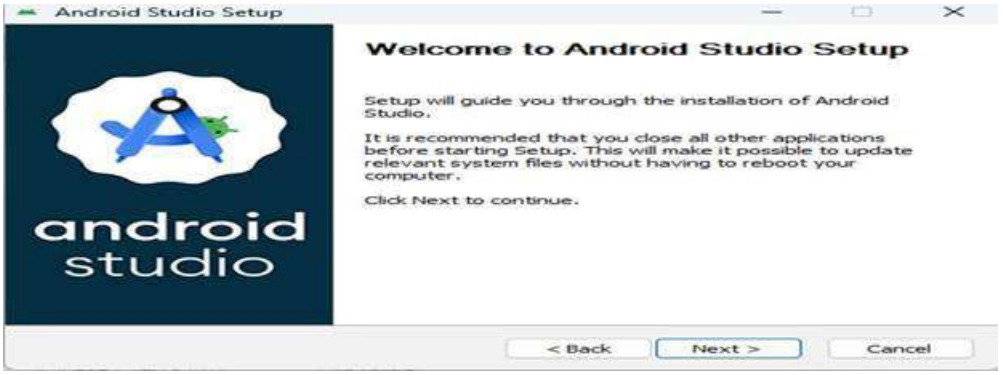
PRACTICAL NO:01

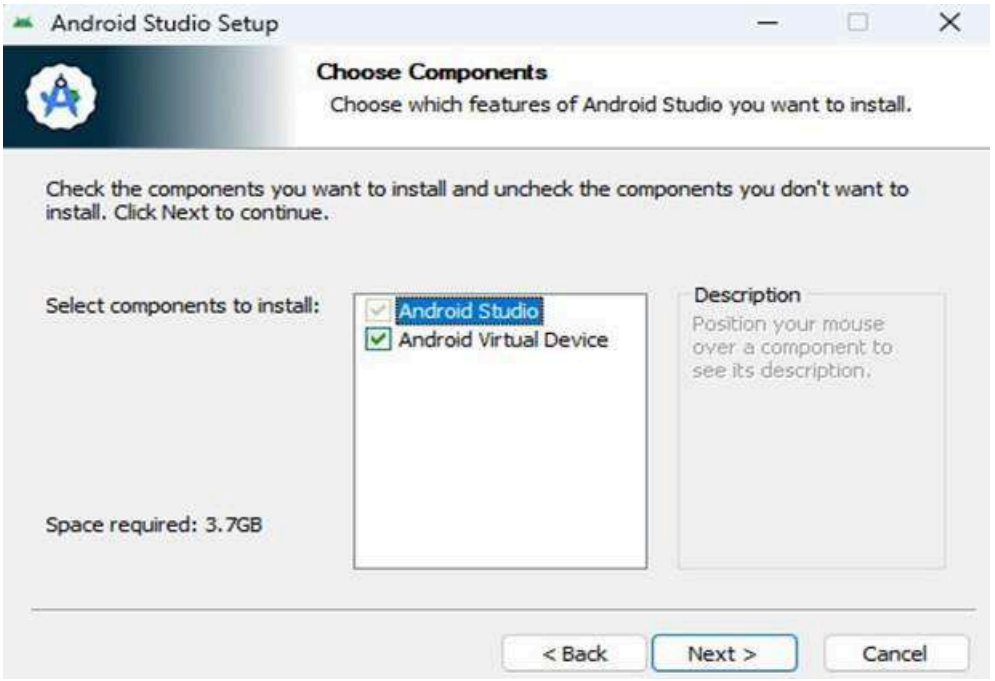
Aim: Installation of Android Studio App

Theory: Installation of Android studio app

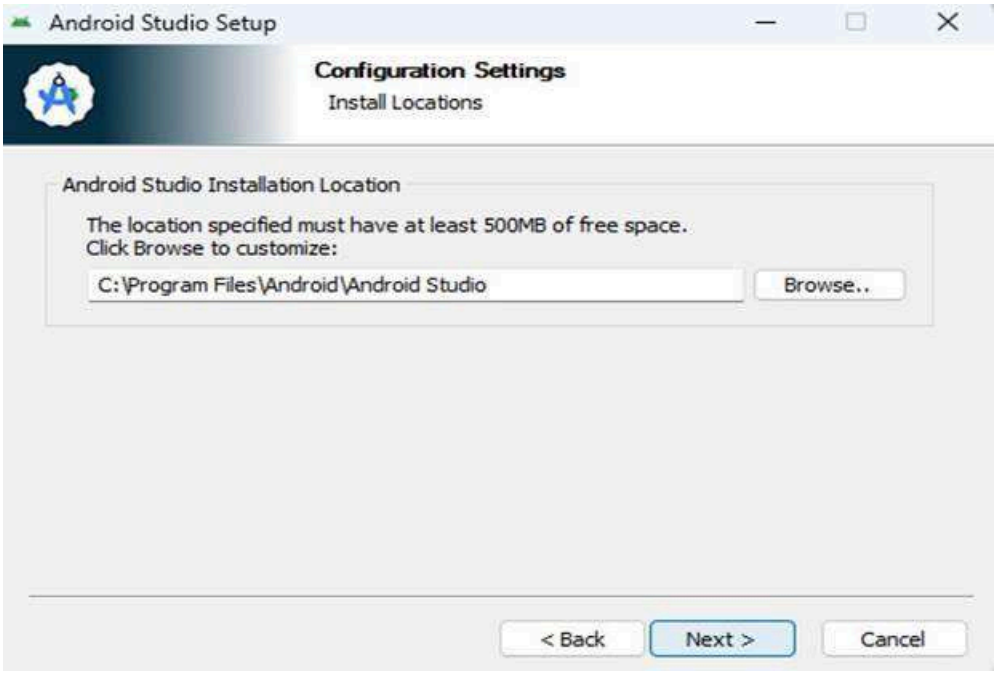
1) **Open the website to download the app**
2) **Click on next to install**

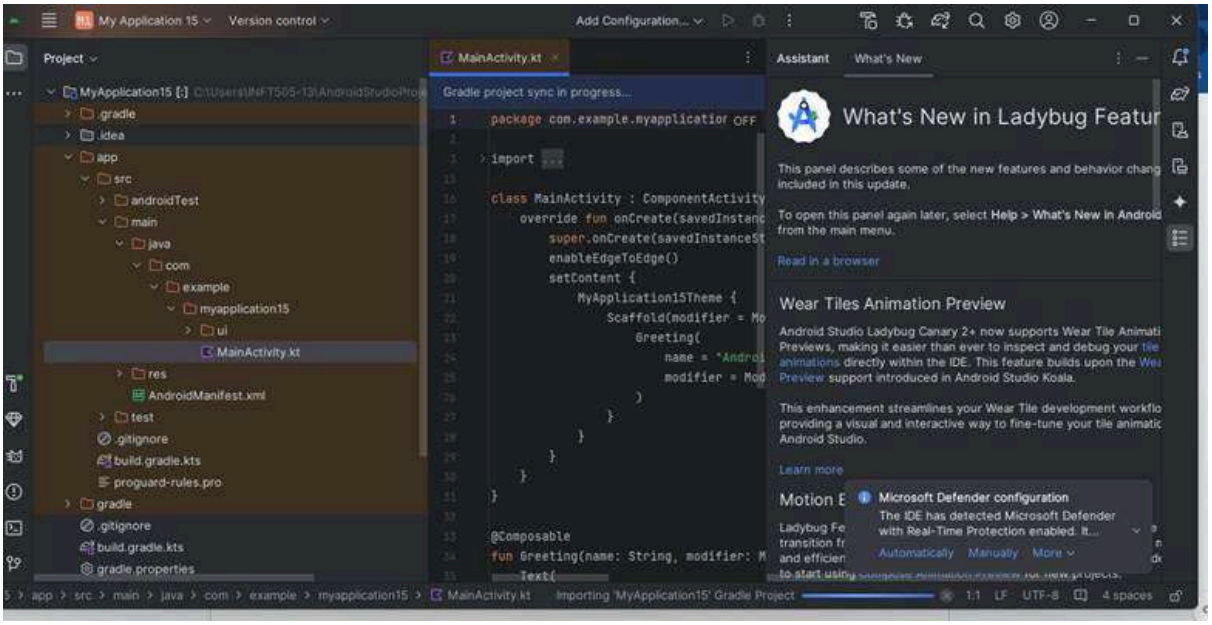


3) **Click on Android Virtual Device and click Next**



4) **Browse the location**





Conclusion: In this experiment, we successfully installed and configured the Flutter environment on a Windows system. We downloaded the Flutter SDK, set up the system path, and verified the installation using the flutter doctor command. Additionally, we installed Android Studio, set up the Android SDK, configured an emulator for testing, and integrated Flutter and Dart plugins into Android Studio.

Experiment No: 02

Aim: To Design Flutter UI by Including Common Widgets

Theory:

Flutter is an open-source UI software development kit (SDK) created by Google, used to develop applications for mobile, web, and desktop from a single codebase. It provides a wide range of widgets that help in building flexible and visually appealing UIs.

Common Flutter widgets include:

1. **Scaffold – Provides the basic structure of an app, including an AppBar, FloatingActionButton, and a body.**
2. **Container – A versatile widget for designing UI components with padding, margin, color, and decoration properties.**
3. **Row and Column – Used to arrange widgets horizontally and vertically, respectively.**
4. **ListView – A scrolling widget used to display a list of items dynamically.**
5. **Text and TextStyle – Used for displaying and styling text in the application.**
6. **Image – Displays images from assets, network, or memory.**
7. **ElevatedButton – A material design button that responds to user interaction.**
8. **Navigator – Helps in navigating between different screens in the application.**

Conclusion:

In this experiment, we successfully designed a Flutter UI using various commonly used widgets. By utilizing Scaffold, Container, Row, Column, ListView, and ElevatedButton, we were able to create a structured and interactive layout.

Experiment No: 03

Aim: To include icons, images, fonts in Flutter app Theory:

Flutter allows developers to enhance their app's UI by incorporating icons, images, and custom fonts. These elements help in improving the app's aesthetics and user experience.

1. **Including Icons:**

Flutter provides built-in Material Icons and supports custom icon sets.

Using Material Icons:
Icon(Icons.home, size: 30, color: Colors.blue);

Using Custom Icons (from packages like FontAwesome): import 'package:font_awesome_flutter/font_awesome_flutter.dart';

Icon(FontAwesomeIcons.heart, size: 30, color: Colors.red);

2. **Adding Images:**

Images can be added from local assets or loaded from a network.

Adding Local Images:

Place images in the assets folder.

Define them in pubspec.yaml:
yaml flutter:
assets:

- assets/images/logo.png

Output:

Conclusion:

By including icons, images, and fonts, we can enhance the visual appeal and usability of a Flutter app. Using built-in Material icons, external icon libraries, asset images, and custom fonts ensures a polished and professional UI. This experiment highlights the flexibility of Flutter in supporting various multimedia elements, making it a powerful tool for mobile app development.

Experiment No: 04

Aim: To create an interactive Form using form widget Reference

Objective

The objective of this experiment is to create an interactive form using Flutter's Form widget, incorporating validation logic for user input and submission handling.

Theory

Key Components of a Flutter Form

1. Form Widget

The Form widget in Flutter is a container for grouping and validating multiple form fields (TextFormField, DropdownButtonFormField, etc.). It simplifies managing the state and validation of form elements collectively.

- **It requires a GlobalKey<FormState> to access and control the form's state (e.g., validate, save, reset).**
- **It enables batch validation of all its children form fields using .validate().**

🔑 2. GlobalKey<FormState>

A GlobalKey<FormState> allows access to the internal state of the Form widget. It enables:

- **Validation using _formKey.currentState!.validate()**
- **Submission logic using _formKey.currentState!.save()**
- **Resetting the form using _formKey.currentState!.reset()**

🔤 3. TextFormField

This is a text input field that integrates well with Form. It supports:

- **Validation via the validator property.**
- **Saving input via the onSaved property.**
- **Controllers to manage and retrieve text input.**

✅ 4. Form Validation

Form validation ensures users input correct and expected data. Flutter supports:

- **Synchronous validation using the validator property.**
- **Real-time validation triggered by typing (when using autovalidateMode).**
- **Custom rules such as checking:**

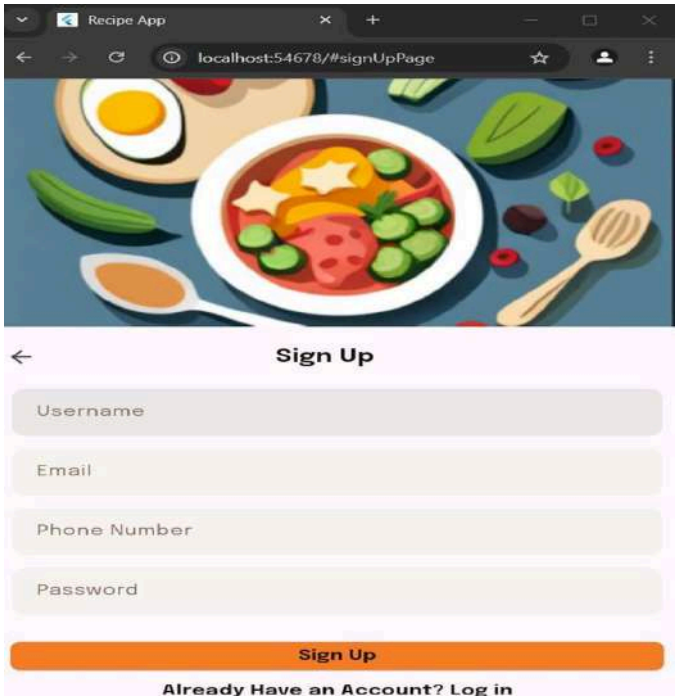
🎯 5. Submit Button and Form Submission

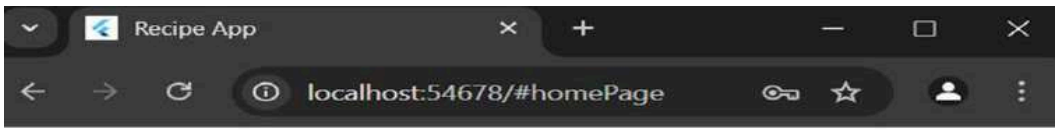
- **Submitting the form is typically handled by a button.**
- **On press, the app checks if the form is valid.**
- **If valid, data is saved or processed.**

Implementation Steps

1. **Create a Form with a GlobalKey<FormState>**
2. **Add a TextFormField with validation logic**
3. **Create a button to validate and submit the form**

Output:




Recipe App
localhost:54678/#homePage


Welcome to Foudly
Account created successfully.
Home
Cart
Favorite

Experiment No: 05

Aim: To Apply Navigation, Routing, and Gestures in a Flutter App

Objective:

The objective of this experiment is to understand and implement navigation between screens, routing using named and anonymous routes, and handling gesture detection in a Flutter application to create interactive and dynamic user interfaces.

Theory:

Flutter provides built-in support for **navigation**, **routing**, and **gesture handling**, which are essential for creating multi-screen, interactive applications.

1. **Navigation in Flutter**

Navigation allows users to move between different screens (also called routes or pages). Flutter uses a **stack-based navigation system** — meaning new screens are pushed onto the navigation stack, and going back pops them off.

- **Navigator Class:**
  - Used to manage routes.
  - Key methods:

■ Navigator.push(context, route) – navigates to a new screen.
■ Navigator.pop(context) – returns to the previous screen.

2. **Routing in Flutter**

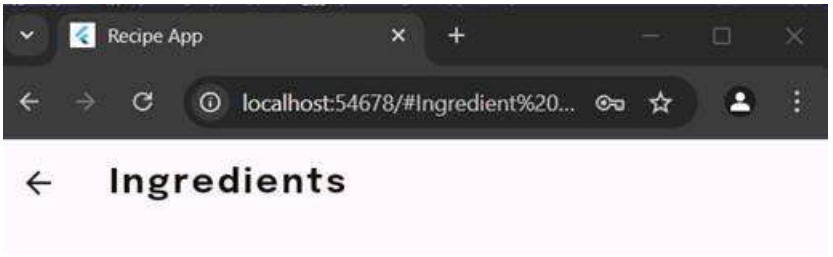
Routing determines how users move between screens and how screens are identified.

- **Anonymous Routing:** Uses MaterialPageRoute directly.
- **Named Routing:** Defines route names in the MaterialApp and navigates using the name.

3. Gestures in Flutter

Gestures refer to user interactions like tapping, swiping, dragging, etc. Flutter provides gesture detection through the `GestureDetector` widget.

Output:



Conclusion: In this experiment, we successfully applied navigation, routing, and gesture detection in a Flutter app. We explored how to move between screens using `Navigator`, how to configure named and anonymous routes, and how to use `GestureDetector` for handling user interactions.

Mastering these core features allows developers to build seamless, interactive, and scalable applications with a better user experience.

EXP 06

Aim: To set up Firebase authentication for the android application.

1. firebase_core Initialization:

o await Firebase.initializeApp(); is crucial. This line initializes Firebase for your app. It's placed in the main() function before runApp().

o WidgetsFlutterBinding.ensureInitialized(); is also important, and is required before Firebase.initializeApp().

2. Firebase Authentication (firebase_auth):

o The code uses FirebaseAuth for user authentication.

o signInWithEmailAndPassword and createUserWithEmailAndPassword are used for login and registration. o Error handling is improved with FirebaseAuthException catching, providing user friendly messages.

3. Cloud Firestore (cloud_firestore):

o FirebaseFirestore is used to store user data in the "users" collection.

o set() is used to write data to Firestore.

4. Platform Setup (iOS and Android):

o To make this work, you must follow the Firebase console setup for both iOS and Android. This involves:

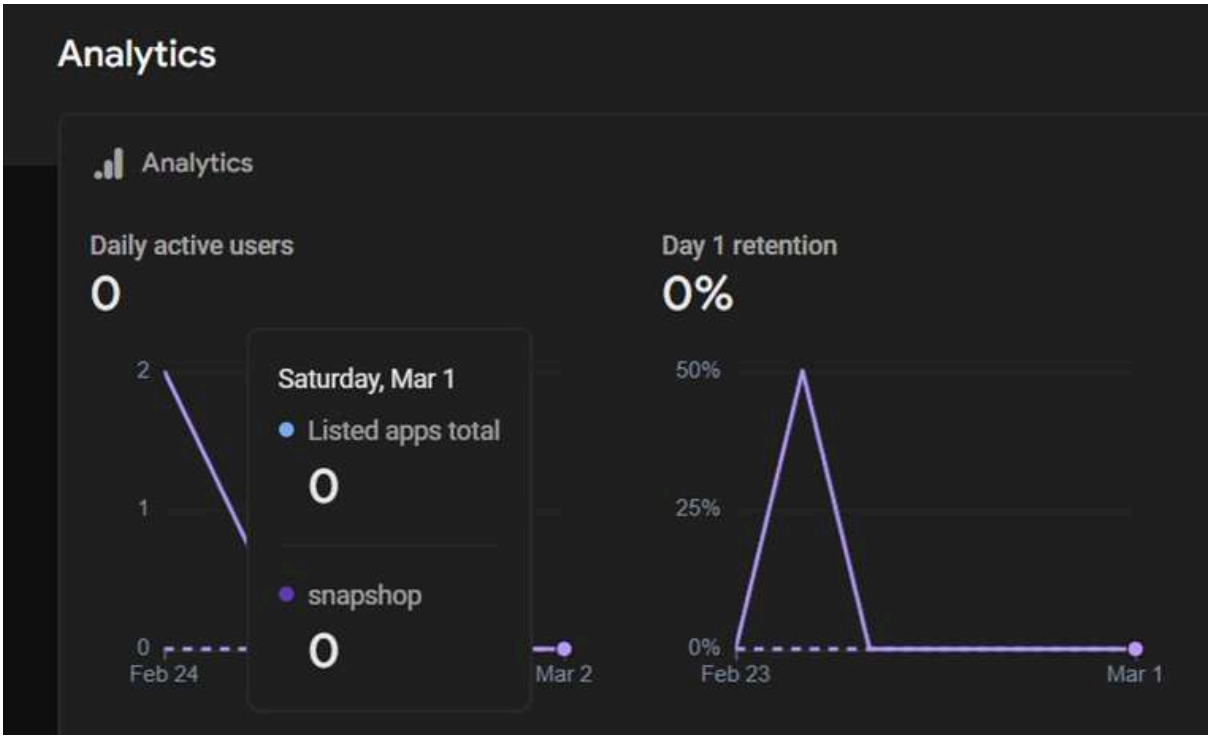
- Creating a Firebase project.
- Adding iOS and Android apps to your project.
- Downloading the google-services.json (Android) and GoogleService-Info.plist (iOS) files and placing them in the correct locations within your Flutter project.
- Ensuring the correct Firebase plugins are added to your pubspec.yaml.

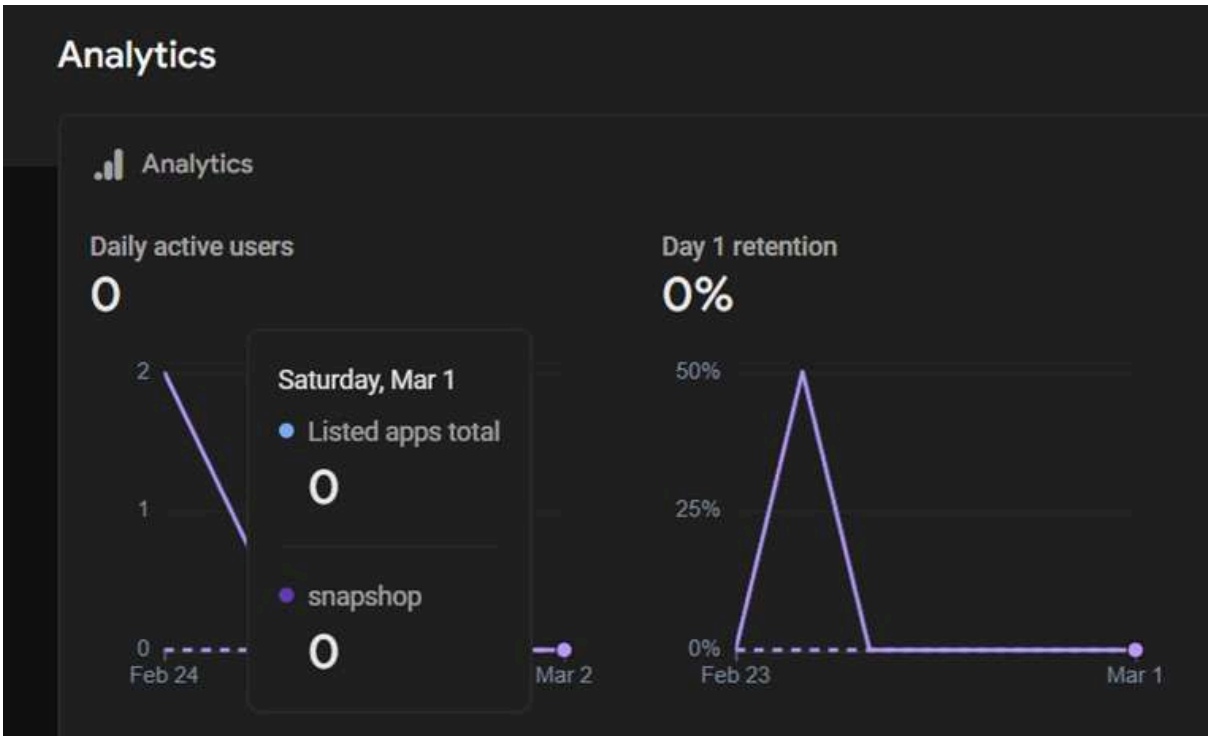
5. Error Handling: FirebaseAuth exceptions are now specifically handled, providing better error messages to the user.

6. Navigation: Navigation is implemented using named routes, allowing for cleaner transitions.

We need the following packages; import 'package:firebase_auth/firebase_auth.dart'; import 'package:firebase_core/firebase_core.dart'; import 'package:cloud_firestore/cloud_firestore.dart';

Output



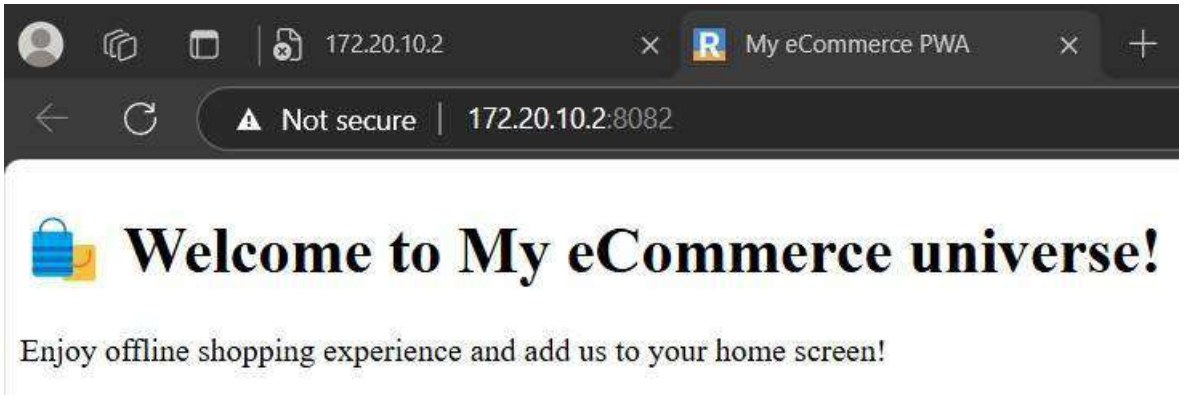


Conclusion: In this experiment, we learnt to integrate our application with firebase and implement authentication functionality. We made the necessary changes to our flutter program to implement the desired functionality.

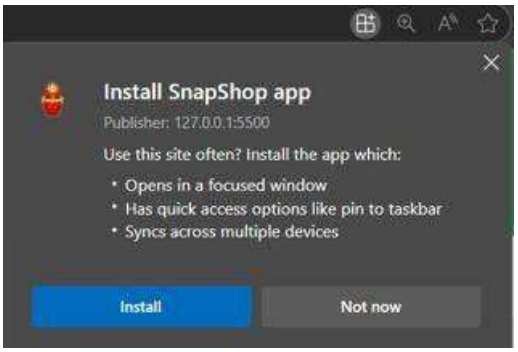
MAD Experiment 7

Aim: Add to your home screen feature on a web application.

On adding Icon images and manifest.json file to the file structure, we could see the option to install the website as if it were an application.



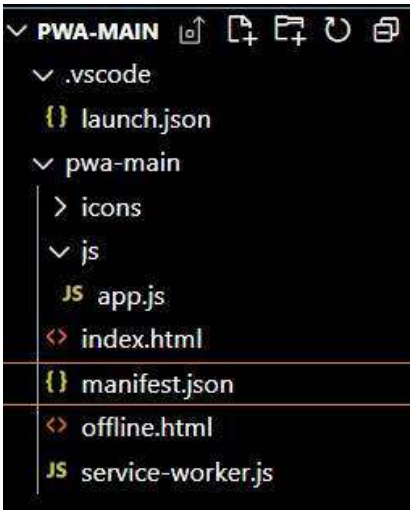
On clicking install, the web application could be accessible from the desktop. Clicking it would create an isolated instance of that web app through which only the features of that particular web app would be accessible.



Conclusion: Through this experiment we learnt to add the ‘add to my webpage’ feature to our web application. This is the most fundamental step to be performed while building progressive web applications.

MAD Experiment 8

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. Create service-worker.js



Create a cacheable file called offline.html to be displayed in the absence of an internet connection.

PWA exp 9

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

pwa-main > JS service-worker.js > ...

```
12  self.addEventListener('install', event => {
16      self.skipWaiting();
17  });
18
19  // Activate
20  self.addEventListener('activate', event => {
21    event.waitUntil(
22      caches.keys().then(keys =>
23        Promise.all(keys.map(key => {
24          if (key !== CACHE_NAME) {
25            return caches.delete(key);
26          }
27        }))
28      )
29    );
30    self.clients.claim();
31  });
32
33  // Fetch
34  self.addEventListener('fetch', event => {
35    event.respondWith(
36      caches.match(event.request).then(cachedResponse => {
37        // Serve from cache if available
38        if (cachedResponse) return cachedResponse;
39
40        // Try fetching from network
41        return fetch(event.request).catch(() => {
42          // Fallback only for navigation (HTML page loads)
43          if (event.request.mode === 'navigate') {
44            return caches.match('/offline.html');
45          }
46        });
47      })
48    );
49  });
```

Make the following changes to the service-worker.js

// Install Event: Cache assets

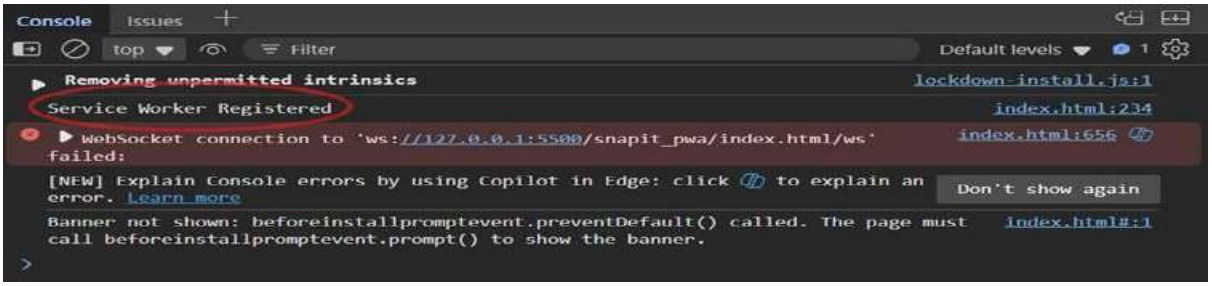
// Activate Event: Cleanup old caches

// Fetch Event: Supports both Cache-First & Network-First

// Sync Event: Retry sending data when online

// Function to send pending screenshots to the server

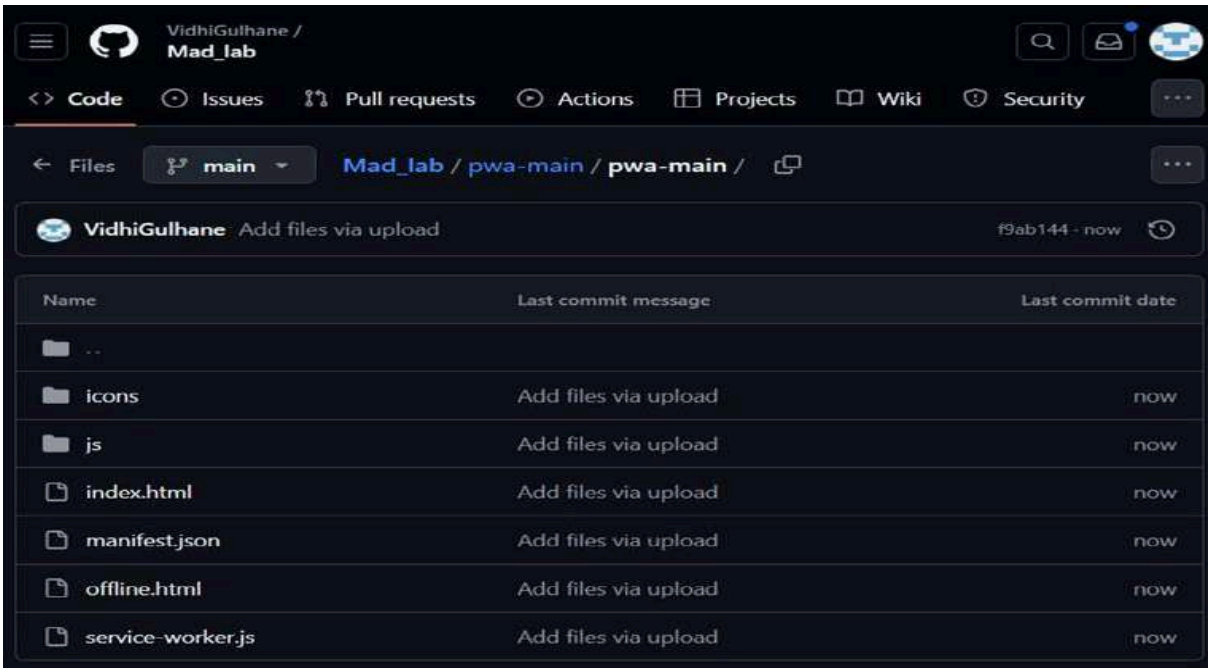
// Push Event: Display push notifications

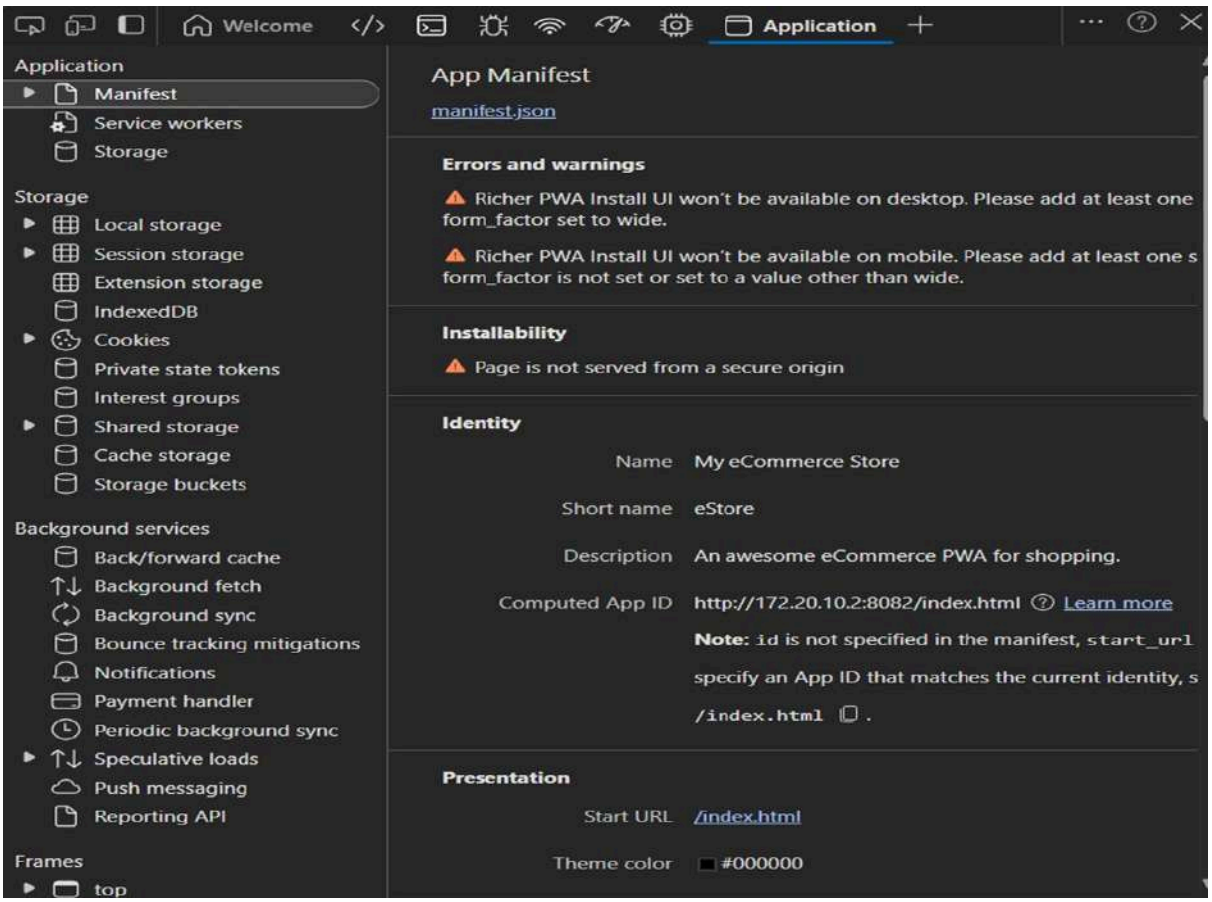


Conclusion: We implemented the functionality of offline web cache capture so that in the absence of a stable internet connection, the app would display a generic waiting page.

Experiment 10

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

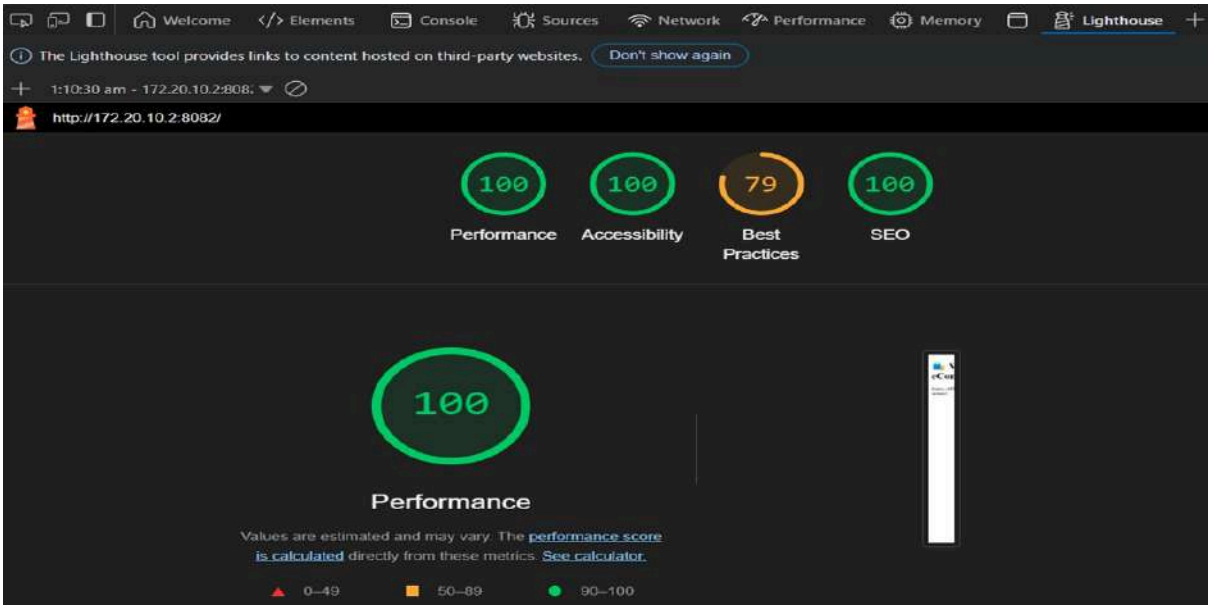




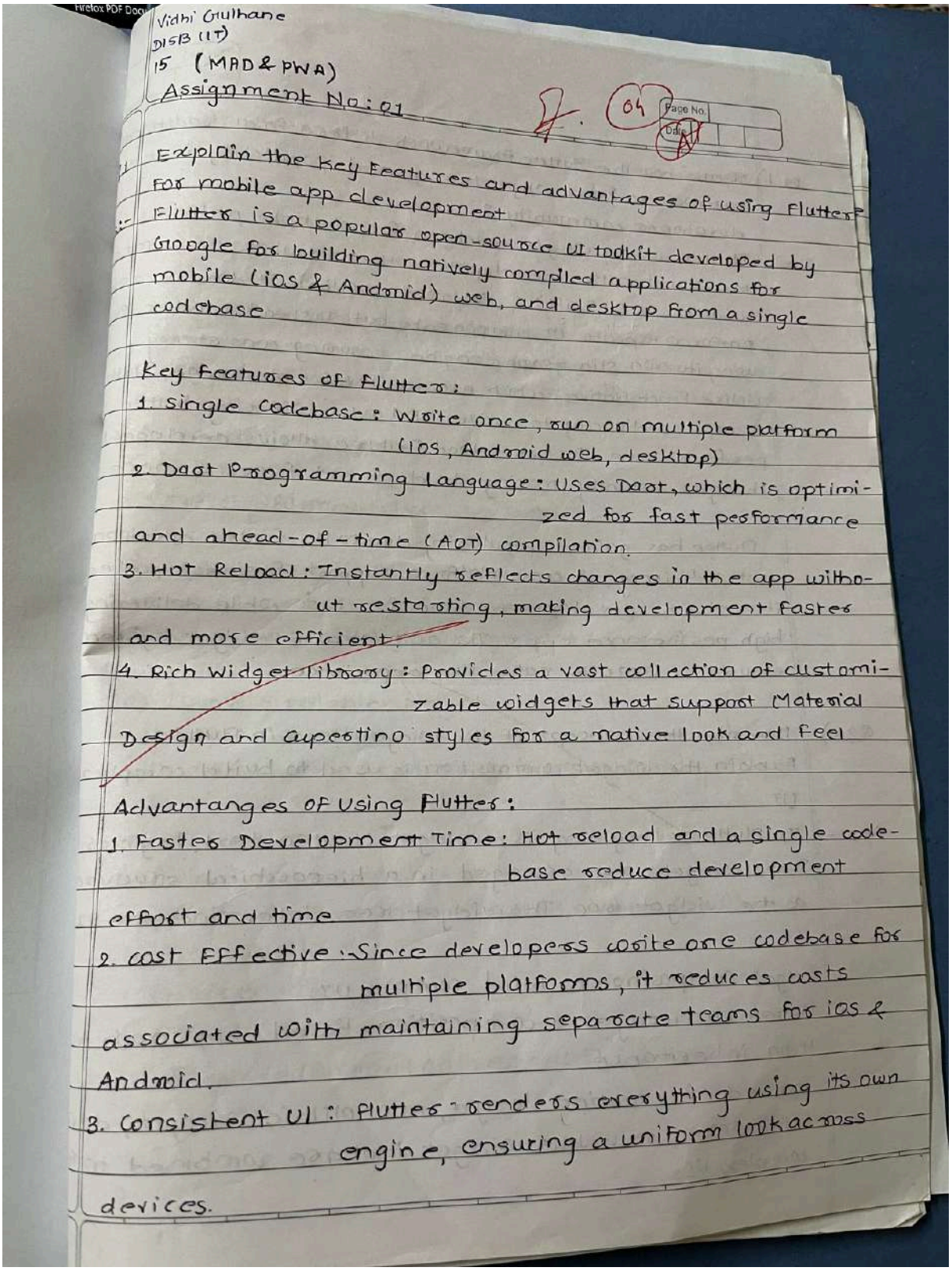
Exp 11

Aim- To study and implement google lighthouse PWA analysis toll to test the progressive Web App (PWA) functionality

Output-



Assignment 01



Vidhi Gulhane
D15B (IT)
15 (MAD & PWA)
Assignment No: 01

Page No.
Date

Explain the key features and advantages of using Flutter for mobile app development

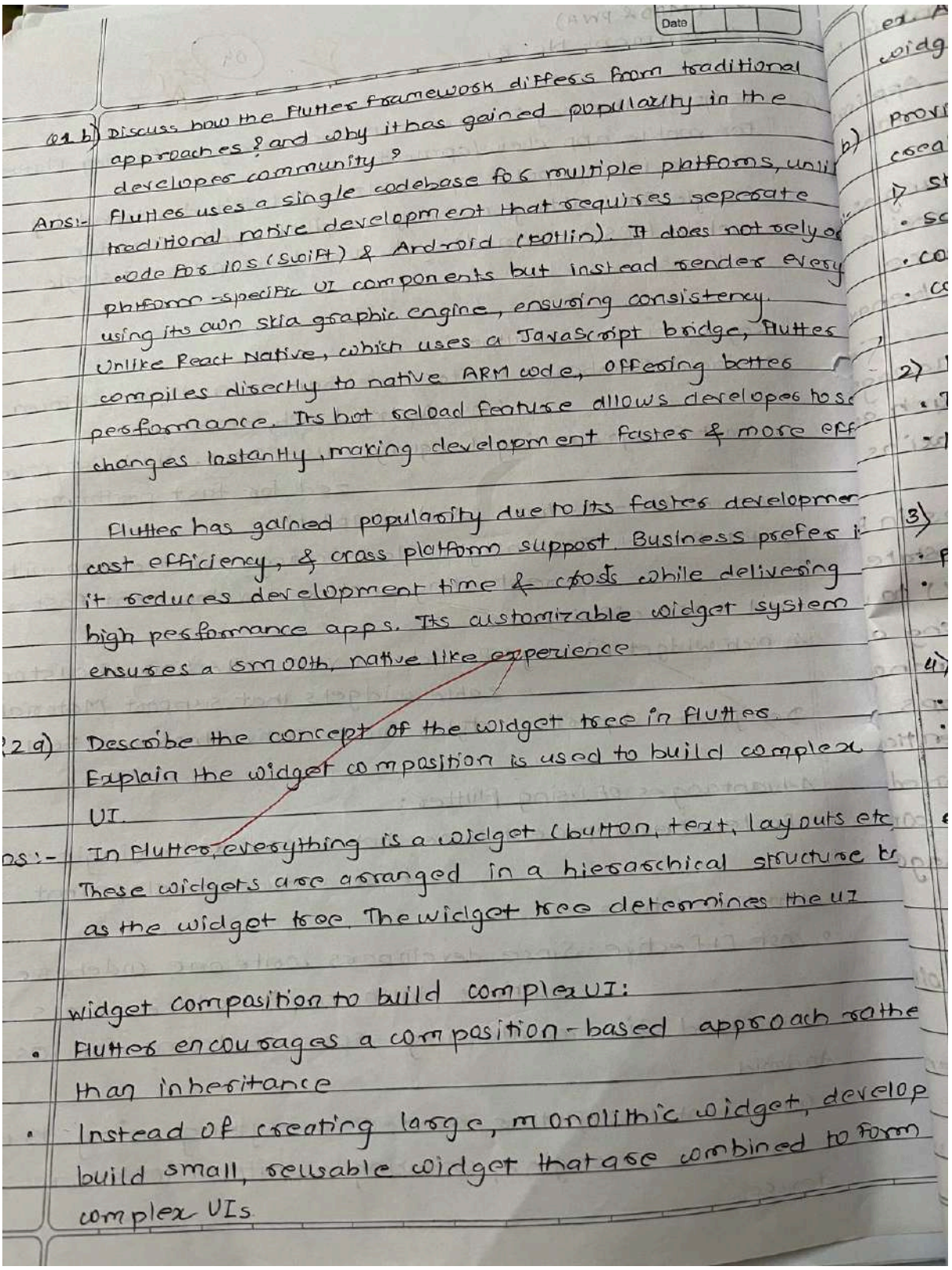
:- Flutter is a popular open-source UI toolkit developed by Google for building natively compiled applications for mobile (iOS & Android) web, and desktop from a single codebase

Key Features of Flutter:

1. Single Codebase: Write once, run on multiple platform (iOS, Android web, desktop)
2. Dart Programming Language: Uses Dart, which is optimized for fast performance and ahead-of-time (AOT) compilation.
3. Hot Reload: Instantly reflects changes in the app without restarting, making development faster and more efficient.
4. Rich Widget Library: Provides a vast collection of customizable widgets that support Material Design and Cupertino styles for a native look and feel

Advantanges of Using Flutter:

1. Faster Development Time: Hot reload and a single codebase reduce development effort and time
2. Cost Effective: Since developers write one codebase for multiple platforms, it reduces costs associated with maintaining separate teams for ios & Android.
3. Consistent UI: Flutter renders everything using its own engine, ensuring a uniform look across devices.



Q1 b) Discuss how the Flutter framework differs from traditional approaches? and why it has gained popularity in the developer community?

Ans:- Flutter uses a single codebase for multiple platforms, unli traditional native development that requires seperate code for iOS (Swift) & Android (Kotlin). It does not rely o platform-specific UI components but instead render every using its own Skia graphic engine, ensuring consistency. Unlike React Native, which uses a Javascript bridge, Flutter compiles directly to native ARM code, offering better performance. Its hot reload feature allows developers to s changes instantly, making development faster & more eff

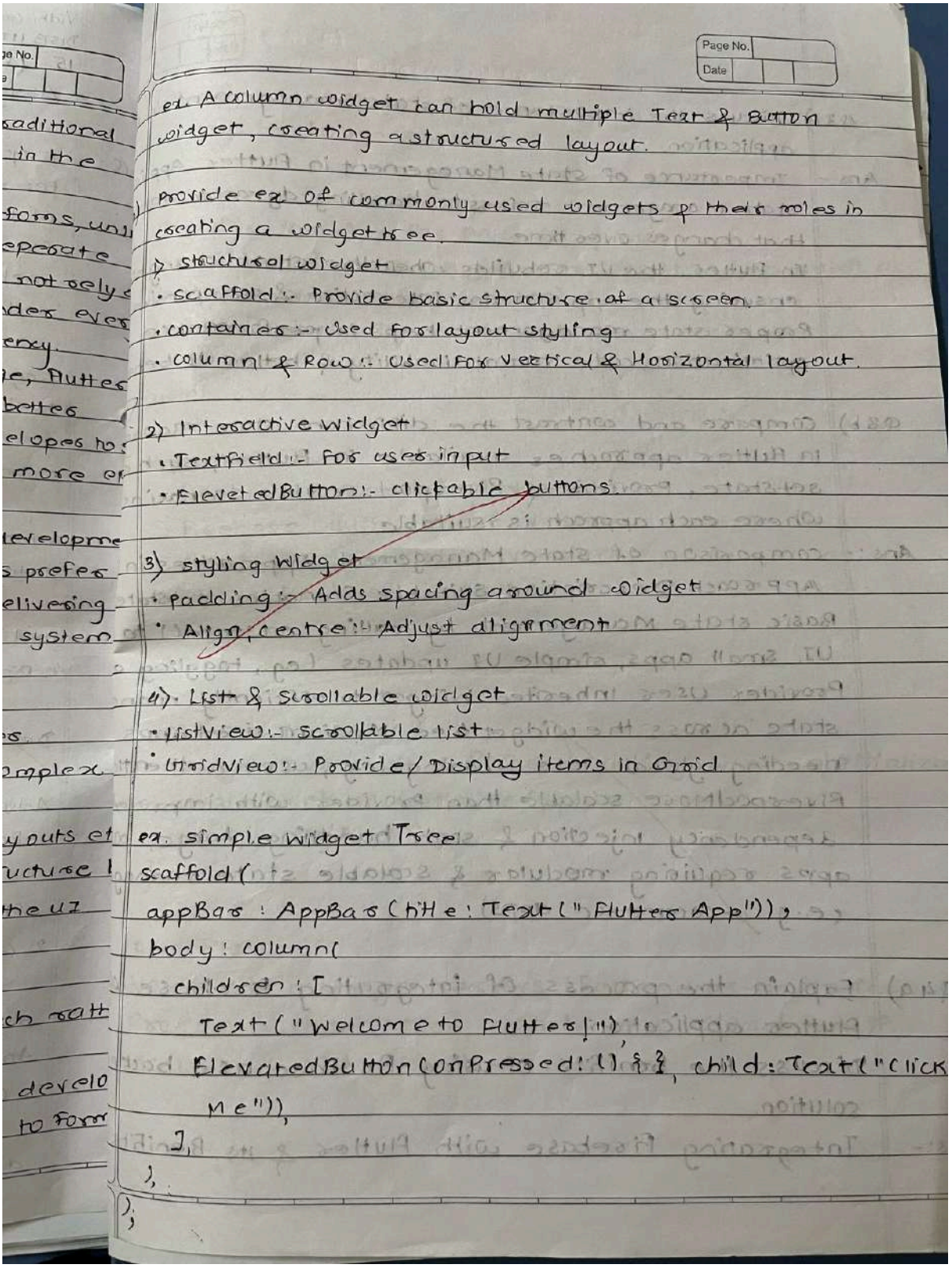
Flutter has gained popularity due to its faster developme cost efficiency, & cross platform support. Business prefer i it reduces development time & costs while delivering high performance apps. Its customizable widget system ensures a smooth, native like experience

Q2 a) Describe the concept of the widget tree in Flutter. Explain the widget composition is used to build complex UI.

Ans:- In Flutter, everything is a widget (button, text, layouts etc, These widgets are arranged in a hierarchical structure k as the widget tree. The widget tree determines the UI

widget composition to build complex UI:

- Flutter encourages a composition-based approach rathe than inheritance
- Instead of creating large, monolithic widget, develop build small, reusable widget that are combined to form complex UIs



ex. A column widget can hold multiple Text & Button widget, creating a structured layout.

Provide ex of commonly used widgets & their roles in creating a widget tree.

1) structural widget
- scaffold :- Provide basic structure of a screen.
- container :- Used for layout styling
- column & Row :- Used for Vertical & Horizontal layout.

2) Interactive Widget
- Textfield :- For users input
- ElevetedButton :- clickable buttons

3) styling Widget
- padding :- Adds spacing around widget
- Align, centre :- Adjust alignment

4) List & Scrollable widget
- ListView :- scrollable list
- GridView :- Provide / Display items in Grid

ex. simple widget Tree

```
scaffold(
  appBar : AppBar(title : Text("FlutterApp")),
  body : column(
    children : [
      Text("Welcome to Flutter!!"),
      ElevatedButton(onPressed: (){}, child: Text("Click Me")),
    ],
  ),
);
```

Date

Q3 a) Discuss the importance of state management in Flut application.

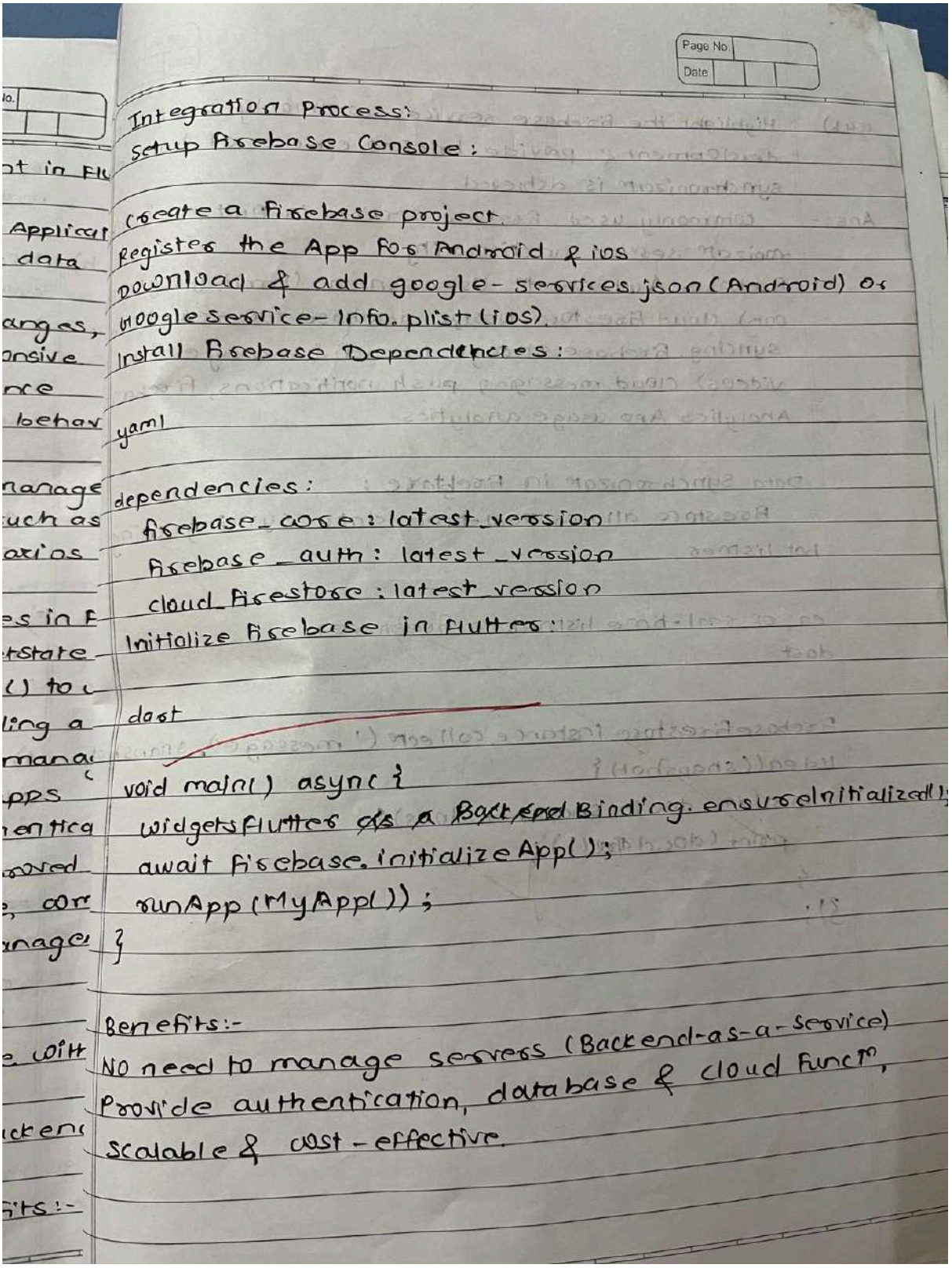
Ans:- Importance of state Management in Flutter Applicati state Management refers to handling dynamic data that changes over time.
In Flutter, the UI rebuilds when the state changes, ensuring the app remains interactive & responsive. Proper state management helps in performance optimization, code maintainability & better UI behavio

Q3 b) Compare and contrast the different state manager in Flutter approaches available in Flutter, such as setstate, Provider & Riverpod. Provide scenarios where each approch is suitable.

Ans :- Comparison of State Management Approaches in Fl
Approach Description Suitable Scenarios setstate
Basic state Management by calling setstate() to up UI. Small apps, simple UI updates (e.g., toggling a s
Provider Uses Inherited Widget to efficiently manag state across the widget tree. Medium sized apps needing global state sharing (e.g., user authentica
Riverpod More scalable than Provider with improved dependency injection & state handling. Large, com apps requiring modular & scalable state managem (e.g. e-commerce apps).

Q4 a) Explain the process of integrating Firebase with Flutter application.
Discuss the benifits of using Firebase as a backend solution.

s:- Integrating Firebase with Flutter & its Benifits:-



Integration Process:

Setup Firebase Console:

Create a Firebase project.

Register the App for Android & iOS

Download & add google-services.json (Android) or GoogleService-Info.plist (iOS).

Install Firebase Dependencies:

yaml

```
dependencies:
  firebase_core: latest_version
  firebase_auth: latest_version
  cloud_firestore: latest_version
```

Initialize Firebase in Flutter:

dart

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

Benefits:-

No need to manage servers (Backend-as-a-Service)

Provide authentication, database & cloud Func^n,

Scalable & cost-effective.

Page No.
Date

(Q4b) Highlight the Firebase services commonly used in Flutter development & provide brief overview of how data synchronizam is achieved.

Ans:- commonly used Firebase services in flutter & Data Synchronizam service Functionality.

Firebase Authenticam Uses sign-in (Email, Google, Facebook) cloud Firestore NoSQL database for real-time data syncing. Firebase storage upload & manage files (images videos) cloud messaging push notifications, firebase Analytics App usage analytics

Data Synchronizam in Firestore:

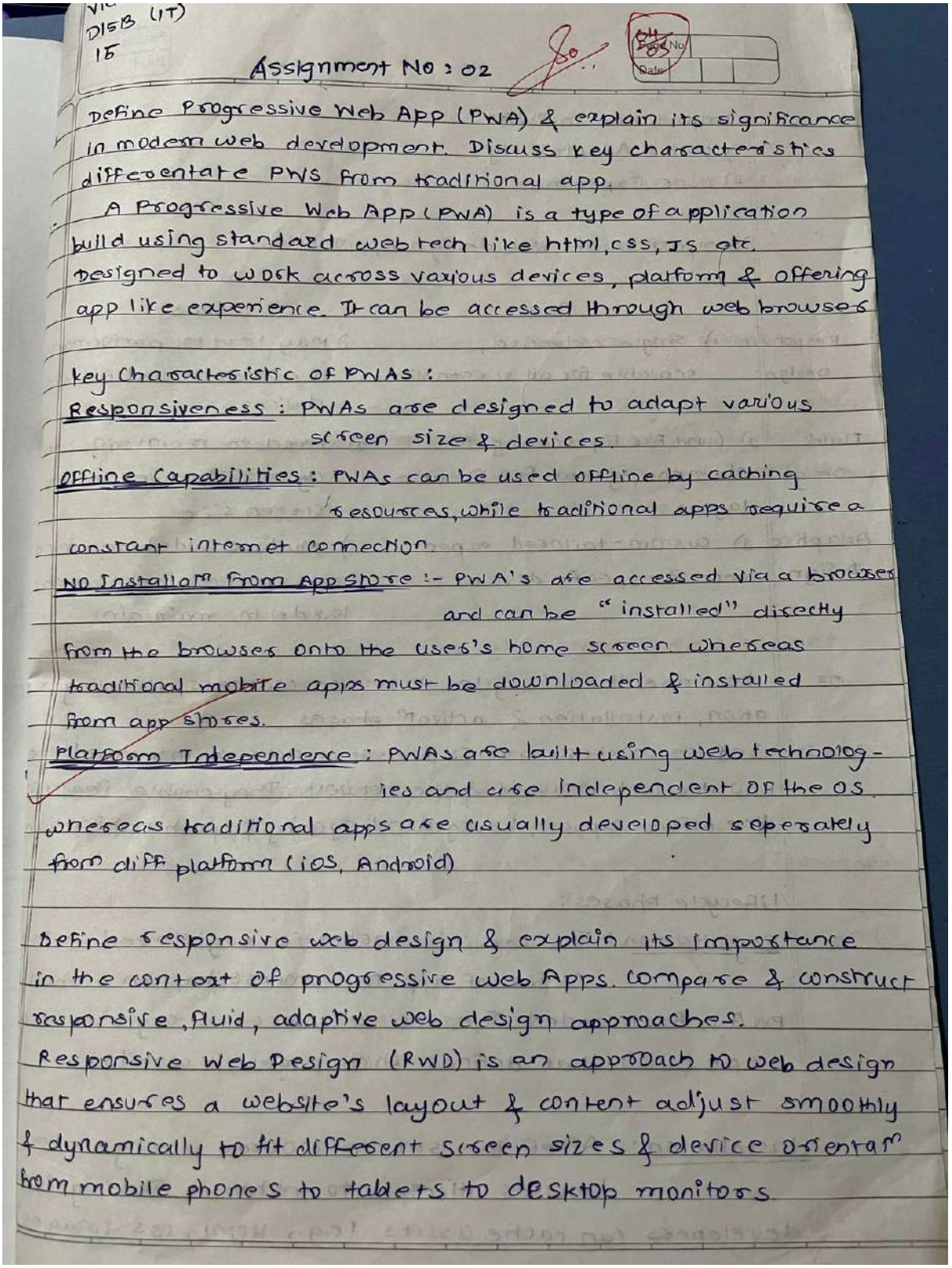
Firestore allows real-time data syncing using snapshot listmes.

eg, of real-time listmes in Firestore:

```
dart

FirebaseFirestore.instance.collem('message').snapshots(
listen((snapshot){
for(var doc in snapshot.docs){
  print(doc.data());
}
});
```

Assignment 02



D15B (IT)
15

Assignment No : 02

Define Progressive Web App (PWA) & explain its significance in modern web development. Discuss key characteristics differentiate PWS from traditional app.

A Progressive Web App (PWA) is a type of application build using standard web tech like html, css, JS etc. designed to work across various devices, platform & offering app like experience. It can be accessed through web browsers

Key Characteristic of PWAS :

Responsiveness : PWAs are designed to adapt various screen size & devices.

Offline Capabilities : PWAs can be used offline by caching resources, while traditional apps require a constant internet connection

No Installation from App Store :- PWA's are accessed via a browser and can be "installed" directly from the browser onto the user's home screen whereas traditional mobile apps must be downloaded & installed from app stores.

Platform Independence : PWAs are built using web technologies and are independent of the OS whereas traditional apps are usually developed seperately from diff platform (iOS, Android)

Define responsive web design & explain its importance in the context of progressive web Apps. Compare & construct responsive, fluid, adaptive web design approaches.

Responsive Web Design (RWD) is an approach to web design that ensures a website's layout & content adjust smoothly & dynamically to fit different screen sizes & device orientation from mobile phones to tablets to desktop monitors.

Importance in PWAs:
1) Device Adaptability
2) User Experience

Comparison of Web design Approaches:

	Pros	Cons
Responsive Design:	1) Single codebase, scalable for all screen size	1) May lead to performance i... if not optimized for differe... devices
uid Design:	2) Good for liq scaling of element, great for flexible layout	2) hard to maintain consist... cy across diverse devices screen size
laptive esign:	1) custom-tailored experience for specific devices	2) Multiple codebases for dif... ent screen sizes, which ca... harder to maintain.

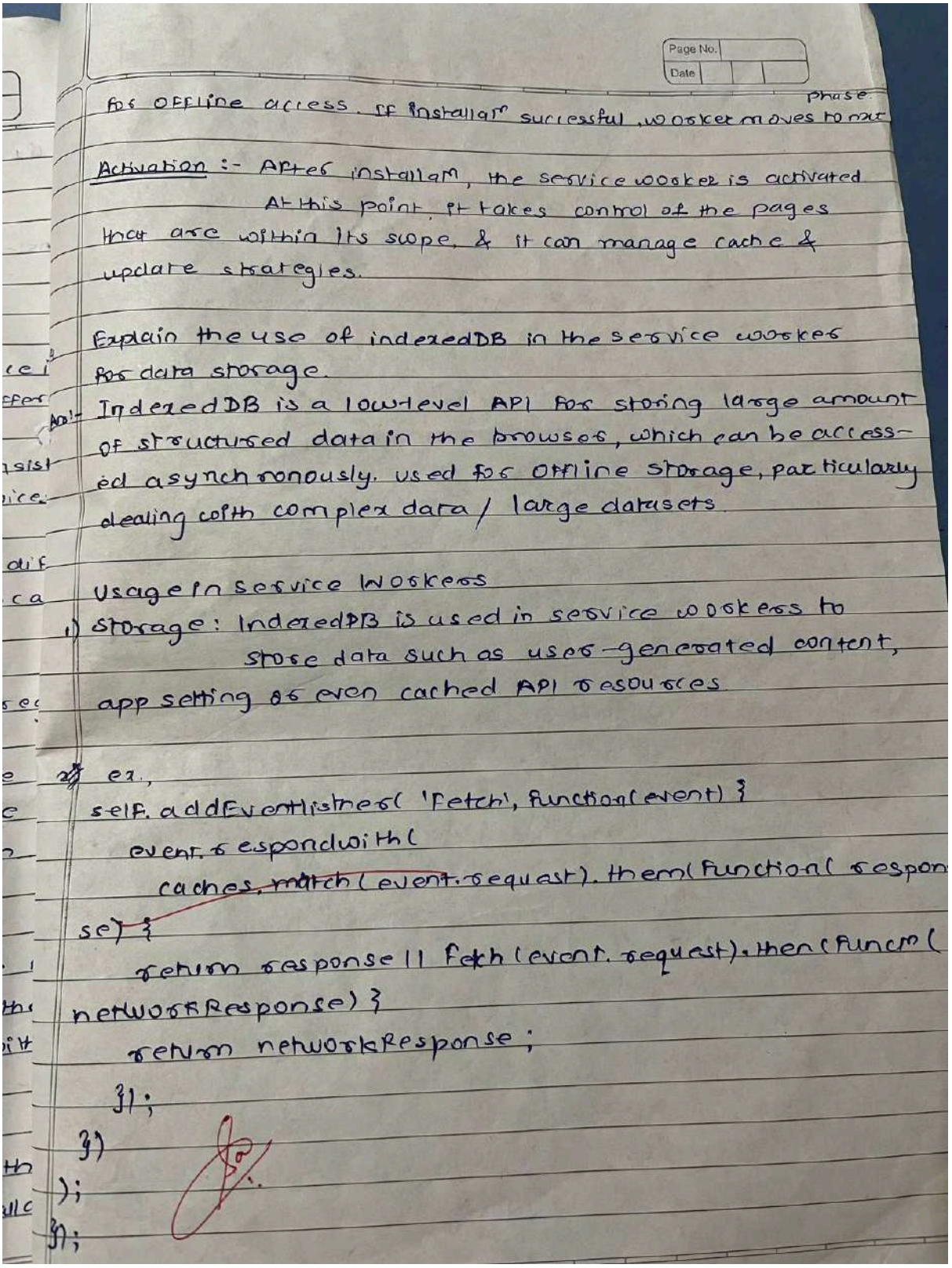
Q3. Describe the lifecycle of service workers, including reg... ation, installation & activat^n phases

ns:- Service workers are JS files that acts as intermediate b/w the web browser & the network. They enable feature... like offline support, background syncing & push notification...

Lifecycle phases:

Registration :- The service worker is registered in the r... JS file of app. The process happens when the PWA is first loaded. The service worker is registered with scope (URL) that defines which pages it controls.

Installation :- After the service worker is registered, th... browser attempts to install it. During installa... developers can cache assets (e.g., HTML, CSS, images)



for OFFline access. If installation successful, worker moves to next phase.

Activation :- After installation, the service worker is activated. At this point, it takes control of the pages that are within its scope, & it can manage cache & update strategies.

Explain the use of indexedDB in the service workers for data storage.

Ans:- IndexedDB is a low-level API for storing large amount of structured data in the browser, which can be accessed asynchronously. used for Offline storage, particularly dealing with complex data / large datasets.

Usage in service Workers

1) storage: IndexedDB is used in service workers to store data such as user-generated content, app setting or even cached API resources.

ex.,

```
self.addEventListner('fetch', function(event) {
  event.respondWith(
    caches.match(event.request).then(function(response) {
      return response || fetch(event.request).then(function(
networkResponse) {
        return networkResponse;
      });
    })
  );
});
```