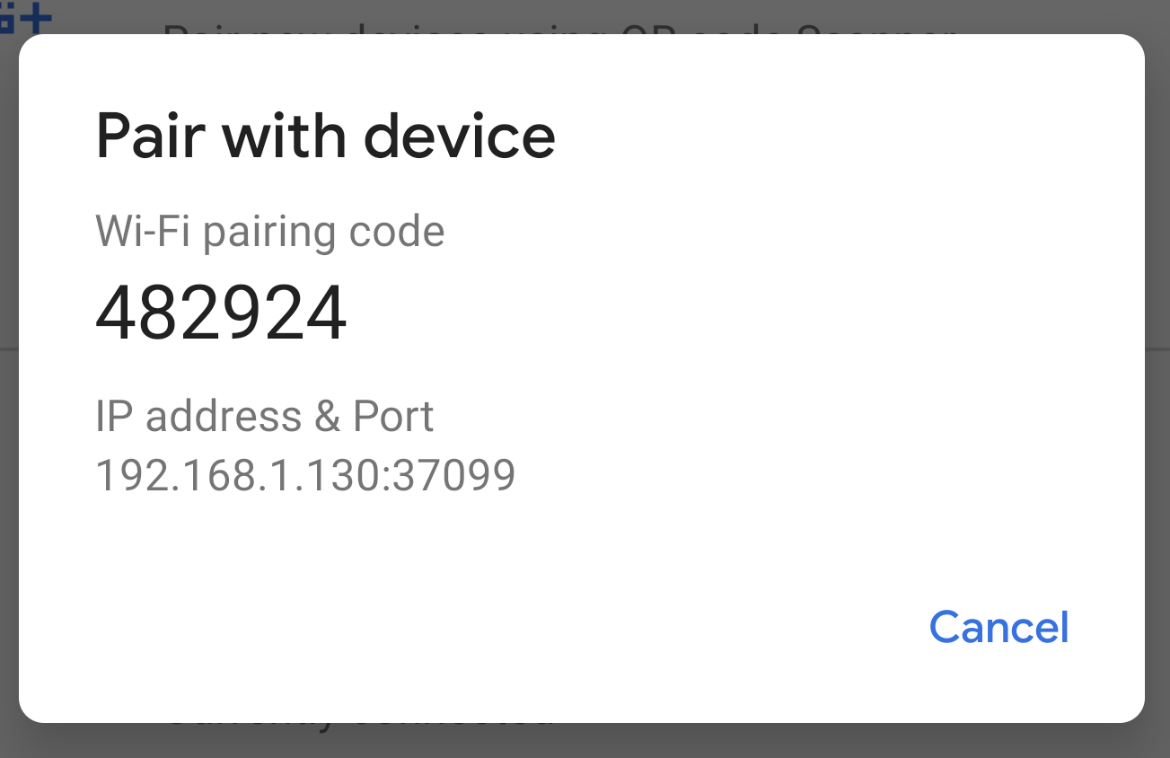
You can also use [adb](https://developer.android.com/studio/command-line/adb) to issue commands, as follows:

* Verify that your device is connected by running the adb devices command from your **android\_sdk**/platform-tools/ directory. If connected, you'll see the device listed.
* Issue any [adb command](https://developer.android.com/studio/command-line/adb#issuingcommands) with the -d flag to target your device.

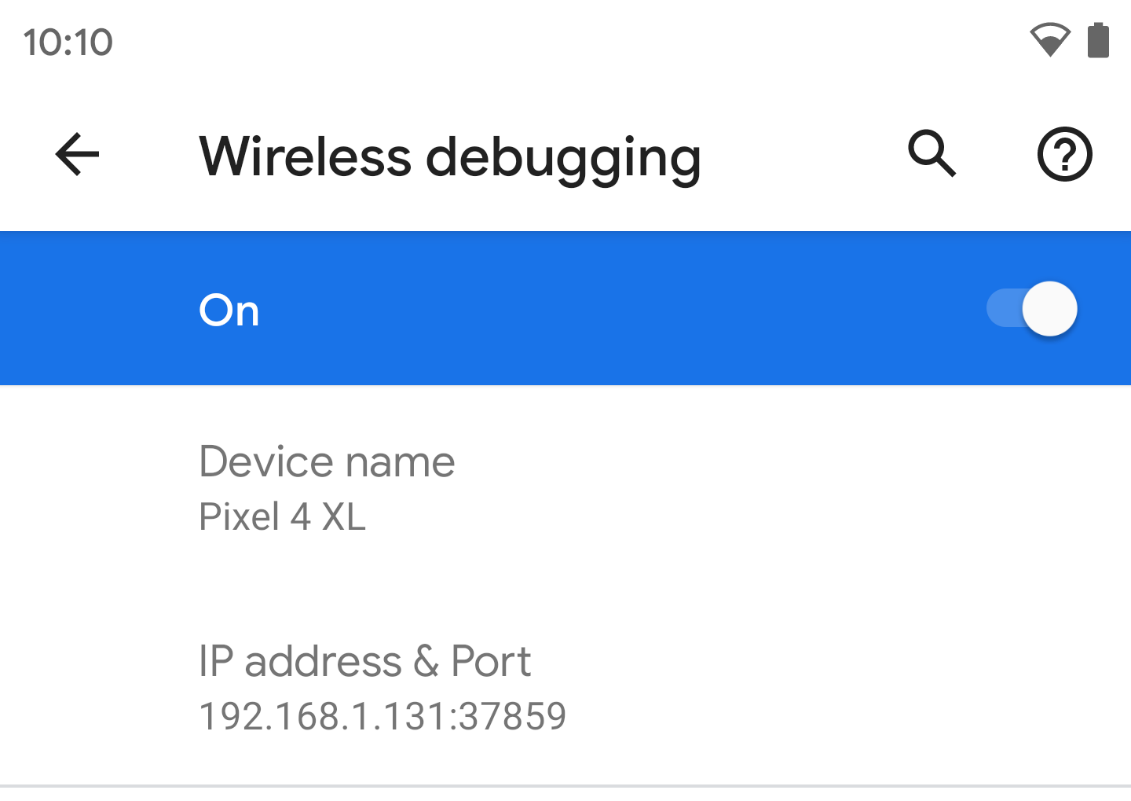
### Connect to your device using Wi-Fi

Android 11 (and later) supports deploying and debugging your app wirelessly from your workstation via Android Debug Bridge (adb). For example, you can deploy your debuggable app to multiple remote devices without physically connecting your device via USB and contending with common USB connection issues, such as driver installation.

To use wireless debugging, you need to pair your device to your workstation using a pairing code. Your workstation and device must be connected to the same wireless network. To connect to your device, follow these steps:



1. On your workstation, update to the latest version of the [SDK Platform-Tools](https://developer.android.com/studio/releases/platform-tools).
2. Enable the **Wireless debugging** option under **Developer options**.
3. On the dialog that asks **Allow wireless debugging on this network?**, click **Allow**.
4. Select **Pair device with pairing code**. Take note of the pairing code, IP address, and port number displayed on the device (see image).
5. On your workstation, open a terminal and navigate to **android\_sdk**/platform-tools.
6. Run adb pair **ipaddr**:**port**. Use the IP address and port number from step 3.
7. When prompted, enter the pairing code that you received in step 4. A message should indicate that your device has been successfully paired.
8. Enter pairing code: 482924
9. Successfully paired to 192.168.1.130:37099 [guid=adb-235XY]
10. Run adb connect **ipaddr**:**port**. Use the IP address and port under **Wireless debugging** (see image below).



**Lab 1 – Session 2**

**Total Time:**

**3 hours**

**Pre-Lab Activities:**

* No Pre-Lab Activity

**Learning Outcomes:**

* Perform the execution, debugging, testing, and profiling of mobile apps in modern IDEs.

**Lab Tasks:**

* + To install and configure the Flutter.
  + To explore the Android Studio functionalities.

**Student Activities:**

* + To install and configure the Flutter.

**Lab Solutions:**

# Lab Task 1: Install and configure Flutter

## System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

* **Operating Systems**: Windows 7 SP1 or later (64-bit), x86-64 based.
* **Disk Space**: 1.64 GB (does not include disk space for IDE/tools).
* **Tools**: Flutter depends on these tools being available in your environment.
  + [Windows PowerShell 5.0](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell) or newer (this is pre-installed with Windows 10)
  + [Git for Windows](https://git-scm.com/download/win) 2.x, with the **Use Git from the Windows Command Prompt** option.

If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

## Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

[flutter\_windows\_2.5.2-stable.zip](https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_2.5.2-stable.zip)

For other release channels, and older builds, see the [SDK releases](https://flutter.dev/docs/development/tools/sdk/releases) page.

1. Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\Users\<your-user-name>\Documents).

**Warning:** Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges.

If you don’t want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the [Flutter repo](https://github.com/flutter/flutter) on GitHub, and change branches or tags as needed. For example:

C:\src>git clone https://github.com/flutter/flutter.git -b stable

You are now ready to run Flutter commands in the Flutter Console.

### Update your path

If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the PATH environment variable:

* From the Start search bar, enter ‘env’ and select **Edit environment variables for your account**.
* Under **User variables** check if there is an entry called **Path**:
  + If the entry exists, append the full path to flutter\bin using ; as a separator from existing values.
  + If the entry doesn’t exist, create a new user variable named Path with the full path to flutter\bin as its value.

You have to close and reopen any existing console windows for these changes to take effect.

### Run flutter doctor

From a console window that has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:

C:\src\flutter>flutter doctor

This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).

For example:

[-] Android toolchain - develop for Android devices

• Android SDK at D:\Android\sdk

**✗ Android SDK is missing command line tools; download from https://goo.gl/XxQghQ**

• Try re-installing or updating your Android SDK,

visit https://flutter.dev/setup/#android-setup for detailed instructions.

## Create the app

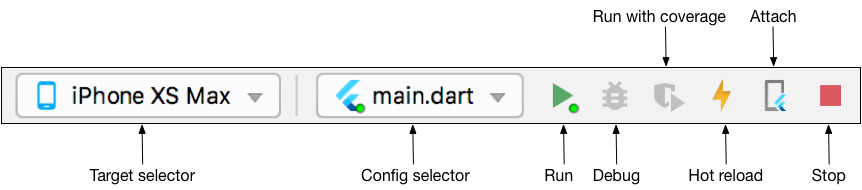
1. Open the IDE and select **Create New Flutter Project**.
2. Select **Flutter Application** as the project type. Then click **Next**.
3. Verify the Flutter SDK path specifies the SDK’s location (select **Install SDK…** if the text field is blank).
4. Enter a project name (for example, myapp). Then click **Next**.
5. Click **Finish**.
6. Wait for Android Studio to install the SDK and create the project.

**Note:** When creating a new Flutter app, some Flutter IDE plugins ask for a company domain name in reverse order, something like com.example. The company domain name and project name are used together as the package name for Android (the Bundle ID for iOS) when the app is released. If you think that the app might be released, it’s better to specify the package name now. The package name can’t be changed once the app is released, so make the name unique.

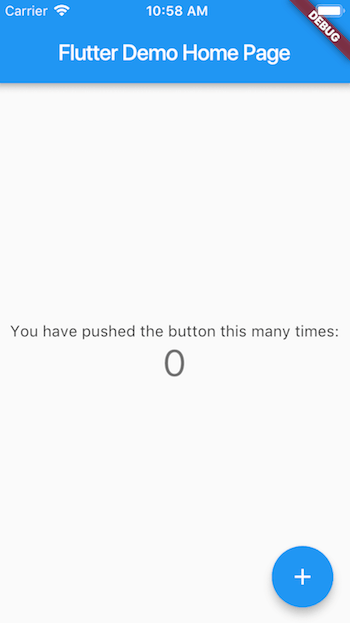
The above commands create a Flutter project directory called myapp that contains a simple demo app that uses [Material Components](https://material.io/guidelines).

**Tip:** The code for your app is in lib/main.dart. For a high-level description of what each code block does, see the comments at the top of that file.

## Run the app

1. Locate the main Android Studio toolbar:  
   
2. In the **target selector**, select an Android device for running the app. If none are listed as available, select **Tools > AVD Manager** and create one there. For details, see [Managing AVDs](https://developer.android.com/studio/run/managing-avds).
3. Click the run icon in the toolbar, or invoke the menu item **Run > Run**.

After the app build completes, you’ll see the starter app on your device.

Starter app

## Try hot reload

Flutter offers a fast development cycle with Stateful Hot Reload, the ability to reload the code of a live running app without restarting or losing app state. Make a change to app source, tell your IDE or command-line tool that you want to hot reload, and see the change in your simulator, emulator, or device.

1. Open lib/main.dart.
2. Change the string

'You have ~~pushed~~ the button this many times'

to

'You have clicked the button this many times'

**Important:** Do not stop your app. Let your app run.

1. Save your changes: invoke **Save All**, or click **Hot Reload**  .

You’ll see the updated string in the running app almost immediately.

## Profile or release runs

**Important:** Do not test the performance of your app with debug and hot reload enabled.

So far you’ve been running your app in debug mode. Debug mode trades performance for useful developer features such as hot reload and step debugging. It’s not unexpected to see slow performance and janky animations in debug mode. Once you are ready to analyze performance or release your app, you’ll want to use Flutter’s “profile” or “release” build modes. For more details, see [Flutter’s build modes](https://flutter.dev/docs/testing/build-modes).

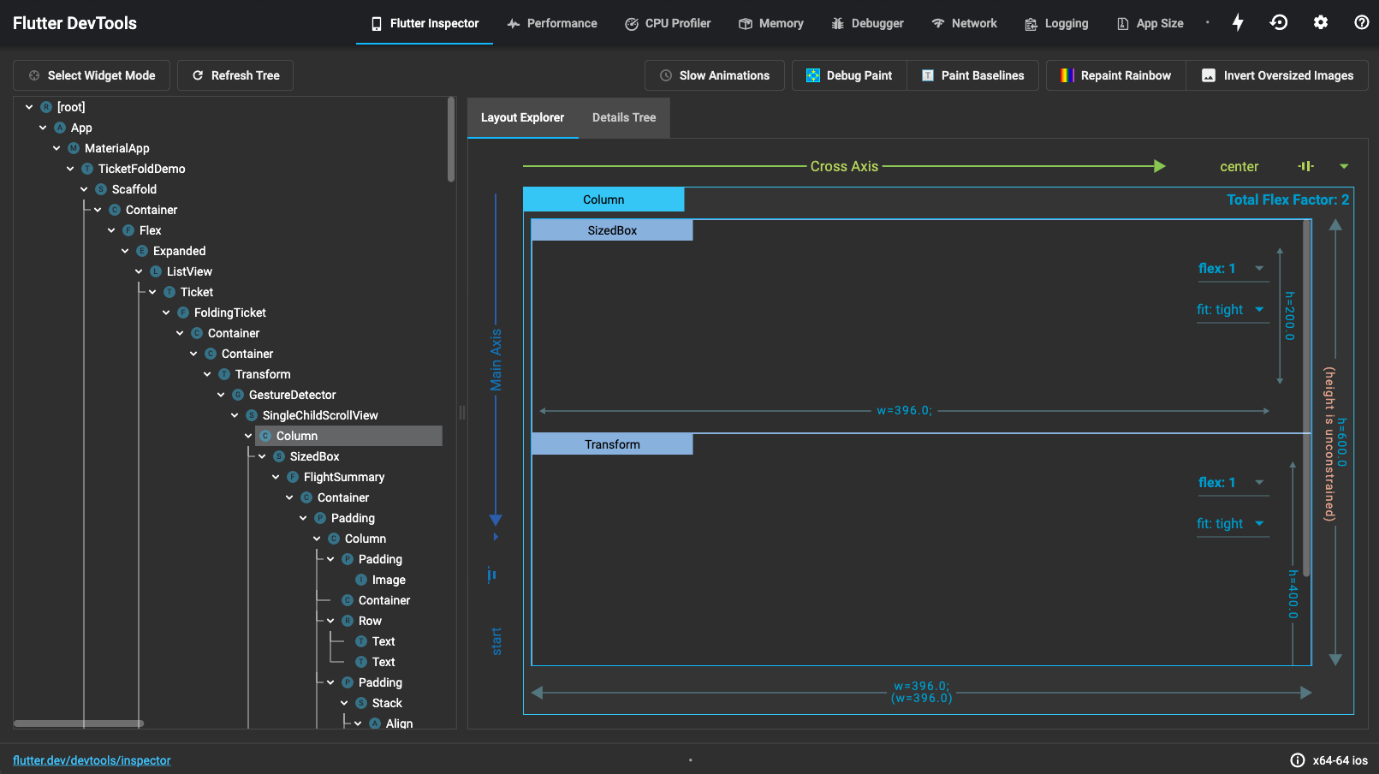
**Important:** If you’re concerned about the package size of your app, see [Measuring your app’s size](https://flutter.dev/docs/perf/app-size).

# 5 - Using the Flutter inspector

## What is it?

The Flutter widget inspector is a powerful tool for visualizing and exploring Flutter widget trees. The Flutter framework uses widgets as the [core building block](https://flutter.dev/docs/development/ui/widgets-intro) for anything from controls (such as text, buttons, and toggles), to layout (such as cantering, padding, rows, and columns). The inspector helps you visualize and explore Flutter widget trees, and can be used for the following:

* understanding existing layouts
* diagnosing layout issues



## Get started

To debug a layout issue, run the app in [debug mode](https://flutter.dev/docs/testing/build-modes#debug) and open the inspector by clicking the **Flutter Inspector** tab on the DevTools toolbar.

**Note:** You can still access the Flutter inspector directly from Android Studio/IntelliJ, but you might prefer the more spacious view when running it from DevTools in a browser.

### Debugging layout issues visually

The following is a guide to the features available in the inspector’s toolbar. When space is limited, the icon is used as the visual version of the label.

Select widget mode icon **Select widget mode**

Enable this button in order to select a widget on the device to inspect it. For more information, see [Inspecting a widget](https://flutter.dev/docs/development/tools/devtools/inspector#inspecting-a-widget).

Refresh tree icon **Refresh tree**

Reload the current widget info.

Slow animations icon [**Slow animations**](https://flutter.dev/docs/development/tools/devtools/inspector#slow-animations)

Run animations 5 times slower to help fine-tune them.

Show guidelines mode icon [**Show guidelines**](https://flutter.dev/docs/development/tools/devtools/inspector#show-guidelines)

Overlay guidelines to assist with fixing layout issues.

Show baselines icon [**Show baselines**](https://flutter.dev/docs/development/tools/devtools/inspector#show-baselines)

Show baselines, which are used for aligning text. Can be useful for checking if text is aligned.

Highlight repaints icon [**Highlight repaints**](https://flutter.dev/docs/development/tools/devtools/inspector#highlight-repaints)

Show borders that change color when elements repaint. Useful for finding unnecessary repaints.

Highlight oversized images icon [**Highlight oversized images**](https://flutter.dev/docs/development/tools/devtools/inspector#highlight-oversized-images)

Highlights images that are using too much memory by inverting colors and flipping them.

## Inspecting a widget

You can browse the interactive widget tree to view nearby widgets and see their field values.

To locate individual UI elements in the widget tree, click the **Select Widget Mode** button in the toolbar. This puts the app on the device into a “widget select” mode. Click any widget in the app’s UI; this selects the widget on the app’s screen, and scrolls the widget tree to the corresponding node. Toggle the **Select Widget Mode** button again to exit widget select mode.

When debugging layout issues, the key fields to look at are the size and constraints fields. The constraints flow down the tree, and the sizes flow back up. For more information on how this works, see [Understanding constraints](https://flutter.dev/docs/development/ui/layout/constraints).

**Lab 2 – Session 1**

**Total Time:**

**3 hours (sessions-1)**

**Pre-Lab Activities:**

* + Familiar with dartpad for dart Editor

**Learning Outcomes:**

* + Demonstrate the basic knowledge of Dart.

**Student Activities:**

* + Open Dartpad, and do some exercise for dart

**Lab Tasks 1:**

Create a program that asks the user to enter their name and their age. Print out a message that tells how many years they have to be 100 years old.

**Lab Tasks 2:**

Ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user.

**Lab Tasks 3:**

Take a list, say for example this one:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

and write a program that prints out all the elements of the list that are less than 5.

**Lab Tasks 4:**

Create a program that asks the user for a number and then prints out a list of all the divisors of that number.

If you don’t know what a divisor is, it is a number that divides evenly into another number. For example, 13 is a divisor of 26 because 26 / 13 has no remainder.

**Lab Tasks 5:**

Take two lists, for example:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

and write a program that returns a list that contains only the elements that are common between them (without duplicates). Make sure your program works on two lists of different sizes.

**Lab Tasks 6:**

Ask the user for a string and print out whether this string is a palindrome or not.

A palindrome is a string that reads the same forwards and backwards.

**Lab Tasks 7:**

Let’s say you are given a list saved in a variable:

a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100].

Write a Dart code that takes this list and makes a new list that has only the even elements of this list in it.

**Lab Solutions:**

**LAB 2.1 :**

|  |
| --- |
| import 'dart:io';  void main() {  stdout.write("What's your name? ");  String name = stdin.readLineSync();  print("Hi, $name! What is your age?");  int age = int.parse(stdin.readLineSync());  int yearsToHunderd = 100 - age;  print("$name, You have $yearsToHunderd years to be 100");  } |

**LAB 2.2 :**

|  |
| --- |
| import 'dart:io';  void main() {  stdout.write("Hi, please choose a number: ");  int number = int.parse(stdin.readLineSync());    if (number % 2 == 0) {  print("Chosen number is even");  }  else {  print("Chosen number is odd");  }  } |

**LAB 2.3 :**

|  |
| --- |
| void main() {  List<int> a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89];  for (var i in a) {  if (i < 5) {  print(i);  }  }    // One liner  print([for (var i in a) if (i < 5) i]);  } |

**LAB 2.4 :**

|  |
| --- |
| import 'dart:io';  void main() {  stdout.write("Please choose a number: ");  int number = int.parse(stdin.readLineSync());  for (var i = 1; i <= number; i++) {  if (number % i == 0) {  print(i);  }  }  } |

**LAB 2.5 :**

|  |
| --- |
| void main() {  List<int> a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89];  List<int> b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 89];  Set<int> c = {};  for (var i in a) {  for (var j in b) {  if (i == j) {  c.add(i);  }  }  }  print(c.toList());  // One liner using set intersections  print(Set.from(a).intersection(Set.from(b)).toList());  } |

**LAB 2.6 :**

|  |
| --- |
| import 'dart:io';  void main() {    stdout.write("Please give a word: ");  String input = stdin.readLineSync().toLowerCase();  String revInput = input.split('').reversed.join('');    // Ternary operator  input == revInput  ? print("The word is palindrome")  : print("The word is not a palindrome");  } |

**LAB 2.7 :**

|  |
| --- |
| void main() {  List<int> a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100];  int i = 0;  List<int> l = [];  for (var e in a) {  if (++i % 2 == 0) {  l.add(e);  }  }  print(l);    // One liner  print([for (var e in a) if (++i % 2 == 0) e]);  } |

**Lab 2 – Session 2**

**Total Time:**

**3 hours (sessions-2)**

**Pre-Lab Activities:**

* + Familiar with dartpad for dart Editor
  + Student must complete Lab-2 session-1 before attempting these labs.

**Learning Outcomes:**

* + Demonstrate the knowledge of Dart OOP.

**Student Activities:**

* + Open Dartpad, and do advance exercise for dart

**Lab Tasks 1:**

Make a two-player Rock-Paper-Scissors game against computer.

Ask for player’s input, compare them, print out a message to the winner.

**Lab Tasks 2:**

Generate a random number between 1 and 100. Ask the user to guess the number, then tell them whether they guessed too low, too high, or exactly right.

**Lab Tasks 3:**

Ask the user for a number and determine whether the number is prime or not.

Do it using a function

**Lab Tasks 4:**

Write a program (function) that takes a list and returns a new list that contains all the elements of the first list minus all the duplicates

**Lab Tasks 5:**

Write a program (using functions!) that asks the user for a long string containing multiple words. Print back to the user the same string, except with the words in backwards order.

For example, say I type the string:

My name is Michele

Then I would see the string:

Michele is name My

**Lab Solutions:**

**LAB 2.1 :**

|  |
| --- |
| import 'dart:io';  import 'dart:math';  void main() {  print("Welcome to Rock, Paper, Scissors\nType 'exit' to stop the game");  final random = Random();  // Rules of the game  Map<String, String> rules = {  "rock": "scissors",  "scissors": "paper",  "paper": "rock"  };  // Initial score  int user = 0;  int comp = 0;  // Options for computer to choose  List<String> options = ["rock", "paper", "scissors"];  // Actual game  while (true) {  String compChoice = options[random.nextInt(options.length)];  stdout.write("\nPlease choose Rock, Paper or Scissors: ");  String userChoice = stdin.readLineSync().toLowerCase();  if (userChoice == "exit") {  print("\nYou: $user Computer: $comp\nBye Bye!");  break;  }  if (!options.contains(userChoice)) {  print("Incorrect choice");  continue;  } else if (compChoice == userChoice) {  print("We have a tie!");  } else if (rules[compChoice] == userChoice) {  print("Computer wins: $compChoice vs $userChoice");  comp += 1;  } else if (rules[userChoice] == compChoice) {  print("You win: $userChoice vs $compChoice");  user += 1;  }  }  } |

**LAB 2.2 :**

|  |
| --- |
| import 'dart:io';  import 'dart:math';  void main() {  print("Type exit to quit the game");  guessingGame();  }  guessingGame() {  final random = Random();  int randNumber = random.nextInt(100);  int attempt = 0;  while (true) {  attempt += 1;  stdout.write("Please choose a number between 0 and 100: ");  String chosenNumber = stdin.readLineSync();  // Make sure user does not go out of limits  if (chosenNumber.toLowerCase() == "exit") {  print("\nBye");  break;  } else if (int.parse(chosenNumber) > 100) {  print("Please do not go over 100");  continue;  }  // Main logic  if (int.parse(chosenNumber) == randNumber) {  print("Bingo! You tried $attempt times\n");  continue;  } else if (int.parse(chosenNumber) > randNumber) {  print("You are higher");  continue;  } else {  print("You are lower");  continue;  }  }  } |

**LAB 2.3 :**

|  |
| --- |
| import 'dart:io';  void main() {  stdout.write("Please give us a number: ");  int chosenNumber = int.parse(stdin.readLineSync());  checkPrime(chosenNumber);  }  void checkPrime(int number) {  // List comprehensions  List<int> a = [  for (var i = 1; i <= number; i++)  if (number % i == 0) i  ];  // Check for prime  a.length == 2  ? print("The chosen number is a prime")  : print("The chosen number is not a prime");  } |

**LAB 2.4 :**

|  |
| --- |
| import 'dart:math';  void main() {  final random = Random();  List<int> randList = List.generate(10, (\_) => random.nextInt(10));  print("Initial list is $randList\n");  print("Cleaned list is ${removeDuplicates(randList)}");  }  List<int> removeDuplicates(List<int> initialList) {  return initialList.toSet().toList();  } |

**LAB 2.5 :**

|  |
| --- |
| import 'dart:io';  void main() {  stdout.write("Please give a sentence: ");  String sentence = stdin.readLineSync();  reverseSentence(sentence);  }  void reverseSentence(String sentence) {  /\* Split the sentence into a list of words  Reverse the list, then join the words back \*/  String a = sentence.split(" ").reversed.toList().join(" ");  print(a);  } |

**Lab 3 – Session 1**

**Total Time:**

**3 hours (session-1)**

**Pre-Lab Activities:**

* + Basics of Flutter
  + Familiar with Android Studio and Flutter Application Mode
  + Familiar with Layout Widgets

**Learning Outcomes:**

* Demonstrate the knowledge of Flutter to design stateful, cross-platform, and interactive user interfaces for mobile applications.

**Student Activities:**

* Student create a new flutter project in Android Studio and Write the code for above application.

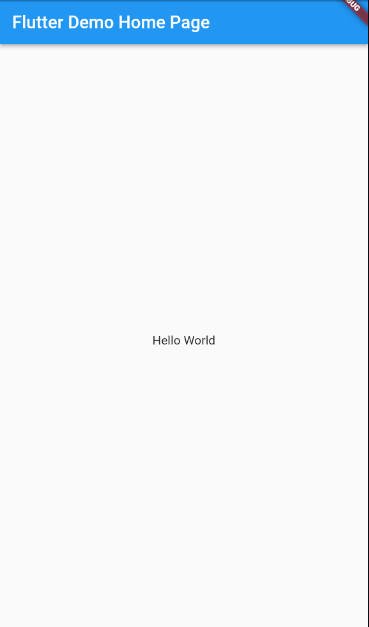
**Lab Tasks 1:**

* + Modify the default application, which you access, when you first time create a new project. You have only change the background color.

****

**Lab Tasks 2:**

* + Modify the code used to develop above task. Remove stateless and statefull classes and use the main function to develop the same app again with displaying a simple Text “Hellow Word”.



**Lab Solutions:**

**LAB 3.1 : Practicing Basic Flutter application design.,**

**CODE:**

|  |
| --- |
| **import 'package:flutter/material.dart';**  **void main() {**  **runApp(MyApp());**  **}**  **class MyApp extends StatelessWidget {**  **// This widget is the root of your application.**  **@override**  **Widget build(BuildContext context) {**  **return MaterialApp(**  **title: 'Flutter Demo',**  **theme: ThemeData(**  **primarySwatch: Colors.blue,**  **),**  **home: MyHomePage(title: 'Flutter Demo Home Page'),**  **);**  **}**  **}**  **class MyHomePage extends StatefulWidget {**  **MyHomePage({Key? key, required this.title}) : super(key: key);**  **final String title;**  **@override**  **\_MyHomePageState createState() => \_MyHomePageState();**  **}**  **class \_MyHomePageState extends State<MyHomePage> {**  **int \_counter = 0;**  **void \_incrementCounter() {**  **setState(() {**  **\_counter++;**  **});**  **}**  **@override**  **Widget build(BuildContext context) {**  **return Scaffold(**  **backgroundColor: Colors.lightGreen,**  **appBar: AppBar(**  **title: Text(widget.title),**  **),**  **body: Center(**  **child: Column(**  **mainAxisAlignment: MainAxisAlignment.center,**  **children: <Widget>[**  **Text(**  **'You have pushed the button $\_counter times:',**  **),**  **Text(**  **'$\_counter',**  **style: Theme.of(context).textTheme.headline4,**  **),**  **],**  **),**  **),**  **floatingActionButton: FloatingActionButton(**  **onPressed: \_incrementCounter,**  **tooltip: 'Increment',**  **child: Icon(Icons.add),**  **), // This trailing comma makes auto-formatting nicer for build methods.**  **);**  **}**  **}** |

**LAB 3.2 : Practicing Basic Flutter application design.,**

**CODE:**

|  |
| --- |
| **import 'package:flutter/material.dart';**  **void main() => runApp(MaterialApp(**  **title: 'Flutter Demo',**  **theme: ThemeData(**  **primarySwatch: Colors.blue,**  **),**  **home: Scaffold(**  **appBar: AppBar(**  **title: Text('Flutter Demo Home Page'),**  **),**  **body: Center(**  **child: Text("Hello World"),**  **),**  **),**  **));** |

**Lab 3 – Session 2**

**Total Time:**

**6 hours (2 sessions)**

**Pre-Lab Activities:**

* + Basics of Flutter
  + Familiar with Android Studio and Flutter Application Mode
  + Familiar with Basic Widgets

**Learning Outcomes:**

* Demonstrate the knowledge of Flutter to design stateful, cross-platform, and interactive user interfaces for mobile applications.
* Demonstrate the basic widgets desgin

**Student Activities:**

* Student create a new flutter project in Android Studio and Write the code for some basic Widgets.

**Lab Tasks 1:**

1. Create a contact-card app that shows your contact details as given in following figure. The components and requirements of this app are given below.
   1. Include safe area
   2. Grey background
   3. Contain a circular avatar with an image, and 50 radius.
   4. Your name with following style guidelines
      1. A screenshot of a cell phone

         Description automatically generated with medium confidenceFont ZenKurenaido (custom Google font)
      2. Bold
      3. White Color
      4. Font size 30
   5. Your Designation with following style guidelines
      1. Font Lato (custom Google font)
      2. Bold
      3. White Color
      4. Font size 20
      5. Letter spacing 3
   6. Phone icon and contact number with green color, white background, padding, and margin
   7. Email icon and email with green color, white background, padding, and margin

**Lab Solutions:**

**LAB 3.1 : Practicing Layout Widgets.**

**CODE:**

|  |
| --- |
| import 'dart:ui';  import 'package:flutter/cupertino.dart';  import 'package:flutter/material.dart';  void main() {  runApp(const MyApp());  }  class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  // This widget is the root of your application.  @override  Widget build(BuildContext context) {  return MaterialApp(  title: 'PortFolew',  home: Scaffold(  backgroundColor: Colors.teal,  body: SafeArea(  child: Column(  mainAxisAlignment: MainAxisAlignment.center,  children: [  CircleAvatar(  backgroundColor: Colors.amber,  radius: 70,  backgroundImage: AssetImage('images/myimage.jpg'),  ),  Text('Muhammad bin Abdul Ghaffar',style: TextStyle(  fontFamily: 'Schyler',  fontSize: 40,  fontWeight: FontWeight.bold,  color: Colors.white,  ),),  Text('Class Sixth', style: TextStyle(  color: Colors.amber,  fontSize: 20, letterSpacing: 2.5, fontWeight: FontWeight.bold),),  SizedBox(width: 150, height: 100,  child: Divider(color: Colors.white70),),  // Container with ListTile without rounded corner  Text('Container with List Tile without rounded corner'),  Container(  color: Colors.white,  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 20),  child: ListTile(  title: Text('03314741950', style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold, fontSize: 18,), ),  leading: Icon(Icons.phone, color: Colors.teal,),  ),  ),  // card with ListTile with round corner  Text('Card with List Tile with round corner'),  Card(  color: Colors.white,  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 20),  child: ListTile(  title: Text('abc@google.com', style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold, fontSize: 18),),  leading: Icon(Icons.email, color: Colors.teal,),  )),  // Container with Row  Text('Container with Row'),  Container(  padding: EdgeInsets.all(10),  color: Colors.white,  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 20),  child: Row(  mainAxisAlignment: MainAxisAlignment.center,  children: [  Icon(Icons.phone, color: Colors.teal,),  SizedBox(width: 30,),  Text('03314741950', style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold, fontSize: 18,), )  ],  ),  ),  // Card with Padding & Row  Text('Card with Padding & Row'),  Card(  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 20),  color: Colors.white,  elevation: 15.0,  child: Padding(  padding: EdgeInsets.all(10),  child: Row(  mainAxisAlignment: MainAxisAlignment.center,  children: [  Icon(Icons.email, color: Colors.teal,),  SizedBox(width: 30,),  Text('abc@google.com', style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold, fontSize: 18),),  ],  ),  ),  ),  ],  ),  ),  )  );  }  } |

**Lab 4 – Session 1**

**Total Time:**

**3 hours (session-1)**

**Pre-Lab Activities:**

* + Students should familier with layout widgets
  + Student should complete week three slides before these labs

**Learning Outcomes:**

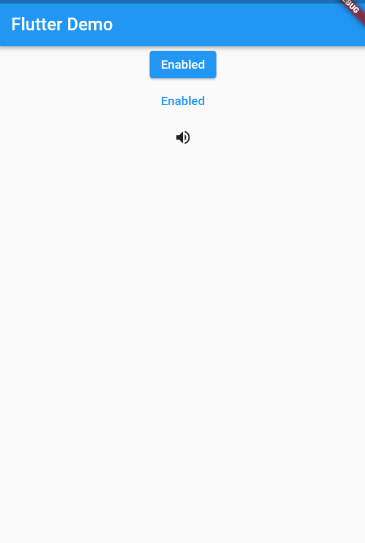
* + Student will learn about Layout Design
  + Student will learn single/multiple child widgets

**Student Activities:**

* + Student use the Android Studio
  + Student will get familiar with different Button styles
  + Student use different layout widget to desgin UI

**Lab Tasks 1:**

Design following button accoridng to formate given in figure

* + ****Elevated button
  + Text Button
  + Icon button

**Lab Tasks 2:**

* + Design below layout by using Expand and flex , assign flex value 2 to yellow container and 1 to blue container to achieve this design

Shape, square

Description automatically generated

**Lab Solutions:**

**LAB 4.1 : Practicing Button Designing.**

|  |
| --- |
| import 'package:flutter/material.dart';  void main() => runApp(MaterialApp(  title: 'Flutter Demo',  theme: ThemeData(  primarySwatch: Colors.blue,  ),  home: Scaffold(  appBar: AppBar(  title: Text('Flutter Demo'),  ),  body: Center(  child: Column(    children: <Widget> [  ElevatedButton(  onPressed: () {}, child: Text('Enabled'),  ),  TextButton(  onPressed: () { },  child: Text('Enabled', style: TextStyle(color: Colors.blue)),  ),  IconButton(  icon: Icon(Icons.volume\_up), onPressed: () { },  ),  ],  ),  ),  ),  )); |

**LAB 4.2 : Practicing Layout Design.**

|  |
| --- |
| import 'package:flutter/material.dart';  void main() => runApp(MaterialApp(  home: Scaffold(  appBar: AppBar(  title: Text('Flutter Demo'),  ),  body: Center(  child: Column(  crossAxisAlignment: CrossAxisAlignment.stretch,  mainAxisAlignment: MainAxisAlignment.center,  children: <Widget> [  Expanded(  child: Container(  color: Colors.blue,  ),  ),  Expanded(  child: Container(  color: Colors.orange,  ),  ),  Expanded(  child: Container(  color: Colors.blue,  ),  )  ],  ),  ),  ),  )); |

**Lab 4 – Session 2**

**Total Time:**

**3 hours**

**Pre-Lab Activities:**

* + Students should familier with layout widgets
  + Student should complete week three slides before this lab.

**Learning Outcomes:**

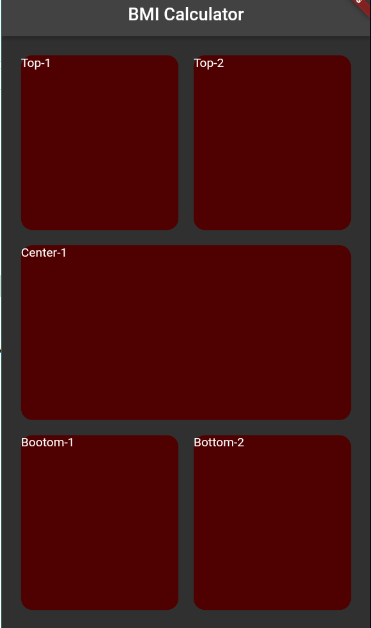
* + Student will learn about Layout Design
  + Student will learn single/multiple child widgets

**Student Activities:**

* + Student use the Android Studio for Flutter
  + Student use different layout widget to desgin UI

**Lab Tasks 3:**

* + Desing the User Interface according to given Layout



**Lab Solutions:**

**LAB 4.3 : Practicing Layout Designing.**

**CODE:**

|  |
| --- |
| body: Container(  margin: EdgeInsets.all(15),  child: Column(  children: [   Expanded(  child: Container(  child: Row(  children: [  Expanded(child: Container( margin: EdgeInsets.all(10),width: double.*infinity*, height: double.*infinity*, child: Text('Top-1'),  decoration: BoxDecoration(borderRadius: BorderRadius.circular(15), color: Color(0xFF4F0000)),),),  Expanded(child: Container( margin: EdgeInsets.all(10), width: double.*infinity*, height: double.*infinity*, child: Text('Top-2'),  decoration: BoxDecoration(borderRadius: BorderRadius.circular(15), color: Color(0xFF4F0000)))),  ],  ),  ),  ),  Expanded(  child: Container(  margin: EdgeInsets.all(10),   width: double.*infinity*,  height: double.*infinity*,  child: Text('Center-1'),  decoration: BoxDecoration(color: Color(0xFF4F0000), borderRadius: BorderRadius.circular(15)),),),   Expanded(  child: Container(  child: Row(  children: [  Expanded(child: Container(margin: EdgeInsets.all(10), width: double.*infinity*, height: double.*infinity*, child: Text('Bootom-1'),  decoration: BoxDecoration(color: Color(0xFF4F0000),borderRadius: BorderRadius.circular(15)),)),  Expanded(child: Container(margin: EdgeInsets.all(10), width: double.*infinity*, height: double.*infinity*, child: Text('Bottom-2'),  decoration: BoxDecoration(color: Color(0xFF4F0000), borderRadius: BorderRadius.circular(15)),)),  ],  ),  ),  ),  ],  ), |

**Lab 5 – Session 1**

**Total Time:**

**3 hours (session-1)**

**Pre-Lab Activities:**

* + Basics of Flutter Scoffold Widget
  + Familiar with Development Coding Editor

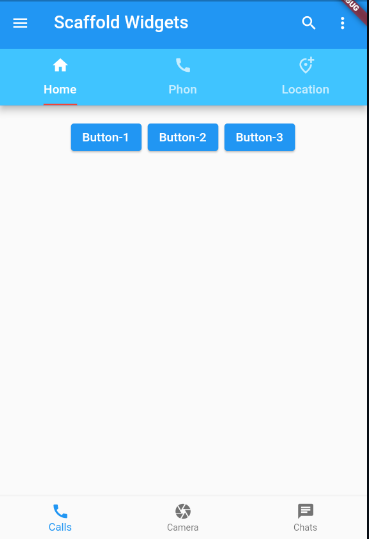
**Learning Outcomes:**

* + Learn Flutter Appbar
  + Learn Flutter BootomBar
  + Learn Flutter Navigation Bar
  + Learn Flutter MenuBar

**Student Activities:**

* + Complete the pre-lab activities
  + Practicing Basic Widgets
  + Practice for Scaffold

**Lab Tasks -1:**

* Design the UI accoriding to below given image.
* Develop Appbar with titile, leading and Action properties
* Develop Tabbar with BottomBar, consist on Call Camera and Chat Icons
* Develop ButtonBar having three buttons

**Lab Solutions:**

**LAB 5.1 :**

**CODE:**

|  |
| --- |
| import 'package:flutter/cupertino.dart';  import 'package:flutter/material.dart';  void main() => runApp(MaterialApp(  home: DefaultTabController(  length: 3,  child: Scaffold(  appBar: AppBar(  title: Text('Scaffold Widgets'),  elevation: 8,  //centerTitle: true,  leading: GestureDetector(  onTap: (){},  child: Icon(Icons.menu),  ),  actions: [  Padding(padding: EdgeInsets.only(right: 20),  child: GestureDetector(  onTap: (){},  child: Icon(Icons.search),  ),),  Padding(padding: EdgeInsets.only(right: 20),  child: GestureDetector(  onTap: (){ },  child: Icon(Icons.more\_vert),)  ),  ],  bottom: PreferredSize(  preferredSize: Size.fromHeight(80),  child: Container(  color: Colors.lightBlueAccent,  child: TabBar(  tabs: [  Tab( icon: Icon(Icons.home), text: 'Home',),  Tab(icon: Icon(Icons.call),text: 'Phon',),  Tab(icon: Icon(Icons.add\_location\_alt\_outlined),text: 'Location',),  ],  indicatorColor: Colors.red,  indicatorSize: TabBarIndicatorSize.label,  ), ), ),  ),  body: Container(  padding: EdgeInsets.all(10),  child: Column(  children: [  ButtonBar(  alignment: MainAxisAlignment.center,  mainAxisSize: MainAxisSize.max,  children: [  ElevatedButton(onPressed: (){}, child: Text('Button-1',)),  ElevatedButton(onPressed: (){}, child: Text('Button-2')),  ElevatedButton(onPressed: (){}, child: Text('Button-3')),  ],  )  ],  ),  ),    bottomNavigationBar: BottomNavigationBar(items: [  BottomNavigationBarItem(  icon: Icon(Icons.call),  label: 'Calls'),  BottomNavigationBarItem(  icon: Icon(Icons.camera),  label: 'Camera'),  BottomNavigationBarItem(  icon: Icon(Icons.chat),  label: 'Chats')  ],),  ),  ),)); |

**Lab 5 – Session 2**

**Total Time:**

**3 hours ( session-2)**

**Pre-Lab Activities:**

* + Basics of Flutter Scoffold appbar Widget

**Learning Outcomes:**

* + Learn layout design
  + Learn lisTtile for Design
  + Learn use of external resources.

**Student Activities:**

* + Complete the pre-lab activities
  + Practicing Basic Widgets
  + ****Practice external font resources

**Lab Tasks -2:**

* Design the UI accoriding to below given image.
* Display your picture in CircleAvatar
* Use external font (google font) for name
* Use the listTile for phone and email icon and text.
* Use this desing for appbar drawer.

**Lab Solutions:**

**LAB 5.2 :**

**CODE:**

|  |
| --- |
| import 'dart:ui';  import 'package:flutter/cupertino.dart';  import 'package:flutter/material.dart';  void main() {  runApp(const MyApp());  }  class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  // This widget is the root of your application.  @override  Widget build(BuildContext context) {  return MaterialApp(  title: 'Portfolew',  home: Scaffold(  backgroundColor: Colors.teal,  body: SafeArea(  child: Column(  mainAxisAlignment: MainAxisAlignment.center,  children: [  CircleAvatar(  backgroundColor: Colors.amber,  radius: 70,  backgroundImage: AssetImage(''),  ),  Text('Muhammad bin Abdul Ghaffar',style: TextStyle(  fontFamily: 'Schyler',  fontSize: 40,  fontWeight: FontWeight.bold,  color: Colors.white,  ),),  Text('Class Sixth', style: TextStyle(  color: Colors.amber,  fontSize: 20, letterSpacing: 2.5, fontWeight: FontWeight.bold),),  SizedBox(width: 150, height: 100,  child: Divider(color: Colors.white70),),  // Container with ListTile without rounded corner  Container(  color: Colors.white,  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 20),  child: ListTile(  title: Text('03314741950', style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold, fontSize: 18,), ),  leading: Icon(Icons.phone, color: Colors.teal,),  ),  ),  // card with ListTile with round corner  Card(  color: Colors.white,  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 20),  child: ListTile(  title: Text('abc@google.com', style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold, fontSize: 18),),  leading: Icon(Icons.email, color: Colors.teal,),  )),  ),  ],  ),  ),  )  );  }  } |

**Lab 6 – Session 1**

**Total Time:**

**3 hours (session-1)**

**Pre-Lab Activities:**

* + Student Must have Knowledge about External Resources calling.

**Learning Outcomes:**

* + Students will learn about external libraries
  + Students will learn more about layout widgets

**Student Activities:**

* + Design different color bars
  + Each bar create a sound

**Lab Tasks 1:**

**Create Xylophone app Audio playing application**

* + Create asset folder with all audio files
  + Create 7 Flat Buttons and upon click on each button Audio must be played , GUI of application is given bellow
  + Import audio player package in your dart file and use audio Cache methods to play files in assets.
  + All sounds are represnet as note1.wav, note2.wav, note3,wav, note4.wav, note5.wav, note6.wav, note7.wav.

**Lab Solutions:**

**LAB 6.1 :**

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:audioplayers/audioplayers.dart';  void main() => runApp(MaterialApp(  home: xylophone(),  ));  class xylophone extends StatelessWidget {  const xylophone({Key? key}) : super(key: key);  final player = AudioCache();    @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.black,  appBar: AppBar(title: Text('Xylophone'),  ),  body: SafeArea(  child: Container(  child: Column(  crossAxisAlignment: CrossAxisAlignment.stretch,  children: [  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note1.wav');  },  ),  )  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note2.wav’);  },  ),  )  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note3.wav’);  },  ),  )  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note4.wav’);  },  ),  )  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note5.wav’);  },  ),  )  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note6.wav’);  },  ),  )  Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { player.play('note7.wav’);  },  ),  )    ],  ),),  ),  );  }  } |

**Lab 6 – Session 2**

**Total Time:**

**3 hours (sessions-1)**

**Pre-Lab Activities:**

* + Student Must have Knowledge about Widget dart function.

**Learning Outcomes:**

* + Students will learn about external libraries
  + Students will learn the use of functions in Flutter
  + Students will learn more about layout widgets

**Lab Task- 2:**

In Lab Task 1, Expanded widgets are repeated seven times. Enclose Expended widgets in a function and call the this function seven time instead 7 Expanded block.

Create another function for sound and pass sound value to this function from onPressed callback function.

**Lab Solutions:**

**LAB 6.2 :**

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:audioplayers/audioplayers.dart';  void main() => runApp(MaterialApp(  home: xylophone(),  ));  class xylophone extends StatelessWidget {  const xylophone({Key? key}) : super(key: key);  void soundPlayer(int note){ final player = AudioCache();  player.play('note$note.wav');}  Expanded xylophonefun({required Color color, required int soundPlay}){  return Expanded(  child: TextButton(  style: ButtonStyle(backgroundColor: MaterialStateProperty.all(color)),  onPressed: () { soundPlayer(soundPlay);  },  child: Text("Press Me"),  ),  );  }  @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.black,  appBar: AppBar(title: Text('Xylophone'),  ),  body: SafeArea(  child: Container(  child: Column(  crossAxisAlignment: CrossAxisAlignment.stretch,  children: [  xylophonefun(color: Colors.orange, soundPlay: 1),  xylophonefun(color: Colors.yellow, soundPlay: 2),  xylophonefun(color: Colors.pink, soundPlay: 3),  xylophonefun(color: Colors.purple, soundPlay: 4),  xylophonefun(color: Colors.indigo, soundPlay: 5),  xylophonefun(color: Colors.red, soundPlay: 6),  xylophonefun(color: Colors.brown, soundPlay: 7),  ],  ),),  ),  );  }  } |

**Lab 7 – Session 1**

**Total Time:**

**3 hours (sessions-1)**

**Pre-Lab Activities:**

* + Student must complete previous labs

**Learning Outcomes:**

* Demonstrate the knowledge of Flutter to design stateful, cross-platform, and interactive user interfaces for mobile applications.
* In this lab we will be reviewing Stateful and Stateless Widgets as well as learning about the fundamental building blocks of Object-Oriented Programming (OOP) - Classes and Objects.

**Student Activities:**

* + Practicing Stateless and Statfull Widgets

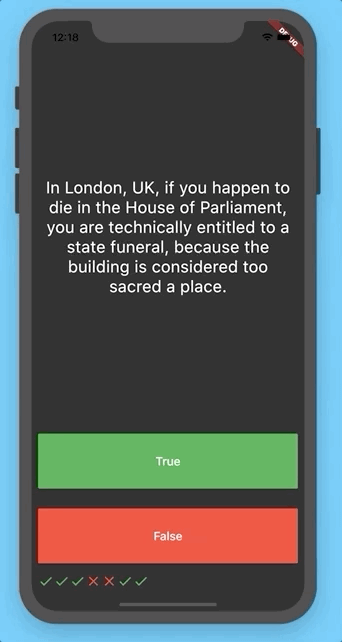
**Lab Tasks:**

* + Create a Flutter UI accoridng to below figure

We’re going to make a quiz app that tests your general knowledge. As one of the most popular types of apps on the app stores, you can create your own quiz for other people to enjoy!

Therefore, you are required to develop the Quiz app (as given in Figure 1) by following the steps.

1. Create a class “Question” that has two data members (i.e., question and answer).
2. Create a class “QuestionBank” that has 10 objects of the “Question” class which adds appropriate questions and answers. Further, this class must have an identifier that locates the current question being displayed on the screen. Furthermore, create a method “nextQuestion” to display the next question on the screen.
3. Display the question, and score on the screen as given the Figure 1. Further, appropriately update the scorekeeper. However, your app must not show more than 10 (maximum number of questions in QuestionBank.



**Lab Solutions:**

**CODE:**

|  |
| --- |
| import 'package:flutter/material.dart';  void main() => runApp(Quizzler());  class Quizzler extends StatelessWidget {  @override  Widget build(BuildContext context) {  return MaterialApp(  home: Scaffold(  backgroundColor: Colors.grey.shade900,  body: SafeArea(  child: Padding(  padding: EdgeInsets.symmetric(horizontal: 10.0),  child: QuizPage(),  ),  ),  ),  );  }  }  class QuizPage extends StatefulWidget {  @override  \_QuizPageState createState() => \_QuizPageState();  }  class \_QuizPageState extends State<QuizPage> {  @override  Widget build(BuildContext context) {  return Column(  mainAxisAlignment: MainAxisAlignment.spaceBetween,  crossAxisAlignment: CrossAxisAlignment.stretch,  children: <Widget>[  Expanded(  flex: 5,  child: Padding(  padding: EdgeInsets.all(10.0),  child: Center(  child: Text(  'This is where the question text will go.',  textAlign: TextAlign.center,  style: TextStyle(  fontSize: 25.0,  color: Colors.white,  ),  ),  ),  ),  ),  Expanded(  child: Padding(  padding: EdgeInsets.all(15.0),  child: FlatButton(  textColor: Colors.white,  color: Colors.green,  child: Text(  'True',  style: TextStyle(  color: Colors.white,  fontSize: 20.0,  ),  ),  onPressed: () {  //The user picked true.  },  ),  ),  ),  Expanded(  child: Padding(  padding: EdgeInsets.all(15.0),  child: FlatButton(  color: Colors.red,  child: Text(  'False',  style: TextStyle(  fontSize: 20.0,  color: Colors.white,  ),  ),  onPressed: () {  //The user picked false.  },  ),  ),  ),  //TODO: Add a Row here as your score keeper  ],  );  }  }  /\*  question1: 'You can lead a cow down stairs but not up stairs.', false,  question2: 'Approximately one quarter of human bones are in the feet.', true,  question3: 'A slug\'s blood is green.', true,\*/ |

**Lab 7 – Session 2**

**Total Time:**

**3 hours (session-2)**

**Pre-Lab Activities:**

* + Student must complete animation lesson

**Learning Outcomes:**

Demonstrate the knowledge of Flutter animation.

**Student Activities:**

* + Practicing Stateless and Statfull Widgets

**Lab Tasks:**

* + Create a Flutter animation application, which scall up and down the Flutter Logo



**Lab Solutions:**

**CODE:**

|  |
| --- |
| import 'package:flutter/animation.dart';  import 'package:flutter/material.dart';  void main() => runApp(MyApp());    class MyApp extends StatelessWidget {  // This widget is the root of your application.  @override  Widget build(BuildContext context) {  return MaterialApp(  title: 'Flutter Animation',  theme: ThemeData(  // This is the theme of your application.  primarySwatch: Colors.blue,  ),  home: MyHomePage(),  );  }  }  class MyHomePage extends StatefulWidget {  \_HomePageState createState() => \_HomePageState();  }  class \_HomePageState extends State<MyHomePage> with SingleTickerProviderStateMixin {  Animation<double> animation;  AnimationController animationController;  @override  void initState() {  super.initState();  animationController = AnimationController(vsync: this, duration: Duration(milliseconds: 2500));  animation = Tween<double>(begin: 0.0, end: 1.0).animate(animationController);  animation.addListener((){  setState((){  print (animation.value.toString());  });  });  animation.addStatusListener((status){  if(status == AnimationStatus.completed){  animationController.reverse();  } else if(status == AnimationStatus.dismissed) {  animationController.forward();  }  });  animationController.forward();  }  @override  Widget build(BuildContext context) {  return Center(  child: AnimatedLogo(  animation: animation,  )  );  }  }  class AnimatedLogo extends AnimatedWidget {  final Tween<double> \_sizeAnimation = Tween<double> (begin: 0.0, end: 500.0);  AnimatedLogo({Key key, Animation animation}):super(key: key, listenable: animation);  @override  Widget build(BuildContext context) {  final Animation<double> animation = listenable;  return Transform.scale(  scale: \_sizeAnimation.evaluate(animation),  child: FlutterLogo(),  );  }  } |

**Lab 8 – Session 1**

**Total Time:**

**3 hours**

**Pre-Lab Activities:**

* + Familiar with Flutter Database

**Learning Outcomes:**

* + Student will learn how data is saved and retrieved from database.

**Student Activities:**

* + create firebase project
  + configure it in Android project

**Lab Tasks 1:**

* + Create a main page and define routes for register.dart, welcome.dart, chat.dart, login.dart
  + Create Welcome screen as below as your first screen

Graphical user interface, application, website

Description automatically generated

**Lab Tasks 2:**

* + Graphical user interface, application

    Description automatically generatedCreate a Register page shown below, on click of registration button it must register user in firebase database.

**Lab Tasks 3:**

* + Create sign in page as shon in figure below , on click of login button it must authenticate user if user exist in database it redirect toward chat,dart screen

Graphical user interface, text, application, chat or text message

Description automatically generated

**Lab 8 – Session 2**

**Total Time:**

**3 hours**

**Pre-Lab Activities:**

* + Familiar with Flutter Database

**Learning Outcomes:**

* + Student will learn how data is saved and retrieved from database.

**Student Activities:**

* + Register user with firebase service
  + sigin user with firebase auth service

# Lab Task 1: Implementing Firebase Authentication in a Flutter app

Whether you’re working a simple to-do application or building your dream app, authentication helps you personalize the experience with user-specific information. It’s also a crucial component for privacy and security.

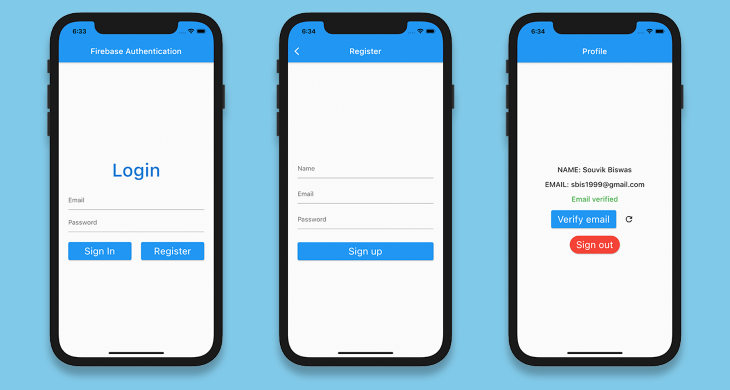
[Firebase Authentication](https://firebase.google.com/docs/auth) is a preconfigured backend service that makes it really easy to integrate with a mobile app using an SDK. You don’t have to maintain any backend infrastructure for the authentication process and Firebase supports integration with popular identity providers such as Google, Facebook, and GitHub.

In this tutorial, we’ll show you how to integrate Firebase Authentication with your [Flutter app](https://blog.logrocket.com/pros-and-cons-of-flutter-app-development/). To demonstrate with a practical example, we’ll walk you through the process of building an email-password registration and login process.

We’ll walk through the following steps:

* [Create a Flutter and Firebase project](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#create-a-flutter-and-firebase-project)
* [Set up Firebase for Android, iOS, and web](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#set-up-firebase-for-android-ios-and-web)
* [Import Firebase plugins](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#import-firebase-plugins)
* [Initialize Firebase App](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#initialize-firebase-app)
* [Register a new user](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#register-a-new-user)
* [User sign-in and sign-out](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#user-signin-and-signout)
* [Refresh user](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#refresh-user)
* [Define validators](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#define-validators)
* [Build the sign-in form](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#build-the-signin-form)
* [Build the profile page](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#build-the-profile-page)
* [Persist the login state](https://blog.logrocket.com/implementing-firebase-authentication-in-a-flutter-app/#persist-the-login-state)

The finished app will look like this:



## Create a Flutter and Firebase project

Create a new Flutter project using the following command:

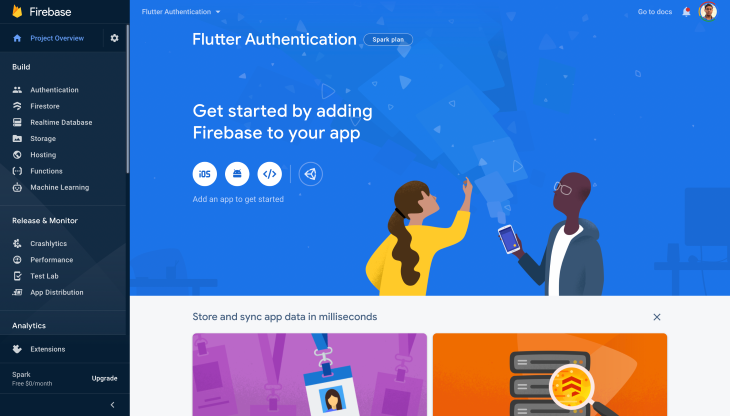
flutter create flutter\_authentication

Open the project in your favorite code editor. Here’s how to open it using VS Code:

code flutter\_authentication

To integrate Firebase with your Flutter project, you have to create a new Firebase project by going to the [console](https://console.firebase.google.com/).

Add a new project and give it a name. We don’t need Google Analytics for this sample project, so you can disable it. Once you’ve created your project, you’ll be directed to your Firebase project dashboard.



## Set up Firebase for Android, iOS, and web

To use Firebase with Android, iOS or web, you have to complete some configurations for each platform. See the complete configuration guides below:

* [Android installation](https://firebase.flutter.dev/docs/installation/android)
* [iOS installation](https://firebase.flutter.dev/docs/installation/ios)
* [Web installation](https://firebase.flutter.dev/docs/installation/web)

Now that we have the basic setup for using Firebase, let’s dive into our Flutter app.

## Import Firebase plugins

Before you start to implement the authentication logic, you need to import the following plugins:

* [firebase\_core](https://pub.dev/packages/firebase_core), which is required to use any Firebase service in a Flutter app
* [firebase\_auth](https://pub.dev/packages/firebase_auth) to get access to Firebase Authentication services

Add these plugins to your pubspec.yaml file:

dependencies:

firebase\_core: ^1.0.4

firebase\_auth: ^1.1.1

## Initialize Firebase App

Before using any Firebase service within the Flutter app, you need to initialize [Firebase App](https://firebase.google.com/docs/reference/js/firebase.app.App).

Modify the main.dart file to the following:

import 'package:flutter/material.dart';

import 'screens/login\_page.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Authentication',

debugShowCheckedModeBanner: false,

theme: ThemeData(

primarySwatch: Colors.cyan,

),

home: LoginPage(),

);

}

}

Define the LoginPage:

import 'package:flutter/material.dart';

class LoginPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Firebase Authentication'),

),

);

}

}

Add a new method to initialize Firebase App:

Future<FirebaseApp> \_initializeFirebase() async {

FirebaseApp firebaseApp = await Firebase.initializeApp();

return firebaseApp;

}

Because this method is asynchronous, you have to use FutureBuilder inside the build method:

class LoginPage extends StatelessWidget {

Future<FirebaseApp> \_initializeFirebase() async {

FirebaseApp firebaseApp = await Firebase.initializeApp();

return firebaseApp;

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Firebase Authentication'),

),

body: FutureBuilder(

future: \_initializeFirebase(),

builder: (context, snapshot) {

if (snapshot.connectionState == ConnectionState.done) {

return Column(

children: [

Text('Login'),

],

);

}

return Center(

child: CircularProgressIndicator(),

);

},

),

);

}

}

For now, we’re just showing a simple Text widget as the async task completes, and otherwise just a CircularProgressIndicator.

## Register a new user

When a new user arrives, before logging in, they have to register to the Firebase authentication.

Create a new dart file called fire\_auth.dart and define a new method called registerUsingEmailPassword():

class FireAuth {

static Future<User?> registerUsingEmailPassword({

required String name,

required String email,

required String password,

}) async {

FirebaseAuth auth = FirebaseAuth.instance;

User? user;

try {

UserCredential userCredential = await auth.createUserWithEmailAndPassword(

email: email,

password: password,

);

user = userCredential.user;

await user!.updateProfile(displayName: name);

await user.reload();

user = auth.currentUser;

} on FirebaseAuthException catch (e) {

if (e.code == 'weak-password') {

print('The password provided is too weak.');

} else if (e.code == 'email-already-in-use') {

print('The account already exists for that email.');

}

} catch (e) {

print(e);

}

return user;

}

}

Here we are registering a new user using the email and password provided and associating the name of the user with this profile.

There can be various FirebaseAuthException errors, which we have handled in the above code snippet.

## User sign-in and sign-out

To sign in a user who has already registered in our app, define a new method called signInUsingEmailPassword(), passing the user email and password:

static Future<User?> signInUsingEmailPassword({

required String email,

required String password,

required BuildContext context,

}) async {

FirebaseAuth auth = FirebaseAuth.instance;

User? user;

try {

UserCredential userCredential = await auth.signInWithEmailAndPassword(

email: email,

password: password,

);

user = userCredential.user;

} on FirebaseAuthException catch (e) {

if (e.code == 'user-not-found') {

print('No user found for that email.');

} else if (e.code == 'wrong-password') {

print('Wrong password provided.');

}

}

return user;

}

The email and password are used to generate the User object provided by Firebase. The User can be used later to retrieve any additional data (e.g., user name, profile picture, etc.) stored in that account.

You can use the signOut() method to log a user out. There is no need to create a separate method for signing out because it’s just a single line of code:

FirebaseAuth.instance.signOut();

## Send email verification

Let’s say you want to verify whether a user has entered the correct email address before proceeding. To send an email verification, you can use the sendEmailVerification() method on the User object:

user.sendEmailVerification();

## Refresh user

We’ll define one more method inside the FireAuth class for refreshing the User:

static Future<User?> refreshUser(User user) async {

FirebaseAuth auth = FirebaseAuth.instance;

await user.reload();

User? refreshedUser = auth.currentUser;

return refreshedUser;

}

## Define validators

Our app will have three form fields for name, email, password. We’ll create a validator for each one. The validators will help to check whether the user has entered any inappropriate value in a specific field and show an error accordingly.

Create a new file called validator.dart, define a class Validator, and specify three methods inside it (each of which will take a String as a parameter):

* validateName() to check whether the name field is empty
* validateEmail() to check whether the email address field is empty and validate that it’s in the correct format using a regular expression
* validatePassword() to check whether the password field is empty and verify that the length is longer than six characters

class Validator {

static String? validateName({required String name}) {

if (name == null) {

return null;

}

if (name.isEmpty) {

return 'Name can\'t be empty';

}

return null;

}

static String? validateEmail({required String email}) {

if (email == null) {

return null;

}

RegExp emailRegExp = RegExp(

r"^[a-zA-Z0-9.!#$%&'\*+/=?^\_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?)\*$");

if (email.isEmpty) {

return 'Email can\'t be empty';

} else if (!emailRegExp.hasMatch(email)) {

return 'Enter a correct email';

}

return null;

}

static String? validatePassword({required String password}) {

if (password == null) {

return null;

}

if (password.isEmpty) {

return 'Password can\'t be empty';

} else if (password.length < 6) {

return 'Enter a password with length at least 6';

}

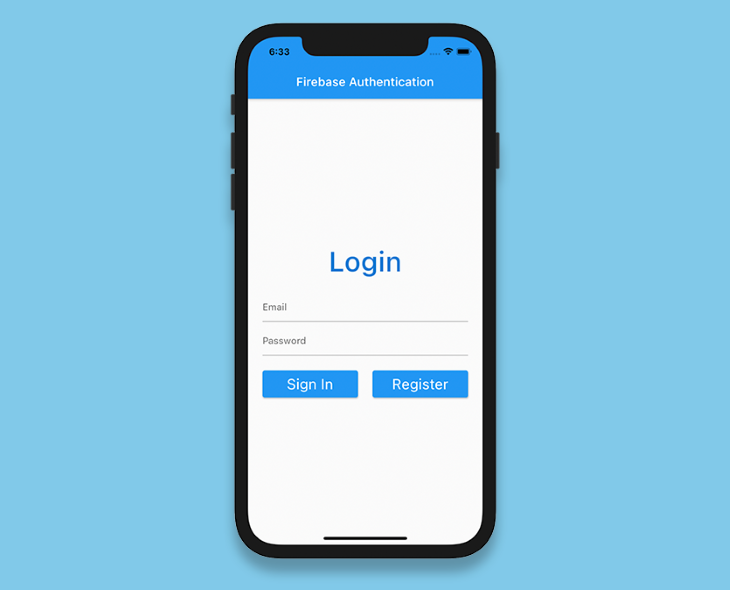
return null;

}

}

## Build the sign-in form

Let’s add a form to the LoginPage for accepting the user’s email address and password:



Define a GlobalKey:

final \_formKey = GlobalKey<FormState>();

Add a form and specify the key:

Form(

key: \_formKey,

child: Column(

children: <Widget>[

// Add widgets

],

),

)

Next, add two TextFormFields to accept the email and password:

Form(

key: \_formKey,

child: Column(

children: <Widget>[

TextFormField(

controller: \_emailTextController,

focusNode: \_focusEmail,

validator: (value) => Validator.validateEmail(email: value),

),

SizedBox(height: 8.0),

TextFormField(

controller: \_passwordTextController,

focusNode: \_focusPassword,

obscureText: true,

validator: (value) => Validator.validatePassword(password: value),

),

],

),

)

Add two buttons inside the Form: one to sign in and the other for navigating to the RegisterPage:

Row(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: [

Expanded(

child: ElevatedButton(

onPressed: () async {

if (\_formKey.currentState!.validate()) {

User? user = await FireAuth.signInUsingEmailPassword(

email: \_emailTextController.text,

password: \_passwordTextController.text,

);

if (user != null) {

Navigator.of(context)

.pushReplacement(

MaterialPageRoute(builder: (context) => ProfilePage(user: user)),

);

}

}

},

child: Text(

'Sign In',

style: TextStyle(color: Colors.white),

),

),

),

Expanded(

child: ElevatedButton(

onPressed: () {

Navigator.of(context).push(

MaterialPageRoute(builder: (context) => RegisterPage()),

);

},

child: Text(

'Register',

style: TextStyle(color: Colors.white),

),

),

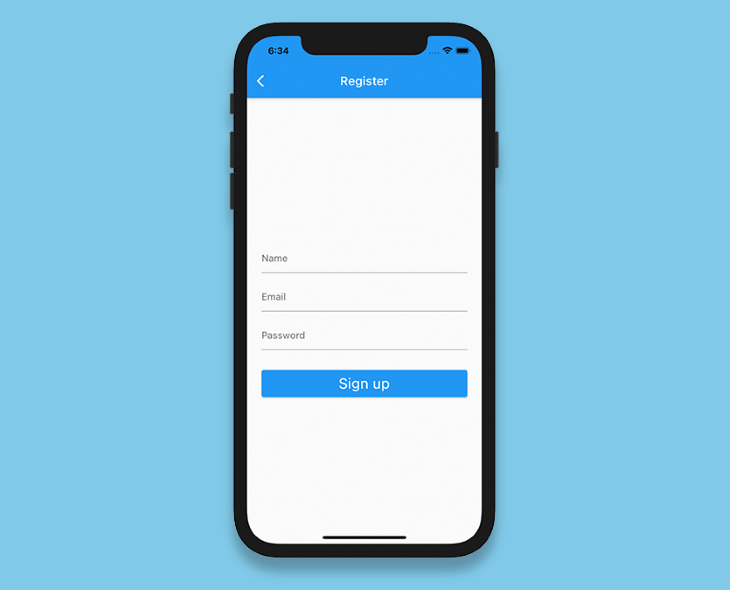
),

],

)

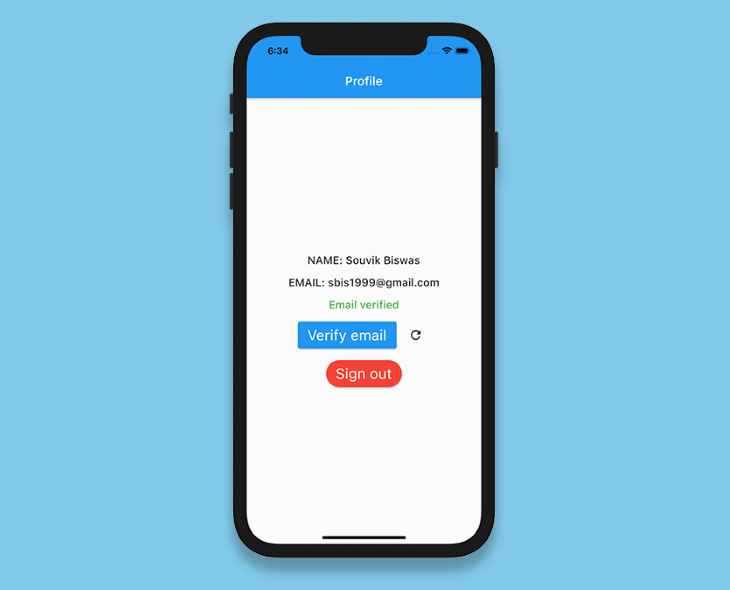
Inside the **Sign In** button we have called FireAuth.signInUsingEmailPassword() for performing the sign in process using Firebase Authentication.

The RegisterPage will also contain a Form similar to this, just one more field would be there for accepting the name of the user as they sign up for the first time.



## Build the profile page

On the ProfilePage, we’ll pass the User object and show the following details: name, email and whether the user has completed the email validation.



This page will also contain two buttons: one for sending email verification and the other for signing out the user.

class ProfilePage extends StatefulWidget {

final User user;

const ProfilePage({required this.user});

@override

\_ProfilePageState createState() => \_ProfilePageState();

}

class \_ProfilePageState extends State<ProfilePage> {

bool \_isSendingVerification = false;

bool \_isSigningOut = false;

late User \_currentUser;

@override

void initState() {

\_currentUser = widget.user;

super.initState();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Profile'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

'NAME: ${\_currentUser.displayName}',

style: Theme.of(context).textTheme.bodyText1,

),

SizedBox(height: 16.0),

Text(

'EMAIL: ${\_currentUser.email}',

style: Theme.of(context).textTheme.bodyText1,

),

SizedBox(height: 16.0),

\_currentUser.emailVerified

? Text(

'Email verified',

style: Theme.of(context)

.textTheme

.bodyText1!

.copyWith(color: Colors.green),

)

: Text(

'Email not verified',

style: Theme.of(context)

.textTheme

.bodyText1!

.copyWith(color: Colors.red),

),

// Add widgets for verifying email

// and, signing out the user

],

),

),

);

}

}

The button for sending email verification is as follows:

ElevatedButton(

onPressed: () async {

await \_currentUser.sendEmailVerification();

},

child: Text('Verify email'),

)

We’ll also add an IconButton, which can be used to refresh the user when the email is verified.

IconButton(

icon: Icon(Icons.refresh),

onPressed: () async {

User? user = await FireAuth.refreshUser(\_currentUser);

if (user != null) {

setState(() {

\_currentUser = user;

});

}

},

)

Lastly, the button for signing out the user:

ElevatedButton(

onPressed: () async {

await FirebaseAuth.instance.signOut();

Navigator.of(context).pushReplacement(

MaterialPageRoute(

builder: (context) => LoginPage(),

),

);

},

child: Text('Sign out')

)

## Persist the login state

There is one more important thing left to do. In most apps, you only have to log in once and it remembers the status on subsequent visits — that is, it automatically signs you into the app so that you don’t have to provide your credentials every time.

Inside the \_LoginPageState class, modify the \_initializeFirebase() method to retrieve the current user. If the User is not null, that means the user is already logged into the app, so just navigate to the UserInfoScreen with the retrieved user.

Future<FirebaseApp> \_initializeFirebase() async {

FirebaseApp firebaseApp = await Firebase.initializeApp();

User? user = FirebaseAuth.instance.currentUser;

if (user != null) {

Navigator.of(context).pushReplacement(

MaterialPageRoute(

builder: (context) => ProfilePage(

user: user,

),

),

);

}

return firebaseApp;

}