Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

CERTIFICATE

This is to certify that **Divesh Bhuvan Punjabi** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Title:Budgettracker **Roll No. 46**

Year/Sem/Class **:** D15A **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Jewani.

Email **:** kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

PO1) Basic Engineering knowledge: An ability to apply the fundamental knowledge in mathematics, science and engineering to solve problems in Computer engineering.

PO2) Problem Analysis: Identify, formulate, research literature and analyze computer engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and computer engineering and sciences.

PO3) Design/ Development of Solutions: Design solutions for complex computer engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.

PO4) Conduct investigations of complex engineering problems using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.

PO5) Modern Tool Usage: Create, select and apply appropriate techniques, resources and modern computer engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6) The Engineer and Society: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to computer engineering practice.

PO7) Environment and Sustainability: Understand the impact of professional computer engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.

PO8) Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of computer engineering practice.

PO9) Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10) Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.

PO11) Project Management and Finance: Demonstrate knowledge and understanding of computer engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12) Life-long Learning: Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Program specific Outcomes

PSO1) An ability to manage and analyze data / information effectively for making better decisions.

PSO2) Demonstrate the ability to use state of the art technologies and tools including Free and Open Source Software (FOSS) tools in developing software.

Title:Budgettracker **Roll No. 46**

Lab Objectives:

Sr. No.	Lab Objectives
The Lab experiments aims:	
1	Learn the basics of the Flutter framework.
2	Develop the App UI by incorporating widgets, layouts, gestures and animation
3	Create a production ready Flutter App by including files and firebase backend service.
4	Learn the Essential technologies, and Concepts of PWAs to get started as quickly and efficiently as possible
5	Develop responsive web applications by combining AJAX development techniques with the jQuery JavaScript library.
6	Understand how service workers operate and also learn to Test and Deploy PWA.

Lab Outcomes:

Sr. No.	Lab Outcomes	Cognitive levels of attainment as per Bloom's Taxonomy
On Completion of the course the learner/student should be able to:		
1	Understand cross platform mobile application development using Flutter framework	L1, L2
2	Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation	L3
3	Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS	L3, L4
4	Understand various PWA frameworks and their requirements	L1, L2
5	Design and Develop a responsive User Interface by applying PWA Design techniques	L3
6	Develop and Analyse PWA Features and deploy it over app hosting solutions	L3, L4

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	17/1/24	24/1/24	13
2.	To design Flutter UI by including common widgets.	LO2	24/1/24	31/1/24	12
3.	To include icons, images, fonts in Flutter app	LO2	31/1/24	7/2/24	12
4.	To create an interactive Form using form widget	LO2	7/2/24	14/2/24	13
5.	To apply navigation, routing and gestures in Flutter App	LO2	14/2/24	21/2/24	13
6.	To Connect Flutter UI with fireBase database	LO3	21/2/24	6/3/24	13
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	6/3/24	13/3/24	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	13/3/24	20/3/24	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	20/3/24	27/3/24	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	27/3/24	27/3/24	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	27/3/24	27/3/24	15
12.	Assignment-1	LO1,LO2,LO3	4/2/24	5/2/24	4
13.	Assignment-2	LO4,LO5,LO6	20/3/24	21/3/24	4

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

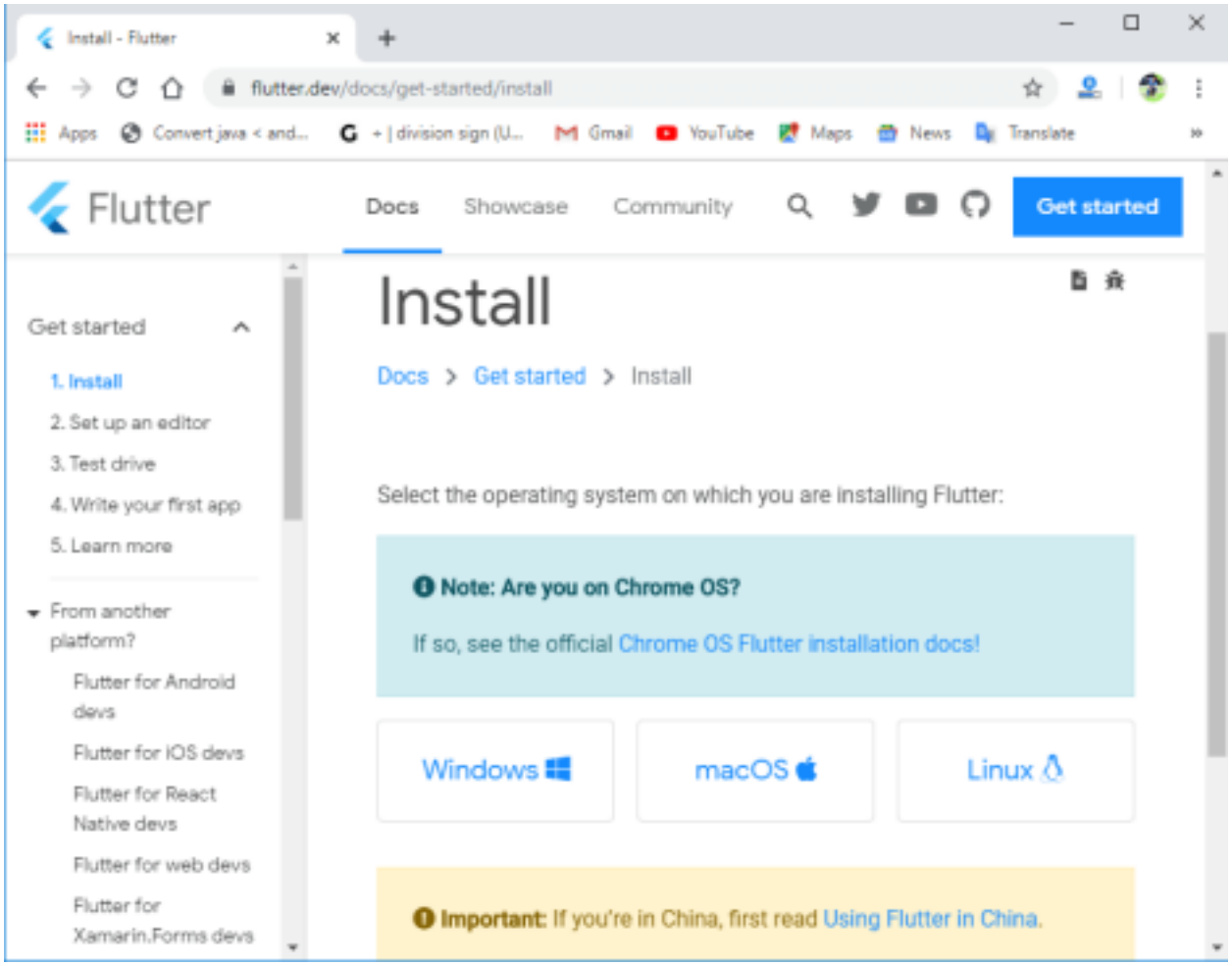
Title:Budgettracker **Roll No. 46**

Aim: To Install and Configure Flutter Environment

Pre Requisites:

Install the Flutter SDK

Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.

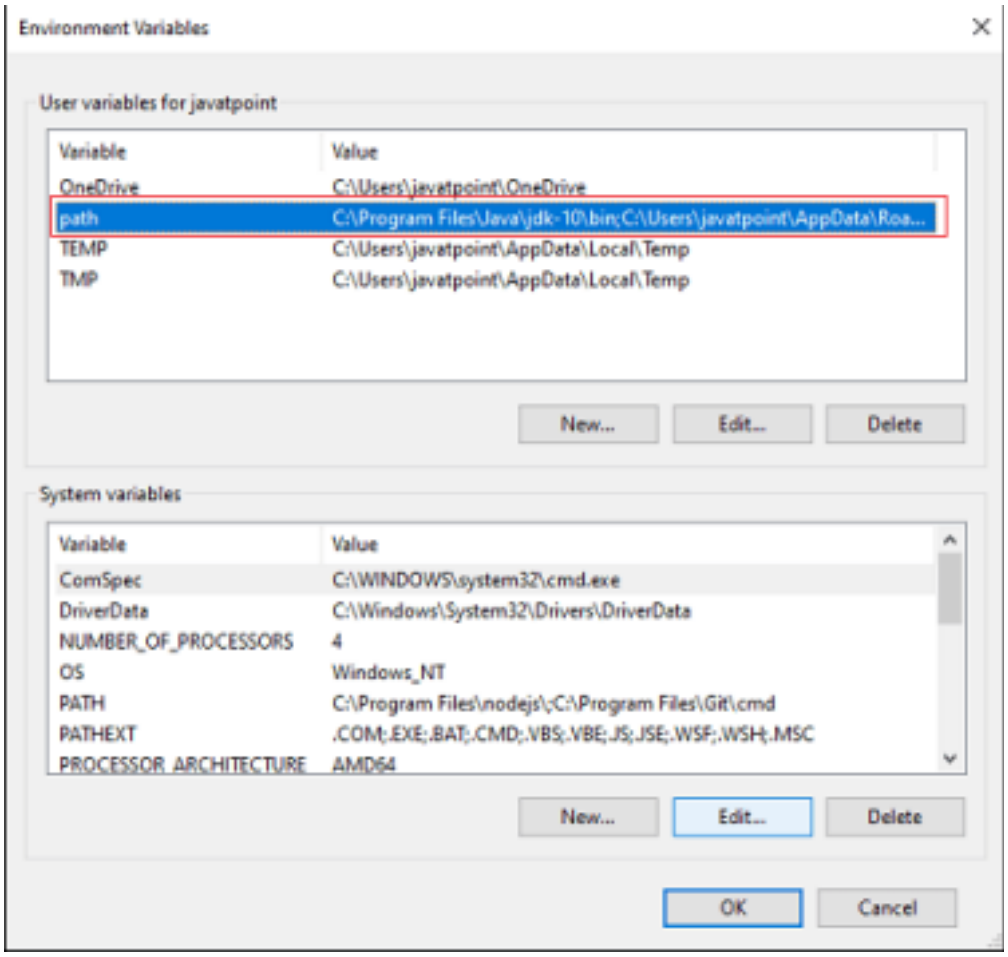


Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

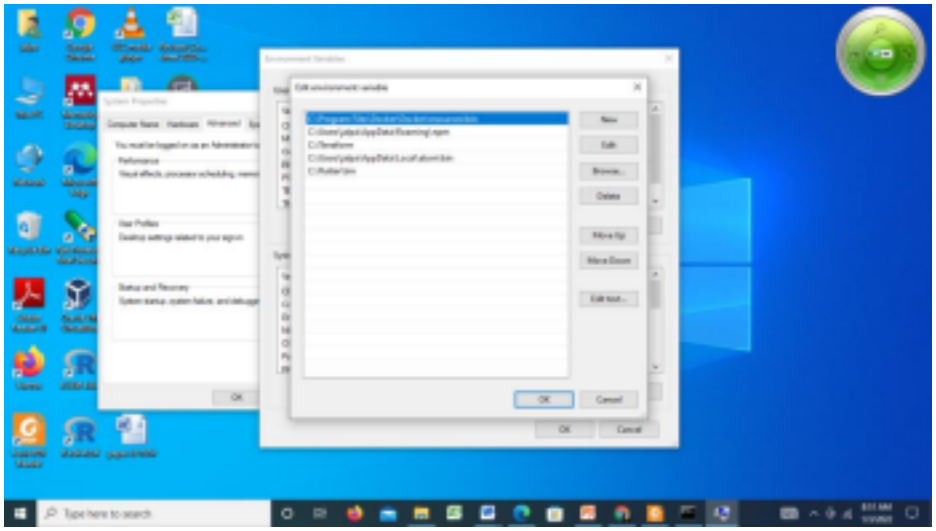
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.



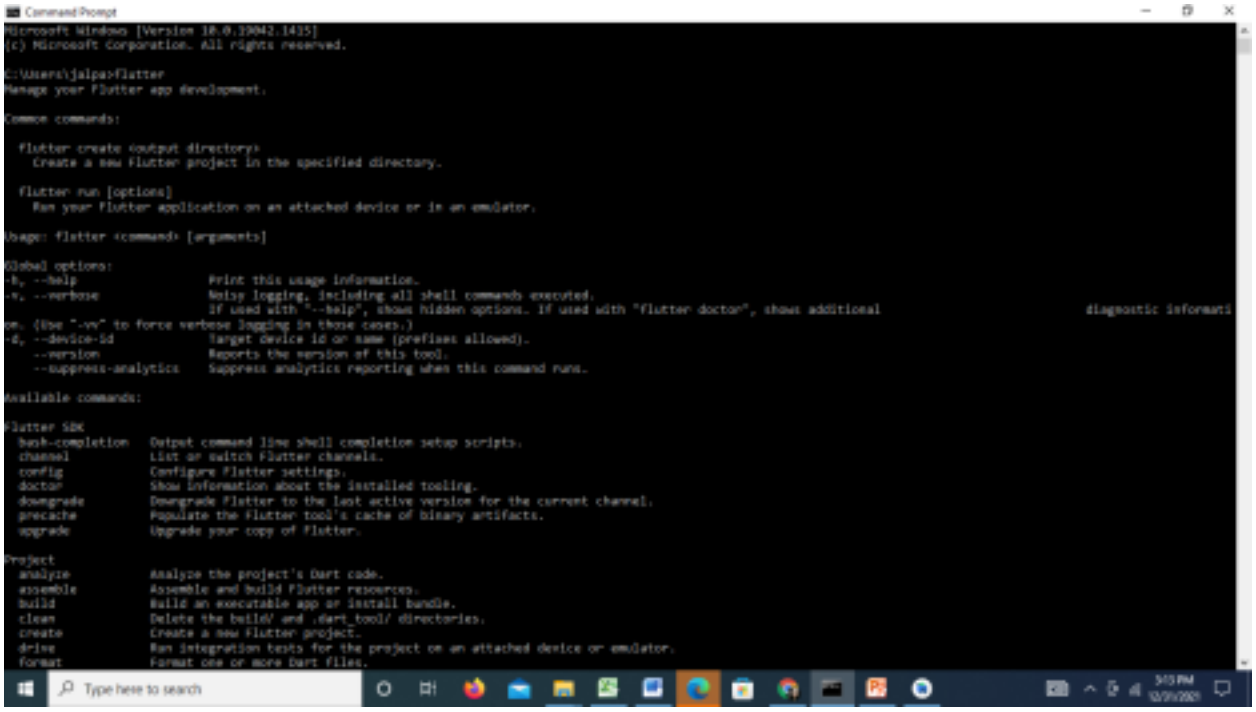
Step 4.2: Now, select path -> click on edit. The following screen appears



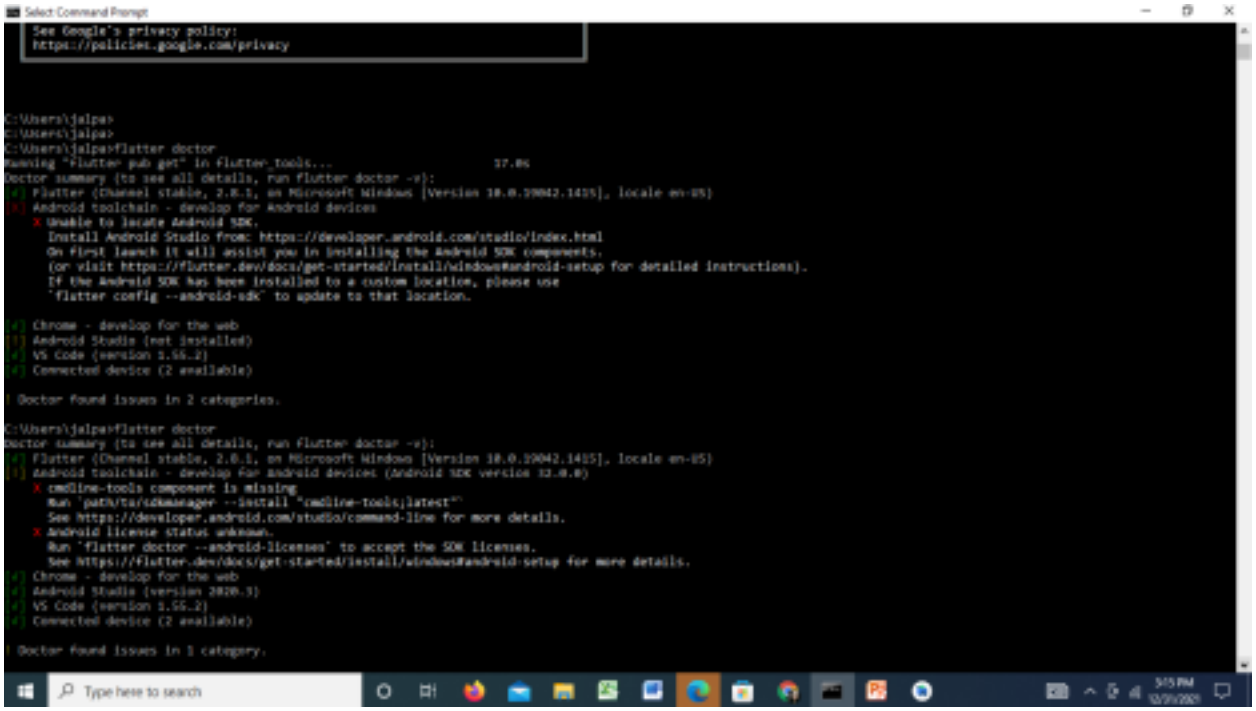
Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable

value - > ok -> ok -> ok.

Step 5: Now, run the $ **flutter** command in command prompt.



Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.



Step 6: When you run the above command, it will analyze the system and show its report, as

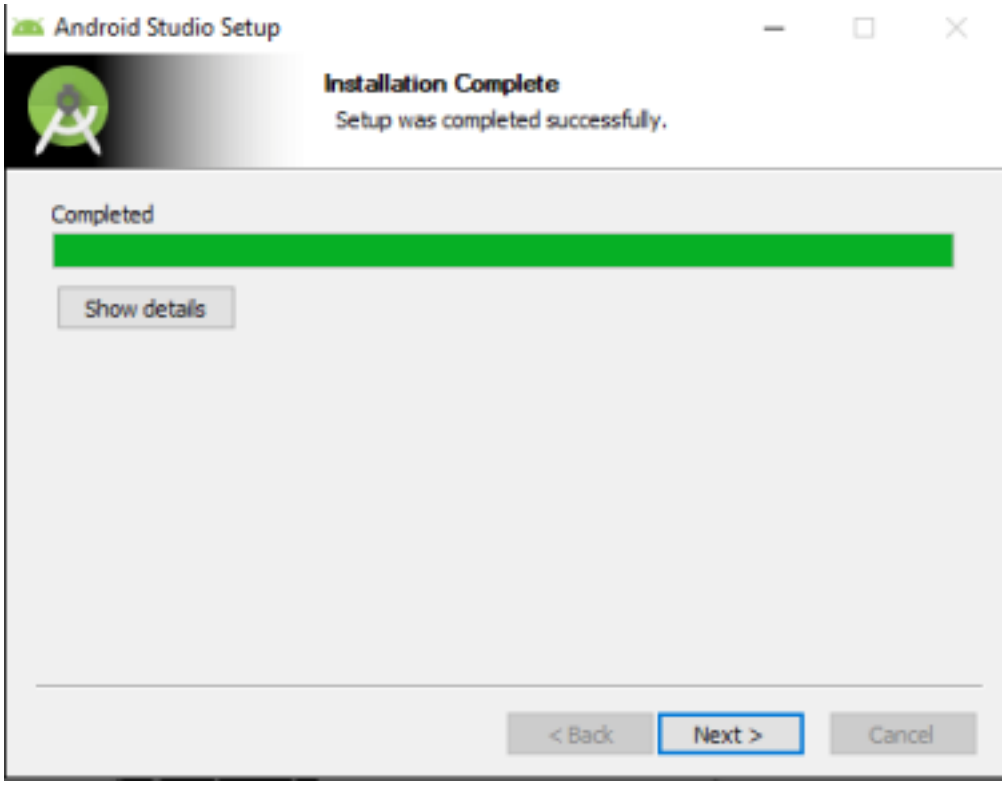
shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

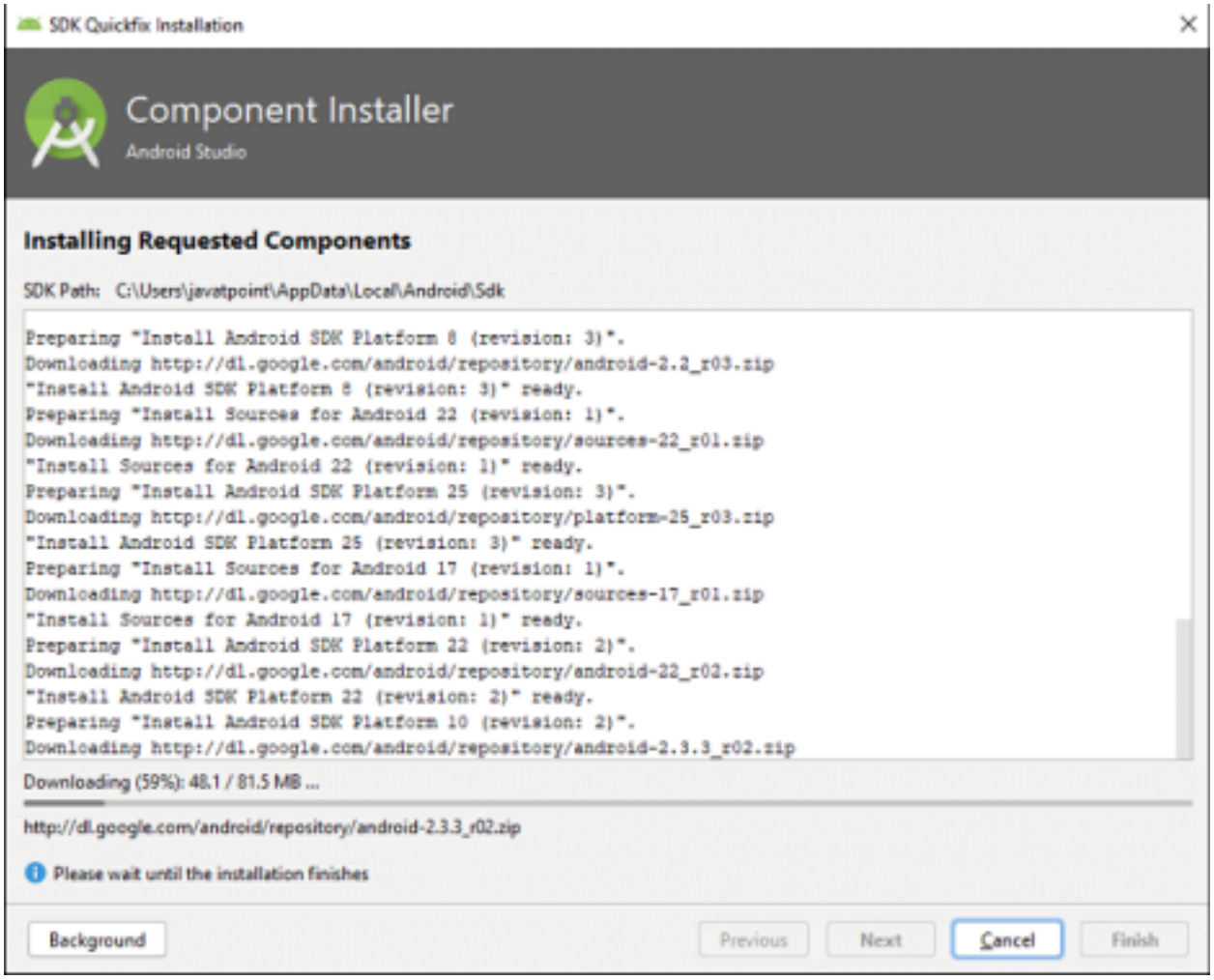
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

Step 7.2: When the download is complete, open the **.exe** file and run it. You will get the following dialog box.

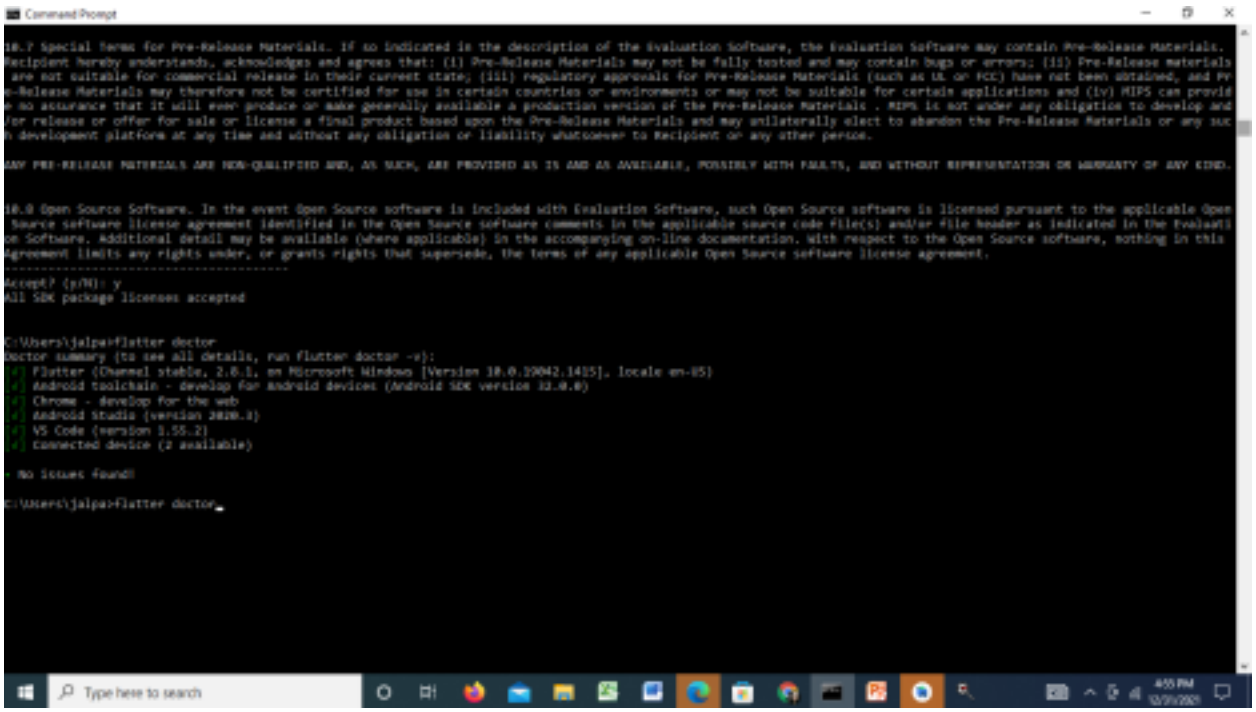
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

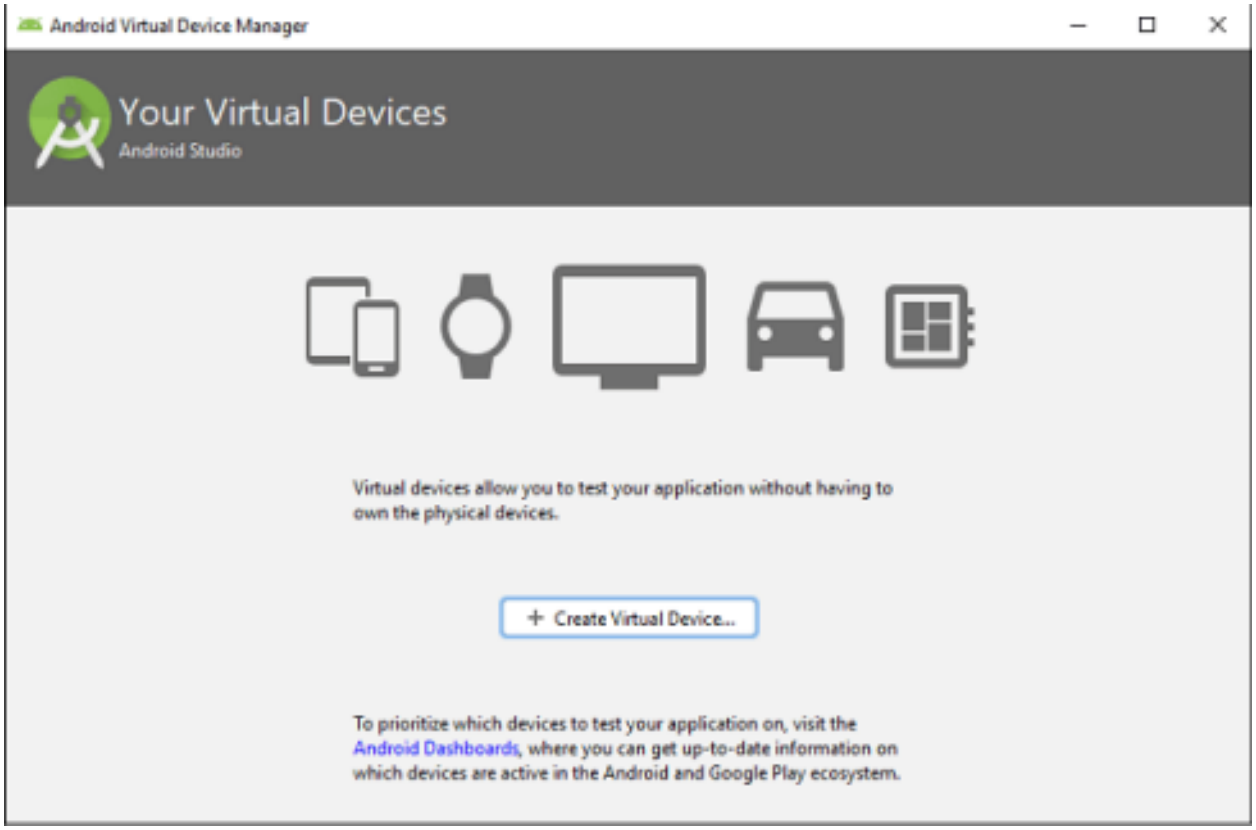


Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.



Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

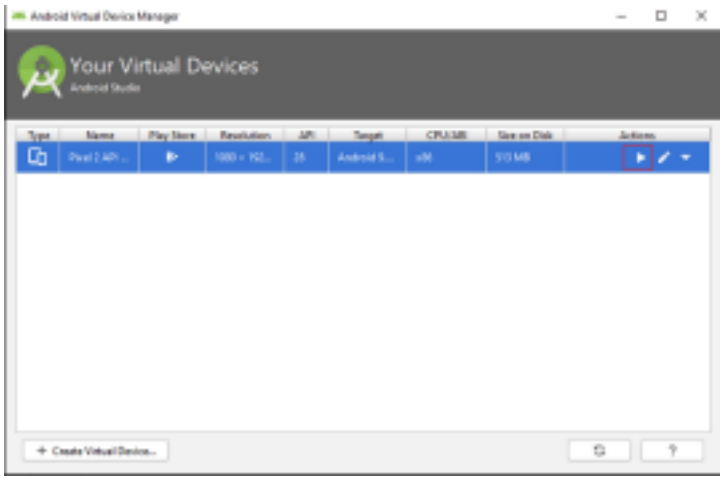
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



Step 8.2: Choose your device definition and click on Next.

Step 8.3: Select the system image for the latest Android version and click on Next.

Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.

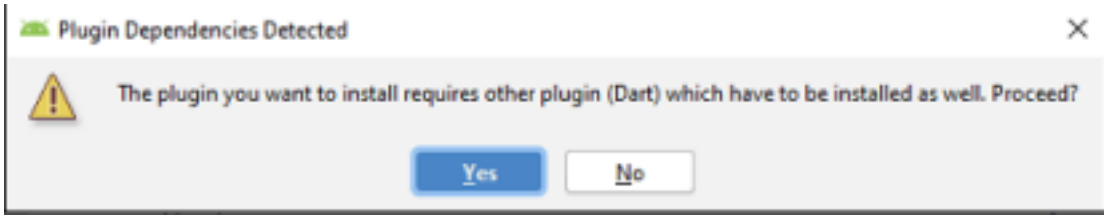


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

Step 9: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.

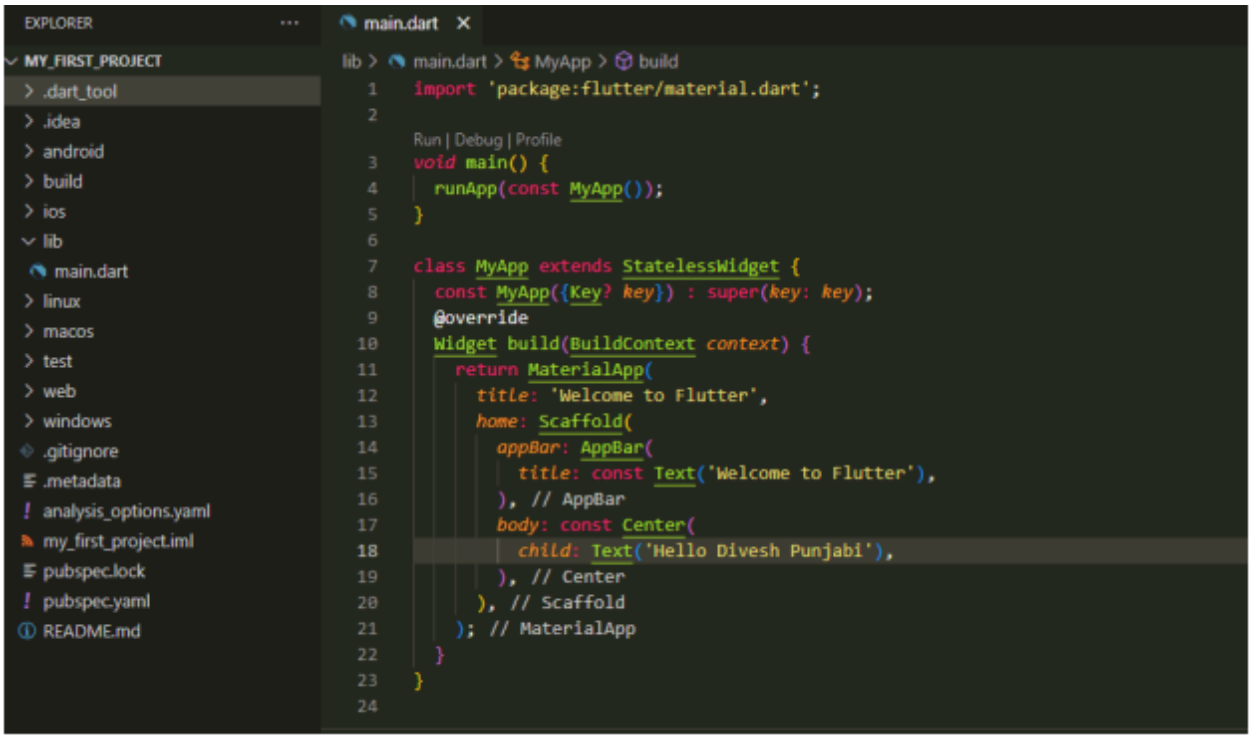
Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.



Step 9.3: Restart the Android Studio.

Title:Budgettracker **Roll No. 46**

Code:



Output:

Title:Budgettracker **Roll No. 46**

Conclusion: Hence We ran a simple program on running a simple text,on flutter, running on a virtual device

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment No 2

AIM: To design Flutter UI by including common widgets.

Theory:

In summary, Flutter widgets are fundamental components in constructing the user interface of a Flutter application.They can be broadly categorized into two types: `StatelessWidget` representing immutable parts of the UI and `StatefulWidget` representing mutable components that can change over time.

Some key Flutter widgets include:

1. Scaffold: The basic structure for a Flutter app, providing layout elements such as AppBar, BottomNavigationBar, and a body for main content.
2. Container: A versatile box model used for layout, padding, margin, decoration, and constraints, capable of containing other widgets.
3. Row & Column: Widgets for arranging child widgets horizontally (Row) or vertically (Column), essential for creating flexible and responsive layouts.
4. Text: Used for displaying text on the screen with support for various styling options like font size, color, and alignment.
5. TextField: Captures user input, such as text, numbers, or passwords, with the `onChanged` property for dynamic updates based on user input.
6. Buttons: Various button widgets like `ElevatedButton` or `TextButton` trigger actions when pressed, providing a means for user interaction.
7. Forms: The `Form` widget manages a group of `TextFormField` widgets, facilitating input validation and submission.
8. Icons: The `Icon` widget displays icons from libraries, enhancing visual elements and conveying meaning through symbols.

Key Design Principles highlighted include:

- Consistency: Common widget usage fosters a consistent design language throughout the app.
- Responsive Layouts: Widgets like `Row` and `Column` aid in creating responsive and flexible layouts, adapting to different screen sizes.
- User Input Handling: `TextField` and `Form` widgets facilitate proper handling, ensuring data integrity and validation.
- Interactive Elements: Buttons and icons contribute to interactivity and user engagement within the app.
- Visual Styling: The `Container` widget and styling properties of other widgets allow for visual customization and theming.

Hero Card: A hero card typically refers to a prominent or featured card within a user interface, often used in web design or mobile app design. A hero card is usually larger in size compared to other cards and is placed prominently on a page or screen to draw attention to a specific piece of content, product, or feature. It may contain a title, description, image, and call-to-action button, among other elements.

Card One: "Cardone" could potentially refer to a variety of things depending on the context. In software development or user interface design, it might refer to the first card in a series of cards, or it could be a specific component or module named "CardOne" within a codebase or design system.

Code:
Home_screen.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';
import 'package:my_project/screens/login_screen.dart';
import 'package:my_project/widgets/add_transaction_form.dart';
import 'package:my_project/widgets/hero_card.dart';
import 'package:my_project/widgets/transactions_cards.dart';

// ignore_for_file:prefer_const_constructors
// ignore_for_file:prefer_const_literals_to_create_immutables
class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  var isLogoutLoading = false;
  logOut() async {
    setState(() {
      isLogoutLoading = true;
    });
    await FirebaseAuth.instance.signOut();
    Navigator.of(context)
        .pushReplacement(MaterialPageRoute(builder: (context) => LoginView()));

    setState(() {
      isLogoutLoading = false;
    });
  }

  _dialoBuilder(BuildContext context) {
```

```
  return showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          content: AddTransactionForm(),
        );
      });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    floatingActionButton: FloatingActionButton(
      backgroundColor: Colors.blue.shade900,
      onPressed: (() {
        _dialoBuilder(context);
      }),
      child: Icon(
        Icons.add,
        color: Colors.white,
      ),
    ),
    appBar: AppBar(
      backgroundColor: Colors.blue.shade900,
      title: Text(
        "Hello",
        style: TextStyle(color: Colors.white),
      ),
      actions: [
        IconButton(
            onPressed: () {
              logOut();
            },
            icon: isLogoutLoading
                ? CircularProgressIndicator()
                : Icon(
                    Icons.exit_to_app,
                    color: Colors.white,
                  ))
      ],
    ),
```

```
      body: Column(
        children: [
          HeroCard(),
          TransactionsCard(),

        ],
      ),
    );
  }
}
```

Herocard.dart

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

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

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.blue.shade900,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Padding(
            padding: const EdgeInsets.all(15),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  "Total Balance",
                  style: TextStyle(
                      fontSize: 18,
                      color: Colors.white,
                      height: 1.2,
                      fontWeight: FontWeight.w600),
                ),
                Text(
                  "₹ 582000",
```

```
                style: TextStyle(
                    fontSize: 50,
                    color: Colors.white,
                    height: 1.2,
                    fontWeight: FontWeight.w600),
              )
            ],
          ),
        ),
        Container(
          padding: EdgeInsets.only(top: 30, bottom: 10, left: 10, right: 10),
          decoration: BoxDecoration(
            borderRadius: BorderRadius.only(
                topLeft: Radius.circular(30), topRight: Radius.circular(30)),
            color: Colors.white,
          ),
          // color: Colors.white,
          child: Row(
            children: [
              CardOne(
                color: Colors.green,
              ),
              SizedBox(
                width: 8,
              ),
              CardOne(
                color: Colors.red,
              ),
            ],
          ),
        )
      ],
    ),
  );
 }
}

class CardOne extends StatelessWidget {
 const CardOne({
   super.key,
   required this.color,
```

```
});
final Color color;
@override
Widget build(BuildContext context) {
 return Expanded(
  child: Container(
   decoration: BoxDecoration(
     color: color.withOpacity(0.1),
     borderRadius: BorderRadius.circular(10)),
   // color: color.withOpacity(0.1),
   child: Padding(
    padding: const EdgeInsets.all(8.0),
    child: Row(
     children: [
      Column(
       crossAxisAlignment: CrossAxisAlignment.start,
       children: [
        Text(
         "Credit",
         style: TextStyle(color: color, fontSize: 20),
        ),
        Text(
         "₹ 58000",
         style: TextStyle(
           color: color,
           fontSize: 25,
           fontWeight: FontWeight.w600),
        )
       ],
      ),
      Spacer(),
      Padding(
       padding: const EdgeInsets.all(8.0),
       child: Icon(
        Icons.arrow_upward_outlined,
        color: color,
       ),
      )
     ],
    ),
   ),
```

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

Transaction Card.dart

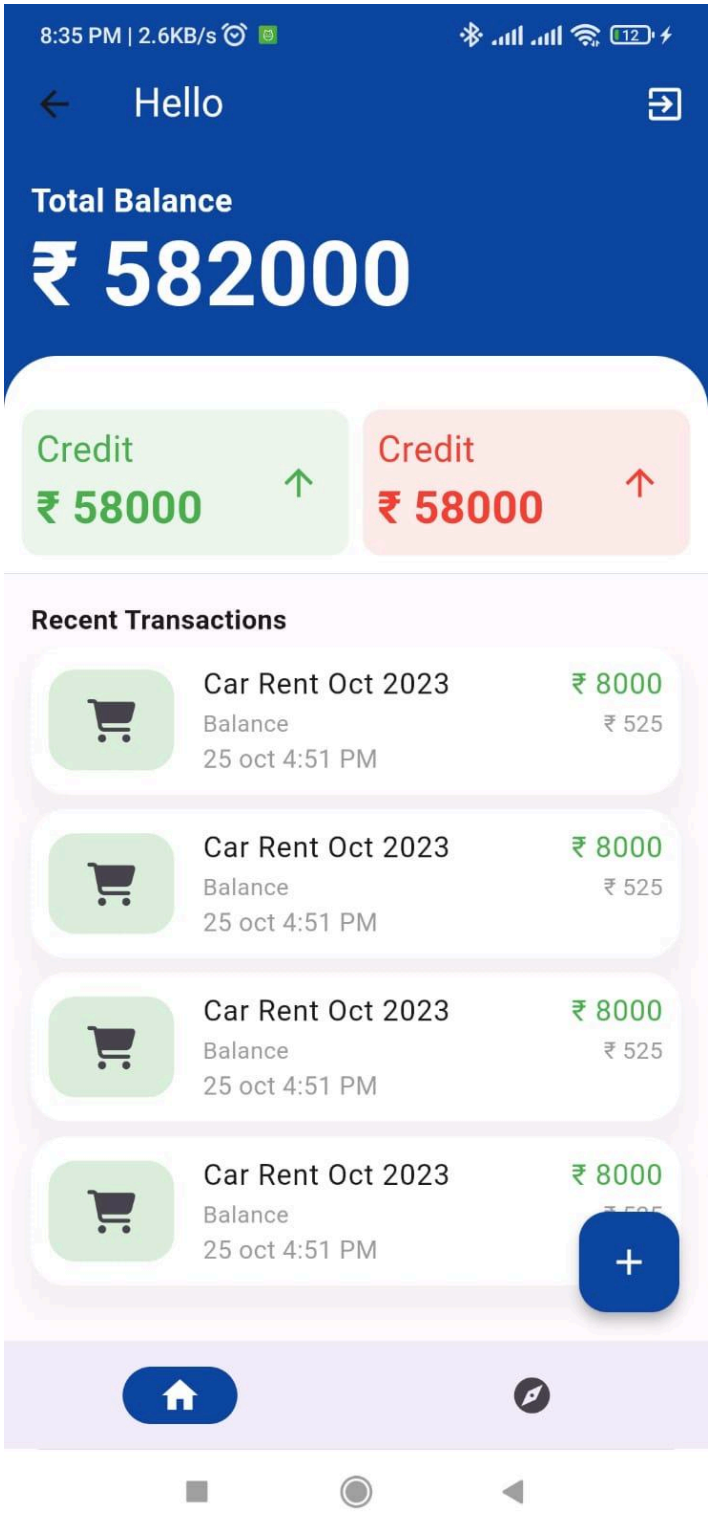
```
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:my_project/utils/icons_list.dart';

class TransactionsCard extends StatelessWidget {
  TransactionsCard({super.key});

// ignore_for_file:prefer_const_literals_to_create_immutables
  var appIcons = AppIcons();
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(15),
      child: Column(
        children: [
          // Padding(
          //     padding: const EdgeInsets.all(15),
          Row(
            children: [
              Text(
                "Recent Transactions",
                style: TextStyle(fontSize: 15, fontWeight: FontWeight.w600),
              )
            ],
          ),
          ListView.builder(
              shrinkWrap: true,
              itemCount: 4,
              physics: NeverScrollableScrollPhysics(),
              itemBuilder: (context, indes) {
                return Padding(
                  padding: const EdgeInsets.symmetric(vertical: 4),
                  child: Container(
                    decoration: BoxDecoration(
                        color: Colors.white,
```

```
    borderRadius: BorderRadius.circular(20),
    boxShadow: [
      BoxShadow(
          offset: Offset(0, 10),
          color: Colors.grey.withOpacity(0.09),
          blurRadius: 10.0,
          spreadRadius: 4.0)
    ]),
  child: ListTile(
   minVerticalPadding: 10,
   contentPadding:
       EdgeInsets.symmetric(horizontal: 10, vertical: 0),
   leading: Container(
    width: 70,
    height: 100,
    child: Container(
      width: 30,
      height: 30,
      decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(15),
          color: Colors.green.withOpacity(0.2)),
      child: Center(
          child: FaIcon(
              appIcons.getExpenseCategoryIcons('home'))),
    ),
   ),
   title: Row(
    children: [
      Expanded(child: Text("Car Rent Oct 2023")),
      Text(
        "₹ 8000",
        style: TextStyle(color: Colors.green),
      )
    ],
   ),
   subtitle: Column(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        children: [
```

```
                    Text(
                      "Balance",
                      style:
                          TextStyle(color: Colors.grey, fontSize: 13),
                    ),
                    Spacer(),
                    Text(
                      "₹ 525",
                      style:
                          TextStyle(color: Colors.grey, fontSize: 13),
                    )
                  ],
                ),
                Text(
                  "25 oct 4:51 PM",
                  style: TextStyle(color: Colors.grey),
                )
              ],
            ),
          ),
        ),
      );
    })
  // ),
 ],
 ),
 );
 }
}
```

Output:

Title:Budgettracker **Roll No. 46**



Conclusion:

implementing common widgets improve user experience, and contribute to the overall success of the application.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment no 3

Aim:To include images and fonts in flutter app.

Theory:
Theory: Certainly! Here's a simplified process for adding images in Flutter:
1. Import Libraries:
Ensure that you have the necessary libraries imported in your Dart file. For images, you'll typically use `dart:ui` and other relevant Flutter packages.
2. Adding Local Images:
- Place your local images in the `assets` folder.
- Declare the images in the `pubspec.yaml` file.
3. Adding Network Images:
- Use the `Image.network` widget for displaying images from the internet.
4. Image Widget:
- Create an `Image` widget and provide it with an `ImageProvider`.
- Use `AssetImage` for local images and `NetworkImage` for network images.
5. ImageProvider:
- Understand that `AssetImage` and `NetworkImage` are subclasses of the `ImageProvider` class.
- You can create custom `ImageProvider` if needed.
6. CachedNetworkImage (Optional):
- If you want to cache network images, consider using the `cached_network_image` package.
7. Image Loading and Error Handling:
- Customize the loading and error behavior using `loadingBuilder` and `errorBuilder` properties of the `Image` widget or other relevant widgets.
Remember, the actual implementation details might vary based on your specific use case and the packages you choose to use. The key is to understand the concepts of working with local and network images and the various widgets and packages available in Flutter for handling images.

Code:
```
import 'package:flutter/material.dart';
import 'package:my_project/screens/dashboard.dart';
import 'package:my_project/screens/sign_up.dart';
import 'package:my_project/services/auth_service.dart';
import 'package:my_project/utils/appvalidator.dart';

class LoginView extends StatefulWidget {
  LoginView({super.key});
```

```
  @override
  State<LoginView> createState() => _LoginViewState();
}

class _LoginViewState extends State<LoginView> {
  final GlobalKey<FormState> _formkey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  var isLoader = false;
  var authServices = AuthServices();
  Future<void> _submitform() async {
    if (_formkey.currentState!.validate()) {
      setState(() {
        isLoader = true;
      });

      var data = {
        "email": _emailController.text,
        "password": _passwordController.text,
      };

      await authServices.login(data, context);
      Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => Dashboard()));
      setState(() {
        isLoader = false;
      });
    }
  }

  var appValidator = AppValidator();

  @override
 Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xFF000000),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
```

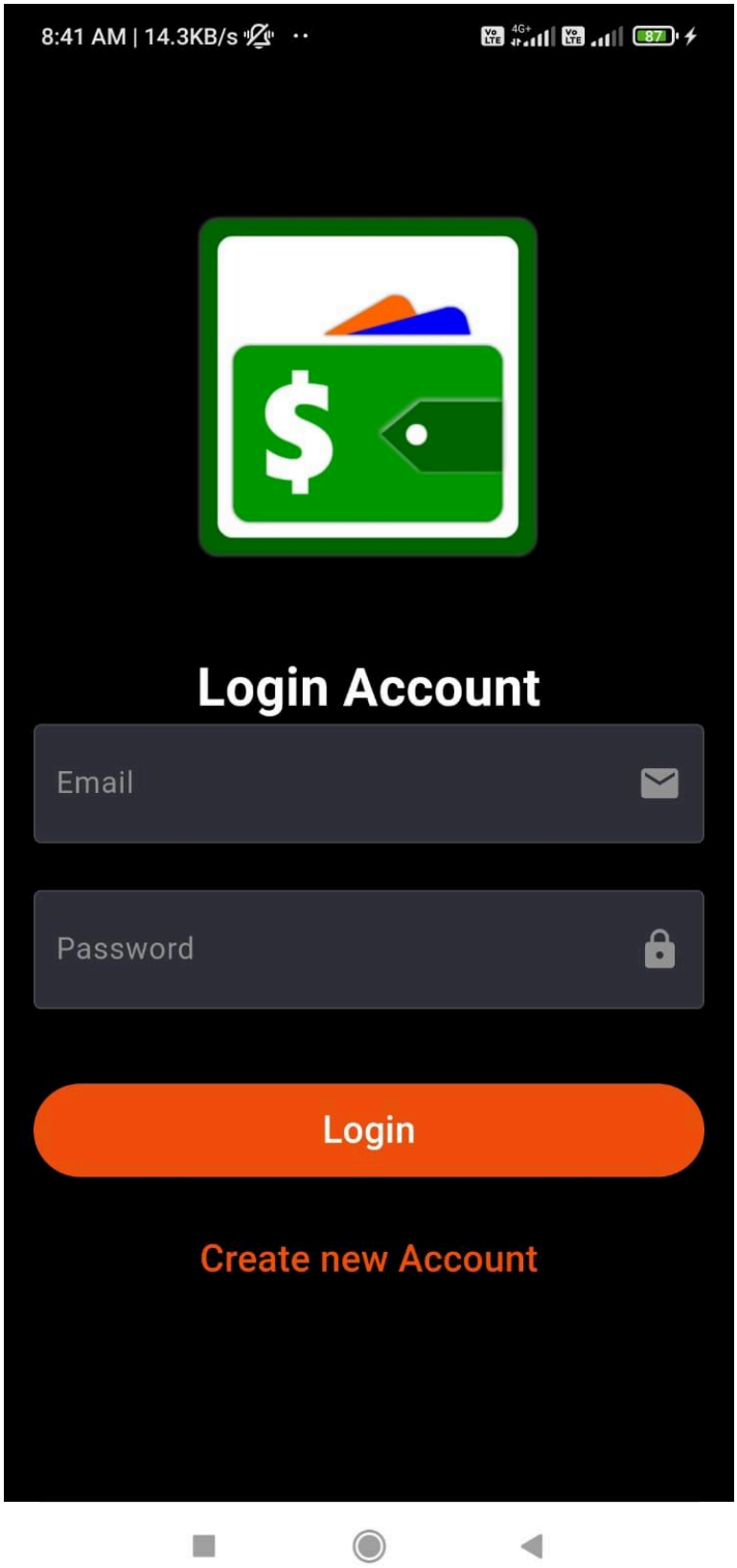
```
children: [
  Image.asset(
    'assets/images/expense-manager-budget-planner-15.png',
    width: 200,
    height: 200,
  ),
  SizedBox(height: 20),
  Form(
    key: _formkey,
    child: Column(
      children: [
        SizedBox(
          height: 20,
        ),
        SizedBox(
          width: 250,
          child: Text(
            "Login Account",
            textAlign: TextAlign.center,
            style: TextStyle(
              color: Colors.white,
              fontSize: 28,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
        TextFormField(
          controller: _emailController,
          keyboardType: TextInputType.emailAddress,
          style: TextStyle(color: Colors.white),
          autovalidateMode: AutovalidateMode.onUserInteraction,
          decoration: _buildInputDecoration("Email", Icons.email),
          validator: appValidator.validateEmail,
        ),
        SizedBox(
          height: 25,
        ),
        TextFormField(
          controller: _passwordController,
          keyboardType: TextInputType.text,
          autovalidateMode: AutovalidateMode.onUserInteraction,
```

```
  decoration: _buildInputDecoration("Password", Icons.lock),
  validator: appValidator.validatePassword,
),
SizedBox(
  height: 40.0,
),
SizedBox(
  height: 50,
  width: double.infinity,
  child: ElevatedButton(
    style: ElevatedButton.styleFrom(
      backgroundColor: Color(0xFFF75104),
    ),
    onPressed: () {
      isLoader ? print("Loading") : _submitform();
    },
    // _submitform,
    child: isLoader
        ? Center(child: CircularProgressIndicator())
        : Text(
            "Login",
            style: TextStyle(
              color: Colors.white,
              fontSize: 20,
            ),
          ),
  ),
),
SizedBox(
  height: 20.0,
),
TextButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => SignUpView()),
    );
  },
  child: Text(
    "Create new Account",
    style: TextStyle(
```

```
                color: Color(0xFFF75104),
                fontSize: 20,
              ),
            ),
          ),
        ],
      ),
    ),
   ],
  ),
 ),
);
}

InputDecoration _buildInputDecoration(String label, IconData suffixIcon) {
 return InputDecoration(
   fillColor: Color(0xAA494A59),
   enabledBorder: OutlineInputBorder(
     borderSide: BorderSide(color: Color(0x35949494))),
   focusedBorder:
     OutlineInputBorder(borderSide: BorderSide(color: Colors.white)),
   filled: true,
   labelStyle: TextStyle(color: Color(0xff949494)),
   labelText: label,
   suffixIcon: Icon(
    suffixIcon,
    color: Color(0xFF949494),
   ),
   border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)));
 }
}
```

Output:

Title:Budgettracker **Roll No. 46**



Conclusion:Thus understood to add images in flutter and also i faced a difficulties in pubspec.yaml file

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment no 4

Aim - To create an interactive Form using form widget

Theory -

Interactive Form Creation in Flutter

Flutter offers an array of powerful widgets and techniques to build adaptable and user-friendly forms that seamlessly integrate with your app's design and functionality. This guide delves into the key concepts and strategies involved in crafting interactive forms.

Fundamental Widgets:

- Form: The overarching container that encompasses form fields and manages validation. Create a unique GlobalKey<FormState> for global access and validation.
- FormField: A base class for input widgets, providing styling, validation, and error handling. Choose the appropriate concrete widget based on the input type (e.g., TextField, Checkbox, DropdownButton).
- TextInputAction: Set the textInputAction property on TextField to control the on-screen keyboard's behavior (e.g., TextInputAction.next, TextInputAction.done).
- FocusNode: Control keyboard focus navigation between form fields or widgets within a field (e.g., for multi-line text editing). Use FocusNode objects with FocusManager for management.

Dynamic Form Building:

- ListViews: Dynamically render form fields by creating a ListView.builder that builds FormField instances based on data (e.g., from a list or JSON). Customize field types and validation based on data properties.
- State Management: Employ state management solutions like Bloc, Provider, or raw setState to handle form state effectively. Store form data, validation errors, and other state information dynamically.

User Interaction and Validation:

- Input Validation: Implement validation rules within FormField or its descendant widgets using a validator callback. Provide clear error messages for invalid input to guide users.
- Focus Management: Use FocusNode to control keyboard focus flow, enabling a natural form-filling experience. Consider using packages like auto_animated for field auto-focusing when entering the screen.
- Interactive UI: Incorporate widgets like Checkbox, DropdownButton, Radio, and custom interactive elements to offer a variety of input options and enhance user engagement.

Additional Considerations:

- Performance: For large forms, consider techniques like lazy loading or pagination to manage memory and rendering overhead.
- Accessibility: Ensure your forms are accessible to users with disabilities by following WCAG guidelines and using appropriate semantic elements.
- Styling: Customize form aesthetics using Flutter's rich customization options to match your app's design and provide visual feedback (e.g., underline active fields, highlight errors)

Code:

Login Page:

```
import 'package:flutter/material.dart';
import 'package:my_project/screens/dashboard.dart';
import 'package:my_project/screens/sign_up.dart';
import 'package:my_project/services/auth_service.dart';
import 'package:my_project/utils/appvalidator.dart';

class LoginView extends StatefulWidget {
  LoginView({super.key});

  @override
  State<LoginView> createState() => _LoginViewState();
}

class _LoginViewState extends State<LoginView> {
  final GlobalKey<FormState> _formkey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  var isLoader = false;
  var authServices = AuthServices();
  Future<void> _submitform() async {
    if (_formkey.currentState!.validate()) {
      setState(() {
        isLoader = true;
      });

      var data = {
        "email": _emailController.text,
        "password": _passwordController.text,
      };

      await authServices.login(data, context);
      Navigator.of(context).pushReplacement(
```

```
        MaterialPageRoute(builder: (context) => Dashboard()));
    setState(() {
      isLoader = false;
    });
  }
}

var appValidator = AppValidator();

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF000000),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
          key: _formkey,
          child: Column(
            children: [
              SizedBox(
                height: 80.0,
              ),
              SizedBox(
                width: 250,
                child: Text(
                  "Login Account",
                  textAlign: TextAlign.center,
                  style: TextStyle(
                      color: Colors.white,
                      fontSize: 28,
                      fontWeight: FontWeight.bold),
                ),
              ),

              TextFormField(
                  controller: _emailController,
                  keyboardType: TextInputType.emailAddress,
                  style: TextStyle(color: Colors.white),
                  autovalidateMode: AutovalidateMode.onUserInteraction,
                  decoration: _buildInputDecoration("Email", Icons.email),
                  validator: appValidator.validateEmail),
```

```
SizedBox(
  height: 16.0,
),

TextFormField(
    controller: _passwordController,
    keyboardType: TextInputType.phone,
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration: _buildInputDecoration("Password", Icons.lock),
    validator: appValidator.validatePassword),
SizedBox(
  height: 40.0,
),
SizedBox(
    height: 50,
    width: double.infinity,
    child: ElevatedButton(
        style: ElevatedButton.styleFrom(
            backgroundColor: Color(0xFFF75104)),
        onPressed: () {
          isLoader ? print("Loading") : _submitform();
        },
        // _submitform,
        child: isLoader
            ? Center(child: CircularProgressIndicator())
            : Text("Login"))),
SizedBox(
  height: 20.0,
),
TextButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => SignUpView()),
      );
    },
    child: Text(
      "Create new Account",
      style: TextStyle(color: Color(0xFFF75104), fontSize: 25),
    ))
// Text(
```

```
            //   "Login",
            //   style: TextStyle(color:Color(0xFFF75104),fontSize: 25 ),
            // )
           ],
          )),
    ),
  );
 }

 InputDecoration _buildInputDecoration(String label, IconData suffixIcon) {
  return InputDecoration(
     fillColor: Color(0xAA494A59),
     enabledBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Color(0x35949494))),
     focusedBorder:
        OutlineInputBorder(borderSide: BorderSide(color: Colors.white)),
     filled: true,
     labelStyle: TextStyle(color: Color(0xff949494)),
     labelText: label,
     suffixIcon: Icon(
      suffixIcon,
      color: Color(0xFF949494),
     ),
     border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)));
 }
}
```

Signup Page:

```
import 'package:flutter/material.dart';
import 'package:my_project/screens/dashboard.dart';
import 'package:my_project/screens/login_screen.dart';
import 'package:my_project/services/auth_service.dart';
import 'package:my_project/utils/appvalidator.dart';

class SignUpView extends StatefulWidget {
 SignUpView({super.key});

 @override
 State<SignUpView> createState() => _SignUpViewState();
}
```

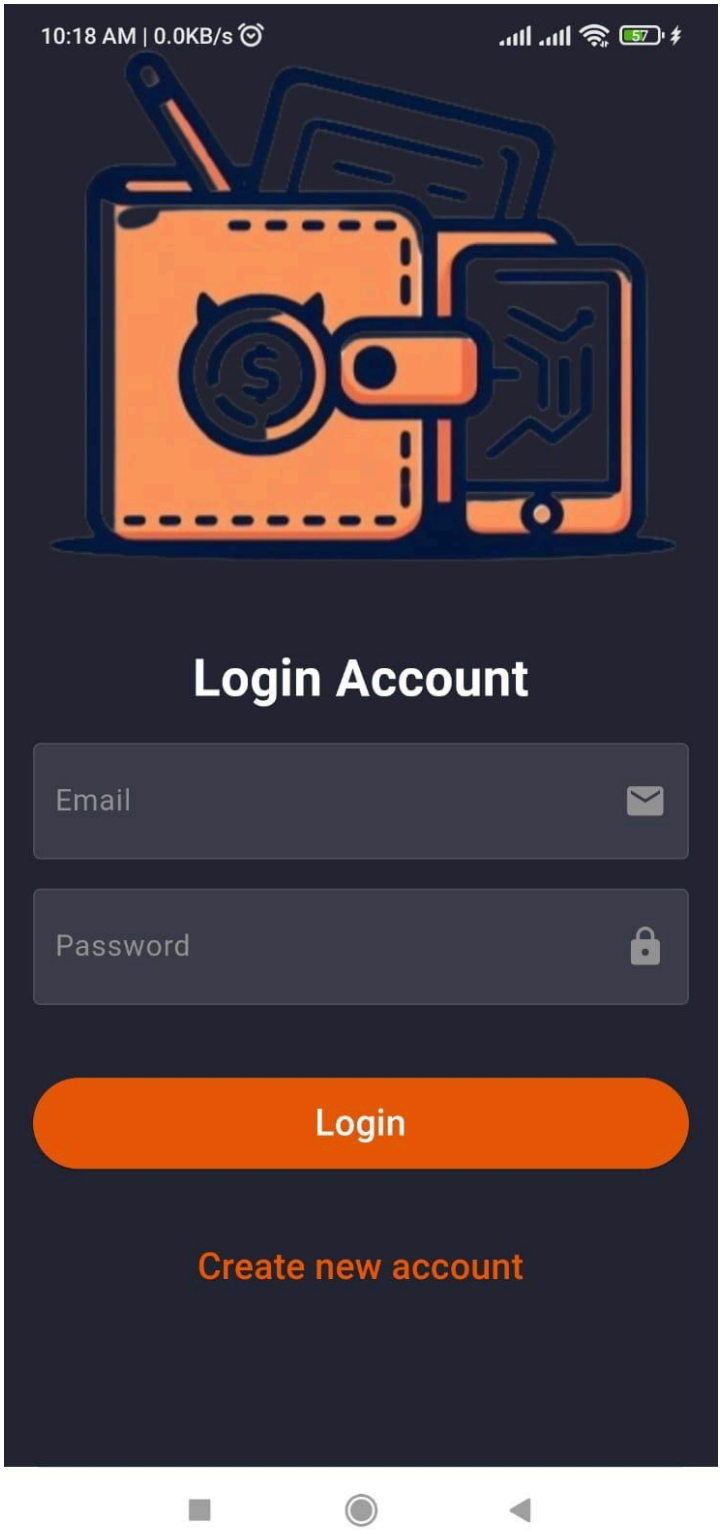
```
class _SignUpViewState extends State<SignUpView> {
  final GlobalKey<FormState> _formkey = GlobalKey<FormState>();

  final _userNameController = TextEditingController();

  final _emailController = TextEditingController();

  final _phoneController = TextEditingController();

  final _passwordController = TextEditingController();

  var authServices = AuthServices();
  var isLoader = false;

  Future<void> _submitform() async {
    if (_formkey.currentState!.validate()) {
      setState(() {
        isLoader = true;
      });

      var data = {
        "username": _userNameController.text,
        "email": _emailController.text,
        "password": _passwordController.text,
        "phone": _phoneController.text
      };

      await authServices.createUser(data, context);
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => Dashboard()),
      );
      setState(() {
        isLoader = false;
      }); // ScaffoldMessenger.of(_formkey.currentContext!).showSnackBar(
      //   const SnackBar(content: Text('Form submitted successfully')),
      // );
    }
  }

  var appValidator = AppValidator();
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF000000),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
          key: _formkey,
          child: Column(
            children: [
              SizedBox(
                height: 80.0,
              ),
              SizedBox(
                width: 250,
                child: Text(
                  "Create new Account",
                  textAlign: TextAlign.center,
                  style: TextStyle(
                      color: Colors.white,
                      fontSize: 28,
                      fontWeight: FontWeight.bold),
                ),
              ),
              SizedBox(
                height: 16.0,
              ),
              TextFormField(
                  controller: _userNameController,
                  style: TextStyle(color: Colors.white),
                  autovalidateMode: AutovalidateMode.onUserInteraction,
                  decoration: _buildInputDecoration("UserName", Icons.person),
                  validator: appValidator.validateUsername),
              //   if (value!.isEmpty) {
              //     return 'Please enter a username';
              //   }
              //   return null;
              // }),
              SizedBox(
                height: 16.0,
```

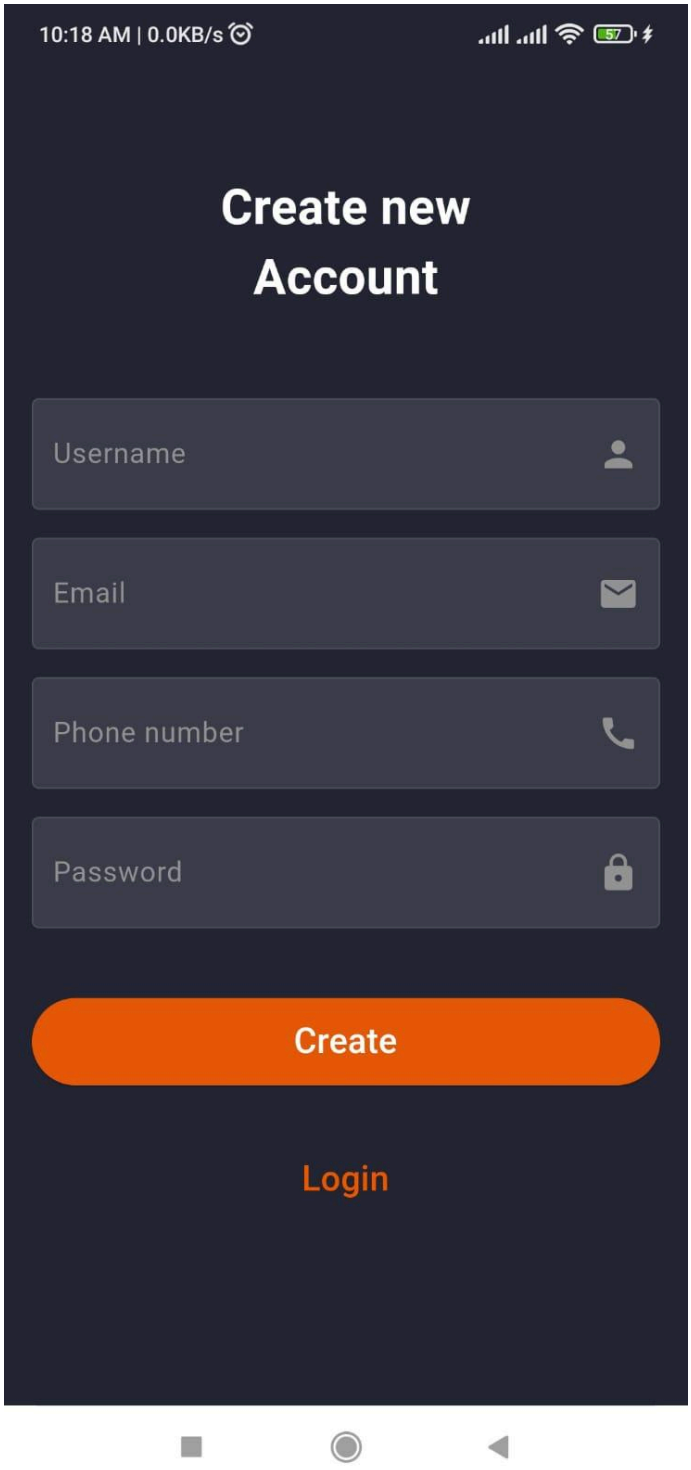
```
),
TextFormField(
   controller: _emailController,
   keyboardType: TextInputType.emailAddress,
   style: TextStyle(color: Colors.white),
   autovalidateMode: AutovalidateMode.onUserInteraction,
   decoration: _buildInputDecoration("Email", Icons.email),
   validator: appValidator.validateEmail),
SizedBox(
  height: 16.0,
),
TextFormField(
   controller: _phoneController,
   keyboardType: TextInputType.phone,
   style: TextStyle(color: Colors.white),
   autovalidateMode: AutovalidateMode.onUserInteraction,
   decoration:
      _buildInputDecoration("Phone Number", Icons.call),
   validator: appValidator.validatePhoneNumber),
SizedBox(
  height: 16.0,
),
TextFormField(
   controller: _passwordController,
   keyboardType: TextInputType.phone,
   autovalidateMode: AutovalidateMode.onUserInteraction,
   decoration: _buildInputDecoration("Password", Icons.lock),
   validator: appValidator.validatePassword),
SizedBox(
  height: 40.0,
),
SizedBox(
   height: 50,
   width: double.infinity,
   child: ElevatedButton(
      style: ElevatedButton.styleFrom(
         backgroundColor: Color(0xFFF75104)),
      onPressed: () {
        isLoader ? print("Loading") : _submitform();
      },
      // _submitform,
```

```
                child: isLoader
                    ? Center(child: CircularProgressIndicator())
                    : Text("Create"))),
            SizedBox(
              height: 20.0,
            ),
            TextButton(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => LoginView()),
                  );
                },
                child: Text(
                  "Login",
                  style: TextStyle(color: Color(0xFFF75104), fontSize: 25),
                ))
            // Text(
            //   "Login",
            //   style: TextStyle(color:Color(0xFFF75104),fontSize: 25 ),
            // )
          ],
        )),
    ),
  );
}

InputDecoration _buildInputDecoration(String label, IconData suffixIcon) {
  return InputDecoration(
      fillColor: Color(0xAA494A59),
      enabledBorder: OutlineInputBorder(
          borderSide: BorderSide(color: Color(0x35949494))),
      focusedBorder:
          OutlineInputBorder(borderSide: BorderSide(color: Colors.white)),
      filled: true,
      labelStyle: TextStyle(color: Color(0xff949494)),
      labelText: label,
      suffixIcon: Icon(
        suffixIcon,
        color: Color(0xFF949494),
      ),
```

```
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)));
  }
}
```



Title:Budgettracker **Roll No. 46**



Conclusion:
Through this experiment we understood how to create interactive form using widgets in flutter and how to validate data input in the Text fields.We also learned dynamic Form building.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment no 5

Aim: To apply routing in Flutter Application

Theory: In Flutter, routing refers to the navigation system that allows users to move between different screens or pages within an app. Flutter uses a widget-based approach for navigation, where each screen is represented by a widget. The `Navigator` class manages the stack of routes and facilitates transitions between them.

Routes are typically defined using the `MaterialPageRoute` class, providing a seamless and platform-aware transition between screens. Developers can use the `Navigator` to push new routes onto the stack or pop existing routes off it. Named routes help in easily identifying and navigating to specific screens.

Flutter also supports route arguments, allowing developers to pass data between screens. Additionally, the `Navigator` provides a flexible set of transitions, such as slide, fade, or custom animations, enhancing the user experience during navigation.

Overall, Flutter's routing system provides a structured and intuitive way to handle navigation within mobile and web applications.

Code:
Login_screen.dart

```
import 'package:flutter/material.dart';
import 'package:my_project/screens/dashboard.dart';
import 'package:my_project/screens/sign_up.dart';
import 'package:my_project/services/auth_service.dart';
import 'package:my_project/utils/appvalidator.dart';

class LoginView extends StatefulWidget {
  LoginView({super.key});

  @override
  State<LoginView> createState() => _LoginViewState();
}

class _LoginViewState extends State<LoginView> {
  final GlobalKey<FormState> _formkey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
```

```
 var isLoader = false;
 var authServices = AuthServices();
 Future<void> _submitform() async {
  if (_formkey.currentState!.validate()) {
    setState(() {
      isLoader = true;
    });

    var data = {
      "email": _emailController.text,
      "password": _passwordController.text,
    };

    await authServices.login(data, context);
    Navigator.of(context).pushReplacement(
        MaterialPageRoute(builder: (context) => Dashboard()));
    setState(() {
      isLoader = false;
    });
  }
}

 var appValidator = AppValidator();

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF000000),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
          key: _formkey,
          child: Column(
            children: [
              SizedBox(
                height: 180,
              ),
              SizedBox(
                width: 250,
                child: Text(
                  "Login Account",
```

```
    textAlign: TextAlign.center,
    style: TextStyle(
      color: Colors.white,
      fontSize: 28,
      fontWeight: FontWeight.bold),
  ),
),

TextFormField(
    controller: _emailController,
    keyboardType: TextInputType.emailAddress,
    style: TextStyle(color: Colors.white),
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration: _buildInputDecoration("Email", Icons.email),
    validator: appValidator.validateEmail),
SizedBox(
  height: 25,
),

TextFormField(
    controller: _passwordController,
    keyboardType: TextInputType.text,
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration: _buildInputDecoration("Password", Icons.lock),
    validator: appValidator.validatePassword),
SizedBox(
  height: 40.0,
),
SizedBox(
    height: 50,
    width: double.infinity,
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: Color(0xFFF75104)),
      onPressed: () {
        isLoader ? print("Loading") : _submitform();
      },
      // _submitform,
      child: isLoader
        ? Center(child: CircularProgressIndicator())
        : Text(
```

```
                    "Login",
                    style: TextStyle(
                        color: Colors.white, fontSize: 20),
                  ))),
            SizedBox(
              height: 20.0,
            ),
            TextButton(
               onPressed: () {
                 Navigator.push(
                   context,
                   MaterialPageRoute(builder: (context) => SignUpView()),
                 );
               },
               child: Text(
                 "Create new Account",
                 style: TextStyle(color: Color(0xFFF75104), fontSize: 20),
               ))
            // Text(
            //   "Login",
            //   style: TextStyle(color:Color(0xFFF75104),fontSize: 25 ),
            // )
          ],
        )),
    ),
  );
}

InputDecoration _buildInputDecoration(String label, IconData suffixIcon) {
  return InputDecoration(
     fillColor: Color(0xAA494A59),
     enabledBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Color(0x35949494))),
     focusedBorder:
        OutlineInputBorder(borderSide: BorderSide(color: Colors.white)),
     filled: true,
     labelStyle: TextStyle(color: Color(0xff949494)),
     labelText: label,
     suffixIcon: Icon(
       suffixIcon,
       color: Color(0xFF949494),
```

```
      ),
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)));
  }
}
```

Signup.dart

```
import 'package:flutter/material.dart';
import 'package:my_project/screens/dashboard.dart';
import 'package:my_project/screens/login_screen.dart';
import 'package:my_project/services/auth_service.dart';
import 'package:my_project/utils/appvalidator.dart';

class SignUpView extends StatefulWidget {
  SignUpView({super.key});

  @override
  State<SignUpView> createState() => _SignUpViewState();
}

class _SignUpViewState extends State<SignUpView> {
  final GlobalKey<FormState> _formkey = GlobalKey<FormState>();

  final _userNameController = TextEditingController();

  final _emailController = TextEditingController();

  final _phoneController = TextEditingController();

  final _passwordController = TextEditingController();

  var authServices = AuthServices();
  var isLoader = false;

  Future<void> _submitform() async {
    if (_formkey.currentState!.validate()) {
      setState(() {
        isLoader = true;
      });

      var data = {
        "username": _userNameController.text,
```

```
      "email": _emailController.text,
      "password": _passwordController.text,
      "phone": _phoneController.text,
      'remainingAmount': 0,
      'totalCredit': 0,
      'totalDebit': 0
    };

    await authServices.createUser(data, context);
    // Navigator.pushReplacement(
    //   context,
    //   MaterialPageRoute(builder: (context) => Dashboard()),
    // );
    setState(() {
      isLoader = false;
    }); // ScaffoldMessenger.of(_formkey.currentContext!).showSnackBar(
    //   const SnackBar(content: Text('Form submitted successfully')),
    // );
  }
}

var appValidator = AppValidator();

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF000000),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
          key: _formkey,
          child: Column(
            children: [
              SizedBox(
                height: 80.0,
              ),
              SizedBox(
                width: 250,
                child: Text(
                  "Create new Account",
                  textAlign: TextAlign.center,
```

```
    style: TextStyle(
        color: Colors.white,
        fontSize: 28,
        fontWeight: FontWeight.bold),
  ),
),
SizedBox(
  height: 16.0,
),
TextFormField(
    controller: _userNameController,
    style: TextStyle(color: Colors.white),
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration: _buildInputDecoration("UserName", Icons.person),
    validator: appValidator.validateUsername),
//   if (value!.isEmpty) {
//     return 'Please enter a username';
//   }
//   return null;
// }),
SizedBox(
  height: 16.0,
),
TextFormField(
    controller: _emailController,
    keyboardType: TextInputType.emailAddress,
    style: TextStyle(color: Colors.white),
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration: _buildInputDecoration("Email", Icons.email),
    validator: appValidator.validateEmail),
SizedBox(
  height: 16.0,
),
TextFormField(
    controller: _phoneController,
    keyboardType: TextInputType.phone,
    style: TextStyle(color: Colors.white),
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration:
        _buildInputDecoration("Phone Number", Icons.call),
    validator: appValidator.validatePhoneNumber),
```

```
SizedBox(
  height: 16.0,
),
TextFormField(
    controller: _passwordController,
    keyboardType: TextInputType.text,
    autovalidateMode: AutovalidateMode.onUserInteraction,
    decoration: _buildInputDecoration("Password", Icons.lock),
    validator: appValidator.validatePassword),
SizedBox(
  height: 40.0,
),
SizedBox(
    height: 50,
    width: double.infinity,
    child: ElevatedButton(
        style: ElevatedButton.styleFrom(
            backgroundColor: Color(0xFFF75104)),
        onPressed: () {
          isLoader ? print("Loading") : _submitform();
        },
        // _submitform,
        child: isLoader
            ? Center(child: CircularProgressIndicator())
            : Text(
                "Create",
                style: TextStyle(
                    color: Colors.white, fontSize: 20),
              ))),
SizedBox(
  height: 20.0,
),
TextButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => LoginView()),
      );
    },
    child: Text(
      "Login",
```

```
                style: TextStyle(color: Color(0xFFF75104), fontSize: 20),
              ))
            // Text(
            //   "Login",
            //   style: TextStyle(color:Color(0xFFF75104),fontSize: 25 ),
            // )
          ],
        )),
    ),
  );
 }

 InputDecoration _buildInputDecoration(String label, IconData suffixIcon) {
  return InputDecoration(
    fillColor: Color(0xAA494A59),
    enabledBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0x35949494))),
    focusedBorder:
      OutlineInputBorder(borderSide: BorderSide(color: Colors.white)),
    filled: true,
    labelStyle: TextStyle(color: Color(0xff949494)),
    labelText: label,
    suffixIcon: Icon(
     suffixIcon,
     color: Color(0xFF949494),
    ),
    border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)));
 }
}
```

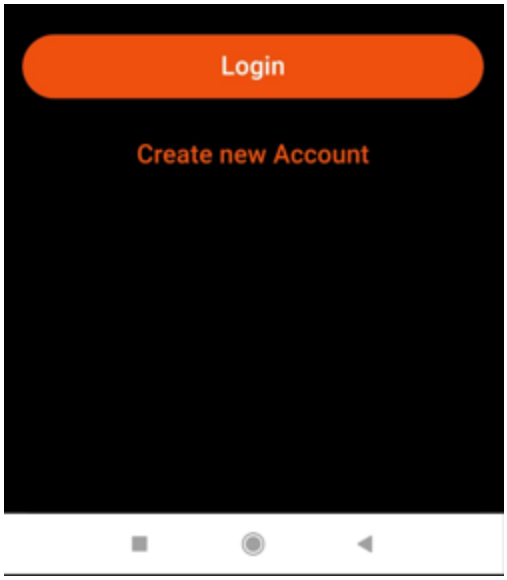
Homescreen.dart

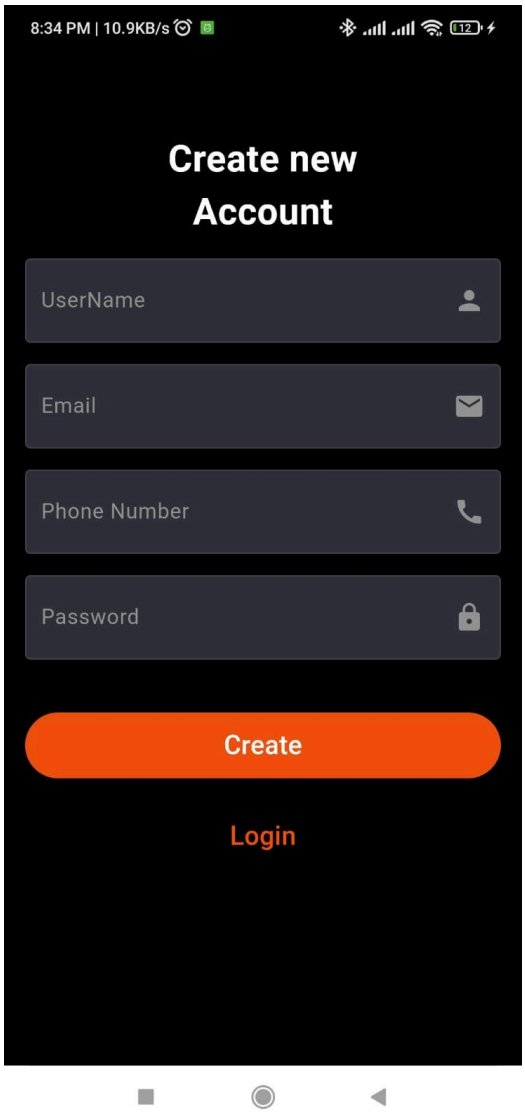
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';
import 'package:my_project/screens/login_screen.dart';
import 'package:my_project/widgets/add_transaction_form.dart';
import 'package:my_project/widgets/hero_card.dart';
import 'package:my_project/widgets/transactions_cards.dart';

// ignore_for_file:prefer_const_constructors
```

```
// ignore_for_file:prefer_const_literals_to_create_immutables
class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  var isLogoutLoading = false;
  logOut() async {
    setState(() {
      isLogoutLoading = true;
    });
    await FirebaseAuth.instance.signOut();
    Navigator.of(context)
        .pushReplacement(MaterialPageRoute(builder: (context) => LoginView()));

    setState(() {
      isLogoutLoading = false;
    });
  }

  _dialoBuilder(BuildContext context) {
    return showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            content: AddTransactionForm(),
          );
        });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.blue.shade900,
        onPressed: (() {
          _dialoBuilder(context);
        }),
```

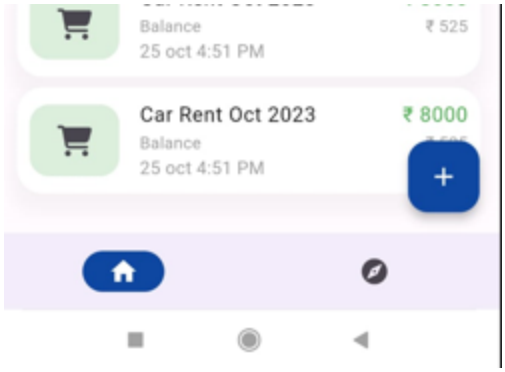
```
        child: Icon(
          Icons.add,
          color: Colors.white,
        ),
      ),
      appBar: AppBar(
        backgroundColor: Colors.blue.shade900,
        title: Text(
          "Hello",
          style: TextStyle(color: Colors.white),
        ),
        actions: [
          IconButton(
              onPressed: () {
                logOut();
              },
              icon: isLogoutLoading
                  ? CircularProgressIndicator()
                  : Icon(
                      Icons.exit_to_app,
                      color: Colors.white,
                    ))
        ],
      ),
      body: Column(
        children: [
          HeroCard(),
          TransactionsCard(),

        ],
      ),
    );
  }
}
```

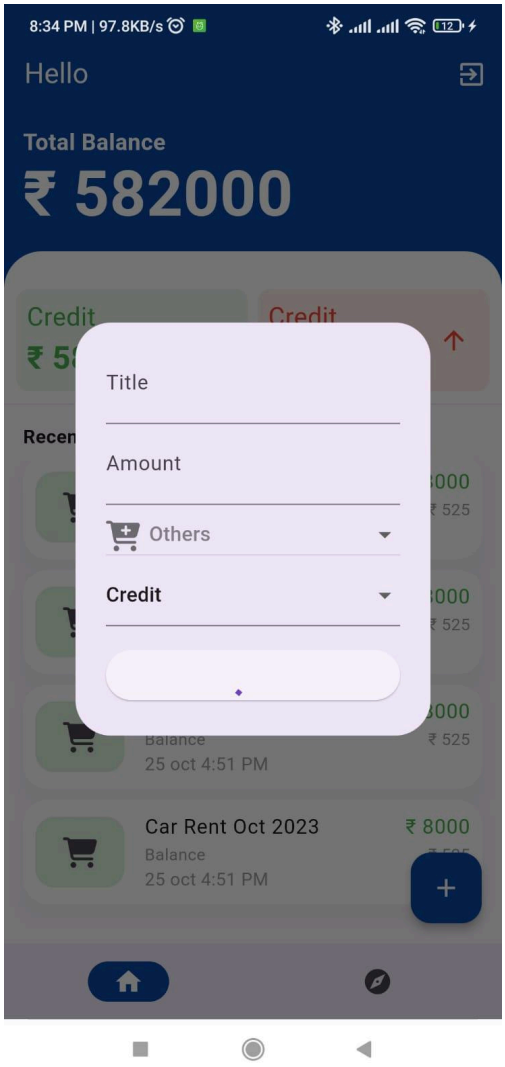
Title:Budgettracker **Roll No. 46**





Title:Budgettracker **Roll No. 46**





Conclusion:In this flutter routing is being understood and implemented successfully and also the routing in flutter is easy to understand and came to know about how the screen are being routed.

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Experiment no 6

Aim:To Connect Flutter UI with FireBase database

Theory:

Prerequisites

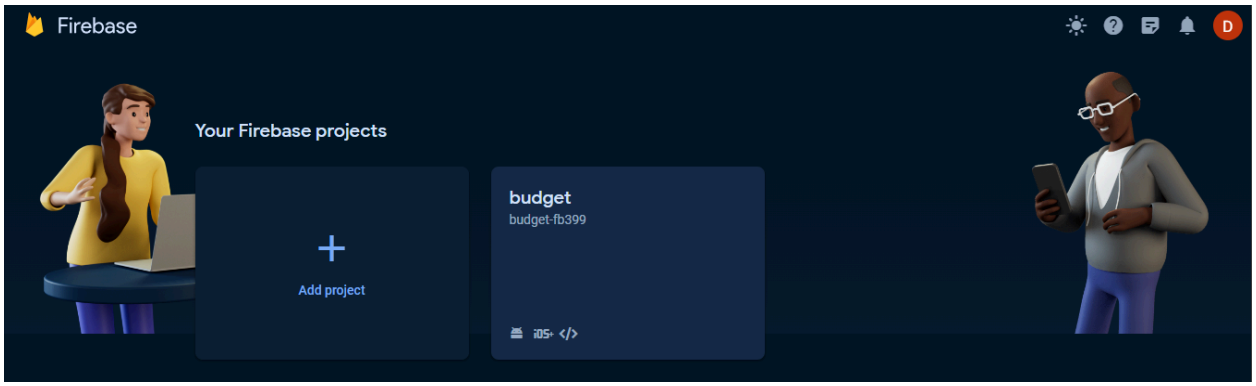
To complete this tutorial, you will need:

- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
  - Flutter and Dart plugins installed for Android Studio.
  - Flutter extension installed for Visual Studio Code.

1)Create a Firebase Project:

First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard,

select the Create new project button and give it a name:



2)Go to the Firebase Console and create a new project.

Add your Flutter app to the Firebase project:

Register your app in the Firebase project, and follow the instructions to download the configuration files (google-services.json for Android, GoogleService-Info.plist for iOS).

The most important thing here is to match up the Android package name that you choose here with the one

inside of our application.

The structure consists of at least two segments. A common pattern is to use a domain name, a company

name, and the application name:budget

3) Add Firebase to your Flutter project:

Add Dependencies:

```
dependencies:
  flutter:
```

```
  sdk: flutter
 firebase_core: 2.24.2
 cloud_firestore: 4.14.0
 firebase_auth: 4.16.0
```

Code:

Db.dart

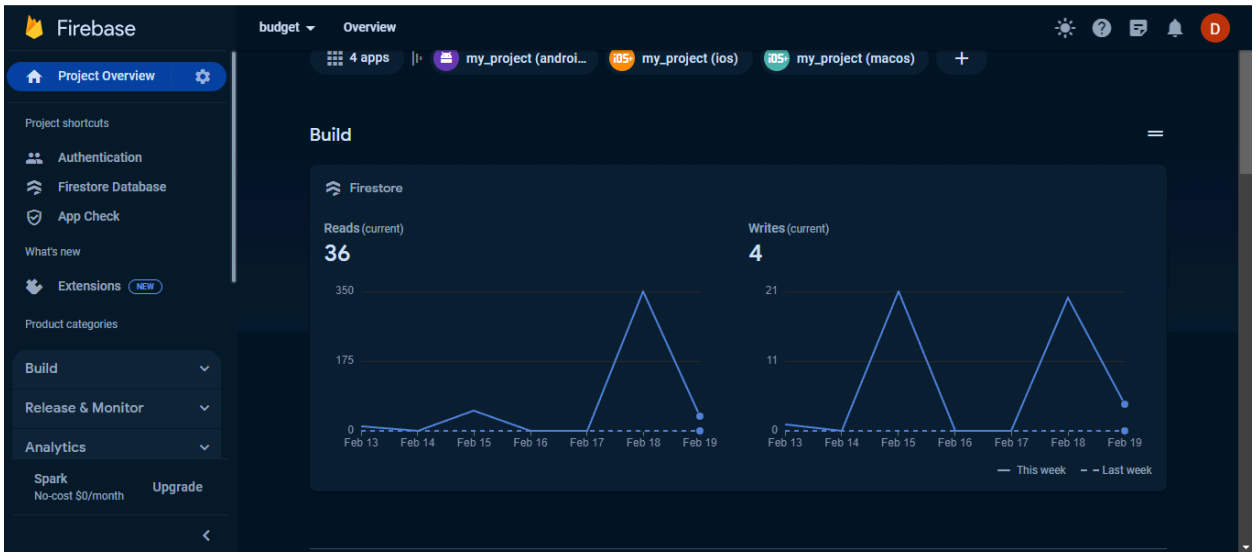
```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class Db {
  CollectionReference users = FirebaseFirestore.instance.collection('users');

  Future<void> addUser(data, context) async {
    final userId = FirebaseAuth.instance.currentUser!.uid;
    await users
        .doc(userId)
        .set(data)
        .then((value) => print("User Added"))
        .catchError((error) {
      showDialog(
          context: context,
          builder: (context) {
            return AlertDialog(
              title: Text("Sign up Failed"),
              content: Text(error.toString()),
            );
          });
    });
  }
}
```

Auth_services.dart

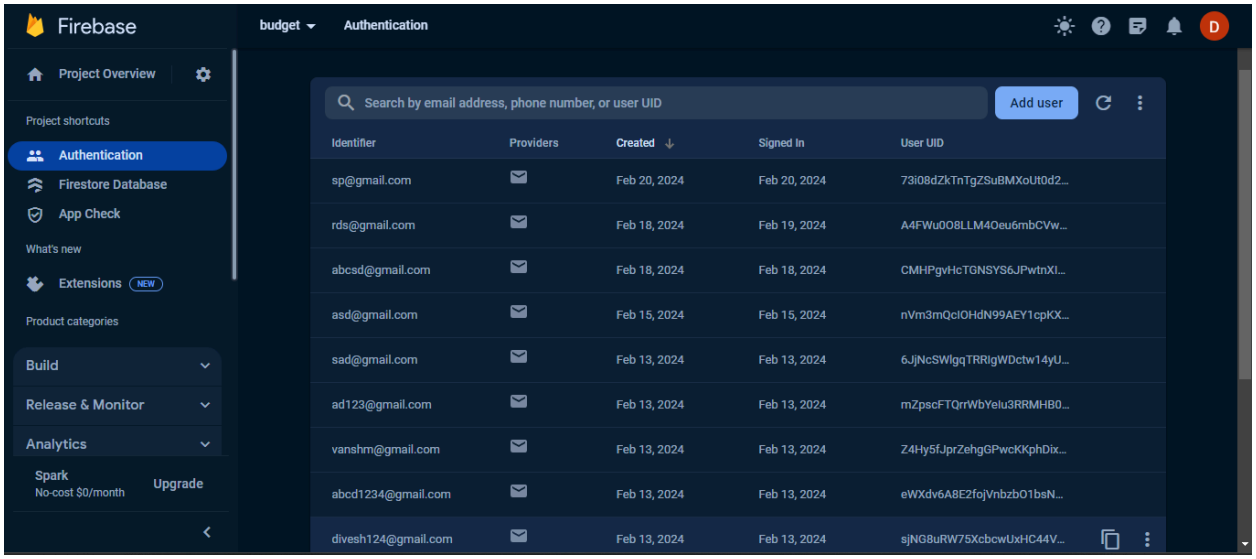
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:my_project/screens/dashboard.dart';
import 'package:my_project/services/db.dart';

class AuthServices {
  var db = Db();
  createUser(data, context) async {
    try {
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: data['email'],
        password: data['password'],
      );
      await db.addUser(data, context);
      Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => Dashboard()));
    } catch (e) {
      showDialog(
          context: context,
          builder: (context) {
            return AlertDialog(
              title: Text("Sign up Failed"),
              content: Text(e.toString()),
            );
          });
    }
  }

  login(data, context) async {
    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: data['email'],
        password: data['password'],
      );
      Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => Dashboard()));
    } catch (e) {
      showDialog(
          context: context,
```

```
        builder: (context) {
          return AlertDialog(
            title: Text("Login Error"),
            content: Text(e.toString()),
          );
        });
    }
  }
}
```

Output:

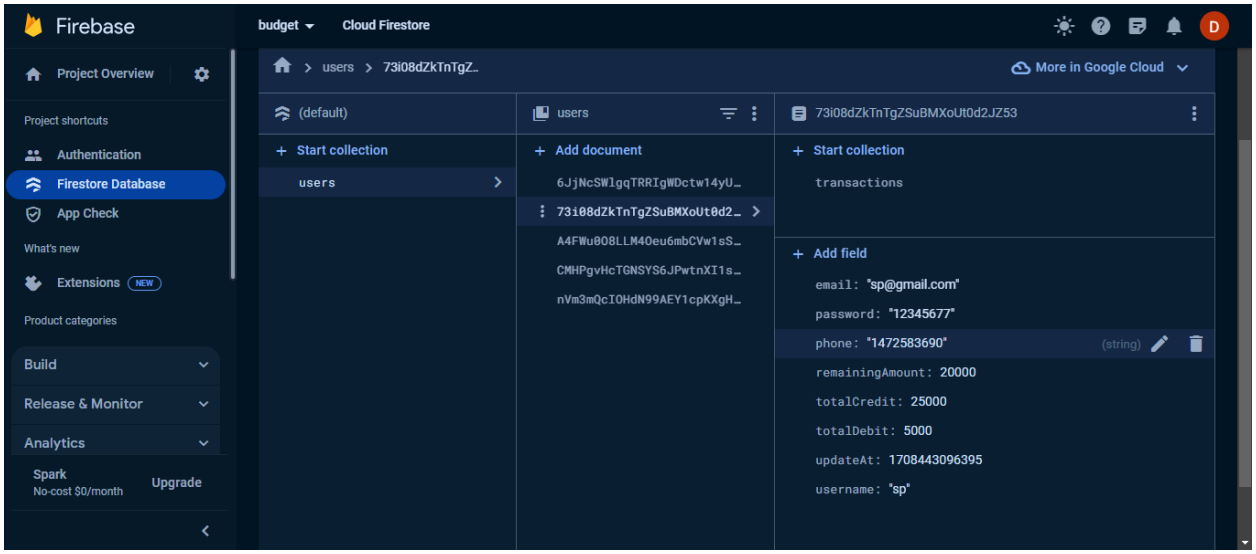


Authentications:

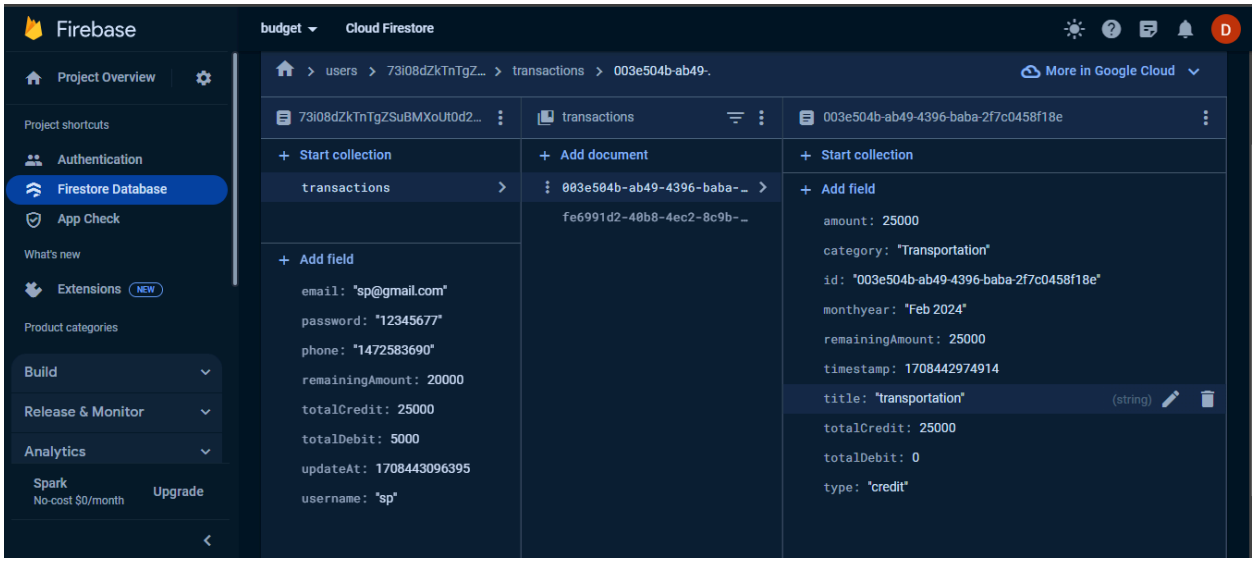


Users and transactions:

Title:Budgettracker **Roll No. 46**



Transactions:



Conclusion:

In this experiment, we have successfully connected firebase database and authenticated using signin and email and password and also added the users and their transactions successfully

MAD & PWA Lab
Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment No 7

Aim:- To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.

Theory:-

Regular Web App
A regular web app is a website that is designed to be accessible on all mobile devices such that the content gets fit as per the device screen. It is designed using a web technology stack (HTML, CSS, JavaScript, Ruby, etc.) and operates via a browser. They offer various native-device features and functionalities. However, it entirely depends on the browser the user is using. In other words, it might be possible that you can access a native-device feature on Chrome but not on Safari or Mozilla Firefox because the browsers are incompatible with that feature.

Progressive Web App
Progressive Web App (PWA) is a regular web app, but some extras enable it to deliver an excellent user experience. It is a perfect blend of desktop and mobile application experience to give both platforms to the end-users.

Difference between PWAs vs. Regular Web Apps:
A Progressive Web is different and better than a Regular Web app with features like:
1. Native Experience
Though a PWA runs on web technologies (HTML, CSS, JavaScript) like a Regular web app, it gives user experience like a native mobile application. It can use most native device features, including push notifications, without relying on the browser or any other entity. It offers a seamless and integrated user experience that it is quite tough for one to differentiate between a PWA and a Native application by considering its look and feel.

2. Ease of Access
Unlike other mobile apps, PWAs do not demand longer download time and make memory space available for installing the applications. The PWAs can be shared and installed by a link, which cuts down the number of steps to install

and use. These applications can easily keep an app icon on the user's home screen, making the app easily accessible to the users and helps the brands remain in the users' minds, and improving the chances of interaction.

3. Faster Services
PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

4. Engaging Approach
As already shared, the PWAs can employ push notifications and other native device features more efficiently. Their interaction does not depend on the browser user uses. This eventually improves the chances of notifying the user regarding your services, offers, and other options related to your brand and keeping them hooked to your brand. In simpler words, PWAs let you maintain the user engagement and retention rate.

5. Updated Real-Time Data Access
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed.
In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.
6. Discoverable
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

7. Lower Development Cost
Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more cost-effective than native mobile applications while offering the same set of functionalities.

Pros and cons of the Progressive Web App
The main features are:

Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.

Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.

App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.

Updated — Information is always up-to-date thanks to the data update process offered by service workers.

Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.

Searchable — They are identified as “applications” and are indexed by search engines.

Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

Linkable — Easily shared via URL without complex installations.

Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Weaknesses refer to:
IOS support from version 11.3 onwards;
Greater use of the device battery;
Not all devices support the full range of PWA features (same speech for iOS and Android operating systems);
It is not possible to establish a strong re-engagement for iOS users (URL scheme, standard web notifications);
Support for offline execution is however limited;
Lack of presence on the stores (there is no possibility to acquire traffic from that channel);
There is no “body” of control (like the stores) and an approval process;
Limited access to some hardware components of the devices;
Little flexibility regarding “special” content for users (eg loyalty programs, loyalty, etc.).

Code:

```
<!DOCTYPE html>
<html lang="en">

<head>
 <meta charset="utf-8">
 <meta name="viewport"content="width=device-width,initial-scale=1">
 <meta http-equiv="X-UA-Compatible"content="ie=edge">
 <meta name="theme-color" content="#4285f4">

 <link rel="manifest" href="manifest.json">
 <script src="service-worker.js"></script>
 <title> shopping center</title>

 <!--
  - favicon
 -->
 <link rel="shortcut icon" href="./favicon.svg" type="image/svg+xml">
```

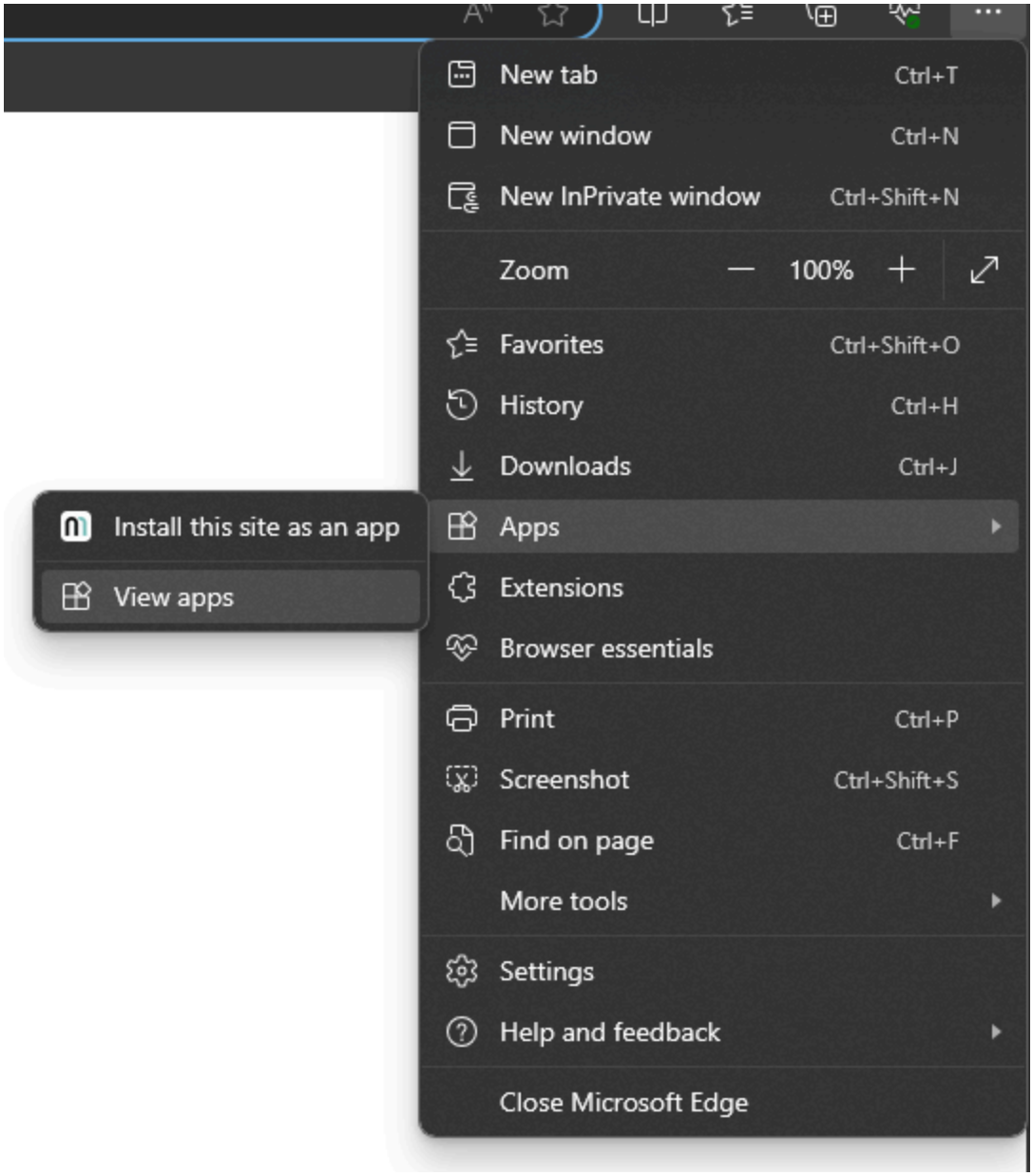
```
  <!--
    - custom css link
  -->
  <link rel="stylesheet" href="./assets/css/style.css">

  <!--
    - google font link
  -->
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Jost:wght@400;500;600;700&di
splay=swap" rel="stylesheet">
</head>

<body>

  <!--
    - #HEADER
  -->

  <header class="header" data-header>
    <div class="container">

      <div class="overlay" data-overlay></div>

      <div class="header-search">
        <input type="search" name="search" placeholder="Search Product..."
class="input-field">

        <button class="search-btn" aria-label="Search">
```

```
      <ion-icon name="search-outline"></ion-icon>
    </button>
  </div>

  <a href="#" class="logo">
    <img src="./assets/images/logo.svg" alt="Casmart logo" width="130"
height="31">
  </a>

  <div class="header-actions">

    <button class="header-action-btn">
      <ion-icon name="person-outline" aria-hidden="true"></ion-icon>

      <p class="header-action-label">Sign in</p>
    </button>

    <button class="header-action-btn">
      <ion-icon name="search-outline" aria-hidden="true"></ion-icon>

      <p class="header-action-label">Search</p>
    </button>

    <button class="header-action-btn">
      <ion-icon name="cart-outline" aria-hidden="true"></ion-icon>

      <p class="header-action-label">Cart</p>

      <div class="btn-badge green" aria-hidden="true">3</div>
    </button>

    <button class="header-action-btn">
```

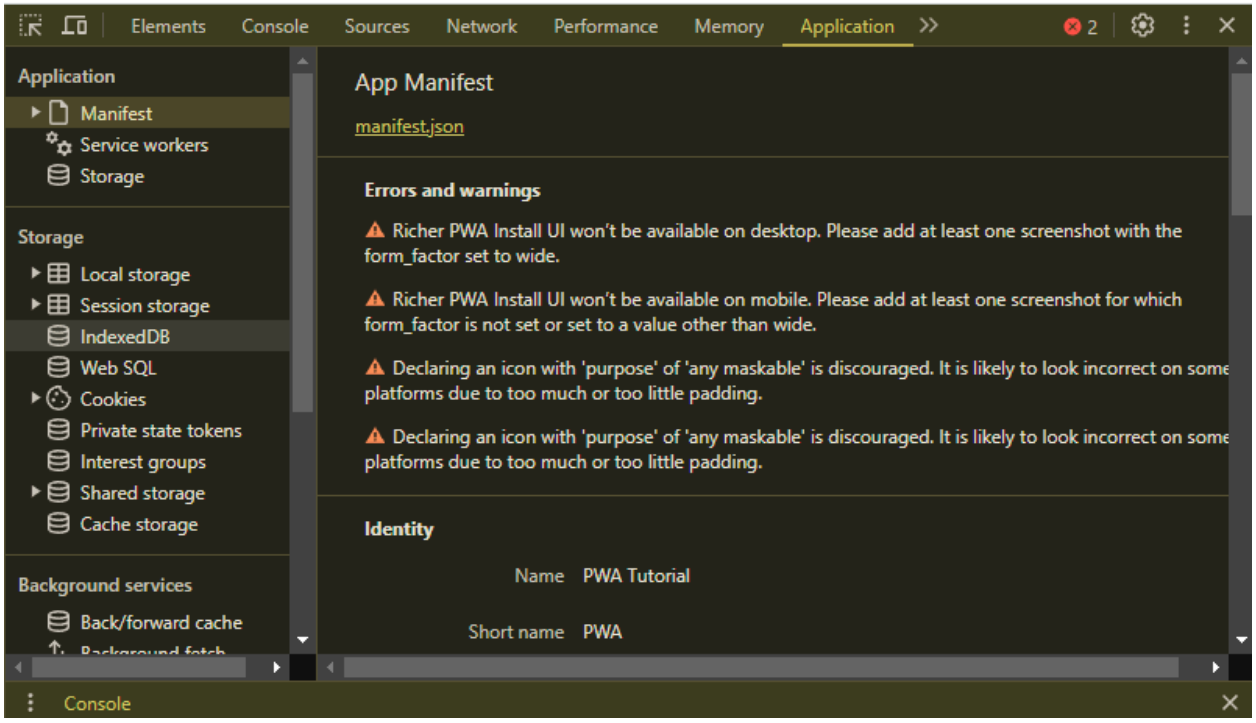
```
        <ion-icon name="heart-outline" aria-hidden="true"></ion-icon>

        <p class="header-action-label">Wishlisht</p>

        <div class="btn-badge" aria-hidden="true">2</div>
      </button>

    </div>

    <button class="nav-open-btn" data-nav-open-btn aria-label="Open Menu">
      <span></span>
      <span></span>
      <span></span>
    </button>

    <nav class="navbar" data-navbar>

      <div class="navbar-top">

        <a href="#" class="logo">
          <img src="./assets/images/logo.svg" alt="Casmart logo" width="130"
height="31">
        </a>

        <button class="nav-close-btn" data-nav-close-btn aria-label="Close
Menu">
          <ion-icon name="close-outline"></ion-icon>
        </button>

      </div>

      <ul class="navbar-list">
```

```
      <li>
        <a href="#home" class="navbar-link">Home</a>
      </li>

      <li>
        <a href="#" class="navbar-link">Shop</a>
      </li>

      <li>
        <a href="#" class="navbar-link">About</a>
      </li>

      <li>
        <a href="#blog" class="navbar-link">Blog</a>
      </li>

      <li>
        <a href="#" class="navbar-link">Contact</a>
      </li>

    </ul>

  </nav>

 </div>
</header>
```

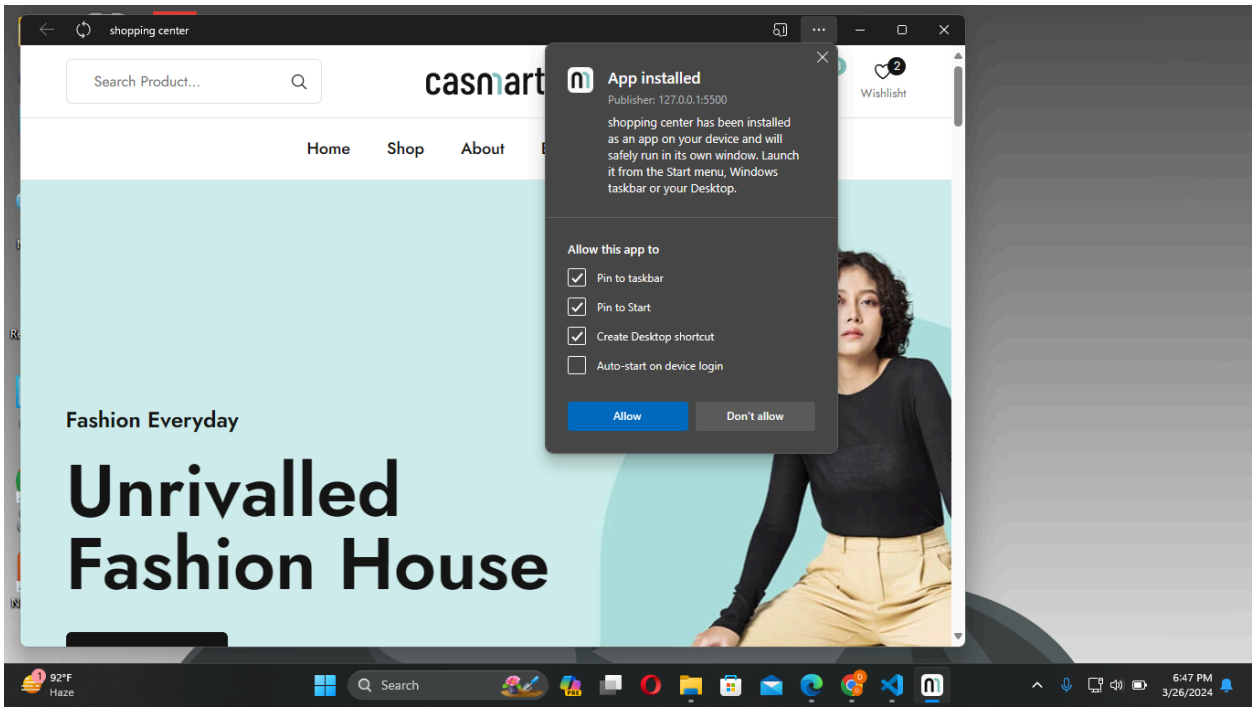
After this add or open the link in microsoft edge


New tab Ctrl+T
New window Ctrl+N
New InPrivate window Ctrl+Shift+N
Zoom 100%
Favorites Ctrl+Shift+O
History Ctrl+H
Downloads Ctrl+J
Apps
Extensions
Browser essentials
Print Ctrl+P
Screenshot Ctrl+Shift+S
Find on page Ctrl+F
More tools
Settings
Help and feedback
Close Microsoft Edge
Install this site as an app
View apps

Title:Budgettracker **Roll No. 46**

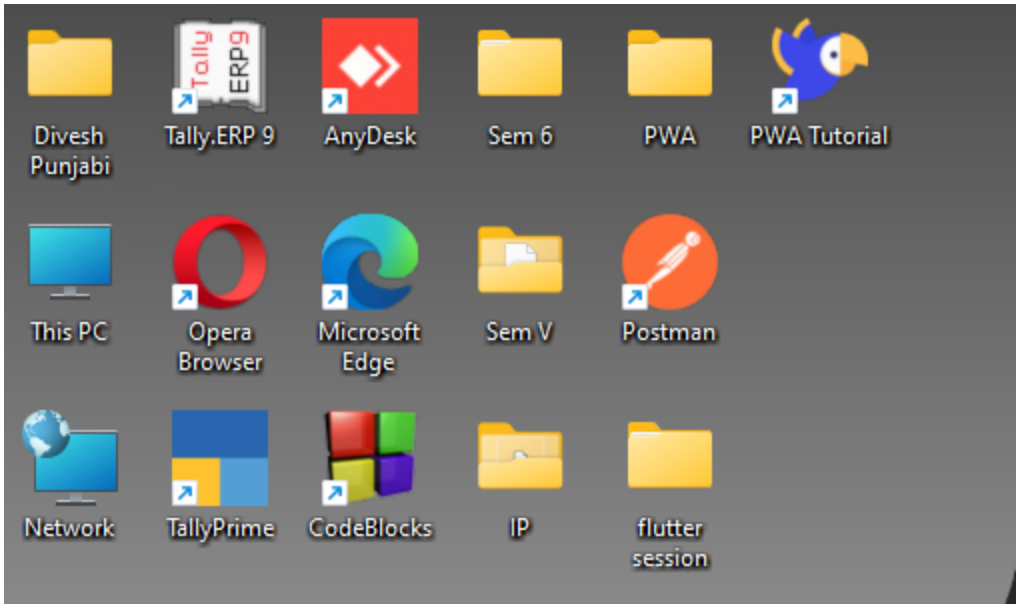


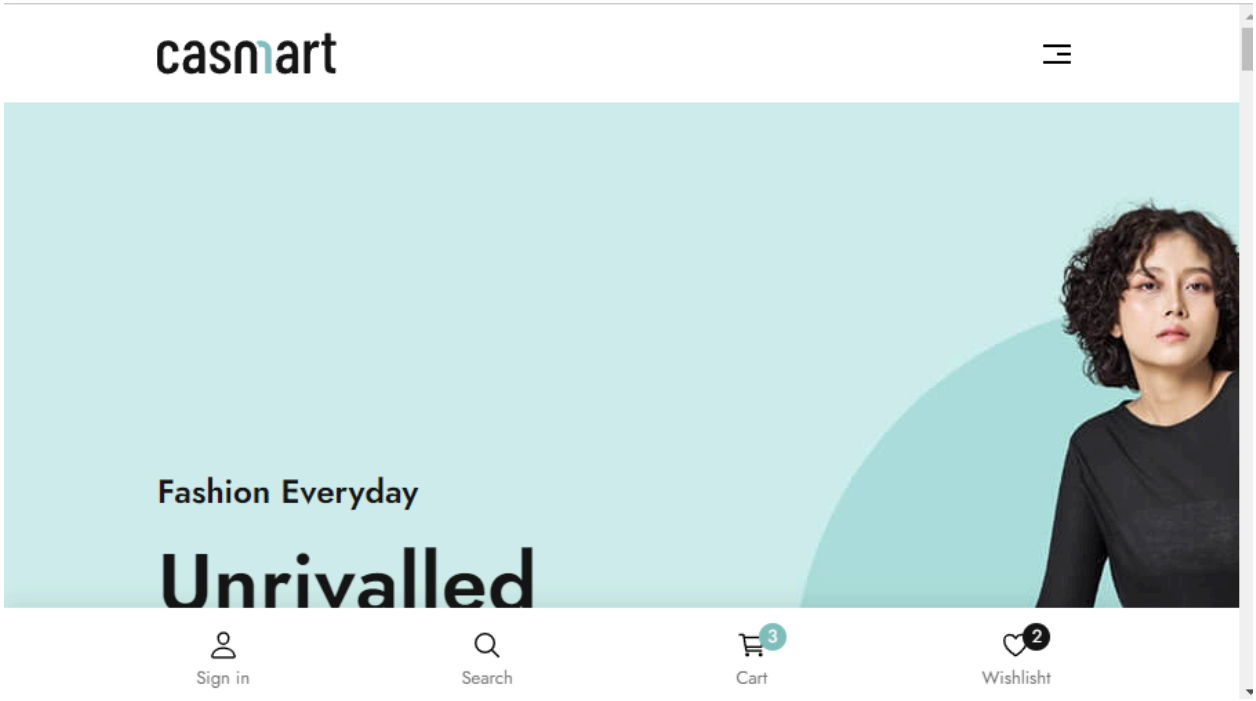
Starting the Server:



After that the app is being installed on the Desktop

Title:Budgettracker **Roll No. 46**





Conclusion:

Hence We wrote meta data of our Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. And it is currently added Successfully on the Desktop

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment number
08

Aim: To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA.

Theory:

Service Worker
Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

What can we do with Service Workers?

- You can dominate Network Traffic
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.

- You can Cache
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.

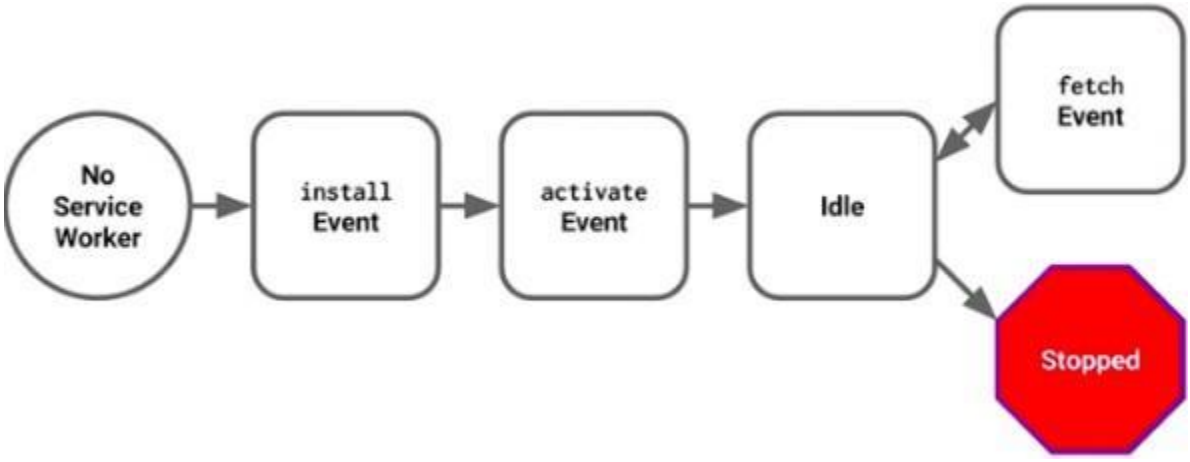
- You can manage Push Notifications
  You can manage push notifications with Service Worker and show any information message to the user.
- You can Continue
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

- You can't access the Window
  You can't access the window, therefore, You can't manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.

- You can't work it on 80 Port
  Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle



A service worker goes through three steps in its life cycle:

- Registration
- Installation
- Activation

Registration
To install a service worker, you need to register it in your main JavaScript

code. Registration tells the browser where your service worker is located, and to start installing it in the background. Let's look at an example:

main.js

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/service-wo
  rker.js')
  .then(function(registration) {
    console.log('Registration successful, scope is:', registration.scope);
  })
  .catch(function(error) {
    console.log('Service worker registration failed, error:', error);
  });
}
```

This code starts by checking for browser support by examining navigator.serviceWorker. The service worker is then registered with navigator.serviceWorker.register, which returns a promise that resolves when the service worker has been successfully registered. The scope of the service worker is then logged with registration.scope. If the service worker is already installed, navigator.serviceWorker.register returns the registration object of the currently active service worker.

The scope of the service worker determines which files the service worker controls, in other words, from which path the service worker will intercept requests. The default scope is the location of the service worker file, and extends to all directories below. So if service-worker.js is located in the root directory, the service worker will control requests from all files at this domain.

You can also set an arbitrary scope by passing in an additional parameter when

registering. For example: main.js

```
navigator.serviceWorker.register('/service-worker.j
  s', { scope: '/app/'
});
```

In this case we are setting the scope of the service worker to /app/, which means the service worker will control requests from pages like /app/, /app/lower/ and /app/lower/lower, but not from pages like /app or /, which are higher.

If you want the service worker to control higher pages e.g. /app (without the trailing slash) you can indeed change the scope option, but you'll also need to set the Service-Worker-Allowed HTTP Header in your server config for the request serving the

service worker script.

main.js

```
navigator.serviceWorker.register('/app/service-wo
  rker.js', { scope: '/app'
});
```

Installation

Once the browser registers a service worker, installation can be attempted. This occurs if the service worker is considered to be new by the browser, either because the site currently doesn't have a registered service worker, or because there is a byte difference between the new service worker and the previously installed one.

A service worker installation triggers an install event in the installing service worker. We can include an install event listener in the service worker to perform some task when the service worker installs. For instance, during the install, service workers can precache parts of a web app so that it loads instantly the next time a user opens it (see caching the application shell). So, after that first load, you're going to benefit from instant repeat loads and your time to interactivity is going to be even better in those cases. An example of an installation event listener looks like this:

Manifest.json

```
{
    "name":"PWA Tutorial",
    "short_name":"PWA",
    "start_url":"index.html",
    "display":"standalone",
    "background_color":"#5900b3",
    "theme_color":"black",
    "scope": ".",
    "description":"This is a PWA tutorial.",
    "icons":[
    {
    "src":"./assets/images/192x192.png",
    "sizes":"192x192",
    "type":"image/png",
  "purpose": "any maskable"
    },
    {
    "src":"./assets/images/512x512.png",
    "sizes":"512x512",
    "type":"image/png",
  "purpose": "any maskable"
    }
]
```

```
}
```

script.js

```
'use strict';

/**
 * navbar toggle
 */

const overlay = document.querySelector("[data-overlay]");
const navOpenBtn = document.querySelector("[data-nav-open-btn]");
const navbar = document.querySelector("[data-navbar]");
const navCloseBtn = document.querySelector("[data-nav-close-btn]");

const navElemArr = [overlay, navOpenBtn, navCloseBtn];

for (let i = 0; i < navElemArr.length; i++) {
  navElemArr[i].addEventListener("click", function () {
    navbar.classList.toggle("active");
    overlay.classList.toggle("active");
  });
}



/**
 * add active class on header when scrolled 200px from top
 */

const header = document.querySelector("[data-header]");

window.addEventListener("scroll", function () {
  window.scrollY >= 200 ? header.classList.add("active")
    : header.classList.remove("active");
})
```

service-worker.js

```
self.addEventListener("install", function (event) {
  event.waitUntil(preLoad());
  });
  self.addEventListener("fetch", function (event) {
```

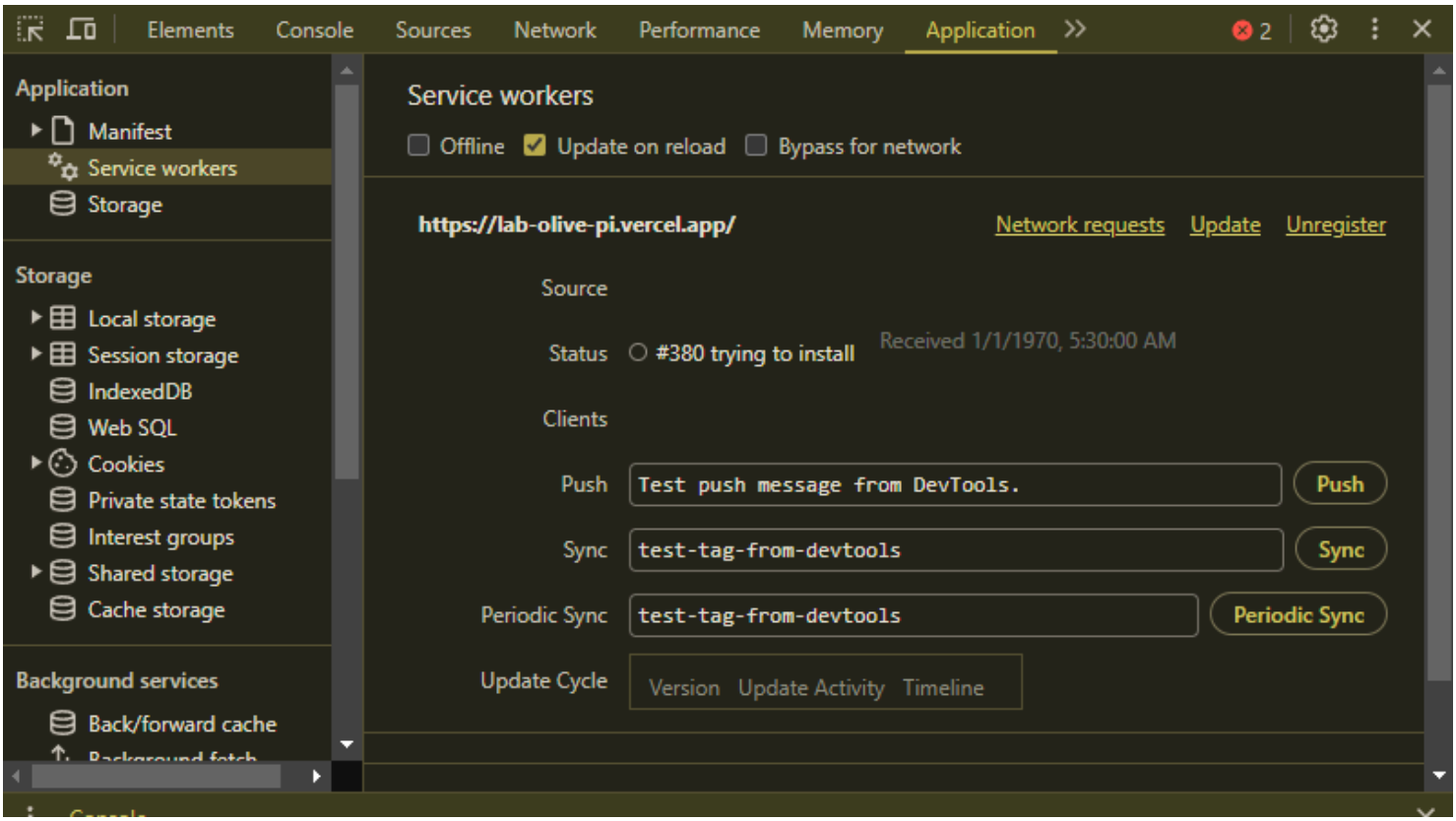
```
event.respondWith(
checkResponse(event.request).catch(function () {
console.log("Fetch from cache successful!");
return returnFromCache(event.request);
})
);
console.log("Fetch successful!");
event.waitUntil(addToCache(event.requesst));
});
self.addEventListener("sync", (event) => {
if (event.tag === "syncMessage") {
console.log("Sync successful!");
}
});
self.addEventListener("push", function (event) {
if (event && event.data) {
try {
var data = event.data.json();
if (data && data.method === "pushMessage") {
console.log("Push notification sent");
self.registration.showNotification("Ecommerce website", {
body: data.message,
});
}
} catch (error) {
console.error("Error parsing push data:", error);
}
}
});
var preLoad = function () {
return caches.open("offline").then(function (cache) {
// caching index and important routes
return cache.addAll([
"/",
```

```
"/index.html",

"manifest.json",
"script.js",
"/css/main.css",
]);
});
};
var checkResponse = function (request) {
return new Promise(function (fulfill, reject) {
fetch(request)
.then(function (response) {
if (response.status !== 404) {
fulfill(response);
} else {
reject(new Error("Response not found"));
}
})
.catch(function (error) {
reject(error);
});
});
};
var returnFromCache = function (request) {
return caches.open("offline").then(function (cache) {
return cache.match(request).then(function (matching) {
if (!matching || matching.status == 404) {
return cache.match("offline.html");
} else {
return matching;
}
});
});
};
```

Title:Budgettracker **Roll No. 46**

```
var addToCache = function (request) {
return caches.open("offline").then(function (cache) {
return fetch(request).then(function (response) {
return cache.put(request, response.clone()).then(function () {
return response;
});
});
});
};
```

Output:



Conclusion:In this experiment, we have registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment no 9

Aim: To implement Service worker events like fetch, sync and push for E-commerce PWA.

Theory:
Service Worker
Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
- Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event
You can track and manage page network traffic with this event. You can check existing cache, manage “cache first” and “network first” requests and return a response that you want.

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.

- CacheFirst - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- NetworkFirst - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

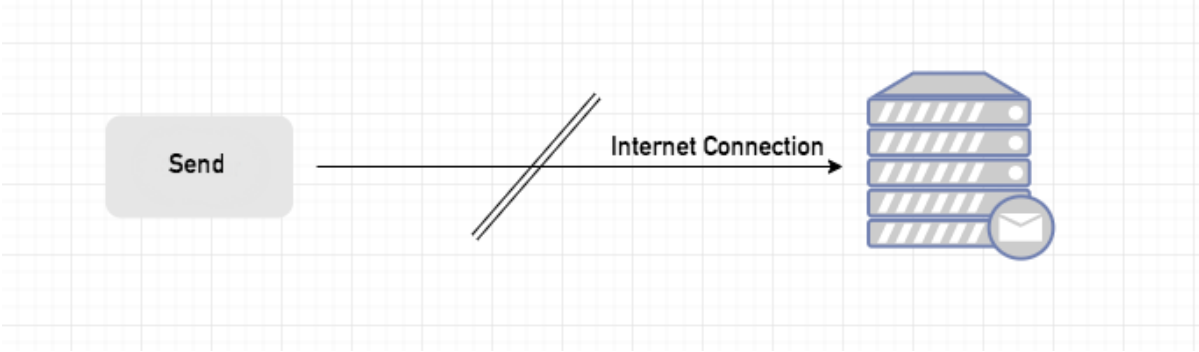
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

    if (url.origin === location.origin) {
        event.respondWith(cacheFirst(req));
    }
    else {
        event.respondWith(networkFirst(req));
    }
});

async function cacheFirst(req) {
    return await caches.match(req) || fetch(req);
}

async function networkFirst(req) {
    const cache = await caches.open("pwa-dynamic");
    try {
        const res = await fetch(req);
        cache.put(req, res.clone());
        return res;
    } catch (error) {
        const cachedResponse = await cache.match(req);
        return cachedResponse || await caches.match("./noconnection.json");
    }
}
```

Sync Event

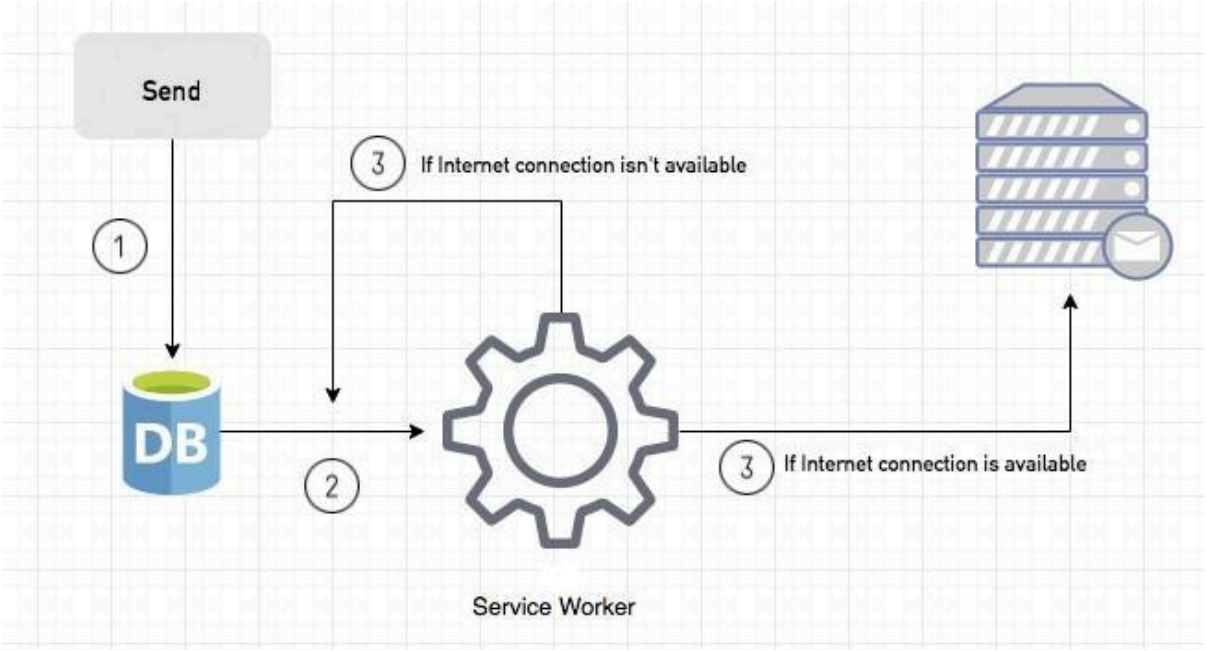
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button.

Here is a job for the Background Sync.

The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.



Here, you can create any scenario for yourself. A sample is in the following for this



case.

1. When we click the “send” button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. If the Internet connection is available, all email content will be read and sent to Mail Server.

   If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

   You can see the working process within the following code block.

Event Listener for Background Sync Registration

```
document.querySelector("button").addEventListener("click", async () => {
    var swRegistration = await navigator.serviceWorker.register("sw.js");
    swRegistration.sync.register("helloSync").then(function () {
        console.log("helloSync success [main.js]");
    });
});
```

Event Listener for sw.js

```
self.addEventListener('sync', event => {
    if (event.tag == 'helloSync') {
        console.log("helloSync [sw.js]");
    }
});
```

Push Event

This is the event that handles push notifications that are received from the server. You can apply any method with received data.

We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has “method” and “message” properties. If the method value is “pushMessage”, we open the information notification with the “message” property.

```
self.addEventListener('push', event => {
    if (event && event.data) {
        var data = event.data.json();
        if (data.method === "pushMessage") {
```

You can use Application Tab from Chrome Developer Tools for testing push notification.

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
  caches.open(staticCacheName).then(function (cache) {
  return cache.addAll(["/"]);
  })
);
});

self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
  caches.match(event.request).then(function (response) {
  return response || fetch(event.request);
  })
);
});
service-worker.js

//serviceworker
self.addEventListener("install", function (event) {
  event.waitUntil(preLoad());
  });
  self.addEventListener("fetch", function (event) {
  event.respondWith(
  checkResponse(event.request).catch(function () {
  console.log("Fetch from cache successful!");
  return returnFromCache(event.request);
  })
  );
  console.log("Fetch successful!");
  event.waitUntil(addToCache(event.requesst));
  });
  self.addEventListener("sync", (event) => {
  if (event.tag === "syncMessage") {
  console.log("Sync successful!");
  }
```

```
});
self.addEventListener("push", function (event) {
if (event && event.data) {
try {
var data = event.data.json();
if (data && data.method === "pushMessage") {
console.log("Push notification sent");
self.registration.showNotification("Ecommerce website", {
body: data.message,
});
}
} catch (error) {
console.error("Error parsing push data:", error);
}
}
});
var preLoad = function () {
return caches.open("offline").then(function (cache) {
// caching index and important routes
return cache.addAll([
"/",
"/index.html",

"manifest.json",
"script.js",
"/css/main.css",
]);
});
};
var checkResponse = function (request) {
return new Promise(function (fulfill, reject) {
fetch(request)
.then(function (response) {
if (response.status !== 404) {
fulfill(response);
} else {
reject(new Error("Response not found"));
}
})
.catch(function (error) {
```

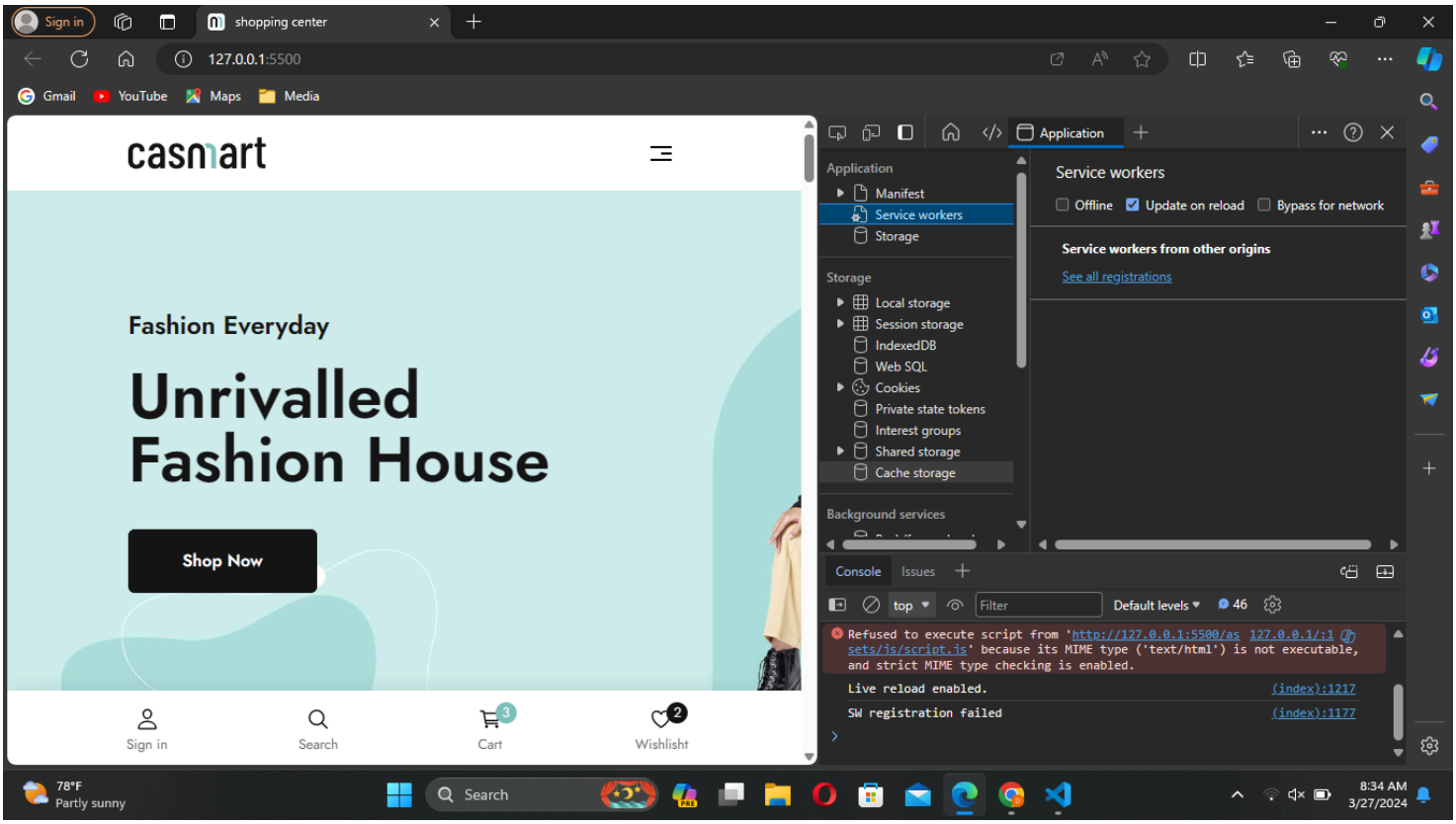
```
  reject(error);
  });
  });
  };
  var returnFromCache = function (request) {
  return caches.open("offline").then(function (cache) {
  return cache.match(request).then(function (matching) {
  if (!matching || matching.status == 404) {
  return cache.match("offline.html");
  } else {
  return matching;
  }
  });
  });
  };
  var addToCache = function (request) {
  return caches.open("offline").then(function (cache) {
  return fetch(request).then(function (response) {
  return cache.put(request, response.clone()).then(function () {
  return response;
  });
  });
  });
  };
```

Index.html

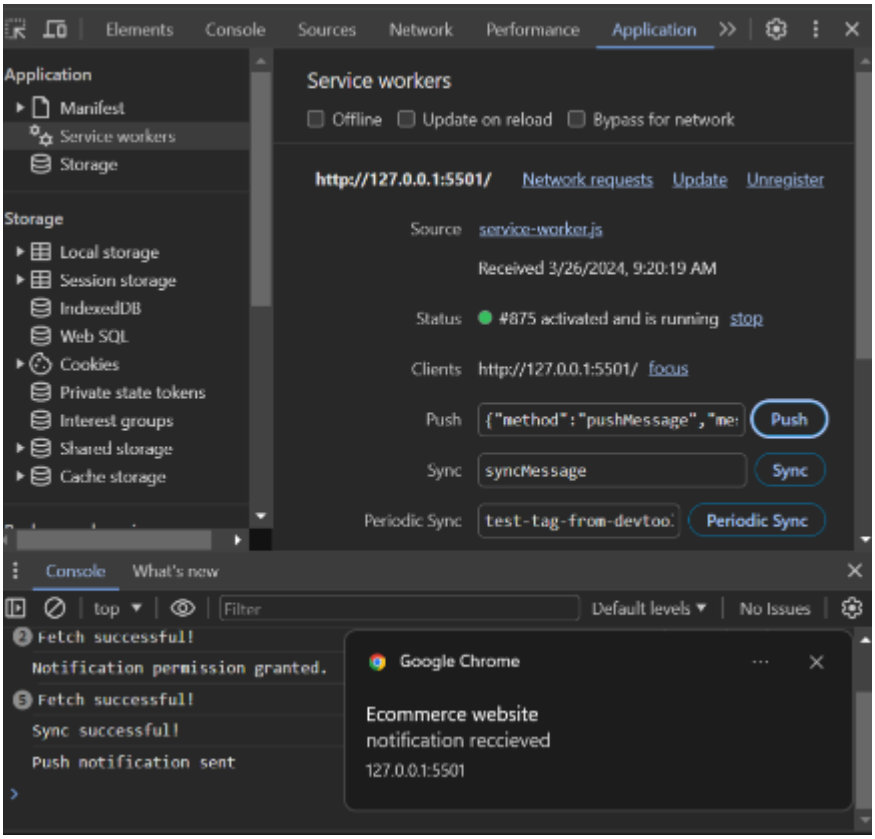
```
<script src="./assets/js/script.js"></script>
 <script>
  window.addEventListener('load', () => {
  registerSW();
  });

  // Register the Service Worker
  async function registerSW() {
  if ('serviceWorker' in navigator) {
    try {
    await navigator
        .serviceWorker
        .register('serviceworker.js');
    }
```

```
    catch (e) {
    console.log('SW registration failed');
    }
  }
  }
</script>

<!--
  - ionicon link
-->
<script type="module" src="https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.esm.js"></script>
<script nomodule src="https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.js"></script>

</body>

</html>
```

Output:



Title:Budgettracker **Roll No. 46**



Conclusion:In this experiment, we have successfully implemented service worker events like fetch, sync and push for
E-commerce PWA and found out output for above implementation.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment no 10

Aim:To study and implement deployment of Ecommerce PWA to GitHub Pages.

Theory:

GitHub Pages

Public web pages are freely hosted and easily published. Public webpages hosted directly from your GitHub repository. Just edit, push, and your changes are live.

GitHub Pages provides the following key features:

1. Blogging with Jekyll
2. Custom URL
3. Automatic Page Generator

Reasons for favoring this over Firebase:

1. Free to use
2. Right out of github
3. Quick to set up

GitHub Pages is used by Lyft, CircleCI, and HubSpot.
GitHub Pages is listed in 775 company stacks and 4401 developer stacks.

Pros

1. Very familiar interface if you are already using GitHub for your projects.
2. Easy to set up. Just push your static website to the gh-pages branch and your website is ready.
3. Supports Jekyll out of the box.
4. Supports custom domains. Just add a file called CNAME to the root of your site, add an A record in the site's DNS configuration, and you are done.

Cons

1. The code of your website will be public, unless you pay for a private repository.

2. Currently, there is no support for HTTPS for custom domains. It's probably coming soon though.
3. Although Jekyll is supported, plug-in support is rather spotty.

Firebase

The Realtime App Platform. Firebase is a cloud service designed to power real-time, collaborative applications. Simply add the Firebase library to your application to gain access to a shared data structure; any changes you make to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.

Some of the features offered by Firebase are:

1. Add the Firebase library to your app and get access to a shared data structure. Any changes made to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.
2. Firebase apps can be written entirely with client-side code, update in real-time out-of-the-box, interoperate well with existing services, scale automatically, and provide strong data security.
3. Data Accessibility- Data is stored as JSON in Firebase. Every piece of data has its own URL which can be used in Firebase's client libraries and as a REST endpoint. These URLs can also be entered into a browser to view the data and watch it update in real-time.

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase
Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

Pros

1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.

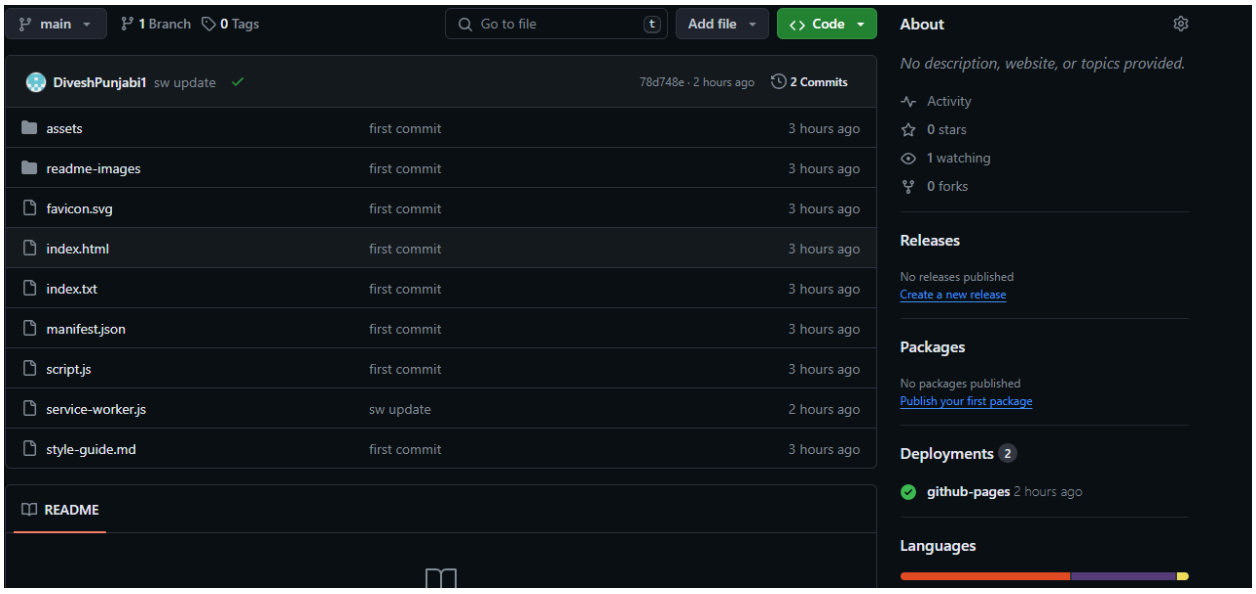
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Cons

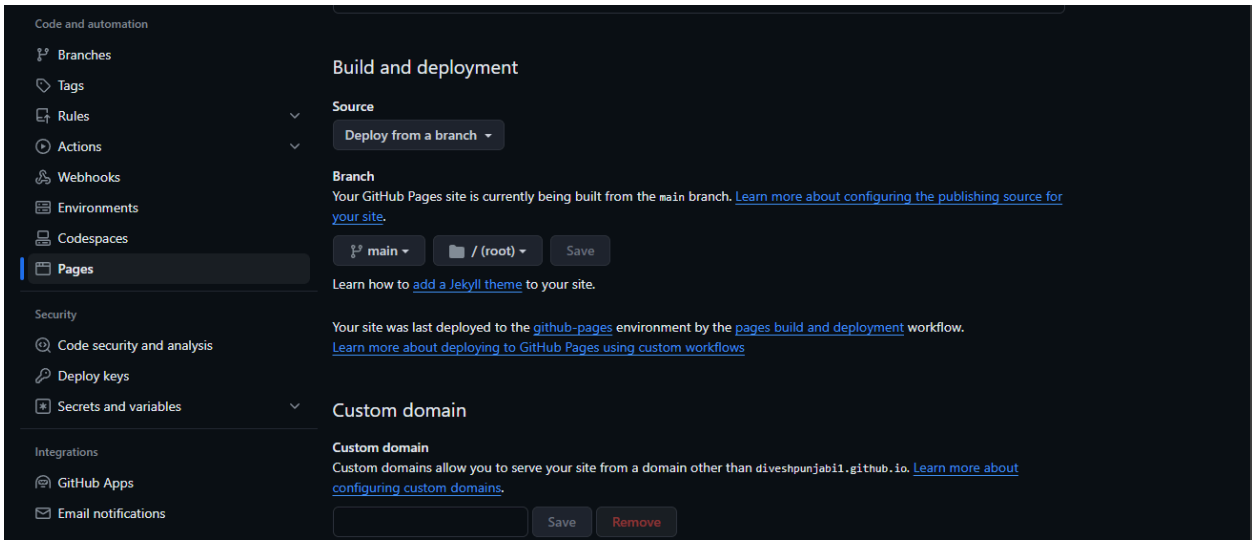
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

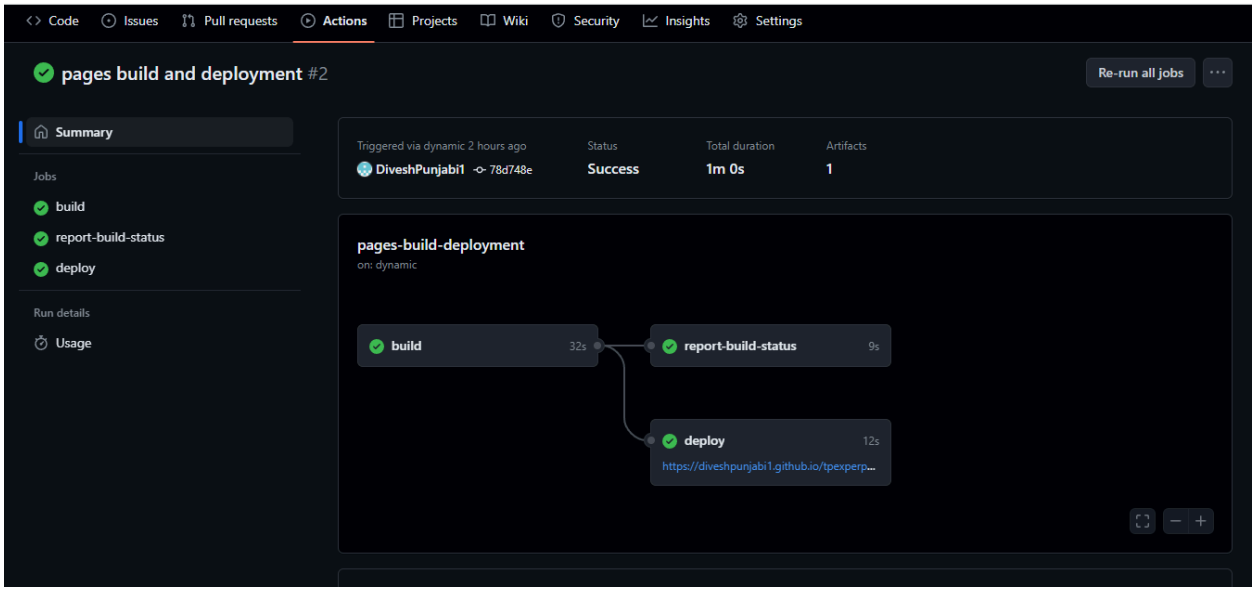
Link to our GitHub repository:https://github.com/DiveshPunjabi1/tpexperpwa

Github Screenshot:

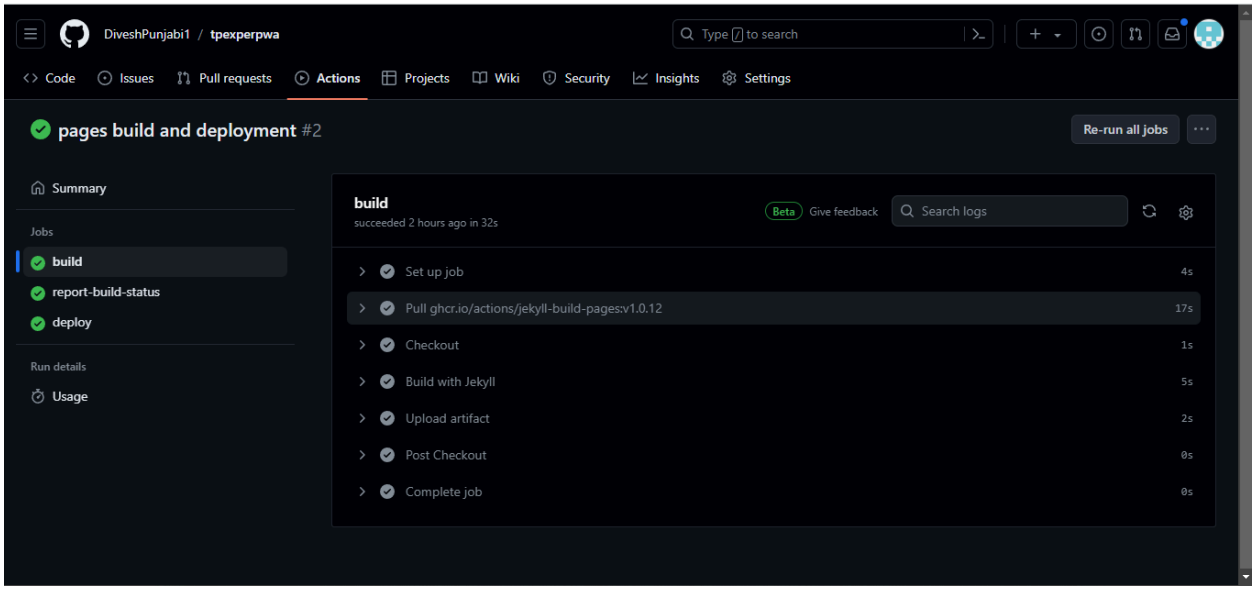


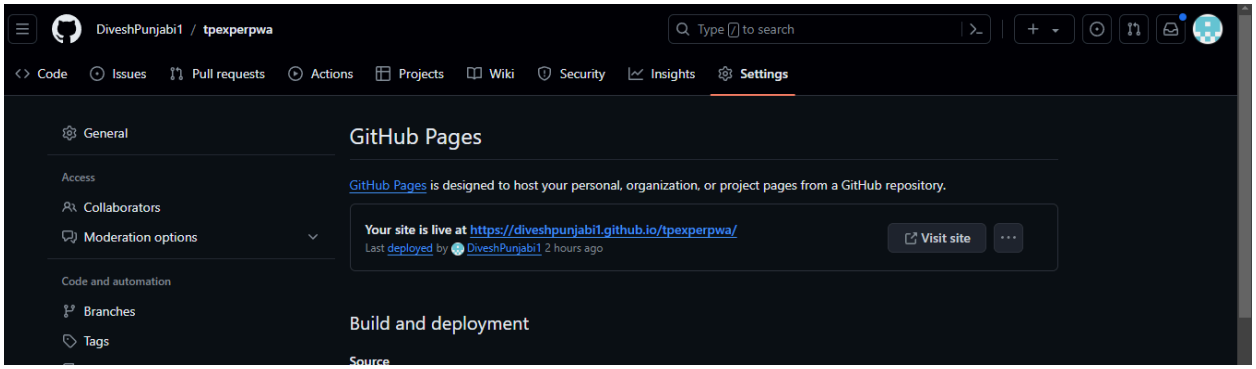
Title:Budgettracker **Roll No. 46**

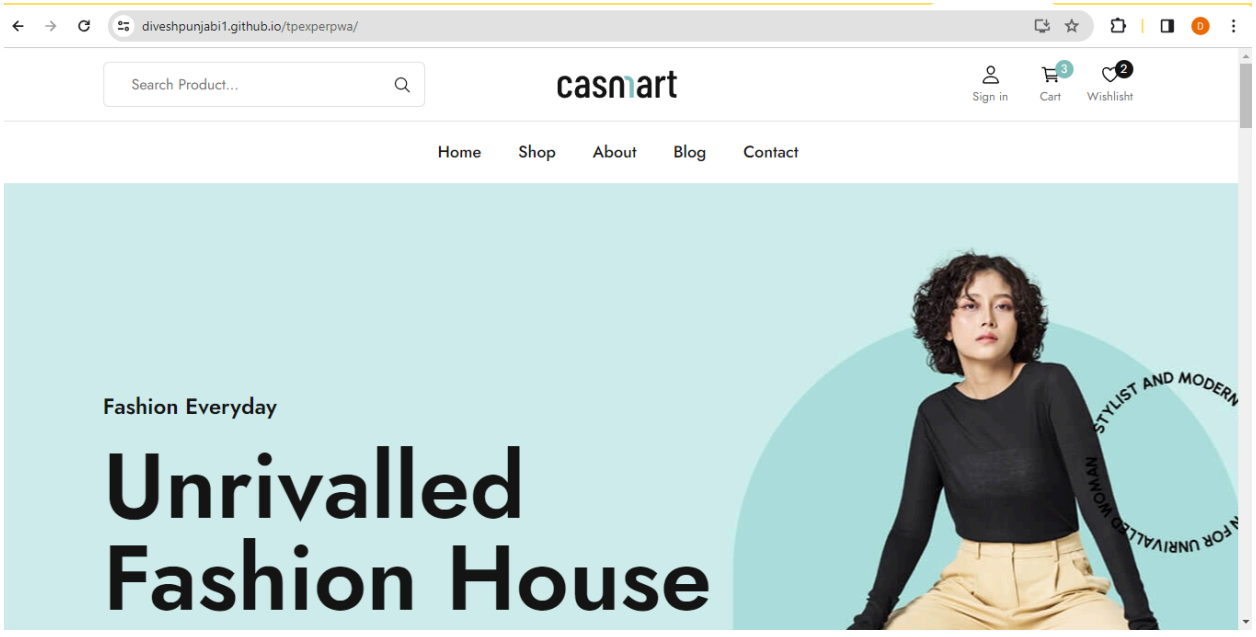




Title:Budgettracker **Roll No. 46**







Conclusion:Hence we have deployed the website on the github pages successfully.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment 11

Aim : To use google Lighthouse PWA Analysis Tool to test the PWA functioning
Theory :Reference : https://www.semrush.com/blog/google-lighthouse/
Google Lighthouse :

Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week.

The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Key Features and Audit Metrics

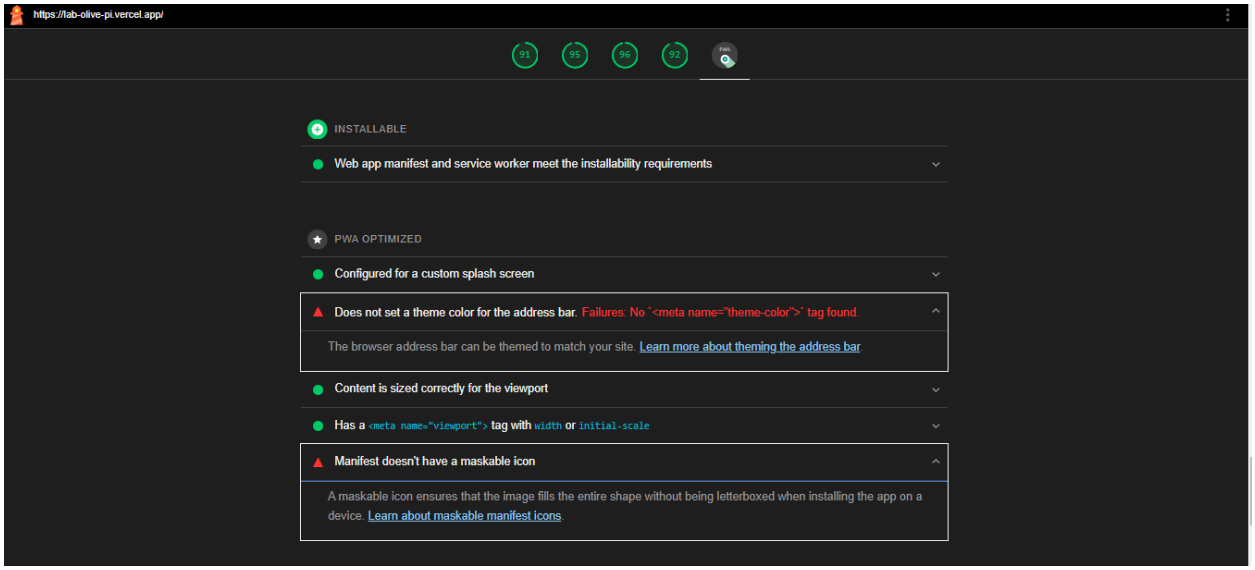
Google Lighthouse has the option of running the Audit for Desktop as well as mobile version of your page(s). The top metrics that will be measured in the Audit are:

1.Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2.PWA Score (Mobile): Thanks to the rise of Service Workers, app

manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.
1.Best Practices: As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS
Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabledGeo-Location and cookie usage alerts on load, etc.

Changes made to the code :

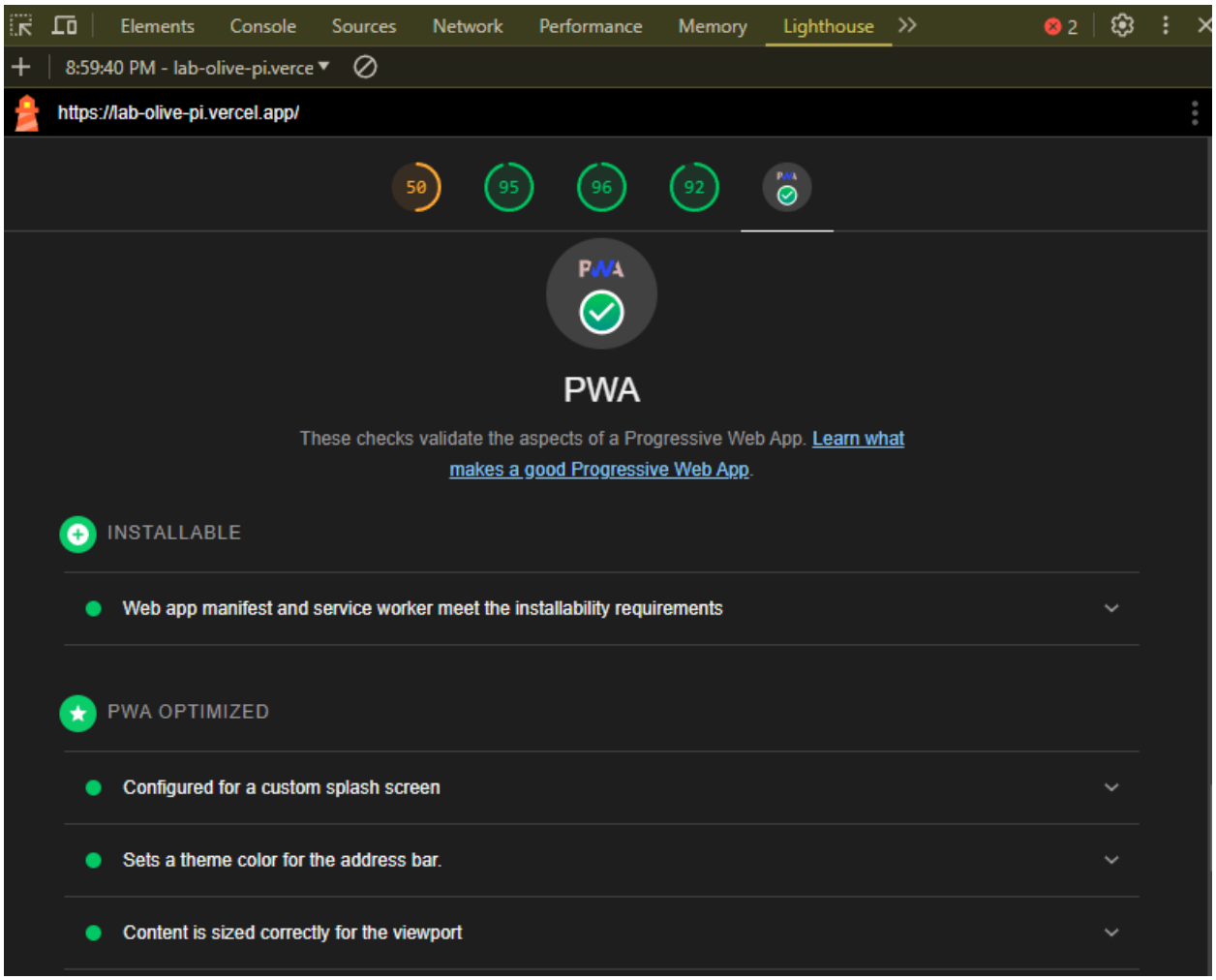
Title:Budgettracker **Roll No. 46**



For theme color add a meta tag in index.html-
<meta name="theme-color" content="#4285f4">
Changes in manifest.json

```
{
  "name":"PWA Tutorial",
  "short_name":"PWA",
  "start_url":"index.html",
  "display":"standalone",
  "background_color":"#5900b3",
  "theme_color":"black",
  "scope": ".",
  "description":"This is a PWA tutorial.",
  "icons":[
  {
  "src":"./assets/images/192x192.png",
  "sizes":"192x192",
  "type":"image/png",
 "purpose": "any maskable"
  },
  {
  "src":"./assets/images/512x512.png",
  "sizes":"512x512",
  "type":"image/png",
```

```
  "purpose": "any maskable"
    }
]
}
```



Conclusion: Thus we successfully used google Lighthouse PWA Analysis Tool for testing the PWA functioning.

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Title:Budgettracker **Roll No. 46**

Divesh. B. Punjabi
D15A 46

FLUTTER

Assignment No 1.

Q.1) Flutter overview: Explain the key features and advantages of using flutter for mobile app developement. Discuss how the flutter framework differs from traditional approaches and why it has gained popularity in the developer community?

Ans Flutter is Google's software developement kit for building ios and Android apps, flutter allows you to create cross platform apps that provide native performance Apps created with flutter feature beautiful and inituitive design and are able to run animations smoothly flutter also increases mobile developement speed, helping lower costs key features and Advantages

1) single codebase - Code written once can be deployed to ios and android platforms. This not only redures developement time but also ensures consistent behaviour across different devices.
2) Hot Reload :- flutter hot reload feature allows developers to instantly see the impact of change made to code.
3) Rich widget Library :- flutter provides comprehensive set of highly customisable widgets for building UI
4) performance :- flutter compiles to native ARM code resulting in high performance applications.
5) Widget based Architecture :- flutter's architecture revolves around widget making it modular and easy to organize code.

6) Cross platform developement :- flutter supports cross platform developement for mobile, web and desktop applications This versatility allows developers to reads a broader audience with minimal extra effort.

traditional Approach :- a) Developers had to maintain different code base for ios and android apps using languages like Java, Swift etc.

b) change in code used to require rebuilding and restarting the entire application leading to slower developement cycles.

c) Achieveing the consistant UI across platforms might require using different UI component which was a problem.

d) Implementing custom animations may be challenging and achieving a highly expensive UI can be time consuming. Such traditional approach have been tackled and these are form that makes flutter framework different from traditional approaches Its key features and advantages is the reason

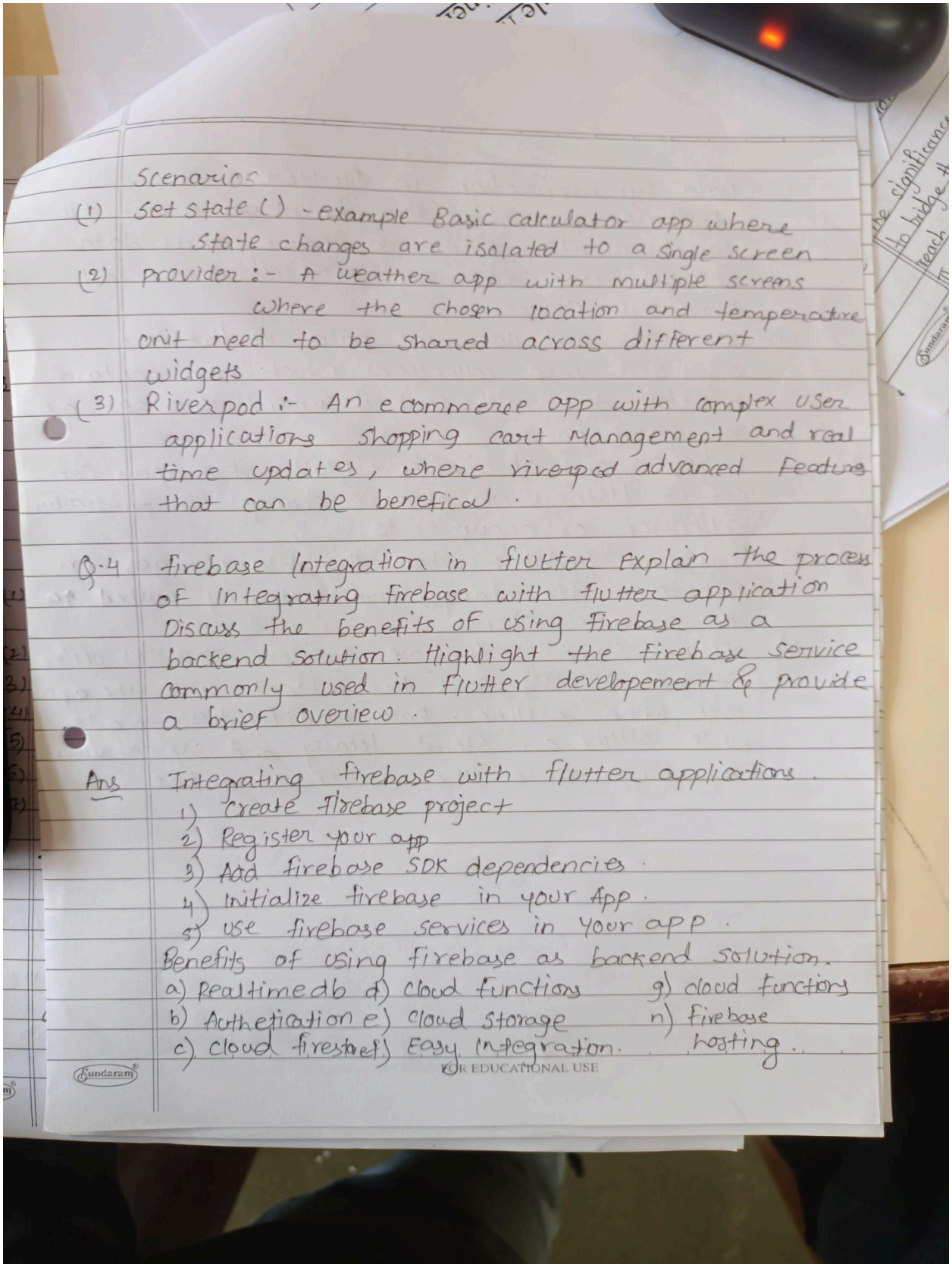
Q.2 widget tree and composition Describe the concept of the widget tree in flutter Explain how widget composition is used to build complex UI. provide Example of commonly used widgets and their roles in creating a widget tree?

4) container :- Basic box Model that can contain other widgets and define ~~pop~~ like padding navigation, I

Q.3 State Management in flutter. Discuss the importance of state Management in flutter application compare and contrast the different state Management approaches available in flutter such as setstate provider and Riverpod. provider scenarios where each approach is suitable.

Ans State Management is critical ~~ap~~ aspect of flutter developement playing a pivotal role in a responsive and efficient user interface.
Importante of the state Management in flutter.

1) Reactivity and UI Updates :- flutter follows reactive programming mode and state Management.
2) performance optimization :- Efficient Management minimizes unnecessary widget rebailds improves performance.
3) code organization and Maintainability :- well Managed state promotes clean code architecture. making it easier to understand Maintain and scale app.
4) Scalability :- As app grows in complexity, proper state Management becomes crucial for maintaining a Manageable databases.
5) Testing :- proper state Management facilites unit Testing and makes it easier to write test cases.
6) comparison of state Management approach.



Scenarios

(1) Set state() - example Basic calculator app where state changes are isolated to a single screen.

(2) provider :- A weather app with multiple screens where the chosen location and temperature unit need to be shared across different widgets.

(3) Riverpod :- An ecommerce app with complex user applications shopping cart Management and real time updates, where riverpod advanced feature that can be benefical.

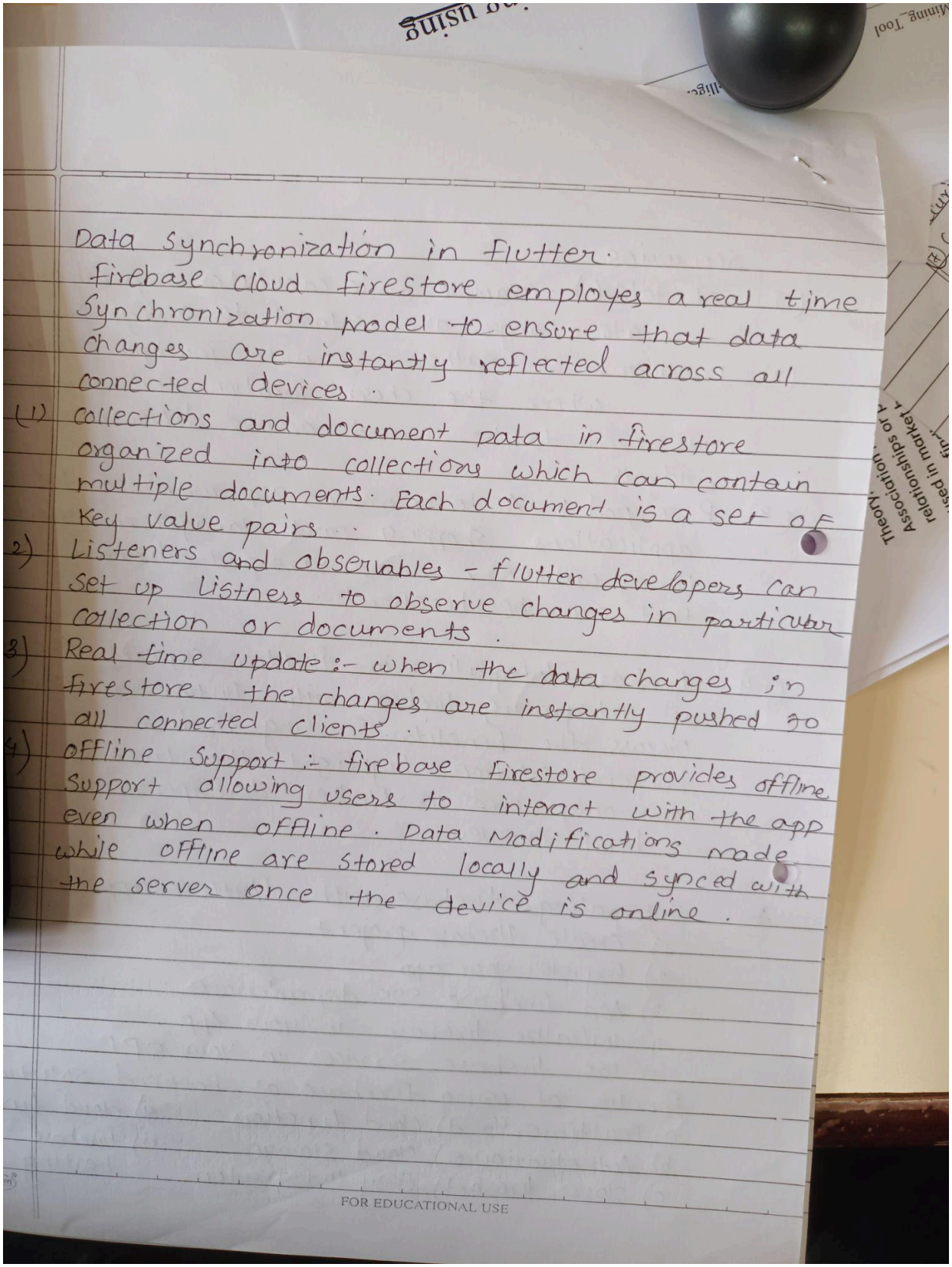
Q.4 Firebase Integration in flutter Explain the process of Integrating firebase with flutter application Discuss the benefits of using firebase as a backend solution. Highlight the firebase service commonly used in flutter developement & provide a brief overiew.

Ans Integrating firebase with flutter applications.

1) Create firebase project
2) Register your app
3) Add firebase SDK dependencies.
4) Initialize firebase in your App.
5) use firebase services in your app.

Benefits of using firebase as backend solution.

a) Realtimedb
b) Authetication
c) Cloud firestore
d) Cloud functions
e) Cloud storage
f) Easy integration.
g) cloud functions
n) Firebase hosting..



Data Synchronization in Flutter.

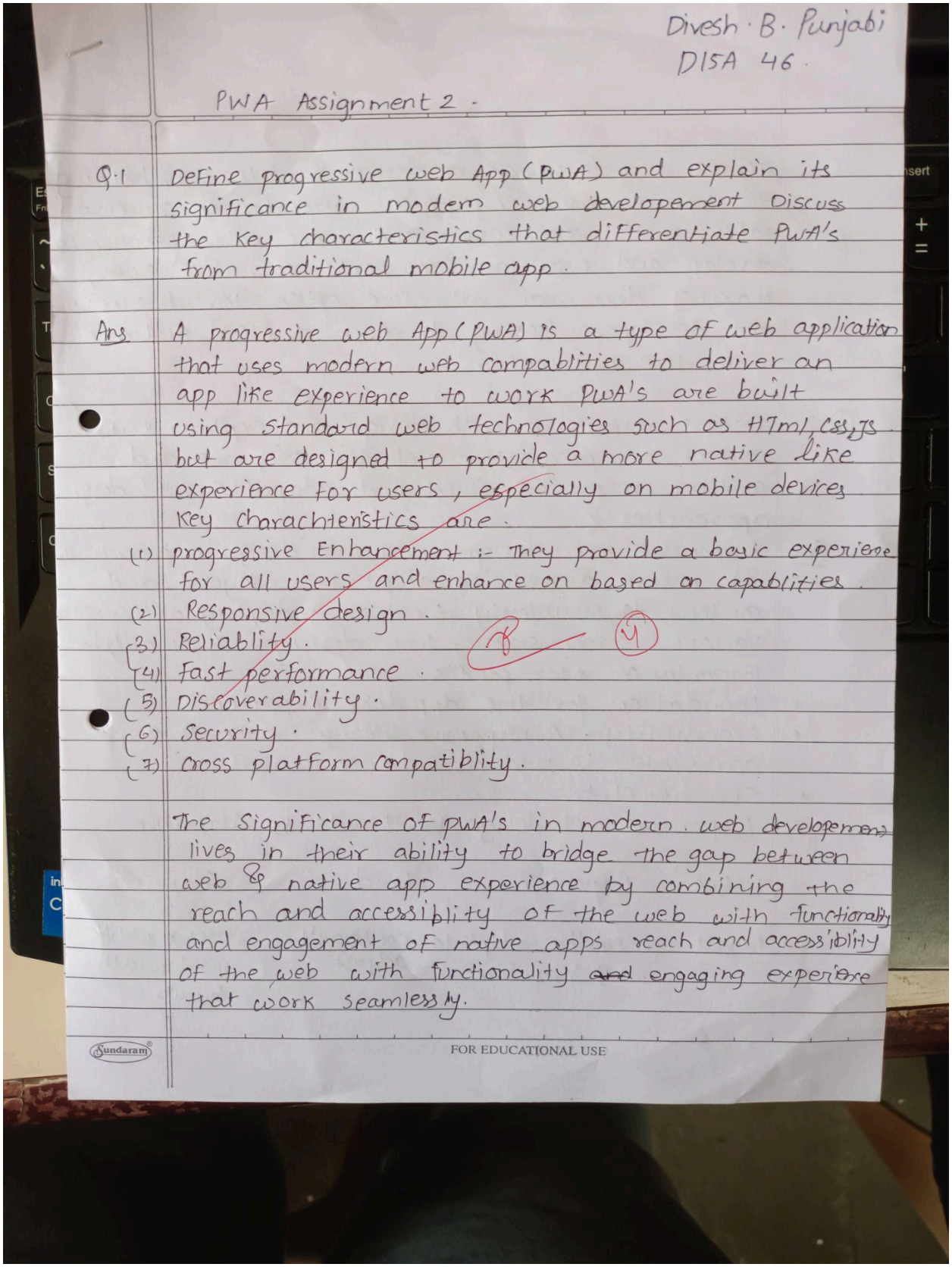
Firebase Cloud Firestore employes a real time Synchronization model to ensure that data changes are instantly reflected across all connected devices.

1) Collections and document data in firestore organized into collections which can contain multiple documents. Each document is a set of key value pairs.
2) Listeners and observables - flutter developers can set up listness to observe changes in particular collection or documents.
3) Real time update :- when the data changes in firestore the changes are instantly pushed to all connected clients.
4) Offline Support :- firebase firestore provides offline Support allowing users to interact with the app even when offline. Data Modifications made while offline are stored locally and synced with the server once the device is online.

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	46
Name	Divesh Bhuvan Punjabi
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	

Title:Budgettracker **Roll No. 46**



Divesh · B · Punjabi
D15A 46.

PWA Assignment 2 .

Q.1 Define progressive web App (PWA) and explain its significance in modern web developement Discuss the key characteristics that differentiate PWA's from traditional mobile app.

Ans A progressive web App (PWA) is a type of web application that uses modern web compabilities to deliver an app like experience to work PWA's are built using standard web technologies such as HTml, Css, Js. but are designed to provide a more native like experience for users, especially on mobile devices Key charachteristics are.

(1) progressive Enhancement :- They provide a basic experience for all users and enhance on based on capablities.
(2) Responsive design.
(3) Reliablity.
(4) Fast performance.
(5) Discoverability.
(6) Security.
(7) Cross platform compatiblity.

The Significance of PWA's in modern web developement lives in their ability to bridge the gap between web & native app experience by combining the reach and accessiblity of the web with functionality and engagement of native apps reach and accessibility of the web with functionality and engaging experience that work seamlessly.

Sundaram

FOR EDUCATIONAL USE

across a wide range of devices & platforms, helping to improve user engagement retention & conversion rates, PWA's are easier and more cost effective to develop and maintain compared to native apps, making them an attractive option for business and developers looking to reach a broad audience with the application.

Q.2 Define responsive web design and explain in the context of progressive web App compare and contract responsive fluid and adapative web design approaches?

Ans Responsive web design is a web design approach that ensure a website adapts its layout and content to seamlessly function & display well across various screen sizes, from desktops to smartphone

Importance of PWA's

- foundation for the app like Experience
- Accessiblity - A responsive design make PWA accessible to wider audience.
- SEO benefits.

(1) Responsive web design Vs fluids Vs Adapative.

Feature	Responsive	fluid	Adaptive.
layout	(1) Flexible adapts to any Screensize	continously adjust	uses multiple fixed width layouts.

control.	(3) Good control over overall layout	less control over element apperance on extreme sizes	High control over layout for each device categ
Suitability	(4) Ideal foundation due to its Versality	can be used but many require adjustment for optimal experience	less common for PWA's due to developement overhead.

Q.3. Describe the lifecycle of service workers including registration installation and activation phases?

Ans Service workers are crucial components of progressive web Apps (PWAS) that enable feature such as offline support, push notification, and background synchronization The lifecycle of service worker involves several ~~by~~ key phases.
registration, installation & activation.

(1) Registeration - 1st phase of lifecycle occurs in the main JS file of web application.

(2) Installation :- once service worker registered, browser downloads and store installation process of service worker script.

(3) Installation process is critical for setting up the service worker's initial cache & preparing it to take control of web application.

(4) Activation :- After installation phase, Addition phase it occurs when service worker is ready to take control of the web application.

Q.4 Explain the use of indicked DB in the service workers for data storage.

Ans (i) IndexedDB is a lowlevel API for client side storage that allows web applications to store ~~dataset~~ different types of data presidently in the web browser's.

(ii) It is a full blown persistent nosQL storage system that supports various data types.

(iii) IndexedDB is particularly useful in scenarios such as caching and pwAs and gaming and it supports transactions to ensure data integrity.

(iv) Service workers can intercept network requests and serve cached response but for application that require data persistence and offline access to data.

(v) This combination can allows web application. to function offline by serving cached assests and data stored in IndexedDB ensuring a seamless ux even without an internet connections.