



Vidyavardhini's

College of Engineering & Technology

Vasai Road (W)

Department of Information Technology

Lab Manual

Semester	VI	Class	TE
Course Code	ITL604	Academic Year	2022-23
Course Name	Mobile App Development & Progressive Web App Lab		
Name of Faculty	Mr. Sainath Patil		
Supporting Staff	Mr. Nitin Shinghane		

Experiment 1

Aim: To install and configure flutter environment.

Theory:

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

- Operating Systems: Windows 7 SPI 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 or newer (this is pre-installed with Windows 10)
 - Git for Windows 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_3.7.11-stable.zip

For other release channels, and older builds, check out the SDK archive.

2. Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\src\flutter).

Warning: Do not install Flutter to a path that contains special characters or spaces.

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 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.

Note: As of Flutter's 1.19.0 dev release, the Flutter SDK contains the dart command alongside the flutter command so that you can more easily run Dart command-line programs. Downloading the Flutter SDK also downloads the compatible version of Dart, but if you've downloaded the Dart SDK separately, make sure that the Flutter version of dart is first in your path, as the two versions might not be compatible. The following command tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible.

```
C:\>where flutter dart
C:\path-to-flutter-sdk\bin\flutter
C:\path-to-flutter-sdk\bin\flutter.bat
C:\path-to-dart-sdk\bin\dart.exe      ::this should go after `C:\path-to-flutter-
sdk\bin\` commands
C:\path-to-flutter-sdk\bin\dart
C:\path-to-flutter-sdk\bin\dart.bat
```

As shown above, the command dart from the Flutter SDK doesn't come first. Update your path to use commands from C:\path-to-flutter-sdk\bin\ before commands from C:\path-to-dart-sdk\bin\ (in this case). After restarting your shell for the change to take effect, running the where command again should show that the flutter and dart commands from the same directory now come first

```
C:\>where flutter dart
C:\dev\src\flutter\bin\flutter
C:\dev\src\flutter\bin\flutter.bat
C:\dev\src\flutter\bin\dart
C:\dev\src\flutter\bin\dart.bat
C:\dev\src\dart-sdk\bin\dart.exe
```

However, if you are using PowerShell, in it there is an alias of Where-Object command, so you need to use where.exe instead.

```
PS C:\> where.exe flutter dart
```

To learn more about the dart command, run `dart -h` from the command line, or see the [dart tool page](#).

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://docs.flutter.dev/setup/#android-setup for detailed instructions.
```

The following sections describe how to perform these tasks and finish the setup process. Once you have installed any missing dependencies, you can run the flutter doctor command again to verify that you've set everything up correctly.

Note: If flutter doctor returns that either the Flutter plugin or Dart plugin of Android Studio are not installed, move on to Set up an editor to resolve this issue.

Warning: The Flutter tool may occasionally download resources from Google servers. By downloading or using the Flutter SDK you agree to the Google Terms of Service.

For example, when installed from GitHub (as opposed to from a prepackaged archive), the Flutter tool will download the Dart SDK from Google servers immediately when first run, as it is used to execute the flutter tool itself. This will also occur when Flutter is upgraded (e.g. by running the flutter upgrade command).

The flutter tool uses Google Analytics to report feature usage statistics and send crash reports. This data is used to help improve Flutter tools over time.

Flutter tool analytics are not sent on the very first run. To disable reporting, run flutter config --no-analytics. To display the current setting, use flutter config. If you opt out of analytics, an opt-out event is sent, and then no further information is sent by the Flutter tool.

Dart tools may also send usage metrics and crash reports to Google. To control the submission of these metrics, use the following options on the dart tool:

- --enable-analytics: Enables anonymous analytics.
- --disable-analytics: Disables anonymous analytics.

The Google Privacy Policy describes how data is handled by these services.

Android setup:

Note: Flutter relies on a full installation of Android Studio to supply its Android platform dependencies. However, you can write your Flutter apps in a number of editors; a later step discusses that.

Install Android Studio:

1. Download and install Android Studio.
2. Start Android Studio, and go through the ‘Android Studio Setup Wizard’. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
3. Run flutter doctor to confirm that Flutter has located your installation of Android Studio. If Flutter cannot locate it, run flutter config --android-studio-dir=<directory> to set the directory that Android Studio is installed to.

Set up your Android device:

To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the Android documentation.
2. Windows-only: Install the Google USB Driver.
3. Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.
4. In the terminal, run the flutter devices command to verify that Flutter recognizes your connected Android device. By default, Flutter uses the version of the Android SDK where your adb tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the ANDROID_SDK_ROOT environment variable to that installation directory.

Set up the Android emulator:

To prepare to run and test your Flutter app on the Android emulator, follow these steps:

1. Enable VM acceleration on your machine.
2. Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device...**
 - In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device...** (The **Android** submenu is only present when inside an Android project.)
 - If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device...**
3. Choose a device definition and select **Next**.
4. Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86_64* image is recommended.

5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable hardware acceleration.
6. Verify the AVD configuration is correct, and select **Finish**.

For details on the above steps, see Managing AVDs.

7. In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

Agree to Android Licenses:

Before you can use Flutter, you must agree to the licenses of the Android SDK platform. This step should be done after you have installed the tools listed above.

1. Make sure that you have a version of Java 11 installed and that your JAVA_HOME environment variable is set to the JDK's folder.

Android Studio versions 2.2 and higher come with a JDK, so this should already be done.

2. Open an elevated console window and run the following command to begin signing licenses.
\$ flutter doctor --android-licenses
3. Review the terms of each license carefully before agreeing to them.
4. Once you are done agreeing with licenses, run flutter doctor again to confirm that you are ready to use Flutter.

Experiment 2

Aim: To develop mobile app using flutter.

Code:

```
import 'package:flutter/material.dart';

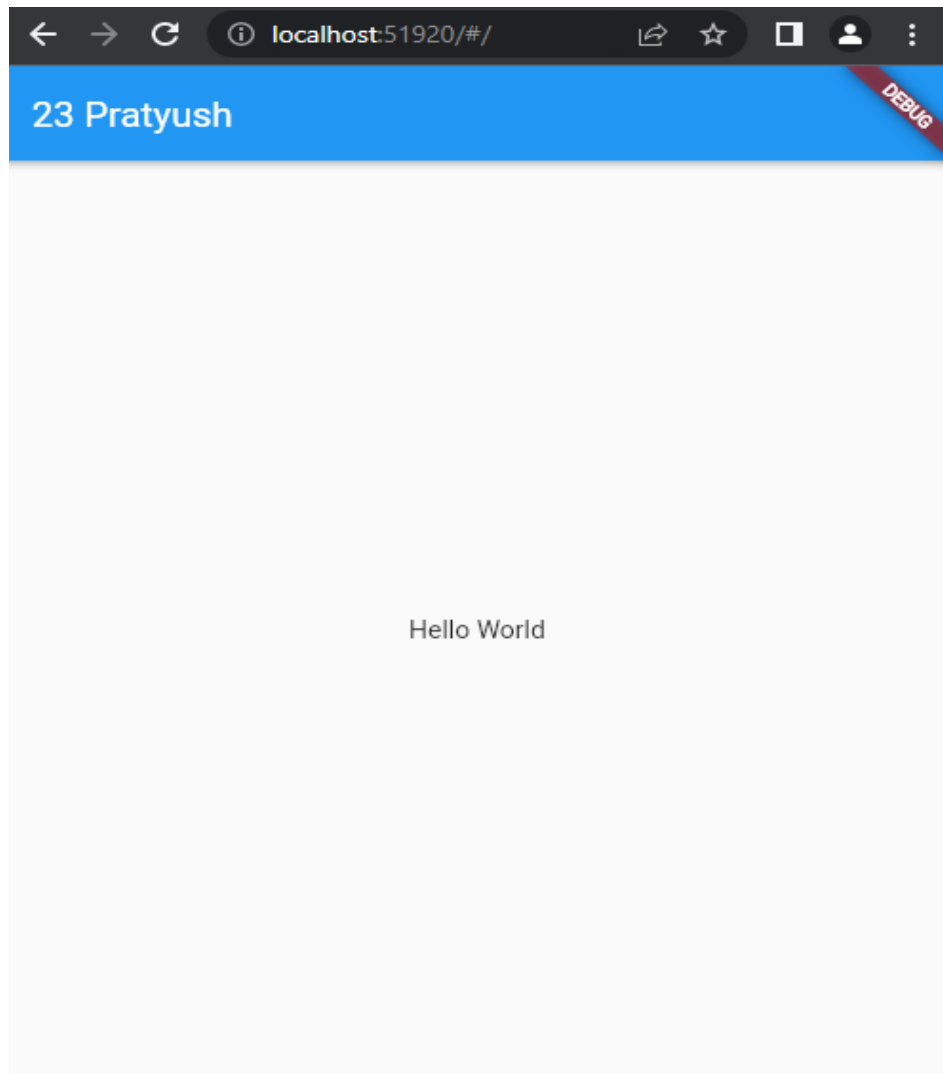
void main() {
  runApp(const hw());
}

class hw extends StatelessWidget {
  const hw({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    // Material App
    return MaterialApp(

      // Scaffold Widget
      home: Scaffold(
        appBar: AppBar(
          // AppBar takes a Text Widget
          // in it's title parameter
          title: const Text('23 Pratyush'),
        ),
        body: const Center(child: Text('Hello World')),
      ));
  }
}
```

Output:



Experiment 3

Aim: To design flutter UI by including common widgets.

Code:

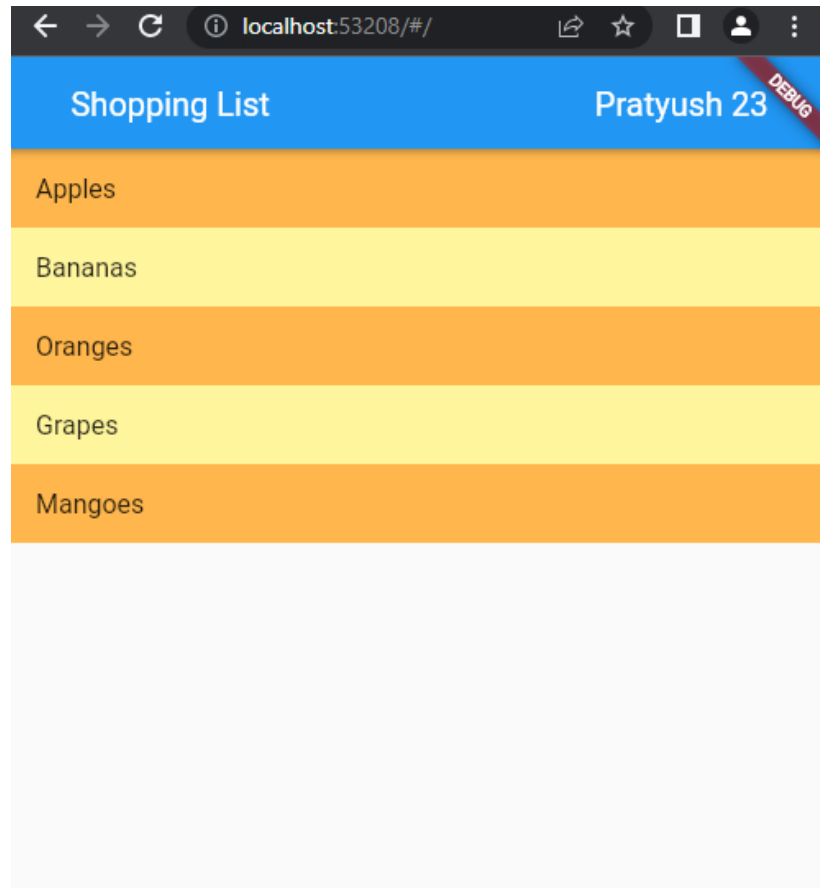
```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  final List<String> _shoppingItems = [ 'Apples', 'Bananas', 'Oranges', 'Grapes',
'Mangoes', ];

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Shopping List',
      home: Scaffold(
        appBar: AppBar(
          centerTitle: true,
          title: Text('Shopping List
Pratyush 23'),
        ),
        body: ListView.builder(
          itemCount: _shoppingItems.length,
          itemBuilder: (BuildContext context, int index) {
            return Container(
              color: index % 2 == 0 ? Colors.orange[300] : Colors.yellow[200],
              child: ListTile(
                title: Text(_shoppingItems[index]),
              ),
            );
          },
        ),
      ),
    );
  }
}
```

Output:



Experiment 4

Aim: To create an interactive form using form widget.

Theory:

Step 1: Create a Custom Input Text Field Class

In **main.dart** file, create a custom class named **MyTextFormField**, Inside the build method, I did some custom style and mainly used the **TextFormField** widget.

And when we instantiate this class to render some fields within the form, we will pass various properties like whether the field is a password or email or just a normal text field.

Also, we will pass a **validator** and **onSaved** functions. In the validator function, we will define the validation logic for the fields of data, and onSaved function, we will define when the validator is okay then where the field data will be stored.

Step 2: Create a Model Class

Let's create a **model.dart** file in the lib directory. When **onSaved** function in **MyTextFormField** class will be called; we will store the field's data in this **Model** object.

And we will pass the instance of this Model class with data from our primary screen to the result screen when the form is submitted, and there are no validation errors.

Step 3: Create a Result Screen

Let's create another class **Result** as a **result.dart** file, When we instantiate this class, we will pass a model object and finally show the model object's data on the screen.

Basically, when our form's data is valid and we submit the form, We will show this screen to the user.

Step 4: Install a Validator Package

In our form, there is an email field. To validate this email field properly, we want to use a function defined in a 3rd party library. So, mention this validator in the pubspec.yaml file, and click **packages get in Android Studio**.

```
dependencies:
  flutter:
    sdk: flutter //The following adds the Cupertino Icons font to your application.
    # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2
  validators: ^2.0.0+1
```

Step 5: Define the Form with Validation

So let's see the full source code of main.dart file. Here in the TestForm widget, we defined the form.

- To create a form, you first have to define a **Global Key**. Here we defined a **_formKey** as a Global Key.
- A **Global Key** is a key which when passed as a key in a widget, can differentiate the widget from all the widgets in a widget tree. Also, a Global Key can hold the state of the widget; in this case, we mentioned this **_formKey** can hold a **FormState**.
- Then we defined the **Form** widget and passed **_formKey** as a key in the Form. Now using **_formKey** we can retrieve the Form widget.
- We used **MyTextFormField** our custom class widget to instantiate all the required fields with various properties like for email we mentioned `isEmail` is true.
- You will see, all the fields have a **validator** and **onSaved** functions defined.
- When the user clicks the **Sign-Up button**, we call **formKey.currentState.validate()**. It will call all the validator functions defined in the TextFormFields.
- A **validator function**, either should return a String to notify what kind of error, **and if there is no error, it must have to return null**.
- If all validator functions inside the form defined in various TextFormFields return null, then the **_formKey.currentState.validate()** will return true which means all data in the form is validated.
- **So now we can save the data.** In our case, we saved data in the Model object and use navigation to pass the data to the Result screen. To save the data use **_formKey.currentState.save()** method which will call **all the onSaved functions** defined in TextFormFields.

Code:

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
```

```

Widget build(BuildContext context) {
  const appTitle = 'Interactive Flutter Form';

  return MaterialApp(
    title: appTitle,
    home: Scaffold(
      appBar: AppBar(
        title: Text('Interactive Flutter Form
Pratyush 23'),
        backgroundColor: Colors.cyan ,
      ),
      body: const MyCustomForm(),
    ),
  );
}

// Create a Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({Key? key}) : super(key: key);

  @override
  MyCustomFormState createState() {
    return MyCustomFormState();
  }
}

// Create a corresponding State class.
// This class holds data related to the form.
class MyCustomFormState extends State<MyCustomForm> {
  // Create a global key that uniquely identifies the Form widget
  // and allows validation of the form.
  //
  // Note: This is a GlobalKey<FormState>,
  // not a GlobalKey<MyCustomFormState>.
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    // Build a Form widget using the _formKey created above.
    return Form(
      key: _formKey,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [

          TextFormField(
            decoration: const InputDecoration(
              border: UnderlineInputBorder(),
              labelText: 'Name: ',
            ),
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter your name';
              }
            }
          )
        ],
      ),
    );
  }
}

```

```

        return null;
      },
    ),
    TextFormField(
      decoration: const InputDecoration(
        border: UnderlineInputBorder(),
        labelText: 'DOB: ',
      ),
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter your date of birth';
        }
        return null;
      },
    ),
    TextFormField(
      decoration: const InputDecoration(
        border: UnderlineInputBorder(),
        labelText: 'Email: ',
      ),
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter your email';
        }
        return null;
      },
    ),
    Padding(
      padding: const EdgeInsets.symmetric(vertical: 16.0),
      child: ElevatedButton(
        onPressed: () {
          // Validate returns true if the form is valid, or false otherwise.
          if (_formKey.currentState!.validate()) {
            // If the form is valid, display a snackbar. In the real world,
            // you'd often call a server or save the information in a database.
            ScaffoldMessenger.of(context).showSnackBar(
              const SnackBar(content: Text('Your data has been fetched')),
            );
          }
        },
        child: const Text('Submit'),
      ),
    ),
  ],
),
);
}
}
),
);
}
}

```

Output:

The screenshot shows a web browser window with the address bar displaying 'localhost:53028/#/'. The page title is 'Interactive Flutter Form' and the user is logged in as 'Pratyush 23'. A red 'DEBUG' banner is visible in the top right corner. The form contains three input fields: 'Name:' with the value 'Pratyush Lokhande', 'DOB:' with the value '02/03/2002', and 'Email:' with the value 'pratyush@gmail.com'. Below the fields is a blue 'Submit' button. At the bottom of the page, a dark grey footer bar contains the text 'Your data has been fetched'.

Name:	Pratyush Lokhande
DOB:	02/03/2002
Email:	pratyush@gmail.com

[Submit](#)

Your data has been fetched

Experiment 5

Aim: To design a layout of flutter app using layout widget.

Theory:

Type of Layout Widgets:

Layout widgets can be grouped into two distinct categories based on its child —

- Widget supporting a single child
- Widget supporting multiple child

Let us learn both type of widgets and its functionality in the upcoming sections.

Single Child Widgets

In this category, widgets will have only one widget as its child and every widget will have a special layout functionality.

For example, *Center* widget just centers its child widget with respect to its parent widget and *Container* widget provides complete flexibility to place its child at any given place inside it using different options like padding, decoration, etc.,

Single child widgets are great options to create high quality widget having single functionality such as button, label, etc.,

Here, we have used two widgets — a *Container* widget and a *Text* widget. The result of the widget is as a custom button as shown below -

Let us check some of the most important single child layout widgets provided by *Flutter* -

- **Padding** - Used to arrange its child widget by the given padding. Here, padding can be provided by **EdgeInsets** class,
- **Align** - Align its child widget within itself using the value of alignment property. The value for alignment property can be provided by Fractional Offset class. The Fractional Offset class specifies the offsets in terms of a distance from the top left.

Some of the possible values of offsets are as follows -

- FractionalOffset(1.0, 0.0) represents the top right.
- FractionalOffset(0.0, 1.0) represents the bottom left.
- **FittedBox** - It scales the child widget and then positions it according to the specified fit.

- **AspectRatio** - It attempts to size the child widget to the specified aspect ratio
- **ConstrainedBox**
- **Baseline**
- **FractinallySizdBox**
- **IntrinsicHeight**
- **IntrinsicWidth**
- **LiimitedBox**
- **OffStage**
- **OverflowBox**
- **SizedBox**
- **SizedOverflowBox**
- **Transform**
- **CustomSingleChildLayout**

Our hello world application is using material based layout widgets to design the home page. Let us modify our hello world application to build the home page using basic layout Widget as specified below -

- **Container** - Generic, single child, box based container widget with alignment, padding, border and margin along with rich styling features.
- **Center** - Simple, Single child container widget, which centers its child widget,

The complete code of the MyHomePage and MyApp widget is given in the code section of this experiment, in which:

- *Container* widget is the top level or root widget, *Container* is configured using *decoration* and *padding* property to layout its content.
- *BoxDecoration* has many properties like color, border, etc., to decorate the *Container* widget here, *color* is used to set the color of the container.
- *Padding* of the *Container* by using *EdgeInsets* class, which provides the option to specify the padding value.
- *Center* is the child widget of the *Container* widget. Again, *Text* is the child of the *Center* widget. *Text* is used to show message and *Center* is used to *Center* the text message with respect to the parent widget, *Container*.

Multiple Child Widgets

In this category, a given widget will have more than one child widgets and the layout of each widget is unique.

For example, *Row* widget allows the laying out of its children in horizontal direction whereas *Column* widget allows laying out of its children in vertical direction. By composing *Row* and *Column*, widget with any level of complexity can be built.

Let us learn some of the frequently used widgets in this section.

- **Row** - Allows arranging its children in a horizontal manner.
- **Column** - Allows arranging its children in a vertical manner.

- **ListView** - Allows arranging its children as list.
- **GridView** - Allows arranging its children as gallery.
- **Expanded** - Used to make the children of Row and Column widget to occupy the maximum possible area.
- **Table** - Table based widget.
- **Flow** - Flow based widget.
- **Stack** - Stack based widget.

Advanced Layout Application

In this section, let us learn how to create a complex user interface of *product listing* with custom design using both single and multiple child layout widgets. Here,

- We have created *MyHomePage* widget by extending *StatelessWidget* instead of default *StatefulWidget* and then removed the irrelevant code.
- Now, create a new widget, *ProductBox* according to the specified design as shown below-
- Please observe the code given in the *Code* section.
- *ProductBox* has used four arguments as specified below-
 - **name** – Product Name
 - **description** – Product Description
 - **price** – Price of the product
 - **image** – Image of the product
- *ProductBox* uses seven build-in widgets as specified below –
 - Container
 - Expanded
 - Row
 - Column
 - Card
 - Text
 - Image
- *ProductBox* is designed using the above-mentioned widget. The arrangement or hierarchy of the widget is specified in the diagram shown below –
- Now, place some dummy image (see below) for product information in the assets folder of the application and configure the assets folder in the *pubspec.yaml* file as shown below-

```
assets:
  - assets/iphone.jpg
  - assets/samsung.png
  - assets/xiomi.png
  - assets/realme.png
  - assets/oneplus.png
```

Code:

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Experiment 4',
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key = const Key('default_key')}} : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Product Listing 36 Tanmay"),
      ),
      body: ListView(
        shrinkWrap: true,
        padding: const EdgeInsets.fromLTRB(2, 10, 2, 10),
        children: <Widget>[
          ProductBox(
            name: "iPhone",
            description: "iPhone is the stylist phone ever",
            price: 100000,
            image: "iphone.jpg",
          ),
          ProductBox(
            name: "Samsung",
            description: "Samsung has the best camera",
            price: 10000,
            image: "samsung.png",
          ),
          ProductBox(
            name: "Xiaomi",
            description: "Xiaomi is the Cheapest phone ever",
            price: 1000,
            image: "xiomi.png",
          ),
          ProductBox(
            name: "Realme",
```

```

        description: "Realme is the Smallest phone ever",
        price: 1000,
        image: "realme.png",
    ),
    ProductBox(
        name: "OnePlus",
        description: "OnePlus is history now",
        price: 1000,
        image: "oneplus.png",
    ),
],
),
);
}
}






class ProductBox extends StatelessWidget {
  final String name;
  final String description;
  final int price;
  final String image;
  ProductBox(
    {super.key,
    required this.name,
    required this.description,
    required this.price,
    required this.image});

  Widget build(BuildContext context) {
    return Container(
      padding: const EdgeInsets.all(2),
      height: 120,
      child: Card(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            Image.asset("assets/$image"),
            Expanded(
              child: Container(
                padding: const EdgeInsets.all(5),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                  children: <Widget>[
                    Text(name,
                      style: const TextStyle(fontWeight: FontWeight.bold)),
                    Text(description),
                    Text("Price: ${price.toString()}"),
                  ],
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
};

```

```
}}
```

Output:

Product Listing		23 Pratyush
 iPhone	iPhone iPhone is the stylist phone ever Price: 100000	
 SAMSUNG	Samsung Samsung has the best camera Price: 10000	
 MI	Xiaomi Xiaomi is the Cheapest phone ever Price: 1000	
 R	Realme Realme is the Smallest phone ever Price: 1000	
 1+	OnePlus OnePlus is history now Price: 1000	

Experiment 6

Aim: To include icons, images, charts in flutter app.

Theory:

ICONS:

An icon is a graphic image representing an application or any specific entity containing meaning for the user. It can be selectable and non-selectable. For example, the company's logo is non-selectable. Sometimes it also contains a hyperlink to go to another page. It also acts as a sign in place of a detailed explanation of the actual entity.

IMAGE:

Images are fundamental to every app. They can provide crucial information by serving as visual aids or simply improve the aesthetic of your app. There are many ways to add an image to your Flutter application. This article will provide a comprehensive guide to the different methods along with detailed examples and sample code.

CHARTS:

A chart is a graphical representation of data where data is represented by a symbol such as a line, bar, pie, etc. In Flutter, the chart behaves the same as a normal chart. We use a chart in Flutter to represent the data in a graphical way that allows the user to understand them in a simple manner. We can also plot a graph to represent the rise and fall of our values. The chart can easily read the data and helps us to know the performance on a monthly or yearly basis whenever we need it.

Code:

Icons:

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Image Demo',
      home: HomePage(),
    );
  }
}

class HomePage extends StatelessWidget {
  const HomePage({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```

appBar: AppBar(
  title: const Text('36 Tanmay Icons'),
  centerTitle: true,
),
body: Column(
  children: <Widget>[
    //icon with label below it
    Container(
      padding: const EdgeInsets.all(30),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: <Widget>[
          Column(
            children: const <Widget>[
              Icon(Icons.camera_front, size: 70),
              Text('Front Camera'),
            ],
          ),
          Column(
            children: const <Widget>[
              Icon(Icons.camera_enhance, size: 70),
              Text('Camera'),
            ],
          ),
          Column(
            children: const <Widget>[
              Icon(Icons.camera_rear, size: 70),
              Text('Rear Camera'),
            ],
          ),
        ],
      ),
    ],
  ),
);
}

```

Image:

```

import 'package:flutter/material.dart';

void main() => runApp(MyApp());

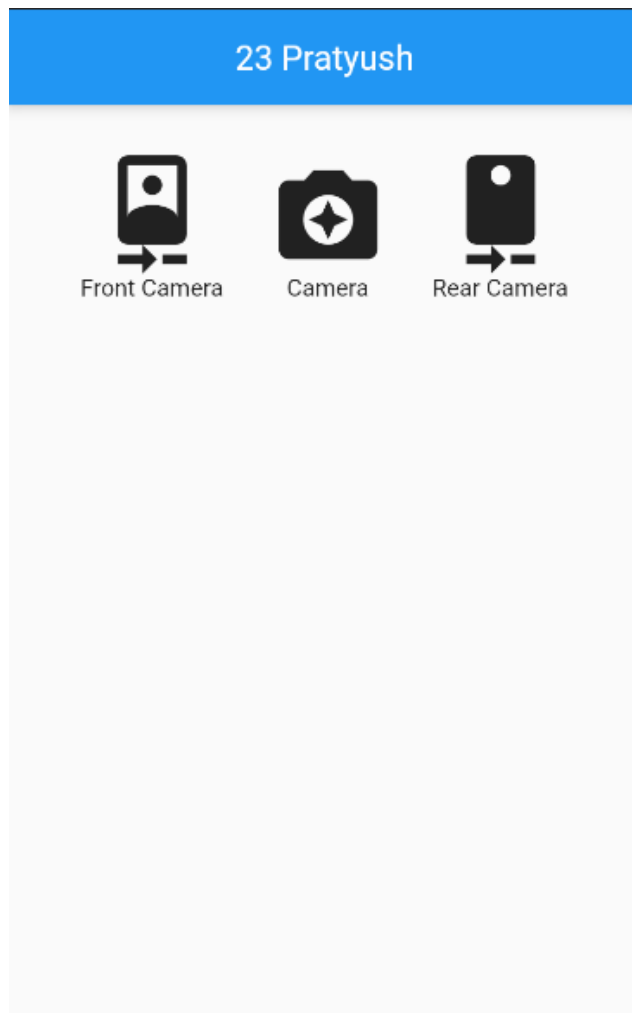
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Image Demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('36 Tanmay - Image'),
        ),
      ),
    );
  }
}

```

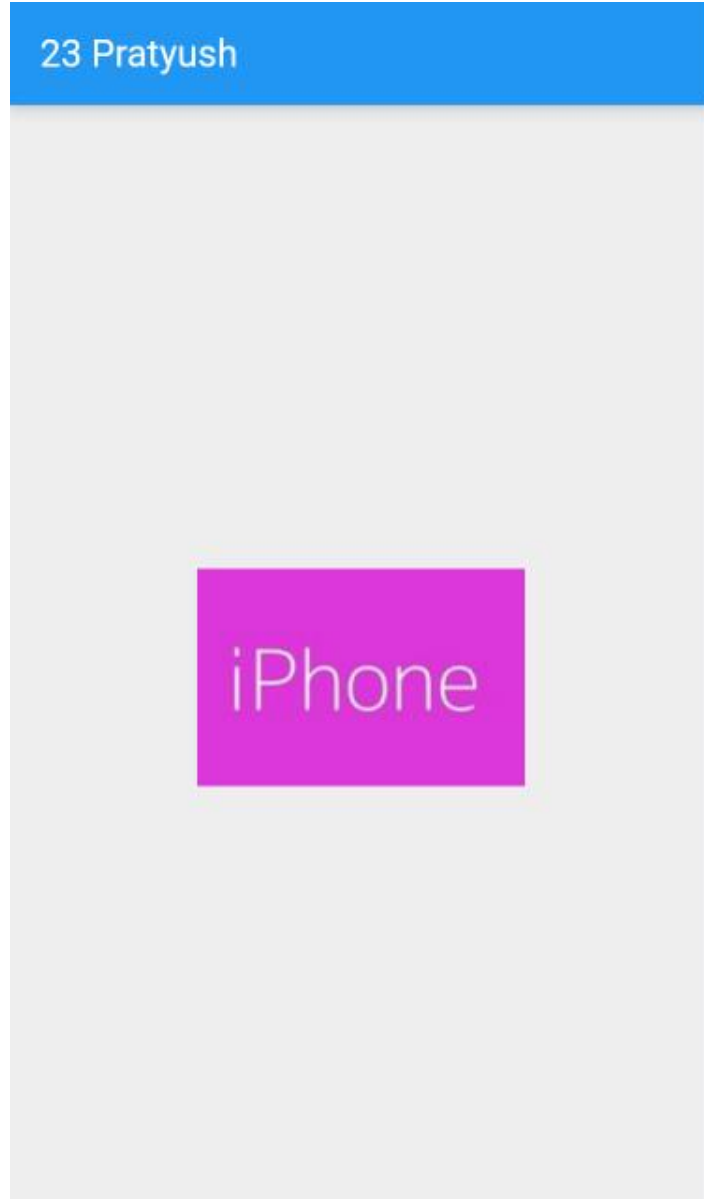
```
body: Container(  
  padding: const EdgeInsets.all(25),  
  color: Colors.grey[200],  
  alignment: Alignment.center,  
  child: Image.asset('xiomi.png'),  
),  
),  
);  
}  
}
```

Output:

Icons:



Images:



Experiment 7

Aim: To apply navigation, routing, and gestures in flutter app-

Theory:

Flutter Navigation and Routing

Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. **For example**, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

In Flutter, the screens and pages are known as routes, are known as **routes** and these routes are just a widget. In Android, a route is similar to an **Activity**, whereas, in iOS, it is equivalent to a **ViewController**.

In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as **routing**. Flutter provides a basic routing class **MaterialPageRoute** and two methods **Navigator.push()** and **Navigator.pop()** that shows how to navigate between two routes. The following steps are required to start navigation in your application.

Step 1: First, you need to create two routes in Java

Step 2: Then, navigate to one route from another route by using the **Navigator.push()** method.

Step 3: Finally, navigate to the first route by using the **Navigator.pop()** method.

Let us take a simple example to understand the navigation between two routes:

Create two routes

Here, we are going to create two routes for navigation. In both routes, we have created only a single button. When we tap the button on the first page, it will navigate to the second page. Again, when we tap the button on the second page, it will return to the first page.

Navigate to the second route using **Navigator.push()** method

The **Navigator.push()** method is used to navigate/switch to a new route/page/screen. Here, the **push()** method adds a page/route on the stack and then manage it by using the **Navigator**. Again we use **MaterialPageRoute** class that allows transition between the routes using the platform-specific animation. The below code explains the use of **Navigator.push()** method.

Return to the first route using `Navigator.pop()` method

Now, we need to use `Navigator.pop()` method to close the second route and return to the first route. The **pop()** method allows us to remove the current route from the stack, which is managed by the `Navigator`.

To implement a return to the original route, we need to update the **onPressed()** callback method in the **SecondRoute** widget.

Now, let us see the full code to implement the navigation between two routes. First, create a Flutter project and insert the following code in the **main.dart** file.

Code:

```
import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Navigation',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: FirstRoute(),
    ),
  );
}

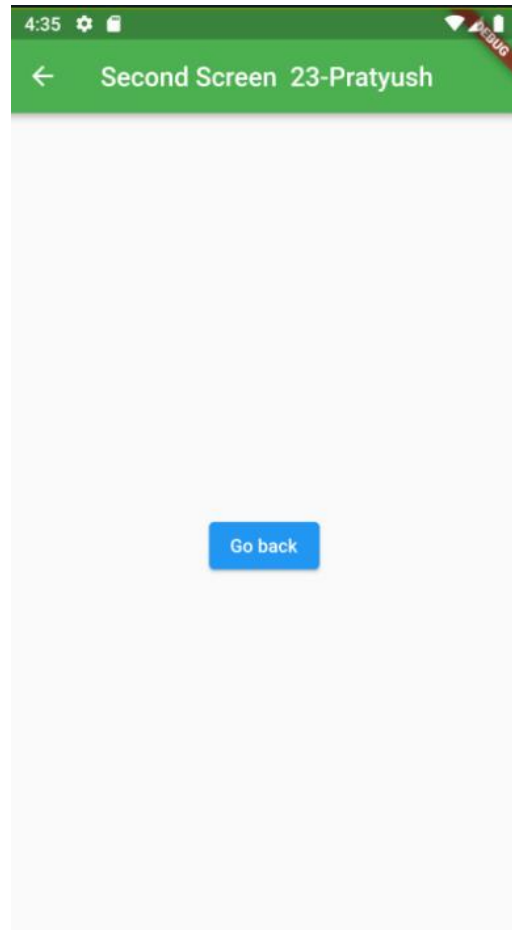
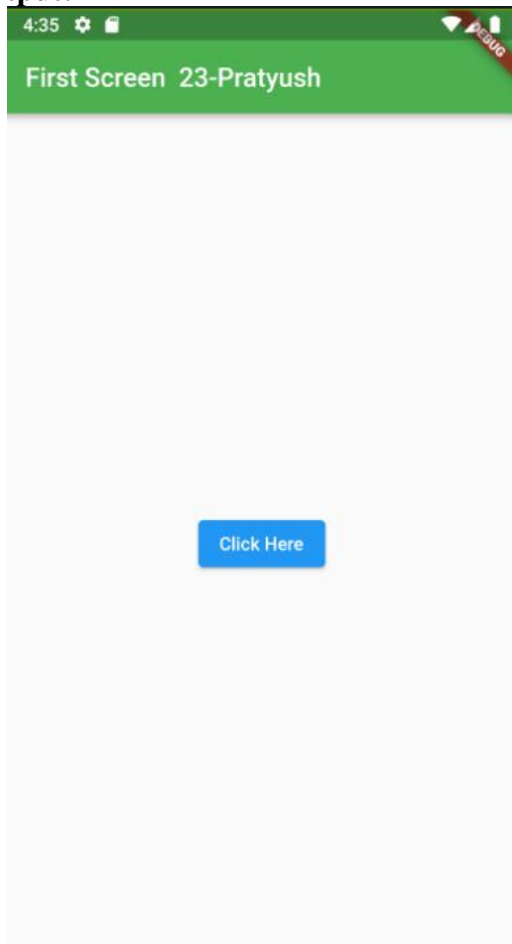
class FirstRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Screen 36 Tanmay'),
      ),
      body: Center(
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blue, // Background color
          ),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondRoute()),
            );
          },
          child: const Text('Click Here'),
        ),
      ),
    );
  }
}
```

```

class SecondRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Screen 36 Tanmay'),
      ),
      body: Center(
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blue,
          ),
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back'),
        ),
      ),
    );
  }
}

```

Output:

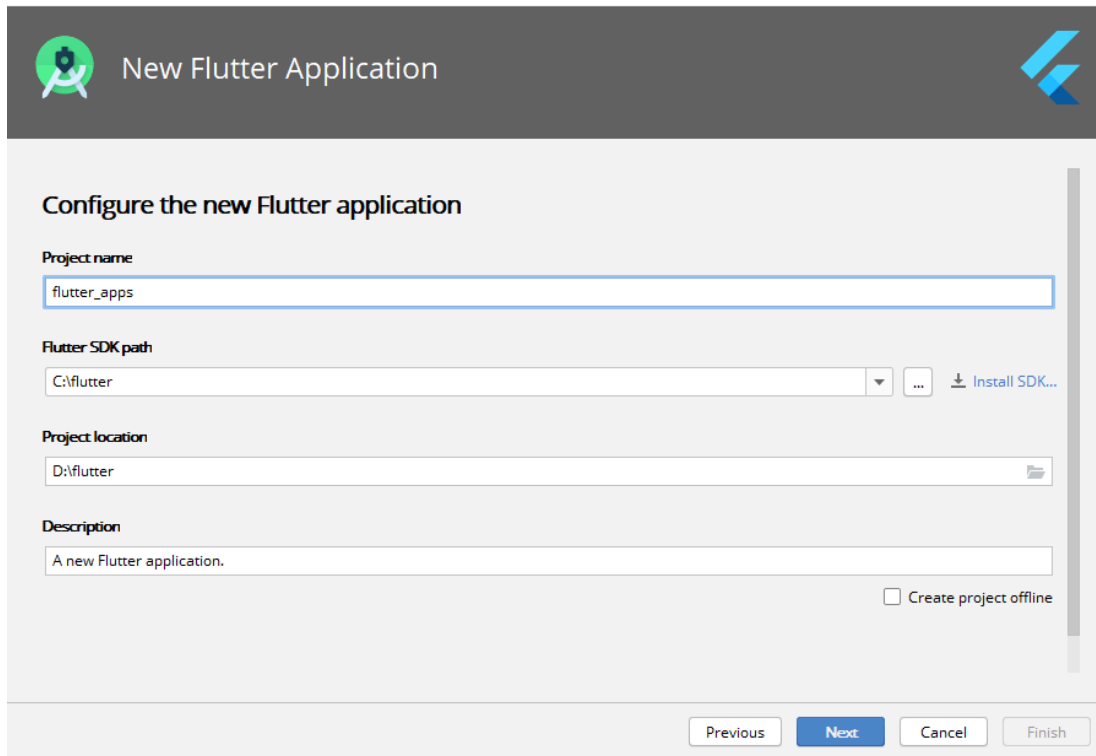


Experiment 8

Aim: To connect flutter UN with firebase database.

Theory: Setting up the project

Launch Android Studio and create a new Flutter project.



The screenshot shows the 'New Flutter Application' dialog box in Android Studio. The dialog has a dark header bar with the Flutter logo on the left and the Android Studio logo on the right. The main area is light gray and contains the following fields and options:

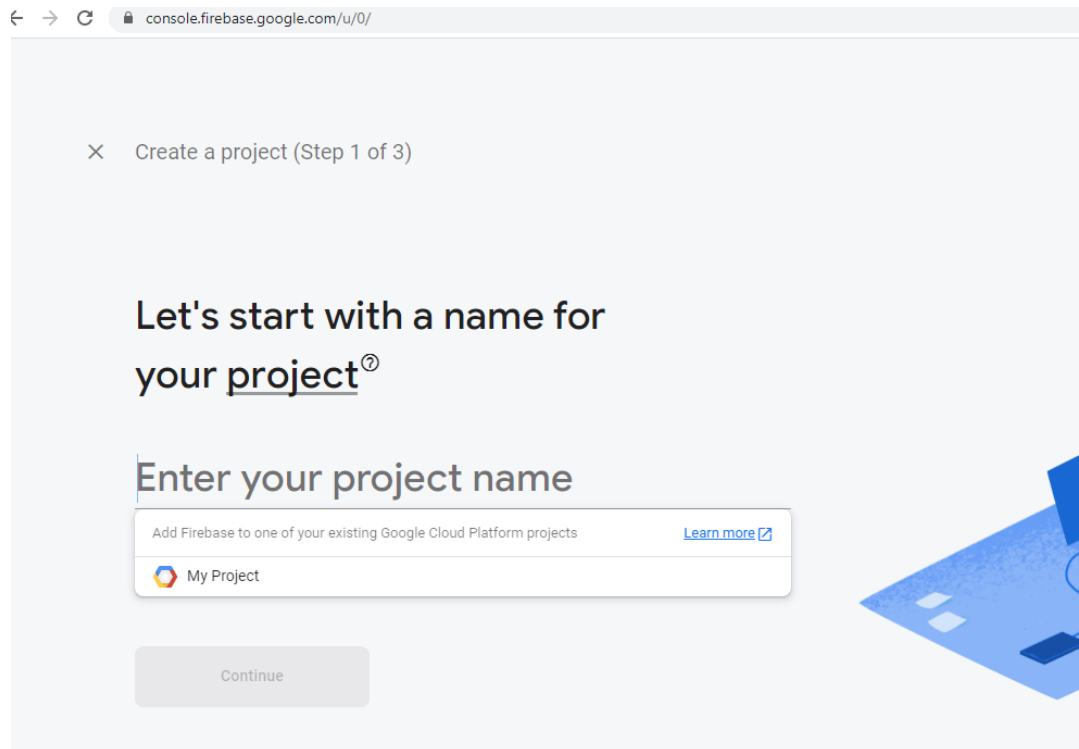
- Project name:** A text field containing 'flutter_apps'.
- Flutter SDK path:** A text field containing 'C:\flutter', a dropdown arrow, an ellipsis button, and a link that says 'Install SDK...'.
- Project location:** A text field containing 'D:\flutter' and a folder icon button.
- Description:** A text field containing 'A new Flutter application.'.
- Create project offline:** An unchecked checkbox.
- Navigation buttons:** 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Finish'.

Open the pubspec.yaml file and add the following Flutter dependencies.

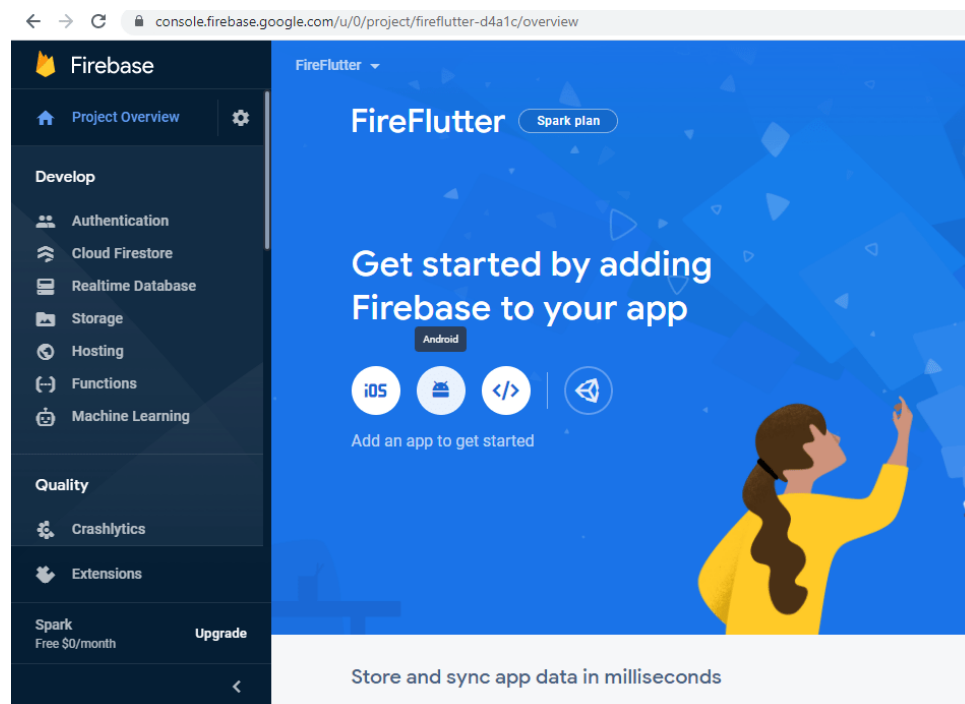
```
dev_dependencies:  
  flutter_test:  
    sdk: flutter  
  firebase_database: ^4.4.0  
  firebase_core : ^0.5.3
```

Use the pub get command to retrieve appropriate dependencies.

Next, open your browser and navigate to Firebase's website. We need to create a new firebase project, as shown below.



Add your application's package name as shown in the image below.



You can find your package name in the app-level - build.gradle file. Click register app after completing this step.

Download the google-services.json file and paste it into the android/app folder.

Paste classpath 'com.google.gms:google-services:4.3.3' in the project level — build.gradle file.

```
dependencies {  
    classpath 'com.android.tools.build:gradle:3.5.0'  
    classpath "org, jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"  
    classpath 'com.google.gms:google-services:4.3.3'  
}
```

Navigate to the app-level - build.gradle file and add apply plugin: 'com.google.gms.google-services' as shown below.

```
apply plugin: 'com.android.application'  
apply plugin: 'kotlin-android'  
apply from: "$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"  
apply plugin: 'com.google.gms.google-services'
```

Alter the project is created successfully, navigate back to the Firebase console's realtime database section, and create a new database.

In the rules section, ensure that you click on test mode rather than the locked mode, This allows us to read and write the data without authentication. Note that these rules should not be used in a production application since anyone can access the data.

```
{  
  "rules": {  
    "read": "now < 1610053200000", //2021-1-8  
    "write": "now < 1610053200000", // 2021-1-8  
  }  
}
```

We have successfully set up Firebase and the Flutter project. Let's design the UI.

Building the UI:

The app will have a simple user interface. It will allow a user to input a word or sentence and click on a button to save it to the Firebase database.

Ensure that you have the following imports at the top of the Home page.

- package:flutter/material.dart - This import allows you to access Flutter's built-in widgets and other functionalities.
- package:firebase_database/firebase_database.dart - This import helps access the Firebase Realtime database features.
- package:firebase_core/firebase_core.dart - This dependency enables You to connect to different Firebase products.

Storing data

We will initialize a Firebase databaseReference object and use it to store data.

```
//database reference object
final databaseRef= FirebaseDatabase._instance_reference();

void addData(String data) {
    databaseRef.push().set({ "name" data, "comment": 'A good season'});
}
```

The addData function stores the user input along with a string ('comment': 'A good season'). We'll call the addData function when the RaisedButton is clicked.

```
onPressed: () {
  addData(textcontroller.text);
  //call method flutter upload
})),
```

We'll access the user input using a textcontroller. This element allows us to listen for changes in a TextField.

We need to include the textcontroller in the HomePageState class.

```
class _HomePageState extends State<HomePage> {
  final textcontroller = TextEditingController();
```

The textcontroller is the added to the TextField widget, as shown below.

```
TextField(controller: textcontroller),
```

Please note that we need to use a FutureBuilder for async operations. This class allows the application to retrieve results once the network operation is completed.

This class also helps us initialize the Firebase app.

```
class _HomePageState extends State<HomePage> {
  final Future<Firebase App> _future = Firebase.initializeApp();
```

_future is then added to the FutureBuider in the Scaffold. This operation is illustrated below.

As noted, FutureBuilder helps in awaiting long-running operations.

```
return Scaffold(
  appBar: AppBar(
    title: Text("lirebase Demo"),
  ),
  body: FuturcBuilder(
    future: _future,
    builder: (context, snapshot) {
      if (snapshot.hasError) {
```



```
        return Text(snapshot.error.toString());  
    } else {  
        return Container(  

```

Retrieving data: For simplicity, we will retrieve and print out the data in the console. This is shown below.

```
void printFirebase(){  
    databaseRef.once().then((DataSnapshot snapshot) {  
        print('Data : ${snapshot.value}");  
    });  
}
```

We'll call the function above in the `_HomePageState` class's build method.

Experiment 9

Aim: To test & deploy production ready flutter app on an android platform.

Theory:

Flutter is Google's mobile app SDK for crafting high-quality native interfaces on iOS and Android in record time. Flutter works with existing code, is used by developers and organizations around the world, and is free and open source.

For users, Flutter makes beautiful app UIs come to life. For developers, Flutter lowers the bar to entry for building mobile apps. It speeds up the development of mobile apps and reduces the cost and complexity of app production across iOS and Android. For designers, Flutter helps deliver the original design vision, without loss of fidelity or compromises. It also acts as a productive prototyping tool.

System requirements:

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

- Operating Systems: macOS / Windows (64-bit).
- Disk Space: 700 MB (does not include disk space for IDE/tools).
- Tools: Flutter depends on these command-line tools being available in your environment.

Setting up Android Studio:

1. Install Android Studio
2. Download and install Android Studio.
3. Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Platform-Tools, and Android SDK Build-Tools, which are required by Flutter when developing for android.

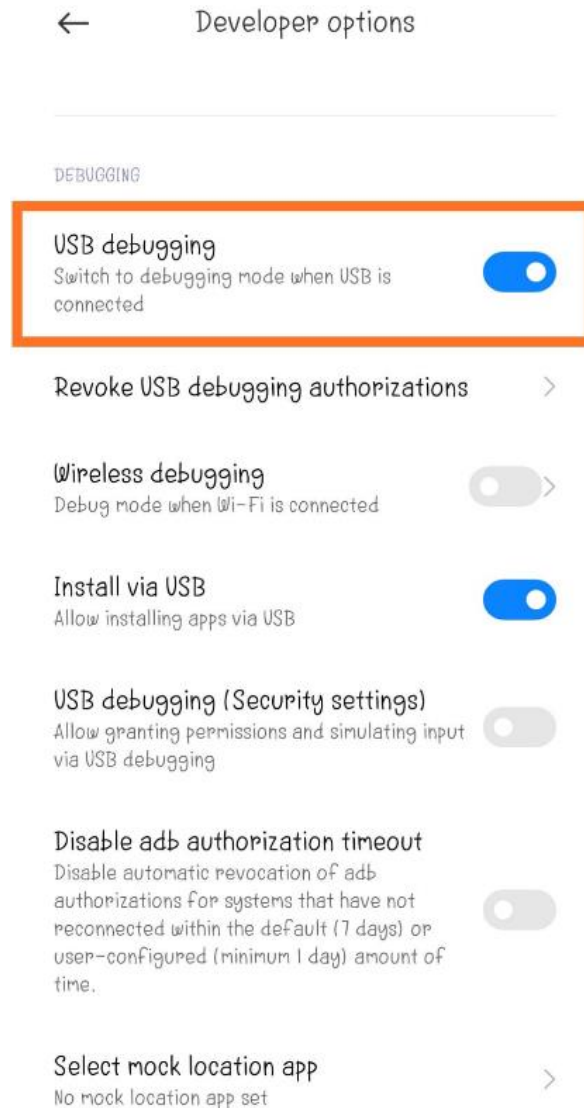
Build your app on an Android device:

To prepare to run and test your Flutter app on an Android device, you'll need an Android device running Android 4.1 (API level 16) or higher.

Get your Android device ready and follow the steps:

1. Enable Developer Options and USB debugging on your device:

- Open the Settings app.
- Search for 'About phone' and select it.
- Find and tap Build number 7 times.
- Return to the previous screen to find Developer options near the bottom.
- Scroll down and enable USB debugging.



- 2. Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.**
- 3. In the terminal, run the *flutter devices* command to verify that Flutter recognizes your connected Android device.**

```
Command Prompt
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

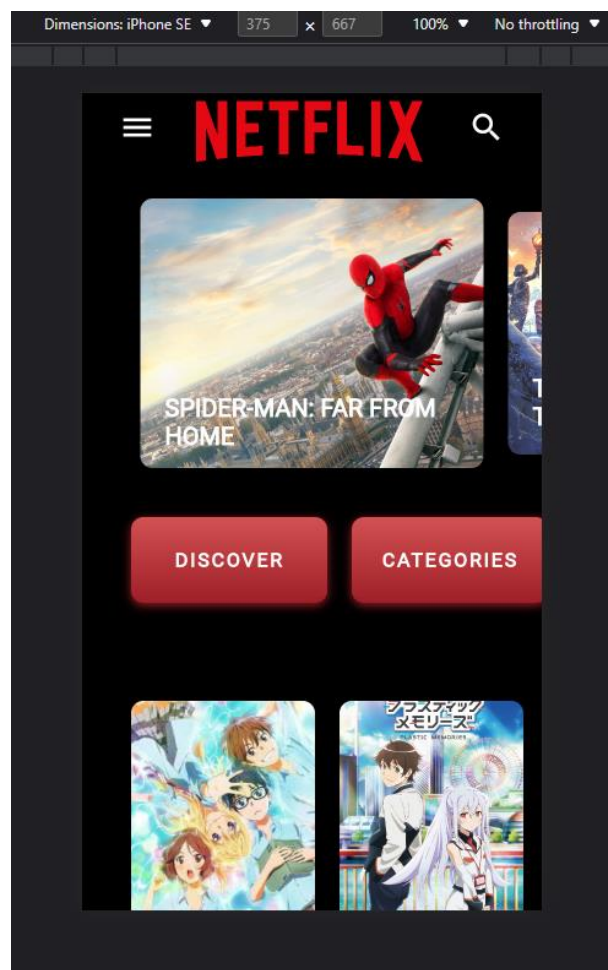
C:\Users\Sahil>flutter devices
4 connected devices:

SM_A505F (mobile) • R2BM424RBWA • android-arm64 • Android 11 (API 30)
Windows (desktop) • windows • windows-x64 • Microsoft Windows [Version 10.0.19044.1645]
Chrome (web) • chrome • web-javascript • Google Chrome 100.0.4896.127
Edge (web) • edge • web-javascript • Microsoft Edge 100.0.1185.50

C:\Users\Sahil>
```

4. Now all you need to do is run *flutter run* to run it on the android device.

Output:



Experiment 10

Aim: To create a responsive user interface using jQuery mobile for Ecommerce application.

Theory:

Over the last few years the devices used to access web applications have grown like anything. We now have mobiles, tablets, desktops, laptops, TV etc. that can be used to access a web site. These all devices vary a lot in sizes. Writing individual applications to cater to all of these different sized and variety of devices is nearly impossible. So we need to design web sites in way that they adjust to the screen size on which they are displayed. The technique developed to solve this problem is Responsive Web Design.

Responsive web design is a series of techniques and technologies that are combined to make delivering a single application across a range of devices as practical as possible. Below is an example of how Google news site looks on different devices.



Pillars of Responsive Design

Fluid Grids

A grid is a way to manage space in a layout in the web world. Fluid grids refer to flexible grid-based layouts that use relative sizing. Traditionally the grids were specified with fixed width columns with widths specified in pixels. But for responsive web design the grid columns widths should be relative to their containers. This helps the responsive web applications to adapt to different screen sizes. In responsive web design the widths are expressed in percentage most of the time.

Media Queries

Media queries assess the capabilities of device browser and apply styles based on the capabilities that match the query criteria. Media queries enable the web application to adjust itself for optimal viewing based on the browser capabilities.

Flexible Images and Media

This feature enables the images and other media to adapt according to the different device sizes and display resolutions. The most basic technique to adapt images and media is by using scaling or CSS overflow property. Though these techniques adapt the image according to the device size but the image download still take up the user's mobile bandwidth. For better user experience one can use different set of images and media for different devices.

jQuery Mobile Framework

The *jQuery Mobile Framework* is an open source cross-platform UI framework. It is built using HTML5, CSS3, and the very popular jQuery JavaScript library, and it follows Open Web standards. It provides touch-friendly UI widgets that are specially styled for mobile devices. It has a powerful theming framework to style your applications. It supports AJAX for various tasks, such as page navigation and transitions.

How jQuery Mobile supports RWD?

Unified UI

jQuery Mobile delivers unified user experience by designing to HTML5 and CSS3. A single jQuery codebase will render consistently across all supported platforms without needing any customizations.

Progressive Enhancement

Progressive enhancement defines layers of compatibility that allow any user to access the basic content, services, and functionality of a website, while providing an enhanced experience for browsers with better support of standards. jQuery Mobile is totally built using this technique. Below is an example of how a page built using jQuery Mobile will look on a basic phone as well as feature rich phone.

CSS Selector

jQuery Mobile has predefined set of CSS classes that it applies to the HTML elements depending on the current orientation or size of the device.

Orientation Classes

If the device is in landscape mode, jQuery Mobile will apply *.landscape* class to the HTML elements. Similarly if the device is in portrait mode, *.portrait* class is applied by jQuery Mobile to the HTML elements. One can override these orientation selectors in application CSS to define new styles for application on device orientation. For example if we want to have an image background in portrait orientation, we can have following CSS rule:

CSS
<pre>.portrait section { background-image: url(images/portrait-background.png); }</pre>

The above rule will set the background image if the device is in portrait mode. In the landscape mode the application will have the default background.

Breakpoint Classes

In addition to orientation changes, jQuery Mobile provides CSS Selectors for handling different screen sizes. jQuery Mobile refers to these as *breakpoint classes*. jQuery Mobile has defined screen size breakpoints at 320 pixels, 480 pixels, 1024 pixels etc. Corresponding to these sizes, min-width and max-width CSS classes are defined in jQuery Mobile like *min-width-320px*, *min-width-480px*, *max-width-1024px* etc.

As with orientation classes, jQuery Mobile applies these breakpoint classes to the HTML elements depending on the screen size. These can be defined in the application CSS to override the default behavior. For example if we wanted to have different background images depending on the screen size, the following CSS could be used:

CSS
<pre>.min-width-320px section, .max-width-480px section { background-image: url(images/login-bg-320.jpg); } .min-width-480px section, .max-width-768px section { background-image: url(images/login-bg-480.jpg); }</pre>

The above CSS will load *login-bg-320.jpg* if the size is between 320 and 480 pixels wide and *login-bg-480.jpg* if the size is between 480 and 768 pixels.

Flexible Layout

In jQuery Mobile most of the components and form elements are designed to be of flexible width so that they comfortably fit the width of any device. Additionally form elements and labels are represented differently based on the screen size. On smaller screens, labels are stacked on the top of form element while on wider screens, labels and elements are styled to be on the same line in 2 column grid layout.

The image displays two versions of a login form to illustrate how jQuery Mobile adapts the layout based on screen size. On the left, a wide screen shows a two-column grid layout where labels ('User Name:', 'Password:', 'Search:') are aligned to the left of their respective input fields. On the right, a narrow screen shows the labels stacked vertically above the input fields to save horizontal space. The 'Search' field includes a magnifying glass icon.

Grid Layout

jQuery Mobile has built in flexible grid layout. The grids are 100% in width, completely invisible and do not have padding or margins. Hence they did not interfere with the styling of components placed inside them. jQuery Mobile grid layout can be used to create layout with two to five columns. The columns widths are adjusted according to the screen width and number of columns. There are predefined layouts defined for creating grid. These are named as ui-grid-a (2 columns grid), ui-grid-b (3 columns grid), ui-grid-c (4 columns grid) and ui-grid-d (5 columns grid). Within the grid container, child elements are assigned ui-block-a/b/c/d/e in a sequential manner which makes each "block" element float side-by-side, forming the grid.

The default behavior of grids in jQuery Mobile is to lay the columns side by side. But for the application to respond to screen sizes we require the columns to be stacked on smaller screens. jQuery Mobile provides a CSS style to make this happen. The style name is *ui-responsive*. When this style is added to the grid container the grid columns would be stacked if the screen width is below 560px and on bigger screens the grid would be represented in multi-column layout.



Responsive Tables

This is a newly added feature to jQuery Mobile. This allows us to display large amount of tabular data in a way that looks good on both mobiles and desktops. There are two modes of responsive tables:

Reflow: This mode vertically stacks the cells in rows by default so that the data could be easily readable on mobile phones. Additional style needs to be applied to make the table display in traditional row-column format. This is the default mode for tables defined using data-role='table'.

S. No.	Book Title	Author	Year	Reviews
1	Book title 1	Author 1	1944	32
2	Book title 2	Author 2	2003	14
3	Book title 3	Author 3	2010	23

S. No.	1
Book Title	Book title 1
Author	Author 1
Year	1944
Reviews	32

S. No.	2
Book Title	Book title 2
Author	Author 2
Year	2003
Reviews	14

S. No.	3
Book Title	Book title 3
Author	Author 3
Year	2010
Reviews	23

Column Toggle: This mode hides the low priority columns at narrower widths. The column priority is specified using data-priority attribute in <th> element. The priority can be between 1 (Highest) and 6 (Lowest). If data-priority attribute is missing for a column then that column is always displayed. A “Column...” button is added alongside the table to allow users to show/hide the columns.

S. No.	Book Title	Author	Year	Reviews
1	Book title 1	Author 1	1944	32
2	Book title 2	Author 2	2003	14
3	Book title 3	Author 3	2010	23

Book Title	Author	Year
Book title 1	Author 1	1944
Book title 2	Author 2	2003
Book title 3	Author 3	2010

Book Title	Author
Book title 1	Author 1
Book title 2	Author 2
Book title 3	Author 3

Similar to grid layouts the preset breakpoint *ui-responsive* can also be applied to tables. This will make the tables responsive.

Media Queries

In addition to the preset breakpoints, the CSS3 media queries can easily be used with jQuery Mobile. As jQuery Mobile is a JavaScript framework for HTML sites, media queries work seamlessly with jQuery Mobile. One can easily define and apply new media queries to jQuery Mobile applications. For example, if we want the grid to be stacked when the screen width is 720px or less, we can define the media query like the following:

CSS

```
/* stack all grids below 45em (720px) */
@media all and (max-width: 45em) {
    .grid-breakpoint .ui-block-a,
    .grid-breakpoint .ui-block-b,
    .grid-breakpoint .ui-block-c,
    .grid-breakpoint .ui-block-d,
    .grid-breakpoint .ui-block-e {
        width: 100%;
        float: none;
    }
}
```

Conclusion

As we can see jQuery Mobile is built keeping responsive design in mind. It provides a lot of out of box features to support responsive design. One of the advantages of jQuery Mobile is that it provides a base framework which is well tested to be working fine on various screen sizes and browsers. This makes it easier to develop applications using jQuery Mobile as one can concentrate on the application features without worrying about the cross browser compatibility and screen size issues.

Experiment 11

Aim: To code and register a service worker & complete the install and activation process for a new service for the Ecommerce PWA.

Theory:

Progressive Web Apps are a new type of web app that offers native-like capabilities, yet their reach and performance are indistinguishable from a website. This means that, unlike a native app, a Progressive Web App's UI won't be limited by the user's device or by the system.

E-commerce PWAs can also access the device's offline capabilities and can use system-level notification channels to meet the needs of users across platforms. They are optimized for all platforms, and they are fast, reliable, engaging, and usable in every major browser.

WHAT IS PWA?

Progressive web apps in Ecommerce are like native apps with a better build on the web. There is no need to download and install a separate app. PWA lives in a browser, providing a fast, more reliable, and engaging user experience. Progressive web apps have an installation experience that makes users feel like they are getting an app for the first time.

In addition to providing a smooth, native-like experience, progressive web apps are more secure. They use an application shell, called a service worker, to cache data and resources and serve it to the user. This allows the app to be installed on a new device without affecting the user's experience.

HOW DOES PWA WORK?

PWA enables users to access data and content, typically through a browser interface that is hosted on remote Servers connected to the internet. In this way, PWA differs from native apps that often store and access data locally on the devices themselves.

Here are 10 common features of progressive web Ecommerce applications:

1. Responsive design
2. Functioning regardless of the internet connectivity of the device
3. Fast app-like interactions and animations
4. Instant updates
5. Compatibility with all devices
6. Security
7. Indexing by search engines
8. Interaction with users via push notifications
9. Possibility to install on a home screen
10. Easy distribution via links without the need for installation

It allows PWAs to have a more seamless experience for 'users, as the web technologies are already supported and well-understood.

Code -

1. Get set up

Clone the E-Commerce lab repository with Git using the following command:

```
git clone https://github.com/google-developer-training/pwa-ecommerce-demo.git
```

Navigate into the cloned repo:

```
cd pwa-ecommerce-demo
```

If you have a text editor that lets you open a project, then open the Project folder in the ecommerce-demo folder. This will make it easier to stay organized. Otherwise, open the folder in your computer's file system. The project folder is where you will build the app.

In a command window at the project folder, run the following command to install the project dependencies (open the package.json file to see a list of the dependencies):

```
npm install
```

Then run the following:

```
npm run serve
```

This runs the default task in gulpfile.babel.js which copies the project files to the appropriate folder and starts a server. Open your browser and navigate to localhost:8080. The app is a mock furniture website, "Modern Furniture Store". Several furniture items should display on the front page.

When the app opens, confirm that a service worker is not registered at local host by checking developer tools. If there is a service worker at localhost, unregister it so it doesn't interfere with the lab.

2. Register the service worker

To complete TODO SW-2 in app/scripts/main.js, write the code to register the service worker at service-worker.js. The code should include a check for whether service worker is supported by the browser. Remember to save the file when you have finished.

3. Cache the application shell

To complete TODO SW-3 in `app/service-worker.js`, write the code to cache the following list of files in the service worker install event, Name the cache `e-commerce-v1`.

- `'/'`,
- `index.html`,
- `scripts/main_min.js`,
- `styles/main.css`,
- `images/products/BarrelChair.jpg`,
- `images/products/C10.jpg`,
- `images/products/C12.jpg`,
- `images/products/CP03_blue.jpg`,
- `images/products/CPC_RECYCLED.jpg`,
- `images/products/CPFS.jpg`,
- `images/products/CP02_red.jpg`,
- `images/products/CPT.jpg`,
- `images/products/CS1.jpg`

Save the file.

4. Use the cache-first strategy

To complete TODO SW-4 in `app/service-worker.js`, write the code to respond to fetch requests with the "cache, falling back to network" strategy. First, look for the response in the cache and if it exists, respond with the matching file. If the file does not exist, request the file from the network and cache a clone of the response. Save the file when you have completed this step.

5. Delete outdated caches

To complete TODO SW-5 in `app/service-worker.js`, write the code to delete unused caches in the activate event handler. You should create an "allowlist" of caches currently in use that should not be deleted (such as the `e-commerce-v1` cache). Use `caches.keys()` to get a list of the cache names. Then, inside `Promise.all`, map the array containing the cache names to a function that deletes each cache not in the allowlist. Save the file when you have completed this step.

6. Test it out

To test the app, close any open instances of the app in your browser and stop the local server (`ctrl+c`).

Run the following in the command line to clean out the old files in the `dist` folder, rebuild it, and serve the app:

```
npm run serve
```

Open the browser and navigate to localhost:8080. Inspect the cache to make sure that the specified files are cached when the service worker is installed. Take the app offline and refresh the page. The app should load normally!

Experiment 12

Aim: To deploy an Ecommerce PWA using SSL enabled static hosting solution.

Theory:

We need the following **two** prerequisites in order to host a website/web-app in AWS:

- An active **AWS** account (Free Tier can be used)
- A **domain** (Can be either a free or purchased domain). The domain which we'll be using for demo will be **devonwijesinghe.tk**

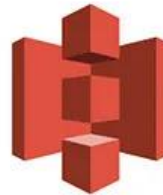
What services in AWS will be used?



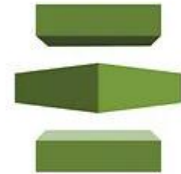
Simple Storage Service
(S3)



Route 53



Cloud Front



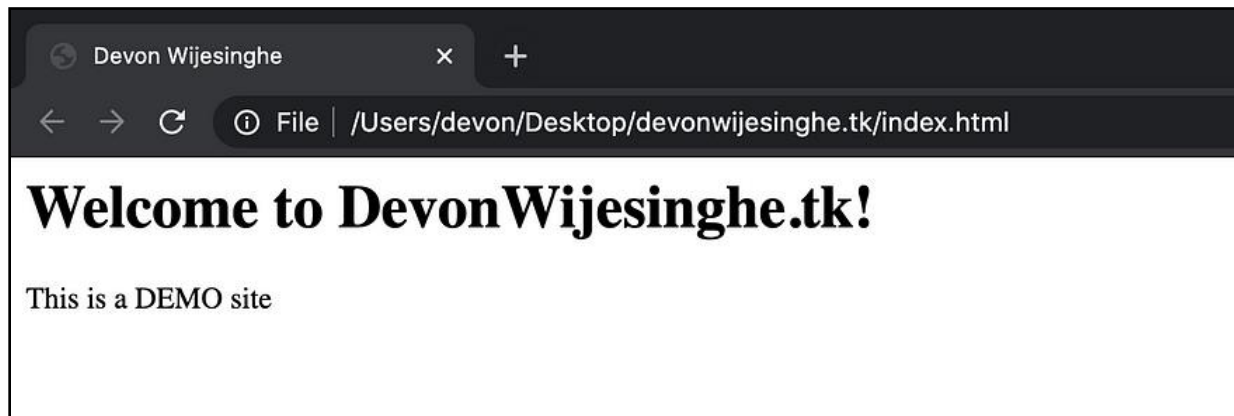
Certificate Manager

Step 1: Prepare your Website or Web-App

If you have a **simple HTML website**, you don't have to do anything in this step, simply **skip this step**.

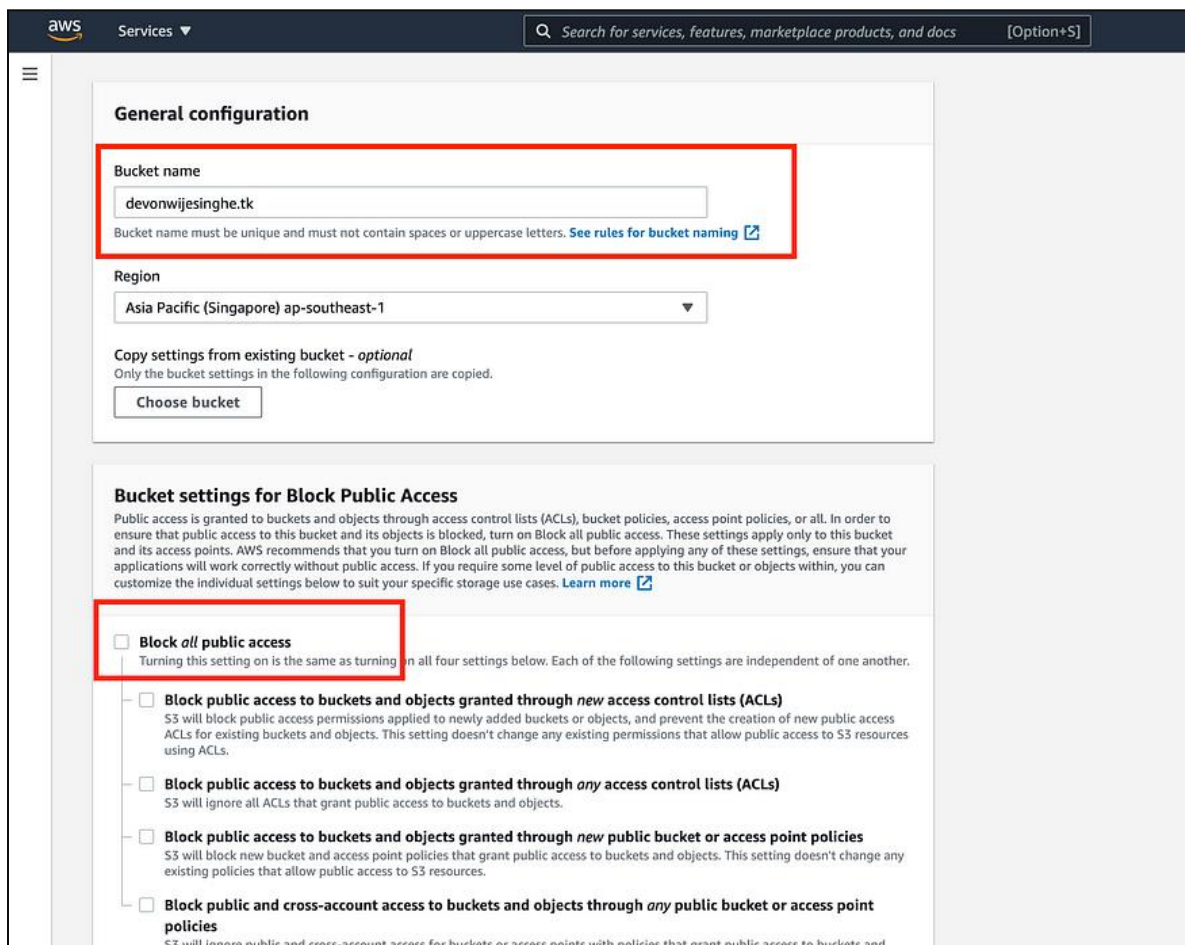
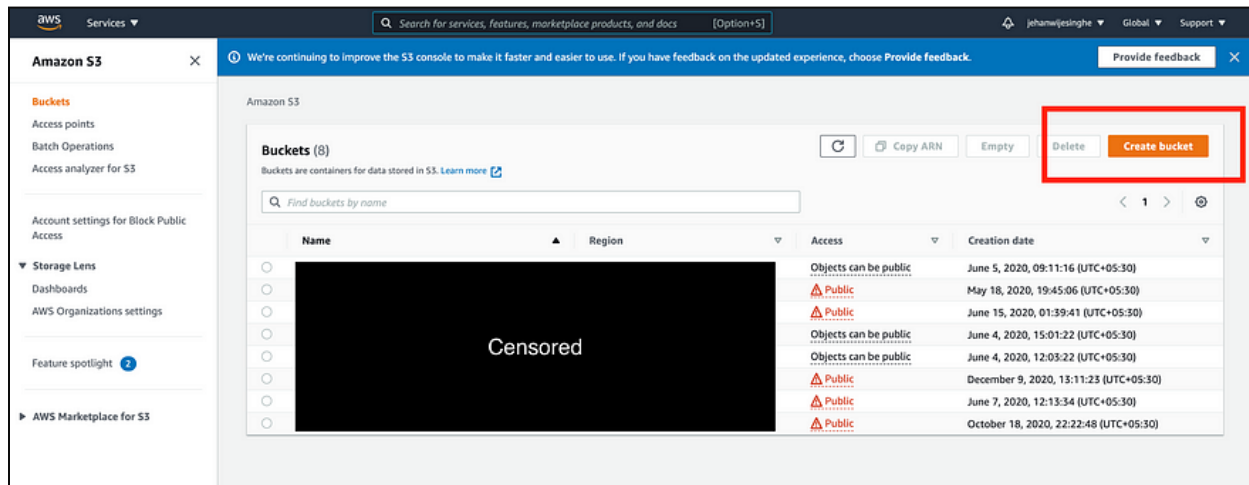
If you have a **Web App** developed using React, Angular or any other frontend framework, you have to generate a production build. How you generate a production build varies between frameworks (You can refer to the framework documentation to learn how this is done).

Either way, at the end of this step you should end up with a folder containing an **index.html** file and relevant style files, assets and scripts (if any). In this case, we have a very simple HTML site which displays the following.



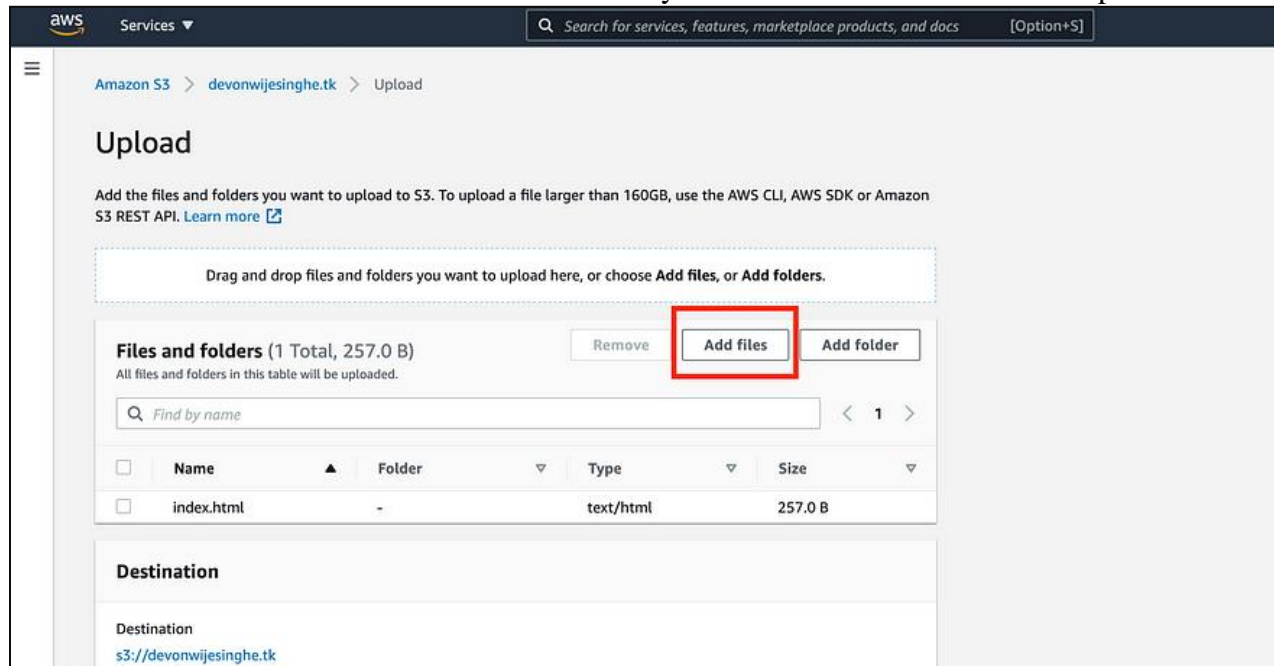
Step 2: Create a S3 bucket, Upload Website and Configure

Once you have your website or web-app ready, you can head over to the **AWS management console**, login and navigate to the “**Simple Storage Service (S3)**” service. We will be utilizing the Static Web Hosting feature of S3 to get the website/web-app up and running! As the first step, you need to **create a brand new S3 bucket**.

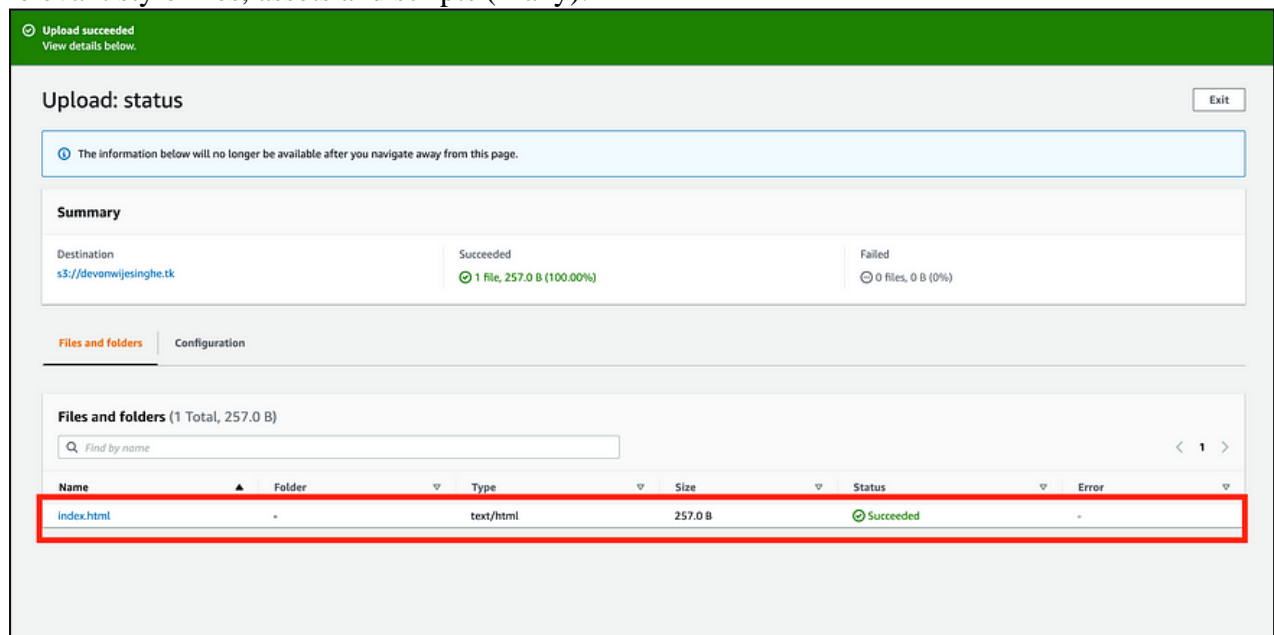


The bucket you create should have the **exact name as your domain**. In this case, the bucket name is “**devonwijesinghe.tk**” (Note that you **should not** put “www” in front of the bucket name). Also, make sure to **untick** the “**Block all public access**” checkbox.

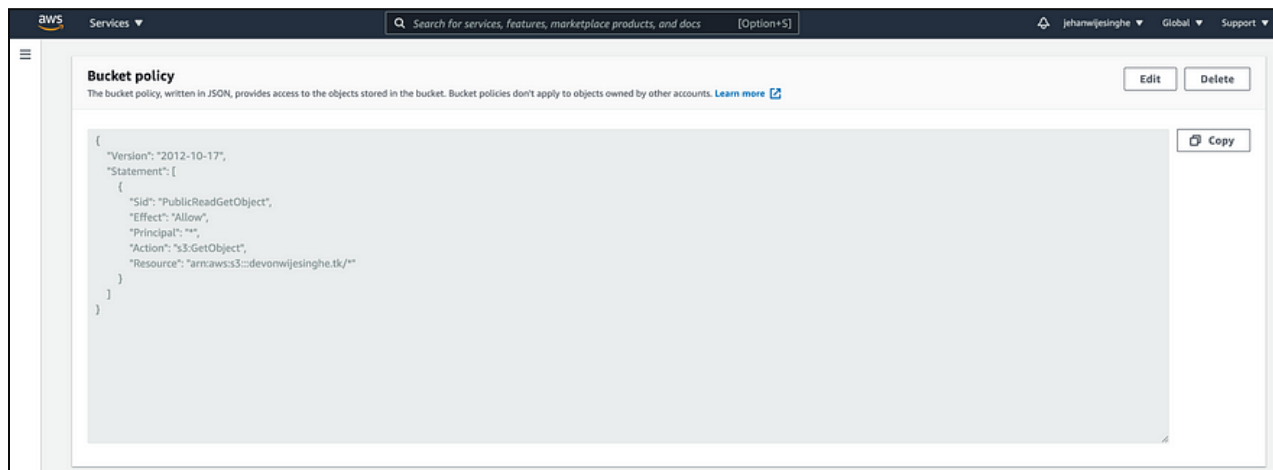
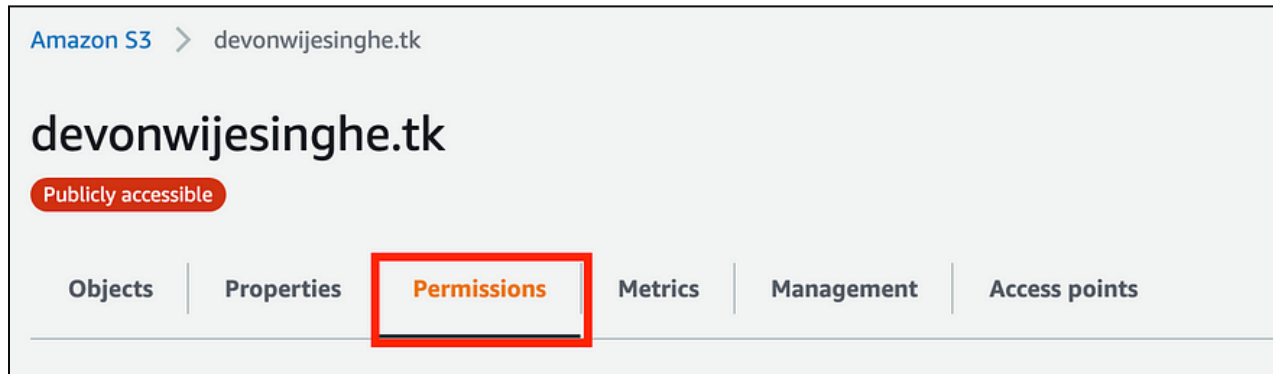
After the bucket is created, **navigate into the created bucket**. Then **upload** the website/web-app files into the S3 bucket. We have to add the files in your website folder and click on upload.



Once the upload is successful, the bucket root should contain the “index.html” file along with the relevant style files, assets and scripts (if any).



Afterwards, we need to allow **PublicReadGetObject** on the S3 bucket. This could simply be done by navigating to the “**Permissions**” tab and attaching the following policy to the S3 bucket.



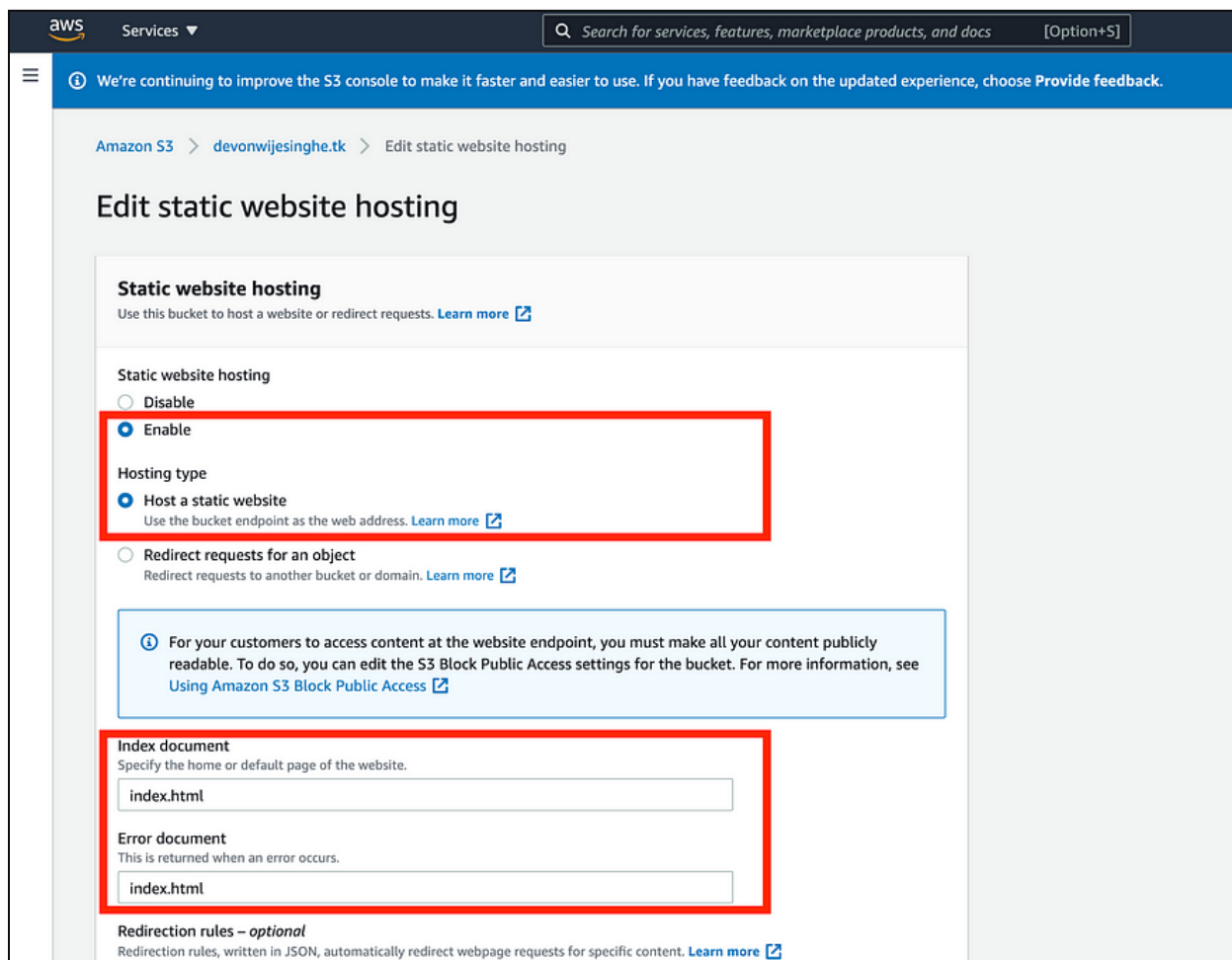
The policy can be **copied** from below and **changed by adding the bucket name**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::<YOUR BUCKET NAME>/*"
    }
  ]
}
```

Next, we have to enable static web hosting under the bucket “**Properties**” tab.



Select “**Host a static website**” option for the hosting type and for both Index Document and Error Document you should enter “**index.html**”.



After this step is complete, you should be able to find a link which can be used to access your website/web-app. **You can test it out now.**

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting

Enabled

Hosting type

Bucket hosting

Bucket website endpoint

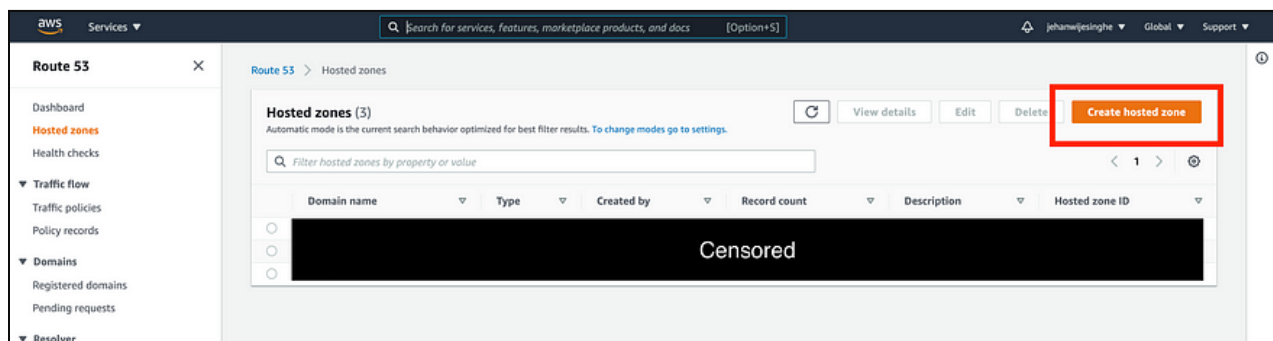
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)

<http://devonwijesinghe.tk.s3-website-ap-southeast-1.amazonaws.com>

Example: <http://devonwijesinghe.tk.s3-website-ap-southeast-1.amazonaws.com/>

Step 3: Create Hosted Zone in Route 53

As mentioned before in this article, you require a free or purchased **domain** to fully complete all the steps. Once you sort out your domain, you should go back to the AWS console and navigate to the “**Route 53**” service. You should then navigate to “**Hosted zones**” and **create a new hosted zone**.



You have to make sure to enter the **exact domain name** and select “**Public hosted zone**” for the type when creating the new hosted zone.

aws Services

Search for services, features, marketplace products, and docs [Option+S]

Route 53 > Hosted zones > Create hosted zone

Create hosted zone [Info](#)

Hosted zone configuration

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com, and its subdomains.

Domain name [Info](#)

This is the name of the domain that you want to route traffic for.

devonwijesinghe.tk

Valid characters: a-z, 0-9, ! * # \$ % & ' () * + , - / : ; < = > ? @ [\] ^ _ ` { | } . ~

Description - optional [Info](#)

This value lets you distinguish hosted zones that have the same name.

The hosted zone is used for...

The description can have up to 256 characters. 0/256

Type [Info](#)

The type indicates whether you want to route traffic on the internet or in an Amazon VPC.

☒ **Public hosted zone**

A public hosted zone determines how traffic is routed on the internet.

☐ **Private hosted zone**

A private hosted zone determines how traffic is routed within an Amazon VPC.

Tags [Info](#)

Apply tags to hosted zones to help organize and identify them.

No tags associated with the resource.

[Add tag](#)

You can add up to 50 more tags.

[Cancel](#) [Create hosted zone](#)

Once you have created the hosted zone, it should contain two records, **NS** (Name Server) record and **SOA** (Start of Authority) record. We will use the **NS** record in the next step.

devonwijesinghe.tk was successfully created.
Now you can create records in the hosted zone to specify how you want Route 53 to route traffic for your domain.

Route 53 > Hosted zones > devonwijesinghe.tk

devonwijesinghe.tk [Info](#)

[Delete zone](#) [Test record](#) [Configure query logging](#)

[Edit](#)

Hosted zone details

[Records \(2\)](#) [DNSSEC signing](#) [Hosted zone tags \(0\)](#)

Records (2) [Info](#)

Automatic mode is the current search behavior optimized for best filter results. [To change modes go to settings.](#)

[Filter records by property or value](#) [Type](#) [Routing policy](#) [Alias](#) [1](#) [Settings](#)

<input type="checkbox"/>	Record name	Type	Routing policy	Difference initiator	Value/Route traffic to
<input type="checkbox"/>	devonwijesinghe.tk	NS	Simple	-	ns-2000wzdnw-01.com. ns-7000wzdnw-02.net. ns-1000wzdnw-01.co.uk. ns-1000wzdnw-09.org.
<input type="checkbox"/>	devonwijesinghe.tk	SOA	Simple	-	ns-2000wzdnw-01.com. devonwijesinghe.tk hostmaster.amazon.com. 1 7200 900 1209600 86400

[Edit](#) [Delete record](#) [Import zone file](#) [Create record](#)

Step 4: Configure Domain Name Servers from Domain Admin Panel

Next, you should head over to the admin panel of your domain provider, in my case it is freenom.com. You should find the section which enables you to configure the ***name servers*** for the domain! For different domain providers, this would look a bit different.

Once you located the place to configure the **Nameservers** in the domain provider, you should add them like shown below

Managing devonwijesinghe.tk

[Information](#)[Upgrade](#)[Management Tools](#)[Manage Freenom DNS](#)

Nameservers

You can change where your domain points to here. Please be aware changes can take up to 24 hours to propagate.

☐ Use default nameservers (Freenom Nameservers)

☒ Use custom nameservers (enter below)

Nameserver 1

ns-200-1-1-37.com

Nameserver 2

ns-705-awsdns-123.net

Nameserver 3

ns-1000-awsdns-01.co.uk

Nameserver 4

ns-1000-awsdns-09.org

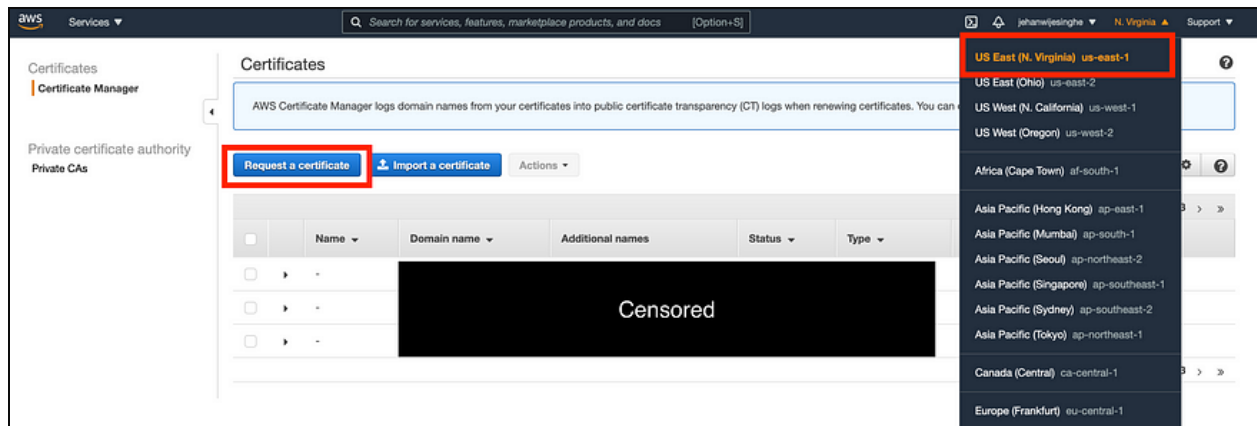
Nameserver 5

Change Nameservers

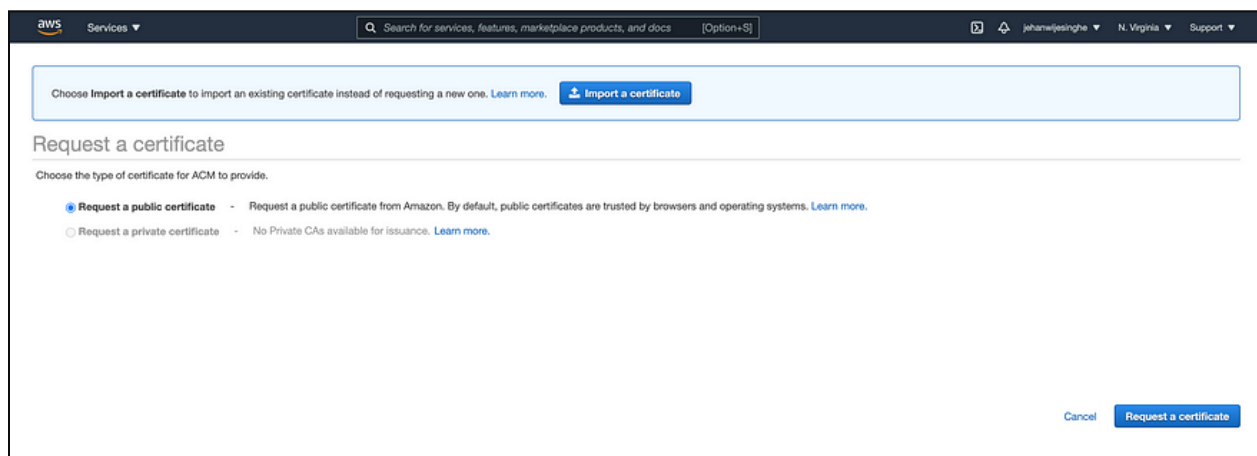
You should be aware that sometimes **Nameservers** takes a **couple of hours to Sync in**. So if your domain doesn't work at the end of this article, be patient and try again in a few hours.

Step 5: Create an SSL/TLS certificate using the Certificate Manager

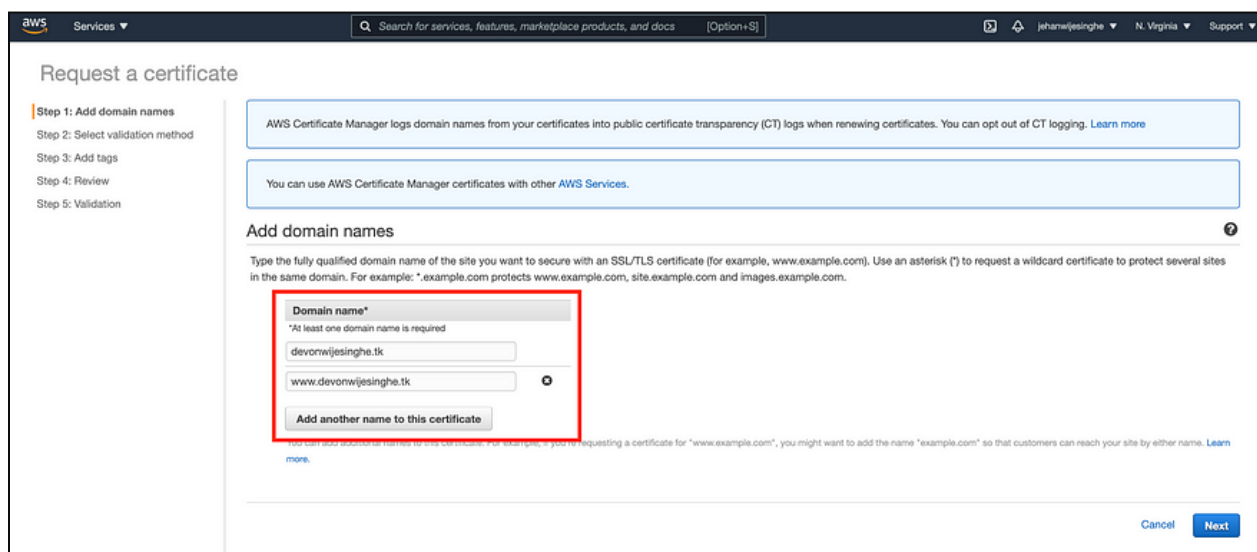
To serve your website via HTTPS, you required a SSL/TLS certificate. You can either get this certificate from the third party or create one from AWS itself using the **Certificate Manager**. Head over to the AWS console and navigate to the “**Certificate Manager**” service. In this step make sure to select the region as **us-east-1**. You should then click on “**Request a certificate**”.



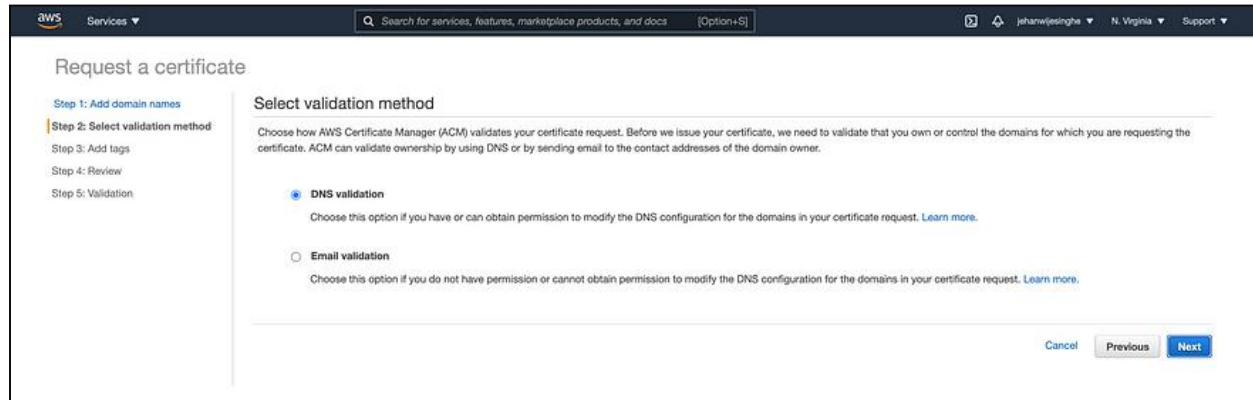
Proceed with “Request a public certificate” option.



In the next step, when adding the domain names for the certificate, make sure to add the exact domain name and also the **www** sub-domain.

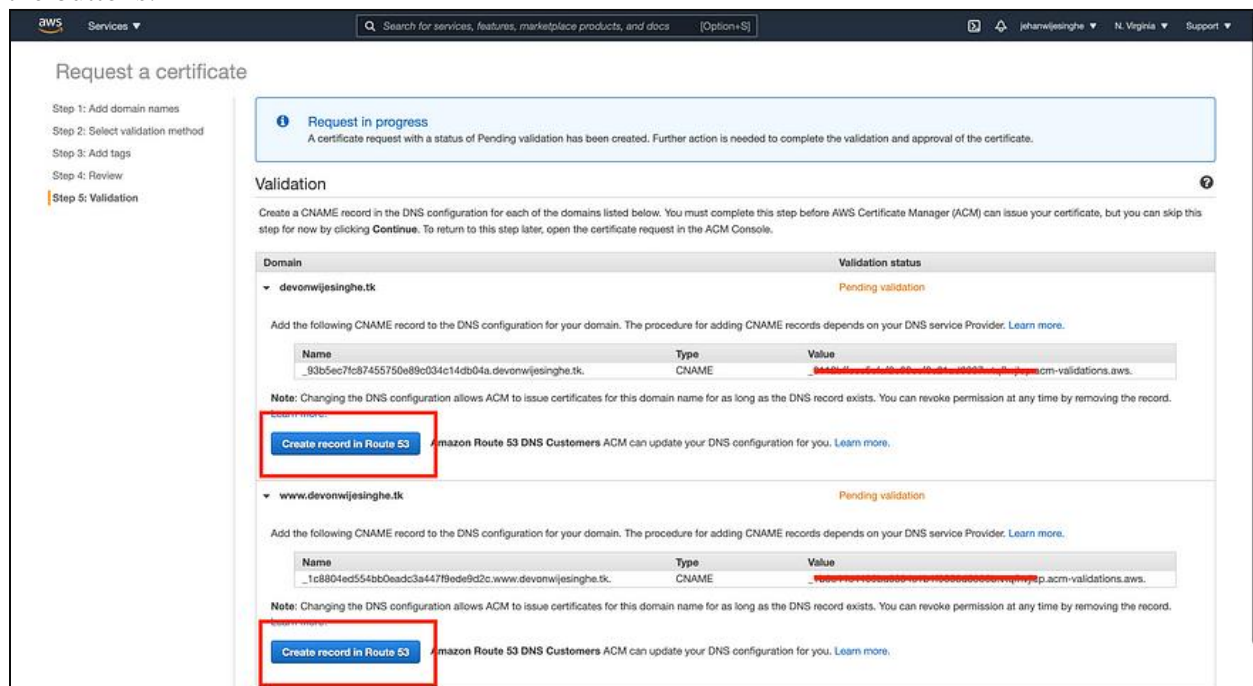


The step to add tags can be skipped for now as it is not mandatory. For the validation method, choose **“DNS validation”** and click on Next.



The screenshot shows the AWS Certificate Manager console. On the left, a sidebar lists the steps: Step 1: Add domain names, Step 2: Select validation method (highlighted), Step 3: Add tags, Step 4: Review, and Step 5: Validation. The main content area is titled 'Select validation method'. It explains that AWS Certificate Manager (ACM) validates the certificate request by using DNS or email. Two options are available: 'DNS validation' (selected with a radio button) and 'Email validation'. Below each option is a brief description and a 'Learn more' link. At the bottom right, there are 'Cancel', 'Previous', and 'Next' buttons.

You will see the following **“Create record in Route 53”** buttons as shown below. Click on both the buttons.



The screenshot shows the AWS Certificate Manager console at the 'Validation' step. A blue banner at the top indicates 'Request in progress'. Below this, instructions state that a CNAME record must be created in the DNS configuration for each domain. Two domains are listed: 'devonwjesinghe.tk' and 'www.devonwjesinghe.tk'. For each domain, a table shows the required CNAME record details (Name, Type, Value). Below each table, a 'Create record in Route 53' button is highlighted with a red box. A note explains that changing the DNS configuration allows ACM to issue certificates as long as the record exists.

Domain	Validation status	
devonwjesinghe.tk	Pending validation	
Add the following CNAME record to the DNS configuration for your domain. The procedure for adding CNAME records depends on your DNS service Provider. Learn more.		
Name	Type	Value
_93b5ec7fc87455750e89c034c14db04a.devonwjesinghe.tk	CNAME	93b5ec7fc87455750e89c034c14db04a.devonwjesinghe.tk.acm-validations.aws.
Note: Changing the DNS configuration allows ACM to issue certificates for this domain name for as long as the DNS record exists. You can revoke permission at any time by removing the record. Learn more.		
Create record in Route 53 Amazon Route 53 DNS Customers ACM can update your DNS configuration for you. Learn more.		
www.devonwjesinghe.tk	Pending validation	
Add the following CNAME record to the DNS configuration for your domain. The procedure for adding CNAME records depends on your DNS service Provider. Learn more.		
Name	Type	Value
_1c8804ed554bb0eadc3a447f9ede9d2c.www.devonwjesinghe.tk	CNAME	1c8804ed554bb0eadc3a447f9ede9d2c.www.devonwjesinghe.tk.acm-validations.aws.
Note: Changing the DNS configuration allows ACM to issue certificates for this domain name for as long as the DNS record exists. You can revoke permission at any time by removing the record. Learn more.		
Create record in Route 53 Amazon Route 53 DNS Customers ACM can update your DNS configuration for you. Learn more.		

To verify that the records are added to Route 53, you can head over to the hosted zone you created earlier and verify if **two CNAME records** are added for validation.



The screenshot shows the AWS Route 53 console. A table lists two CNAME records added for validation. The first record is for the domain 'devonwjesinghe.tk' and the second is for 'www.devonwjesinghe.tk'. Both records are of type 'CNAME' and have a 'Simple' TTL. The values are the respective domain names followed by '.acm-validations.aws.'.

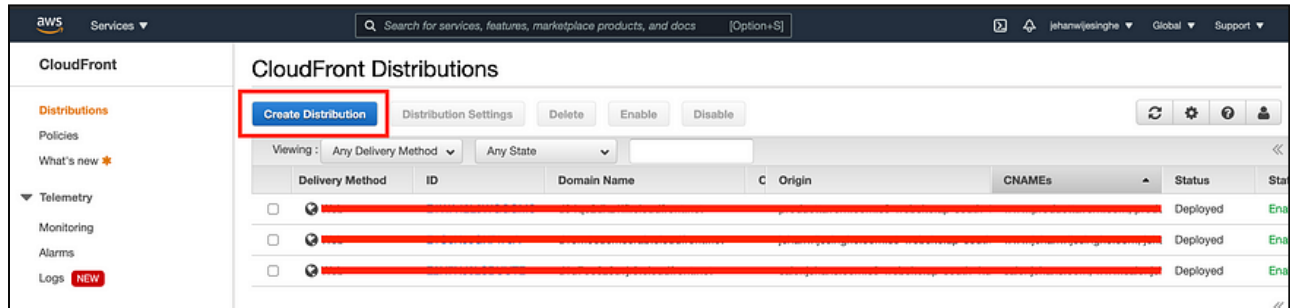
Domain	Type	TTL	Value
devonwjesinghe.tk	CNAME	Simple	93b5ec7fc87455750e89c034c14db04a.devonwjesinghe.tk.acm-validations.aws.
www.devonwjesinghe.tk	CNAME	Simple	1c8804ed554bb0eadc3a447f9ede9d2c.www.devonwjesinghe.tk.acm-validations.aws.

If everything goes smoothly, the status of the certificate should be **“Issued”** as shown below. Sometimes this may take a while.

Viewing certificates 1 to 4							
	Name	Domain name	Additional names	Status	Type	In use?	Renewal eligibility
<input type="checkbox"/>	-	devonwijesinghe.tk	www.devonwijesinghe.tk	Issued	Amazon Issued	No	Ineligible

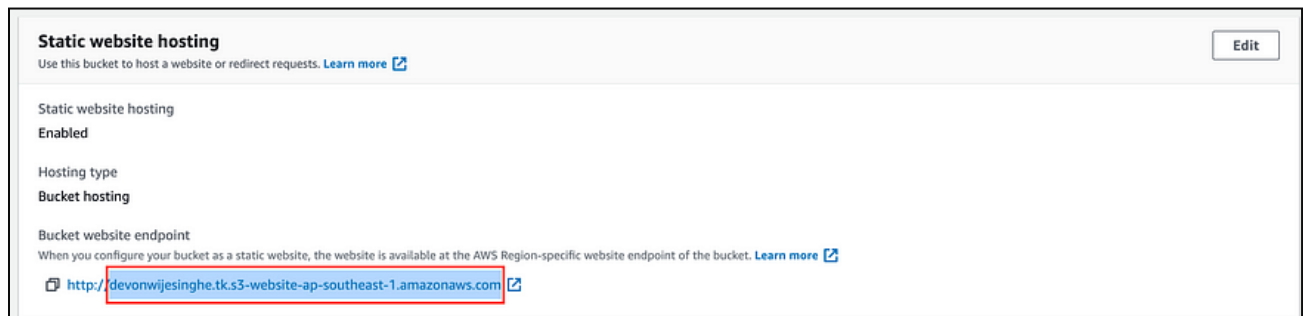
Step 6: Create a Cloud Front Distribution

Next, you have to navigate to the “**Cloud Front**” service in AWS and go to the “**Distributions**” section. Click on the “**Create Distribution**” button and complete the following steps.



There are so many configurations that you could do when creating a new Cloud Front Distribution. You can keep most of the configurations as default. We **only have to focus on** setting the **Origin Domain Name**, **View Protocol Policy** and **Alternate Domain Names (CNAMEs)** and **SSL certificate**.

For the **Origin Domain Name**, you should make sure to paste the URL you got from the **S3 static web hosting** when you configured the S3 bucket. Refer screenshot below.



It’s really tempting to select the Origin Domain Name from the drop down you get when you click on the textfield, but you should not do it. One more important thing you should do is select “**Redirect HTTP to HTTPS**” for “**View Protocol Policy**”. This will make sure that the HTTP traffic to your website will also be served via HTTPS.

Delivery Method	ID	Domain Name	Origin	CNAMEs	Status	State	Last Modified
Web	E11ED362JFW9B6	d34qpo65dtcaco.cloudfront.net	devonwijesinghe.tk.s3-website-ap-southe	www.devonwijesinghe.tk, devon	Deployed	Enabled	2021-01-28 22:16 UTC+5:30

You can make sure this step is successful by visiting the generated URL found under **Domain Name** in the created distribution. You should be able to see your website.

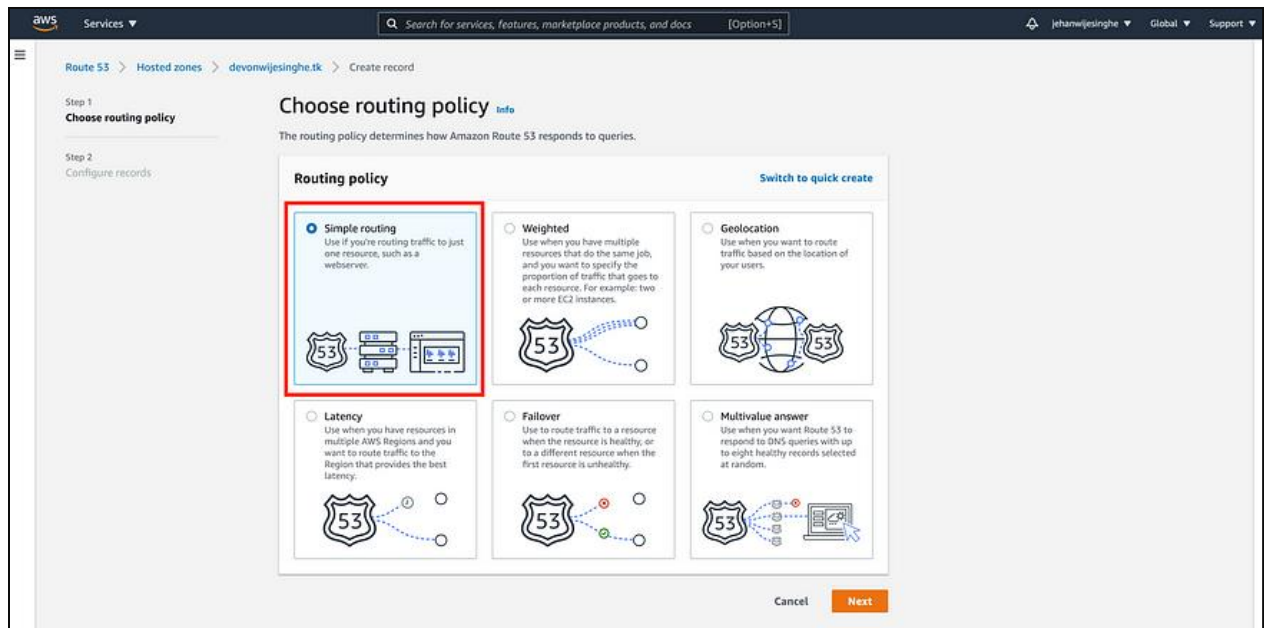
Example: <https://d34qpo65dtcaco.cloudfront.net/>

CloudFront Distributions > E11ED362JFW9B6	
<div>General Origins and Origin Groups Behaviors Error Pages Restrictions Invalidations Tags</div>	
<div>Edit</div>	
Distribution ID	E11ED362JFW9B6
ARN	arn:aws:cloudfront::102619622354:distribution/E11ED362JFW9B6
Log Prefix	-
Delivery Method	Web
Cookie Logging	Off
Distribution Status	Deployed
Comment	-
Price Class	Use All Edge Locations (Best Performance)
AWS WAF Web ACL	-
State	Enabled
Alternate Domain Names (CNAMEs)	www.devonwijesinghe.tk devonwijesinghe.tk
SSL Certificate	devonwijesinghe.tk (5d653528-9b25-499b-b53a-c4ee2398d854)
Domain Name	d34qpo65dtcaco.cloudfront.net
Custom SSL Client Support	Clients that Support Server Name Indication (SNI) - (Recommended)
Security Policy	TLSv1.2_2019
Supported HTTP Versions	HTTP/2, HTTP/1.1, HTTP/1.0
IPv6	Enabled
Default Root Object	-
Last Modified	2021-01-28 22:16 UTC+5:30
Log Bucket	-

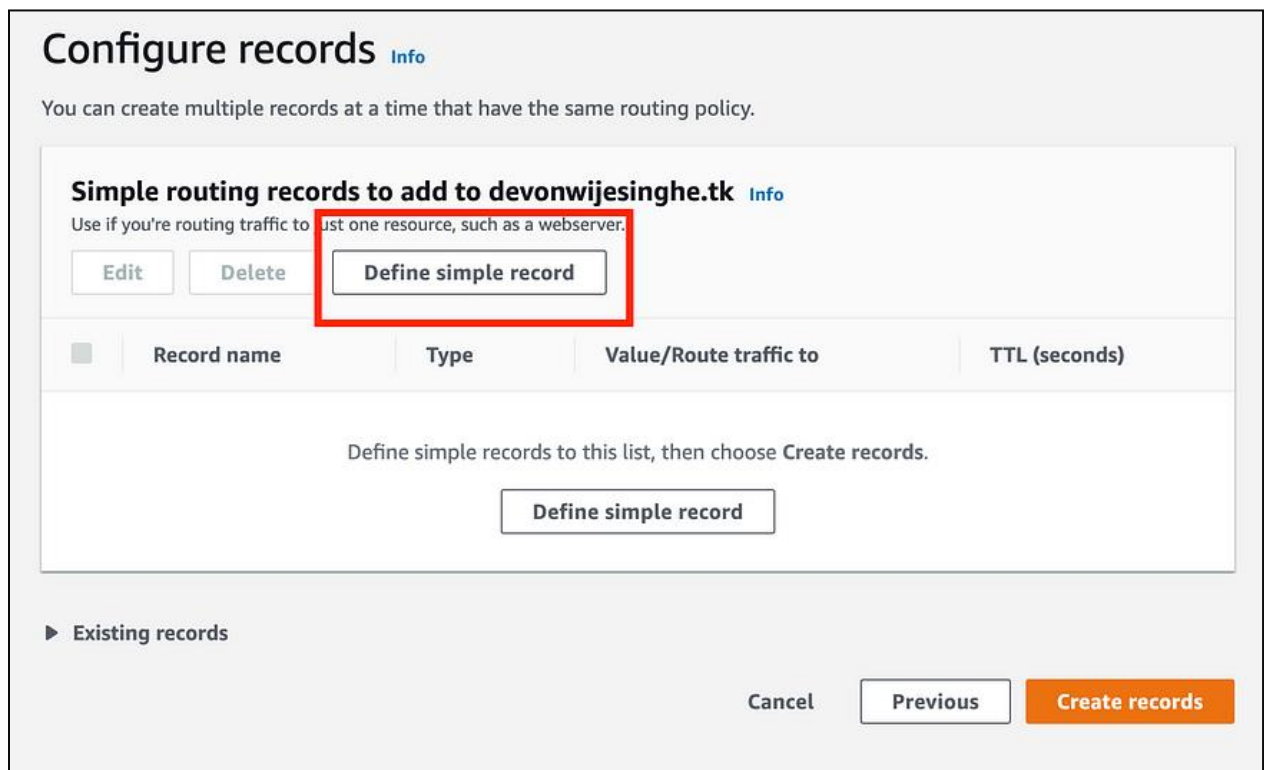
Step 7: Create Records in Route 53 to direct to CloudFront

Just one more final step! You have to head back to “**Route 53**” service in AWS and create two more records in the hosted zone that you created earlier. Click on the “**Create Record**” button and complete the following steps.

Select “**Simple routing**” as the routing policy and click on Next.



Next, click on the “Define simple record” button.



For the first simple record, leave the “Record name” empty and for “Value/Route traffic to” select “Alias to CloudFront distribution” and select or paste the Domain Name of the CloudFront distribution you created earlier.

In this case this is d34qpo65dtcaco.cloudfront.net. The “Record type” should be an “A record”.

Define simple record Leave this empty

Record name
To route traffic to a subdomain, enter the subdomain name. For example, to route traffic to `blog.example.com`, enter `blog`. If you leave this field blank, the default record name is the name of the hosted zone.

`blog` `.devonwijesinghe.tk`

Value/Route traffic to
The option that you choose determines how Route 53 responds to DNS queries. For most options, you specify where you want to send incoming traffic.

Alias to CloudFront distribution

US East (N. Virginia)

`d54apo65dtkaco.cloudfront.net`

Record type
The DNS type of the record determines the format of the value that Route 53 returns in response to DNS queries.

A - Routes traffic to an IPv4 address and some AWS resources

Evaluate target health
Select Yes if you want Route 53 to use this record to respond to DNS queries only if the specified AWS resource is healthy.

☐ No

Cancel **Define simple record**

After the previous record is defined, click on the **“Define simple record”** button once again. This time, for the **“Record name”**, enter **“www”** and for **“Value/Route traffic to”** select **“Alias to another record in this hosted zone”** and type in your main domain. In this case it is devonwijesinghe. The **“Record type”** of this record should also be an **“A record”**.

Define simple record www

Record name
To route traffic to a subdomain, enter the subdomain name. For example, to route traffic to `blog.example.com`, enter `blog`. If you leave this field blank, the default record name is the name of the hosted zone.

`www` `.devonwijesinghe.tk`

Value/Route traffic to
The option that you choose determines how Route 53 responds to DNS queries. For most options, you specify where you want to send incoming traffic.

Alias to another record in this hosted zone

US East (N. Virginia)

`devonwijesinghe.tk`

Record type
The DNS type of the record determines the format of the value that Route 53 returns in response to DNS queries.

A - Routes traffic to an IPv4 address and some AWS resources

Evaluate target health
Select Yes if you want Route 53 to use this record to respond to DNS queries only if the specified AWS resource is healthy.

☒ Yes

Cancel **Define simple record**

Your final record set should look like this once you are done defining the simple records

Records (6) Info					
Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.					
<input type="text" value="Filter records by property or value"/> Type Routing policy Alias < 1 > ⚙					
<input type="checkbox"/>	Record name	Type	Routing policy	Differentiator	Value/Route traffic to
<input type="checkbox"/>	devonwijesinghe.tk	A	Simple	-	d34qpo65dtcaco.cloudfront.net.
<input type="checkbox"/>	devonwijesinghe.tk	NS	Simple	-	
<input type="checkbox"/>	devonwijesinghe.tk	SOA	Simple	-	
<input type="checkbox"/>	_93b5ec7fc87455750e89c034c14db04a.devonwijesinghe.tk	CNAME	Simple	-	
<input type="checkbox"/>	www.devonwijesinghe.tk	A	Simple	-	devonwijesinghe.tk.
<input type="checkbox"/>	_1c8804ed554bb0eadc3a447f9ede9d2c.www.devonwijesinghe.tk	CNAME	Simple	-	

```
gradle
<no devices>
main.dart
Pixel 3 XL API 22
pubspec.yaml
home.dart
app/build.gradle
android/build.gradle
google-services.json

android {
    compileSdkVersion 28

    sourceSets {
        main.java.srcDirs += 'src/main/kotlin'
    }

    lintOptions {
        disable 'InvalidPackage'
    }

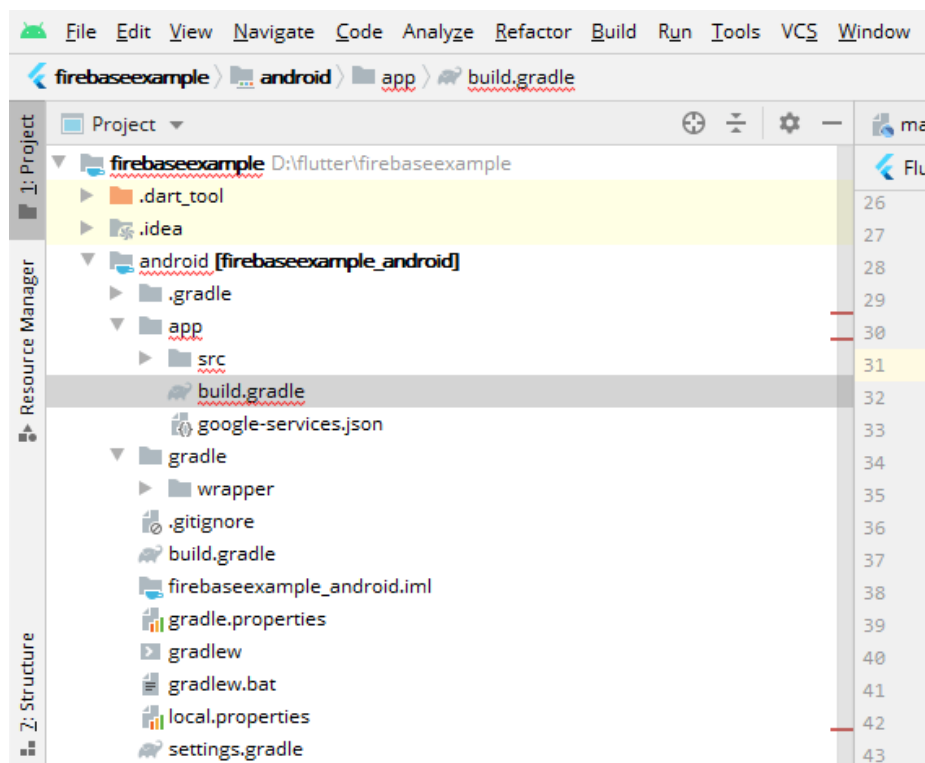
    defaultConfig {
        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id...)
        applicationId "com.firebaseexample"
        minSdkVersion 16
        targetSdkVersion 28
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }

    buildTypes {
        release {
            // TODO: Add your own signing config for the release build.
            // Signing with the debug keys for now, so `flutter run --release` works.
            signingConfig signingConfigs.debug
        }
    }
}
```

You can find your package name in the app-level – build.gradle file. Click register app after completing this step.

[illegible]

Download the `google-services.json` file and paste it into the `android/app` folder.



Paste classpath 'com.google.gms:google-services:4.3.3' in the project level – build.gradle file.

```
dependencies {  
  classpath 'com.android.tools.build:gradle:3.5.0'  
  classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"  
  classpath 'com.google.gms:google-services:4.3.3'  
}
```

Navigate to the app-level – build.gradle file and add apply plugin: 'com.google.gms.google-services' as shown below.

```
apply plugin: 'com.android.application'  
apply plugin: 'kotlin-android'  
apply from: "$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"  
apply plugin: 'com.google.gms.google-services'
```

After the project is created successfully, navigate back to the Firebase console's realtime database section, and create a new database.

In the rules section, ensure that you click on test mode rather than the locked mode. This allows us to read and write the data without authentication. Note that these rules should not be used in a production application since anyone can access the data.

```
{  
  "rules": {  
    ".read": "now < 1610053200000", // 2021-1-8  
    ".write": "now < 1610053200000", // 2021-1-8  
  }  
}
```

You can find more information about changing the rules above for a production application [here](#). We have successfully set up Firebase and the Flutter project. Let's design the UI.

Building the UI

The app will have a simple user interface. It will allow a user to input a word or sentence and click on a button to save it to the Firebase database.

Ensure that you have the following imports at the top of the Home page.

```
import 'package:flutter/material.dart';  
import 'package:firebase_database/firebase_database.dart';  
import 'package:firebase_core/firebase_core.dart';
```

- package:flutter/material.dart - This import allows you to access Flutter's built-in widgets and other functionalities.
- package:firebase_database/firebase_database.dart - This import helps access the Firebase Realtime database features.

- package:firebase_core/firebase_core.dart - This dependency enables you to connect to different Firebase products.

Storing data

We will initialize a Firebase databaseReference object and use it to store data.

```
final databaseRef = FirebaseDatabase._instance_.reference(); //database reference object

void addData(String data) {
  databaseRef.push().set({'name': data, 'comment': 'A good season'});
}
```

The addData function stores the user input along with a string ('comment': 'A good season').

Essentially, we can add any object to the database.

We'll call the addData function when the RaisedButton is clicked.

```
onPressed: () {
  addData(textcontroller.text);
  //call method flutter upload
})),
```

We'll access the user input using a textcontroller. This element allows us to listen for changes in a TextField.

We need to include the textcontroller in the HomePageState class.

```
class _HomePageState extends State<HomePage> {
  final textcontroller = TextEditingController();
```

The textcontroller is the added to the TextField widget, as shown below.

```
TextField(controller: textcontroller),
```

Please note that we need to use a FutureBuilder for async operations. This class allows the application to retrieve results once the network operation is completed.

This class also helps us initialize the Firebase app. You can learn more about FutureBuilder from [here](#).

```
class _HomePageState extends State<HomePage> {
  final Future<FirebaseApp> _future = Firebase.initializeApp();
```

_future is then added to the FutureBuilder in the Scaffold. This operation is illustrated below. As noted, FutureBuilder helps in awaiting long-running operations.

```
return Scaffold(
  appBar: AppBar(
    title: Text("Firebase Demo"),),
  body: FutureBuilder(
```

```
future: _future,  
builder: (context, snapshot) {  
  if (snapshot.hasError) {  
    return Text(snapshot.error.toString());  
  } else {  
    return Container(  

```

Retrieving data

For simplicity, we will retrieve and print out the data in the console. This is shown below.

```
void printFirebase(){  
  databaseRef.once().then((DataSnapshot snapshot) {  
    print('Data : ${snapshot.value}');  
  });  
}
```

We'll call the function above in the `_HomePageState` class's build method. Here is the complete code for the `HomePage` class.

CODE:

```
class HomePage extends StatefulWidget {  
  @override  
  _HomePageState createState() => _HomePageState();  
}  
  
class _HomePageState extends State<HomePage> {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(); // add scaffold here  
  }  
}  
  
Scaffold(  
  appBar: AppBar(title: Text("Firebase Demo")),  
  body: Container(  
    child: Column(  
      children: <Widget>[  
        SizedBox(height: 250.0),  
        Padding(  
          padding: EdgeInsets.all(10.0),  
          child: TextField(),  
        ),  
        SizedBox(height: 30.0),  
        Center(  
          child: RaisedButton(  
            color: Colors.pinkAccent,  
            child: Text("Save to Database"),  

```

```

        onPressed: () {
          //call method flutter upload
        }
      ),
    ],
  ),
),);

```

```

import 'package:flutter/material.dart';
import 'home.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,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: HomePage(),
    );
  }
}

import 'package:flutter/material.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:firebase_core/firebase_core.dart';

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  final textcontroller = TextEditingController();
  final databaseRef = FirebaseDatabase.instance.reference();

```

```

final Future<FirebaseApp> _future = Firebase.initializeApp();

void addData(String data) {
  databaseRef.push().set({'name': data, 'comment': 'A good season'});
}

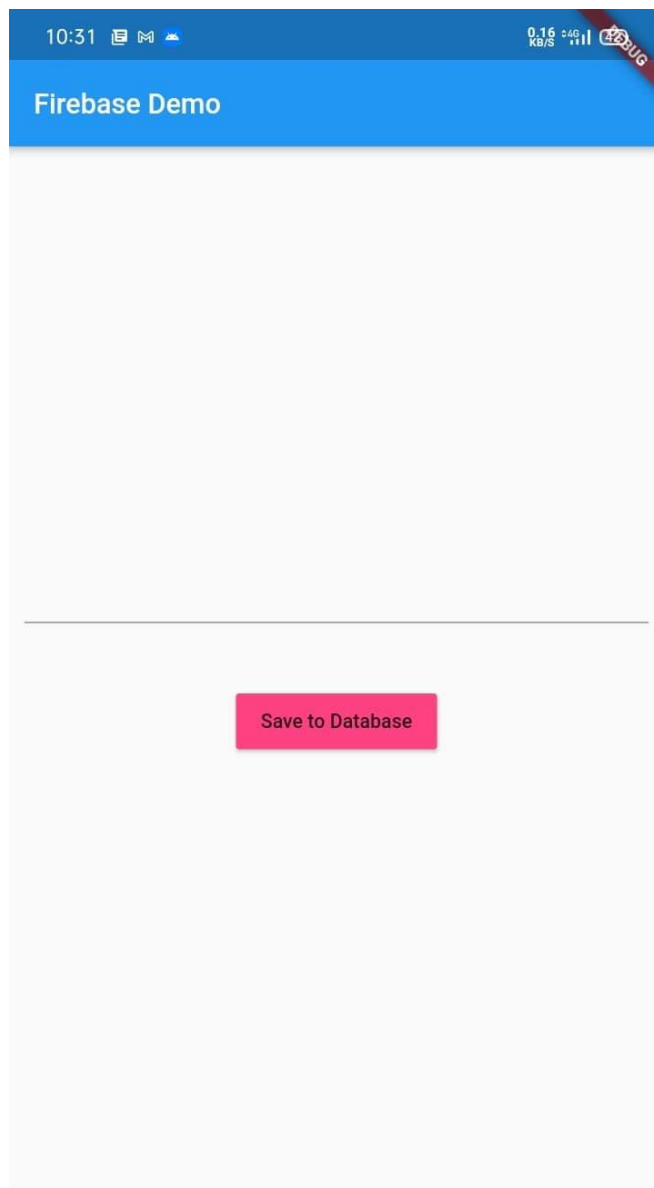
void printFirebase(){
  databaseRef.once().then((DataSnapshot snapshot) {
    print('Data : ${snapshot.value}');
  });
}

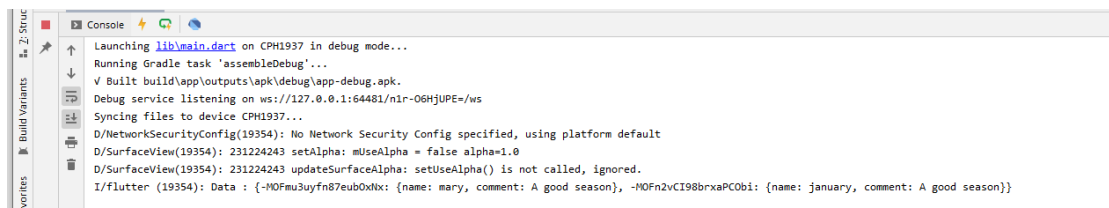
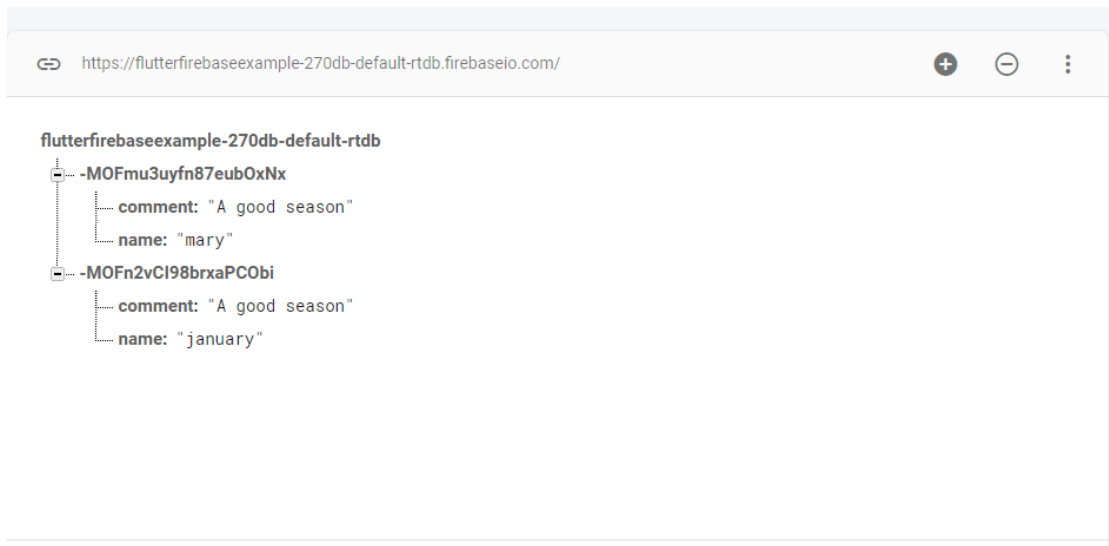
@override
Widget build(BuildContext context) {
  printFirebase();
  return Scaffold(
    appBar: AppBar(
      title: Text("Firebase Demo"),
    ),
    body: FutureBuilder(
      future: _future,
      builder: (context, snapshot) {
        if (snapshot.hasError) {
          return Text(snapshot.error.toString());
        } else {
          return Container(
            child: Column(
              children: <Widget>[
                SizedBox(height: 250.0),
                Padding(
                  padding: EdgeInsets.all(10.0),
                  child: TextField(
                    controller: textcontroller,
                  ),
                ),
                SizedBox(height: 30.0),
                Center(
                  child: RaisedButton(
                    color: Colors.pinkAccent,
                    child: Text("Save to Database"),
                    onPressed: () {
                      addData(textcontroller.text);
                      //call method flutter upload
                    }
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    ),
  ),

```

```
    ],  
    ),  
    );  
  }  
  }  
  ),  
  );  
}  
}
```

OUTPUT:





MAD Mini Project

Name: Pratyush Lokhande

Roll No.: 23

College ID: 202044105

Topic: Netflix clone

Problem statement

Flutter SDK is an open-source software development kit for building beautiful UI which is natively compiled. We utilized flutter framework and its library to build Netflix clone application which allows people to watch all their favourite tv shows, movies, ect. On the go anytime without theatrical experiences and rush hours. For that reason, we delimit the feature set by the following:

- The user can play any movie or show of his/her choice
- They can skip and increase the speed of the show.
- Get a featured list of famous shows as well.

Code:

Main Dart:

```
import 'package:flutter/material.dart';
import 'package:flutter_netflix_ui_redesign/screens/home_screen.dart';
import 'package:flutter_netflix_ui_redesign/screens/login.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 Netflix',
      debugShowCheckedModeBanner: false,
      home: OnboardingScreen(),
    );
  }
}
```

Login.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_netflix_ui_redesign/main.dart';
import 'package:flutter_netflix_ui_redesign/screens/home_screen.dart';
// import 'package:provider/provider.dart';

class OnboardingScreen extends StatefulWidget {
  // const OnboardingScreen({Key? key}) : super(key: key);

  @override
  State<OnboardingScreen> createState() => _OnboardingScreenState();
}

class _OnboardingScreenState extends State<OnboardingScreen> {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  int _selectedIndex = 0;

  @override
  void initState() {
    super.initState();
  }

  @override
  void dispose() {
    super.dispose();
  }

  Widget _renderSignIn() {
    return Container(
      padding: const EdgeInsets.fromLTRB(60, 0, 60, 0),
      color: Colors.black,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          Center(
```



```
      child: Image.asset('assets/images/netflix_logo1.png', width: 200),
    ),
    const SizedBox(height: 60),
    TextField(
      controller: _emailController,
      autofocus: false,
      autocorrect: false,
      enableSuggestions: false,
      decoration: const InputDecoration(
        filled: true,
        fillColor: Color.fromARGB(255, 231, 231, 231),
        labelText: 'Email',
        floatingLabelStyle: TextStyle(color: Colors.black),
        focusedBorder: InputBorder.none,
        border: InputBorder.none,
      ),
    ),
    Container(
      height: 0.1,
      color: Colors.black,
    ),
    TextField(
      controller: _passwordController,
      obscureText: true,
      autofocus: false,
      autocorrect: false,
      enableSuggestions: false,
      decoration: const InputDecoration(
        filled: true,
        fillColor: Colors.grey,
        labelText: 'Password',
        floatingLabelStyle: TextStyle(color: Colors.black),
        focusedBorder: InputBorder.none,
        border: InputBorder.none,
      ),
    ),
    const SizedBox(height: 20.0),
    SizedBox(
      width: double.infinity,
      child: OutlinedButton(
```

```

        style: OutlinedButton.styleFrom(
          padding: const EdgeInsets.fromLTRB(0, 10, 0, 10),
          side: const BorderSide(width: 1.0, color: Colors.grey),
        ),
        child: const Text(
          "Sign in",
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.bold,
            fontSize: 22.0),
        ),
        onPressed: () async {
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(builder: (context) => HomeScreen()),
          );
        })),
    ),
    const SizedBox(height: 40.0),
    MaterialButton(
      child: const Text(
        "Don't have an account? Sign up",
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {
        setState(() {
          _selectedIndex = 1;
        });
      },
    ),
    const SizedBox(height: 10.0),
    MaterialButton(
      child: const Text(
        "Forgot your password?",
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {},
    ),
  ],
),

```

```
);  
}
```

```
Widget _renderSignUp() {  
  return Container(  
    padding: const EdgeInsets.fromLTRB(40, 0, 40, 0),  
    color: Colors.black,  
    child: Column(  
      crossAxisAlignment: CrossAxisAlignment.center,  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: [  
        Center(  
          child: Image.asset('assets/images/netflix_logo1.png', width: 200),  
        ),  
        const SizedBox(height: 60),  
        TextField(  
          controller: _nameController,  
          autofocus: false,  
          autocorrect: false,  
          enableSuggestions: false,  
          decoration: const InputDecoration(  
            filled: true,  
            fillColor: Colors.grey,  
            labelText: 'Your name',  
            floatingLabelStyle: TextStyle(color: Colors.black),  
            focusedBorder: InputBorder.none,  
            border: InputBorder.none,  
          ),  
        ),  
        Container(  
          height: 0.1,  
          color: Colors.black,  
        ),  
        TextField(  
          controller: _emailController,  
          autofocus: false,  
          autocorrect: false,  
          enableSuggestions: false,  
          decoration: const InputDecoration(  
            filled: true,
```

```

        fillColor: Colors.grey,
        labelText: 'Email address',
        floatingLabelStyle: TextStyle(color: Colors.black),
        focusedBorder: InputBorder.none,
        border: InputBorder.none,
    ),
),
Container(
    height: 0.1,
    color: Colors.black,
),
TextField(
    controller: _passwordController,
    obscureText: true,
    autofocus: false,
    autocorrect: false,
    enableSuggestions: false,
    decoration: const InputDecoration(
        filled: true,
        fillColor: Colors.grey,
        labelText: 'Password',
        floatingLabelStyle: TextStyle(color: Colors.black),
        focusedBorder: InputBorder.none,
        border: InputBorder.none,
    ),
),
const SizedBox(height: 20.0),
SizedBox(
    width: double.infinity,
    child: OutlinedButton(
        style: OutlinedButton.styleFrom(
            padding: const EdgeInsets.fromLTRB(0, 10, 0, 10),
            side: const BorderSide(width: 1.0, color: Colors.grey),
        ),
        child: const Text(
            "Sign up",
            style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.bold,
                fontSize: 22.0),
        ),
    ),
),

```

```

    ),
    onPressed: () async {
      setState(() {
        _selectedIndex = 0;
      });
    },
  ),
),
const SizedBox(height: 10.0),
MaterialButton(
  child: const Text(
    "Forgot your password?",
    style: TextStyle(color: Colors.white),
  ),
  onPressed: () {},
),
const SizedBox(height: 40.0),
MaterialButton(
  child: const Text(
    "Already have an account? Sign in",
    style: TextStyle(color: Colors.white),
  ),
  onPressed: () {
    setState(() {
      _selectedIndex = 0;
    });
  },
),
],
),
);
}

```

```

@override
Widget build(BuildContext context) {
  final Size screenSize = MediaQuery.of(context).size;

  // final current = context.watch<AccountProvider>().current;

  // _emailController.text = current?.email ?? ""

```

```

return Scaffold(
  extendBodyBehindAppBar: true,
  body: IndexedStack(
    index: _selectedIndex,
    children: [
      _renderSignIn(),
      _renderSignUp(),
    ],
  ));
}
}

```

Home_screen.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_netflix_ui_redesign/models/movie_model.dart';
import 'package:flutter_netflix_ui_redesign/screens/movie_screen.dart';
import 'package:flutter_netflix_ui_redesign/widgets/content_scroll.dart';

class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  PageController _pageController;

  @override
  void initState() {
    super.initState();
    _pageController = PageController(initialPage: 1, viewportFraction: 0.8);
  }

  _movieSelector(int index) {
    return AnimatedBuilder(
      animation: _pageController,
      builder: (BuildContext context, Widget widget) {
        double value = 1;
        if (_pageController.position.haveDimensions) {

```

```
    value = _pageController.page - index;
    value = (1 - (value.abs() * 0.3) + 0.06).clamp(0.0, 1.0);
  }
  return Center(
    child: SizedBox(
      height: Curves.easeInOut.transform(value) * 270.0,
      width: Curves.easeInOut.transform(value) * 400.0,
      child: widget,
    ),
  );
},
child: GestureDetector(
  onTap: () => Navigator.push(
    context,
    MaterialPageRoute(
      builder: (_) => MovieScreen(movie: movies[index]),
    ),
  ),
  child: Stack(
    children: <Widget>[
      Center(
        child: Container(
          margin: EdgeInsets.symmetric(horizontal: 10.0, vertical: 20.0),
          decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(10.0),
            boxShadow: [
              BoxShadow(
                color: Colors.black54,
                offset: Offset(0.0, 4.0),
                blurRadius: 10.0,
              ),
            ],
          ),
        ),
      ),
      child: Center(
        child: Hero(
          tag: movies[index].imageUrl,
          child: ClipRRect(
            borderRadius: BorderRadius.circular(10.0),
            child: Image(
              image: AssetImage(movies[index].imageUrl),
```

```

        height: 220.0,
        fit: BoxFit.cover,
      ),
    ),
  ),
),
),
),
Positioned(
  left: 30.0,
  bottom: 40.0,
  child: Container(
    width: 250.0,
    child: Text(
      movies[index].title.toUpperCase(),
      style: TextStyle(
        color: Colors.white,
        fontSize: 20.0,
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
),
),
),
],
),
),
);
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      backgroundColor: Colors.black,
      elevation: -0.0,
      title: Image(
        image: AssetImage('assets/images/netflix_logo.png'),
      ),
      leading: IconButton(

```



```

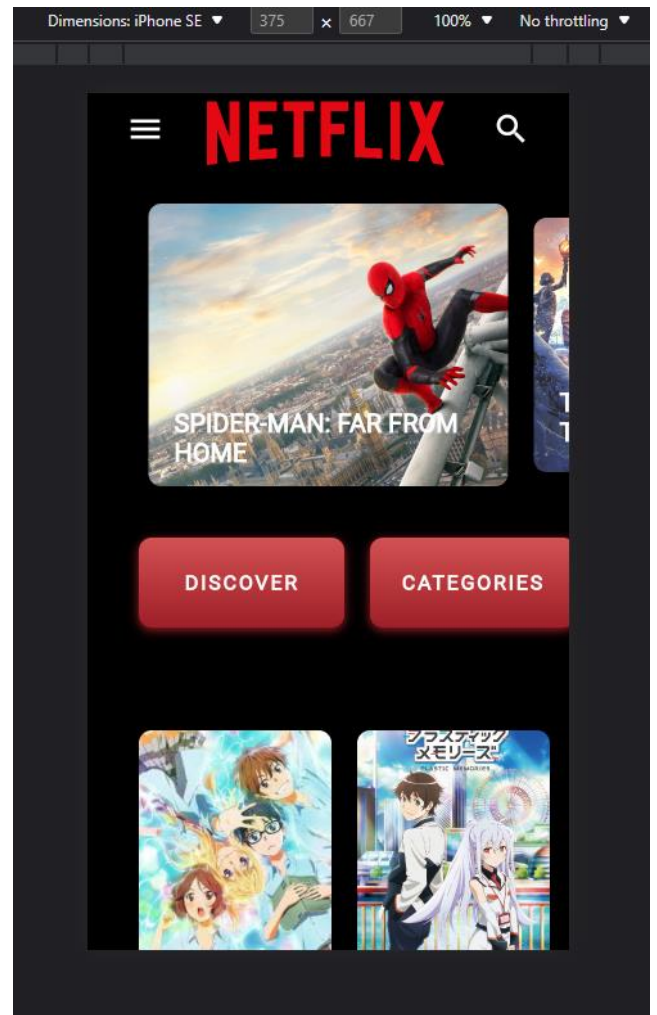
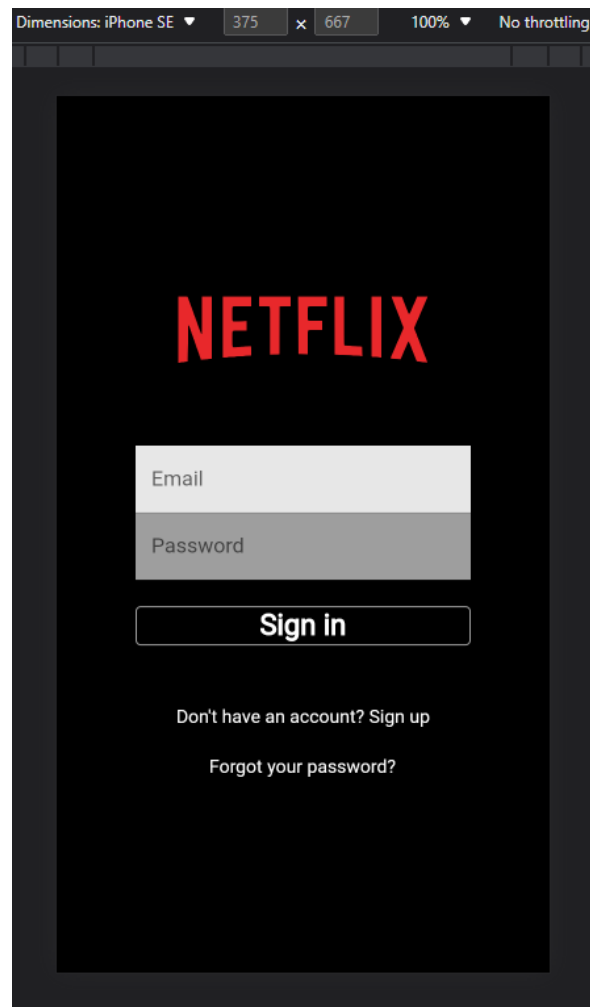
padding: EdgeInsets.only(left: 30.0),
onPressed: () => print('Menu'),
icon: Icon(Icons.menu),
iconSize: 30.0,
color: Colors.white,
),
actions: <Widget>[
  IconButton(
    padding: EdgeInsets.only(right: 30.0),
    onPressed: () => print('Search'),
    icon: Icon(Icons.search),
    iconSize: 30.0,
    color: Colors.white,
  ),
],
),
body: ListView(
  children: <Widget>[
    Container(
      height: 280.0,
      width: double.infinity,
      child: PageView.builder(
        controller: _pageController,
        itemCount: movies.length,
        itemBuilder: (BuildContext context, int index) {
          return _movieSelector(index);
        },
      ),
    ),
    Container(
      height: 90.0,
      child: ListView.builder(
        padding: EdgeInsets.symmetric(horizontal: 30.0),
        scrollDirection: Axis.horizontal,
        itemCount: labels.length,
        itemBuilder: (BuildContext context, int index) {
          return Container(
            margin: EdgeInsets.all(10.0),
            width: 160.0,
            decoration: BoxDecoration(

```

```
bordersRadius: BorderRadius.circular(10.0),  
gradient: LinearGradient(  
    begin: Alignment.topCenter,  
    end: Alignment.bottomCenter,  
    colors: [  
        Color(0xFFD45253),  
        Color(0xFF9E1F28),  
    ],  
),  
boxShadow: [  
    BoxShadow(  
        color: Color(0xFF9E1F28),  
        offset: Offset(0.0, 2.0),  
        blurRadius: 6.0,  
    ),  
],  
),  
child: Center(  
    child: Text(  
        labels[index].toUpperCase(),  
        style: TextStyle(  
            color: Colors.white,  
            fontSize: 16.0,  
            fontWeight: FontWeight.bold,  
            letterSpacing: 1.8,  
        ),  
    ),  
),  
);  
},  
),  
),  
),  
SizedBox(height: 20.0),  
ContentScroll(  
    images: myList,  
    title: 'My List',  
    imageHeight: 250.0,  
    imageWidth: 150.0,  
),  
SizedBox(height: 10.0),
```

```
ContentScroll(  
  images: popular,  
  title: 'Popular',  
  imageHeight: 250.0,  
  imageWidth: 150.0,  
),  
],  
),  
);  
}  
}
```

Output:



Dimensions: iPhone SE ▾

375

x

667

100% ▾

No throttling ▾



NETFLIX



SPIDER-MAN: FAR FROM HOME

Fantasy, Sci-fi



Year

2018

Country

USA

Length

129 min

Our friendly neighborhood Super Hero decides to join his best friends Ned, MJ, and the rest of the gang on a European vacation. However, Peter's plan to leave super heroics behind for a